



Future Technology Devices International Ltd.

User Manual

AN_151

Vinculum II User Guide

Document Reference No.: FT_000289

Version 1.2.2

Issue Date: 2010-**10-12**

This provides information and examples on using the Vinculum II
Toolchain, Firmware, Libraries and Sample code.

Future Technology Devices International Limited (FTDI)

Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758
E-Mail (Support): support1@ftdichip.com Web: <http://www.ftdichip.com>

© 2010 Future Technology Devices International Ltd.

Table of Contents

1 Introduction.....	8
2 Getting Started Guide	9
2.1 Introduction.....	9
2.2 Overview	9
2.3 Building Your First Application.....	9
2.4 Writing 'Hello World' Application.....	14
2.5 Writing an Application.....	20
2.6 Code Listing.....	26
3 Toolchain.....	32
3.1 Toolchain Basics	32
3.2 VinC Compiler	32
3.2.1 Introduction	32
3.2.1.1 Compiler Overview	32
3.2.1.2 ANSI C Feature Support Summary.....	33
3.2.1.3 C Language Restrictions.....	34
3.2.2 Compiler Command Line Options.....	34
3.2.2.1 Compiler File Type.....	35
3.2.2.2 Compile Stage Selection.....	35
3.2.2.3 Compiler Output	35
3.2.2.4 Compiler Information Options.....	36
3.2.2.5 Compile Time Options.....	36
3.2.2.6 Preprocessing Options.....	36
3.2.2.7 Linker Options.....	37
3.2.3 Data Types	37
3.2.3.1 Type Qualifiers	37
3.2.3.2 Storage Type Specifiers.....	38
3.2.3.3 Type Specifiers	38
3.2.3.4 Data Conversion References.....	43
3.2.4 Special VNC2 Reference.....	44
3.2.4.1 Special Features.....	44
3.2.4.2 Function Call.....	45
3.2.4.3 Architecture Issues.....	47
3.2.4.4 Considerations of local vs global variables.....	47
3.2.4.5 Sequence Points.....	47
3.2.5 Error reference.....	48
3.2.5.1 Examples for General Errors.....	50
3.2.5.2 Examples for Syntax Error Codes.....	51
3.2.5.3 Examples for General Syntax Error Codes.....	51
3.2.5.4 Examples for Conditional Statement Error Codes.....	52
3.2.5.5 Examples for Storage Classes Error Codes.....	55

3.2.5.6 Examples for Declaration Error Codes	56
3.2.5.7 Examples for Constant Range Error Codes	58
3.2.5.8 Examples for Constant Error Codes	60
3.2.5.9 Examples for Variable Error Codes	60
3.2.5.10 Examples for Array Error Codes	60
3.2.5.11 Examples for Structure Union Error Codes	61
3.2.5.12 Examples for Initialisation Error Codes	62
3.2.5.13 Examples for Function Error Codes	62
3.2.5.14 Examples for Pointer Error Codes	63
3.2.5.15 Examples for Bitfield Error Codes	65
3.2.6 Pre-processor	66
3.2.6.1 Pre-processor Directives	66
3.2.6.2 Error reference	68
3.3 VinAsm Assembler	73
3.3.1 Assembler Command Line Options	73
3.3.2 Assembly Language	74
3.3.2.1 Lexical Conventions	74
3.3.3 Assembler Directives	75
3.3.3.1 Data Directives	75
3.3.3.2 Debugger Directives	80
3.3.3.3 End Directive	91
3.3.3.4 File Inclusion Directive	91
3.3.3.5 Location Control Directives	91
3.3.3.6 Symbol Declaration Directives	93
3.3.4 Machine Instructions	95
3.3.4.1 CPU General Instructions	95
3.3.4.2 CPU Stack Operation Instructions	98
3.3.4.3 CPU Memory Operation Instructions	100
3.3.4.4 CPU Bitwise Shift Operation Instructions	101
3.3.4.5 CPU Logic Operation Instructions	106
3.3.4.6 CPU Arithmetic Operation Codes	109
3.3.4.7 CPU Bitwise Operation Instructions	116
3.3.4.8 CPU I/O Operation Instructions	117
3.3.4.9 CPU Comparison Instructions	118
3.3.4.10 CPU Program Flow Instructions	119
3.3.5 Error Reference	122
3.4 VinL Linker	124
3.4.1 Linker Command Line Options	124
3.4.2 Memory and Segment	125
3.4.3 Linker Optimization	126
3.4.4 Map File	127
3.4.5 Archive File	127
3.4.6 Error Reference	128
3.4.7 Special VNC2 Reference	130
3.5 VinIDE	131

3.5.1 About VinIDE.....	131
3.5.2 The User Interface.....	131
3.5.2.1 The Tabbed Toolbar.....	132
3.5.2.2 The Source Code Editor.....	133
3.5.2.3 The Project Manager.....	134
3.5.2.4 The Messages Window.....	135
3.5.2.5 The Watchlist Window.....	136
3.5.2.6 The Memory Window.....	136
3.5.2.7 The Breakpoint Window.....	136
3.5.2.8 Managing the Panels.....	137
3.5.3 Using VinIDE.....	140
3.5.3.1 Project/File Handling.....	140
3.5.3.2 Building a project.....	150
3.5.3.3 Debugging a project.....	154
3.5.3.4 Project Options.....	158
3.5.3.5 The IDE Options.....	164
3.5.3.6 Plugins.....	171
3.5.3.7 Keyboard Shortcuts.....	174
4 Firmware.....	176
4.1 VOS Kernel.....	176
4.1.1 VOS Definitions.....	177
4.1.2 Kernel Initialisation.....	177
4.1.2.1 vos_init().....	178
4.1.3 Thread Creation.....	178
4.1.3.1 vos_create_thread().....	179
4.1.4 Kernel Scheduler.....	180
4.1.4.1 vos_start_scheduler().....	180
4.1.4.2 vos_delay_msecs().....	180
4.1.4.3 vos_delay_cancel().....	181
4.1.5 Mutexes.....	181
4.1.5.1 vos_init_mutex().....	182
4.1.5.2 vos_lock_mutex().....	183
4.1.5.3 vos_trylock_mutex().....	183
4.1.5.4 vos_unlock_mutex().....	184
4.1.5.5 vos_get_priority_ceiling() Advanced.....	184
4.1.5.6 vos_set_priority_ceiling() Advanced.....	184
4.1.6 Semaphores.....	185
4.1.6.1 vos_init_semaphore().....	186
4.1.6.2 vos_wait_semaphore().....	186
4.1.6.3 vos_wait_semaphore_ex().....	187
4.1.6.4 vos_signal_semaphore().....	188
4.1.6.5 vos_signal_semaphore_from_isr().....	188
4.1.7 Condition Variables.....	189
4.1.7.1 vos_init_cond_var().....	190
4.1.7.2 vos_wait_cond_var().....	191

4.1.7.3 vos_signal_cond_var()	191
4.1.8 Critical Sections	192
4.1.9 Device Manager	192
4.1.9.1 Driver Initialisation	194
4.1.9.2 Driver Operation	196
4.1.10 Hardware Information and Control	199
4.1.10.1 vos_set_clock_frequency() and vos_get_clock_frequency()	199
4.1.10.2 vos_get_package_type()	199
4.1.10.3 vos_get_chip_revision()	200
4.1.10.4 vos_power_down()	200
4.1.10.5 vos_halt_cpu()	200
4.1.11 Kernel Services	201
4.1.11.1 DMA Service	201
4.1.11.2 IOMUX Service	206
4.2 FTDI Drivers	209
4.2.1 Hardware Device Drivers	209
4.2.1.1 UART, SPI and FIFO Drivers	209
4.2.1.2 USB Host Driver	229
4.2.1.3 USB Slave Driver	263
4.2.1.4 GPIO Driver	273
4.2.1.5 Timer Driver	283
4.2.1.6 PWM Driver	290
4.2.2 Layered Drivers	300
4.2.2.1 Mass Storage Interface	300
4.2.2.2 USB Host Class Drivers	302
4.2.2.3 USB Slave Class Drivers	324
4.2.2.4 File Systems	324
4.3 FTDI Libraries	378
4.3.1 ctype	378
4.3.1.1 isalnum	379
4.3.1.2 isalpha	379
4.3.1.3 iscntrl	379
4.3.1.4 isdigit	380
4.3.1.5 isgraph	380
4.3.1.6 islower	380
4.3.1.7 isprint	380
4.3.1.8 ispunct	381
4.3.1.9 isspace	381
4.3.1.10 isupper	381
4.3.1.11 isxdigit	382
4.3.2 stdio	382
4.3.2.1 fsAttach	384
4.3.2.2 stdioAttach	384
4.3.2.3 stdinAttach	385
4.3.2.4 stdoutAttach	385

4.3.2.5 stderrAttach	385
4.3.2.6 printf	386
4.3.2.7 fopen	387
4.3.2.8 fread	387
4.3.2.9 fwrite	388
4.3.2.10 fclose	389
4.3.2.11 feof	389
4.3.2.12 ftell	389
4.3.2.13 fprintf	390
4.3.2.14 stdout	391
4.3.2.15 stdin	391
4.3.2.16 stderr	391
4.3.3 stdlib	391
4.3.3.1 abs	392
4.3.3.2 strtol	392
4.3.3.3 atol	393
4.3.3.4 atoi	393
4.3.3.5 malloc	393
4.3.3.6 calloc	394
4.3.3.7 free	394
4.3.4 string	394
4.3.4.1 memcpy	395
4.3.4.2 memset	396
4.3.4.3 strcmp	396
4.3.4.4 strncmp	396
4.3.4.5 strcpy	397
4.3.4.6 strncpy	397
4.3.4.7 strcat	398
4.3.4.8 strlen	398
4.3.5 errno	398
4.3.5.1 errno	399

5 Sample Firmware Applications.....400

5.1 Sample Firmware Overview.....400

5.2 General Samples.....400

5.2.1 Template Sample 400 |

5.2.2 GPIOKitt Sample 401 |

5.2.3 PWMBreathe Sample 402 |

5.2.4 Philosophers Sample 403 |

5.2.5 Runtime Sample 404 |

5.2.6 HelloWorld Sample 406 |

5.3 USB Host Samples.....407

5.3.1 StillImageApp Sample 407 |

5.3.2 USBHostGeneric Sample 409 |

5.3.3 USBHostGPSLogger Sample 410 |

5.3.4 USBHostHID Sample 411 |

5.3.5 USBHostHID2 Sample	412
5.3.6 USBMic Sample.....	413
5.4 USB Slave Samples.....	415
5.4.1 USBSlaveFT232App Sample.....	415
5.5 Firmware Samples	416
5.5.1 VNC1L Firmware.....	416
5.5.1.1 V2DAP Firmware.....	416
5.5.1.2 V2DPS Firmware.....	417
6 Contact Information.....	419
7 Revision History.....	421

1 Introduction



- [Getting Started Guide](#)
- [Vinculum II Toolchain](#) - Compilation and development tools for Vinculum II.
 - [VinC Compiler](#)
 - [VinAsm Assembler](#)
 - [VinL Linker](#)
 - [VinIDE](#) Integrated Development Environment
- [Firmware](#) - RTOS, device drivers, runtime libraries for Vinculum II.
 - [VOS Kernel](#)
 - [FTDI Drivers](#)
 - [FTDI Libraries](#)
- [Sample Firmware Overview](#)

2 Getting Started Guide

2.1 Introduction

The scope of this document is to provide an introduction to using the VNC2 toolchain. This document is intended for people who have successfully installed the VNC2 toolchain and is provided as a getting started guide for first time users.

FTDI provide a number of sample applications with the toolchain installation, these samples are designed to familiarise users with the supplied FTDI drivers, libraries and development environment. It is recommended that to follow this tutorial more easily all files installed during the installation are kept in their default location as per the installation wizard.

All code samples in this document are provided for illustration purposes only. They are not guaranteed or supported by FTDI.

2.2 Overview

The intention of this document is to give novice users of the Vinculum-II software development toolchain the knowledge to build and run their first sample application and to then use this knowledge to go on to write and build a first application from scratch. It does this using a short tutorial.

The tutorial firstly focuses on the Kitt sample, provided along with the Vinculum-II toolchain, to demonstrate: the opening of projects; building firmware for the VNC2; loading this firmware onto the device; running the firmware on the VNC2 and finally using the debugger to step through code. Secondly, it introduces writing an application from scratch based on the Hello World sample. This demonstrates how to use FTDI supplied device drivers and outlines the general structure applications may take.

2.3 Building Your First Application

Installing the VNC2 toolchain (using the default settings) results in the toolchain being installed within the `Program Files/FTDI/Vinculum II Toolchain` directory on the PC's local hard disk; the installer creates a start menu shortcut, again under the `FTDI/Vinculum II Toolchain` folder heading. The VNC2 IDE is located within either of these two folders; to launch the application double click on Vinculum II IDE icon.

Opening the Sample Project

This is an overview of the IDE GUI, the layout may not match exactly Figure 1, however, this can be easily modified using the built-in docking manager.

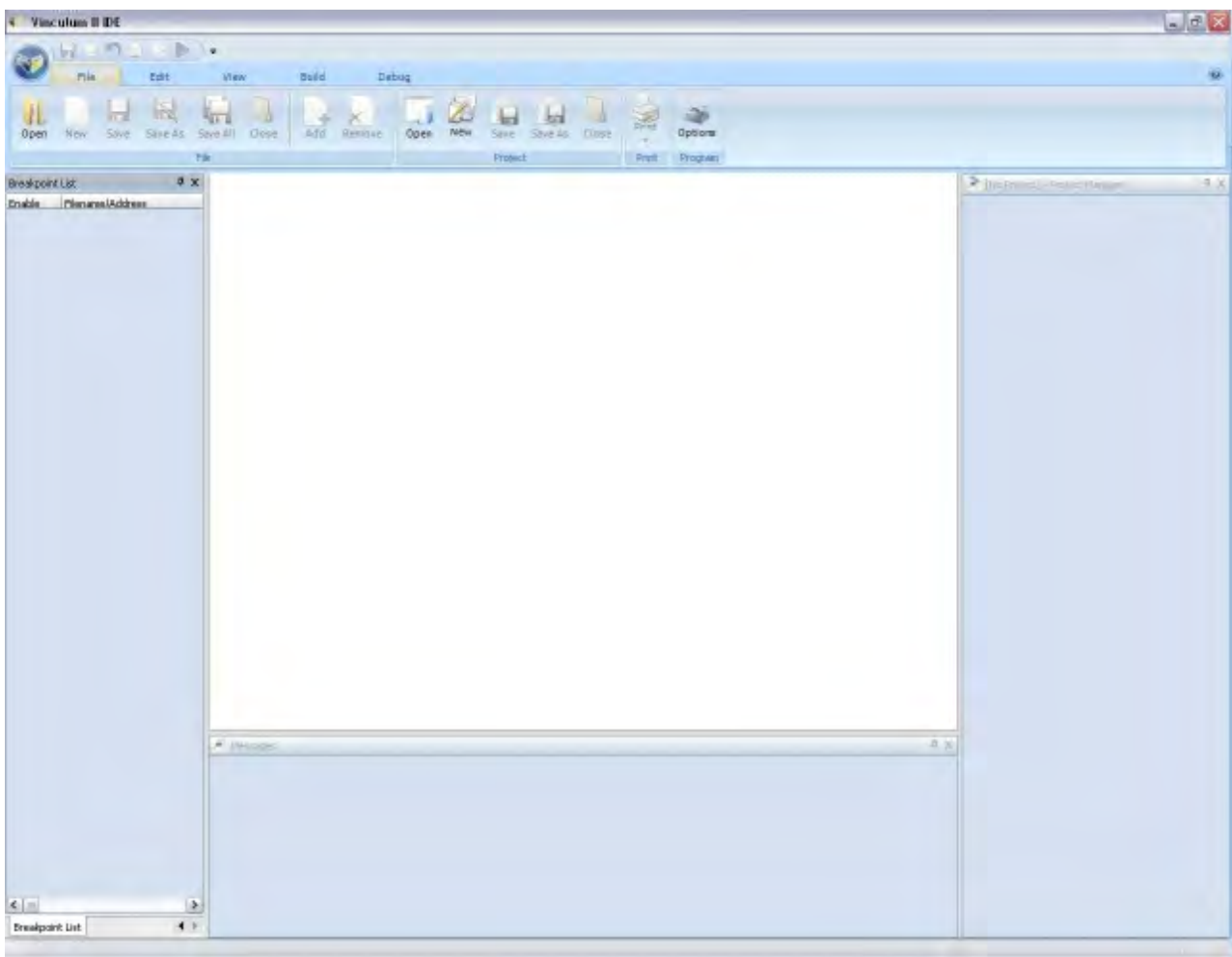


Figure 1

The tabbed tool strip running along the top of the screen gives access to the menu and sub menu items within the IDE.

Within the File tab, as above, notice the Project subcategory, click on the Open button. By following the default settings within the installation wizard the FTDI provided samples are saved within the My Documents folder. Using the project dialog box, browse to My Documents and find the folder FTDI/Firmware/Samples/ReleaseVersion/General. Within this is a folder called kitt containing a file kitt.vproj (vproj is the file extension used by all VNC2 project files) double click this file to launch it within the IDE.

Building the Application

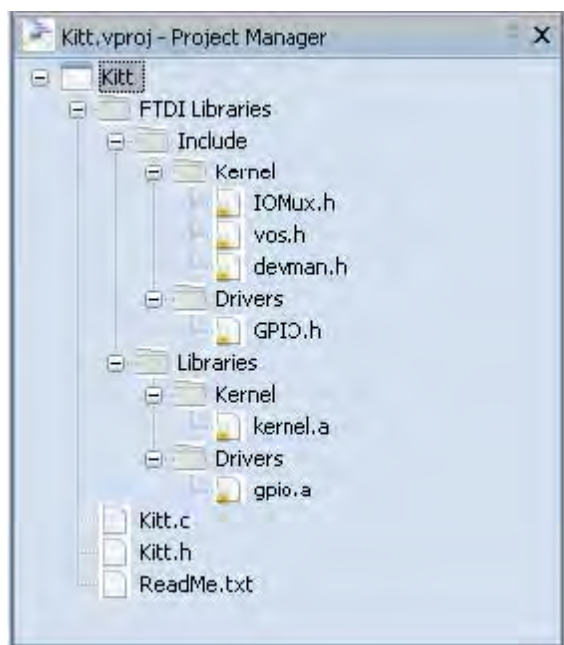
Notice that when an application is opened within the IDE the Project Manager window now contains all the files relevant to that project. If the Project Manager window is not visible go back to the tabbed tool strip, along to view and make sure that the Project Manager box is checked (Figure 2). IDE panels can be dragged and docked anywhere on the screen using the built-in docking manager, simply click and hold the title bar of a panel to free it and then drag it to the desired area of the screen.

Files within this project can now be opened within the editor window by double clicking them; the editor window allows multiple files to be open concurrently. The archive files under the Libraries folder contain FTDI supplied drivers and VOS Kernel Services, these files cannot be opened or edited.



Figure 2

To build the sample Kitt application: go to the tabbed tool strip and along to the Build tab. In the left hand side of the panel is a button called Build, this will generate the ROM file (firmware) that can be programmed into the VNC2 IC. Note that under the Build Configuration sub-category the project is set in Debug mode; this is important at this stage as it will allow debugging of source code after the ROM image has been loaded into the VNC2 device.



After clicking Build the IDE will attempt to compile, assemble and link the source code into a format that can be loaded into the VNC2. If the source code within kitt.c hasn't been altered there should be no compilation errors, meaning the Kitt application should build first time.



The outcome from a build attempt is displayed within the Messages Window at the bottom of the screen. The last line within the Messages Window indicating that there have been 0 errors from the build shows that the IDE has successfully created the ROM file.

[VinL.exe] : 0 errors, 0 warnings and 0 informational messages

Flashing VNC2

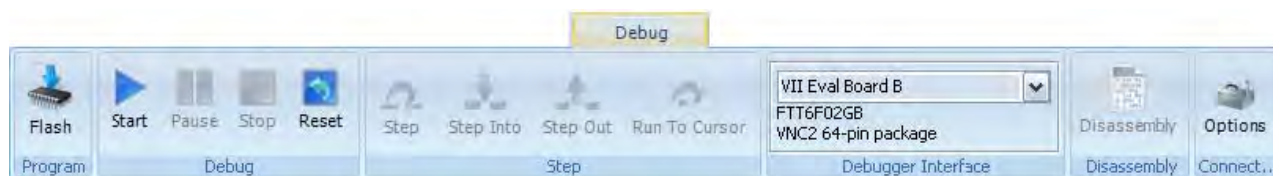
The next step is to program the ROM file from the above build process into the VNC2 flash memory, but first connect the VNC2 to the host PC (Figure 3). This demonstration uses the V2EVAL with the

64 pin QFN daughter card installed. The debugger port is connected to the host PC via the blue USB cable (shown at the top of figure 3). The power switch for the device is located just below the black power supply socket in the top left corner of the screen; in this mode, this device is self powered and therefore does not require an external power supply to operate. When the V2EVAL board is connected, the host PC may attempt to install FTDI drivers for the FT4232H connected to the debugger port.



Figure 3

After connecting the V2EVAL board as shown in Figure 3, open up the IDE and select the Debug tab within the tool strip. Selecting the drop-down menu within the Debugger Interface subcategory initiates the IDE to search for connected devices and, as can be seen below, automatically selects the debugger interface of the V2EVAL board. When a debugger interface is selected the Flash, Start and Reset buttons also become active. To program the flash memory of the VNC2 select Flash from the Debug tab, a dialog box appears showing the Kitt.rom image file that was built earlier, select this file and press open.



The IDE attempts to program the VNC2 flash, all relevant information will be shown in the Message Window at the bottom of the screen.

Running the Application

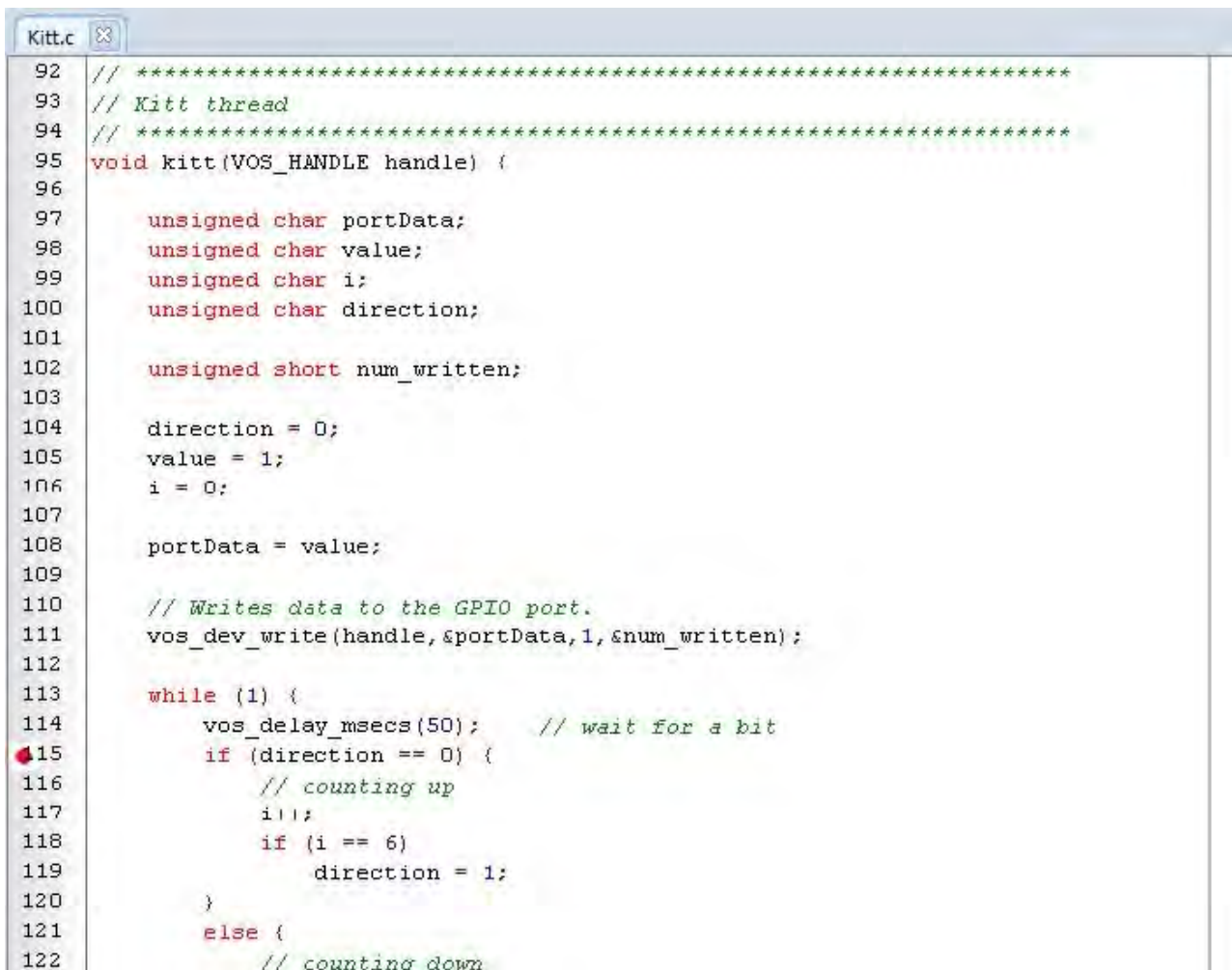
To run the Kitt sample, press the start button in the Debug tab. The four LEDs located in the bottom left hand corner of the V2EVAL board should now be sequentially flashing signifying that the device is running correctly. The Pause and Stop buttons become active only if the ROM file loaded into the VNC2 has been built in debug mode. Pause can be used to suspend execution at the current line

within the disassembly file. Stop can be used to halt executing and stop the firmware on the VNC2 executing.

Debugging the Application

The debugger interface supplied by FTDI allows for source code debugging at C and assembly level; this tutorial illustrates using the C level debugging.

The IDE allows breakpoints to be added to the C source, in the below diagram (Figure 4) a breakpoint has been added to the source code at line 115. A breakpoint is added by clicking the desired line number in the left hand side gutter of the screen. Breakpoints can be placed on lines with no code, for example lines with comments, but these are grayed out when the debugger starts and will not be hit. VNC2 supports 3 breakpoints being set concurrently; any extra breakpoints are deselected within the Breakpoint List window and are grayed out within the editor.



```
Kitt.c
92 // *****
93 // Kitt thread
94 // *****
95 void kitt(VOS_HANDLE handle) {
96
97     unsigned char portData;
98     unsigned char value;
99     unsigned char i;
100     unsigned char direction;
101
102     unsigned short num_written;
103
104     direction = 0;
105     value = 1;
106     i = 0;
107
108     portData = value;
109
110     // Writes data to the GPIO port.
111     vos_dev_write(handle, &portData, 1, &num_written);
112
113     while (1) {
114         vos_delay_msecs(50);    // wait for a bit
115         if (direction == 0) {
116             // counting up
117             i++;
118             if (i == 6)
119                 direction = 1;
120         }
121         else {
122             // counting down
```

Figure 4

To hit a breakpoint press Start within the debug menu. The application runs to this breakpoint and execution from the VNC2 stops; this allows for lines of code to be single stepped using the Step control panel within the Debug menu.

Individual variables within the source code can also be added to a watch list; this allows for the value of certain variables to be monitored during execution of the source code. To add a watch bring up the Watch window from the View tab within the tool strip. Right click within the Watch List and select Add Watch; enter the name of the variable to be monitored and press Add Watch. Figure 5 illustrates the variable value added to a watch list. All watches that have been added are displayed within the Watch List; a watch that has a value of undefined is either outside the current scope of execution or is not defined within the current application. During single stepping of code it is now possible to monitor changes within the value field of each variable to aid with debugging.



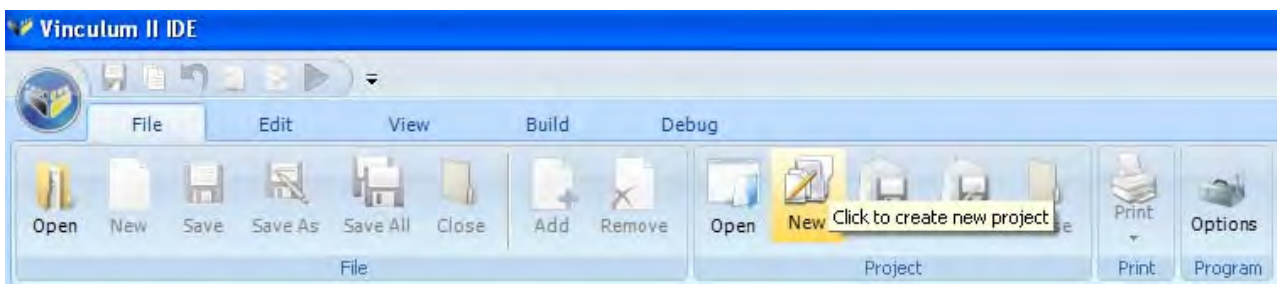
Figure 5

2.4 Writing 'Hello World' Application

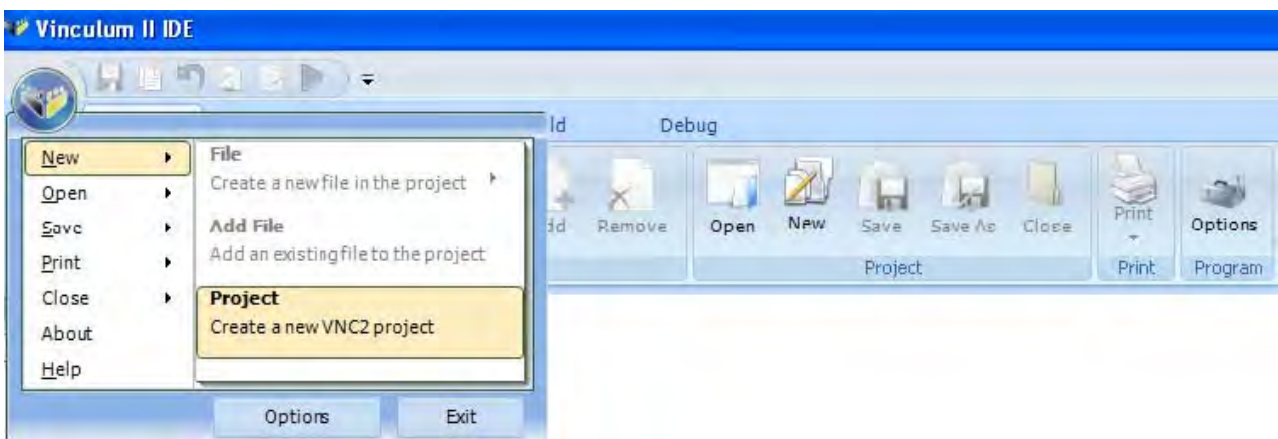
This section illustrates the creation of an application from scratch based upon the Hello World sample provided by FTDI. Hello World is a simple application that connects to a USB flash drive, creates a new text file on the drive and writes the string "Hello World" to this file. This project demonstrates the main components of writing an application and how to use a selection of the provided drivers, Kernel services and libraries.

Creating a new Project

To create a new project, go to the File tab within the toolbar and click New under the Project tab (see Figure below).



Alternatively, go to the circular Vinculum button and click New->Project



This pops up a New Project dialog box which allows for browsing to the project location and renaming of the project and solution. It is necessary to complete the text boxes before clicking OK.

Create a new project called HelloWorld as demonstrated within Figure 6.

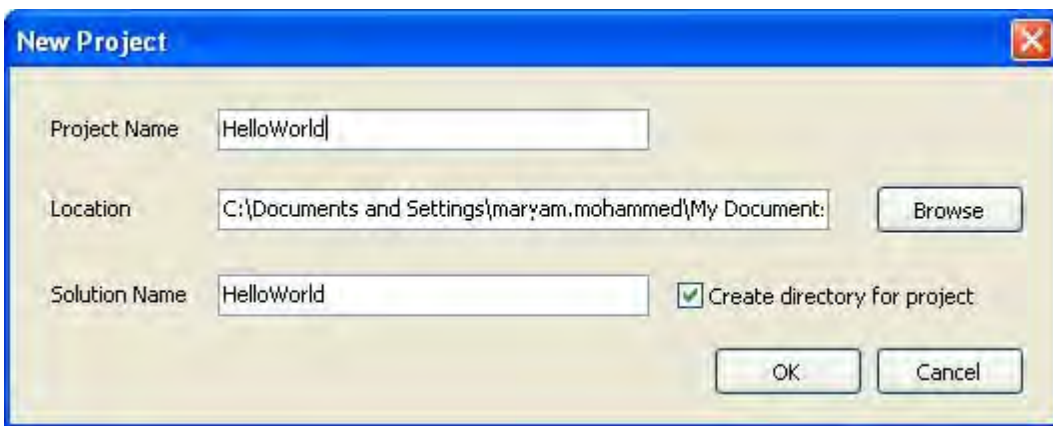


Figure 6

Note: The project and solution names do not need to match. It is recommended that the Create directory for project check box is selected; this creates a project directory containing the project and any subsequent files.

The result of creating a new project in the Project Manager panel is illustrated in Figure 7 which shows a new project called HelloWorld which has been created.

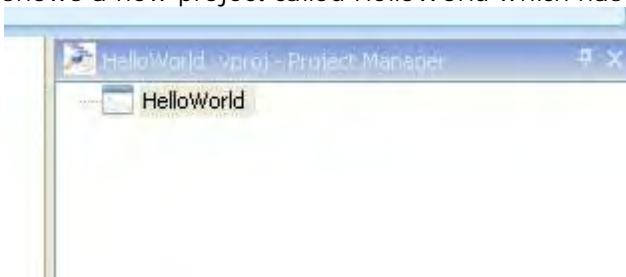
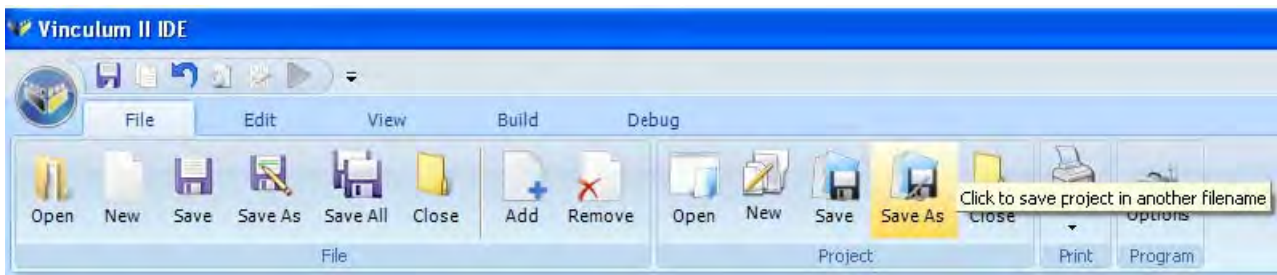


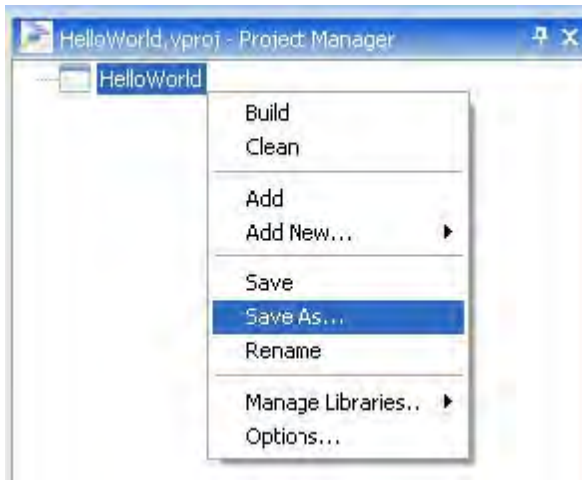
Figure 7

Saving a Project

To save a project, use the Save As button within the File tab.



Alternatively, go to the Project Manager and right click "HelloWorld" ->Save As.



This will result in the Save Project As dialog box (Figure 8) appearing.

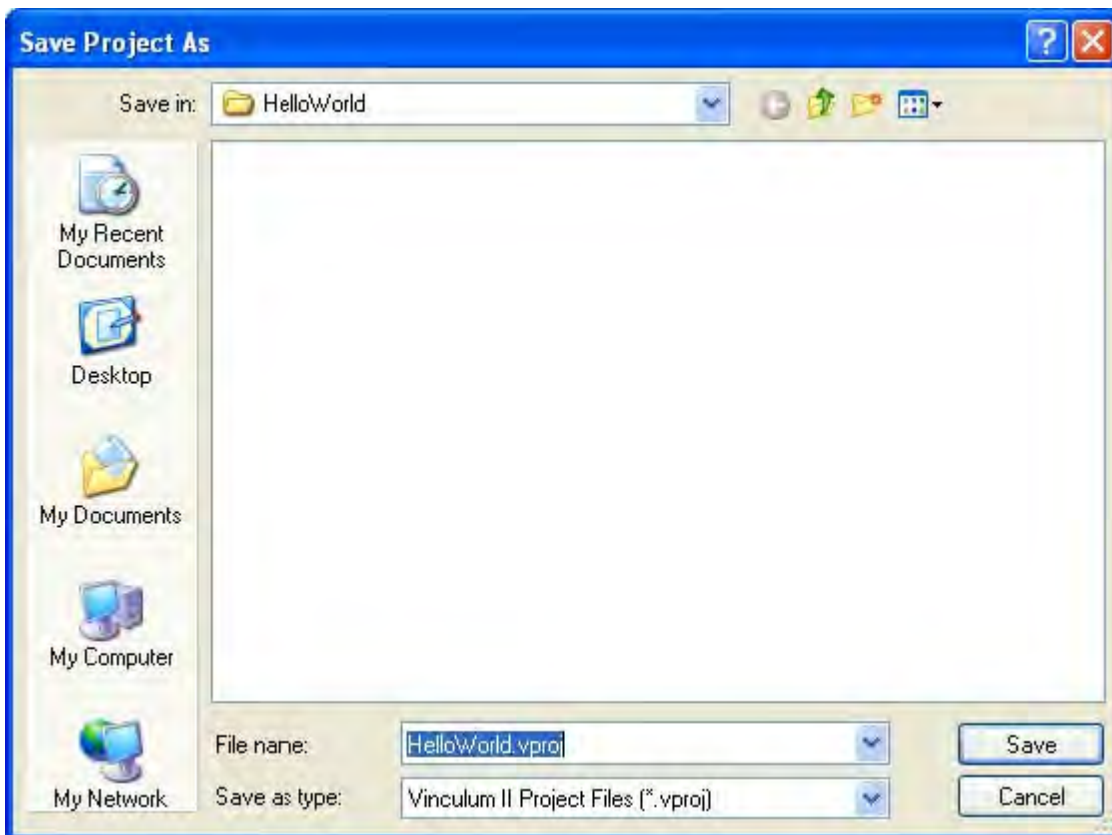
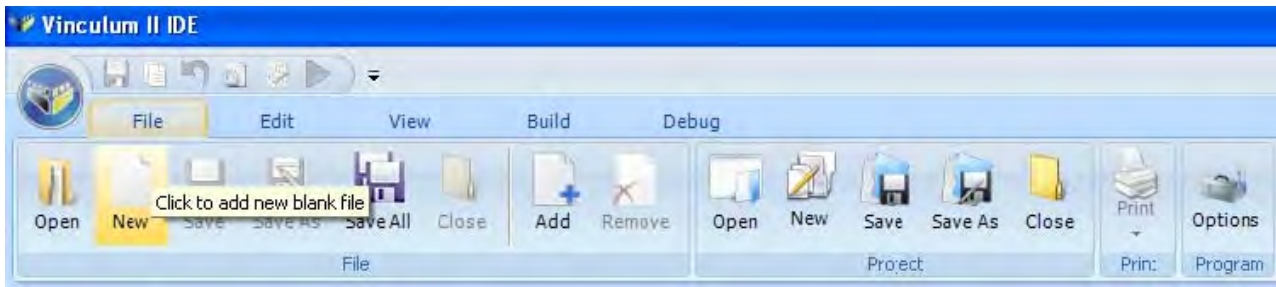


Figure 8

Select a location and filename for saving the project.

Adding New Files to a Project

To add a new file to the HelloWorld project, go to the File tab within the toolbar and click New in the File group.



The New File pop-up allows different types of file to be added to a project. Firstly add a new C File which will contain the main body of code for the application. Select C File in the New File pop-up and press Add. Repeat this to add another new file to the project, this time a Header File.



Notice that within the Project Manager window, as shown in Figure 9, there are now two new files under the project heading. To rename both files: right click on File.c within the Project Manager and select Rename. The IDE prompts to save this file first before renaming it, click Yes to confirm this. Within the save dialog box rename this file as HelloWorld.c and click OK. Repeat these steps for the header file in the project, calling it HelloWorld.h.

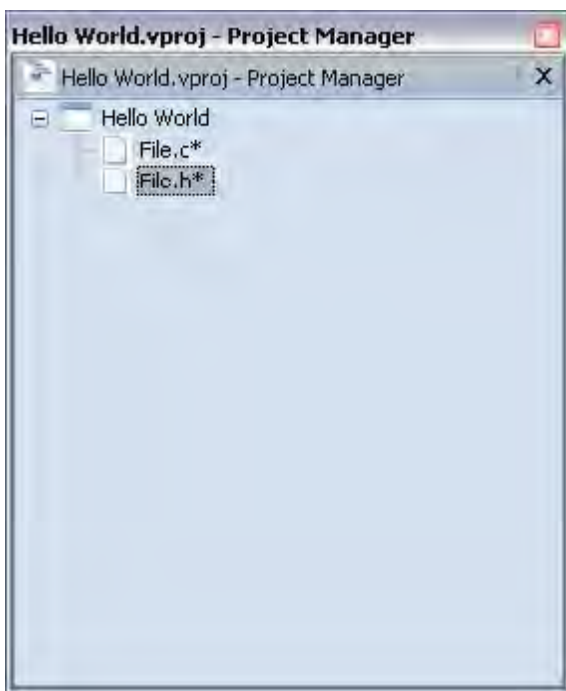
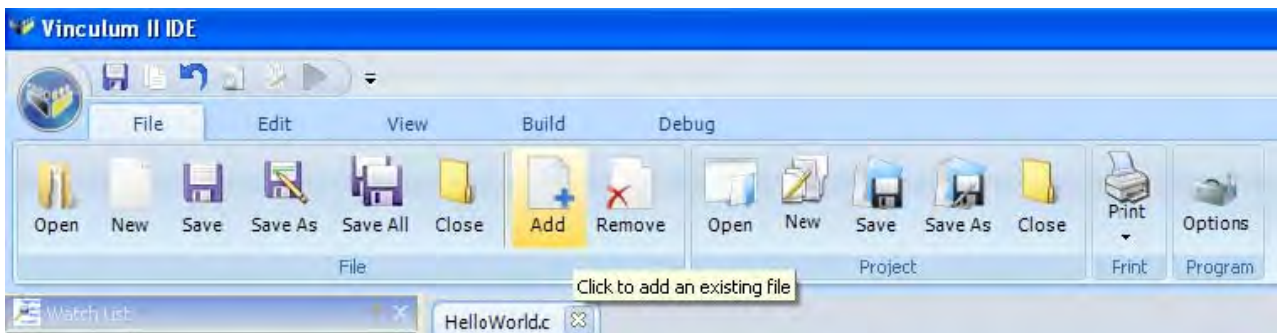


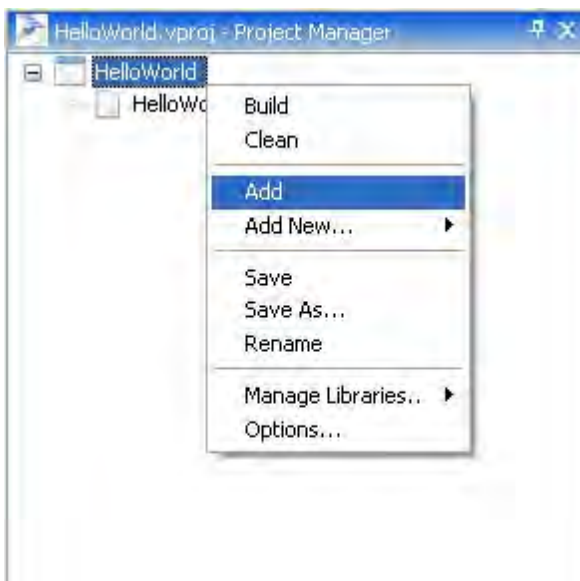
Figure 9

Adding Existing Files to a Project

To add an existing file to the "HelloWorld" project, go to the File tab of the toolbar and click Add in File tab group. The open dialog (Figure 10) allows for different file types(C,ASM, Header or Text) to be added from this project or any other project. It is also possible to add multiple files by holding the CTRL key while clicking on each of the files required to be added.



Alternatively, go to the Project Manager and right click "HelloWorld" project ->Add



To add a file click the Open button. All added files are added to the Project Manager window as illustrated in Figure 11.

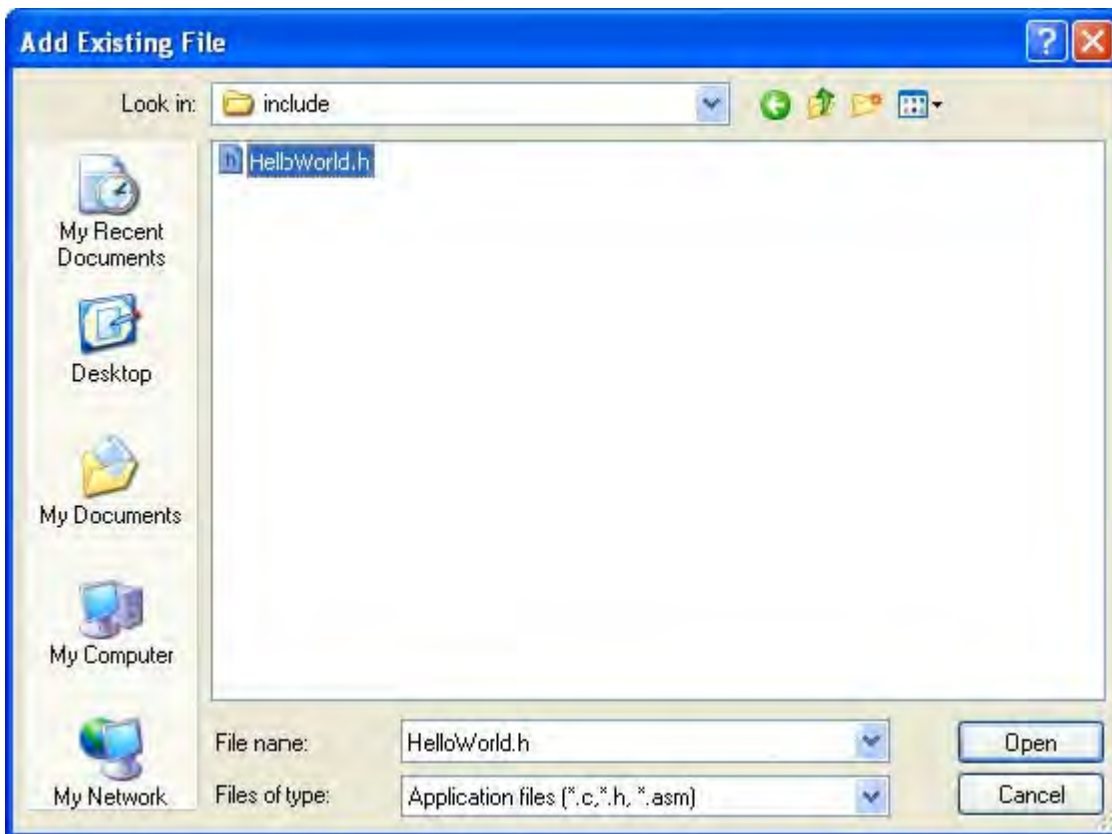


Figure 10

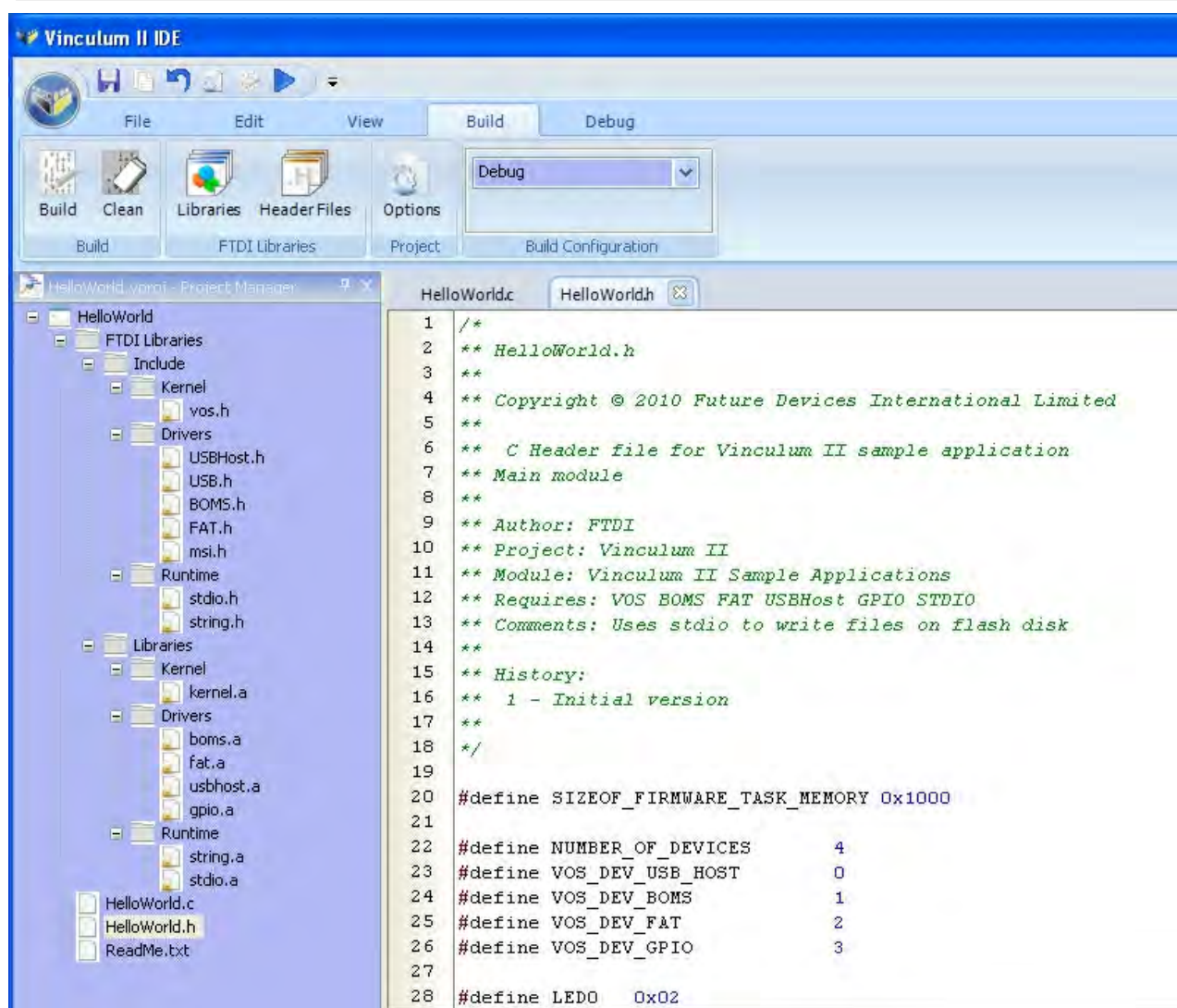


Figure 11

2.5 Writing an Application

In this section, an application is written which writes a line of text to a file on a disk. The listing is available in the [Getting Started Code Listing](#) topic and it is in the samples directory under `General/HelloWorld`.

Header File

To write some example code for this application starting with the header file: double click `HelloWorld.h` within the Project Manager to open this file in the IDE editor. Header files contain forward declarations of functions, constant values and any other global variable declarations that are shared throughout the application. Although it is not strictly necessary to use a header file within this project it is good programming practice to get into the habit of using them, especially when dealing with more complicated projects than the HelloWorld application.

The first thing to define within the header is the size of stack memory that the application thread is going to need. Details of this will be explained further when it comes to creating a thread within the application.

Paste the following code fragment into the header file:

```
#define SIZEOF_FIRMWARE_TASK_MEMORY 0x1000
```

Next decide the number of device interfaces that are used within the application. The HelloWorld app requires: a USB Host driver to connect to the USB flash drive; a BOMS driver and FAT file system driver to allow communication to the flash disc and also a GPIO driver allowing for visual feedback to

the user using the LEDs on the V2EVAL board. When initializing each device they must have a unique identifier that is used within the Kernel's device manager. As well as this the number of devices used within the application must be explicitly specified.

Again, copy and paste the following code fragment into the .h file.

```
#define NUMBER_OF_DEVICES 4
#define VOS_DEV_USB_HOST 0
#define VOS_DEV_BOMS 1
#define VOS_DEV_GPIO 2
#define VOS_DEV_FAT 3
```

Lastly, add a forward declaration for the application thread:

```
void firmware(void);
```

The header file should now match closely the code shown in Figure 12.

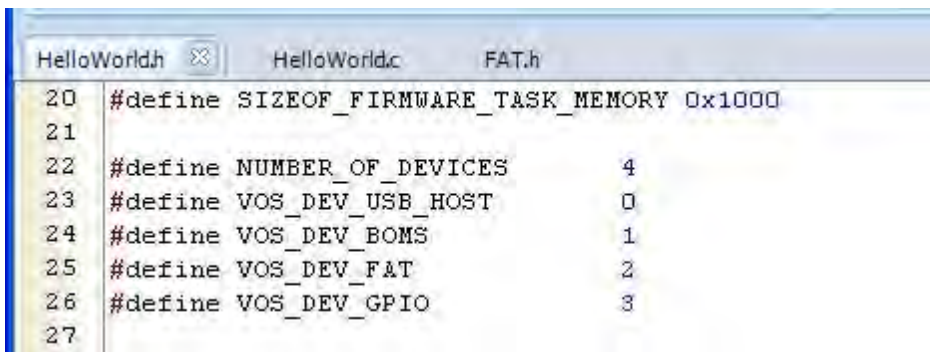


Figure 12

Other definitions of LED combinations are found in the listing of the header file in the [Getting Started Code Listing](#) topic.

FTDI Libraries

With VNC2, all applications integrate with FTDI provided libraries that contain VOS Kernel Services, device drivers and runtime libraries. Kernel Services provide all the data structures and primitives that an application uses, as well as providing control throughout the lifetime of the application.

The Device Manager defines a standard API for device drivers. All devices are accessed using this standard API to make application development easier. Device Manager is the interface between user applications and Kernel Services. Runtime libraries contain functions which are common to most C language implementations, for example string and standard IO.

The Hello World application requires a selection of Runtime Libraries, FTDI drivers as well as Kernel Services to run. These are provided in the form of archive files which come with the VNC2 toolchain installation. To utilize the provided libraries they must be included in the application. Each archive file has a corresponding header file that defines its API, providing information on functions and data structures that are contained within the archive files.

The Hello World application requires the following device drivers: USBHost acting as an interface to the USB drive; the BOMS driver to communicate with a mass storage device; the FAT driver to communicate with the device file structure and the GPIO driver which allows for visual feedback using the LEDs. The [string](#) runtime library which contains string manipulation functions and [stdio](#) to provide file I/O functions also needs inclusion. Finally Kernel Services, which provide overall control of the device drivers, need to be added. As well as adding the archive files the corresponding header files require inclusion.

3.4.2.1 Adding Library Files

To add Library Archive Files to the HelloWorld project, go to the Build tab of the toolbar and click Libraries in the FTDI Libraries tab group.



Figure 13 shows the Project Library dialog box which appears. To add a library, click the desired archive file in the left hand pane and press the Add button. The list of added libraries is displayed within the right hand pane.

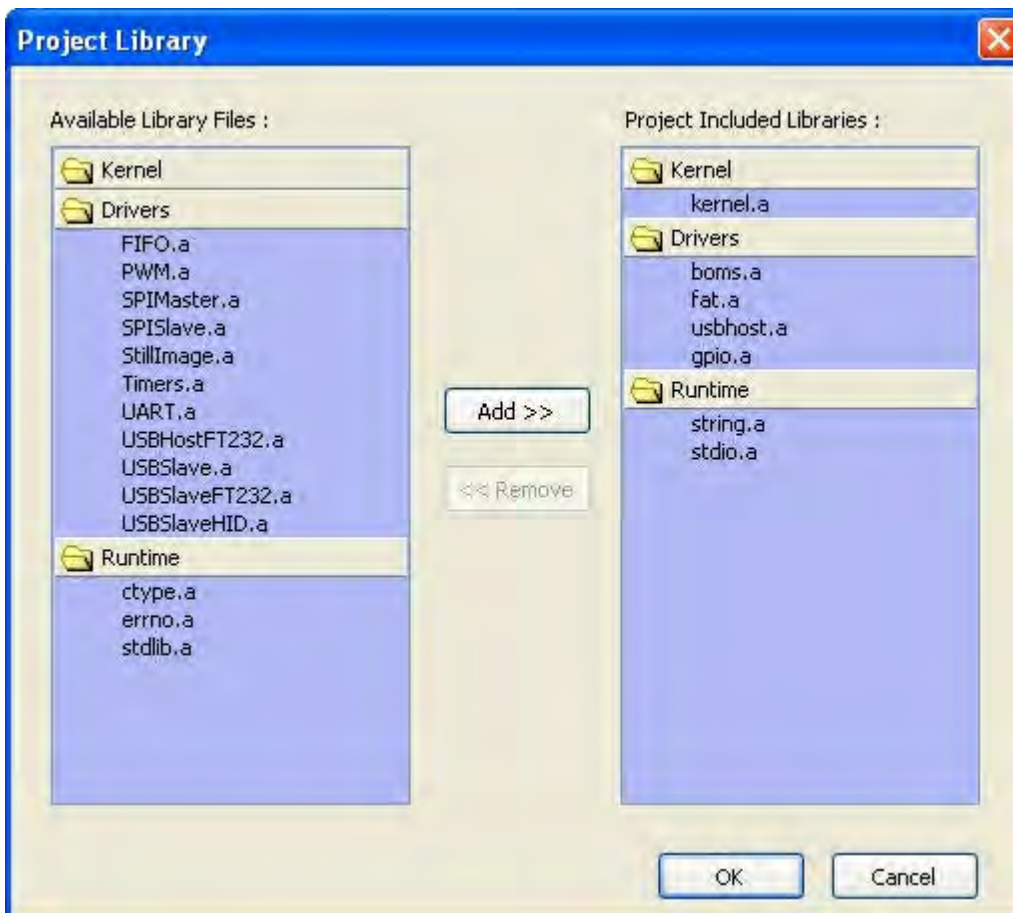


Figure 13

The list of archive files that are required for the HelloWorld project are: Kernel.a, BOMS.a, fat.a, usbhost.a, gpio.a, stdio.a and string.a.

The corresponding header files must also be added to the project. This is achieved by going to the Build tab and selecting Header Files. Adding header files is done in the same manner as library files. The header files required for HelloWorld are: vos.h, USBHost.h, USB.h, BOMS.h, Fat.h, GPIO.h, stdio.h and string.h.

Application Code

This section illustrates writing the main application code. A full listing is in the [Getting Started Code Listing](#) topic.

There are three distinct parts to a VNC2 application.

The first of these is the [includes section](#) and global definitions; this is where declarations of the Kernel services, runtime libraries and driver header files that are used within the application are.

The second section is the [main function](#); this is the entry point into the application and must only be defined once. Within this function most of the setup and IOMux routines, as well as initializing application threads, are taken care of.

The final component is [user threads](#); these contain the main functionality of the system. An application can have any number of user threads, however in this simple example there is only one. In our application we will only have one thread. When it is created we require to keep a handle to that thread. This is of type `vos_tcb_t` and defined as a global.

```
vos_tcb_t      *tcbFirmware;
```

Driver Includes and Handles

The head of the HelloWorld.c file must contain include statements for all the header files, Kernel, drivers and runtime libraries that are used. The files are the same as the header files that were added during the FTDI libraries section of this tutorial.

```
#include "vos.h"
#include "USBHost.h"
#include "USB.h"
#include "BOMS.h"
#include "FAT.h"
#include "GPIO.h"
#include "string.h"
#include "helloWorld.h"
```

As well as header files there must also be declarations for any global variables that are to be used throughout the application. When an FTDI driver is opened Device Manager returns a unique handle for that device, each handle is of type `VOS_HANDLE`, declared within `devman.h`. These handles are used throughout the application to uniquely identify each device so are therefore declared as global variables.

```
VOS_HANDLE      hUsb,
                 hBOMS,
                 hFAT,
                 hGpio;
```

Main Function

The main function is the entry point each time the application is run. Within this routine are most of the initialization routines which are run before starting the application threads.

To begin, declare a context for the USB Host and GPIO drivers, the context is used later to configure the device before opening it.

```
void main(void)
{
    usbhost_context_t usb_ctx;
    gpio_context_t gpioCtx;
```

Next, initialize the Kernel for the number of devices being used, the time slice for each thread (Quantum) and the interval for timer interrupts (tick). The `NUMBER_OF_DEVICES` comes from the header file where it was explicitly set to 3; when writing a system that requires more devices it is important to remember to increase this number otherwise any extra devices are not registered with the Kernel and Device Manager. The default clock frequency for the CPU is 48MHz; this has been added for completeness.

```
vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);
vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);
```

VNC2 features several peripherals, however it is not possible to route all of these signals concurrently. To allow signals to be routed to their required pins VNC2 comes with an I/O Multiplexer (IOMux) which provides a simple API to allow signals to be routed to specific pins. FTDI provides an IOMux configuration utility as part of the installation, giving a visual representation of the pins to aid with routing signals. The utility will generate C code that can be cut-n-paste straight into any application.

The IOMux code used to routed to connect to a V2EVAL board allows routing to 64, 48 or 32 pin devices. The code here is edited for clarity, refer to [Getting Started Code Listing](#) for the full listing.

```
if (vos_get_package_type() == VINCULUM_II_64_PIN)
{
// GPIO port A bit 1 to pin 12
vos_iomux_define_output(12,IOMUX_OUT_GPIO_PORT_A_1); //LED3
...
vos_iomux_define_input(42,IOMUX_IN_UART_CTS_N); //UART CTS#
}
else if (vos_get_package_type() == VINCULUM_II_48_PIN)
{
// GPIO port A bit 1 to pin 12
vos_iomux_define_output(12,IOMUX_OUT_GPIO_PORT_A_1); //LED3
...
vos_iomux_define_input(34,IOMUX_IN_UART_CTS_N); //UART CTS#
}
else // VINCULUM_II_32_PIN
{
// GPIO port A bit 1 to pin 12
vos_iomux_define_output(12,IOMUX_OUT_GPIO_PORT_A_1); //LED3
...
vos_iomux_define_input(26,IOMUX_IN_UART_CTS_N); //UART CTS#
}
```

Next, configure devices and open a handle to each of these devices. VNC2 has two USB Host interfaces available. HelloWorld configures the second USB Host to connect to the flash drive. Within the `usbhost_context_t`, declare the maximum number of interfaces to be enumerated. When calling the [usbhost_init\(\)](#) function, specify the device number to register with Device Manager. In this example it is only necessary to register the second USB Host interface. Therefore pass -1 as the first parameter and the device number for our USB Host (from the header file) as the second parameter. The third parameter is USB host context.

```
// Initialize the USBHost driver and open a handle to the device...
usb_ctx.if_count = 4; // Use a max of 4 USB interfaces
usbhost_init(-1, VOS_DEV_USB_HOST, &usb_ctx);
```

To initialize the GPIO, the port number to be used (A,B,C,D or E) is passed with the device context when calling the [gpio_init\(\)](#) function. This is illustrated as follows:

```
// Initialize the GPIO driver and open a handle to the device...
gpioCtx.port_identifier = GPIO_PORT_A;
gpio_init(VOS_DEV_GPIO,&gpioCtx);
```

The [BOMS Driver](#) and [FAT File System](#) Driver are simpler to call and do not require a context to initialize the device, again they pass the device number to Device Manager to register the driver when [boms_init\(\)](#) and [fatdrv_init\(\)](#) are called.

```
// Initialize the BOMS driver and open a handle to the device...
boms_init(VOS_DEV_BOMS);
fatdrv_init(VOS_DEV_FAT);
```

All user application threads must be declared within the main routine. When creating a thread, use [vos_create_thread\(\)](#) and pass a pointer to the thread function. In this example a forward declaration for a thread called `firmware` was created in the header file and this is the name that is passed into [vos_create_thread\(\)](#).

The first parameter in [vos_create_thread\(\)](#) is the thread priority; this value determines the priority of the thread in relation to other application threads. The thread priority must be a value between 1 and 31 with 1 being the lowest priority thread.

The `SIZEOF_FIRMWARE_TASK_MEMORY`, as defined within the header file, is the amount of stack usage that is allocated to the application thread. The stack size required for a thread depends on its complexity, for this application a stack size of 0x1000 will be more than adequate.

The last parameter is the arg size field; [vos_create_thread\(\)](#) allows for any number of extra parameters to be passed into the function. The arg size field must reflect the total size of the arguments passed into the function. In this example the thread has no arguments and therefore arg size is zero.

```
// Create our application thread here...
vos_create_thread(29, SIZEOF_FIRMWARE_TASK_MEMORY, firmware, 0);
```


The last step within the main routine is to call the [Kernel Scheduler](#) to start the application threads. The call to [vos_start_scheduler\(\)](#) is an indication that setup and initialization is finished, and control passes from main to the application threads.

```
// Start the scheduler to kick off our thread...
vos_start_scheduler();
```

Application Thread

This is the body of the application and contains firmware code to control the VNC2. To start, declare the thread function and local variables

```
void firmware(void)
{
    unsigned char *tx_buf = "Hello World! \r\n";
    unsigned char connectstate;
    unsigned char status;
    // USB host variables
    usbhost_device_handle *ifDev;
    usbhost_ioctl_cb_t hc_iocb;
    usbhost_ioctl_cb_class_t hc_iocb_class;
    // BOMS device variables
    msi_ioctl_cb_t boms_iocb;
    boms_ioctl_cb_attach_t boms_att;
    // FAT file system variables
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_attach_t fat_att;
    FILE *file;
    // GPIO variables
    gpio_ioctl_cb_t gpio_iocb;
    unsigned char leds;
```

Firstly open the USB Host controller driver. The function [vos_dev_open\(\)](#) requires the device number of the USB host driver and returns a handle to the instance of the driver.

```
hUsb = vos_dev_open(VOS_DEV_USB_HOST);
```

Next, configure the GPIO driver so that all signals are set to output, this enables feedback to be shown through the LEDs. Control requests to drivers are performed through I/O control calls where the call to be performed is specified and any extra data required by the call is passed in.

```
hGpio = vos_dev_open(VOS_DEV_GPIO);

gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_MASK;
gpio_iocb.value = 0xff; // set all as output
vos_dev_ioctl(hGpio, &gpio_iocb);
```

To determine whether there is a USB device connected to USB Host 2, use the [VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#) IOCTL function. When a device is detected, information is relayed back to the user via the LEDs.

```
do
{
    //wait for enumeration to complete
    hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_GET_CONNECT_STATE;
    hc_iocb.get = &connectstate;
    vos_dev_ioctl(hUsb, &hc_iocb);
    if (connectstate == PORT_STATE_ENUMERATED)
    {
        // connected to USB device and enumerated
        leds = 0xAA; vos_dev_write(hGpio,&leds,1,NULL);
    }
}
```

Now, use the [VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS](#) IOCTL to determine if the connected device is a BOMS class device. The class, subclass and protocol of the device must also be passed to this IOCTL. If the driver finds a device matching the BOMS flash disk then a handle is returned in the get section of the IOCTL block.

```
// find BOMS class device
hc_iocb_class.dev_class = USB_CLASS_MASS_STORAGE;
```

```
hc_iocb_class.dev_subclass = USB_SUBCLASS_MASS_STORAGE SCSI;
hc_iocb_class.dev_protocol = USB_PROTOCOL_MASS_STORAGE_BOMS;
// user ioctl to find first hub device
hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
hc_iocb.handle.dif = NULL;
hc_iocb.set = &hc_iocb_class;
hc_iocb.get = &ifDev;
vos_dev_ioctl(hUsb, &hc_iocb);
if(!ifDev)
{
    // We didn't manage to find a device matching the required class.
    break;
}
```

Using the ifDev handle as received from the previous IOCTL, attach the BOMS driver to the flash disc using the [BOMS MSI_IOCTL_BOMS_ATTACH](#) IOCTL.

```
hBoms = vos_dev_open(VOS_DEV_BOMS);

// Attach BOMS driver to our USB Flash Disk
boms_att.hc_handle = hUsb;
boms_att.ifDev = ifDev;
boms_iocb.ioctl_code = MSI_IOCTL_BOMS_ATTACH;
boms_iocb.set = &boms_att;
boms_iocb.get = NULL;
status = vos_dev_ioctl(hBoms, &boms_iocb);
```

The FAT driver is required to match the file structure on BOMS devices and allow reading and writing of files. Calling the [FAT_IOCTL_FS_ATTACH](#) will cause subsequent file system operations to be sent to the BOMS disk.

```
hFat = vos_dev_open(VOS_DEV_FAT);

// Attach the FAT driver to the BOMS device
fat_ioctl.ioctl_code = FAT_IOCTL_FS_ATTACH;
fat_ioctl.set = &fat_att;
fat_att.boms_handle = hBoms;
fat_att.partition = 0;
status = vos_dev_ioctl(hFAT, &fat_ioctl);
```

Once the FAT file system and BOMS are attached then the [stdio](#) library can be initialised with the [fsAttach](#) function.

```
fsAttach(hFAT);
```

The [stdio](#) library can now be used to access files on the disk.

Notice the use of the [strlen](#) function as defined within the [string](#) runtime library to calculate the length of the Hello World buffer.

```
file = fopen("TEST.TXT", "a+");
fwrite(tx_buf, strlen(tx_buf), sizeof(char), file);
fclose(file);
```

Follow the instructions in [Building Your First Application](#) to build the project and flash the VNC2.

2.6 Code Listing

HelloWorld.h

```
/*
** HelloWorld.h
**
** Copyright © 2010 Future Devices International Limited
**
** C Header file for Vinculum II sample application
** Main module
**
** Author: FTDI
```

```
** Project: Vinculum II
** Module: Vinculum II Sample Applications
** Requires: VOS BOMS FAT USBHost GPIO STDIO
** Comments: Uses stdio to write files on flash disk
**
** History:
** 1 - Initial version
**
*/

#define SIZEOF_FIRMWARE_TASK_MEMORY 0x1000

#define NUMBER_OF_DEVICES          4
#define VOS_DEV_USB_HOST          0
#define VOS_DEV_BOMS               1
#define VOS_DEV_FAT                2
#define VOS_DEV_GPIO              3

#define LED0    0x02
#define LED1    0x04
#define LED2    0x20
#define LED3    0x40
```

HelloWorld.c

```
/*
** HelloWorld.c
**
** Copyright © 2010 Future Devices International Limited
**
** C Source file for Vinculum II sample application
** Main module
**
** Author: FTDI
** Project: Vinculum II
** Module: Vinculum II Sample Applications
** Requires: VOS BOMS FAT UART USBHost GPIO STDIO
** Comments: Uses stdio to write files on flash disk
**
** History:
** 1 - Initial version
**
*/

#include "vos.h"

#include "USBHost.h"
#include "USB.h"
#include "MSI.h"
#include "BOMS.h"
#include "FAT.h"
#include "GPIO.h"

#include "stdio.h"
#include "string.h"

#include "HelloWorld.h"

VOS_HANDLE      hUsb,
                hBoms,
                hGpio,
                hFAT;

vos_tcb_t      *tcbFirmware;
```

```
char *tx_buf = "Hello World! \n";

void firmware(void);

void main(void)
{
    // USB Host configuration context
    usbhost_context_t usb_ctx;
    // GPIO configuration context
    gpio_context_t gpioCtx;

    vos_init(10, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);
    vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);

    if (vos_get_package_type() == VINCULUM_II_64_PIN)
    {
        // GPIO port A bit 1 to pin 12
        vos_iomux_define_output(12, IOMUX_OUT_GPIO_PORT_A_1); //LED3
        // GPIO port A bit 2 to pin 13
        vos_iomux_define_output(13, IOMUX_OUT_GPIO_PORT_A_2); //LED4
        // GPIO port A bit 5 to pin 29
        vos_iomux_define_output(29, IOMUX_OUT_GPIO_PORT_A_5); //LED5
        // GPIO port A bit 6 to pin 31
        vos_iomux_define_output(31, IOMUX_OUT_GPIO_PORT_A_6); //LED6
        // UART to V2EVAL board pins
        vos_iomux_define_output(39, IOMUX_OUT_UART_TXD); //UART Tx
        vos_iomux_define_input(40, IOMUX_IN_UART_RXD); //UART Rx
        vos_iomux_define_output(41, IOMUX_OUT_UART_RTS_N); //UART RTS#
        vos_iomux_define_input(42, IOMUX_IN_UART_CTS_N); //UART CTS#
    }
    else if (vos_get_package_type() == VINCULUM_II_48_PIN)
    {
        // GPIO port A bit 1 to pin 12
        vos_iomux_define_output(12, IOMUX_OUT_GPIO_PORT_A_1); //LED3
        // GPIO port A bit 2 to pin 13
        vos_iomux_define_output(13, IOMUX_OUT_GPIO_PORT_A_2); //LED4
        // GPIO port A bit 4 to pin 45
        vos_iomux_define_output(45, IOMUX_OUT_GPIO_PORT_A_4); //LED6
        // GPIO port A bit 5 to pin 46
        vos_iomux_define_output(46, IOMUX_OUT_GPIO_PORT_A_5); //LED5
        // UART to V2EVAL board pins
        vos_iomux_define_output(31, IOMUX_OUT_UART_TXD); //UART Tx
        vos_iomux_define_input(32, IOMUX_IN_UART_RXD); //UART Rx
        vos_iomux_define_output(33, IOMUX_OUT_UART_RTS_N); //UART RTS#
        vos_iomux_define_input(34, IOMUX_IN_UART_CTS_N); //UART CTS#
    }
    else // VINCULUM_II_32_PIN
    {
        // GPIO port A bit 1 to pin 12
        vos_iomux_define_output(12, IOMUX_OUT_GPIO_PORT_A_1); //LED3
        // GPIO port A bit 2 to pin 14
        vos_iomux_define_output(14, IOMUX_OUT_GPIO_PORT_A_2); //LED4
        // UART to V2EVAL board pins
        vos_iomux_define_output(23, IOMUX_OUT_UART_TXD); //UART Tx
        vos_iomux_define_input(24, IOMUX_IN_UART_RXD); //UART Rx
        vos_iomux_define_output(25, IOMUX_OUT_UART_RTS_N); //UART RTS#
        vos_iomux_define_input(26, IOMUX_IN_UART_CTS_N); //UART CTS#
    }

    // use a max of 4 USB devices
    usb_ctx.if_count = 4;
    usbhost_init(-1, VOS_DEV_USB_HOST, &usb_ctx);

    boms_init(VOS_DEV_BOMS);
    fatdrv_init(VOS_DEV_FAT);
}
```

```
gpioCtx.port_identifier = GPIO_PORT_A;
gpio_init(VOS_DEV_GPIO,&gpioCtx);

tcbFirmware = vos_create_thread(29, SIZEOF_FIRMWARE_TASK_MEMORY, firmware, 0);

vos_start_scheduler();

main_loop:
    goto main_loop;
}

void firmware(void)
{
    unsigned char connectstate;
    unsigned char status;

    usbhost_device_handle *ifDev;
    usbhost_ioctl_cb_t hc_iocb;
    usbhost_ioctl_cb_class_t hc_iocb_class;

    msi_ioctl_cb_t boms_iocb;
    boms_ioctl_cb_attach_t boms_att;

    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_attach_t fat_att;

    gpio_ioctl_cb_t gpio_iocb;
    unsigned char leds;

    FILE *file;

    // open host controller
    hUsb = vos_dev_open(VOS_DEV_USB_HOST);

    // open GPIO device
    hGpio = vos_dev_open(VOS_DEV_GPIO);

    gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_MASK;
    gpio_iocb.value = 0xff;    // set all as output
    vos_dev_ioctl(hGpio, &gpio_iocb);

    do
    {
        //wait for enumeration to complete
        vos_delay_msecs(250);
        leds = LED0;  vos_dev_write(hGpio,&leds,1,NULL);
        vos_delay_msecs(250);
        leds = 0;  vos_dev_write(hGpio,&leds,1,NULL);

        // user ioctl to see if bus available
        hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_GET_CONNECT_STATE;
        hc_iocb.get = &connectstate;
        vos_dev_ioctl(hUsb, &hc_iocb);

        if (connectstate == PORT_STATE_ENUMERATED)
        {
            leds = LED1;  vos_dev_write(hGpio,&leds,1,NULL);

            // find and connect a BOMS device

            // USBHost ioctl to find first BOMS device on host
            hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
            hc_iocb.handle.dif = NULL;
            hc_iocb.set = &hc_iocb_class;
            hc_iocb.get = &ifDev;
```

```
hc_iocb_class.dev_class = USB_CLASS_MASS_STORAGE;
hc_iocb_class.dev_subclass = USB_SUBCLASS_MASS_STORAGE_SCSI;
hc_iocb_class.dev_protocol = USB_PROTOCOL_MASS_STORAGE_BOMS;

if (vos_dev_ioctl(hUsb, &hc_iocb) != USBHOST_OK)
{
    leds = LED3; vos_dev_write(hGpio,&leds,1,NULL);
    vos_delay_msecs(1000);
    break;
}

// now we have a device, initialise a BOMS driver for it
hBoms = vos_dev_open(VOS_DEV_BOMS);

// BOMS ioctl to attach BOMS driver to device on host
boms_iocb.ioctl_code = MSI_IOCTL_BOMS_ATTACH;
boms_iocb.set = &boms_att;
boms_iocb.get = NULL;
boms_att.hc_handle = hUsb;
boms_att.ifDev = ifDev;

status = vos_dev_ioctl(hBoms, &boms_iocb);
if (status != MSI_OK)
{
    leds = LED3; vos_dev_write(hGpio,&leds,1,NULL);
    vos_delay_msecs(1000);
    break;
}

// now we have the BOMS connected open the FAT driver
hFAT = vos_dev_open(VOS_DEV_FAT);

fat_ioctl.ioctl_code = FAT_IOCTL_FS_ATTACH;
fat_ioctl.set = &fat_att;
fat_att.boms_handle = hBoms;
fat_att.partition = 0;

status = vos_dev_ioctl(hFAT, &fat_ioctl);
if (status != FAT_OK)
{
    leds = LED3; vos_dev_write(hGpio,&leds,1,NULL);
    vos_delay_msecs(1000);
    break;
}

// lastly attach the stdio file system to the FAT file system
fsAttach(hFAT);

// now call the stdio runtime functions
file = fopen("TEST.TXT", "a+");

if (file == NULL)
{
    leds = LED3; vos_dev_write(hGpio,&leds,1,NULL);
    vos_delay_msecs(1000);
    break;
}

if (fwrite(tx_buf, strlen(tx_buf), sizeof(char), file) == -1)
{
    leds = LED3; vos_dev_write(hGpio,&leds,1,NULL);
    vos_delay_msecs(1000);
}

if (fclose(file) == -1)
{
    leds = LED3; vos_dev_write(hGpio,&leds,1,NULL);
```

```
        vos_delay_msecs(1000);
    }

    leds = LED1;  vos_dev_write(hGpio,&leds,1,NULL);

    fat_ioctl.ioctl_code = FAT_IOCTL_FS_DETACH;
    if (vos_dev_ioctl(hFAT, &fat_ioctl) != FAT_OK)
    {
        leds = LED3;  vos_dev_write(hGpio,&leds,1,NULL);
        vos_delay_msecs(1000);
        break;
    }

    vos_dev_close(hFAT);

    boms_iocb.ioctl_code = MSI_IOCTL_BOMS_DETACH;
    if (vos_dev_ioctl(hBoms, &boms_iocb) != MSI_OK)
    {
        leds = LED3;  vos_dev_write(hGpio,&leds,1,NULL);
        vos_delay_msecs(1000);
        break;
    }

    vos_dev_close(hBoms);

    leds = 0;  vos_dev_write(hGpio,&leds,1,NULL);

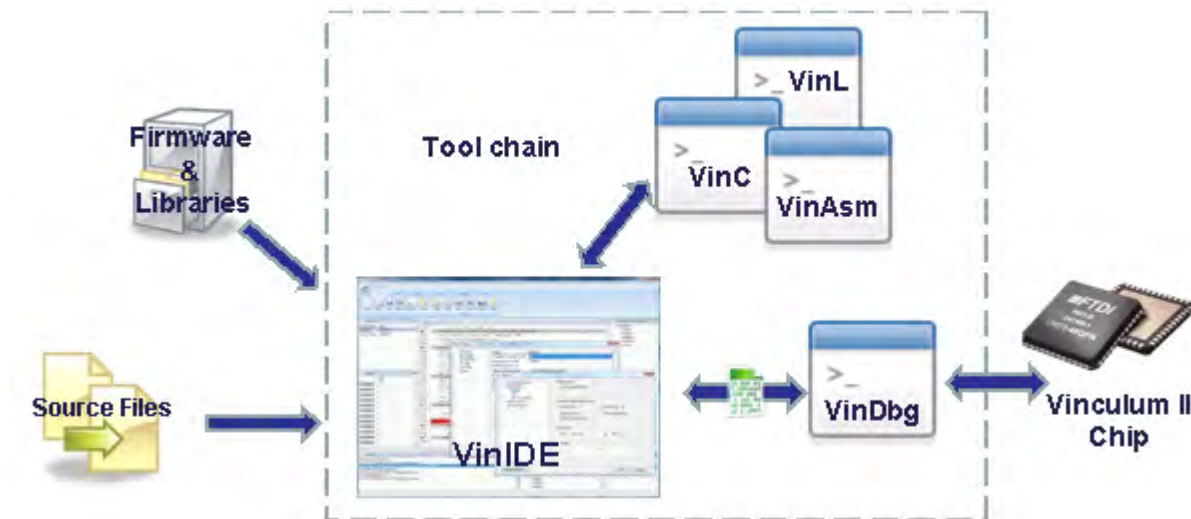
    vos_delay_msecs(5000);
}

} while (1);
}
```

3 Toolchain

FTDI has created set of tools for Vinculum II (VNC2) which includes a C compiler, assembler, linker, debugger and integrated development environment.

These tools facilitate application development on VNC2 using a kernel, device driver and runtime libraries provided by FTDI.



3.1 Toolchain Basics

The toolchain is designed to integrate with the firmware (RTOS, drivers and libraries) supplied by FTDI.

Addresses for ROM and RAM are handled differently:

- All ROM addresses are specified in word addresses as this is the size of data which is stored in the Flash ROM.
- All memory addresses are specified as byte addresses.

3.2 VinC Compiler

VNC2 C Compiler is a tool that generates machine code instructions. Input of compiler is a C source file which may contain standard C syntax with some deliberate restrictions and supports VNC2 specific language extensions.

3.2.1 Introduction

VNC2 C compiler (VinC) is a fully fledged C compiler developed for the VNC2 and it gives the developers the capability to quickly produce efficient code using C high level language. VinC is an ANSI C compatible compiler (with restrictions) with built in language level functions to access the Harvard style memory architecture of VNC2 and I/O ports.

3.2.1.1 Compiler Overview

VinC Compiler, implemented as part of the overall Toolchain, for the VNC2 is:

- ANSI 'C' compatible (with restrictions)
- Support for structures, unions and arrays - Structures and arrays can be comprised of base data types or other data structures
- Language level Support for specifying and accessing flash memory
- Support for pointers including function pointers. Pointers can be used the usual ways
- Support for typedef

-
- Support for ANSI C control flow statements, selection statements and operations
 - Support Inline Assembly - Inline assembly allows hand crafted assembly code to be used when necessary.
 - Efficient RAM usage - local variables in different code sections can share memory. The linker analyzes the program to prevent any clashes.
 - Comes with a Preprocessor
 - Comes with the ANSI C Library subset
 - Produces optimized code for VNC2

3.2.1.2 ANSI C Feature Support Summary

VinC is a compiler that supports following features from ANSI C.

1.Data types

- a.Basic Data Types
- b.Arrays
- c. Structures
- d.Unions
- e.Pointers

2.Operations

- a.Precedence and Order of Evaluation
- b.Arithmetic Conversions
- c. Postfix Operators
- d.C Unary Operators
- e.Cast Operators
- f. Multiplicative Operators
- g.C Additive Operators
- h.Bitwise Shift Operators
- i. Relational and Equality Operators
- j. C Bitwise Operators
- k. Logical Operators
- l. Conditional-Expression Operator
- m.Assignment Operators
- n.Sequential-Evaluation Operator
- o.Side effects

3.Control flow

- a.for loop
- b.while loop
- c. do-while loop
- d.Label and Jump statements
- e.return,
- f. break, continue,
- g.switch

4.Other Statements

- a.Function calls
- b.Compound statements

- c. Expression and null statements
- d. Selection statements (if-then-else, switch-case)
- e. Conditional

3.2.1.3 C Language Restrictions

VinC has added some specific restrictions on C language to support the VNC2 architecture, these restrictions are:

- Cascading typecast are not supported
- Function pointer definitions as parameters in function declaration not supported
- Passing or returning structures and unions by value to functions not supported
- Vacuous definition for struct or union not supported
- Passing structures or unions as parameters to a function is not supported: Structures and unions have to be passed as pointers
- Returning structures or unions from function calls is not supported: Returning a pointer to a structure or union is allowed
- Floating point data type and arithmetic is not supported
- Declarations must have a type specifier, a type qualifier is not sufficient (i.e. int will not be assumed)
- Bit ranges in structures or unions must have a type specifier (i.e. int will not be assumed)

3.2.2 Compiler Command Line Options

VinC allows the user control various stages of compiling with the help of command line options. In addition, VinC also acts as a driver for other tools in the Toolchain and allows it to be controlled with additional command line options. The following command line options are supported in the VinC Compiler:

```
VinC [options] [file ...] [-L linker options]
```

Option	Description
-E	Run the preprocessor and stop.
-S	Stop processing after preprocessing and compilation.
-c	Preprocess, compile and assemble but do not invoke linker.
-O level	Specify optimisation for compiler.
-d level	Specify debug information level for compiler.
-D macro=defn	Defines macro with optional defn in preprocessor.
-U macro	Undefines macro in preprocessor.
-I dir	Add a search directory for include files.
-o file	Specify output filename.
-l file	Specify log filename.
--save-temps	Save all temporary files.
--save-temp	Save temporary files: 'a' assembler, 'i' preprocessor, 'o' object.
--combine	Combine source files on command line into single input file for compiling.
--library	Create a single output file for all source files on command line.
-v	Verbose output of command lines for tools.
-q	Quieten output of command lines for tools.
--version	Display tool version.

<code>--all-versions</code>	Display all tool versions.
<code>--max-warn limit</code>	Limit the number of warnings displayed.
<code>--max-err limit</code>	Limit the number of errors displayed.
<code>--help</code>	Display this help message.
<code>-L</code>	Process all following options as linker options.

3.2.2.1 Compiler File Type

The compiler can control the 4 stages of compilation: Preprocessing, compilation, assembling and linking. The type of file passed to the compiler and the compiler command parameters determine which stages are performed.

The following file extensions are used for specifying the start stage of compilation:

<code>file.c</code>	An unpreprocessed C source file which will be preprocessed prior to compilation
<code>file.i</code>	A preprocessed C source file which will be compiled, assembled and linked.
<code>file.asm</code>	An assembler file to be assembled and linked.
<code>file.obj</code>	An object file to be sent to the linker

3.2.2.2 Compile Stage Selection

It is possible to specify which stage to stop to the compiler. These flags can stop after preprocessing, compilation, assembling or allow linking to complete.

<code>-E</code>	
<code>--pp</code>	Run the preprocessor and stop. Any unpreprocessed C source files will be preprocessed and given a '.i' extension.
<code>-S</code>	
<code>--ppx</code>	Stop processing after preprocessing and compilation. Any source files will be compiled to a assembly file and given a '.asm' extension.
<code>-c</code>	
<code>--ppca</code>	Preprocess, compile and assemble but do not invoke linker. Source files will be compiled or assembled to an object file and given a '.obj' extension.

3.2.2.3 Compiler Output

The following options control the type of output of compiler.

<code>-o</code>	
<code>--output file</code>	Specifies the output file name for the last stage of compilation performed. If this option is set then there can only be one input file specified on the command line. The output file name also overrides any file extension that may be given to a particular output file. The default action is to use the filename of the input file and modify the file extension for the output file.
<code>--library</code>	Combine all source files at the compile stage into a single assembler file. This cannot be used in conjunction with C source files and assembler or object files. The global namespace in each C source file is kept separate.
<code>--combine</code>	Similar to <code>--library</code> except that the global namespaces of C source files are combined.
<code>-T</code>	
<code>--save-temps</code>	Keeps all intermediate files during the compilation. All preprocessed C source files, assembler files and object files are retained, usually only the file produced for the final stage of compilation is retained.
<code>-t opt</code>	
<code>--save-temp opt</code>	

Keeps selected intermediate files depending on the opt parameter. 'a' to keep assembler files, 'i' for preprocessor files and 'o' for object files. Multiple options can be combined, for instance, "--save-temp ao" to retain both assembler and object files.

-q
--quiet
Reduces the output of the compiler to a minimum. Only a final status message and error messages will be displayed.

-v
--verbose
Displays the more information during compilation.

-l file
--log file
Copy preprocessor, compiler, assembler and linker output to a log file.

3.2.2.4 Compiler Information Options

-V
--version
Displays the version of the compiler.

--all-versions
Displays the version of preprocessor, compiler, assembler and linker. No further action is taken (all other command line options ignored).

-h
--help
Shows a summary of command options. No further action is taken (all other command line options ignored).

3.2.2.5 Compile Time Options

The behaviour of the compiler stage is governed by the following options.

-O level
--optimise level
Select the level of optimisation for the compiler. The level is defined as:

- 0 - No optimisation
- 1 - Register allocation only
- 2 - Register allocation and some intermediate code optimisations
- 3 - Register allocation and full intermediate code optimisations
- 4 - Register allocation, intermediate code and peephole optimisations

-d level
--debug level
Specify whether debug information is generated by the compiler in the assembler output. Available options are:

- 0 - No debug information
- 1 - Generate debug information

By default this is set to zero.

If an optimisation level is not specified then the default the optimisation level is set according to the debug level. If debugging is turned on then optimisation is set to zero (for no optimisation); if it is set to on, then the optimisation level is set to 4 for full optimisation.

3.2.2.6 Preprocessing Options

Preprocessor options can be specified on the command line for the compiler.

-D macro[=defn]
--define macro[=defn]
Predefine macros with an optional definition.

-U macro[=defn]
--undefmacro[=defn]

Remove definition of a macro.

```
-I dir
--include dir
```

Add a directory to the include directory search path.

3.2.2.7 Linker Options

There are many options which affect the linker operation. These can be specified on the compiler command line.

```
-L opts...
--linker opts...
```

When this option is encountered on the compiler command line ALL further options are passed directly to the linker and are not processed by the compiler.

3.2.3 Data Types

Compiler supports the C language standard integral data types (char, short, long, int and void) and an additional data type (port) which is used to access I/O ports directly.

Data Type Name	Size in Bits
char	8
short	16
long	32
int	32
void	0
port	8

NOTE: There is no support for floating point types.

To generate optimum code the char data type should be used as much as possible. Long and int should only ever be used when 32-bit values are required.

A declaration of an identifier is made up of a type definition followed by an identifier. A type definition must contain a type specifier. It may also contain any valid combination of type qualifiers and a storage class specifier.

3.2.3.1 Type Qualifiers

Data types can be qualified with the keywords signed or unsigned, const and volatile. All datatypes except port can be specified as rom.

signed and unsigned

These determine if the data type can be used for signed calculations. If it is signed then one data storage bit is used for a sign bit. By default all data types except port will be signed unless otherwise qualified.

NOTE: unsigned data types will produce smaller, faster code compared to signed data types.

const

A const qualifier enables type checking to ensure that its value is not modified by code in the scope of the declaration. One of its main uses is where values may be passed to functions but not modified by that function.

When const is used with pointers the following applies:

```
char val;
char * const ptr = &val;    // ptr is a constant pointer, the value it points to can be modified.
char const * ptr = &val;    // ptr is a normal pointer and can be modified, the value it points to
```

Const cannot be used before and after the pointer operator in the same declaration.

volatile

The volatile qualifier tells the compiler to not re-use the value of a data type during a calculation. It always reads a fresh value of the data each time it is required in a calculation. It is used mainly where a value may change outside the linear program flow (e.g. by an interrupt or thread).

When volatile is used with pointers the following applies:

```
char val;
char * volatile ptr = &val;    // ptr is a volatile pointer, the value it points to is not
char volatile * ptr = &val;    // ptr is a normal non-volatile pointer, the value it points to is
```

Volatile cannot be used before and after the pointer operator in the same declaration.

rom

When a storage type is qualified with rom then the data to be stored must be initialised.

Non-pointer variables may be used transparently in code but special rules apply for pointers.

If the type is a pointer then a pointer to data stored in ROM is created. For arrays and strings the compiler will store the initialisation data in the code section and a rom pointer will be generated to point to the data. All access must be via the rom pointer. A rom pointer cannot be modified so all access to the data must be made through the offset operators [].

```
rom char x[] = { 1,2,3,4,5};
rom int y[10] = { 1,2,3,4,5,2,3,4,5,6};
rom char *str = "Hello";
rom int buffersize = 20;
```

All initialisation data must be constant values or string literals. rom is not applicable to function definitions, declarations or parameters. See the topic [ROM Access](#) for example code.

3.2.3.2 Storage Type Specifiers

The data storage type defines where and how data is stored.

	Locals	Globals
auto	scope of the function destroyed when scope left	global scope
static	scope of the function preserved between calls to scope	file scope only
extern	not allowed	link to global scope
typedef	not allowed	new data type defined with file scope
enum		

The default is auto, except for function declarations whose storage class is extern.

3.2.3.3 Type Specifiers

char

Bit Size	Signed Range (decimal)	Unsigned Range(decimal)
8	-128 to 127	0 to 255

Supported Qualifiers: signed or unsigned, volatile, const, pointer, rom

Default Qualifiers: signed

Supported Storage Type: auto, static, extern, typedef

Remarks: This is the basic 8-bit data type for storage and stores a single byte of data.

Example:

```
char x = 4; // simple char variable
unsigned char * const y; // pointer to a constant char value
```

short

Bit Size	Signed Range (decimal)	Unsigned Range(decimal)
16	-32768 to 32767	0 to 65535

Supported Qualifiers: signed or unsigned, volatile, const, pointer, rom

Default Qualifiers: signed

Supported Storage Type: auto, static, extern, typedef

Remarks: This is a short integer data type and stores a single word or two bytes of data.

long, int

Bit Size	Signed Range (decimal)	Unsigned Range(decimal)
32	-2147483648 to 2147483647	0 to 4294967295

Supported Qualifiers: signed or unsigned, volatile, const, pointer, rom

Default Qualifiers: signed

Supported Storage Type: auto, static, extern, typedef

Remarks: This is a long integer data type and stores two words or four bytes of data. This is the default type specifier used if it is not explicitly specified.

void

Bit Size	Signed Range (decimal)	Unsigned Range(decimal)
0	N/A	N/A

Supported Qualifiers: signed or unsigned, volatile, const, pointer, rom

Default Qualifiers: signed

Supported Storage Type: auto, static, extern, typedef

Remarks: void is a special data type that does not hold any data. Only a pointer to void (interpreted as a pointer to anything) can be used to declare identifiers. Another use is to mark cases where no data is to be transferred or when a pointer is to an unspecified data type.

Example:

```
void *p = &x; // pointer to an unknown data type
void main(void) // function with no parameters and returning no data
```

port

Bit Size	Signed Range (decimal)	Unsigned Range(decimal)
8	N/A	0 to 255

Supported Qualifiers: N/A

Default Qualifiers: unsigned, volatile

Supported Storage Type: auto, extern

Remarks: port is a special type that allows direct access to I/O ports. Ports must be initialised with an I/O register address when declared using the @ operator. It is not possible to have a static or typedef port. Allowable I/O address range is 0 to 512 (0x0 to 0x1ff).

Examples:

```
port interrupt_reg@200; // define interrupt register at I/O address 200
interrupt_reg = 4;      // clear interrupt register bit
```

struct and union

struct format: struct <structure identifier (optional)> {
 <type definition> <member name identifier>(<bit range>);
 ...
} <structure variable (optional)>;

union format: union <union identifier (optional)> {
 <type definition> <member name identifier>(<bit range>);
 ...
} <union variable (optional)>;

Remarks: Both structures and unions can be defined from either base data types (except ports), other structures, enumerations, arrays, typedefs and pointers. The format of struct and union is the same.

Either the variable or the identifier must be present in a declaration of a struct or union. Both may be used to make a definition and a variable in the same declaration.

Bitfields may be specified for base data types only by using the range operator (:) after the member name identifier. This value must be a constant value and must always specify a size less than or equal to the size of the base type in the type definition. It must always be padded to fill the whole size of the base data type and is never allowed to overrun the end of the base data type.

enum

enum format: enum <enum identifier (optional)> {
 <constant identifier> (= <constant value>);
 ...
} <enum variable (optional)>;

Remarks: An enumeration creates a range of identifiers with constant values which will, by default, increment by one as the list is defined. It is also possible to specify a value for an identifier, in this case the list will continue to increment from the specified value.

Either the variable or the identifier must be present in a declaration of an enum. Both may be used to make a definition and a variable in the same declaration.

All variables generated with an enumeration are of type int. In the absence of a constant value, the first enumerator is assigned the constant value zero.

typedefs

typedef format: typedef <type definition> <identifier>;

Remarks: New types can be defined using the typedef keyword. The resulting identifier can be used in place of the type definition.

The base data types void, char, short, long, int or valid combinations of enumerations, structures, unions and arrays may be used in the definition of the new type. Valid combinations of qualifiers and storage types are also allowed.

Strings

string format: "<string text>"

Remarks: String are enclosed in double quotes. It is not allowed to interrupt string literals by closing then reopening the double quotes during a string definition.

All non-printable ASCII characters, single quotes, double quotes, question marks and backslashes must be represented with escape sequences in strings.

\?	ASCII character	? is a decimal value of the ASCII character to use. (\0 for NUL character)
\x?	ASCII character	? is a hexadecimal value representing the ASCII character to
\a	0x07	Bell
\b	0x08	Backspace
\f	0x0c	Form Feed
\n	0x0a	Carriage Return
\r	0x0d	Line Feed
\t	0x09	Horizontal Tab
\v	0x0b	Vertical Tab
\'	0x27	Single quote character
\"	0x22	Double quote character
\?	0x3f	Question mark
\\	0x5c	Backslash

Arrays

typedef <type definition> <array identifier> [<constant number of elements (optional)>] (=
format: { <constant value>...}) ;

Remarks: An array is used to hold multiple data types in a contiguous sequence. The data types may be any of the base data types (except port), structures, enumerations, other arrays, typedefs and pointers.

The number of elements must be specified or it must be initialised with data to reserve storage. If it is not specified or initialised then it will be assumed to be a pointer. If the number of elements is not specified but the array is initialised then the number of initialisation data elements is used as number of elements in the array.

The number of elements in the array must be a constant value. Likewise, the initialisation data must be constant values too.

Pointers

typedef <type definition> * <identifier> (= <constant value>);
format:

Remarks: A pointer is a variable that holds the memory address of something. It can be a pointer to any of the base data types (except port), structures, enumerations, arrays, typedefs and other pointers. The * symbol is used to form a pointer in the type definition.

Pointers to other identifiers or functions are allowed.

Examples:

```
void * ptr; // pointer to a void (pointer to anything)
int *fn(int); // pointer to a function returning int and taking a single int as a parameter
```

Constants

Comments: A constant can be written as a decimal, octal or hexadecimal value, and have an optional unsigned specified.

Constant decimals, other than zero, are not prefixed by any characters. They may have a capital 'U' postfixed to the value to indicate that it is unsigned.

Octal numbers are prefixed by a zero ('0') character. Octals may not be signed

Hexadecimal values are always unsigned and are prefixed by the characters "0x" or "0X".

Binary numbers are prefixed by the characters "0b" or "0B".

Character constants are be represented by a single character in single quotes. The escape sequences used for strings apply to character constants.

Constants may also have a capital 'L' appended to indicate that they are to be treated as a long integer. A non-zero decimal number may not be prefixed with a zero ('0') character or it will be interpreted as an octal value.

Examples:

```
int x = 0765;      // octal constant
int q = 0xaaaaaaaa; // hexadecimal constant
signed char z = 254U; // set a signed value with the equivalent unsigned value
int w = 0;        // zero
```

3.2.3.4 Data Conversion References

Data is converted between different variable sizes in expressions according to the following rules. The left column is the data size on the left side of an expression, right is the data size on the right.

Left	Right	Result	Bits affected
char	char	OK	
char	short	Type mismatch	Right 8 to 16 lost
char	int or long	Type mismatch	Right 8 to 32 lost
short	char	OK	
short	short	OK	
short	int or long	Type mismatch	Right 16 to 32 lost

In an assignment expression, when the left and right operand of data type is char

Example

```
char cVar1;
char cVar2;
cVar1 = cVar2 ;
```

In the assignment expression, when the left operand of type is char and right operand of type is short

Example

```
char cVar1;
short sVar2;
cVar1 = sVar2 ; //Warning type mismatch
```

In the assignment expression, when the left operand of type is char and right operand of type is integer

Example

```
char cVar1;
int iVar2;
cVar1 = iVar2 ; //Warning type mismatch
```

In the assignment expression, when the left operand of type is short and right operand of type is character

Example

```
short sVar1;
char cVar2;
sVar1 = cVar2 ;
```

In the assignment expression, when the left operand of type is short and right operand of type is short

Example

```
short sVar1;
short sVar2;
```

```
sVar1 = sVar2 ;
```

In the assignment expression, when the left operand of type is short and right operand of type is long

Example

```
short sVar1;  
long lVar2;  
sVar1 = lVar2 ; //warning type mismatch
```

3.2.4 Special VNC2 Reference

VinC has defined calling convention in order for assembler functions to be called from C and vice versa. It also defines the Port data type as a special types to allow direct access to I/O ports. It also defines the global variables are placed in DATA segment and local variable allocated in stack.

3.2.4.1 Special Features

3.2.4.1.1 Accessing ports

Syntax:

```
PORT <name> @ <address>;
```

Description

VinC compiler defines Port data type as a special types to allow direct access to I/O ports. Port data types must be defined and initialised with an I/O register address when declared using the @ operator. PORT has to be defined with global scope, and the allowable I/O address range is 0 to 511 (0x0 to 0x1ff).

Example

```
port interrupt_reg@200;      /* define interrupt register at I/O address 200 */  
interrupt_reg = 4;          /* clear interrupt register bit */
```

3.2.4.1.2 Bit mapping

Syntax

```
<variable>.<constant offset>
```

Description

Hardware extensions are available for bitmap operations. These can be accessed using bitmapped operations in expressions.

Bitmapping of all data types is possible by post-fixing the bit offset constant after the variable name with a member operator. The maximum bit offset is one less than the size of the data type, the first bit offset is zero.

Examples

```
unsigned char x = 0xaa;  
if (x.3) return 1; // bit 3 set
```

3.2.4.1.3 ROM Access

Syntax

```
rom <type>;
```

Description

Variables, arrays and structures may be stored in ROM. The qualifier `rom` in the declaration will switch

storage from RAM to ROM.

Rom data must be initialised at declaration time and must be in global scope.

It is not allowed to create a pointer to be used to access an array or string in ROM. Therefore all data transfers from ROM must be done as an array.

Examples

To read in a data array:

```
rom char myarray[16] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};

void readdata(char *buff16byte)
{
    for (x=0; x < 16; x++)
    {
        buff16byte[x] = myarray[x];
    }
}
```

To copy a string from ROM to a RAM buffer:

```
rom char charversion[] = "Version1";

void main(void)
{
    int offset;
    char buffer[20];
    for (offset = 0; offset < 20; offset++)
    {
        if ((buffer[offset] = charversion[offset]) == 0) break;
    }
}
```

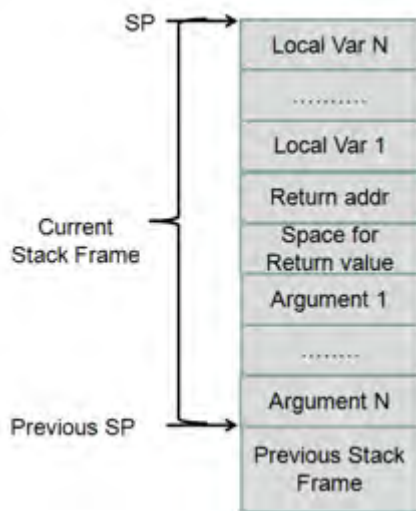
To query the program size:

```
struct mystruct {int x; short q;};
rom struct mystruct myst[4] = {{4,2}, {5,3}, {6,4}, {7,5}};

int cfpair(char pair)
{
    int x;
    short q;
    x = myst[pair].x;
    q = myst[pair].q;
    if (x > 5)
        return x;
    return q;
}
```

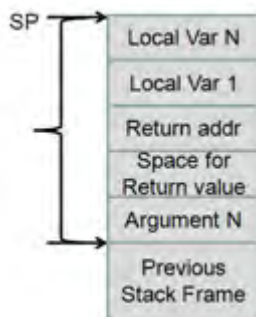
3.2.4.2 Function Call

VinC has defined a calling convention in order for assembler functions to be called from C and vice versa. As shown in the calling convention figure below, caller function pushes the parameters into the stack(right to left). Then, space is reserved for the return value in the stack by the caller and the function is called. Return address pushed into the stack by H/W and control is transferred to the function called, where called function allocate the space for local variables. Then performs the operation as defined by the function and store the return value in the appropriate place reserved for the return value. Called function clears the space allocated for local variables and return from function. Restoration of PC from the stack is done by H/W and the return value is obtained from the stack by caller.



3.2.4.2.1 Calling ASM File from C

If a C function is to be called from ASM arguments have to be pushed into the stack and return address reserved before calling the function.



Consider the following function in C:

```
unsigned char * get_offset (uint8 arg, unsigned char *start);
{
    while (arg--)
    {
        start = start + sizeof(int);
    }
    return start;
}
```

Following ASM code shows how above C function can be called in ASM. Note that parameters are pushed in the order right-to-left to appear in the correct order on the stack in the C code.

```
PUSH16 $start          ; push argument to stack
PUSH8 $arg              ; push argument to stack
SP_DEC $2               ; allocate space for return values
CALL get_offset         ; call
POP16 ret               ; pop return value from the stack
SP_INC $3               ; to clear the argument space
```

3.2.4.2.2 Calling C from ASM

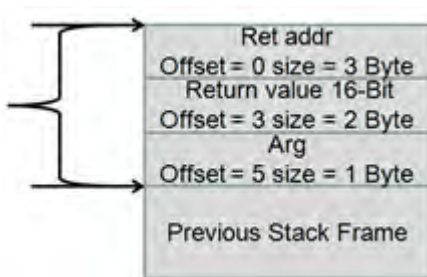
If an ASM function is to be called from C, an equivalent prototype based on the calling convention is defined and called from the C program.

For example, if following ASM function, which conforms to the calling convention, needs to be called. A C prototype based on the calling convention is defined as:

```
unsigned char * get_offset (uint8 arg, unsigned char *start);
```

Then the function `get_offset()` is called as if it is implemented in C.

```
# get pointer to device struct
get_offset:
    SP_RD8 %eax $arg
    SP_RD16 %ebx $start
get_offset_start_loop:
    CMP %eax $0
    JZ get_offset_exit
    DEC8 %eax $1
    INC16 %ebx $4
    JUMP get_offset_start_loop
get_offset_exit:
    SP_WR16 %ebx $3
    RTS
```



3.2.4.3 Architecture Issues

The VNC2 architecture supports 8, 16 and 32 bit operations on memory addresses without registers. Operations of all sizes involving only data stored in RAM are fast and efficient.

Loading 8, 16 and 32 bit values into memory is achieved in a single operation. Most operations can take one immediate value of 8, 16 or 32 bit size. Using 16 and 32 bit data value will, in practice, result in larger code though.

Signed arithmetic is fully supported by hardware. Although conversions between signed values of differing sizes is performed in software will always generate larger and slower code than unsigned

Intermediate values in calculations are assumed to be of type unsigned integer (32 bits).

There is no floating point support in hardware.

3.2.4.4 Considerations of local vs global variables

Global variables are placed in fixed addresses in memory and local variables are allocated on the call stack. The call stack moves from high memory addresses to lower memory addresses. Global variables are allocated from low addresses working up to higher addresses in memory.

VNC2 does not have general purpose registers, hence, to optimize accesses, memory locations will be emulated as registers. These memory locations may hold local variables temporarily and they might not be allocated on the call stack depending on certain conditions.

3.2.4.5 Sequence Points

Between consecutive sequence points an object's value can be modified only once by an expression. The C language defines the following sequence points

- Left operand of the logical-AND operator (&&). The left operand of the logical-AND operator is completely evaluated and all side effects complete before continuing. If the left operand evaluates to false (0), the other operand is not evaluated.

- Left operand of the logical-OR operator (||). The left operand of the logical-OR operator is completely evaluated and all side effects complete before continuing. If the left operand evaluates to true (nonzero), the other operand is not evaluated.

3.2.5 Error reference

Compiler error messages take the following form:

```
<filename> line <line number>: (error|warning|info) C<code> <description>
```

Error codes take one of the following values.

Error codes	Description
0011	<u>Preprocessing failed</u>
0012	<u>Assembling failed</u>
0013	<u>Linking failed</u>
0900	<u>internal error</u>
0901	<u>floating point not implemented</u>
0910	<u>could not open output file</u>
0911	<u>no such file or directory</u>
1000	<u>syntax error</u>
1001	<u>syntax error unterminated string</u>
1002	<u>syntax error illegal escape sequence</u>
1003	<u>syntax error character operation too long</u>
1004	<u>syntax error asm directive not allowed</u>
1005	<u>syntax error preprocessor directive not allowed</u>
1100	<u>undeclared identifier</u>
1101	<u>l-value required</u>
1103	<u>illegal use of structure or union operation</u>
1104	<u>incompatible operation</u>
1200	<u>undefined label</u>
1202	<u>integral type expected</u>
1203	<u>too many default cases</u>
1204	<u>constant expression required</u>
1205	<u>case outside of switch</u>
1206	<u>default outside of switch</u>
1207	<u>duplicate case</u>
1208	<u>misplaced continue</u>
1209	<u>misplaced break</u>
1210	<u>no body or expression for conditional</u>
1214	<u>duplicate label</u>
1300	<u>storage class extern not allowed here</u>
1301	<u>storage class static not allowed here</u>
1302	<u>storage class auto not allowed here</u>
1303	<u>storage class port not allowed here</u>
1304	<u>struct or union not allowed here</u>
1305	<u>too many storage classes in declaration</u>
1306	<u>function declaration not allowed here</u>
1307	<u>export symbol not allowed here</u>
1400	<u>no storage type</u>
1401	<u>not an allowed type</u>

1402	<u>conflicting storage type qualifier</u>
1403	<u>too many storage type qualifiers</u>
1404	<u>too many types in declaration</u>
1405	<u>type mismatch</u>
1407	<u>parameter number type mismatch (error)</u>
1408	<u>parameter mismatch in redeclaration</u>
1409	<u>parameter type mismatch (warning)</u>
1410	<u>multiple declaration</u>
1411	<u>cannot use void type in an expression</u>
2000	<u>integer arithmetic overflow</u>
2001	<u>expression not constant</u>
2002	<u>constant out of range (error)</u>
2003	<u>constant out of range (warning)</u>
2004	<u>divide by zero</u>
2005	<u>initialisation make a pointer from an integer without a cast</u>
2101	<u>cannot modify a const object</u>
2102	<u>cannot modify rom variable</u>
2201	<u>rom variable for structure member should be minimum short datatype</u>
2300	<u>size of the type is unknown or zero</u>
2301	<u>excess elements in array initialiser</u>
2302	<u>array must have at least one element</u>
2402	<u>pointer to structure required on left side of "->"</u>
2403	<u>structure required on left side of "."</u>
2404	<u>structure or union cannot contain an instance of itself</u>
2405	<u>left of pointer or member not a struct or union</u>
2500	<u>address of port is not specified</u>
2501	<u>initialisation of port is not allowed</u>
2601	<u>initialisation of extern is not allowed</u>
2700	<u>too few parameters in call</u>
2702	<u>void functions may not return a value</u>
2703	<u>function should return a value</u>
2704	<u>parameter of type void not allowed</u>
2800	<u>invalid pointer addition</u>
2801	<u>copy of volatile pointer into normal pointer</u>
2802	<u>copy of const pointer into normal pointer</u>
2803	<u>illegal port access</u>
2804	<u>not a member of struct or union</u>
2805	<u>illegal use of pointer subtraction</u>
2806	<u>pointer subtraction</u>
2807	<u>illegal function pointer declaration</u>
2808	<u>illegal use of pointer</u>
2809	<u>illegal use of array</u>
2810	<u>non portable pointer conversion</u>

2811	suspicious pointer conversion
2812	invalid indirection
2900	bit fields must contain at least one bit
2901	bit field exceeds boundary
2902	bit field exceeds the range
2903	illegal use of bit operation access
2950	local variable declarations exceed total memory size

Example

An error for a case statement outside a switch statement in file test.c line 45 will give the following message.

```
test.c line 45: (error) C1205 case outside of switch
```

3.2.5.1 Examples for General Errors

Error Description: floating point not implemented

There is no floating point support on the VNC2 hardware. The compiler will therefore not support float or double datatypes. Only integer datatypes are supported.

Example

```
float iVar; //error floating point not implemented
```

Error Description: no such file or directory

One of the source files specified in the command line or an intermediate file used by the compiler could not be found or could not be opened. Check that the file exists.

Error Description: could not open output file

An output file could not be opened for writing. Output files can either be the file named in the command line for the final stage of compilation or an intermediate file whose filename is generated by the compiler. The name of the file which caused the error is given in the error message.

Check to make sure that the file and directory is not read only and that the current user has permissions to write to that file and directory. Also check that no other program has the named file open.

Error Description: internal error

An error occurred in the compiler that could not be resolved to a specific error code. For resolution please contact FTDI technical support.

Error Description: local variable declarations exceed total memory size

The sum of local variable declarations in a function exceeds the maximum RAM size.

Error Description: Preprocessing failed

There was an error in the preprocessor which resulted in compilation being stopped. Refer to the pre-processor [Error reference](#) section.

Error Description: Assembling failed

There was an error in the preprocessor which resulted in compilation being stopped. Refer to the assembler [Error Reference](#) section.

Error Description: Linking failed

There was an error in the preprocessor which resulted in compilation being stopped. Refer to the linker [Error Reference](#) section.

3.2.5.2 Examples for Syntax Error Codes

Error Description: syntax error

The compiler could not recognise a statement or could not resolve a sequence of statements and identifiers.

Error Description: syntax error unterminated string

A string declaration was found which spans more than one line. Strings must include escaped carriage returns and new line characters.

Example

```
char cVar;  
cVar = "\n;  
cVar = "\n; // unterminated string
```

Error Description: syntax error illegal escape sequence

An escape sequence in a char or string was not recognised. Valid escape sequences are listed in [Type Specifiers](#).

Example

```
int cVar = '\!'; //illegal escape sequence
```

Error Description: syntax error character operation too long

More than one character was used for a char declaration. The size of a char is always 1 character so multiple characters cannot be used in a char declaration.

Example

```
char cVar;  
cVar = '\f1'; // character operation too long
```

Error Description: syntax error asm directive not allowed

Assembler directives cannot be used in inline assembler.

Example

```
asm {  
    .ORG 0  
    asmvar .DB 45 1  
}
```

Error Description: syntax error preprocessor directive not allowed

An unsupported preprocessor directive was encountered. The compiler will implement #line and #pragma directives only.

Example

```
#mydirective this is illegal
```

3.2.5.3 Examples for General Syntax Error Codes

Error Description: undeclared identifier

The compiler detected an attempt to use an identifier that has not been declared in the current scope.

Example

```
int x;  
x = y; //error undeclared identifier 'y'
```

Error Description: l-value required

An expression was detected that is missing the l-value (target identifier).

Example

```
int iVar;  
++iVar=2; //error lvalue required
```

Error Description: illegal use of struct or union operation

A structure or union was used incorrectly in an expression. Pointers to structs and unions must be references and dereferenced appropriately when used.

Example

```
struct MyStruct  
{  
    int iMem;  
};  
  
struct MyStruct *myPointer;  
struct MyStruct Obj;  
myPointer = Obj; //error Illegal use of structure or union operation
```

Error Description: incompatible operation

An operation where the l-value and r-value are incompatible was found.

Example

```
int a;  
int *x;  
struct MyStruct  
{  
    int m;  
}obj;  
  
x=&a;  
a = ((struct MyStruct)x)->m; //error incompatible operation
```

3.2.5.4 Examples for Conditional Statement Error Codes

Error Description: undefined label

An reference to an undefined label was made in a goto statement. Goto statements can only address labels in the current function body.

Example

```
goto L1; //error Undefined Label
```

Error Description: integral type expected

All values used with case statements must be constant integer values. The error is reported if value used in a case statement is a floating point value. Values with decimal points are defined as floating point.

Example

```
int iVar;
switch (iVar)
{
    case 1.0: // error Integral type expected
        break ;
}
```

Error Description: too many default cases

A switch statement may have only one default case. If there are more multiple default statements then this error will be reported.

Example

```
int iVar;
switch (iVar)//error Too many default cases
{
    default :
    default :
}
```

Error Description: constant expression required

When an enumeration is initialised the value used must be a constant.

Example

```
int iVar;
enum eTag
{
    a = iVar //constant expression required
}eObj;
```

Error Description: case outside of switch

A case statement was encountered which was not inside the body of a switch statement.

Example

```
int iVar;
if(iVar)
{
    case 1:// error case outside the switch
}
```

Error Description: default outside of switch

A default case statement was encountered which was not inside the body of a switch statement.

Example

```
int main()
{
    if(1)
    {
        return 1;
    }
}

int function()
{
    int iVar;
    switch (iVar)
```

```
{
    case 1:
        return 1;
        break;
}

void function1(void)
{
    default: //error default outside the switch
    if (1)
    {
        default: //error default outside the switch
        {
            default: //error default outside the switch
        }
    }
}
```

Error Description: duplicate case

More than one case statement in a switch statement evaluated to the same value. The value of every case statement in a switch must be unique.

Example

```
int iVar;
switch (iVar) //error Duplicate case
{
    case 1:
        break ;
    case 1:
        break ;
}
```

Error Description: misplaced continue

A continue statement could not resolve to a loop statement such as for, while or do.

Example

```
int iVar;
switch (iVar)
{
    case 1:
        break;
    case 2:
        break;

    case 3:
        continue;//error misplaced continue
}
```

Error Description: misplaced break

A break statement could not resolve to a statement such as switch, for, while or do.

Example

```
int iVar;
if (iVar)
{
    break; //error Misplaced break
}
```


Error Description: no body or expression for conditional

A do or switch expression was encountered with no body.

Example

```
do int x; while (1);
```

Error Description: duplicate label

A label name has already been used within the current scope. Label names in function must be unique.

Example

```
void function(void)
{
    cleanup:
    return;
    cleanup: //error duplicate label
    return;
}
```

3.2.5.5 Examples for Storage Classes Error Codes

Error Description: storage class extern not allowed here

An extern storage class was used where it is not allowed. For instance in the parameters of a function.

Example

```
function(extern int iVar) //error Storage class 'extern' is not allowed here
```

Error Description: storage class static not allowed here

A static storage class was used where it is not allowed.

Example

```
function(static int iVar) //error Storage class 'static' is not allowed here
```

Error Description: storage class auto not allowed here

The auto storage class was used where it is not allowed.

Example

```
auto int iVar; //error Storage class 'auto' is not allowed here
```

Error Description: storage class port not allowed here

A port declaration was detected in a function, a parameter or a structure or union member. Only global scope ports are allowed. It will also be issued when a pointer to a port is defined.

Example

```
port *p4@45; //error storage class 'port' not allowed here
function(port p3) //error storage class 'port' not allowed here

void main(void)
{
    port p2@23; //error storage class 'port' not allowed here
}
```

Error Description: struct or union not allowed here

A struct or union has been used as a return value or parameter to a function. Only pointers to structs or unions may be used as parameters and return values of functions.

Example

```
struct tag
{
    char j;
};
struct tag *my[2];
struct tag funct(int); //error struct or union not allowed here

funct(2);

struct tag funct(int k) // error struct or union not allowed here
{
    int c;
    return my[1];
}
```

Error Description: too many storage classes in declaration

More than one storage class have been used in a declaration. There may be only one used at a time, the default is auto.

Example

```
static extern int iVar; // error Too many storage classes in declaration
```

Error Description: function declaration not allowed here

A function may not be declared within a function, an expression or a code block.

Example

```
function(int fVar()) //error function declaration not allowed here
```

Error Description: export symbol not allowed here

The symbol export has been used inside a function or in a declaration which does not have global scope.

Example

```
function()
{
    export int iVar; //error Export not allowed here
}
```

3.2.5.6 Examples for Declaration Error Codes

Warning Description: no storage type

No storage type has been specified for a declaration.

Example

Error Description: not an allowed type

An operation on a variable failed because of it's type. It is not allowed to dereference non-pointer types nor use pointer types in some types of operation.

Example

```
int iVar1,iVar2;
iVar1 = *--iVar2; //error not an allowed type
```

Error Description: conflicting storage type qualifier

The signed and unsigned storage type qualifiers for a declaration conflict. A declaration must be either signed or unsigned.

Example

```
signed unsigned int iVar1; //error conflicting storage type qualifier
```

Error Description: too many storage type qualifiers

A declaration has more than one storage classes in the declaration. It must contain one of auto, extern, static or port.

Example

```
extern static int iVar1; //error too many storage classes in declaration
```

Error Description: too many types in declaration

A declaration has more than one type in it's declaration. Only one data type may be specified for each variable, if none is specified then int will be assumed.

Example

```
char int iVar1; //error too many types in declaration
```

Warning Description: type mismatch

A the type of the l-value and r-value do not match. This may be due to a sign mismatch or data size mismatch.

Example

```
char cVar;
int iVar;
cVar = iVar ; //Warning Type Mismatch
```

Error Description: parameter number type mismatch

A mismatch occurred between the type of a parameter in a prototype and the type used in the function declaration. The mismatch is for data size and data type differences.

Example

```
void function(int iVar );
int iLVar;
function(iLVar);

void function(int *iVar) //Error Parameter Number 1 type mismatch
{
}
```

Warning Description: parameter mismatch in redeclaration

A mismatch occurred between the parameters in a function's declaration or prototype and the parameters passed to a function in a call. May also occur when a mismatch in the count of the parameters occurs in a redeclaration.

Example

```
void function(int iVar);  
void function(int iVar, int qVar) //error parameter mismatch in redeclaration  
{  
}  
  
function(var1, var2, var3); //error parameter mismatch in redeclaration
```

Warning Description: parameter number type mismatch

A mismatch occurred between the type of a parameter in a prototype and the type used in the function declaration. The mismatch is for differences in sign.

Example

```
void function(int iVar );  
unsigned int iLVar;  
function(iLVar);  
  
void function(unsigned int iVar) //Error Parameter Number 1 type mismatch  
{  
}
```

Error Description: multiple declaration

The same variable name has been used in the same scope more than once.

Example

```
int iVar;  
char iVar; // error multiple declaration
```

Error Description: cannot use a void type expression

If a void type is used then this cannot be used in an expression. For example, a function may return void, in which case it may not be used in an expression.

Example

```
void function()  
{  
}  
int iVar;  
  
if(iVar+function()) //error cannot use a void type in an expression  
{  
    iVar++;  
}
```

3.2.5.7 Examples for Constant Range Error Codes

Error Description: integer arithmetic overflow

A constant exceeds the maximum size of an integer (2^{32}).

Example

```
char cVar = 0x100000000; //error integer arithmetic overflow
```

Error Description: expression not constant

A value used in a case statement, port address assignment, global variable declaration or array initialiser list is not a constant.

Example

```
char cGlobal1 = 87;
char cGlobal2 = cGlobal1 + 10; //error expression not constant
```

Example

```
int cVar = 1;
switch (x) {
case cVar: break; //error expression not constant
}
```

Warning Description: constant out of range

A value used in an assignment exceeds the range which can be stored in the variable.

Example

```
char cVar = 87654; //warning constant out of range
```

Error Description: constant out of range

A value used in an operation exceeds the limits for the destination variable.

Example

```
unsigned char cVar;

typedef struct myStruct
{
    int a;
    int b;
} X;

X x;

unsigned char *pa;
unsigned char *pv;

cVar = 0xfe;
x.a = 0x55555555;
x.b = 0xaaaaaaaa;
pv = (unsigned char*)&x;
pv += 4;
if (*pv == 0xaaaaaaaa) //error Constant out of range
{}
```

Error Description: divide by zero

An operation has a divide operation where the denominator is a constant zero.

Example

```
int iVar;
iVar = iVar / 0; //error divide by zero
```

Warning Description: initialization makes pointer from integer without cast

A pointer is initialised with a constant or variable that is not a pointer type.

Example

```
char *cVar = 0x12345678; //warning initialization makes pointer from integer without a cast
```

3.2.5.8 Examples for Constant Error Codes

Error Description: cannot modify const object

An attempt has been made to modify the value of a variable which has been declared as a constant.

Example

```
const unsigned int uiVar = 2;
uiVar++; //cannot modify a const object
```

Error Description: cannot modify rom variable

Code which attempts to modify a variable held in ROM has been found.

Example

```
rom int iVar;
iVar = 2; //error cannot modify rom variable
```

3.2.5.9 Examples for Variable Error Codes

Error Description: rom variable for structure member should be minimum short datatype

If a structure or union is defined as a ROM storage type then the minimum size of any member datatype is a word. Datatype short, int, long and all pointers are allowed but char is not.

Example

```
rom struct stx {
    int x;
    short y;
    char z; // error rom variable for structure member should be minimum short datatype
};
```

3.2.5.10 Examples for Array Error Codes

Error Description: size of the type is unknown or zero

An operation was attempted on a variable for which it cannot determine the correct size. Or the variable size was determined to be zero. This may also be generated when an array size cannot be determined from the initialiser.

Example

```
void *vPtr1;
vPtr1-- ; //error size of the type is unknown or zero
char x[]; //error size of the type is unknown or zero
```

Error Description: excess elements in array initialiser

When an array was initialised there were more initialisers than the declared size of the array.

Example

```
int array[3] = {1,2,3,4}; //excess elements in array initialiser
```

Error Description: array must have at least one element

An array must have one or more elements. A zero length array cannot be declared.

Example

```
int iArr[0]; //error Array must have at least one element
```

3.2.5.11 Examples for Structure Union Error Codes

Error Description: pointer to structure required on left side of "->"

The variable on the left side of a structure pointer operator is not a pointer to a structure or union.

Example

```
int iVar;
struct myStruct
{
    int iMem;
};

struct myStruct StObj;
iVar=StObj->iMem; //error pointer to structure required on left side of "->"
```

Error Description: structure required on left side of "."

The variable on the left side of a structure member operator is not a structure or union.

Example

```
int iVar;
struct myStruct
{
    int iMem;
};

struct myStruct *StObj;
iVar=StObj.iMem; //error structure required on left side of "."
```

Error Description: structure or union cannot contain an instance of itself

Only pointers to self-referential instances of structures are allowed.

Example

```
struct myStruct
{
    struct myStruct
    {
        int iMem;
    }myInnerStruct; //error structure or union cannot contain an instance of itself
};

struct myStruct2
{
    struct myStruct2 st2; //error structure or union cannot contain an instance of itself
    struct *myStruct2 pst2; // pointer to instance of self OK
};
```

Error Description: left of pointer or member not a struct or union

Only pointers to self-referential instances of structures are allowed.

Example

```
int x;

x.member = 4; //error left of pointer or member not a struct or union
x->pointer = 5; //error left of pointer or member not a struct or union
```

3.2.5.12 Examples for Initialisation Error Codes

Error Description: address of port is not specified

A port datatype must be initialised with the address of the port.

Example

```
port pt1 ; //address of port is not specified
```

Error Description: initialisation of port is not allowed

A port datatype must only be initialised with the address of the port. There can be no numerical value assigned to the port at initialisation time.

Example

```
port pt1 =1 ; //initialisation of port is not allowed
```

Error Description: initialisation of extern is not allowed

VinC issues following error message for the below given example

Example

```
extern int iVar =1 ; //initialisation of externals is not allowed
```

3.2.5.13 Examples for Function Error Codes

Error Description: too few parameters in call

A call to a function or a declaration of a function or it's prototype contains fewer parameters than expected.

Example

```
void x(int, int, int);

void x(int a, int b) //error too few parameters in call & parameter mismatch in redeclaration
{
}
void main(void)
{
    x(1); //error too few parameters in call
}
```

Warning Description: void function may not return value

A value was returned from a function which was not declared to return a value.

Example

```
void function() //warning void function may not return value
{
    return 1;
}
```

Warning Description: function should return a value

No suitable return statement was found in a function which was declared to return a value.

Example

```
int function() //warning function should return a value
{
}
```

```
}
```

Error Description: parameter of type void is not allowed

A parameter to a function may not be of void type.

Example

```
int function(void x) //error parameter of type void is not allowed
{
}
```

3.2.5.14 Examples for Pointer Error Codes

Error Description: invalid pointer addition

An addition of 2 pointers was detected. The increment used in a pointer addition is a multiple of the pointer size, therefore adding 2 pointers is not a valid operation.

Example

```
int *iPtr1,*iPtr2,*iPtr3;
int iVar;
if (iVar)
{
    iPtr1 = iPtr2 + iPtr3 ;//error invalid pointer Addition
}
```

Warning Description: copy volatile pointer into normal pointer

A volatile pointer was copied into a normal pointer. The normal pointer does not inherit the volatile properties of the original pointer.

Example

Warning Description: copy const pointer into normal pointer

A const pointer was copied into a normal pointer. The normal pointer does not inherit the const properties of the original pointer.

Example

Error Description: illegal port access

Not all operations are allowed on ports. Unary operations, array operations, reference, de-reference, and port-to-port assignments are not allowed. It is not possible to create a pointer to a port or pass or return a port from a function.

Example

```
port p1@40, p2@50;

void main(void)
{
    char *p3;
    p2 = p1; //error illegal port access
    p2++; //error illegal port access
    p3 = &p1; //error illegal port access
}
```

Error Description: not a member of struct or union

The name to the right of a structure member or structure pointer operation is not a member of that

structure.

Example

```
struct q {  
    int a;  
    int b;  
} stq;  
  
if (stq.c > 1) //error not a member of struct or union
```

Error Description: illegal use of pointer subtraction

A error is issued when pointer subtraction is encountered when constant and pointer operand.

Example

```
int Result;  
int *x1;  
Result = 2 - x1; //error Illegal use of pointer subtraction
```

Warning Description: pointer subtraction

A warning is issued when pointer subtraction is encountered when both operands are not both pointers.

Example

```
unsigned char ucArray[10];  
unsigned char *ucPtr;  
ucPtr = ucArray;  
ucPtr = ucArray - 3; // warning pointer subtraction
```

Error Description: illegal function pointer declaration

A function pointer was declared without the name of the function being made a pointer.

Example

```
int (PF)(int x, int y); //error illegal function pointer declaration
```

Error Description: illegal use of pointer

Pointers may only be added or subtracted. All other operations are illegal.

Example

```
int *iPtr1,*iPtr2,*iPtr3;  
int iVar;  
if(iVar)  
{  
    iPtr1 = iPtr2 * iPtr3 ;//error illegal use of pointer  
}
```

Error Description: illegal use of array

Arrays may only be addressed by an offset on the left size of an expression. On the right side they may be treated as a pointer. It is not permitted to modify the actual address of an array.

Example

```
unsigned char ucArray[10];  
++ucArray; // error illegal use of array
```

Warning Description: non portable pointer conversion

A conversion between a non-pointer and a pointer will result in a non-portable operation.

Example

```
struct myStruct
{
    char mem;
};
int iVar;
struct myStruct *myPointer;
struct myStruct StArray[2];
myPointer = iVar;//warning Non portable pointer conversion
```

Warning Description: suspicious pointer conversion

A referencing or de-referencing operation on a pointer resulted in an inconclusive pointer type.

Example

```
char ***cPtr1;
char *cPtr2;
char *Ptr;

Ptr = *cPtr1; //warning supspicious pointer conversion
Ptr = *cPtr2; //warning supspicious pointer conversion
```

Error Description: invalid indirection

VinC issues following error message for the below given example

Example

```
int iArray;
int iVar;
iVar = 2 - iArray[2]; //error Invalid indirection
```

3.2.5.15 Examples for Bitfield Error Codes

Error Description: bit fields must contain at least one bit

Zero length bit fields are not permitted.

Example

```
struct myStruct
{
    unsigned char ucMem:0;//error Bit fields must contain at least one bit
} ;
```

Error Description: bit field exceeds the range

The size of a bit field exceeds the maximum range of the specified storage type.

Example

```
struct myStruct
{
    unsigned int uiMem:1;
    unsigned char ucMem1:9;//error Bit Field Exceeds Range
    unsigned char ucMem2:5;
    unsigned char ucMem3:5;
}myStObj;
```

Warning Description: bit field exceeds boundary

The size of a bit field causes it to run over the maximum size of the storage type.

Example

```
struct myStruct
{
    unsigned short uiMem:12;
    unsigned short ucMem:6; //warning Bit Field Exceeds Boundary
}myStObj;
```

Warning Description: illegal use of bit operation access

A bit operation exceeded the range of the variable storage type.

Example

```
char x = 0xAA;

if (x.3 == 1) {}
if (x.9 == 0) {} //error illegal use of bit operation access
```

3.2.6 Pre-processor

3.2.6.1 Pre-processor Directives

The preprocessor parses special directives used in the C files. A directive starts with a '#' token followed immediately by a preprocessor keyword. By default it ends with a new-line character but it can also end with a backslash '\' character which is continuation marker. Leading and trailing spaces in the backslash character is allowed.

#if

The #if directive is used to conditionally include or exclude a portion of code depending on the outcome of a certain expression.

Format:

```
#if <expression>
    statements.....
#endif
```

#else

The #else directive conditionally include a portion of code (codes between #else directive and #endif directive) if the condition in #if, #ifdef or #ifndef directive results to 0.

Format:

```
#if <expression>
    Statements...
#else
    Statements...
#endif
```

#endif

The #endif directive indicates the end of a conditional directive - #if, #ifndef and #ifdef. Please refer to #if, #ifndef or #ifdef for the syntax.

#error

The #error directive is used in generating an error message and results to stop compilation process.

This directive is passed to the compiler unaltered.

Format:

```
#error "error message"
```

#pragma

The #pragma directive is used to specify information to the compiler. It provides a way for the compiler to offer machine /operating system-specific features while retaining the compatibility with the C language. This directive is passed to the compiler unaltered.

Format:

```
#pragma <argument/s>
```

#define

The #define directives is used for defining a symbol or macro.

For defining a symbol:

```
#define <symbol>
```

For defining a macro:

```
#define <symbol> <value>  
#define <symbol>(A, ...) <value>
```

#undef

The #undef directive is used to undefine a symbol but it can be redefined later

Format:

```
#undef <symbol>
```

#ifdef

The #ifdef directive is used to conditionally include a portion of the code if the symbol has been defined using a #define directive.

Format:

```
#ifdef <name>  
    Statements...  
#endif
```

#ifndef

This directive is used to conditionally include a portion of the code if the symbol has not been defined using a #define directive. It is useful for once-only headers. It allows the header files to be included once.

Format:

```
#ifndef <symbol>  
    Statements...  
#endif
```

#elif

The #elif directive is used if there are more than two possible alternatives

Format:

```
#if <expression>  
    Statement...  
#elif < expression >
```

```
Statement...  
#endif
```

#include

The #include directive is used in including the contents of the included file in the current source file.

Format:

```
#include "header file"  
#include <header file>
```

#line

The #line directive is used to set the line number and filename of the current file (but can be omitted). The line number in #line directive shall be assigned to the next line of code following the said directive

Format:

```
#line <line number>  
#line <line number> "filename"
```

3.2.6.1.1 Predefined Macros

The Preprocessor defines standard and VNC2 specific macros.

Standard Macros

__DATE__

Expands to the date in the current time zone.

The output is of the format: "Aug 13 2009".

__TIME__

Expands to the time in the 24 hour clock in the current time zone.

The output is of the format: "23:12:02".

__LINE__

This macro expands to the current line of the file where it appears. It is an integer number.

__FILE__

Expands to the name and relative path of the file in which it appears.

The path is the actual path used to compile the file relative to the current directory of the compiler. There are no quotes placed around the filename.

VinC Specific Macros

_VINC

This is always defined by the VinC compiler. It can be used to separate code targeted specifically for a Vinculum device from code for other devices in the same file.

_VINCULUM

This macro expands to the model of Vinculum device. Currently it is set to 2 signifying that the compiler supports VNC2.

_VINCULUM_VERSION

The version of the VinC compiler is available in this macro. This can be used to work out any differences between versions of the compiler.

3.2.6.2 Error reference

Preprocessor error messages take the following form:

```
<filename> line <line number>: (error|warning|info) P<code> <description>
```

Error codes take one of the following values.

Error codes	Description
-------------	-------------

1001	missing (in expression
1002	missing) in expression
1005	numeric too large
1006	unterminated #if
1007	unterminated #ifdef
1008	unterminated #ifndef
1010	#elif without #if
1011	#else without #if
1012	#else after #else
1013	#endif without #if
1014	constant too large
1016	redefined
1017	location of the previous definition
2000	invalid identifier
2001	bad directive syntax
2002	missing closing ')'
2003	invalid include filename
2004	macro argument syntax error
2005	missing expression
2006	expression syntax error
3001	too many input files
3002	no input file specified
3003	no such file or directory
3006	#include nested too deep
3008	macro names must be identifiers
4001	argument mismatch
4003	unterminated argument list

3.2.6.2.1 Examples for Directive Error Codes

Error Description: missing (in expression

Open parenthesis is missing in the #if directive expression.

Example

```
#if x==1)
```

Error Description: missing) in expression

Close parenthesis is missing in the #if directive expression.

Example

```
#if (x==1
```

Error Description: numeric too large

Constant value given in the #if directive expression exceeds 0xFFFFFFFF.

Example


```
#include "file.h"
```

Error Description: macro argument syntax error

#define directive expects valid argument

Example

```
#define FUNC((x) #x
```

Error Description: missing expression

#if directive expects constant expression instead of NULL.

Example

```
#if
```

Error Description: expression syntax error

#define directive expects token string for the identifier

Example

```
#define test
#if test++1 //error
#define x 10
#endif
```

3.2.6.2.3 Examples for General Error Codes

Error Description: too many input files

Too many input files specified after application name.

Example

```
Vincpp file1.c file2.c
```

Error Description: no input file specified

No input file is specified after the application name

Example

```
VinCpp
```

Error Description: no such file or directory

File does not exist

Example

```
#include "file.h"
```

Warning Description: #include nested too deep

Recursive inclusion of file

Example

```
//file1.h
#include "file2.h"
//file2.h
#include "file1.h"
```

Warning Description: macro names must be identifiers

Macro names in -U option does not start with an alphabet

Example

```
VinCpp file.c -U 9MACRO
```

Error Description: argument mismatch

Number of arguments in a macro call is not correct

Example

```
#define func(x,y) x##y
void main()
{
    int x = func(10); //lacking argument
}
```

Error Description: unterminated argument list

Unterminated string value in a function-like macro

Example

```
#define func(x,y,) x##y
void main()
{
    int x = func(10,"test); //unterminated string
}
```

3.3 VinAsm Assembler

The assembler is a tool that generates hardware-specific machine codes for VNC2. It processes both directives and instructions.

The input of the assembler is an asm file which may contain comments, directives, instructions and white spaces.

Normally, the assembler generates an object file which is in ELF format. If debug flag is enabled, the assembler generates debug information which is in DWARF2 format. For cases where problems occurred, error messages will be displayed and no object file will be created.

3.3.1 Assembler Command Line Options

The VNC2 Assembler Command Line options are listed in the table below.

```
VinASM [options] [file ...]
```

Option	Description
-v	Verbose output of the command lines.
-d level	Includes debugging information in the object file.

-o filename	Specify output filename
-I directory	Adds a search directory for include files
-l filename	Specify a log filename
--help	Display help message
--version	Display version number
-c	Case-sensitive checking for labels/symbols
-u	Ignores underscores in symbols/labels

3.3.2 Assembly Language

The programming syntax used by the VNC2 Assembler is similar to the other assemblers. There may be variation on some aspects yet in general, still the same with other existing assemblers.

3.3.2.1 Lexical Conventions

There are conventions that needs to be followed when creating a source program using the VNC2 Assembler. The convention for the following must be noted.

3.3.2.1.1 Comments

The assembler supports single-line comments. This can be done using the '#' character. Thus, any text following the '#' character is ignored.

3.3.2.1.2 Functions

There is a mechanism to distinguish a label from a function. If a certain label needs to be considered as a function, then .FUNCTION directive must be present right after the label declaration. Then .FUNC_END must be present at the end of the function scope.

Related Links

[.FUNCTION](#)

[.FUNC_END](#)

3.3.2.1.3 Identifiers

Identifiers may be used as label, function name, etc. It consists of alphanumeric and selected special characters namely:

- % (percent)
- _ (underscore)
- @ (at sign)
- () (open and close parenthesis)
- * (asterisk)

3.3.2.1.4 Keywords

Keywords are tokens which have special meaning to the assembler. It can either be assembler mnemonics (assembler instructions) or directives.

Upon invoking the keywords, correct syntax must be provided and take note that it must be in uppercase.

3.3.2.1.5 Labels

A label consists of an identifier and followed by a colon ':'. By default, a label name is case-insensitive unless -c option is used.

A warning will be issued if similar label is defined.

3.3.2.1.6 Numeric Value

Numeric value can be of two forms - decimal and hexadecimal. There's no restriction on where to use each type. Using either of the forms will do.

3.3.2.1.7 White Space Characters

Any number of white spaces is allowed between tokens. For directives and instructions, it must be contain in a single line.

3.3.3 Assembler Directives

There are several kinds of directives that are supported by the VNC2 Assembler. The following are the classifications:

- Data Directives
- Debugger Directives
- End Directive
- File Inclusion Directive
- Location Control Directives
- Symbol Declaration Directives

3.3.3.1 Data Directives

Data directives are used in allocating memory. The allocated memory may be initialized with some values.

3.3.3.1.1 .ASCII Directive

Syntax

```
label .ASCII string
```

Parameters

`label`

The name of the identifier to which the character string will be assigned to.

`string`

The string literal to be assigned to label. It must be enclosed with double quotes.

Description

The .ASCII directive allows the assembler to accept string literal value for a certain label. The assembler generates the ASCII codes of all the characters in the string and store in consecutive bytes in memory. The assembler does not add NULL terminator to the string literal value.

Example

```
str .ASCII "The quick brown fox jumps over the lazy dog."
```

Related Links

[.ASCIIZ](#)

3.3.3.1.2 .ASCIIZ Directive

Syntax

```
label .ASCIIZ string
```

Parameters

`label`

The name of the identifier to which the character string will be assigned to.

`string`

The string literal to be assigned to label. It must be enclosed with double quotes.

Description

The `.ASCIIZ` directive is similar to `.ASCII` except that the assembler automatically adds a NULL terminator to the string literal value.

Example

```
str .ASCIIZ "The quick brown fox jumps over the lazy dog."
```

Related Links

[.ASCII](#)

3.3.3.1.3 .CONST Directive

Syntax

```
label .CONST value
```

Parameters

`label`

The name of the identifier to which the value will be assigned to.

`value`

The integer value to be assigned to label. It may be decimal or hexadecimal value.

Description

The `.CONST` directive creates symbols for use in our assembler expressions. Constants may not be reset after having once being initialized, and the expression must be fully resolvable at the time of the assignment. This is the principal difference between symbols declared as `.CONST` and those declared as `.DB/.DW/.DD`.

Example

```
_value .CONST 0xff  
var .CONST 100
```

Related Links

[.DB](#)

[.DD](#)

[.DW](#)

3.3.3.1.4 .DATA_DEF Directive

Syntax

```
label .DATA_DEF size
```

Parameters

`label`

The name of the identifier.

`size`

The size (in bytes) of the symbol to be declared.

Description

The `.DATA_DEF` directive is used to allocate memory for a structure variable. The structure variable must be initialized with values.

To have a complete declaration of structure variable, the following is the syntax.

```
argument1 .DATA_DEF argument2
argument1 .DATA_INIT argument2...argument n
:
:
argument1 .DATA_END
```

Notes

1. The directives must be in correct order.
2. Instances of `.DATA_INIT` directive will depend on the number of structure fields need to be initialized.
3. There must be one instance of `.DATA_DEF` and `.DATA_DEF_END` in every structure declaration.

Example

See `.DATA_INIT` example.

Related Links

[.DATA_DEF_END](#)

[.DATA_INIT](#)

3.3.3.1.5 .DATA_DEF_END Directive

Syntax

```
label .DATA_DEF_END
```

Parameters

label

The name of the identifier. This should correspond to the label in `.DATA_DEF` directive.

Description

The `.DATA_DEF_END` directive is used to indicate the end of a structure variable declaration. Prior to this directive, a `.DATA_DEF` directive must be present.

Example

See `.DATA_INIT` example.

Related Links

[.DATA_DEF](#)

[.DATA_INIT](#)

3.3.3.1.6 .DATA_INIT Directive

Syntax

```
label .DATA_INIT value offset size_of_each_value total_size
```

Parameters

label

The name of the identifier. This should correspond to the label in `.DATA_DEF` and `.DATA_DEF_END` directives.

value

The value of each field in the structure. For arrays, initialization can be done by separating values with commas e.g. 10, 20, 30.

The value can be any of the following form:

Form	Example
Character	'a'
Numeric	10, 0xff
String Literal	"this is a string"

offset

The offset (in bits) of the field member in the structure. For the first field, it should start with offset 0. Subsequent field offset should be relative to the preceding structure field.

size_of_each_value

The size (in bits) of each member field e.g. 32 bits for field with integer as the datatype.

total_size

The total size of the member field e.g. 96 bits for field which is an integer array having 3 elements.

Description

The `.DATA_INIT` directive is used to initialize the member fields of the structure. Number of instances of this directive will depend on the number of structure fields need to be initialized. This directive must be within the `.DATA_DEF` and `DATA_DEF_END` directives.

Example

C structure:

```
struct st
{
    char name[10];
    int x;
    char c;
    int y[2];
};
```

Initial values of the struct variable:

```
struct st var = { "testing",
                  100,
                  'c',
                  10, 20
                };
```

Equivalent directives:

```
var .DATA_DEF 23
var .DATA_INIT "testing" 0      8      80
var .DATA_INIT 100      80      32      32
var .DATA_INIT 'c'      112      8      8
var .DATA_INIT 10, 20    120      32      64
var .DATA_DEF_END
```

The total size of struct st is 23 bytes.

The 'name' field starts at offset 0 since it is the first member field. The size of char is 8 bits (1 byte) , since it is an array of 10, then the total size is 80 bits.

The 'x' field starts at offset 80 since the prior field occupies from 0th-79th bit. The size of int is 32 bits (4 bytes), then the total size is still 32 bits (4 bytes).

The 'c' field starts at offset 112 (offset of 'x' plus total size of 'x'). The `size_of_each_value` and `total_size` are equal to 8 bits (1 byte).

The 'y' field starts at offset 120 (offset of 'c' plus total size of 'c'). The `size_of_each_value` is 32 bits

(4 bytes), then the total_size is 64 (8 bytes).

NOTE: The units used is in bits so that it will still cater for bitfields.

Related Links

[.DATA_DEF](#)

[.DATA_DEF_END](#)

3.3.3.1.7 .DB Directive

Syntax

```
label .DB num_of_bytes value/s
```

Parameters

label
The name of the identifier.

num_of_bytes
The number of bytes to be allocated.

value/s
The initial values to be assigned to label.

Description

The .DB directive declares a number of bytes of memory in either DATA, TEXT, or .BSS segment. If the size of the values is less than num_of_bytes, values will be padded with NULL value/s. If values is a string, initialization can be done by enclosing the string with double quotes. Other possible way is that it can be taken one character at a time and it will be comma-separated.

If values is a '?', it means that the data will be part of the BSS segment rather than the DATA or TEXT segment.

Example

```
_var1 .DB 10 0xff
var2  .DB 5 "TEST"
var_3 .DB 5 'T', 'E', 'S', 'T'
data  .DB 20 ?
```

Related Links

[.DW](#)

[.DD](#)

3.3.3.1.8 .DD Directive

Syntax

```
label .DD num_of_double_words value/s
```

Parameters

label
The name of the identifier.

num_of_double_words
The number of double words to be allocated.

value/s
The initial values to be assigned to label.

Description

The .DD directive declares a number of double words of memory in either .data, TEXT, or BSS

segment. If the size of the values is less than `num_of_double_words` , values will be padded with NULL value/s . If values is a string, initialization can be done by enclosing the string with double quotes. Other possible way is that it can be taken one character at a time and it will be comma-separated.

If values is a '?', it means that the data will be part of the BSS segment rather than the DATA or TEXT segment.

Example

```
_var1 .DD 10 0xff
var2   .DD 5 "TEST"
var_3  .DD 5 'T', 'E', 'S', 'T'
data   .DD 20 ?
```

Related Links

[.DW](#)

[.DD](#)

3.3.3.1.9 .DW Directive

Syntax

```
label .DW num_of_words value/s
```

Parameters

`label`
The name of the identifier.

`num_of_words`
The number of words to be allocated.

`value/s`
The initial values to be assigned to label.

Description

The `.DW` directive declares a number of words of memory in either `.data`, `TEXT`, or `BSS` segment. If the size of the values is less than `num_of_words` , values will be padded with NULL value/s . If values is a string, initialization can be done by enclosing the string with double quotes. Other possible way is that it can be taken one character at a time and it will be comma-separated.

If values is a '?', it means that the data will be part of the BSS segment rather than the DATA or TEXT segment.

Example

```
_var1 .DW 10 0xff
var2   .DW 5 "TEST"
var_3  .DW 5 'T', 'E', 'S', 'T'
data   .DW 20 ?
```

Related Links

[.DB](#)

[.DD](#)

3.3.3.2 Debugger Directives

Debugger directives are used by the compiler to pass debugging information to the Debugger.

The following are the debug information:

- Enum
- Structure
- Union

-
- File
 - Line Number
 - Function
 - Typedef
 - Variable

3.3.3.2.1 .ENUM Directive

Syntax

```
.ENUM name
```

Parameters

name

The name of the enum declaration. It should be enclosed with double quotes.

Description

The .ENUM directive is used to pass the name of the enum as part of the enum debug information.

For a complete enum declaration, the following must be the syntax:

```
.ENUM name_of_enum  
.ENUMERATOR name value  
:  
:  
.ENUM_END name_of_enum
```

Notes

1. Directives should be in proper order.
2. Instances of .ENUMERATOR directive will depend on the number of enumerators present.
3. The 3 directives should be present.

Example

Related Links

[.ENUMERATOR](#)

[.ENUM_END](#)

3.3.3.2.2 .ENUMERATOR Directive

Syntax

```
.ENUMERATOR name value
```

Parameters

name

The name of the enumerator. It should be enclosed with double quotes.

value

The value of the enumerator.

Description

The .ENUMERATOR directive is used to specify an enum value. Each .ENUMERATOR directive corresponds to one enumerator, thus at least one instance of this directive must be present in setting the enum debug information.

For a complete enum declaration, the following must be the syntax:

```
.ENUM name_of_enum
.ENUMERATOR name value
:
:
.ENUM_END name_of_enum
```

Example

See example in .ENUM.

Related Links

[.ENUM](#)

[.ENUM_END](#)

3.3.3.2.3 .ENUM_END Directive

Syntax

```
.ENUM_END name
```

Parameters

name

The name of the enum declaration which should correspond to the name in .ENUM directive. It should be enclosed with double quotes.

Description

The .ENUM_END directive is used to indicate the end of an enum declaration. Prior to this, a .ENUM directive must be present.

For a complete enum declaration, the following must be the syntax:

```
.ENUM name_of_enum
.ENUMERATOR name value
:
:
.ENUM_END name_of_enum
```

Example

See example in .ENUM.

Related Links

[.ENUM](#)

[.ENUMERATOR](#)

3.3.3.2.4 .FILE Directive

Syntax

```
.FILE filename
```

Parameters

filename

The filename of the C file. It should be enclosed with double quotes.

Description

The .FILE directive specifies the C filename.

Example

`.FILE "filename.c"`

3.3.3.2.5 .FUNCTION Directive

Syntax

`.FUNCTION function_name modifier1..modifier n`

Parameters

`function_name`

The name of the function. It should be enclosed with double quotes.

`modifier`

The modifier/s of the function e.g. static, volatile, etc.

Description

The `.FUNCTION` directive is used to pass a function information from the compiler to the debugger. It is also used as a mechanism to distinguish a label from a function.

To declare a function, `.FUNCTION` and `.FUNC_END` must be indicated.

To complete a function declaration, the following is the syntax:

```
.FUNCTION argument 1..argument n
.RETURN argument1..argument n
.PARAMETER argument 1..argument n
:
:
.FUNC_END argument
```

Notes

1. If the function has no parameters, then `.PARAMETER` can be omitted from the declaration.
2. `.RETURN` and `.PARAMETER` come in any order.
3. `.FUNCTION` must be the starting directive while `.FUNC_END` should be the ending directive.
4. Only `.PARAMETER` can have multiple instances.

Example

C function:

```
static int function( const int x, const int y )
{
    return( x + y );
}
```

Equivalent ASM Directives:

```
function:
.FUNCTION      "function"
.RETURN "int"   32      SIGNED      NORMAL      3      NORMAL      0
.PARAMETER "x"   32 "int" SIGNED      NORMAL      11      NORMAL      0
.PARAMETER "y"   32 "int" SIGNED      NORMAL      15      NORMAL      0

.... asm instructions here...

.FUNC_END      "function"
```

Related Links

[.PARAMETER](#)

[.RETURN](#)

[.FUNC_END](#)

3.3.3.2.6 .FUNC_END Directive

Syntax

```
.FUNC_END name
```

Parameters

name

The name of the function. This should correspond to the name in .FUNCTION directive. The name must be enclosed with double quotes.

Description

The .FUNC_END directive is used to indicate the end of a function. Prior to .FUNC_END, a .FUNCTION directive must be present.

Example

See example in .FUNCTION.

Related Links

[.FUNCTION](#)

[.PARAMETER](#)

[.RETURN](#)

3.3.3.2.7 .LINE Directive

Syntax

```
.LINE line_number
```

Parameters

line_number

The line number in the C source file. Line number must be greater than 0.

Description

The .LINE directive is used to pass the line number information for each C statement.

Example

```
.LINE 10
```

3.3.3.2.8 .PARAMETER Directive

Syntax

```
.PARAMETER name size type sign_flag pointer_flag  
offset array_flag array_dimension pointer_dimension  
line_number modifier 1..modifier n
```

Parameters

name

The name of the parameter.

size

The size (in bits) of the parameter.

type

The datatype of the parameter.

sign_flag

Indicates if the datatype is signed or unsigned. The following are the two possible values:

a) SIGNED - signed type

b) UNSIGNED - unsigned type

`pointer_flag`

Indicates if the variable is a pointer type. The following are the two possible values:

a) POINTER - pointer

b) NORMAL - not a pointer

`offset`

The stack offset of the parameter.

`array_flag`

Indicates if the variable is an array. The following are the possible values:

a) ARRAY - Array

b) NORMAL - Not an array

`array_dimension`

The dimension of the array. If `array_flag` is 1, `array_dimension` must be greater than 0.

`pointer_dimension`

The dimension of the pointer. If `pointer_flag` is 1, `pointer_dimension` must be greater than 0.

`line_number`

The line number in the C source file where the parameter is declared.

`modifier`

The modifiers of the parameter e.g. `const`, `volatile`, etc.

Description

Example

See example in `.FUNCTION`.

Related Links

[.FUNCTION](#)

[.RETURN](#)

[.FUNC_END](#)

3.3.3.2.9 .RETURN Directive

Syntax

```
.RETURN datatype size sign_flag pointer_flag  
      offset array_flag array_dimension pointer_dimension
```

Parameters

`datatype`

The return type of the function.

`size`

The size (in bits) of the return type.

`sign_flag`

Indicates if the datatype is signed or unsigned. The following are the two possible values:

a) SIGNED - signed type

b) UNSIGNED - unsigned type

`pointer_flag`

Indicates if the variable is a pointer type. The following are the two possible values:

a) POINTER - pointer

b) NORMAL - not a pointer

offset

The stack offset of the return value.

array_flag

Indicates if the variable is an array. The following are the possible values:

a) ARRAY - Array

b) NORMAL - Not an array

array_dimension

The dimension of the array. If array_flag is 1, array_dimension must be greater than 0.

pointer_dimension

The dimension of the pointer. If pointer_flag is 1, pointer_dimension must be greater than 0.

Description

The .RETURN directive is used to pass the function return type debug information. This directive should always be present once function debug information is set.

Example

See example in .FUNCTION.

Related Links

[.FUNCTION](#)

[.PARAMETER](#)

[.FUNC_END](#)

3.3.3.2.10 .STRUCT Directive

Syntax

```
.STRUCT name size
```

Parameters

name

The name of the structure. It should be enclosed with double quotes.

size

The size (in bits) of the structure.

Description

The .STRUCT directive is used to pass the name of the structure as part of the structure debug information. This should be the first directive to used once a structure debug information needs to be passed.

To pass the complete structure information, the following is the syntax:

```
.STRUCT argument
.STRUCTMEM argument1..argument n
:
:
.STRUCT_END argument
```

Example

Related Links

[.STRUCTMEM](#)

[.STRUCT_END](#)

3.3.3.2.11 .STRUCTMEM Directive

Syntax

```
.STRUCTMEM name datatype size sign_flag pointer_flag offset  
array_flag array_dimension pointer_dimension
```

Parameters

name

The name of the structure field. It must be enclosed with double quotes.

datatype

The datatype of the structure field.

size

The size (in bits) of the structure field.

sign_flag

Indicates if the datatype is signed or unsigned. The following are the two possible values:

- a) SIGNED - signed type
- b) UNSIGNED - unsigned type

pointer_flag

Indicates if the structure field is a pointer type. The following are the two possible values:

- a) POINTER - pointer
- b) NORMAL - not a pointer

offset

The offset (in bits) of structure field. Offset should start with 0. - Not used

array_flag

Indicates if the structure field is an array. The following are the possible values:

- a) ARRAY - Array
- b) NORMAL - Not an array

array_dimension

The dimension of the array. If array_flag is 1, array_dimension must be greater than 0.

pointer_dimension

The dimension of the pointer. If pointer_flag is 1, pointer_dimension must be greater than 0.

Description

The .STRUCTMEM directive is used to pass the structure field debug information. The number of instances of this directive depends on how many fields are present within a structure. This directive should be within .STRUCT and .STRUCT_END.

Example

See example in .STRUCT

Related Links

[.STRUCT](#)

[.STRUCT_END](#)

3.3.3.2.12 .STRUCT_END Directive

Syntax

```
.STRUCT_END name
```

Parameters

name

The name of the structure. This should match with the name in .STRUCT directive.

Description

The .STRUCT_END directive is used to indicate the end of a structure declaration. Prior to .STRUCT_END, a .STRUCT directive should be present.

Example

See example in .STRUCT.

Related Links

[.STRUCT](#)

[.STRUCTMEM](#)

3.3.3.2.13 .TYPEDEF Directive

Syntax

```
.TYPEDEF name type_defined_name
```

Parameters

name

The name of the datatype. It must be enclosed with double quotes.

type_defined_name

The new name of the datatype. It must be enclosed with double quotes.

Description

The .TYPEDEF directive is used to pass a typedef debug information.

Example

```
.TYPEDEF "unsigned int" "uint_32"
```

3.3.3.2.14 .UNION Directive

Syntax

```
.UNION name size
```

Parameters

name

The name of the union. It must be enclosed with double quotes.

size

The size (in bits) of the union.

Description

The .UNION directive is used to pass the name of the union as part of the union debug information. This should be the first directive to used once a union debug information needs to be passed.

Example

Related Links

[.UNIONMEM](#)

[.UNION_END](#)

3.3.3.2.15 .UNIONMEM Directive

Syntax

```
.UNIONMEM name datatype size sign_flag pointer_flag offset  
array_flag array_dimension pointer_dimension
```

Parameters

name

The name of the union field. It must be enclosed with double quotes.

datatype

The datatype of the union field. It must be enclosed with double quotes.

size

The size (in bits) of the union field.

sign_flag

Indicates if the datatype is signed or unsigned. The following are the two possible values:

- a) SIGNED - signed type
- b) UNSIGNED - unsigned type

pointer_flag

Indicates if the structure field is a pointer type. The following are the two possible values:

- a) POINTER - pointer
- b) NORMAL - not a pointer

offset

The offset of the union field. This is always set to 0.

array_flag

Indicates if the structure field is an array. The following are the possible values:

- a) ARRAY - Array
- b) NORMAL - Not an array

array_dimension

The dimension of the array. If array_flag is 1, array_dimension must be greater than 0.

pointer_dimension

The dimension of the pointer. If pointer_flag is 1, pointer_dimension must be greater than 0.

Description

The .UNIONMEM directive is used to pass the union field debug information. The number of instances of this directive depends on how many fields are present within a union. This directive should be within .UNION and .UNION_END.

Example

See example in .UNION.

Related Links

[.UNION](#)

[.UNION_END](#)

3.3.3.2.16 .UNION_END Directive

Syntax

```
.UNION_END name
```

Parameters

name

The name of the structure. This should match with the name in .STRUCT directive.

Description

The .UNION_END directive is used to indicate the end of a structure declaration. Prior to .UNION_END, a .UNION directive should be present.

Example

See example in .UNION.

Related Links

[.UNION](#)

[.UNIONMEM](#)

3.3.3.2.17 .VARIABLE Directive

Syntax

```
.VARIABLE "name" size "datatype" sign_flag pointer_flag offset  
array_flag array_dimension pointer_dimension line_number  
modifier1...modifier n
```

Parameters

name

The name of the variable.

size

The size of the variable in bits.

datatype

The datatype of the variable.

sign_flag

Indicates if the datatype is signed or unsigned. The following are the two possible values:

- a) SIGNED - signed type
- b) UNSIGNED - unsigned type

pointer_flag

Indicates if the variable is a pointer type. The following are the two possible values:

- a) POINTER - pointer
- b) NORMAL - not a pointer

offset

The offset address of the variable in the memory. The following are the possible values:

- a) ≥ 0 - Local variables (stack offset)
- b) -1 - Global variables
- c) -2 - Weak variables

array_flag

Indicates if the variable is an array. The following are the possible values:

- a) ARRAY - Array
- b) NORMAL - Not an array

array_dimension

The dimension of the array. If array_flag is 1, array_dimension must be greater than 0.

pointer_dimension

The dimension of the pointer. If pointer_flag is 1, pointer_dimension must be greater than 0.

line_number

The line number where the variable is defined in the C source file. This must be greater than

0.

modifier

The list of modifiers e.g. static, volatile, etc.

Description

The .VARIABLE directive is used to specify a variable declaration.

Example

C variable declaration:

```
volatile int *w[10];
```

Equivalent ASM Directive:

```
.VARIABLE      "w"      160      "int"      SIGNED      POINTER      -1      ARRAY
```

3.3.3.3 End Directive

End directive is used in terminating an asm source program.

3.3.3.3.1 .ENDP Directive

Syntax

```
.ENDP
```

Parameter

None

Description

The .ENDP directive indicates the end of the program. Any instructions after .ENDP shall be discarded. This is only applicable for TEXT segment.

3.3.3.4 File Inclusion Directive

File inclusion directive is used to add the contents of an include file into the current file.

3.3.3.4.1 .INCLUDE Directive

Syntax

```
.INCLUDE filename
```

Parameters

filename

The name of the include file. The file extension must be .asm. It must be enclosed with double quotes. Relative or absolute path can be appended into the filename. In cases wherein path can be eliminated in filename, the path can be set using the -I command-line option or it may be set using the VINASM_INCLUDE environment variable.

Description

The .INCLUDE directive is used to tell the assembler to treat the contents of the include file as if those contents are part of the current asm file.

Examples

```
.INCLUDE "include.asm"
.INCLUDE "path/include.asm"
```

3.3.3.5 Location Control Directives

Location control directives are used to control the location counter or current section.

3.3.3.5.1 .ABS Directive

Syntax

`.ABS`

Parameter

None

Description

The `.ABS` directive is used to tell the linker that the code generated is absolute and not relocatable. That is, it allows the programmer to change the way the assembler generates object code.

The `.ABS` directive must be used for [.ORG directives](#) to be heeded by the assembler. If the `.ABS` directive is not used, `.ORG` directives will be ignored.

3.3.3.5.2 .BSS Directive

Syntax

`.BSS`
`.BSS symbol size`

Parameters

symbol
The name of the symbol to be placed in the BSS segment.

size
The size (in bytes) of the symbol.

Description

If `.BSS` has no argument, it implies that the assembler will change the current section to `.bss`.

If `.BSS` has arguments, it instructs the assembler to define a symbol in the BSS segment and increments the location counter by size. At this time, the current section is not change to `.bss`.

3.3.3.5.3 .DATA Directive

Syntax

`.DATA`

Parameter

None

Description

The `.DATA` directive is used to declare that the content which follows the asm file is part of the DATA segment.

3.3.3.5.4 .EVEN Directive

Syntax

`.EVEN`

Parameter

None

Description

The `.EVEN` directive directs the assembler to place the following content of the asm file in an even

address. That is, the location counter is adjusted to an even value if it is currently odd. For TEXT segment, address are in terms of word. On the other hand, address in DATA segment are in terms of byte.

3.3.3.5.5 .ODD Directive

Syntax

```
.ODD
```

Parameter

None

Description

The .ODD directive directs the assembler to place the following content of the asm file in an odd address. That is, the location counter is adjusted to an odd value if it is currently even. For TEXT segment, address are in terms of word. On the other hand, address in DATA segment are in terms of byte.

3.3.3.5.6 .ORG Directive

Syntax

```
.ORG address
```

Parameter

address

The origin address. The address may be decimal or hexadecimal value.

Description

The .ORG directive instructs the assembler to place the content that follows at the specified address. The operand must be a valid address.

Note that the origin address will only be considered once the assembler is in absolute mode, otherwise this will be discarded. By default, the assembler is not in absolute mode. The assembler enables absolute mode using the [.ABS directive](#).

For TEXT segment, address are in terms of word. On the other hand, address in DATA segment are in terms of byte.

Example

```
.ORG 0xff
.ORG 255
```

3.3.3.5.7 .TEXT Directive

Syntax

```
.TEXT
```

Parameter

None

Description

The .TEXT directive is used to declare that the content which follows in the asm file is part of code or TEXT segment of the program.

3.3.3.6 Symbol Declaration Directives

Symbol declaration directives are used to define symbolic constants. Also, these are used in setting the attribute of a certain symbol.

3.3.3.6.1 .EQU Directive

Syntax

```
label .EQU value
```

Parameters

label

The name of the identifier to which the value will be assigned to.

value

The constant value that will be assigned to label.

Description

The .EQU directive assigns a constant value to an identifier.

Examples

```
address .EQU 0xff  
_label .EQU 1000
```

3.3.3.6.2 .GLOBAL Directive

Syntax

```
.GLOBAL export_flag symbol1, ..., symboln
```

Parameters

export_flag

Indicates if the symbol needs to be exported or not. The following are the possible values:

- a) EXPORT - symbol needs to be exported
- b) DO_NOT_EXPORT - symbol does not need to be exported

symbol

The name of the symbol to be declared as global.

Description

The .GLOBAL directive lets a particular symbol to be global in scope.

Note: The symbol must be declared first before using it in .GLOBAL directive.

Examples

```
.GLOBAL main  
.GLOBAL func, func1
```

3.3.3.6.3 .LOCAL Directive

Syntax

```
.LOCAL symbol1, ..., symboln
```

Parameters

symbol

The name of the symbol to be declared as local.

Description

The .LOCAL directive lets a particular symbol to be local in scope, thus the symbol is just visible within the file.

Note: The symbol must be declared first before using it in .LOCAL directive.

Examples

```
.LOCAL main  
.LOCAL func, func1
```

3.3.3.6.4 .WEAK Directive

Syntax

```
.WEAK symbol1, ..., symboln
```

Parameters

symbol
The name of the symbol to be declared as weak.

Description

The .WEAK directive is used for extern variables.

Note: In order for a weak symbol to be valid, the symbol must have a global declaration in another file. If not, an error will be issued by the linker once all the object files will be linked.

Examples

```
.WEAK main  
.WEAK func, func1
```

3.3.4 Machine Instructions

VNC2 offers various instructions. The following are the categories of the instructions:

- CPU General
- CPU Stack Operation
- CPU Memory Operation
- CPU Bitwise Shift Operation
- CPU Logic Operation
- CPU Arithmetic Operation
- CPU Bitwise Operation
- CPU I/O Operation
- CPU Comparison
- CPU Program Flow

3.3.4.1 CPU General Instructions

This set of instructions describes the general operations of the CPU. This includes flag handling, interrupts, CPU states and ROM access.

3.3.4.1.1 NOP

Syntax

```
NOP
```

Description

The NOP (No Operation) instruction advances the program counter without altering the CPU state.

3.3.4.1.2 HALT

Syntax

```
HALT
```

Description

Halt the processor at this instruction. Once the CPU has been halted it cannot be un-halted.

3.3.4.1.3 WAIT

Syntax

WAIT

Description

Halt the processor at this instruction, and enter low power mode, until an interrupt is received.

The WAIT instruction uses the general interrupt pin only and the debug interrupt pin is ignored.

Syntax

WAIT ia

Description

Halt the processor at this instruction, and enter low power mode, until hardware interrupt ia is received. If an interrupt other than interrupt ia is received, it will be ignored and the CPU will resume in low power mode.

The WAIT instruction uses the general interrupt pin only and the debug interrupt pin is ignored.

3.3.4.1.4 STOP

Syntax

STOP

Description

Halt the processor, and shut down all internal subsystems.

3.3.4.1.5 RTS

Syntax

RTS

Description

Return from a subroutine call. 3 bytes are removed from the stack and loaded to the program counter, PC. The next instruction executes the instruction at this new PC address.

3.3.4.1.6 IRET

Syntax

IRET

Description

Return from an interrupt. 5 bytes are removed from the stack, 2 bytes for the flags register and 3 bytes for the program counter, PC. The next instruction executes the instruction at this new PC address.

3.3.4.1.7 HCF

Syntax

HCF

Description

N/A

3.3.4.1.8 SAVEF

Syntax

SAVEF

Description

The 16 CPU status flags F are saved in the 16 alternate (mirror) flags F'.

3.3.4.1.9 SWAPF

Syntax

SWAPF

Description

Exchange the 16 primary EMCU flags F with the alternate flag set F'.

3.3.4.1.10 INT

Syntax

INT ia

Description

Generates a software interrupt. The flags register and program counter are pushed to the stack (5 bytes). The program counter is loaded with the address of the debug interrupt service routine. INTT is mapped to interrupt 2 and INTD is mapped to interrupt 1.

3.3.4.1.11 SETfI

Syntax

SETfI

Description

Set the flag indexed by fl.

3.3.4.1.12 CLRfI

Syntax

CLRfI

Description

Clear the flag indexed by fl.

3.3.4.1.13 CPYF

Syntax

CPYF fl fu

Description

Copy CPU flag indexed by fu to flag indexed by fl.

3.3.4.1.14 TXL

Syntax

TXL da sa

Description

Reads a single byte of data from the ROM address pointed to by the 32 bit value stored in memory address sa to memory location specified by da (which may be indirect).

The address pointed to by sa is always a 32 bit value. Note. For all other ROM accesses the ROM is word addressable. However, for this instruction, the ROM is byte addressable.

3.3.4.1.15 WRCODE

Syntax

N/A

Description

N/A

3.3.4.2 CPU Stack Operation Instructions

The VNC2 operates a stack in hardware with a dedicated stack pointer.

3.3.4.2.1 PUSHF

Syntax

PUSHF

Description

Push the flags register to the stack (2 bytes).

3.3.4.2.2 POPF

Syntax

POPF

Description

Retrieve the flags register from the stack (2 bytes).

3.3.4.2.3 SP_INC

Syntax

SP_INC \$b

Description

Increment the stack pointer (SP) by an 8-bit constant value. This has the effect of removing entries from the stack.

3.3.4.2.4 SP_DEC

Syntax

SP_DEC \$b

Description

Decrement the stack pointer (SP) by an 8-bit constant value. This creates additional uninitialised entries on the stack.

3.3.4.2.5 SP_WR#

Syntax

```
SP_WR# da $b
```

Description

Copy the value from a specified memory address to a specified offset from the stack pointer SP. The offset may be up to 255 bytes.

3.3.4.2.6 SP_RD

Syntax

```
SP_RD# da $b
```

Description

Read the memory location at a specified offset from the stack pointer SP, and copy that value to a specified memory address. The offset may be up to 255 bytes.

3.3.4.2.7 SP_STORE

Syntax

```
SP_STORE da
```

Description

Store the stack pointer (WORD) value at a specified memory address. The di bit determines whether the associated stack pointer store is direct or indirect.

3.3.4.2.8 SP_LOAD

Syntax

```
SP_LOAD $w
```

Description

Load a constant value to the stack pointer register (SP).

Syntax

```
SP_LOAD sa
```

Description

Load the stack pointer with the WORD value at a specified memory location. The si bit determines whether the associated stack pointer load is direct or indirect.

3.3.4.2.9 PUSH#

Syntax

```
PUSH8 $b
```

Description

Decrement the SP by 1 and Store a BYTE constant d[7..0] on the stack at the memory location pointed to by the new SP.

Syntax

```
PUSH16 $w
```

Description

Decrement the SP by 2 and store a WORD constant d[15..0] on the stack at the memory location pointed to by the new SP.

Syntax

```
PUSH32 $d
```

Description

Decrement the SP by 4 and store a DWORD constant d[31..0] on the stack at the memory location pointed to by the new SP.

Syntax

```
PUSH# sa
```

Description

Decrement the SP by 1 (BYTE), 2 (WORD), or 4 (DWORD) and load a BYTE, WORD or DWORD value from memory, and store it at the new SP value.

The si bit determines if the memory load is direct or indirect.

3.3.4.2.10 POP

Syntax

```
POP# da
```

Description

Load a BYTE, WORD or DWORD value from the address pointed to by the stack pointer. Store the value to a specified memory address. Increment the stack pointer SP by 1, 2 or 4.

The di bit determines if the memory write is direct or indirect.

3.3.4.3 CPU Memory Operation Instructions

The VNC2 has general purpose memory commands to copy data from ROM to memory or between memory locations.

3.3.4.3.1 LD#

Syntax

```
LD8 da $b
```

Description

Store a BYTE constant at a memory address. The di bit determines whether the memory write is direct or indirect.

Syntax

```
LD16 da $w
```

Description

Store a WORD constant at a memory address. The di bit determines whether the memory write is direct or indirect.

Syntax

```
LD32 da $d
```

Description

Store a DWORD constant at a memory address. The di bit determines whether the memory write is direct or indirect.

3.3.4.3.2 CPYROM

Syntax

```
CPYROM da sa $sc
```

Description

Copy a specified number of words from ROM to memory. The sc[7..0] field specifies the number of words to copy. The di field determines whether the final memory address is the address specified by da[13..0] or the address stored at that memory address. The sa[13:0] address is an address where the ROM memory location is read from.

3.3.4.3.3 CPYMEM

Syntax

```
CPYMEM da sa $sc
```

Description

Copy a specified number of bytes from memory to memory. The sc[7..0] field specifies the number of bytes to copy. The si field determines whether the source memory address is the address specified by da[13..0] or the address stored at that memory address, while the di field determines the final memory address. The ud bit indicates direction (1 for up and 0 for down).

3.3.4.4 CPU Bitwise Shift Operation Instructions

There are 4 types of bitwise shift operations on the VNC2. These are shift, arithmetic shift, rotate and rotate with carry which can be performed on 8, 16 or 32 bit values.

Note: Whereas these operations can shift from 0 to 31 places, the RORC and ROLC instructions only shift by one place. When a value other than 1 is specified for RORC or ROLC, a rotate of 1 will always be performed.

3.3.4.4.1 SHR#

Syntax

```
SHR# da $sc
```

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to the same memory address. The di bit determines whether the memory load/stores are direct or indirect load/stores.

Syntax

```
SHR# da sa $sc
```

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to a different memory address. The di and si bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

```
SHR# da ta
```

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stored to the second data address. The di and ti bits determine whether the memory load/stores are

direct or indirect load/stores.

Syntax

SHR# da sa ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stores to the original data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

3.3.4.4.2 SHL#

Syntax

SHL# da \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to the same memory address. The di bit determines whether the memory load/stores are direct or indirect load/stores.

Syntax

SHL# da sa \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to a different memory address. The di and si bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

SHL# da ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stored to the second data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

SHL# da sa ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stores to the original data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

3.3.4.4.3 SAR#

Syntax

SAR# da \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to the same memory address. The di bit determines whether the memory load/stores are direct or indirect load/stores.

Syntax

SAR# da sa \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to a different memory address. The di and si bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

SAR# da ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stored to the second data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

SAR# da sa ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stores to the original data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

3.3.4.4.4 SAL#

Syntax

SAL# da \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to the same memory address. The di bit determines whether the memory load/stores are direct or indirect load/stores.

Syntax

SAL# da sa \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to a different memory address. The di and si bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

SAL# da ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stored to the second data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

SAL# da sa ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stores to the original data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

3.3.4.4.5 ROR#

Syntax

ROR# da \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to the same memory address. The di bit determines whether the memory load/stores are direct or indirect load/stores.

Syntax

ROR# da sa \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to a different memory address. The di and si bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

ROR# da ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stored to the second data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

ROR# da sa ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stores to the original data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

3.3.4.4.6 ROL#

Syntax

ROL# da \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to the same memory address. The di bit determines whether the memory load/stores are direct or indirect load/stores.

Syntax

ROL# da sa \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to a different memory address. The di and si bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

ROL# da ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stored to the second data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

ROL# da sa ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stores to the original data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

3.3.4.4.7 RORC#

Syntax

RORC# da \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to the same memory address. The di bit determines whether the memory load/stores are direct or indirect load/stores.

Syntax

RORC# da sa \$sc

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to a different memory address. The di and si bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

RORC# da ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stored to the second data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

RORC# da sa ta

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stores to the original data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

3.3.4.4.8 ROLC#

Syntax

```
ROLC# da $sc
```

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to the same memory address. The di bit determines whether the memory load/stores are direct or indirect load/stores.

Syntax

```
ROLC# da sa $sc
```

Description

Loads a BYTE,WORD, OR DWORD value from memory, performs the logical shift operation specified by op[2..0] and sc[4..0], and stores the result to a different memory address. The di and si bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

```
ROLC# da ta
```

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stored to the second data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

Syntax

```
ROLC# da sa ta
```

Description

Loads a BYTE,WORD, OR DWORD data value, and a shift count from two separate memory locations, and performs the logical shift operation specified by op[2..0] and the loaded shift count. The result is stores to the original data address. The di and ti bits determine whether the memory load/stores are direct or indirect load/stores.

3.3.4.5 CPU Logic Operation Instructions

The VNC2 supports various bitwise logic commands. Logic operations can be performed on 8, 16 or 32 bit values.

3.3.4.5.1 AND#

Syntax

```
AND8 da $b
```

Description

Loads a data BYTE from memory, performs a logical operation on it, using the BYTE constant b[7..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
AND16 da $w
```

Description

Loads a data WORD from memory, performs a logical operation on it, using the WORD constant w [15..0] as the second operand, and stores the result to the same memory address. The di bit

determines whether the associated memory load/store is direct or indirect.

Syntax

```
AND32 da $d
```

Description

Loads a data DWORD from memory, performs a logical operation on it, using the DWORD constant d [31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
AND# da sa
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs a logical operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

```
AND# da sa ta
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs a logical operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.5.2 OR#

Syntax

```
OR8 da $b
```

Description

Loads a data BYTE from memory, performs a logical operation on it, using the BYTE constant b[7..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
OR16 da $w
```

Description

Loads a data WORD from memory, performs a logical operation on it, using the WORD constant w [15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
OR32 da $d
```

Description

Loads a data DWORD from memory, performs a logical operation on it, using the DWORD constant d [31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
OR# da sa
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs a logical operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

```
OR# da sa ta
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs a logical operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.5.3 XOR#

Syntax

```
XOR8 da $b
```

Description

Loads a data BYTE from memory, performs a logical operation on it, using the BYTE constant b[7..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
XOR16 da $w
```

Description

Loads a data WORD from memory, performs a logical operation on it, using the WORD constant w[15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
XOR32 da $d
```

Description

Loads a data DWORD from memory, performs a logical operation on it, using the DWORD constant d[31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
XOR# da sa
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs a logical operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

```
XOR# da sa ta
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs a logical operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third

memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.5.4 XNOR#

Syntax

```
XNOR8 da $b
```

Description

Loads a data BYTE from memory, performs a logical operation on it, using the BYTE constant b[7..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
XNOR16 da $w
```

Description

Loads a data WORD from memory, performs a logical operation on it, using the WORD constant w[15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
XNOR32 da $d
```

Description

Loads a data DWORD from memory, performs a logical operation on it, using the DWORD constant d[31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
XNOR# da sa
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs a logical operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

```
XNOR# da sa ta
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs a logical operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.6 CPU Arithmetic Operation Codes

The VNC2 supports arithmetic operations performed on 8, 16 or 32 bit values.

Operand Order

Arithmetic operations are calculated using two or three operand instructions.

2 Operands

For two operand instructions the first operand is both the destination and the first term in the expression. The second operand is the second term in the expression.

This can be represented as:

```
a <- a operation b
```

Where the equivalent machine instruction would be:

```
operation a b
```

3 Operands

The first operand is exclusively for the result. The second and third are the first and second terms in the expression.

To write this as an expression:

```
a <- b operation c
```

Where the equivalent machine instruction would be:

```
operation a b c
```

3.3.4.6.1 ADD#

Syntax

```
ADD8 da $b
```

Description

Loads a data BYTE from memory, performs an arithmetic operation on it, using the BYTE constant b [7..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
ADD16 da $w
```

Description

Loads a data WORD from memory, performs an arithmetic operation on it, using the WORD constant w [15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
ADD32 da $d
```

Description

Loads a data DWORD from memory, performs an arithmetical operation on it, using the DWORD constant d [31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
ADD# da sa
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

```
ADD# da sa ta
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.6.2 ADC#

Syntax

```
ADC8 da $b
```

Description

Loads a data BYTE from memory, performs an arithmetic operation on it, using the BYTE constant b [7..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
ADC16 da $w
```

Description

Loads a data WORD from memory, performs an arithmetic operation on it, using the WORD constant w [15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
ADC32 da $d
```

Description

Loads a data DWORD from memory, performs an arithmetical operation on it, using the DWORD constant d [31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
ADC# da sa
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

```
ADC# da sa ta
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.6.3 SUB#

Syntax

```
SUB8 da $b
```

Description

Loads a data BYTE from memory, performs an arithmetic operation on it, using the BYTE constant b [7..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
SUB16 da $w
```

Description

Loads a data WORD from memory, performs an arithmetic operation on it, using the WORD constant w[15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
SUB32 da $d
```

Description

Loads a data DWORD from memory, performs an arithmetical operation on it, using the DWORD constant d[31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
SUB# da sa
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

```
SUB# da sa ta
```

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.6.4 SBC#

Syntax

```
SBC8 da $b
```

Description

Loads a data BYTE from memory, performs an arithmetic operation on it, using the BYTE constant b[7..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
SBC16 da $w
```

Description

Loads a data WORD from memory, performs an arithmetic operation on it, using the WORD constant w[15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

```
SBC32 da $d
```

Description

Loads a data DWORD from memory, performs an arithmetical operation on it, using the DWORD constant d[31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

SBC# da sa

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

SBC# da sa ta

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.6.5 MUL#

Syntax

MUL16 da \$w

Description

Loads a data WORD from memory, performs an arithmetic operation on it, using the WORD constant w[15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

MUL32 da \$d

Description

Loads a data DWORD from memory, performs an arithmetical operation on it, using the DWORD constant d[31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

MUL# da sa

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

MUL# da sa ta

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.6.6 DIV#

Syntax

DIV16 da \$w

Description

Loads a data WORD from memory, performs an arithmetic operation on it, using the WORD constant w[15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

DIV32 da \$d

Description

Loads a data DWORD from memory, performs an arithmetical operation on it, using the DWORD constant d[31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

DIV# da sa

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

DIV# da sa ta

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.6.7 REM#

Syntax

REM16 da \$w

Description

Loads a data WORD from memory, performs an arithmetic operation on it, using the WORD constant w[15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

REM32 da \$d

Description

Loads a data DWORD from memory, performs an arithmetical operation on it, using the DWORD constant d[31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

REM# da sa

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the

second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

REM# da sa ta

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.6.8 NEG#

Syntax

NEG16 da \$w

Description

Loads a data WORD from memory, performs an arithmetic operation on it, using the WORD constant w[15..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

NEG32 da \$d

Description

Loads a data DWORD from memory, performs an arithmetical operation on it, using the DWORD constant d[31..0] as the second operand, and stores the result to the same memory address. The di bit determines whether the associated memory load/store is direct or indirect.

Syntax

NEG# da sa

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it, using a value loaded from a second memory location as the second operand, and stores the result to the second memory address. The si and di bits determine whether the associated memory load/store is direct or indirect.

Syntax

NEG# da sa ta

Description

Loads a data BYTE, WORD, or DWORD from memory, performs an arithmetic operation on it using a value loaded from a second memory location as the second operand, and stores the result to a third memory address. The si, ti, and di bits determine whether the associated memory load/store is direct or indirect.

3.3.4.6.9 INC#

Syntax

INC# da \$b

Description

Increment the BYTE, WORD, or DWORD value at the specified memory location. The di bit determines whether the memory access is direct or indirect.

3.3.4.6.10 DEC#

Syntax

```
DEC# da $b
```

Description

Decrement the BYTE, WORD, or DWORD value at the specified memory location. The di bit determines whether the memory access is direct or indirect.

3.3.4.7 CPU Bitwise Operation Instructions

This class of instruction performs bit-wise comparisons and inversion. Bit operations can be made with 8, 16 or 32 bit values.

3.3.4.7.1 BTST#

Syntax

```
BTST# sa $b [fl]
```

Description

Load a BYTE, WORD, or DWORD value from memory. Copy the logic value of the bit specified by b [4..0] to the EMCU flag specified by the fl[3..0] bits. The si bit determines whether the memory access is direct or indirect. Default flag to change is the Z flag.

3.3.4.7.2 BCLR#

Syntax

```
BCLR# da $b
```

Description

Load a BYTE, WORD, or DWORD value from memory. Set the bit specified by b[4..0] to zero, and store the result in the same memory location. The di bit determines whether the memory access is direct or indirect.

3.3.4.7.3 BSET#

Syntax

```
BSET# da $b
```

Description

Load a BYTE, WORD, or DWORD value from memory. Set the bit specified by b[4..0] to one and store the result in the same memory location. The di bit determines whether the memory access is direct or indirect.

3.3.4.7.4 INV#

Syntax

```
INV# da $b
```

Description

Invert the BYTE, WORD, or DWORD value at the specified memory location. The di bit determines whether the memory access is direct or indirect.

Syntax

```
INV# da sa
```


Description

Load the BYTE, WORD, or DWORD value at a specified memory location, invert it, and store to another location. The di and si bits determines whether the associated memory access is direct or indirect.

3.3.4.7.5 CPY#

Syntax

```
CPY# da sa
```

Description

Load the BYTE, WORD, or DWORD value at a specified memory location, copy it, and store to another location. The di and si bits determines whether the associated memory access is direct or indirect.

3.3.4.8 CPU I/O Operation Instructions

The VNC2 supports commands to perform operations on I/O ports.

3.3.4.8.1 OUTPORT

Syntax

```
OUTPORT io $b
```

Description

Write the constant BYTE value b[7..0] to the I/O port specified by io[8..0]

Syntax

```
OUTPORT io sa
```

Description

Copy a BYTE value from memory to the I/O port specified by io[8..0]. The si bit determines whether the memory load/store operation is direct or indirect.

3.3.4.8.2 ANDPORT

Syntax

```
ANDPORT io $b
```

Description

Read a BYTE value from the I/O port specified by io[8..0]. AND the value with the BYTE constant b [7..0] and write the result back to the same I/O port.

Syntax

```
ANDPORT io da $b
```

Description

Read a BYTE value from the I/O port specified by io[8..0]. AND the value with the BYTE constant b [7..0], and write the result to a memory location. The di bit determines whether the memory write operation is direct or indirect.

3.3.4.8.3 ORPORT

Syntax

```
ORPORT io $b
```

Description

Read a BYTE value from the I/O port specified by io[8..0]. OR the value with the BYTE constant b[7..0] and write the result back to the same I/O port.

Syntax

```
ORPORT io da $b
```

Description

Read a BYTE value from the I/O port specified by io[8..0]. OR the value with the BYTE constant b[7..0], and write the result to a memory location. The di bit determines whether the memory write operation is direct or indirect.

3.3.4.8.4 INPORT

Syntax

```
INPORT io da
```

Description

Copy a BYTE value from the I/O port specified by io[8..0] to memory. The di bit determines whether the memory load/store operation is direct or indirect.

3.3.4.8.5 PORTTST

Syntax

```
PORTTST io $b [fl]
```

Description

Read a BYTE value from the I/O port specified by io[8..0] and copy the value of the bit specified by b[3..0] to the EMCU flag specified by fl[3..0] Default flag to change is the Z flag.

3.3.4.9 CPU Comparison Instructions

The compare instruction compares the sa value to an immediate or da value. The sa parameter is used as the 'A' value and the immediate or da parameter is the 'B' value in the condition.

3.3.4.9.1 CMP#

Syntax

```
CMP8 sa $b
```

Description

Load the BYTE value at a specified memory address, and compare to a BYTE constant b[7..0], setting the O S C and Z flags according to the result. The si bit determines whether the memory reference is direct or indirect.

Syntax

```
CMP16 sa $w
```

Description

Load the WORD value at a specified memory address, and compare to a WORD constant w[15..0], setting the O S C and Z flags according to the result. The si bit determines whether the memory reference is direct or indirect.

Syntax

```
CMP32 sa $d
```

Description

Load the DWORD value at a specified memory address, and compare to a DWORD constant d[31..0],

setting the O S C and Z flags according to the result. The si bit determines whether the memory reference is direct or indirect.

Syntax

```
CMP# sa da
```

Description

Load two BYTE, WORD or DWORD values from two memory addresses, and compare them, setting the O S C and Z flags according to the result. The si and di bits determine whether the associated memory reference is direct or indirect.

3.3.4.10 CPU Program Flow Instructions

Program flow is controlled by straight jumps or calls, indirect jumps and calls or conditional jumps and calls. Conditional jumps and calls are made either on flags states or on comparison results.

3.3.4.10.1 JUMP

Syntax

```
JUMP ro
```

Description

Load the program counter PC with the value specified by ro. The next instruction will use the new address, with the effect that program execution jumps to that location.

Syntax

```
JUMP (sa)
```

Description

Load the program counter PC with the value specified by sa. The next instruction will use the new address, with the effect that program execution jumps to that location.

3.3.4.10.2 JGT

Syntax

```
JGT ro  
JGTS ro
```

Description

If the flags indicate a greater than result, jump to the rom address (ro), otherwise continue to the next address. The sg bit determines a signed (1) or unsigned (0) comparison. This is indicated by an S on the assembly instruction.

3.3.4.10.3 JGE

Syntax

```
JGE ro  
JGES ro
```

Description

If the flags indicate a greater than or equal to result, jump to the rom address (ro), otherwise continue to the next address. The sg bit determines a signed (1) or unsigned (0) comparison. This is indicated by an S on the assembly instruction.

3.3.4.10.4 JLT

Syntax

```
JLT ro
```

JLTS ro

Description

If the flags indicate a less than result, jump to the rom address (ro), otherwise continue to the next address. The sg bit determines a signed (1) or unsigned (0) comparison. This is indicated by an S on the assembly instruction.

3.3.4.10.5 JLE

Syntax

JLE ro
JLES ro

Description

If the flags indicate a less than or equal to result, jump to the rom address (ro), otherwise continue to the next address. The sg bit determines a signed (1) or unsigned (0) comparison. This is indicated by an S on the assembly instruction.

3.3.4.10.6 JNfI

Syntax

JNfI ro

Description

If the flag specified by fl[3..0] is zero, then jump to the ROM address RO, otherwise continue to the next instruction.

3.3.4.10.7 JfI

Syntax

JfI ro

Description

If the flag specified by fl[3..0] is set then jump to the ROM address RO, otherwise continue to the next instruction.

3.3.4.10.8 CALL

Syntax

CALL ro

Description

Store the value of the program counter + 2 (the address of the next instruction) in the memory address pointed to by SP. Decrement SP by 3. Load the new ROM address RO to the program counter PC, and continue execution from the new address.

Syntax

CALL (sa)

Description

Store the value of the program counter + 2 (the address of the next instruction) in the memory address pointed to by SP. Decrement SP by 3. Load the new ROM address from the memory location specified by sa to the program counter PC, and continue execution from the new address.

3.3.4.10.9 CALLGT

Syntax

```
CALLGT ro
CALLGTS ro
```

Description

If the flags indicate a greater than result, jump to the rom address (ro), otherwise continue to the next address. The sg bit determines a signed (1) or unsigned (0) comparison. This is indicated by an S on the assembly instruction.

If the branch is taken, store the value of the program counter + 2 (the address of the next instruction) in the memory address pointed to by SP. Decrement SP by 3. Load the new ROM address RO to the program counter PC, and continue execution from the new address.

3.3.4.10.10 CALLGE

Syntax

```
CALLGE ro
CALLGES ro
```

Description

If the flags indicate a greater than or equal to result, jump to the rom address (ro), otherwise continue to the next address. The sg bit determines a signed (1) or unsigned (0) comparison. This is indicated by an S on the assembly instruction.

If the branch is taken, store the value of the program counter + 2 (the address of the next instruction) in the memory address pointed to by SP. Decrement SP by 3. Load the new ROM address RO to the program counter PC, and continue execution from the new address.

3.3.4.10.11 CALLLT

Syntax

```
CALLLT ro
CALLLTS ro
```

Description

If the flags indicate a less than result, jump to the rom address (ro), otherwise continue to the next address. The sg bit determines a signed (1) or unsigned (0) comparison. This is indicated by an S on the assembly instruction.

If the branch is taken, store the value of the program counter + 2 (the address of the next instruction) in the memory address pointed to by SP. Decrement SP by 3. Load the new ROM address RO to the program counter PC, and continue execution from the new address.

3.3.4.10.12 CALLLE

Syntax

```
CALLLE ro
CALLLES ro
```

Description

If the flags indicate a less than or equal to result, jump to the rom address (ro), otherwise continue to the next address. The sg bit determines a signed (1) or unsigned (0) comparison. This is indicated by an S on the assembly instruction.

If the branch is taken, store the value of the program counter + 2 (the address of the next instruction) in the memory address pointed to by SP. Decrement SP by 3. Load the new ROM address RO to the program counter PC, and continue execution from the new address.

3.3.4.10.13 CALLNfI

Syntax

```
CALLNfI ro
```

Description

If the EMCU flag specified by fl[3..0] is zero, then branch to the new address RO storing the return address, otherwise continue to the next instruction.

If the branch is taken, store the value of the program counter + 2 (the address of the next instruction) in the memory address pointed to by SP. Decrement SP by 3. Load the new ROM address RO to the program counter PC, and continue execution from the new address.

3.3.4.10.14 CALLfl

Syntax

```
CALLfl ro
```

Description

If the EMCU flag specified by fl[3..0] is set, then branch to the new address RO storing the return address, otherwise continue to the next instruction.

If the branch is taken, store the value of the program counter + 2 (the address of the next instruction) in the memory address pointed to by SP. Decrement SP by 3. Load the new ROM address RO to the program counter PC, and continue execution from the new address.

3.3.5 Error Reference

Assembler error messages take the following form:

```
<filename> line <line number>: (error|warning|info) A<code> <description>
```

Error codes take one of the following values.

Error codes	Description
1000	neither instruction nor directive
1001	invalid directive syntax.
1002	not supported directive
1003	not supported instruction
1004	instruction format not supported
1005	syntax error.
2000	missing .ENUM directive.
2001	invalid line number.
2002	missing ENUM name.
2003	missing .STRUCT directive.
2004	missing struct name.
2005	struct/union name mismatch.
2006	missing .UNION directive.
2007	missing union name
2008	function name mismatch.
2009	enum name mismatch
2010	missing enumerators
2011	missing struct/union members
2012	missing .FUNCTION directive
2013	missing .FUNC_END directive
2014	missing function name.
2015	missing parameter name
2016	missing datatype name

2017	missing function return type
2018	invalid size specified.
2019	invalid array dimension.
2020	invalid pointer dimension
2021	invalid signed/unsigned flag.
2022	invalid array flag.
2023	invalid pointer flag.
2024	duplicate return type
2025	missing .DATA_DEF directive.
2026	missing struct initialized value/s.
2027	multiple global declaration.
3000	failed to assemble instruction
3001	instruction not allowed in data section
3002	unknown section '%s'
3003	invalid section type
3004	invalid section flag
3005	symbol not yet declared '%s'.
3006	symbol not found in string table
3007	invalid filename
3008	missing .FILE directive.
3009	failed to create object file.
3010	symbol does not exist in .symtab
3011	symbol does not need to be relocated
3012	unresolved symbol '%s'
3013	invalid ORG value, address overlaps
3014	indirect access is not allowed.
3015	value is greater than the operand size.
3016	duplicate symbol
3017	directive not allowed in text section.
3018	directive is only allowed in text section.
4000	unknown datatype '%s'
4001	failed to create .debug_abbrev section.
4002	failed to create .debug_info section.
5000	malloc() failed
5001	internal error
5002	failed to create .symtab section.
5003	failed to create .rel.text section.
5004	has reached the maximum number of sections.
5005	no entry in symtab yet.
5006	unable to open file '%s'.
5007	unable to open log file.

Example

An error for an undefined label in line 45 will give the following message.

```
test.asm line 45: (error) C1200 undefined label
```

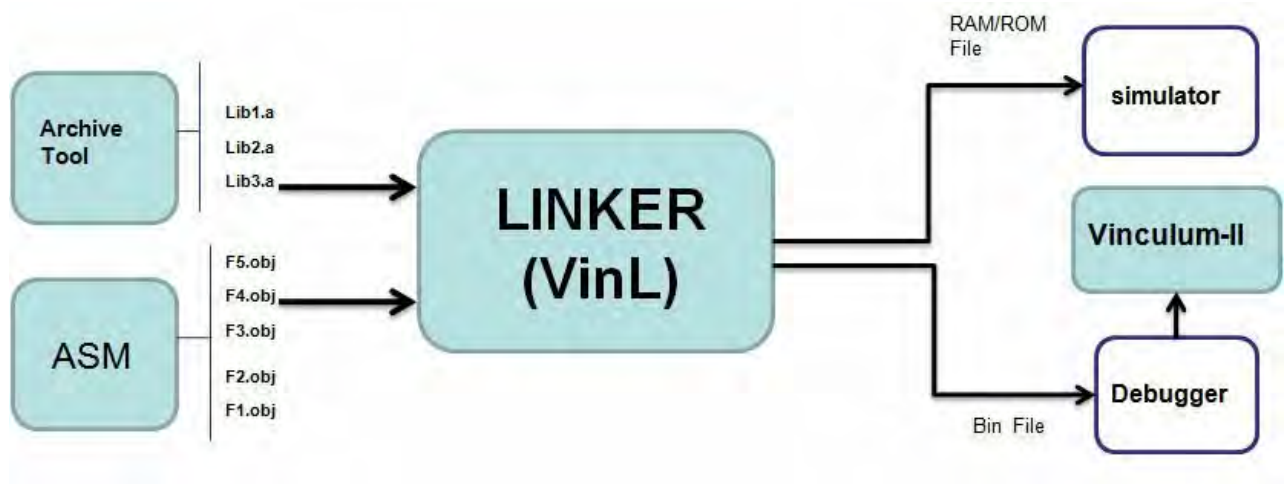
3.4 VinL Linker

The Linker is a tool that generates single executable/loadable file after linking multiple object files provided by assembler and it supports archive file processing and DWARF2 compliant debug file generation.

The input of linker is an Object file which is in ELF format. The output of linker is RAM,ROM and Binary Files.

RAM and ROM files are generated for the Simulator.

Binary file generated for Debugger and VNC2 programming.



3.4.1 Linker Command Line Options

The VNC2 Linker command line options are listed below.

```
VinL [options] [file ...]
```

Option	Description
-e symbol	Define Entry Symbol.
-k bytes	Set size of stack in bytes
-O	Enable optimisations in the linker
-d level	Specify the debug level for the linker output
-o filename	Set the output filename and path File extensions will be appended to this filename
-U	Specify that full archives are to be included.
-h	Print Command help message of linker options
-v	Verbose flag
-V	Version number
--no-loader	User defined program loader
-B	Image offset in ROM specified in words Default is 0x3C0
-T	Text section offset in ROM specified in words Default is zero
-D	Data section offset in RAM specified in bytes Default is zero

Examples

How to view the Linker options?

```
VinL -h
```

Application name followed by option name while viewing command line option

How to see version number of linker?

```
VinL -v
```

How to Execute Object files?

```
VinL object1.obj object2.obj bootloader.obj --no-loader
```

Linker Application name followed by object files.

How to enable linker optimization?

```
VinL -O
```

How to generate the output file?

```
VinL object1.obj -o=c:\object
```

or

```
VinL object1.obj -o=object
```

Command line specifying an output file of "object" will generate the output files "object.bin" and "object.rom".

How to Use different commands in linker?

All the command mentioned here are based on GPIOKitt sample project.

- How to specify data segment start address? Default Data Segment Start Address is 0x0

```
VinL.exe kernel.a gpio.a Kitt.obj -U -D 0x20 -o Kitt
```

- How to specify Text Section Start Address? Default is 0x0.

```
VinL.exe kernel.a gpio.a Kitt.obj -U -T 0x10 -o Kitt
```

- How to specify the Code segment Start Address? Default Code Start Address is 0x3C0. Please contact customer support as there are implications with moving the start address of the code.

```
VinL.exe kernel.a gpio.a Kitt.obj -U -B 0x400 -o Kitt
```

- How to specify user defined boot loader and what care needs to be taken?

```
VinL.exe kernel.a gpio.a Kitt.obj bootloader.obj -e Start --no-loader -U -o Kitt
```

If you specify the start symbol as "Start". It must also be defined as that will be the code entry point. The default entry symbol is "main".

- How to specify the stack size?

```
VinL.exe kernel.a gpio.a Kitt.obj -k 0x200 -U -D 0x20 -B 0x400 -o Kitt
```

- How to include archive files?

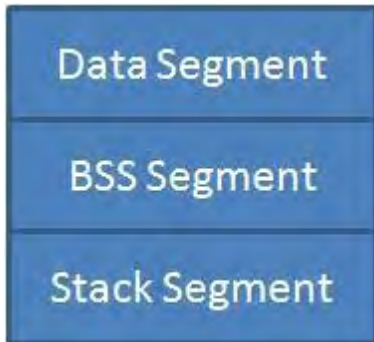
To include archive file -U option must be provided, failing to do so will result in an error being reported.

3.4.2 Memory and Segment

FLASH - ROM Memory



RAM Memory



TEXT Segment :

This is the code segment. All C Functions, Procedures etc. are merged together and created as one text segment. Assembler replaces C functions with CPU specific instructions, so all CPU instructions are part of this segment. This Segment is part of FLASH(ROM) Memory in Vinculum II.

DATA Segment :

This is the DATA segment. All Global Variables, Structures etc. are merged together and created as one data segment. DATA segment is the fixed memory locations reserved for global variables of all type. i.e. char, int, long etc.

BSS Segment :

This is the DATA segment. All uninitialised Global Variables, Structures etc. are merged together and created as one BSS segment. BSS segment is same as DATA segment, is the fixed memory locations reserved for uninitialised global variables of all type. Program loader (Part of boot loader) initializes this segment

RO Segment :

This is the DATA segment. All Global Variables, Structures etc. are merged together and created as one data segment. DATA segment is the fixed memory locations reserved for global variables of all type. i.e. char, int, long etc

DEBUG Segment :

This is the Debug segment. Debug information for all sections, variables are part of this section. Debug segment is not the part of physical memory so it is not part of CPU memory map. This segments are used by debugger for providing debug information.

3.4.3 Linker Optimization

The Linker optimization technique will reduce the speed of processor and code size of executable file.

Vinculum-2 linker supports Dead code elimination optimization.

Dead Code Elimination is an optimization technique to eliminate the unused functions/data symbol (dead data) from the code/data. Result of this optimization will reduce the code size in executable or binary file

The linker optimization is enabled when -O is given in command line.

Example for Linker Optimization

Please consider below given test case:

```
char Result = 0x00;
void updateResult(int);
void main()
{
    int x = 1;
    Result ++;
}
void updateResult(int k) // function defined but not used anywhere in the testcase
{
    // so function is considered as deadcode
    k++;
}
```

3.4.4 Map File

Linker Map file contains following information

- RAM and ROM size of CPU
- Version info of Linker and Loader
- Start address of segment and size of each segment
- Symbol information (Address of symbol, symbol name, symbol type, memory type, filename and size of symbol)

Example for Linker Map File

Future Technology Devices International Limited (c) 2007, All Rights Reserved

CPU_CORE = VC2, CPU_RAM_SIZE= 16 KB, CPU_ROM_SIZE=256 KB

Linker Version = V1.100000 Loader Version = V1.000000

Fri Nov 20 13:53:13 2009

SEGMENT_MAP

Start Address	Segment Name	Segment Size
000000	.text	0000000012
000012	.dataFlash	000000001e
000000	.dataRAM	000000001e
00001e	.bss	0000000000

SYMBOL_TABLE

SYM ADDRESS	PUBLIC	SYM NAME	SYM TYPE	MEM TYPE	FILE NAME	SYM SIZE
NOTE: L=LABEL F=FUNCTION 0=FLASH and 1=RAM						
000000		main	F	0	test.obj	00007
000007		updateResult	F	0	test.obj	00001
000008		updateResult1	F	0	test.obj	00001
000000		%eax	L	1	test.obj	00004
000004		%ebx	L	1	test.obj	00004
000008		%ecx	L	1	test.obj	00004
00000c		%r0	L	1	test.obj	00004
000010		%r1	L	1	test.obj	00004
000014		%r2	L	1	test.obj	00004
000018		%r3	L	1	test.obj	00004
00001c		a	L	1	test.obj	00001
00001d		Result	L	1	test.obj	00001

3.4.5 Archive File

VinL linker will support only archive file generated by VinAr tool.

Features

Following is the feature list of archive support in toolchain. This list includes items for both tool and linker support.

Feature	Description	Benefit/Reason
Selective Archive extraction	Selective archive extraction uses technique to import symbol based on the symbols required as per the relocation from application. Global Symbol table created by archive tool will be used to extract the data.	This removes need of including code first and then removing. This will help to reduce binary size
Full archive extraction	This is an option which by default is disabled	Faster inclusion time
Command line options	Command line option to archive tool	Ease of use
Name Mangling	To protect internal working and hide internal functions from user	Security of code
Debug information support	Not Supported	Not possible as source code debugging is not allowed
Tool compatibility	Archive tool will be only FTDI's toolchain compatible. Linker will support archives generated by Archive tool only.	Security of IP and software
Export symbol creation	This symbols will be created by archive tool to be used for name mangling processing	

3.4.6 Error Reference

Linker error messages take the following form:

```
<filename>: (error|warning|info) L<code> <description>
```

Error codes take one of the following values.

Error codes	Description
0001	error in command line argument
0002	memory overflow
0003	error invalid parameter
0004	start symbol missing
0005	symbol without section
0006	symbol redefined (symbol name)
0007	symbol allocation error
0008	file allocation
0009	section not supported
0010	error in file open
0011	error in memory allocation
0012	error in parameter
0013	error in parameter section information
0014	error in parameter symbol data
0015	error in parameter in relocation data
0016	error in updating linker variables
0017	invalid archive file

0018	-U option is missing for archive
0019	archive symbol is missing (symbol name)
0020	syntax error
0021	error in maximum code limit

Error Code: command line argument Error

This error means one or more command line parameters to the linker is wrong.

```
test.obj : (error) L0001 error in command line parameter
```

Error Code: memory overflow

The generated ROM code was too large to fit into the Flash memory on the device.

```
test.obj : (error) L0002 memory overflow if text/data/bss sections are large to corrupt with any of the other section
```

Error Code: invalid parameter

```
test.obj : (error) L0003 invalid parameter (generated due to invalid elf format)
```

Error Code: start symbol missing

An error for a program which does not have a main() function will give the following message. or if it's user defined

start symbol missing then this error will be issued

```
test.obj : (error) L0004 start symbol missing
```

Error Code: symbol without section

Internal error thrown due to incorrect elf format.

Error Code: symbol redefined

Internal error thrown due to incorrect elf format.

Error Code: symbol allocation error

Internal error thrown due to incorrect elf format.

Error Code: file allocation

Internal error thrown due to incorrect elf format.

Error Code: section not supported

Internal error thrown due to incorrect elf format.

Error Code: error in file open

File provided as an input parameter is not found/corrupted.

Error Code: error in memory allocation

Internal error thrown due to incorrect elf format.

Error Code: error in parameter

Internal error thrown due to incorrect elf format.

Error Code: error in updating linker variables

Internal error thrown due to incorrect elf format.

Error Code: error in parameter in relocation data

A function or variable was declared (prototyped) and called but the linker could not find a definition for the function or variable.

```
test.obj: (error) L0015 error in parameter in relocation data notRealFunction
```

Error Code: archive file is not valid and wrong/duplicate archive symbol

This error will be generated if generated archive file is not standard FTDI format.

Error Code: -U option is missing for archive

Command line parameter -U is required since an archive file was used as an input file.

Error Code: archive symbol missing

Internal error for corrupt archive file.

Error Code: syntax error

Internal error for corrupt archive file.

Error Code: maximum code limit

if code size increases beyond FLASH ROM size.

3.4.7 Special VNC2 Reference

Certain symbols are defined in the linker which are available to a user program.

The following are symbols defined in ROM which may be read by a program.

userDataArea	8 word array for user programmable area	extern rom char userDataArea [16] extern rom short userDataArea [8] extern rom long userDataArea[4]
Start	Label for start of program	
progDataArea	8 word array for program data area	reserved
progSize	2 word size of program ROM	extern rom int progSize[1]

The data accessible from the ROM in this case can only be accessed as an array.

Examples

To read in the user data area:

```
extern rom char userDataArea[16];

void getProgSize(char *buff16byte)
{
    int x;
    for (x=0; x < 16; x++)
    {
        buff16byte[x] = userDataArea[x];
    }
}
```

To query the program size:

```
extern rom int progSize[1];
```

```
int getProgSize()
{
    int x;
    x = progSize[0];
    return x;
}
```

3.5 VinIDE

The Integrated Development Environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. The IDE consists of:

- A source code editor
- A compiler
- Build automation tools
- A debugger

3.5.1 About VinIDE

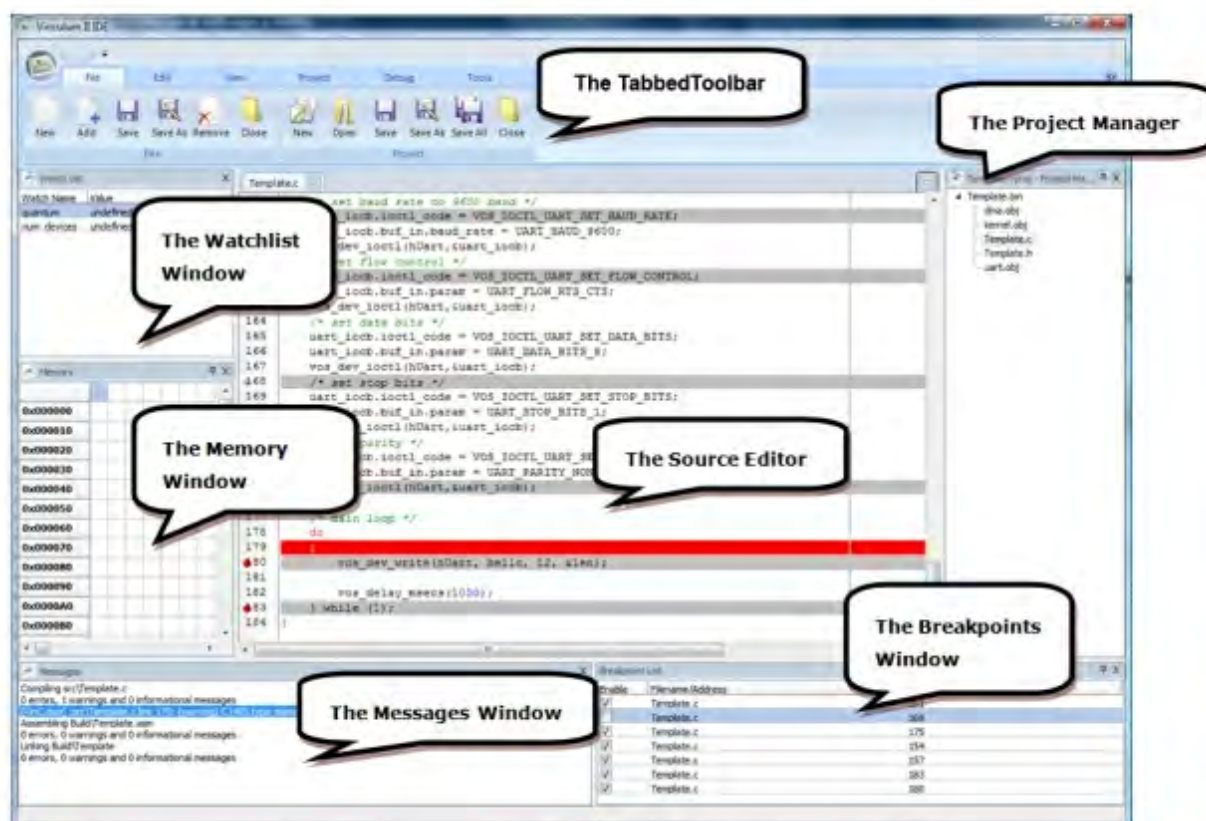
VinIDE is an IDE that can be used to create user applications for the VNC2 chip. It has its own built-in source editor to help you write your own source files. It also allows you to build your source files into the binary output by using the integrated tools in the toolchain namely:

- The Compiler (VinC.exe)
- The Assembler (VinAsm.exe)
- The Linker (VinL.exe)
- The Debugger (VinDbg.exe)

It also allows the user to manage the files in a project.

3.5.2 The User Interface

The IDE is divided up into 6 parts : the Tabbed Toolbar, the Source Code Editor, the Project Manager, the Messages Window, the Watchlist Window, and the Memory Window.



3.5.2.1 The Tabbed Toolbar

The Tabbed Toolbar is context-sensitive, automatically displaying the functions relevant to what you are doing at the moment. Functions that cannot be used in the current context are greyed out.



3.5.2.1.1 The File tab

The File tab group is a collection of file IO-related commands that is available for the user. Below are the commands on the File tab group.



□ File Group

- New

Adds a new file to the current project

- Add

Adds a an existing file to the current project

- Save

Saves to the disk the active file on the editor

- Save As

Saves on a different filename the active file on the editor

- Remove

Removes the active file from the current project

- Close

Closes the active file from the current project

□ Project Group

- New

Creates a new project

- Add

Opens an existing project

- Save

Saves the project to the disk

- Save As

Saves the project on a different filename

- Save All

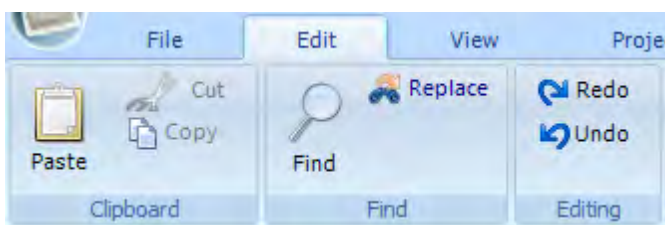
Saves the project as well as all the edited files

- Close

Closes the project

3.5.2.1.2 The Edit tab

The Edit tab group is a collection of commands that are used in conjunction with the text editor.



3.5.2.2 The Source Code Editor

This is the large area below the tabbed toolbar where the contents of the files in the project are shown and edited. It has many of the features of an advanced text editor such as multiple opened tabbed files, line highlighting, syntax styling, and many more.

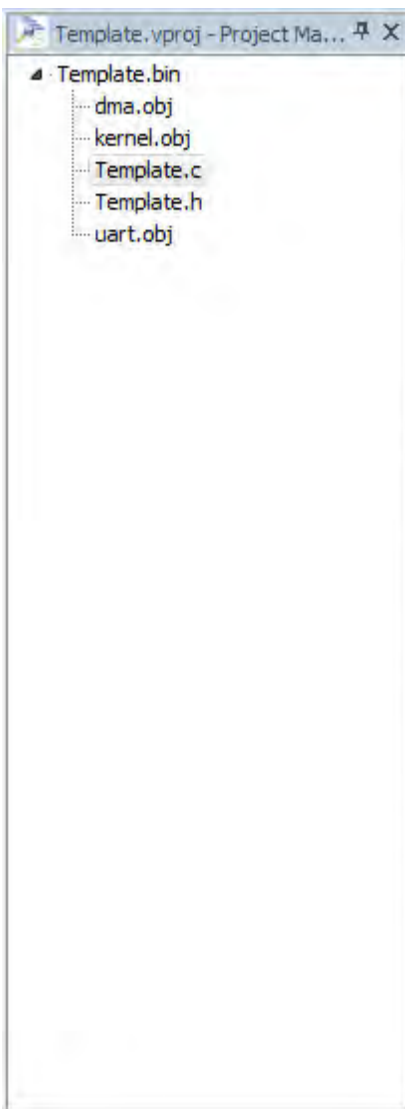
```

Template.c
156  /* set baud rate to 9600 baud */
157  uart_iocb.iocctl_code = VOS_IOCTL_UART_SET_BAUD_RATE;
158  uart_iocb.buf_in.baud_rate = UART_BAUD_9600;
159  vos_dev_ioctl(hUart, &uart_iocb);
160  /* set flow control */
161  uart_iocb.iocctl_code = VOS_IOCTL_UART_SET_FLOW_CONTROL;
162  uart_iocb.buf_in.param = UART_FLOW_RTS_CTS;
163  vos_dev_ioctl(hUart, &uart_iocb);
164  /* set data bits */
165  uart_iocb.iocctl_code = VOS_IOCTL_UART_SET_DATA_BITS;
166  uart_iocb.buf_in.param = UART_DATA_BITS_8;
167  vos_dev_ioctl(hUart, &uart_iocb);
168  /* set stop bits */
169  uart_iocb.iocctl_code = VOS_IOCTL_UART_SET_STOP_BITS;
170  uart_iocb.buf_in.param = UART_STOP_BITS_1;
171  vos_dev_ioctl(hUart, &uart_iocb);
172  /* set parity */
173  uart_iocb.iocctl_code = VOS_IOCTL_UART_SET_PARITY;
174  uart_iocb.buf_in.param = UART_PARITY_NONE;
175  vos_dev_ioctl(hUart, &uart_iocb);
176
177  /* main loop */
178  do
179  {
180      vos_dev_write(hUart, hello, 12, &len);
181
182      vos_delay_msecs(1000);
183  } while (1);
184  }

```

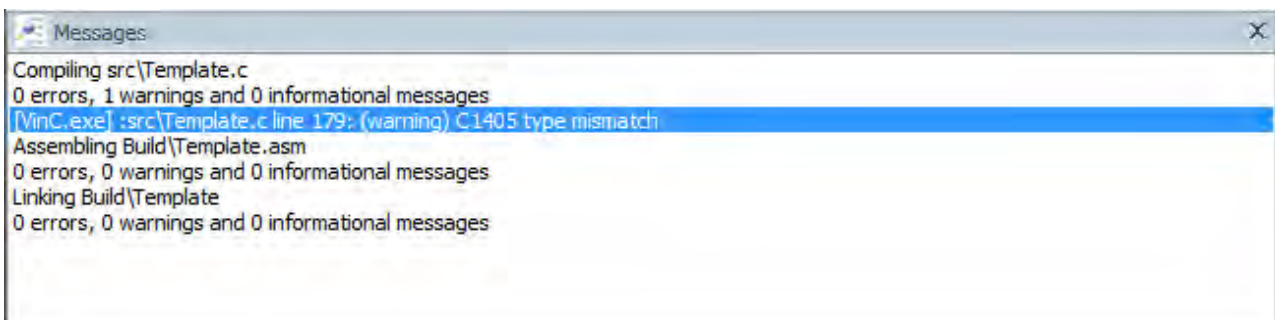
3.5.2.3 The Project Manager

The Project Manager displays the project in a tree view form showing the files that are included in the application. Many of the commands in the toolbar can also be quickly accessed in the Project Manager thru the right click mouse button.



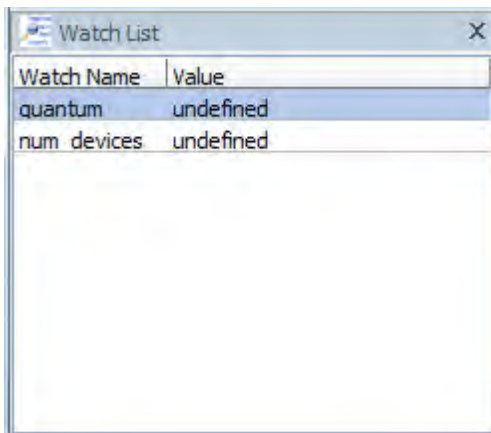
3.5.2.4 The Messages Window

The Messages Window displays the information that are being sent by the other tools such as the compiler and linker as well as the result from the search commands.



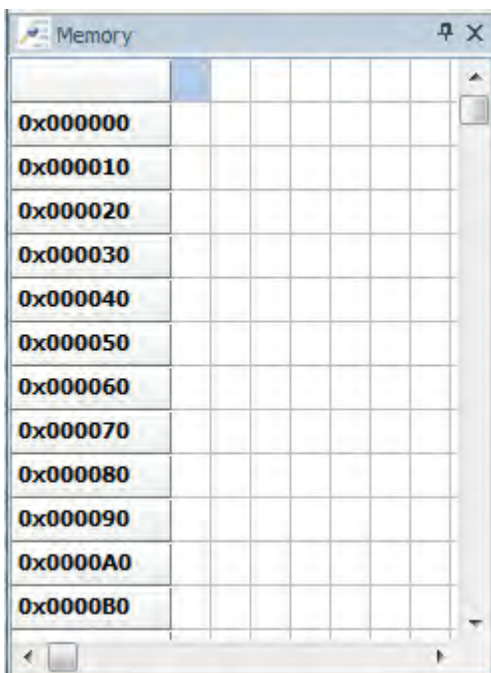
3.5.2.5 The Watchlist Window

The Watchlist Window is used to evaluate values of variables and expressions..



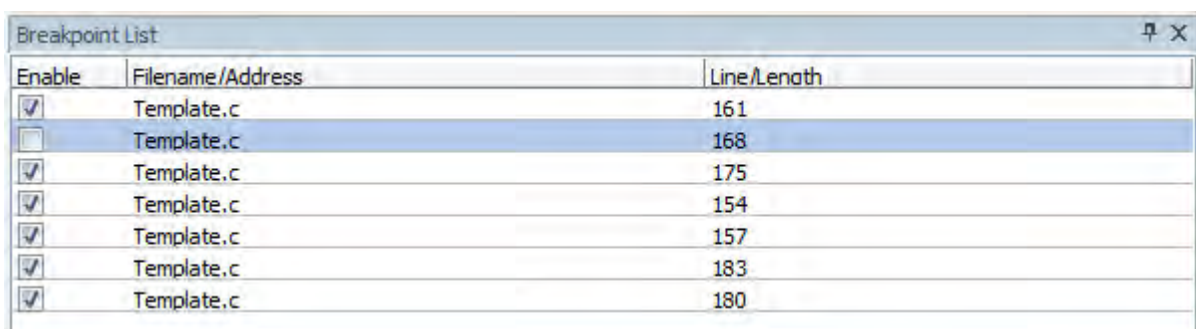
3.5.2.6 The Memory Window

The Memory Window is used to display and evaluate the current contents of the memory of the target chip.



3.5.2.7 The Breakpoint Window

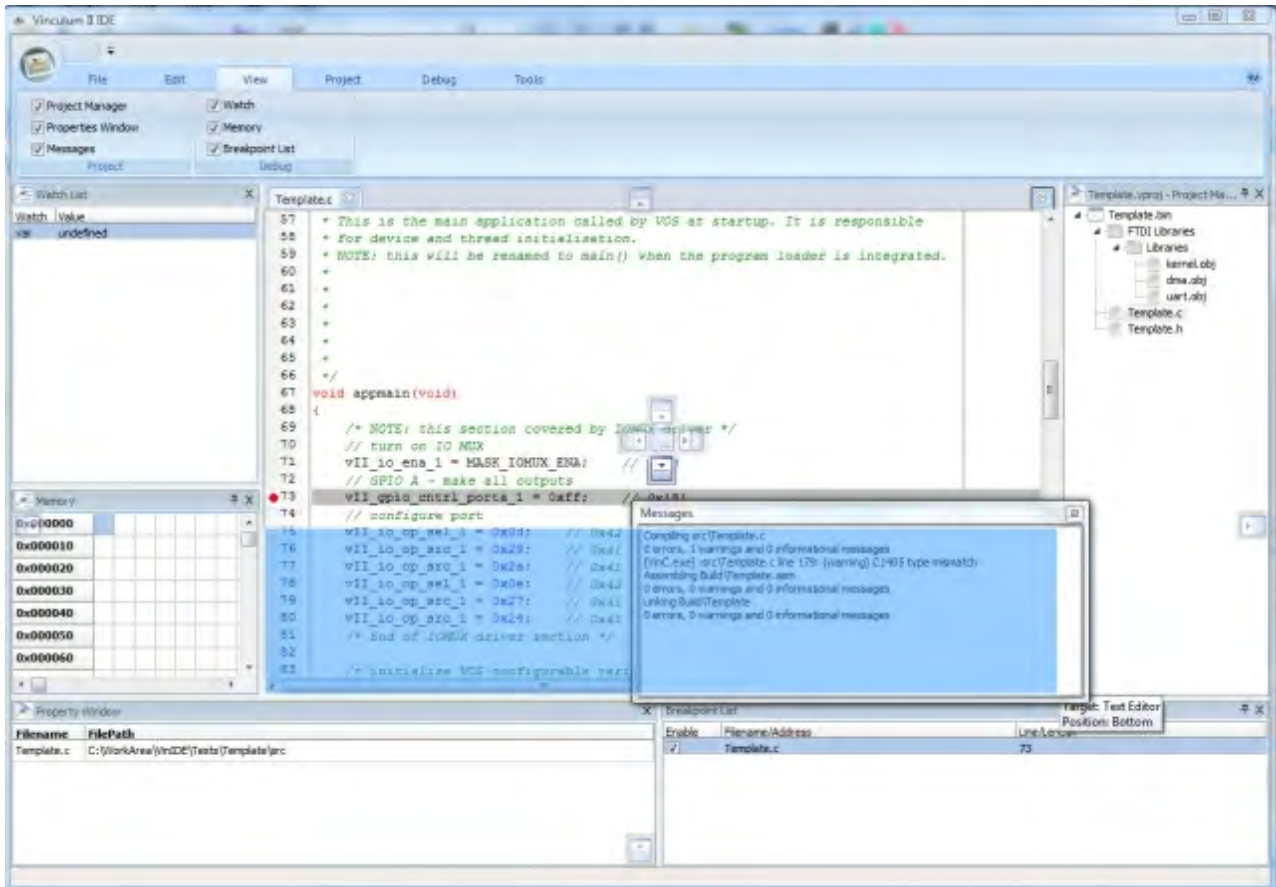
The Breakpoint Window is used to display and manage the breakpoint for debugging the project application



Note : Any number of breakpoints can be entered but only first three will be enabled and active during debugging.

3.5.2.8 Managing the Panels

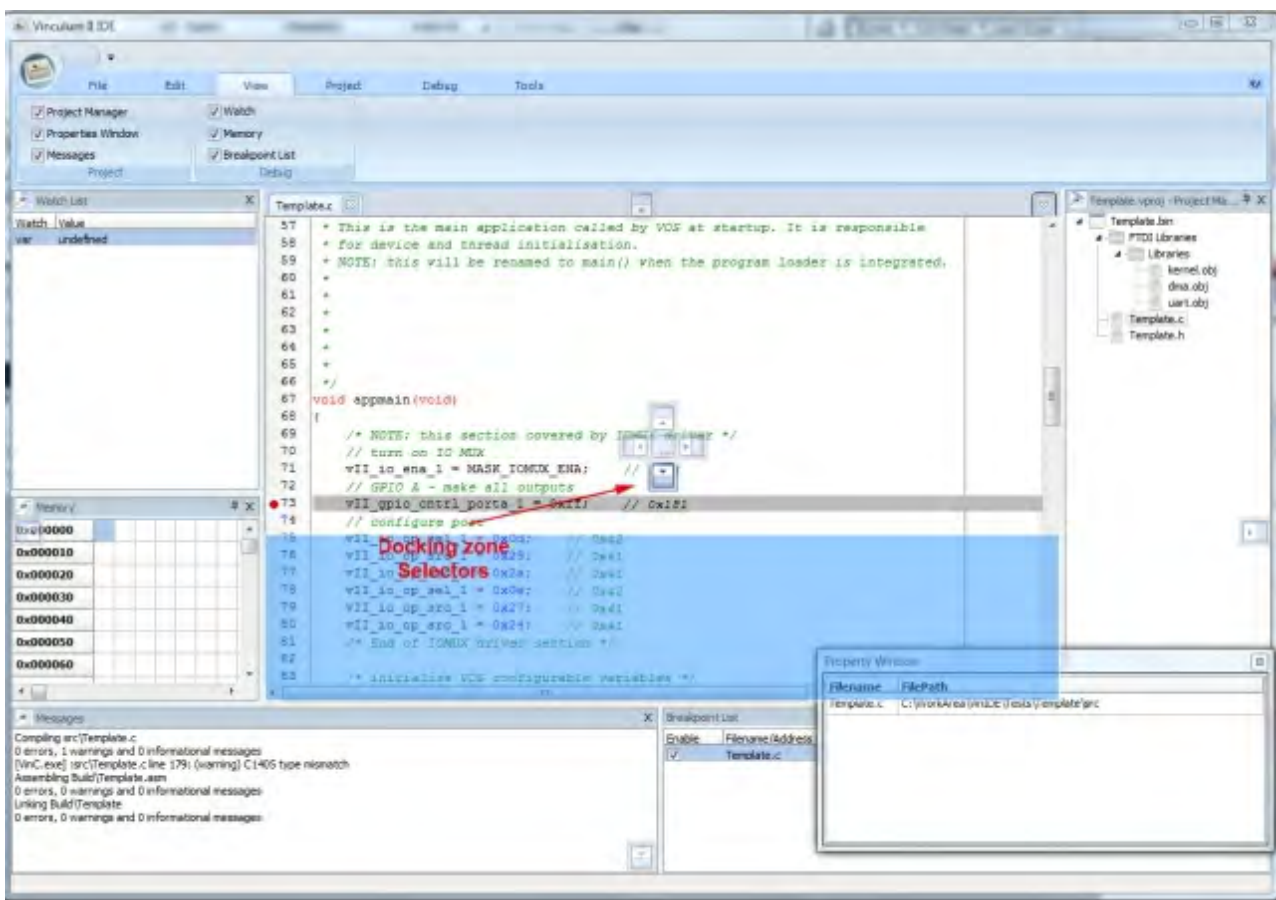
VinIDE is equipped with docking facilities that enables the user to drag the panels and then dock them to docking sites or stack them on top of one another. The panels could also be pinned to the sides and be visible only when the mouse hovers on top of their tabs. Or the user could simply hide the panels and display them whenever so they chooses.



Please note that when you close the IDE the last states of the panels will be saved and will be used when the IDE starts again so that the user will not have to rearrange them every time the IDE starts.

3.5.2.8.1 Docking/Undocking Panels

To dock a panel simply click on the title bar and drag to any of the docking zone selectors that will be displayed. The blue preview highlight will show you where the panel will be docked if you release it.

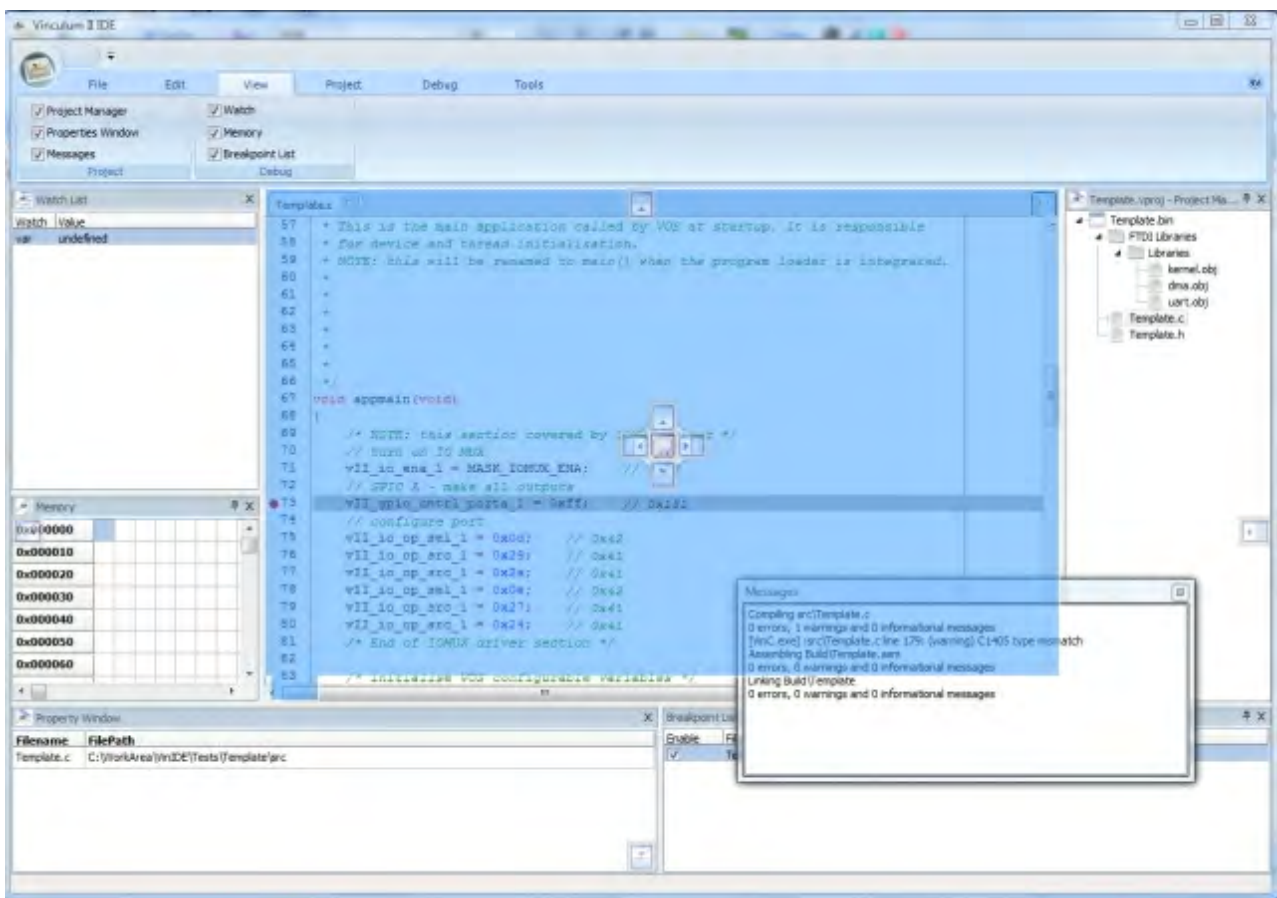


To undock a panel just click on the title bar again and drag to any area in the main window.

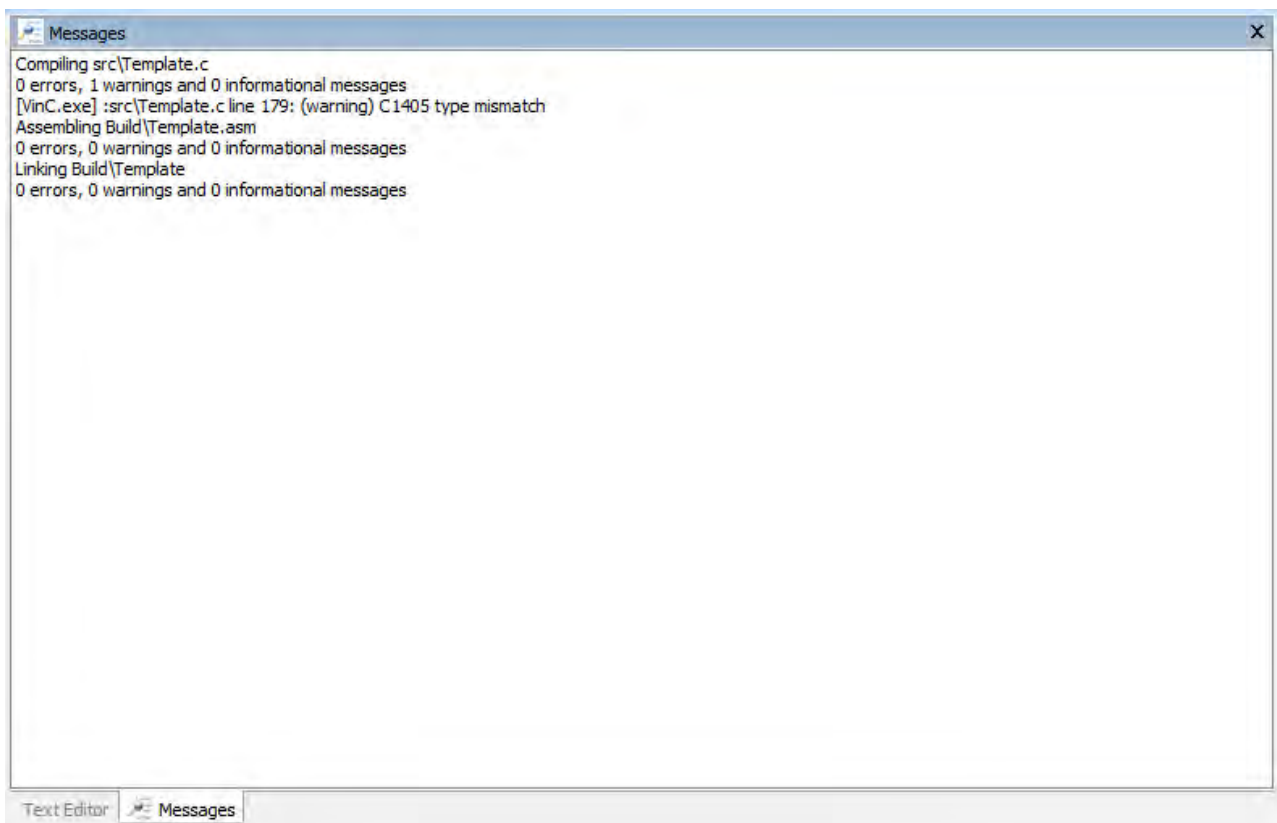
3.5.2.8.2 Stacking Panels

Stacking panels is just another way of docking them. But while docking a panel in a docking site will make the two panels occupy half of the area, stacking them together will enable both panels to use the whole area for themselves but with only one of the stacked panels being displayed at any given time.

To stack a panel on top of another panel just click the title bar and drag on top of the middle docking zone selector you want and then release.



Below is how the panels will be stacked together.

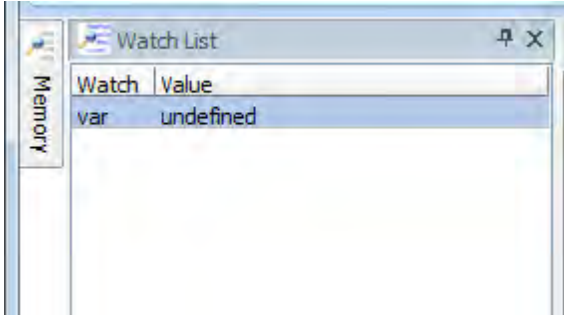


3.5.2.8.3 Pinning Panels

Another way to save display area aside from stacking panels together is by pinning them on the sides of your main window so that only a tab of that panel will appear. Then just hover your mouse over that tab to display the entire panel.

To pin a window to its side simply click on the pin icon on the title bar of that panel.

Below is an example of how pinning works. The Memory window is pinned to the side with only the tab being displayed. The entire panel will slide out if the mouse is hovered on top of the tab.



3.5.2.8.4 Hiding Panels

One last way to get more display area is to simply display the panels that you may need and then hiding those that you don't. This can be easily done just by clicking the close button on the right side of each panel or by unchecking them on the View tab group on the Tabbed Toolbar. Closing them will cause them to be hidden from view completely.

To display the hidden panel simply go to the View tab group on the toolbar and check the panel you want to be displayed.

3.5.3 Using VinIDE

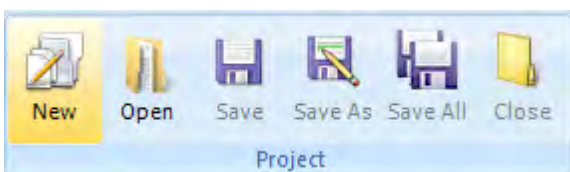
Starting the VinIDE from the Start Menu or the shortcut to the executable on the desktop will display a splash screen and the main window.

3.5.3.1 Project/File Handling

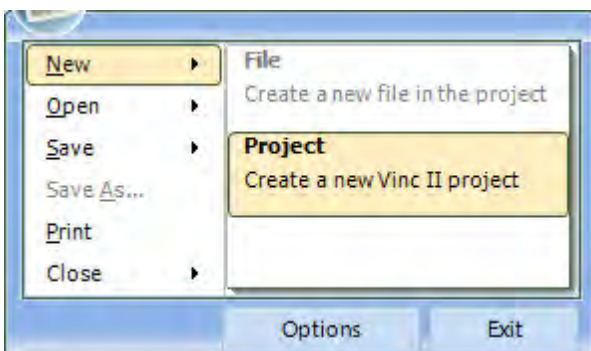
VinIDE project files have a ".vproj" extension. It supports C source files, assembler files, header files, object files as well as other files. The project files uses relative addressing.

3.5.3.1.1 Creating a new project

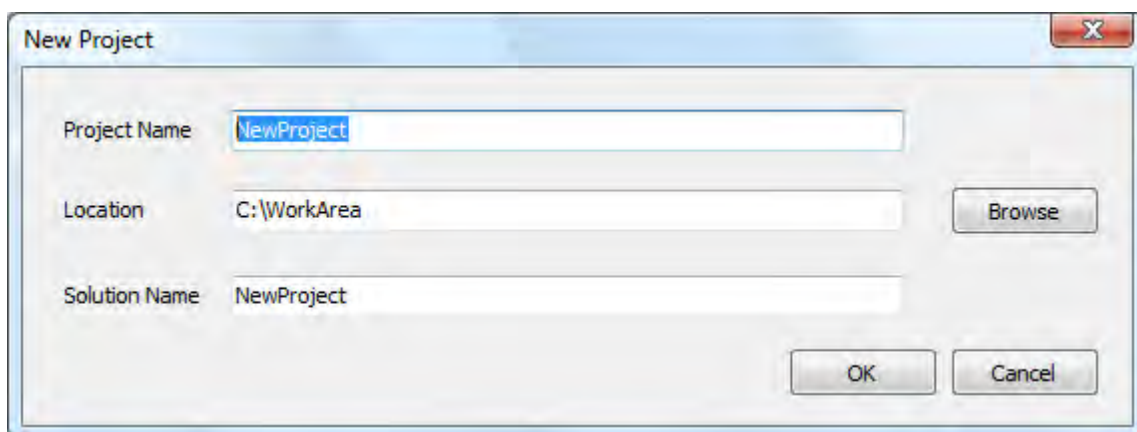
1. Click on New Project



or



The New Project window will appear.



2. Type the name of the project (the filename of the project file), the path where the project file will be saved, and the solution name.

The solution name will be the name of the output binary file when the project is built.

3. Select OK.

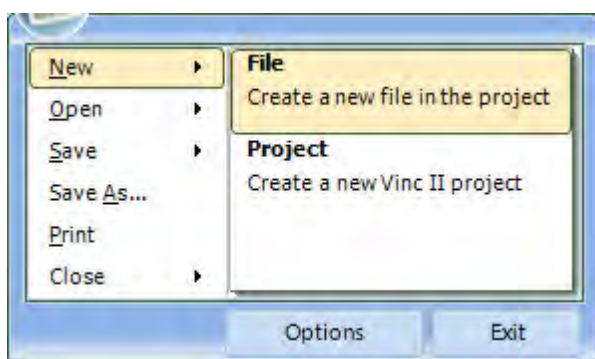
3.5.3.1.2 Adding files to your project

3.5.3.1.2.1 Adding new empty file

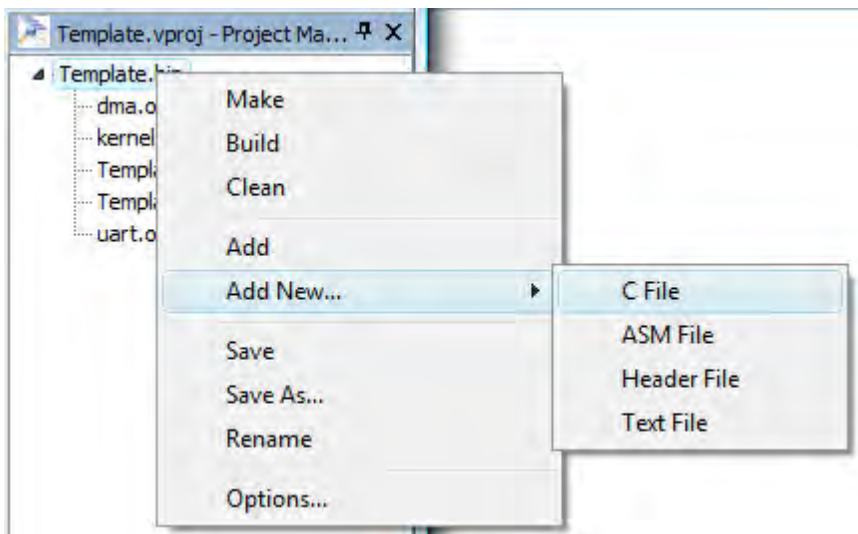
1. Click on New File



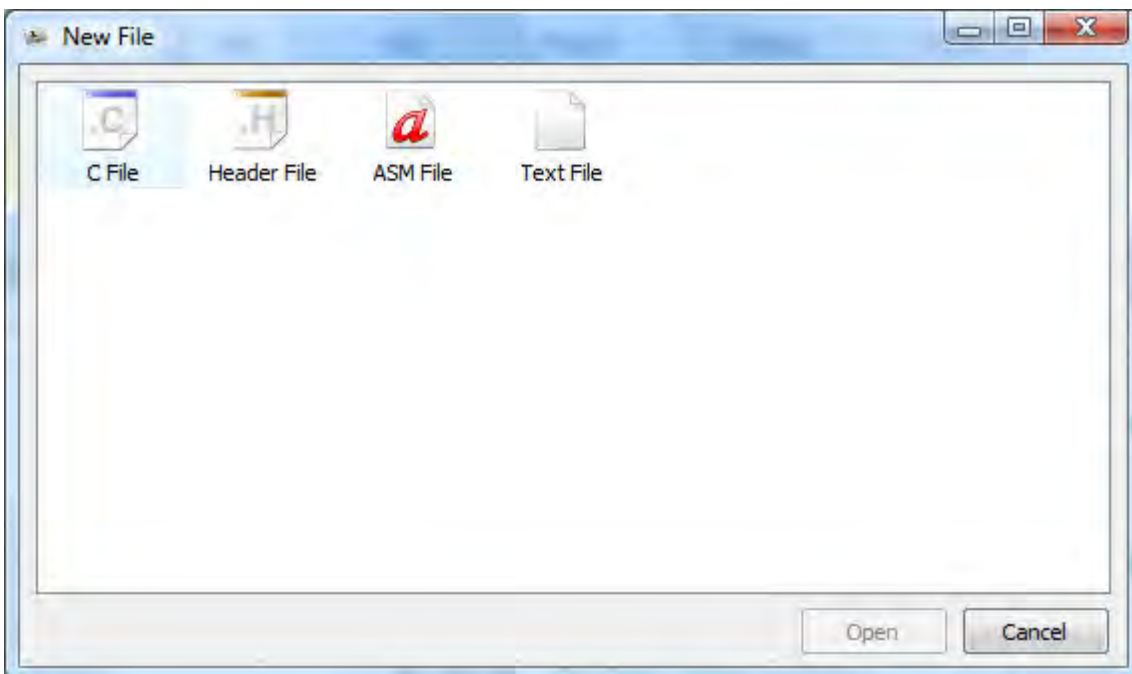
or



Or by right-clicking the project node on the Project Manager panel.



The Add New File window will appear



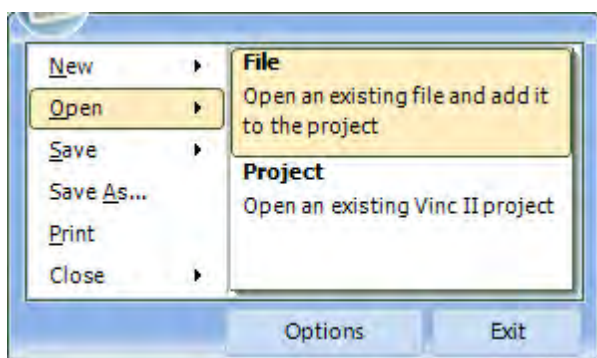
2. Select the type of file you want to add and press OK.

3.5.3.1.2.2 Adding existing files

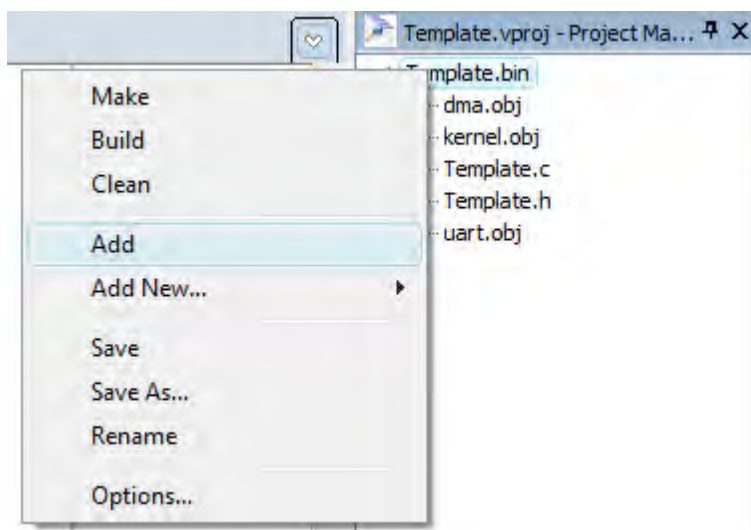
1. Click on Add File



or



or by right-clicking the project node on the Project Manager panel.



The open file dialog will appear. The open dialog have 6 filters to choose from (C, Assembly, Header, Object, Library, All file types). You can multi-select files by holding the CTRL key while clicking on the files you wish to add.

2. Press Open button.

3.5.3.1.3 Saving the project and files

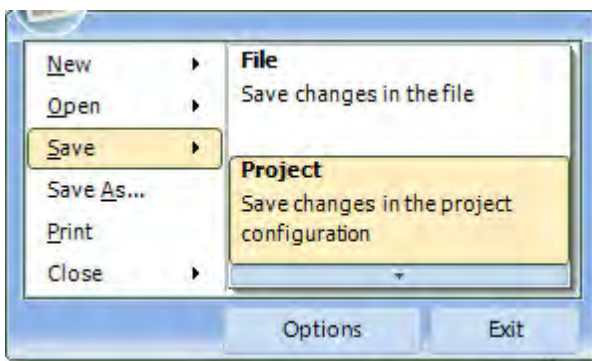
3.5.3.1.3.1 Saving the project

To save the project in its current filename :

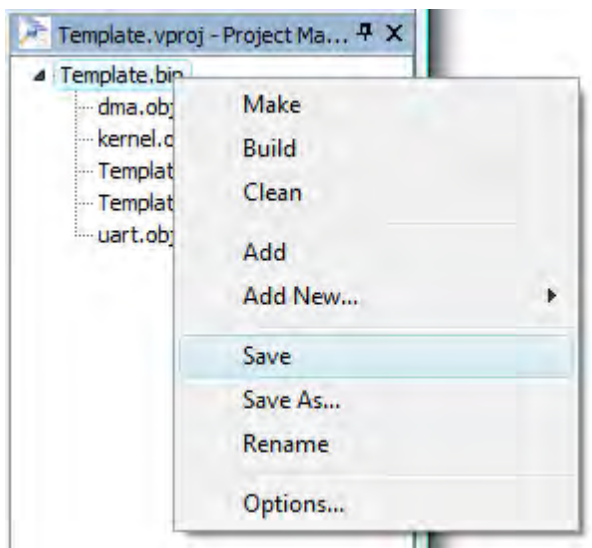
1. Click on Save Project



or



or by right-clicking the project node on the Project Manager panel

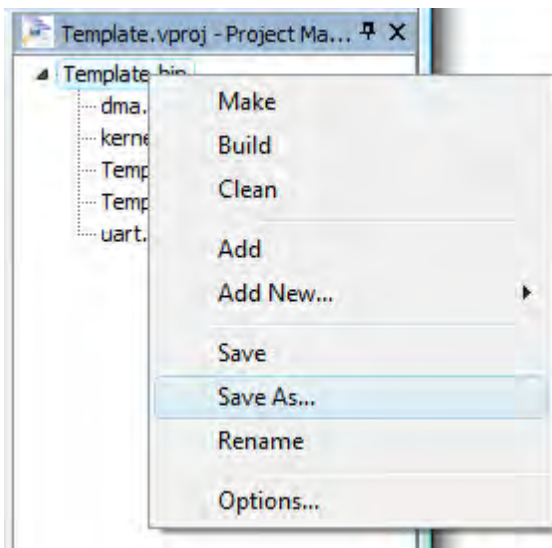


To save the project in another filename :

1. Click on Save As in the Project group



Or by right-clicking the project node on the Project Manager panel



3.5.3.1.3.2 Saving individual files

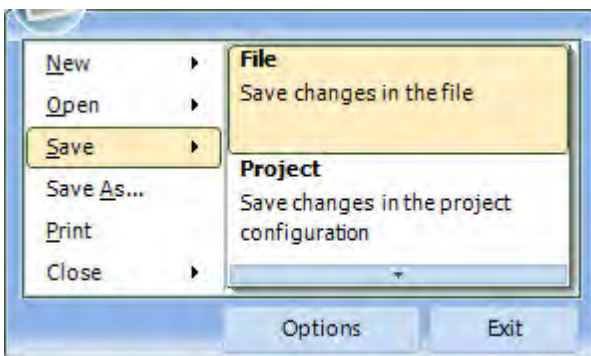
To save a file, it should first be opened in the editor.

To save the file in its current filename,

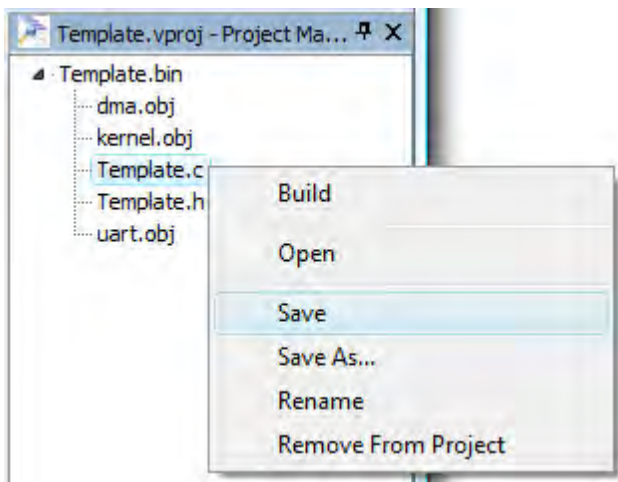
1. Click on Save File



or

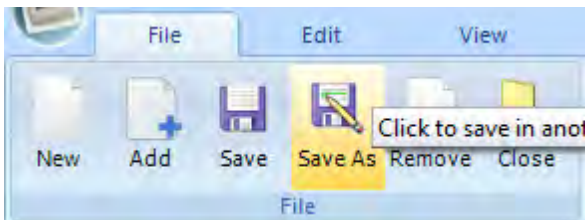


or by right-clicking the filename on the Project Manager panel

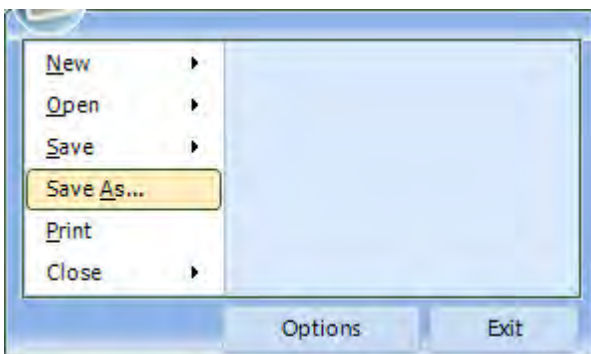


To save the file in another filename,

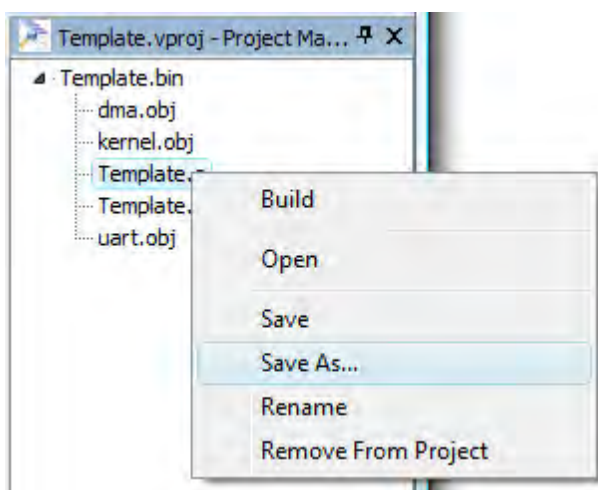
1. Click on Save As in the File group



or



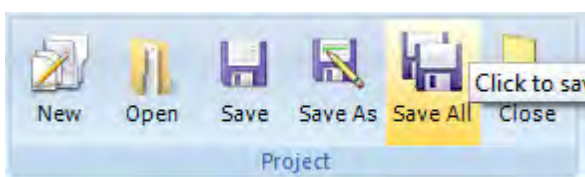
or by right-clicking the filename on the Project Manager panel



3.5.3.1.3.3 Saving the project and the files

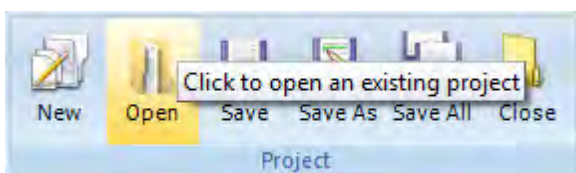
You can also save both the project and the files in the project with just one action

1. Click on Save All

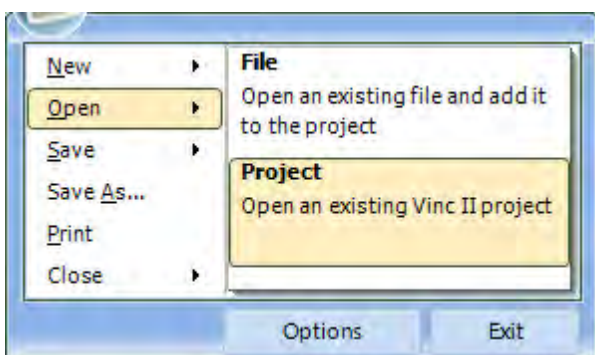


3.5.3.1.4 Opening an existing project

1. Click on Open Project



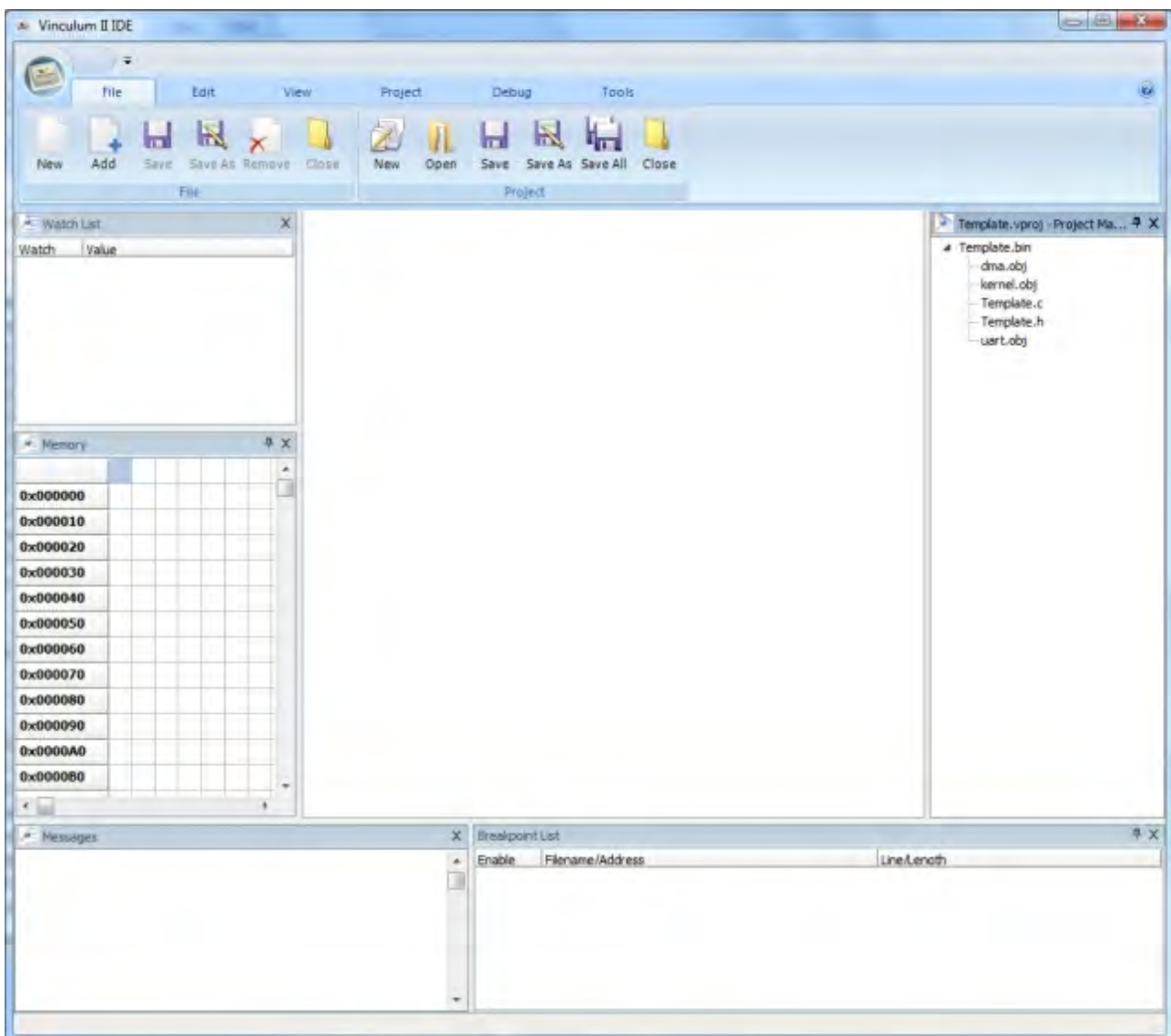
or



The open project dialog will appear.

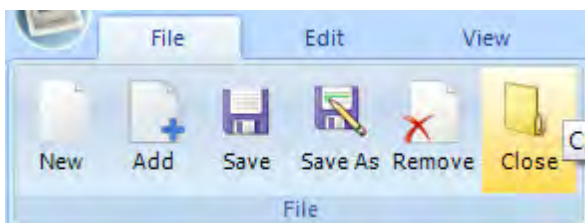
2. Select a project file (*.vproj) and press Open

The project as well as the files included in the project will be added into the Project Manager panel

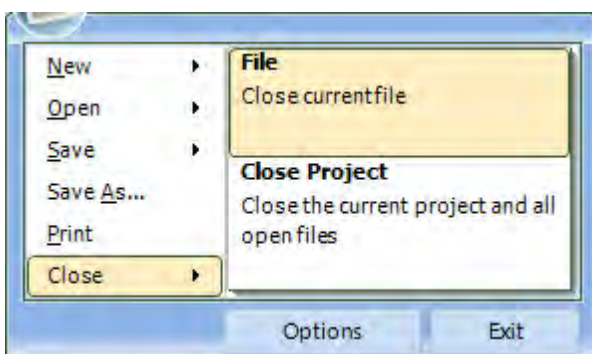


3.5.3.1.5 Closing a file

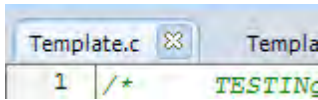
1. Click on Close File



or



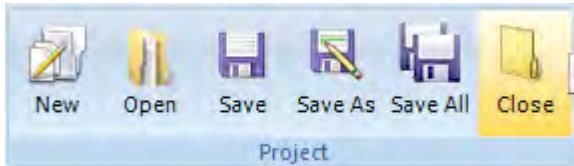
or by clicking the X button on the file's tab in the editor



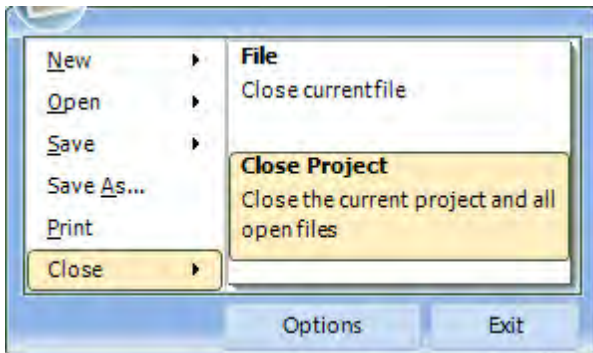
The file will be closed for editing but will remain as part of the project. To remove a file from the project, please see under Removing a file from the project

3.5.3.1.6 Closing a project

1. Click on Close Project



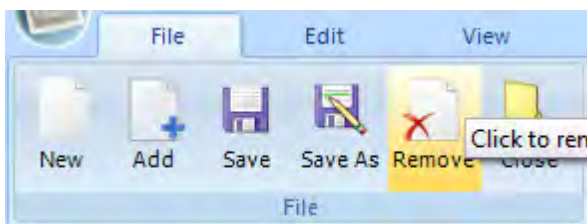
or



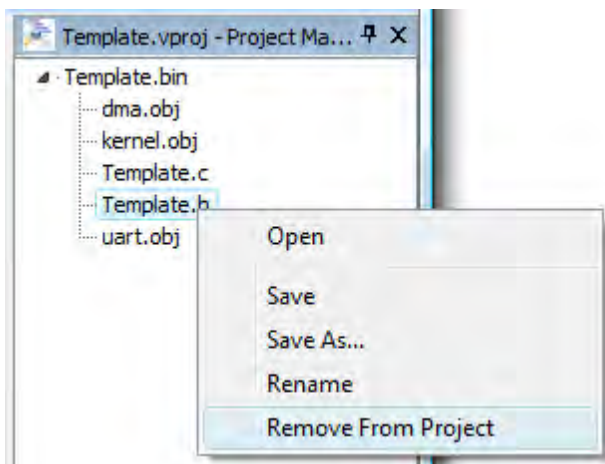
The user will be asked to save all unsaved files if any. All open files will be closed as well

3.5.3.1.7 Removing a file from the project

1. Click on Remove File



or by right-clicking the filename on the Project Manager panel



The file will be closed if open and will be removed from the list of files

3.5.3.2 Building a project

Building a project involves compiling, assembling, and linking the source files into the output binary and ROM files. The object files as well as the final executable and ROM file is saved in either a sub-folder called "debug" or "release" in the folder where the project file is stored. These folders are automatically created by the IDE when you create a project.

3.5.3.2.1 What you need to do before you can build your project?

Before you can build your project:

1. Make sure you have the latest executable files for the other modules of the toolchain, namely:

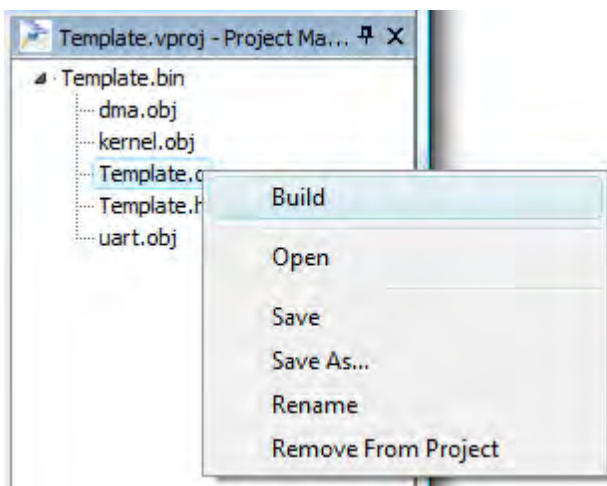
- The Preprocessor (VinCpp.exe)
- The Compiler (VinC.exe)
- The Assembler (VinAsm.exe)
- The Linker (VinL.exe)

2. Make sure the paths of the above executables is declared either by adding them to the path environment variable or by explicitly declaring them in the IDE's options module (see IDE options).

The path environment variable will be automatically updated by the supplied installer for the VNC2 IDE.

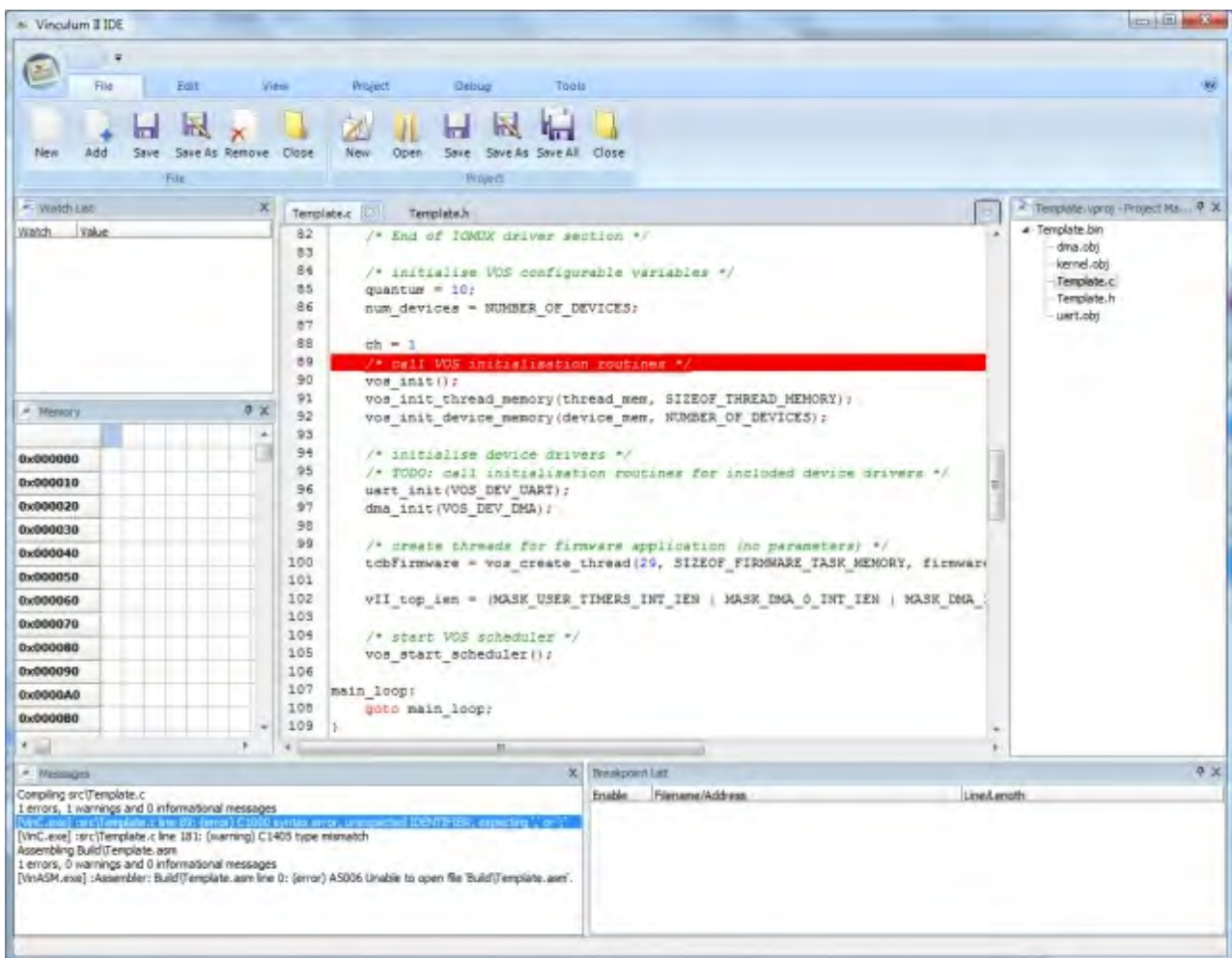
3.5.3.2.2 Compiling a single source file

1. Right-click the file in the Project Manager panel.



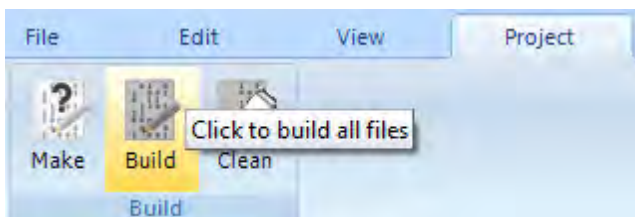
2. If there are any compile errors, these errors appear below in the Messages panel.

Double click the compile errors to highlight the line number of the file where the error is generated.

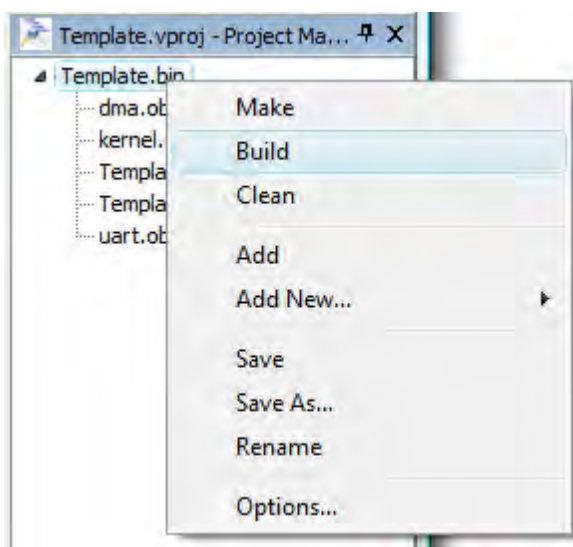


3.5.3.2.3 Compiling the project

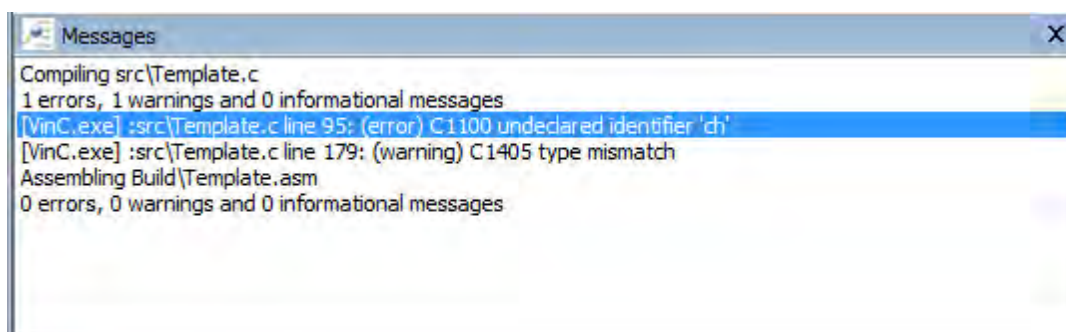
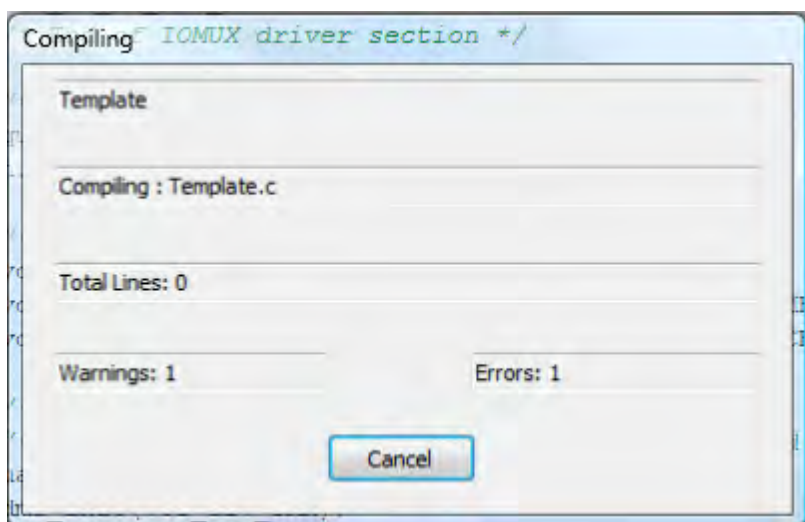
1. Click on Build



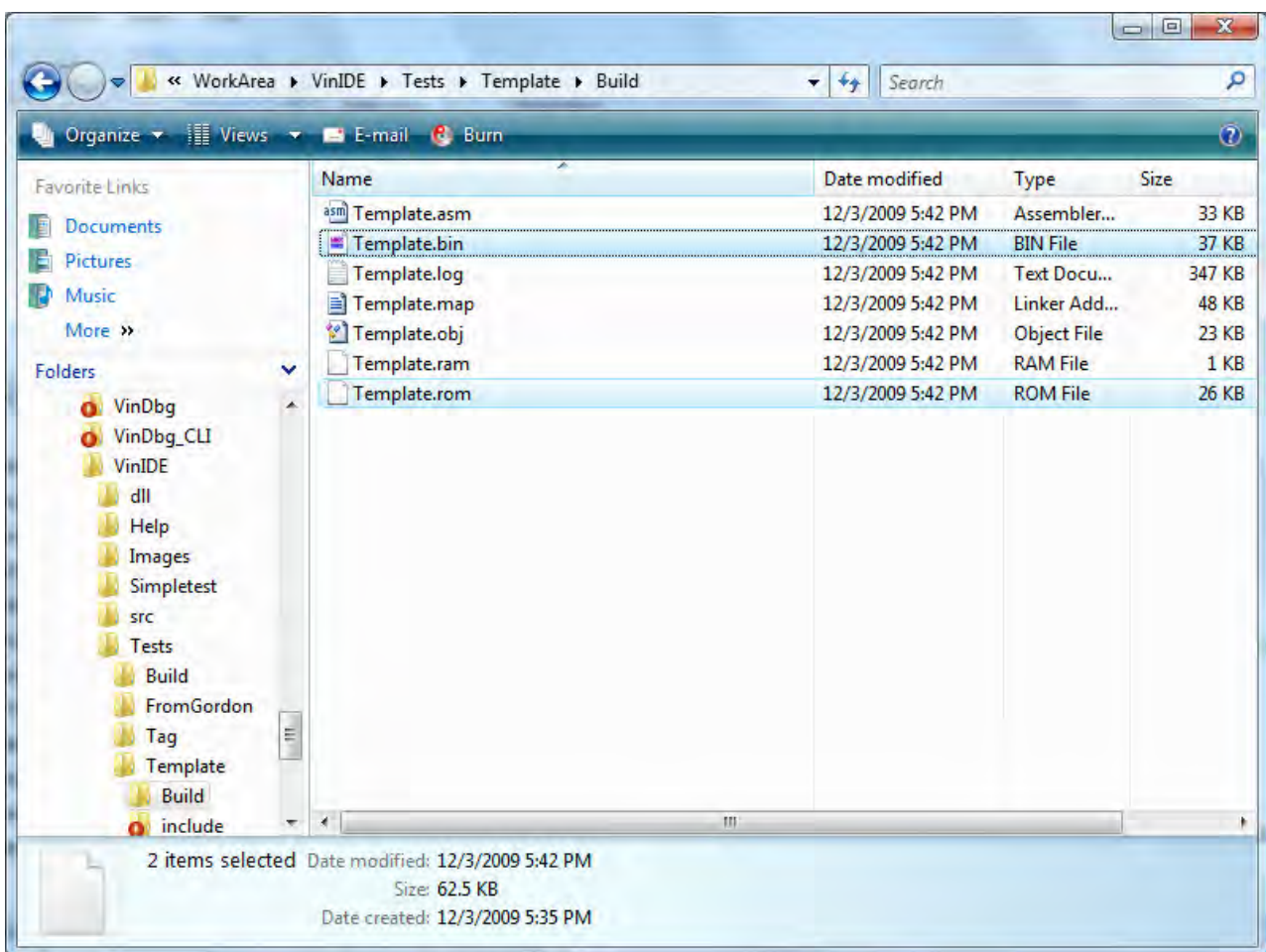
Or right-click the project node in the Project Manager panel



If there are any compile errors, these errors appear below in the Messages panel



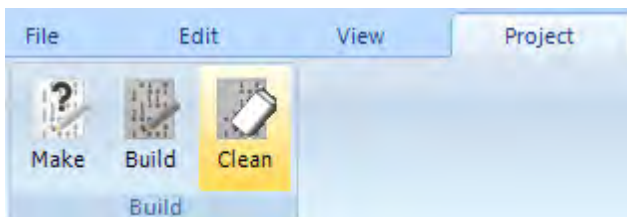
2. If the build is successful, the output binary file as well as the object files are written on the project file's build folder



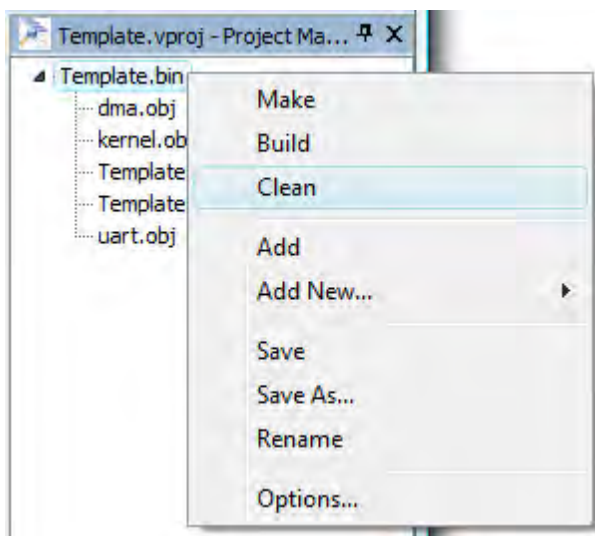
3.5.3.2.4 Cleaning the build files

You can clean the build folder to remove the generated files.

1. Click on Clean



or



3.5.3.3 Debugging a project

The IDE also includes debugging functions to allow the user to debug their projects easily using the user interface.

3.5.3.3.1 What you need to be able to debug using the IDE?

Before you can start debugging:

1. Make sure you have these files to be able to use the debugging functionalities:

- The debugger (WinDbg.exe).
- The WinDbgLib.dll library.

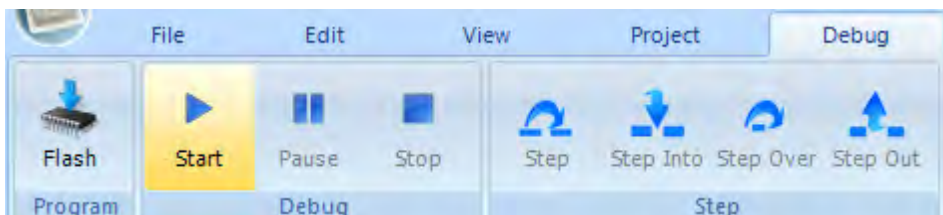
2. Make sure the paths of the above executables is declared either by adding them to the path environment variable or by explicitly declaring them in the IDE's options module (see IDE options).

3. Also make sure that the evaluation board is connected and turned on.

4. Make sure that the debug option for Compiler, Assembler and linker are set to 1 or set the Build Configuration option to debug. (Click here to see [Build Configuration Option](#))

The files required are installed by default by the installer program

3.5.3.3.2 Debugging Commands



Below are some of the debugging commands that you can do using the IDE :

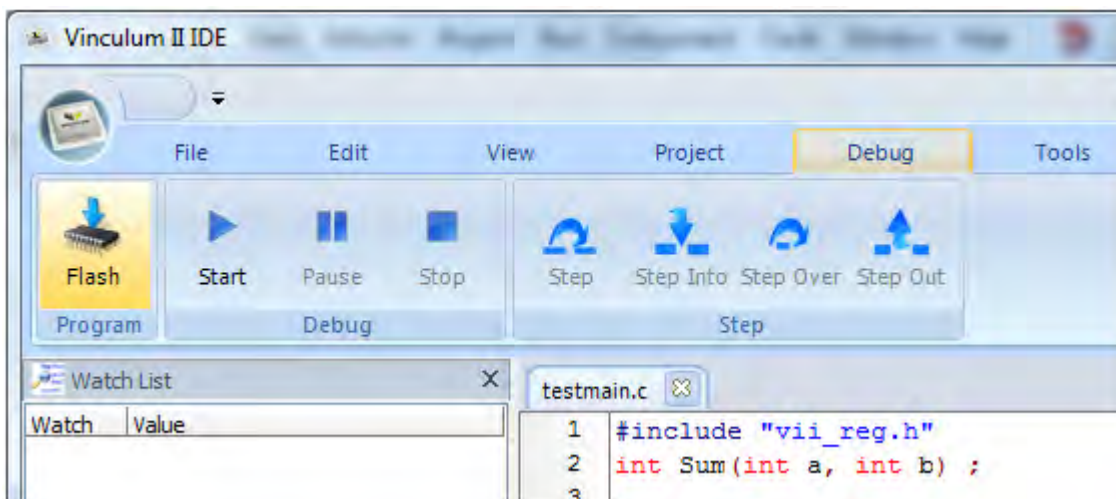
- Flash
 - Used to program the board without starting the debugger tool
- Start
 - Used to start/continue execution of the program.
- Pause
 - Used to halt the execution of the program.
- Stop
 - Used to stop execution of the program.

- Step
 - Used to do a line by line execution of the program (must halt execution first before doing a step).
- Step Into
 - If the current execution line is a function call, Step Into will go inside that function and execute the first line inside (must halt execution first before doing a step).
- Step Over
 - If the current execution line is a function call, Step Over will execute all the lines inside the function (must halt execution first before doing a step).
- Step Out
 - Step Out will execute all the executable lines in the current function until the first line after (must halt execution first before doing a step).

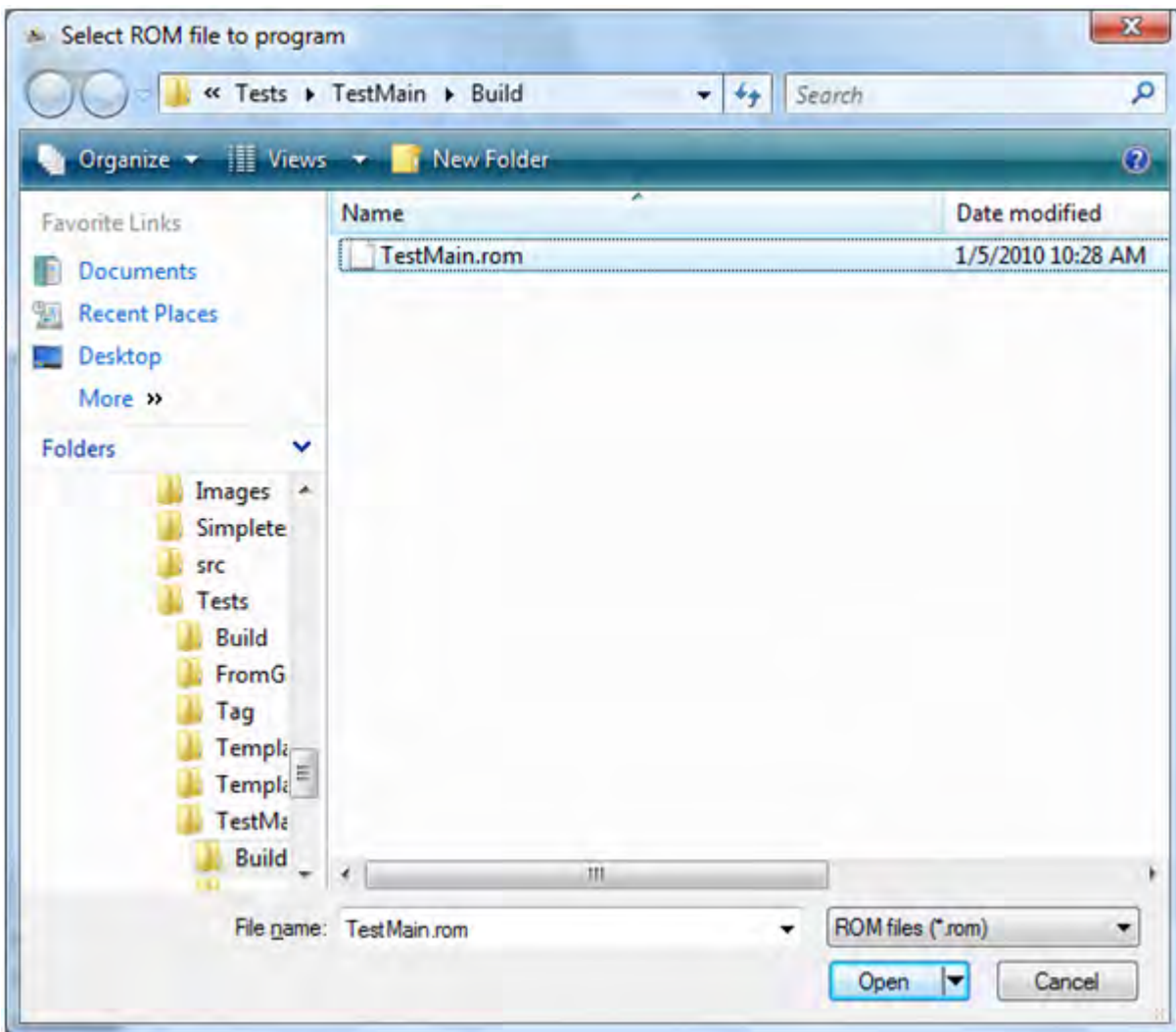
3.5.3.3.2.1 Programming the chip

The VNC2 chip can be programmed with the ROM file without starting the source-level debugging. To do this :

- a) Under the Debug tab, click on Flash



- b) Select a ROM file then click the Open button



c) Click the OK button.

Note: Make sure to flash the chip with the Rom file (.rom) of the active project. If the flashed rom file is different from the active project, the code cannot be debugged.

3.5.3.3.3 Adding/Removing Breakpoints

Breakpoints are used to interrupt and halt execution of the program for debugging purposes.

To add a breakpoint, click on the line number in the gutter part of the source editor corresponding to the instruction you want the program to halt.

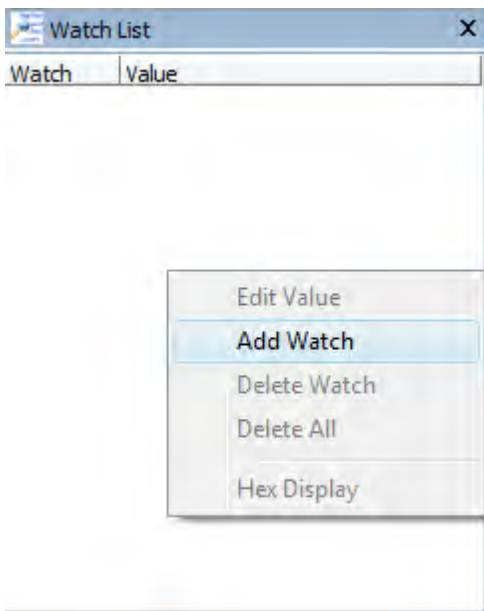
```

73    vII_gpio_cntrl_porta_1 = 0xff;    // 0x181
74    // configure port
75    vII_io_op_sel_1 = 0x0d;    // 0x42
76    vII_io_op_src_1 = 0x29;    // 0x41
77    vII_io_op_src_1 = 0x2a;    // 0x41
78    vII_io_op_sel_1 = 0x0e;    // 0x42
  
```

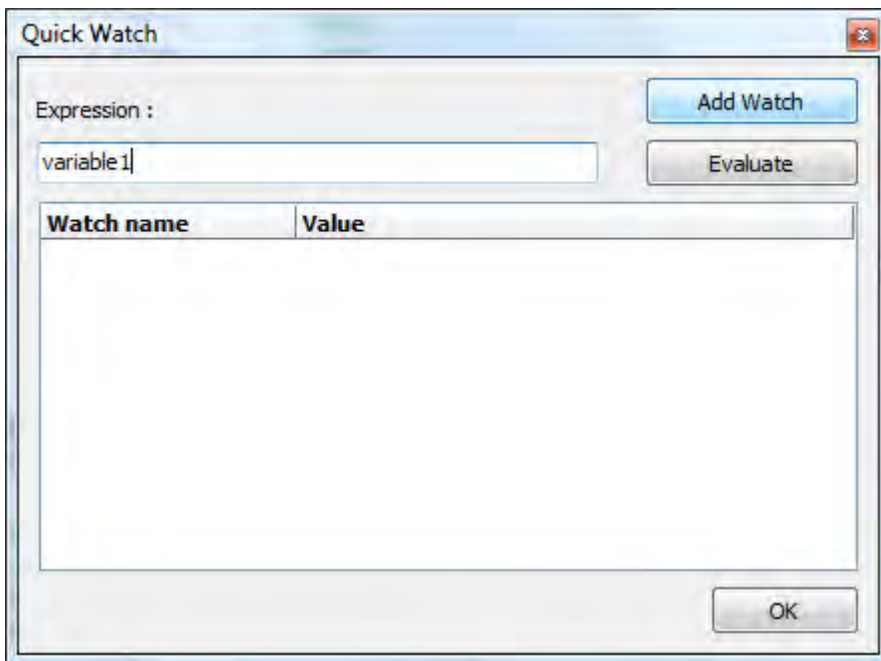
To remove the breakpoint, click on the line number again.

3.5.3.3.4 Adding Watch variable

The watch window lists the variables that are being evaluated during the debugging process. These variables are updated after the program has paused execution. Right-click on the Watch window to add a watch variable.



The Quick Watch window should appear.



Enter the name of the variable you would like to evaluate and then click "Add Watch". The new watch item will be added in the Watch window.

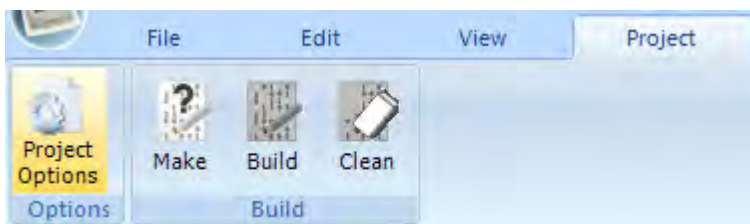


3.5.3.4 Project Options

The Project Options module lets the user change the behavior of the various modules of the toolchain and even the build process itself.

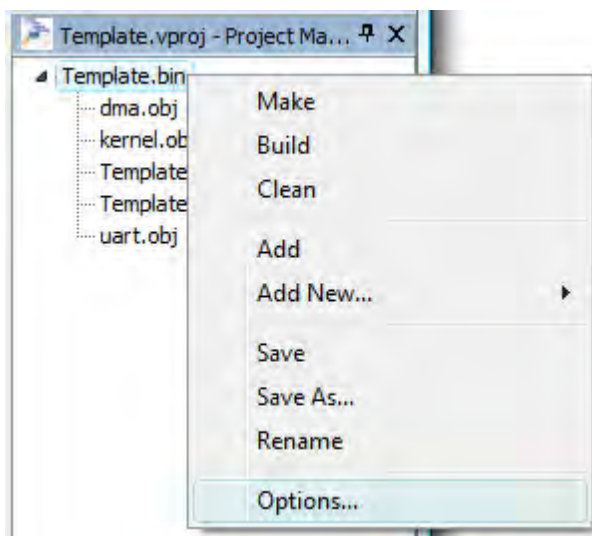
3.5.3.4.1 Bringing up the Project Options window

1. Go to the Project toolbar tab and click Project Options.



or

Right click the project node in the Project Manager panel.

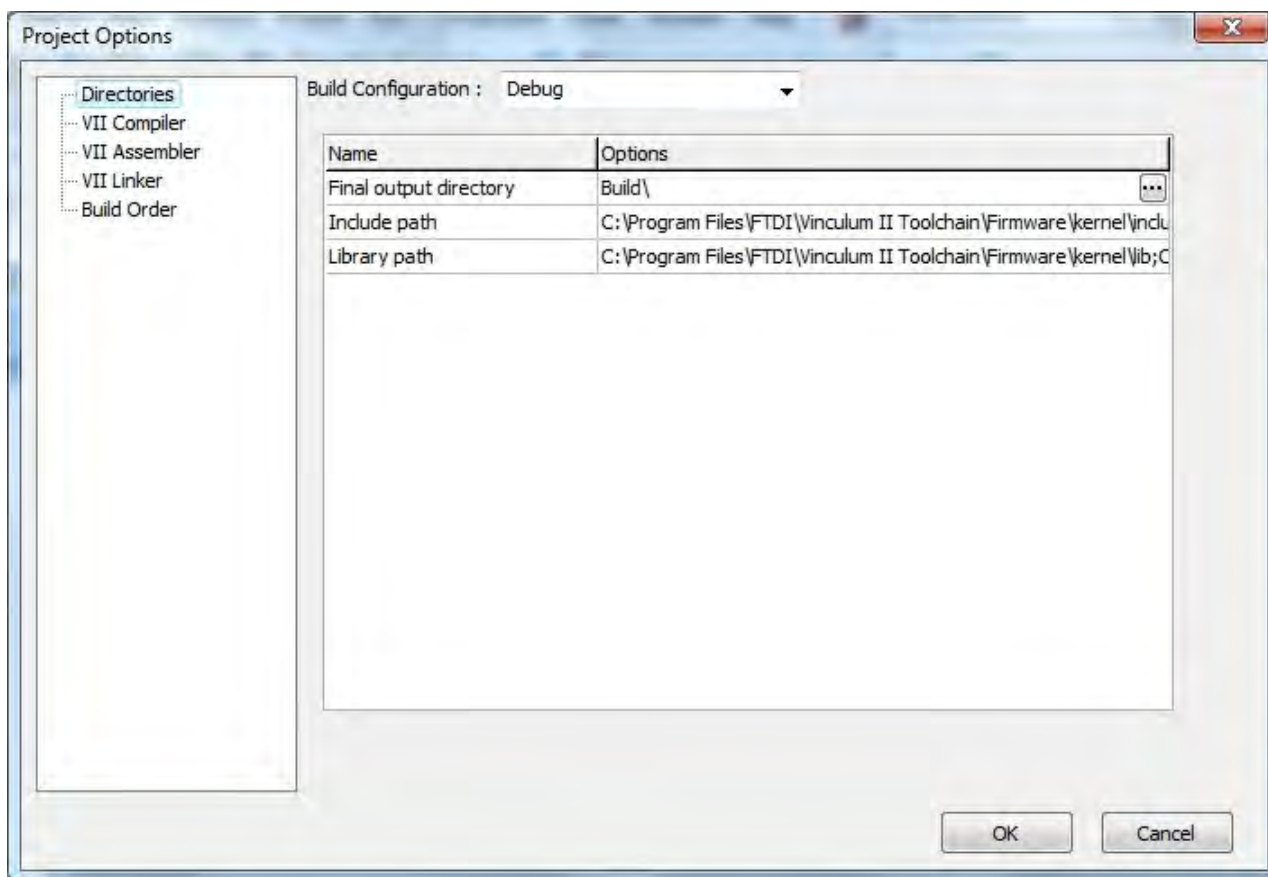


The Project Options window will appear.

Changes made in the project options window are saved when you click OK, otherwise changes are discarded.

3.5.3.4.2 The Directories Options

The Directories options allows the user to change some path-related options like the include and library paths as well as where the final binary is saved. To view the Directories options In the Project Options, click the Directories node in the left hand list.



➤ Final output directory

The directory path where the binary output of the build will be saved. If no path is specified, the default location will be inside the Build folder of the current project

➤ Include path

A list of directories where the other tools (i.e. Compiler, Assembler) will look into for included files in the project

➤ Library path

A list of directories where the other tools (i.e. Compiler, Assembler) will look into for library files that are needed for the project

3.5.3.4.2.1 Changing where the final output is stored

The default path where the final output is stored is in the build folder alongside the project file. To change the path where you want to save the binary output :

1. Go to the Project Options module
2. Click on the Directories node on the options tree view
3. Type the path where you want to save the output in the Final output directory textbox (absolute or relative path)



4. Alternatively, you can click on the button with the "..." beside the textbox to bring up the folder selection dialog box. Select the folder and click OK in the dialog box.

5. Click the OK button to save the changes in the options.

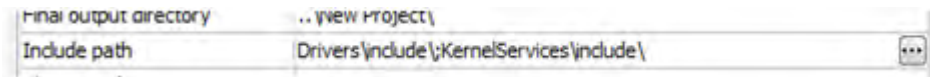
3.5.3.4.2.2 Adding directories in the Include path option

You can add multiple directories, separated by a semicolon, to the include path

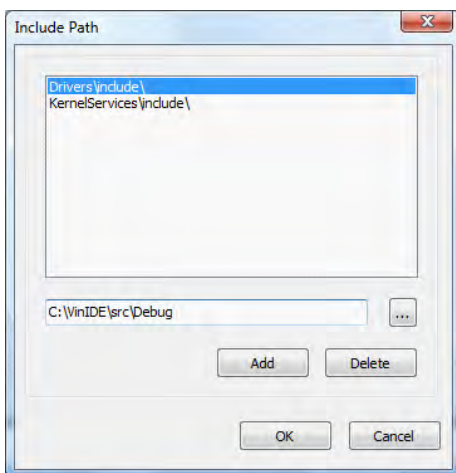
1. Go to the Project Options module

2. Click on the Directories node on the options tree view

3. Type the paths, separated by a semicolon, in the Include path textbox (absolute or relative path)



4. Alternatively, you can click on the button with the "..." beside the textbox to bring up the multi-folder selection dialog box.



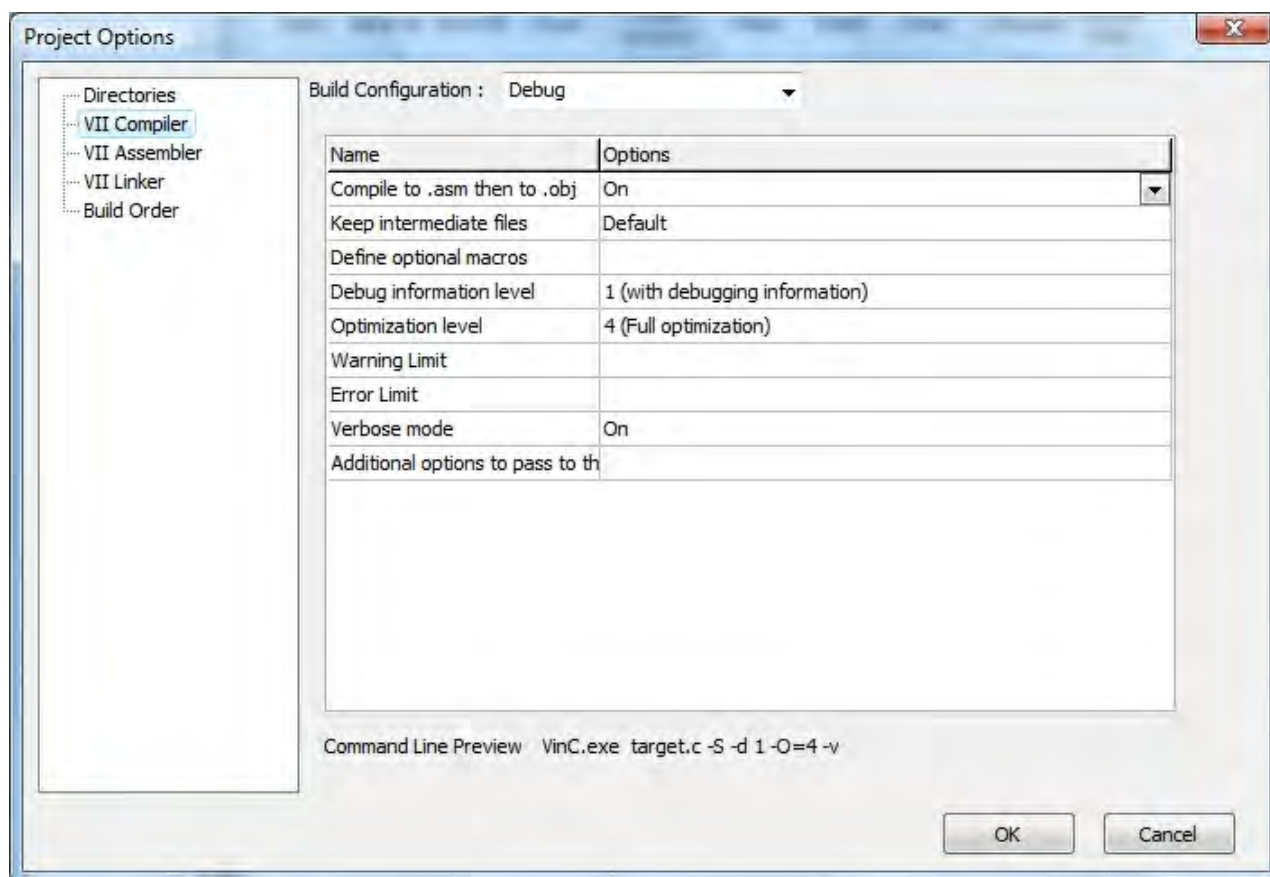
5. Click the OK button to save the changes in the options.

3.5.3.4.2.3 Adding directories in the Library paths option

To add multiple directories to the Library path option, do the same as with the Include path option

3.5.3.4.3 The Compiler Options

The Compiler options allows the user to change some compiler-related options and arguments. To view the Compiler options In the Project Options, click the VNC2 Compiler node in the left hand list.



Compile to .asm then to .obj

On – The compiler is invoked with the –S switch. The compiler stops processing after preprocessing and compilation. The assembler is invoked in another process.

Off – the compiler is invoked with the –c switch. Preprocess, compile and assemble in one call

Keep intermediate files

Default – Use the compiler's default setting.

Off – Intermediate files are deleted after the compiler execution.

Assembly files – The compiler is invoked with the "--save-temp a" switch. The compiler keeps the .asm files it generated before calling the assembler.

Preprocessor files – The compiler is invoked with the "--save-temp i" switch. The compiler keeps the preprocessor files after the call to the preprocessor.

Both – The compiler is invoked with the "--save-temps" switch. Both assembler and preprocessor files are kept.

Define additional macros

User macro definitions that are to be used for compilation are entered here

Debug information level

Default – Use the compiler's default debug information level setting.

0 – No debugging information is included in the compiler output.

1 – Debugging information is included in the compiler output.

Optimization level

Default – Use the compiler's default setting.

0 – No optimizations.

1 – Register Allocation optimization is invoked.

- 2 – Register Allocation + Partial IC optimization is invoked.
- 3 – Register Allocation + Full IC optimization is invoked.
- 4 – Full optimization (RA + Full IC + Peephole) is invoked.

Warning Limit

The limit of the warning messages to be issued by the compiler.

Error Limit

The limit of the error messages to be issued by the compiler.

Verbose mode

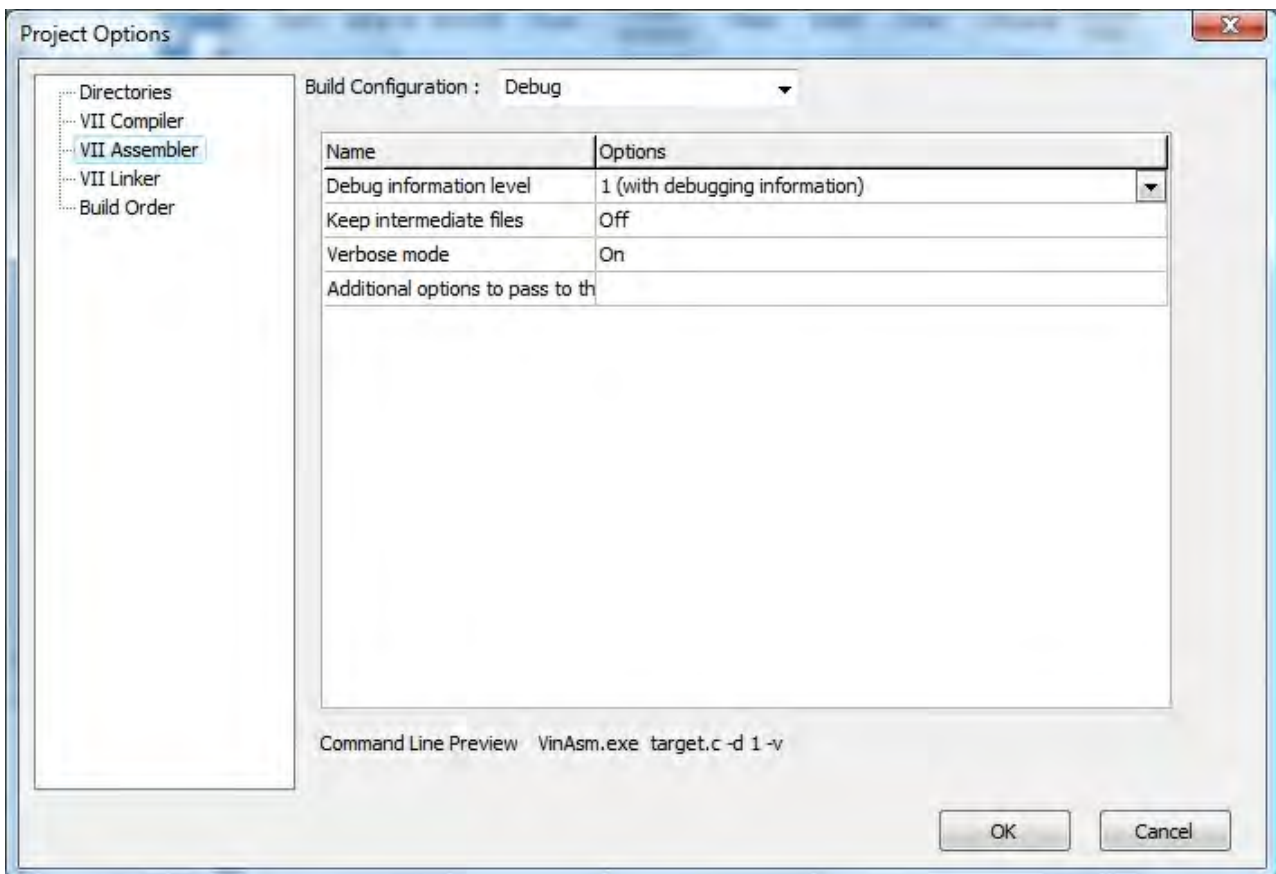
- On – The compiler is in verbose mode.
- Off – Quieten output of compiler.

Additional options to pass to the compiler

Any additional switches that have to be passed to the compiler are entered here.

3.5.3.4.4 The Assembler Options

The Assembler options allows the user to change some assembler-related options and arguments. To view the Assembler options In the Project Options, click the VNC2 Assembler node in the left hand list.



Debug information level

- Default – Use the assembler's default debug information level setting.
- 0 – No debugging information is included in the assembler output.
- 1 – Debugging information is included in the assembler output.

Keep intermediate files

This is not yet implemented.

Verbose mode

On – The assembler is in verbose mode.

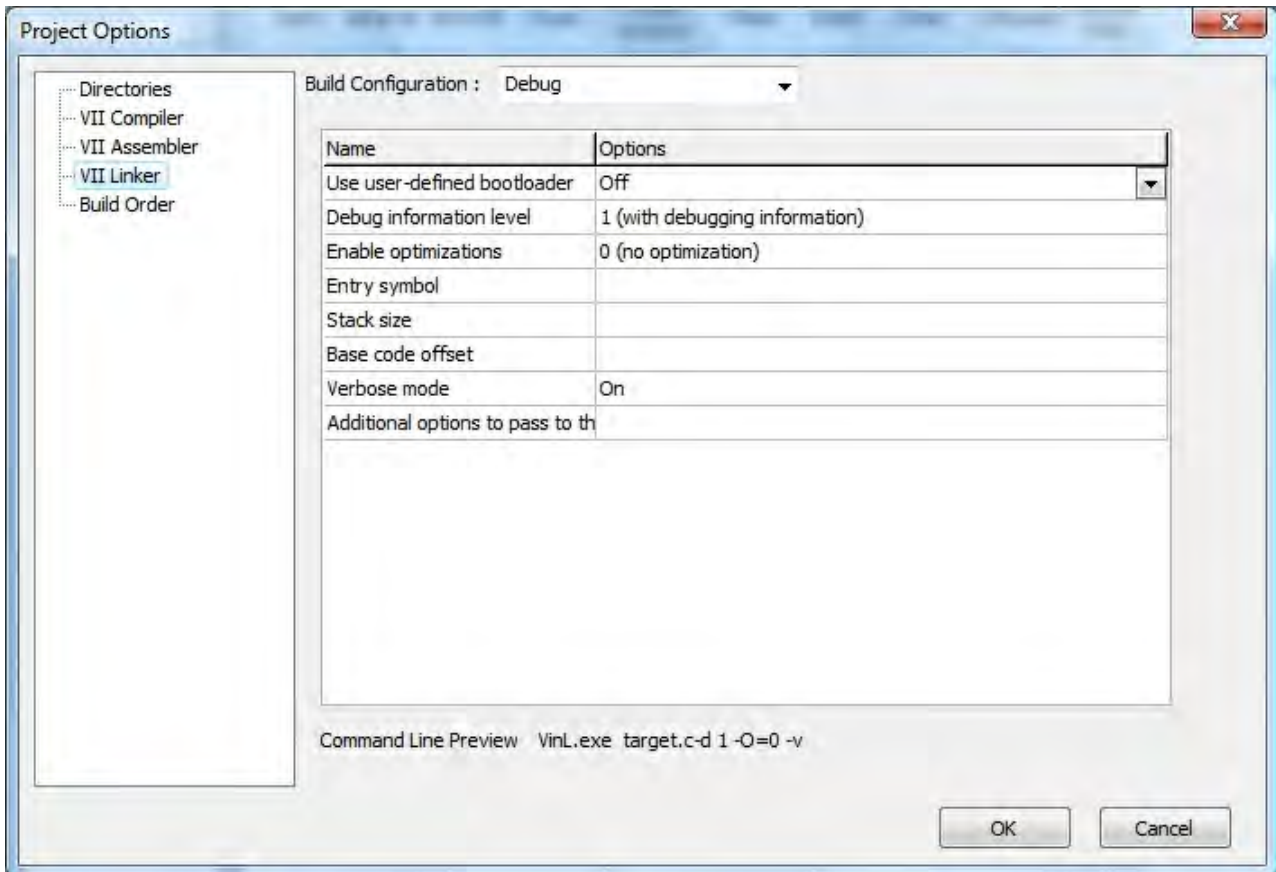
Off – Quieten output of assembler.

Additional options to pass to the assembler

Any additional switches that have to be passed to the assembler are entered here.

3.5.3.4.5 The Linker Options

The Linker options allows the user to change some linker-related options and arguments. To view the linker options In the Project Options, click the VNC2 Linker node in the left hand list.



Use user-defined bootloader

On – The linker will not link the firmware bootloader and use a user-supplied bootloader instead

Off – The firmware bootloader will be automatically used.

Debug information level

Default – Use the linker's default debug information level setting.

0 – No debugging information is included in the linker output.

1 – Debugging information is included in the linker output.

Enable Optimizations

Default – Use the linker's default setting.

On – Linker optimization is invoked.

Off – Linker optimization is not invoked.

Entry Symbol

Entry symbol name to be used by the linker.

Stack Size

The size in bytes of the stack to be used by the linker.

Base Code Offset

The base code offset value to be used by the linker.

Verbose mode

On – The linker is in verbose mode.

Off – Quieten output of linker.

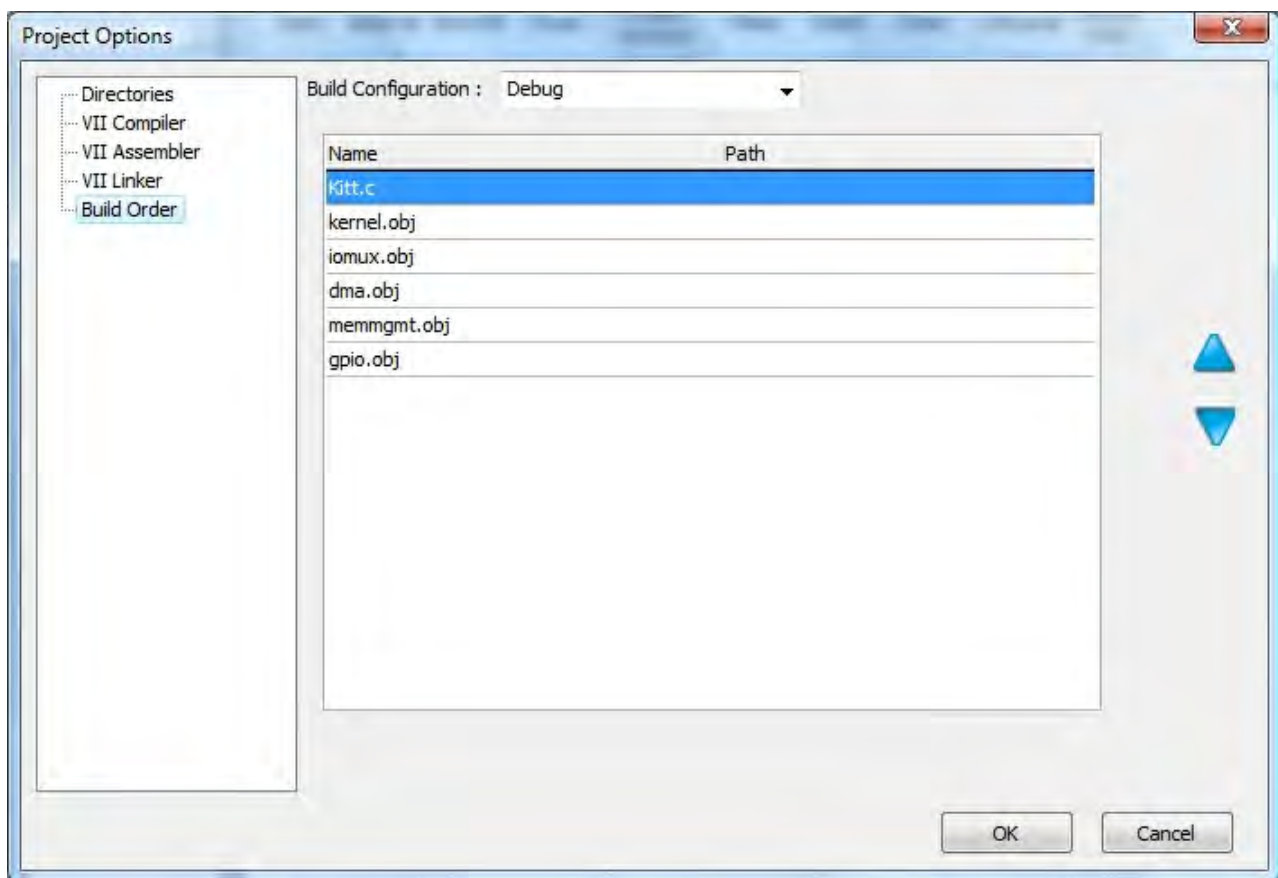
Additional options to pass to the linker

Any additional switches that have to be passed to the linker are entered here.

3.5.3.4.6 The Build Order

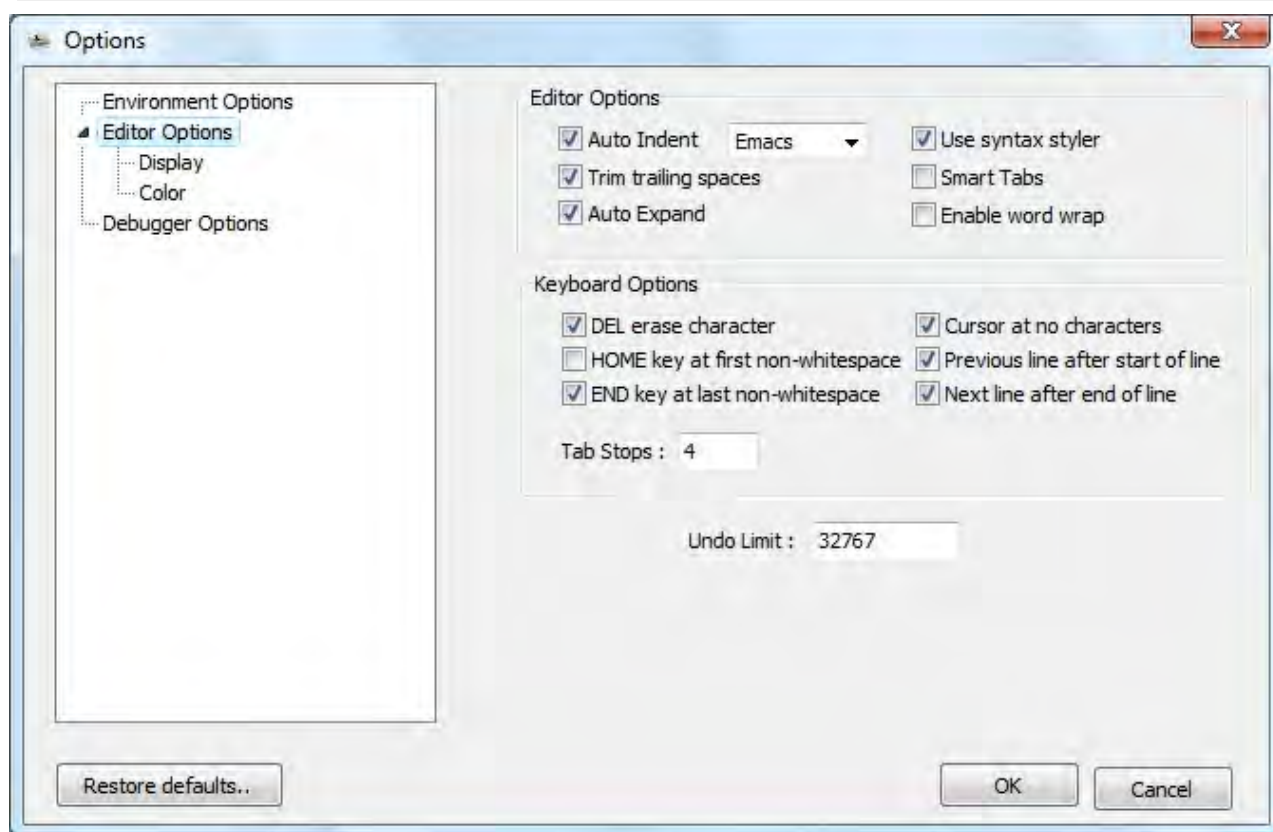
The Build Order options allows the user to specify the order on which the object files are linked. The source files and included object files in the project are listed and the order can be changed through the UP/DOWN arrow buttons.

It is recommended that the kernel.a file is placed first.



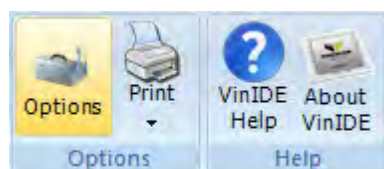
3.5.3.5 The IDE Options

The IDE options window allows the user to change the behavior and appearance of the IDE including the built in text editor. The user can customize the editor's display and colours.

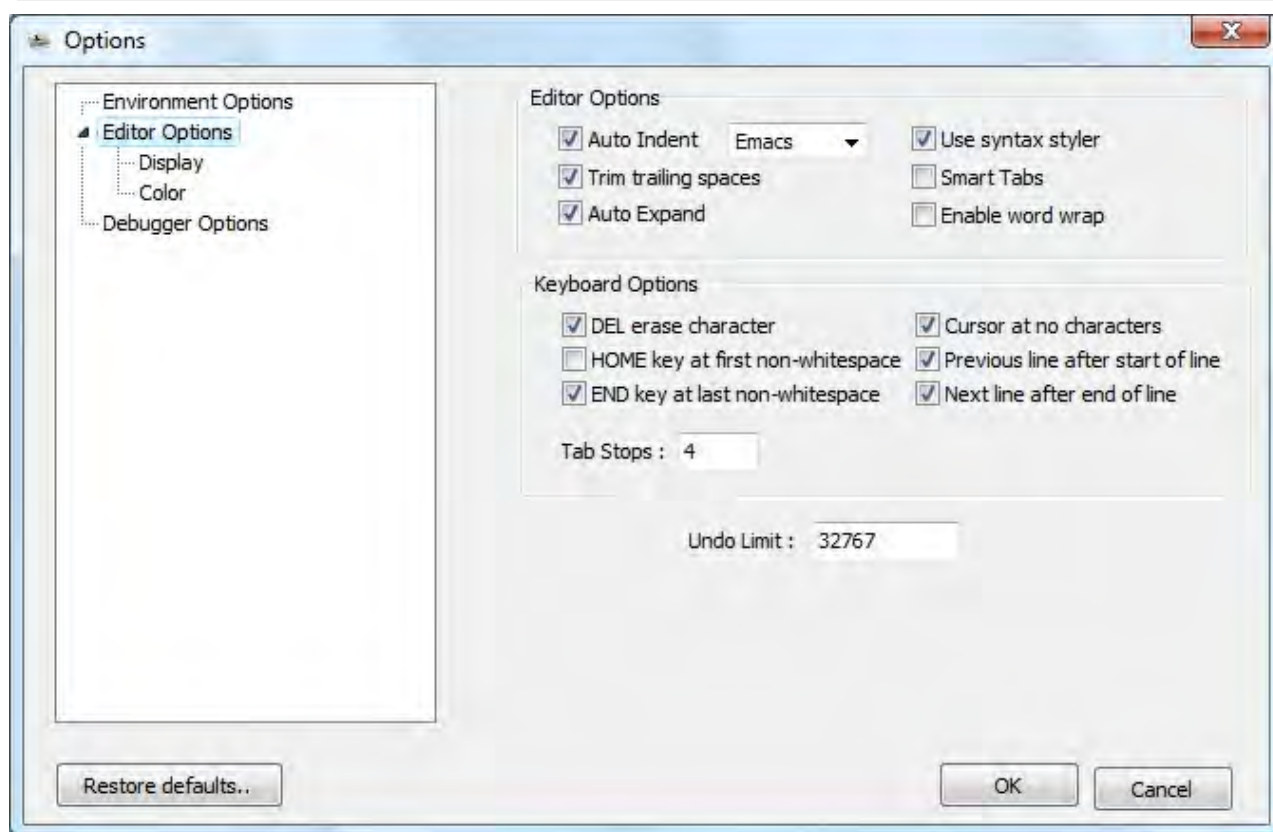


3.5.3.5.1 Bringing up the IDE Options window

1. Go to the Tools toolbar tab and click Options



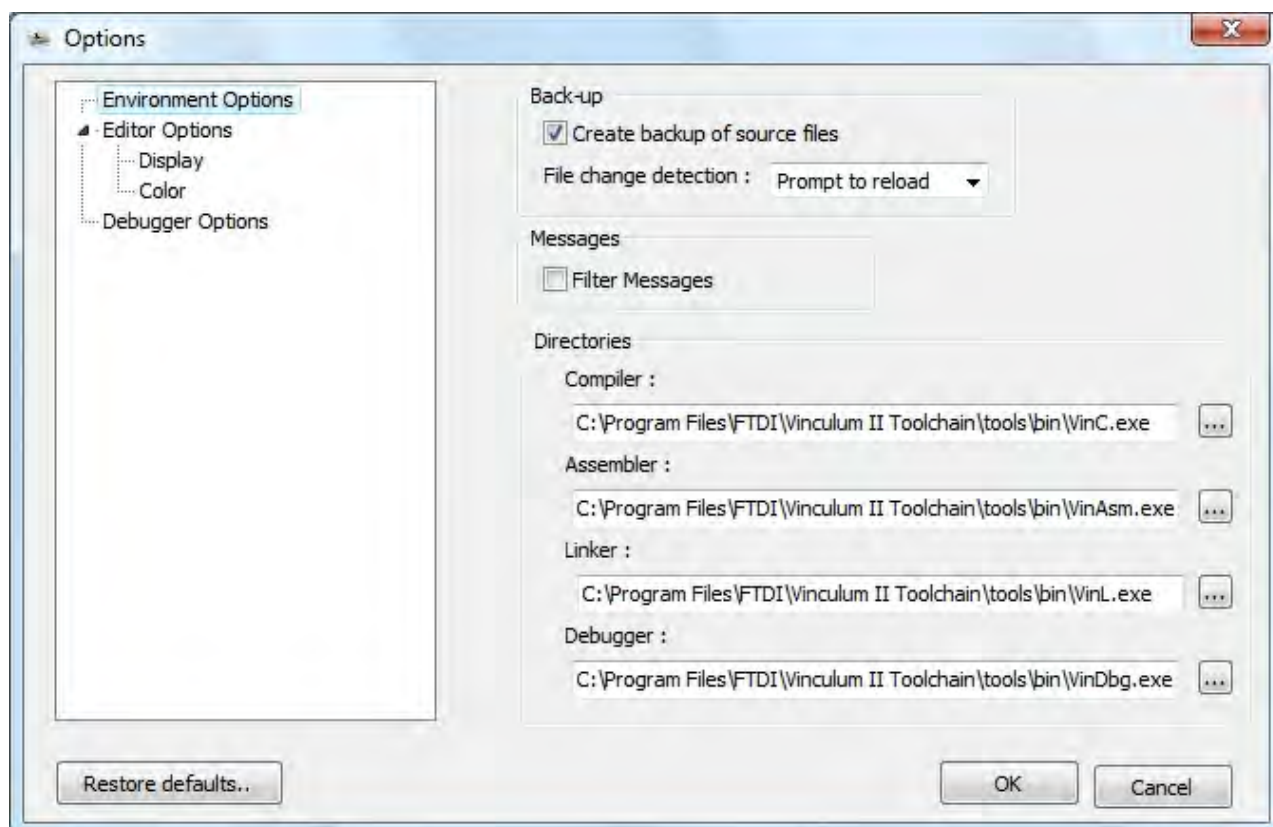
The IDE Options window should appear



NOTE : Some of the features in the IDE options are not yet implemented.

3.5.3.5.2 Environment Options

The Environment Options section lets the user configure some of the general aspects of the IDE like the directories of the tools to be used, the backing up of the files before editing, and the filtering behavior of the Message window.



Create backup of source files

If checked, the IDE creates a backup file (.bak) of the file before being opened in the editor

File Change Detection

Sets the behavior of the IDE if a file is changed outside of the editor

No Action – Disregard the changes.

Prompt to reload – A message box informing the user that the file has changed and asks if to reload the file with the changes.

Reload automatically – The file is automatically reloaded with the changes disregarding the modifications in the editor

Filter Messages

If checked, the Messages windows only shows the messages concerning errors and warnings. If unchecked, all messages from the tools are shown.

Compiler Directory

Specifies the directory where the compiler tool is found. If there is no path specified, the IDE uses the path environment variable for the path.

Assembler Directory

Specifies the directory where the assembler tool is found. If there is no path specified, the IDE uses the path environment variable for the path.

Linker Directory

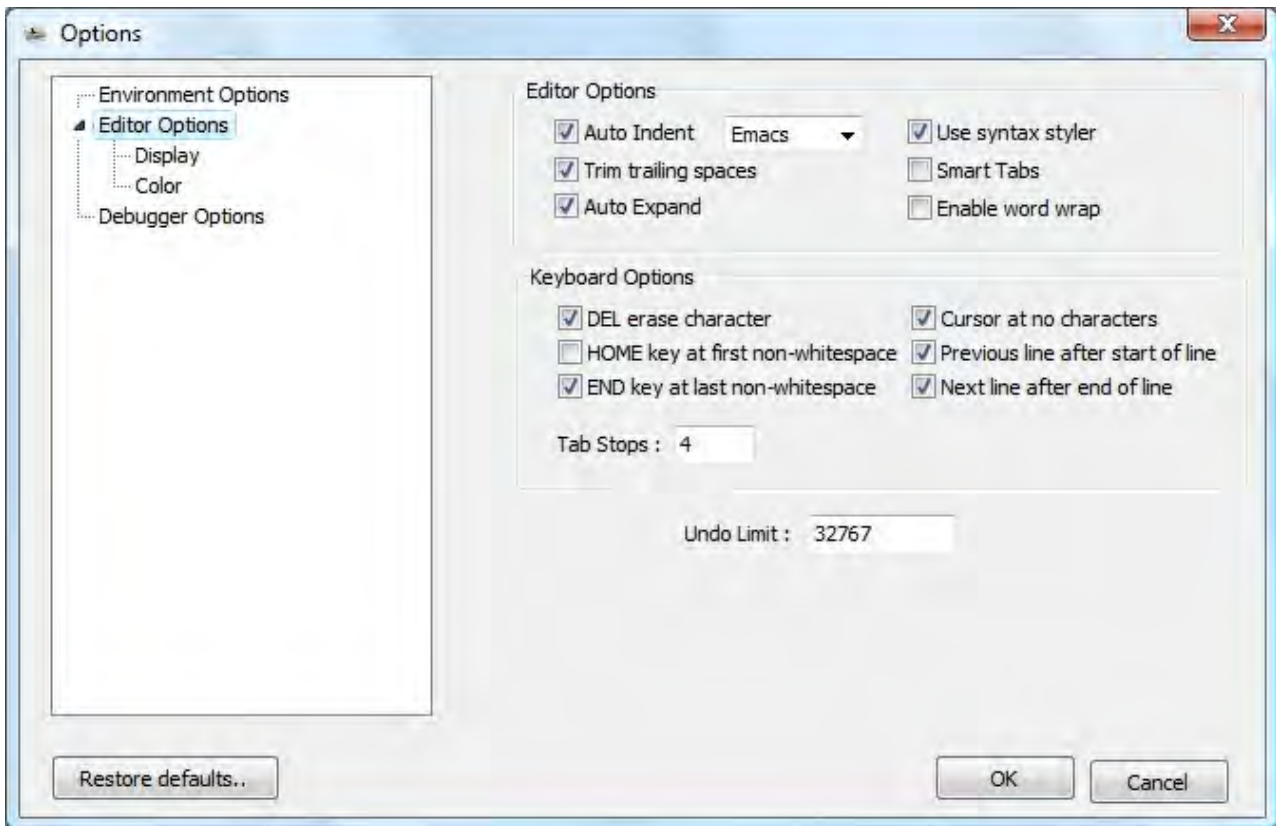
Specifies the directory where the linker tool is found. If there is no path specified, the IDE uses the path environment variable for the path.

Debugger Directory*

Specifies the directory where the debugger tool is found. If there is no path specified, the IDE uses the path environment variable for the path. At present the debugger directory is overridden by the path environment variable.

3.5.3.5.3 Editor Options

The Editor Options section allows for configuring the functional behavior of the source editor.



Auto Indent

If checked, auto indenting of next line is implemented

Auto Indent type

Sets the behavior of the auto indenting function

Trim trailing spaces

If checked, spaces at the end of lines are automatically removed.

Auto Expand

If checked, setting the cursor one or more characters after the last character of a line will automatically expand the line with spaces till the cursor position.

Use syntax styler

If checked, the editor will format the text using a built in syntax styler.

Smart tabs

If checked, smart tabs are used, performing tabs based on columns in the previous line of the memo.

Enable word wrap

If checked, word-wrapping of text is activated.

DEL erase character

If checked, Delete key erases text instead of removing the character.

HOME key at first non-whitespace

If checked, pressing HOME key will bring the cursor to the first non-whitespace character in the line. If not, the cursor will go to the first column

END key at last non-whitespace

If checked, pressing END key will bring the cursor to the last non-whitespace character in the line. If not, the cursor will go to the last column including whitespaces

Cursor at no whitespace

If unchecked, clicking the cursor in an area in the line beyond the last non-whitespace will bring the cursor after the last non-whitespace

Previous line after start of line

If checked, pressing the left arrow when the cursor is at column 0 will bring the cursor at the end of the previous line, else the cursor stays at column 0

Next line after end of line

If checked, pressing the right arrow when the cursor is at the last non-whitespace character will bring the cursor to the start of the next line.

Tab size

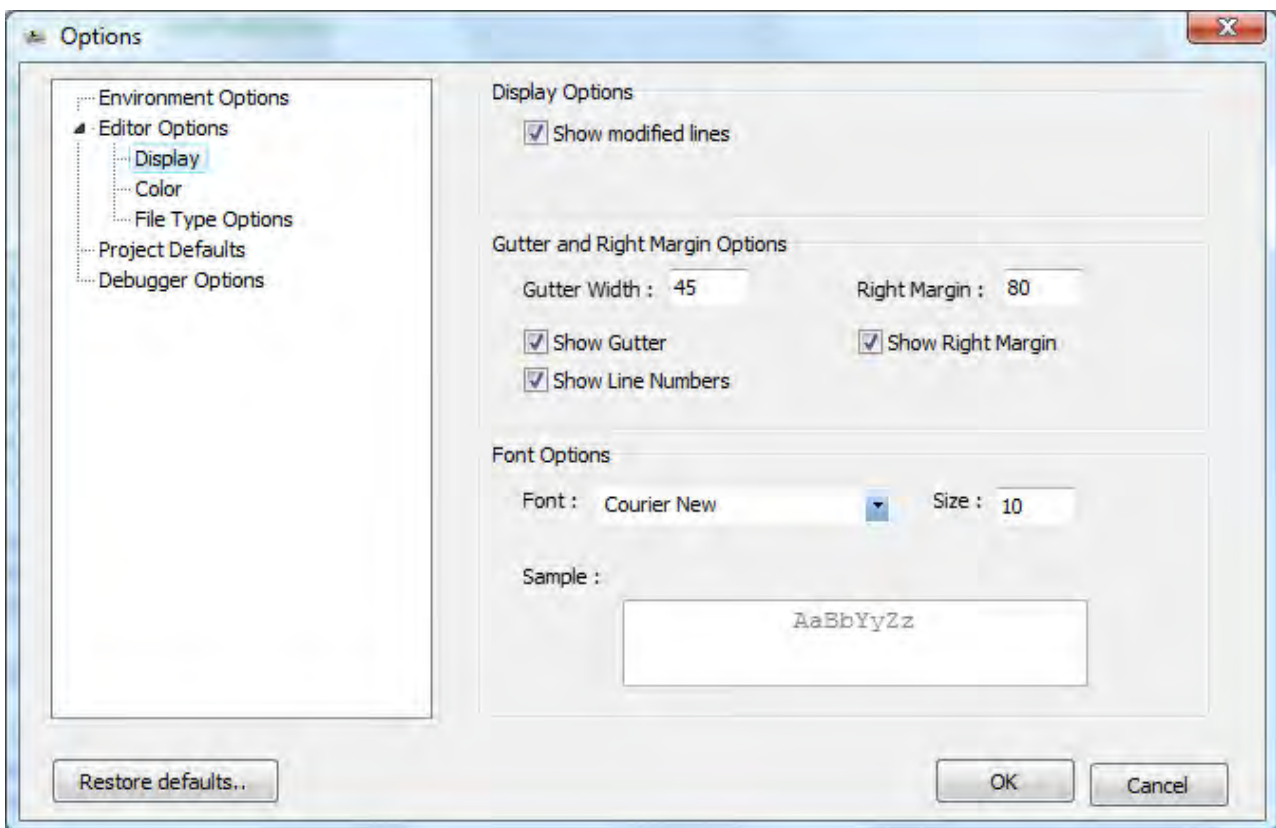
Sets the size of the tab in the memo.

Undo limit

Sets the maximum number of undo operations allowed.

3.5.3.5.4 Display Options

The Display Options sections covers the visual configuration of the source editor.



Show modified lines

If checked, a yellow colour is displayed on the gutter corresponding to the modified line numbers of the active file

Gutter Width

Sets the width of the gutter on the left.

Show Gutter

If checked, the gutter is displayed on the left side of the source editor.

Show Line Numbers

If checked, the line numbers of the active file is shown on the gutter.

Right Margin Width

Sets the width from the first column of the right margin line

Show Right Margin

If checked, a line is displayed on the right side of the editor signifying the right margin.

Font

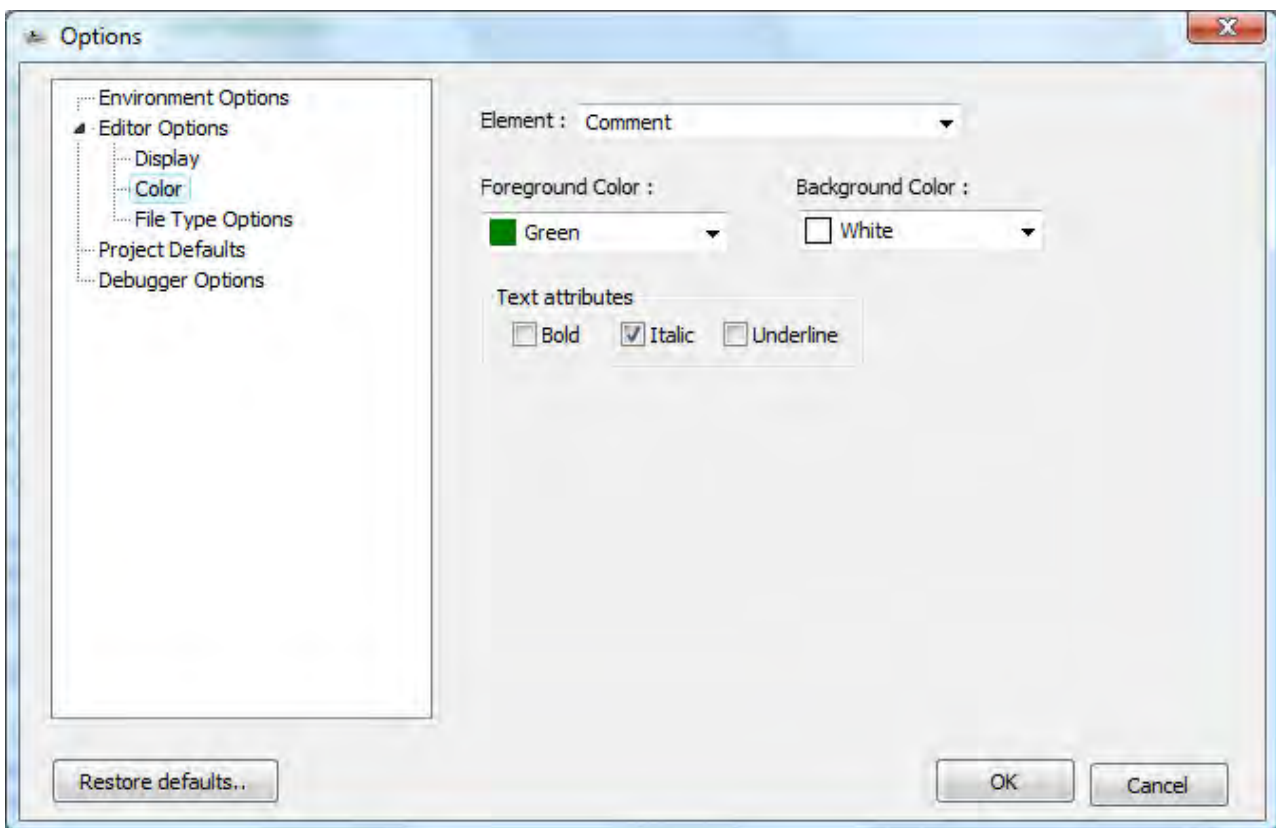
Sets the font name to be used for the editor

Size

Sets the font size

3.5.3.5.5 Colour Options

The Colour Options sections covers the colour and other font settings for the various syntax elements of the editor.



Element

Sets the element where the current settings will apply.

Foreground Colour

Sets the text colour of the selected element.

Background Colour

Sets the text background colour of the selected element.

Bold

Sets the element into bold font style.

Italic

Sets the element into italic font style.

Underline

Sets the element into underline font style.

3.5.3.6 Plugins

3.5.3.6.1 IOMux

Introduction

VNC2 comes with a variety of peripheral interfaces including UART, SPI and FIFO. The IC is available in 32-pin, 48-pin or 64-pin packages. Each of the peripheral interfaces on the chip can be routed for all the available package types. Routing of interfaces on VNC2 is achieved through the use of an I/O Multiplexer (IOMux). The IOMux provides a simple interface to route signals to their required pins.

To aid with routing signals using the IOMux interface FTDI provide an IOMux Configuration Utility. The utility provides a visual representation of all of the available package types and allows for available signals to be routed on individual pins. The utility will generate C code that can be automatically referenced and added to the current project.

Launching Application

The IOMux plugin comes as part of the VNC2 toolchain installation. The application appears as an icon within the View menu of the IDE menu items. To launch the application click on the IOMux icon. Figure 1 shows the main screen after launching the application.

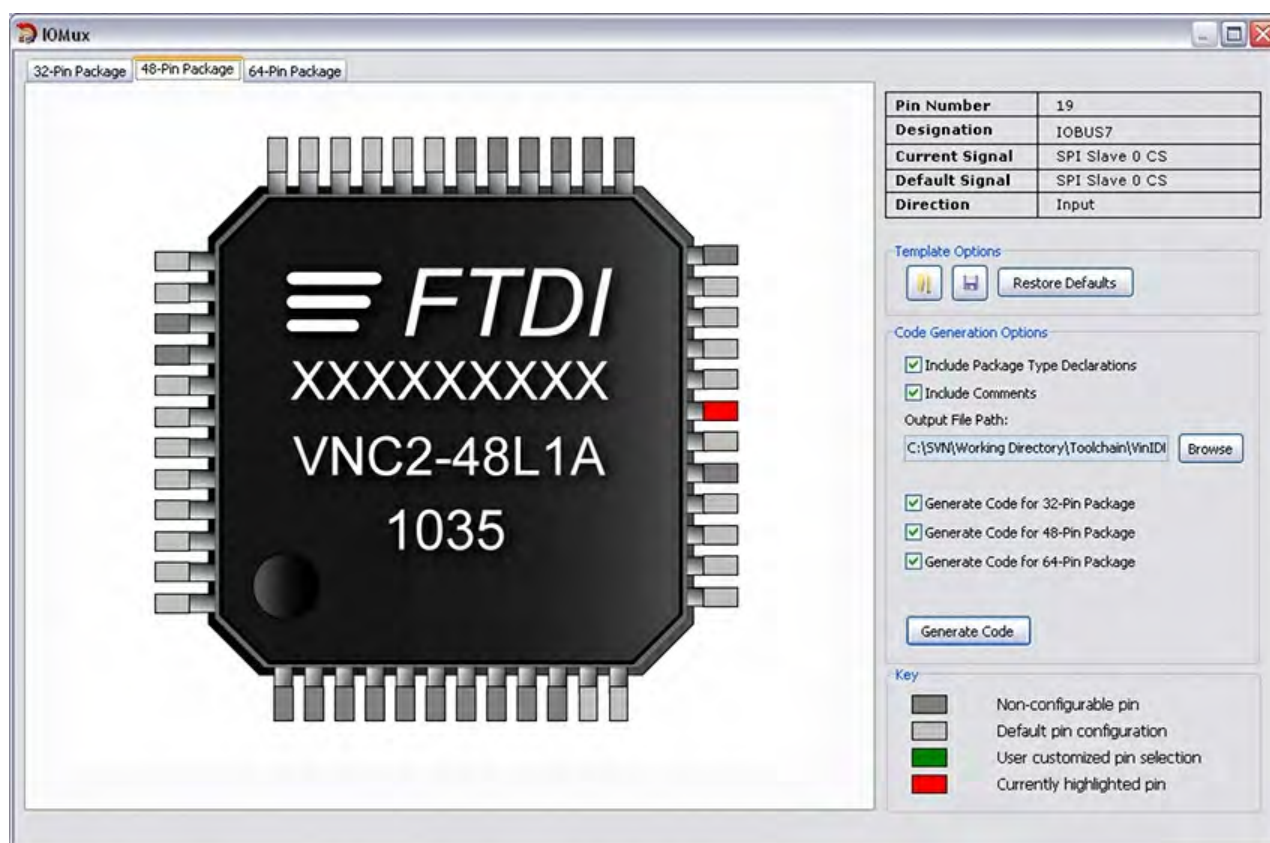


Figure 1 - IOMux Configuration Utility Overview

Configuring Pins

Each VNC2 package has a default pin state following a hard reset. Rolling the mouse over individual pins within the utility will change the pin information in the top right hand corner of the screen to reflect the default signal currently routed to each pin. Signals that cannot be re-routed (GND, VCC, Prog etc) are displayed in Dark Grey, configurable pins will highlight Red upon mouse over.

Individual signals have different attributes regarding the flow of data through them. For example, the UART transmit signal (UART TXD) only allows data to flow as an output signal whereas the UART receive signal (UART RXD) only allows data to flow as an input signal. Certain signals allow for data

to be transferred in both directions; an example of this would be the GPIO signals.

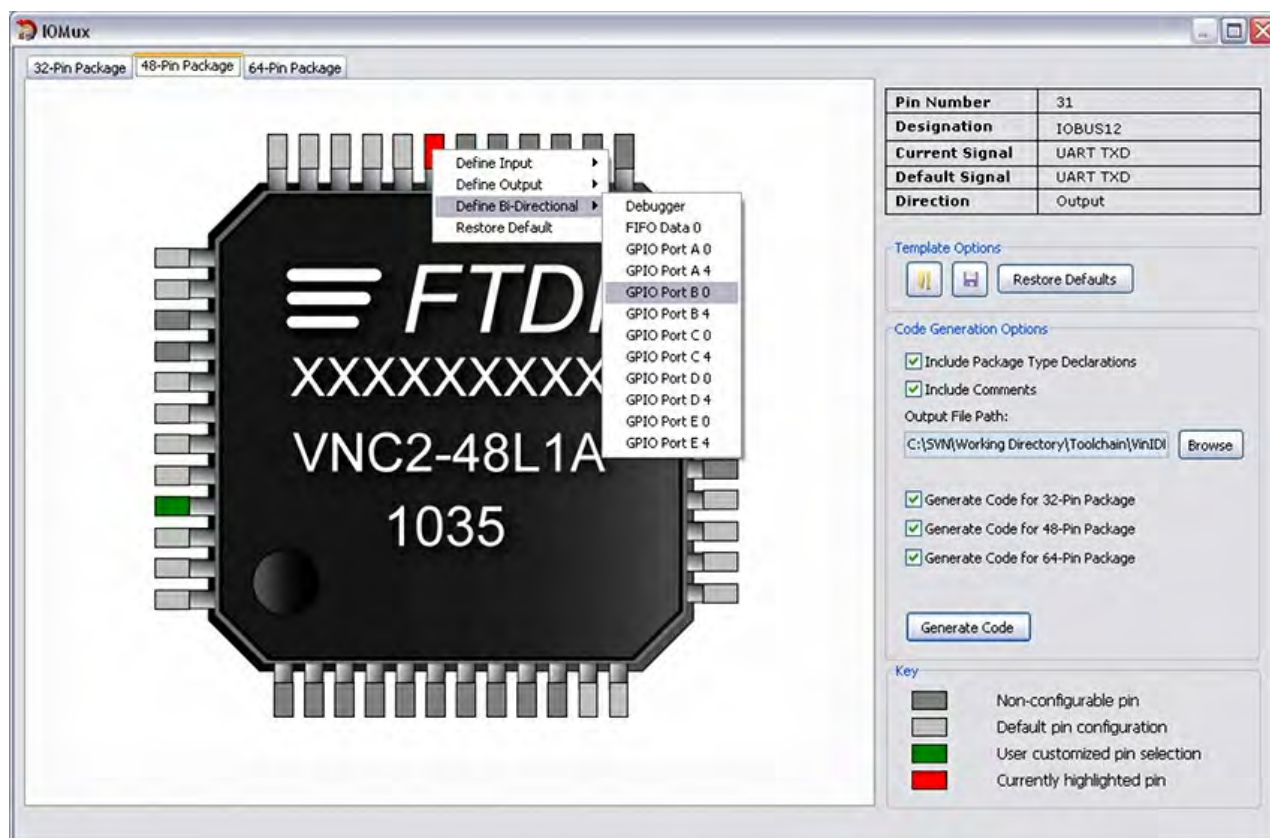


Figure 2 - IOMux Pin Configuration

To configure a routable pin: left-click on the desired pin; select whether this should be an input, output or bi-directional signal; finally, choose the required signal from the available list. Pins that are no longer routed to their default signal will turn Green. To return a pin to its default signal, left-click the pin and select Restore Default from the menu. Note that each signal can only be routed to one pin on any given package; for example the Debugger signal, default pin 11, can be routed to pin 31 (amongst others) on the 48 pin package; if the signal is re-routed to pin 31 this will have the effect of removing the reference to the debugger on pin 11.

Using Templates

It is possible to save IOMux configurations in xml template form allowing for configurations to be altered at a later date. To save a configuration in template form, click the Save Template icon under Template Options section and enter the desired template name within the save dialog box. Previous templates can be loaded into the utility by selecting the Load Template icon, again from the Template Options section. To completely clear all re-routed signals from the current template on all package types click Restore Defaults which will return all pins to their default signal state.

Generating Code

The utility will generate C code that will be automatically added to the current project if one is open. Code can be generated for all or only one of the available package types. The Code Generation Options section contains a list of packages that the application will generate code for.

After clicking the Generate Code button the application will generate a C file containing IOMux configuration code for the current IOMux setup at the location specified by the Output File Path text box. If there is a project currently open the application will attempt to add the generated file to the current project and to then reference the SetupIOMux function from within the main line of the user application. As a check list the utility will perform the following tasks to reference the code within the project:

- Add IOMux C file to the active project.
- Add void SetupIOMUX(); forward declaration at the head of the file containing the main function.

- Call the SetupIOMUX(); function prior to starting the scheduler within the main function.

Note that if no project is currently open IOMux code will still be created at the specified file location, however no attempt will be made to reference the code within a project as outlined above.

3.5.3.6.2 Code Inspector

Introduction

Code Inspector displays an overall hierarchical representation of the current application code structure. The treeview displays references to user functions/variables/data types as well as VOS kernel functions and their appropriate variables/data types. Double clicking items within the treeview will open the appropriate file containing the reference and highlight the relevant text within the editor. Code Inspector runs as a realtime thread on user code therefore, after saving changes to an application, the Code Inspector treeview will be updated appropriately.

Launching Application



The plugin comes as part of the VNC2 toolchain installation. The application appears as an icon within the *View* menu of the IDE menu items. To launch the application click on the *Code Inspector* icon. Figure 1 shows the main screen after launching the application. The treeview will be updated automatically when a project is opened and a C file is opened within the editor.





Figure 1 - Code Inspector Overview

Toolbar








The Toolbar menu allows for customization of the Code Inspector treeview.

	Sort Alphabetically	Sorts the list of functions and data types in A-Z alphabetical order.
	Group by Type	Groups the items within the treeview by type.

	Show Function/Variable Types	Shows the return type of functions and the type of variables.
	GoTo Definition	Jumps to the file containing the current item highlighted within the treeview.

Treeview Symbols

The following key is used for items within the treeview:

	Structure
	Enum
	Enum Value
	Variable
	Function Prototype
	Function
	Label

3.5.3.7 Keyboard Shortcuts

These are the keyboard shortcuts to perform various functions in VinIDE :

CTRL-SHIFT-N	New Project
CTRL-N	New File
CTRL-SHIFT-S	Save All
CTRL-S	Save File
CTRL-F	Find
CTRL-C	Copy
CTRL-V	Paste
CTRL-X	Cut
CTRL-Z	Undo
CTRL-Y	Redo
CTRL-SHIFT-O	Open Project
CTRL-O	Open File
CTRL-H	Replace
F7	Build Project
CTRL-SHIFT-F11	Project Options
CTRL-TAB	Next Tab

CTRL-SHIFT-TAB	Previous Tab
CTRL-F4	Close Tab
F3	Search Again
F9	Breakpoint
F1	Help
CTRL-0	Select Project Manager
CRTL-1	Select Properties Panel
CTRL-2	Select Messages Panel
CTRL-3	Select Watch Panel
CTRL-4	Select Memory Panel
CTRL-5	Select Breakpoint List
F5	Start
F10	Step
F11	Step Into
SHIFT-F11	Step Out
SHIFT-F12	Run To Cursor
CTRL-F4	Show Disassembly
SHIFT-F6	Reset
F6	Stop
SHIFT-F5	Pause
F4	Flash
CRTL-ALT-L	Open Libraries
CRTL-ALT-H	Open Header Files
CRTL-P	Print

4 Firmware

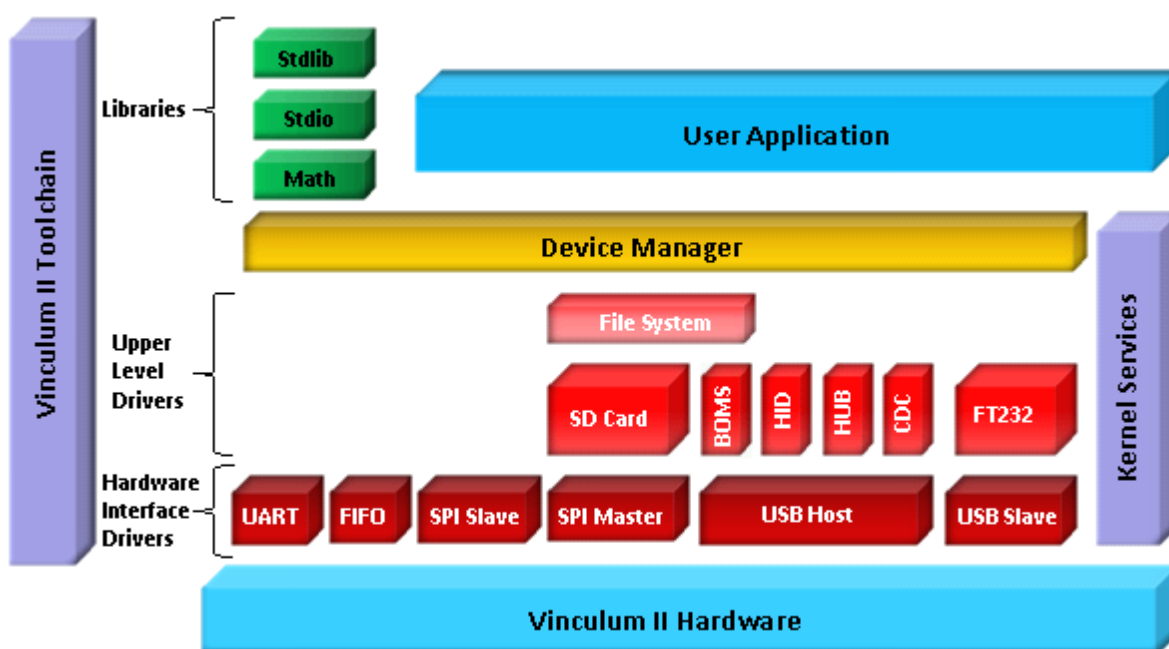
The VNC2 firmware model consists of three layers.

- [VOS Kernel](#). This is responsible for managing hardware resources, interrupts and scheduling.
- [FTDI Drivers](#). These control either a hardware resource or other device drivers. They provide a programming interface to higher level code.
- [FTDI Libraries](#).
- User Applications. This is where the functionality of the firmware is provided by controlling various device drivers.

The user application calls an API to communicate with device drivers which access the hardware resources. The kernel provides an API for device drivers and user applications to control the operation of the VNC2.

Some device drivers require a thread to control a hardware resource - others are able to work using only events (interrupts and API calls).

User applications are allowed to run multiple threads.



4.1 VOS Kernel

The VNC2 RTOS (VOS) is a pre-emptive priority-based multi-tasking operating system.

VOS has the following features:

- Priority based tasks. Tasks are run by a [Kernel Scheduler](#) which decides which tasks will run based on their priority. The scheduler also has the ability to provide priority boosts to low priority tasks from being completely denied processor time. Tasks can also be deliberately delayed.
- Task switching. When a task's allotted processor time has elapsed, the task is paused and the next task in the ready list allowed to run. In order for this to happen, VOS will save the context of the running task and restore the context of the task to be run. The time which a task is allocated depends on a value called the quantum.
- Task synchronisation. Several mechanisms are provided:
 - [Mutexes](#) are provided so tasks can achieve exclusive access to a resource.
 - [Semaphores](#) are also available for inter-task communication.
 - [Condition Variables](#) are provided to allow tasks to synchronise based on the value of data.

- [Critical Sections](#) are provided as a method of claiming additional CPU time beyond the normal allotted time slice for a task to run, allowing important code paths to complete before allowing a task switch.
- [Device Manager](#). The device manager provides the interface between user application code and the device drivers. When a user application successfully opens a device, the device manager allocates a handle to the device that the application can use for subsequent communication. Opening a device obtains exclusive access to the device. User applications have access to a standard set of device driver functions via the device manager:
 - [vos_dev_open\(\)](#) – obtain exclusive access and a handle to the device.
 - [vos_dev_close\(\)](#) – release the device handle and relinquish exclusive access.
 - [vos_dev_read\(\)](#) – read data from the device.
 - [vos_dev_write\(\)](#) – write data to the device.
 - [vos_dev_ioctl\(\)](#) – device specific operations.
 - interrupt – hardware interrupt handler.

4.1.1 VOS Definitions

There are certain definitions for variable and function types which are used throughout the kernel and drivers. They are available to applications in the vos.h header file.

Null pointer and logic definitions:

```
#define NULL          0
#define TRUE          1
#define FALSE         0
```

Variable type definitions:

```
#define uint8          unsigned char
#define int8           char
#define int16          short
#define uint16         unsigned short
#define uint32         unsigned int
#define pvoid          unsigned char *
```

Function type definitions:

```
typedef uint8 (*PF)(uint8);
typedef void (*PF_OPEN)(void *);
typedef void (*PF_CLOSE)(void *);
typedef uint8 (*PF_IOCTL)(pvoid);
typedef uint8 (*PF_IO)(uint8 *, unsigned short, unsigned short *);
typedef void (*PF_INT)(void);
```

4.1.2 Kernel Initialisation

The kernel maintains lists and data structures which must be initialised prior to use. The [vos_init\(\)](#) call is used to initialise the kernel data and also set the task switching quantum and tick count.

The tick count is the number of milliseconds the kernel will wait before evaluating the quantum of a process. The quantum is the number of kernel ticks for which a process will run until it is either:

- interrupted
- blocked - the thread calls a kernel function that blocks until completion e.g. [vos_lock_mutex\(\)](#) or [vos_wait_semaphore\(\)](#)
- pre-empted - a higher priority task has become unblocked by an interrupt
- delayed - a call to [vos_delay_msecs\(\)](#) will make the process delay

When any of the above occur or at the expiry of the quantum the kernel will switch to the next highest priority task which is not blocked or delayed. This is called task switching.

The default tick count is one and default quantum is 50. These values give balanced performance for general purpose programs that may want to perform some data processing as well as input and output operations. Decreasing the quantum will increase the number of task switches which occur and may allow multiple threads to collaborate better in sharing data or resources.

It is not recommended to alter the tick count from it's default value unless the application is to continue processing at the detriment of responsiveness.

There is very little performance overhead in switching tasks so a quantum of 10 or below is possible and can be beneficial in some systems. The quantum will have a larger effect on applications where the multiple threads share the same priority level and where a thread performs significant amounts of processing without blocking or delaying.

Where the time required to respond to an event is critical a tick count of 1, a quantum of 10 combined with a thread priority scheme where the more important threads have a higher priority will provide a good solution.

4.1.2.1 vos_init()

Syntax

```
void vos_init(uint8 quantum, uint16 tick_cnt, uint8 num_devices);
```

Description

Initialise the kernel and kernel memory structures.

Parameters

quantum

The quantum parameter is used to set the time period in milliseconds between task switches by the kernel.

tick_cnt

The tick_cnt value specifies the number of milliseconds (timer ticks) between context switches by the scheduler.

num_devices

The device manager is initialised with the number of devices passed in the num_devices parameter.

Returns

The function does not return any value.

Comments

The following definitions, providing default values, are available for use in vos_init() function calls:

```
VOS_TICK_INTERVAL  
VOS_QUANTUM
```

The num_devices parameter reserves slots for drivers which are managed by the device manager. Each and every slot must be configured using the vos_dev_init() function before the scheduler can be started or any interrupts enabled.

4.1.3 Thread Creation

A thread is created with the [vos_create_thread\(\)](#) call. No threads will run until the [Kernel Scheduler](#) is started.

Each thread has a block of memory set aside for it's stack and state information. This is allocated dynamically by [vos_create_thread\(\)](#).

Multiple threads can be run, the only limitation is the amount of memory available for them.

Each thread can be allocated a priority. Higher priority threads have a higher priority value assigned when they are created. Values less than 32 may be used by an application, however, zero is reserved for the idle task.

Optional parameters may be passed to a thread. The number of bytes for the arguments is specified

followed by the arguments themselves.

If multiple threads share the same priority level then they will be run sequentially in a round robin fashion. To allow a thread to respond to an event more quickly increase it's priority relative to less important threads.

As a general guideline avoid tight code loops where a thread is polling for an event. It is far better to use a kernel synchronisation event to notify a thread. If the higher priority thread is polling without a mutex, semaphore, condition variable or delay then it will prevent the lower priority thread from running causing a deadlock situation. Where tight loops are unavoidable add in a short delay call to [vos_delay_msecs\(\)](#) to allow another thread a short time to perform calculations.

4.1.3.1 vos_create_thread()

Syntax

```
vos_tcb_t *vos_create_thread(uint8 priority, uint16 stack, fnVoidPtr function, int16 arg_size,...
```

Description

Create a thread to call a function and pass optional parameters.

Parameters

priority
The priority parameter specifies a kernel priority to run the tread at.

stack
The stack parameter is the size of the stack to allocate to the thread.

function
The function is a pointer to a function to run as the entry point of the thread.

arg_size
Specifies the size of the optional parameters to pass.

Returns

The function returns a pointer to a kernel task control block.

Comments

The memory for thread stack space is reserved and initialised vos_create_thread(). If optional parameters are not required for the thread then set arg_size to zero. Multiple parameters may be passed with the total size of them set in arg_size.

Tasks are not actually started until vos_start_scheduler() is called.

The priority of the thread specified should be greater than zero and less than 32. A higher number indicates a higher priority.

Example

```
#define SIZEOF_TASK_1_MEMORY 0xa00
#define SIZEOF_TASK_2_MEMORY 0x800

void task1();
void task2(int);

vos_tcb_t *tctl1, *tcb2;

void main(void)
{
    int x = 4;
    vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, 0);

    tcb1 = vos_create_thread(30, SIZEOF_TASK_1_MEMORY , task1, 0);
    tcb2 = vos_create_thread(28, SIZEOF_TASK_2_MEMORY , task2, sizeof(int), x);
}
```

4.1.4 Kernel Scheduler

When scheduler starts control is passed from the `main()` function to kernel, threads are started. Control never returns to `main()`.

Delay timers, semaphores, mutexes and condition variables cannot be used before the scheduler is started.

All devices declared in the `num_devices` parameter of the call to [vos_init\(\)](#) must be initialised and registered with the [Device Manager](#) before [vos_start_scheduler\(\)](#) is called.

Delays may be added to any thread using the [vos_delay_msecs\(\)](#), and another thread may cancel a delay in a thread using [vos_delay_cancel\(\)](#). Delays may be longer than requested due to a higher priority thread running.

4.1.4.1 vos_start_scheduler()

Syntax

```
void vos_start_scheduler(void);
```

Description

Pass control to the kernel scheduler.

Parameters

There are no parameters required.

Returns

The function does not return any value.

Comments

Control is passed to the kernel scheduler. The function will never return to the calling routine. This is normally found in the `main()` function of an application.

4.1.4.2 vos_delay_msecs()

Syntax

```
uint8 vos_delay_msecs(uint16 ms);
```

Description

Delay a thread for a minimum period of time.

Parameters

ms
The `ms` parameter specifies the minimum number of milliseconds which the current thread will be delayed by. It may delay a longer time depending on the state of other threads.

Returns

The function returns zero for normal completion or non-zero if another thread has cancelled the delay.

Comments

This may only be called from a thread after the kernel scheduler has started.

Example

```
void powerOnTest()  
{  
    uint16 delay;
```



```
    delay = 100;
    if (sendPowerOn() == 1)
    {
        // add an extra 500ms to delay
        delay += 500;
    }
    // wait until power good
    vos_delay_msecs(delay);
}
```

4.1.4.3 vos_delay_cancel()

Syntax

```
void vos_delay_cancel(vos_tcb_t *tcb);
```

Description

Cancel a delay in another thread.

Parameters

tcb

The tcb parameter specifies another thread which may be in a delayed state.

Returns

The function does not return any value.

Comments

This may only be called from a thread after the kernel scheduler has started.

4.1.5 Mutexes

Mutexes are used for synchronisation or to enforce mutual exclusion, and hence serialise access to a shared resource. The resource is not actually specified for a particular mutex, it is up to the programmer to ensure that the mutex is used for all instances of access to the resource.

Mutexes must be initialised before use but can be initialised as locked or unlocked using [vos_init_mutex\(\)](#).

A [vos_lock_mutex\(\)](#) request will block until the mutex is unlocked. However, a [vos_trylock_mutex\(\)](#) will return and report if a mutex is locked. If the mutex is free then it will be locked.

The lock status of a mutex can be tested using the [vos_trylock_mutex\(\)](#) feature followed by a [vos_unlock_mutex\(\)](#) call if the mutex was free.

The mutex is defined as:

```
typedef struct _vos_mutex_t {
    vos_tcb_t *threads;           // list of threads blocked on mutex
    vos_tcb_t *owner;            // thread that has locked mutex
    uint8 attr;                  // attribute byte
    uint8 ceiling;               // priority for priority ceiling protocol
} vos_mutex_t;
```

Advanced mutex operations are available to raise or lower the priority ceiling allowing the priority of the mutex to increase until it is processed.

Example

Consider an application with two threads which require to be synchronised. The first thread is used to initialise some application specific data and then both threads can begin operation. A mutex is used to signal thread th2 that the first thread th1 is complete.

```
vos_mutex_t mReady;
void main(void)
{
```

```
vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, 1);
vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);
// initialise mutex to be locked
vos_init_mutex(&mReady, 1);
vos_create_thread(30, MEMTHREAD1, th1, 2, &mReady);
vos_create_thread(30, MEMTHREAD2, th2, 2, &mReady);
vos_start_scheduler();
}
void th1(vos_mutex_t *m)
{
    // perform initialisation
    vos_unlock_mutex(m);
    // continue thread tasks
}
void th2(vos_mutex_t *m)
{
    // wait for th1 to complete initialisation
    vos_lock_mutex(m);
    vos_unlock_mutex(m);
    // continue thread tasks
}
```

Example

Another example is where access to a variable is controlled or gated by a mutex.

```
vos_mutex_t mBusy;
char chBusy;
void main(void)
{
    vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, 1);
    vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);
    // initialise mutex to be unlocked
    vos_init_mutex(&mBusy, 0);
    vos_create_thread(30, MEMTHREAD1, th1, 0);
    vos_create_thread(30, MEMTHREAD2, th2, 0);
    vos_start_scheduler();
}
void dp1()
{
    while (1)
    {
        // lock chBusy until we write it
        vos_lock_mutex(&mBusy);
        chBusy = 'a';
        vos_unlock_mutex(&mBusy);
        // continue thread tasks
    }
}
void dp2()
{
    while (1)
    {
        // don't read chBusy unless it's locked
        vos_lock_mutex(&mBusy);
        if (chBusy == 'a') chBusy = 'b';
        vos_unlock_mutex(&mBusy);
        // continue thread tasks
    }
}
```

4.1.5.1 vos_init_mutex()

Syntax

```
void vos_init_mutex(vos_mutex_t *m, uint8 state);
```

Description

Initialises a mutex and sets it's initial value.

Parameters

- m**
The m parameter is a pointer to a mutex structure.
- state**
The value of state is the initial value of the mutex after initialisation.

Returns

The function does not return any value.

Comments

The initial value of the mutex must be one of the two options:

VOS_MUTEX_UNLOCKED
VOS_MUTEX_LOCKED

4.1.5.2 vos_lock_mutex()

Syntax

```
void vos_lock_mutex(vos_mutex_t *m);
```

Description

Performs a lock operation on a mutex to prevent any other process from locking it. If the mutex is already locked then the function will block until the mutex is released by the other process.

Parameters

- m**
The m parameter is a pointer to a mutex structure.

Returns

The function does not return any value.

Comments

To test if a mutex is already locked and therefore prevent a locking situation use the vos_trylock_mutex() function.

4.1.5.3 vos_trylock_mutex()

Syntax

```
uint8 vos_trylock_mutex(vos_mutex_t *m);
```

Description

Tests the lock status of a mutex and performs a lock operation if it is unlocked. If it is already locked then it will return immediately.

Parameters

- m**
The m parameter is a pointer to a mutex structure.

Returns

The function returns 0 if the mutex was available and is now locked. Otherwise, 1 will be returned and the mutex will continue to be locked by another process.

Comments

The following definitions are available for testing the return value of vos_trylock_mutex().

```
#define VOS_MUTEX_UNLOCKED    0
#define VOS_MUTEX_LOCKED     1
```

4.1.5.4 vos_unlock_mutex()

Syntax

```
void vos_unlock_mutex(vos_mutex_t *m);
```

Description

Performs an unlock operation on a mutex allowing it to be locked by other processes.

Parameters

^m
The m parameter is a pointer to a mutex structure.

Returns

The function does not return any value.

Comments

The next process with the highest priority which is waiting on the mutex with the vos_lock_mutex() function will lock the mutex.

4.1.5.5 vos_get_priority_ceiling() Advanced

Syntax

```
uint8 vos_get_priority_ceiling(vos_mutex_t *m);
```

Description

Returns the priority ceiling of a mutex.

Parameters

^m
The m parameter is a pointer to a mutex structure.

Returns

The function returns the priority ceiling of the mutex.

Comments

The priority ceiling is the priority to which a mutex is allowed to rise to prevent deadlock situations.

4.1.5.6 vos_set_priority_ceiling() Advanced

Syntax

```
void vos_set_priority_ceiling(vos_mutex_t *m,uint8 priority);
```

Description

Sets the priority ceiling of a mutex.

Parameters

^m

The m parameter is a pointer to a mutex structure.

priority

Specifies the maximum priority to which a thread blocked on a mutex can rise to.

Returns

The function does not return a value.

Comments

4.1.6 Semaphores

A semaphore is similar to a mutex but has a count value associated with it. This allows an application to specify a number of concurrent access to a shared resource or to queue multiple events to be processed.

Semaphores must be initialised before use with an initial count value using [vos_init_semaphore\(\)](#).

An ideal use of a semaphore is when multiple resources are available and need to be tracked to make sure that only the required number of these resources are in use at any one time.

A call to [vos_wait_semaphore\(\)](#) will block until a semaphore is available. A call to [vos_wait_semaphore_ex\(\)](#) can be used for waiting on either one semaphore to be available from a list or all semaphores in a list to become available.

A semaphore is defined as:

```
typedef struct _vos_semaphore_t {
    int16 val;
    vos_tcb_t *threads;
    int8 usage_count;
} vos_semaphore_t;

typedef struct _vos_semaphore_list_t {
    struct _vos_semaphore_list_t *next;
    int8 siz;
    uint8 flags; // bit 7 set for WAIT_ALL clear for WAIT_ANY
    vos_semaphore_t *list[1];
} vos_semaphore_list_t;
```

The Philosophers Sample application shows semaphores used to do multi-process synchronisation.

Example

An example of a semaphore would be a buffer containing 16 bytes. A producer thread will add bytes to the buffer and a consumer thread remove bytes. The producer must not write more than 16 bytes, but can write more bytes after the consumer takes them off the stack.

```
// resource semaphore
vos_semaphore_t semBuf;
// the resources to protect
char buffer[16];
char *pBuf;
// mutex to protect buffer pointer
vos_mutex_t mBuf;
void producer();
void consumer();
void main(void)
{
    vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, 1);
    vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);
    // initialise semaphore to be sizeof buffer
    vos_init_semaphore(&semBuf, sizeof(buffer));
    vos_init_mutex(&mBuf, 0);
    pBuf = &buffer[0];
    vos_create_thread(30, MEMTHREAD1, producer, 0);
    // consumer thread has a lower priority than producer
    vos_create_thread(29, MEMTHREAD2, consumer, 0);
    vos_start_scheduler();
}
```

```
}
void producer()
{
    char queueCount;
    while (1)
    {
        vos_wait_semaphore(&semBuf);
        vos_lock_mutex(&mBuf);
        *pBuf = queueCount;
        pBuf++;
        vos_unlock_mutex(&mBuf);
        queueCount++;
    }
}
void consumer()
{
    char myCount;
    while (1)
    {
        vos_lock_mutex(&mBuf);
        pBuf--;
        myCount = *pBuf;
        vos_unlock_mutex(&mBuf);
        vos_signal_semaphore(&semBuf);
    }
}
```

4.1.6.1 vos_init_semaphore()

Syntax

```
void vos_init_semaphore(vos_semaphore_t *sem,int16 count);
```

Description

Initialises a semaphore and sets its initial value

Parameters

sem
Pointer to a semaphore structure. The initial value of the semaphore is set to count.

Returns

There is no return value.

Comments

Example

The following code fragment shows how to declare a semaphore and initialise it with an initial value of 1.

```
vos_semaphore_t sem;
vos_init_semaphore(&sem,1);
```

4.1.6.2 vos_wait_semaphore()

Syntax

```
void vos_wait_semaphore(vos_semaphore_t *s);
```

Description

Perform a wait operation on a semaphore. The semaphore's count field is decremented and the current thread is blocked if the value of count is less than zero.

Parameters

`sem`
Pointer to a semaphore structure.

Returns

There is no return value.

Comments

4.1.6.3 vos_wait_semaphore_ex()

Syntax

```
int8 vos_wait_semaphore_ex(vos_semaphore_list_t *l);
```

Description

Perform a wait operation on multiple semaphores. The semaphores are passed to this function on a list, and the wait operation can be performed for all semaphores on the list or any semaphore on the list.

Parameters

1
Pointer to a semaphore list structure.

Returns

For VOS_SEMAPHORE_FLAGS_WAIT_ANY, return index in the semaphore list of the semaphore that was signalled.

For VOS_SEMAPHORE_FLAGS_WAIT_ALL, return 0.

Comments

Between repeated calls to vos_wait_semaphore_ex the pointers stored in the list array require to be updated. All elements in this array are set to zero before the vos_wait_semaphore_ex function returns. The next, siz and flags members are not modified and do not need to be updated.

Example

The following code fragments show how to use vos_wait_semaphore_ex.

In the first example the current thread will block until either sem1 or sem2 is signalled.

```
#define NUMBER_OF_SEMAPHORES 2
vos_semaphore_list_t *sem_list;           // pointer to semaphore list
vos_semaphore_t sem1;
vos_semaphore_t sem2;
int8 n;

vos_init_semaphore(&sem1,0);              // initialise semaphores
vos_init_semaphore(&sem2,0);

sem_list = (vos_semaphore_list_t *) malloc(VOS_SEMAPHORE_LIST_SIZE(NUMBER_OF_SEMAPHORES));
sem_list->next = NULL;                    // initialise semaphore list
sem_list->siz = NUMBER_OF_SEMAPHORES;    // 2 semaphores
sem_list->flags = VOS_SEMAPHORE_FLAGS_WAIT_ANY;
sem_list->list[0] = &sem1;
sem_list->list[1] = &sem2;
n = vos_wait_semaphore_ex(sem_list);

if (n == 0) {
    // sem1 has signalled
}
else if (n == 1) {
    // sem2 has signalled
}
```

```
free(sem_list);
```

In the second example the current thread will block until both sem1 and sem2 are signalled.

```
#define NUMBER_OF_SEMAPHORES 2
vos_semaphore_list_t *sem_list;           // pointer to semaphore list
vos_semaphore_t sem1;
vos_semaphore_t sem2;
int8 n;

vos_init_semaphore(&sem1,0);               // initialise semaphores
vos_init_semaphore(&sem2,0);

sem_list = (vos_semaphore_list_t *) malloc(VOS_SEMAPHORE_LIST_SIZE(NUMBER_OF_SEMAPHORES));
sem_list->next = NULL;                    // initialise semaphore list
sem_list->siz = NUMBER_OF_SEMAPHORES;     // 2 semaphores
sem_list->flags = VOS_SEMAPHORE_FLAGS_WAIT_ALL;
sem_list->list[0] = &sem1;
sem_list->list[1] = &sem2;
n = vos_wait_semaphore_ex(sem_list);

if (n == 0) {
    // sem1 and sem2 have signalled
}
free(sem_list);
```

4.1.6.4 vos_signal_semaphore()

Syntax

```
void vos_signal_semaphore(vos_semaphore_t *s);
```

Description

Perform a signal operation on a semaphore. The *count* variable is incremented and if the value of *count* is less than or equal to zero then the first thread on the semaphore's blocked list is removed and placed on the ready list.

Parameters

sem
Pointer to a semaphore structure.

Returns

There is no return value.

Comments

If, as a result of `vos_signal_semaphore`, a thread with a higher priority than the current thread becomes ready to run, then a task switch will occur and the higher priority thread will become the current thread.

4.1.6.5 vos_signal_semaphore_from_isr()

Syntax

```
void vos_signal_semaphore_from_isr(vos_semaphore_t *s);
```

Description

Perform a signal operation on a semaphore. The count variable is incremented and if the value of count is less than or equal to zero then the first thread on the semaphore's blocked list is removed and placed on the ready list.

Parameters

`sem`
Pointer to a semaphore structure.

Returns

There is no return value.

Comments

`vos_signal_semaphore_from_isr` is used to signal a semaphore from an interrupt service routine. It differs from `vos_signal_semaphore` in that no task switch can occur if, as a result of `vos_signal_semaphore_from_isr`, a thread with a higher priority than the current thread becomes ready to run. In this case, the task switch will occur after the interrupt service routine has been completed.

4.1.7 Condition Variables

Condition variables are used to synchronise threads based on the value of data. They are used in conjunction with a mutex that allows exclusive access to the data value.

Condition variables must be initialised before use using [vos_init_cond_var\(\)](#).

Calling [vos_wait_cond_var\(\)](#) will block until the condition variable becomes true. A mutex is passed in this function which is used to provide exclusive access to the variable which is being tested. To signal that a condition variable is true the [vos_signal_cond_var\(\)](#) function is called.

Type definition for condition variable:

```
typedef struct _vos_cond_var_t {
    vos_tcb_t *threads;
    vos_mutex_t *lock;
    uint8 state;
} vos_cond_var_t;
```

Example

This is a pseudocode example that demonstrates how to use a condition variable. Typically, the condition variable, mutex and data are initialised in a mainline routine. In this example, a thread (not shown) produces a byte of data and calls `add_byte()` to store the data in a buffer. A second thread (not shown) calls `read10bytes()` to read 10 bytes of data from the buffer. These code fragments provide a template for thread synchronisation using a condition variable in conjunction with a mutex.

```
vos_cond_var_t readXferCV;
vos_mutex_t readXferLock;

unsigned short bytesAvailable;

main()
{
    //
    // somewhere in mainline initialisation:
    //     initialise condition variable
    //     initialise mutex
    //     initialise data
    //

    vos_init_cond_var(&readXferCV);
    vos_init_mutex(&readXferLock,0);
    bytesAvailable = 0;
}

unsigned char add_byte(unsigned char b)
{
    //
```

```
// store byte in a buffer
//

//
// lock mutex and increment bytesAvailable
//

vos_lock_mutex(&readXferLock);

++bytesAvailable;

if (bytesAvailable >= 10) {

    //
    // signal that 10 bytes are available
    //
    vos_signal_cond_var(&readXferCV);
}

//
// unlock mutex
//

vos_unlock_mutex(&readXferLock);
}

unsigned char read10bytes(char *xfer)
{
    //
    // lock mutex and check number of bytes available
    //

    vos_lock_mutex(&readXferLock);

    if (bytesAvailable < 10) {

        //
        // wait on condition variable until 10 bytes are available
        //
        vos_wait_cond_var(&readXferCV,&readXferLock);
    }

    //
    // reach here when 10 bytes are available
    //

    vos_unlock_mutex(readXferLock);

    //
    // copy data into transfer buffer and return
    //

    return OK;
}
```

4.1.7.1 vos_init_cond_var()

Syntax

```
void vos_init_cond_var(vos_cond_var_t *cv);
```

Description

Initialises a condition variable.

Parameters

cv
Pointer to a condition variable structure.

Returns

There is no return value.

Comments

Example

The following code fragment shows how to declare a condition variable and initialise it.

```
vos_cond_var_t cv;  
vos_init_cond_var(&cv);
```

4.1.7.2 vos_wait_cond_var()

Syntax

```
void vos_wait_cond_var(vos_cond_var_t *cv, vos_mutex_t *m);
```

Description

Wait on the condition variable *cv*. The calling thread is blocked until another thread performs a *vos_signal_cond_var* operation on *cv*.

Parameters

cv
Pointer to a condition variable structure.

m
Pointer to a mutex structure.

Returns

There is no return value.

Comments

This function works in conjunction with *vos_signal_cond_var* to provide thread synchronisation based on the value of data. Condition variables are always used in conjunction with a mutex that must be locked when *vos_wait_cond_var* is called.

Example

See later.

4.1.7.3 vos_signal_cond_var()

Syntax

```
void vos_signal_cond_var(vos_cond_var_t *cv);
```

Description

Signal the condition variable *cv*.

Parameters

cv
Pointer to a condition variable structure.

Returns

There is no return value.

Comments

This function works in conjunction with *vos_wait_cond_var* to provide thread synchronisation based on the value of data. Condition variables are always used in conjunction with a mutex. The mutex must be locked before *vos_signal_cond_var* is called. After signalling the condition variable, a thread that had previously called *vos_wait_cond_var* will be unblocked and made ready to run. The calling function must unlock the mutex to allow the *vos_wait_cond_var* operation in the unblocked thread to complete.

Example

See later.

4.1.8 Critical Sections

It is possible to define critical sections for when code must act atomically. This is done with the following definitions:

```
#define VOS_ENTER_CRITICAL_SECTION asm{SETI;};  
#define VOS_EXIT_CRITICAL_SECTION asm{CLRI;}
```

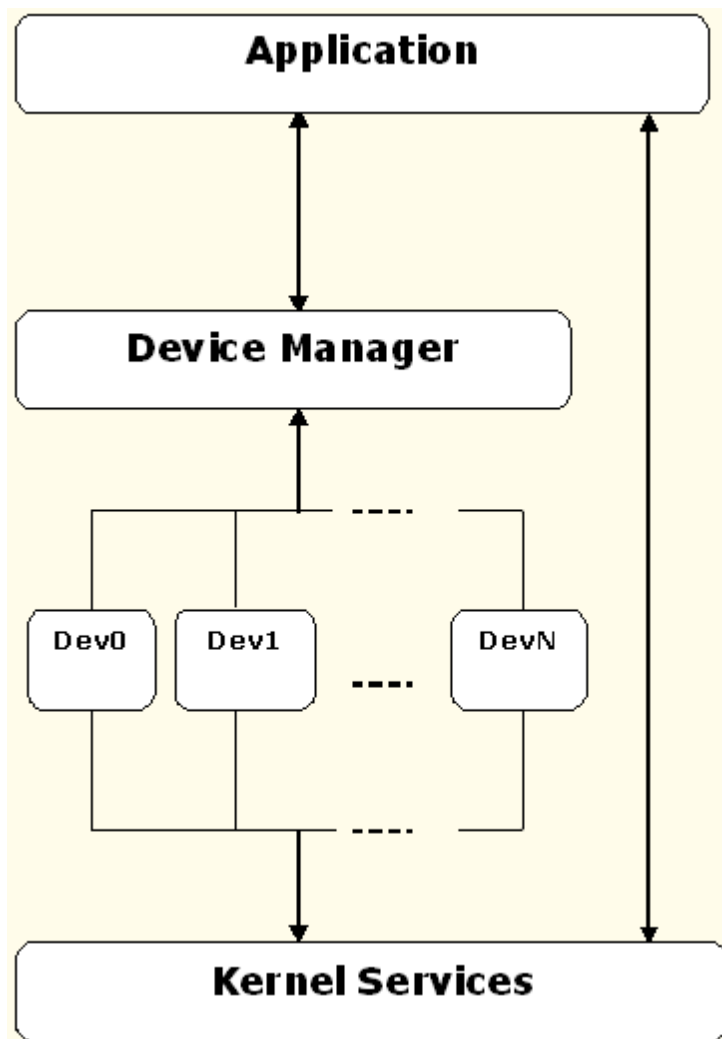
Example

For example, the following code will make sure that the sequence is not interrupted:

```
// find end of linked list  
VOS_ENTER_CRITICAL_SECTION  
while (pX->next)  
{  
    pX = pX->next;  
}  
VOS_EXIT_CRITICAL_SECTION
```

4.1.9 Device Manager

Devices are controlled by drivers and are managed by the device manager in the kernel. The device manager is layered below the application and presents a standard interface to the devices in the system.



The device manager controls access to the drivers for each device. Typically, an application opens a device by calling the device manager function [vos_dev_open\(\)](#) and obtains a handle that is used to represent the device in all subsequent device accesses. The application sends requests to its devices through the device manager using the functions [vos_dev_read\(\)](#), [vos_dev_write\(\)](#) and [vos_dev_ioctl\(\)](#). The device manager uses the handle to route the requests to the appropriate device.

Drivers are allocated a number by the application. The application specifies the number of drivers in the call to [vos_init\(\)](#). This number must be unique for each instance of a driver, it must start at zero and each driver must be numbered contiguously. All allocated drivers must be initialised using [vos_dev_init\(\)](#) before the scheduler is started or any interrupt is enabled with [vos_enable_interrupts\(\)](#).

The stages for using the device manager are:

- Driver Initialisation
 - Registering memory for device storage
 - Registering each driver
- Driver Operation
 - Opening Drivers
 - Sending Read/Write or IOCTL Operations to the Drivers
 - Closing Drivers

4.1.9.1 Driver Initialisation

The device manager must be told how many devices are present in the system with the `vos_init()` call before any devices are initialised by `vos_dev_init()`.

Once when drivers are initialised it is possible to access a driver. Driver initialisation will normally be handled by the driver rather than an application. Each driver should supply an initialisation function for this purpose.

All devices must be initialised with the `vos_dev_init()` function before the scheduler is started with `vos_start_scheduler()`. In the following structure function pointers for driver entry points must be filled out by the driver before registering with `vos_dev_init()`.

```
typedef struct _vos_driver_t {
    PF_OPEN open;           // dev_open()
    PF_CLOSE close;         // dev_close()
    PF_IO read;             // dev_read()
    PF_IO write;            // dev_write()
    PF_IOCTL ioctl;         // dev_ioctl()
    PF_INT interrupt;       // interrupt routine
    uint8 flags;            // miscellaneous flags
} vos_driver_t;
```

The open, close and interrupt function pointers are optional and if no open or close function is required should be set to NULL. The read, write and ioctl function pointers are only required if that function is to be available for calling. These should be set to NULL if the function is not supported.

The flags member is reserved for future use.

FTDI supplied drivers supply interrupt handlers when required and will manage all interrupt enabling and disabling. It is not necessary to use interrupts for layered and non-hardware device drivers.

4.1.9.1.1 vos_dev_init()

Syntax

```
void vos_dev_init(uint8 dev_num, vos_driver_t *driver_cb, void *context);
```

Description

Initialise a device driver and add an optional context pointer.

Parameters

`dev_num`

This parameter specifies the index of the device in the driver control block.

`driver_cb`

A completed driver control block must be passed in the `driver_cb` parameter. This is used by the device manager for calling driver entry points and must be persistent storage (i.e. not a local variable in a function).

`context`

An optional value may be specified in the context parameter to allow a driver to differentiate different instances or be configured with external data. The parameter is a void pointer allowing any driver specific data to be passed.

Returns

The function does not return any value.

Comments

All drivers must be initialised with this call before the scheduler starts.

Example

```
#define NUMBER_OF_DEVICES 2
#define VOS_DEV_TEST1     0
#define VOS_DEV_TEST2     1
```

```
vos_driver_t cb_test;
test_context tctx1;
test_context tctx2;

void test_init()
{
    cb_test.open = test_open;
    cb_test.close = test_close;
    cb_test.read = test_read;
    cb_test.write = test_write;
    cb_test.ioctl = test_ioctl;
    cb_test.interrupt = NULL;

    // the same driver control block can be used for multiple drivers
    vos_dev_init(VOS_DEV_TEST1, &cb_test, &tctx1);
    vos_dev_init(VOS_DEV_TEST2, &cb_test, &tctx2);
}

void main(void)
{
    vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);

    test_init();

    start_scheduler();
}
```

4.1.9.1.2 vos_enable_interrupts() vos_disable_interrupts()

Syntax

```
void vos_enable_interrupts(uint32 mask);
void vos_disable_interrupts(uint32 mask);
```

Description

Enable or disable hardware interrupts.

Parameters

mask
The interrupts to enable or disable are specified by the mask parameter.

Returns

The functions return no values.

Comments

Interrupts should only be enabled after all devices have been initialised with vos_dev_init(). An interrupt handler must be present for each hardware interrupt enabled. The mask parameter may have one of the following values. Values may be combined by bitwise or operation.

```
VOS_UART_INT_IEN
VOS_USB_0_DEV_INT_IEN
VOS_USB_1_DEV_INT_IEN
VOS_USB_0_HC_INT_IEN
VOS_USB_1_HC_INT_IEN
VOS_GPIO_INT_IEN
VOS_SPI_MASTER_INT_IEN
VOS_SPI_0_SLAVE_INT_IEN
VOS_SPI_1_SLAVE_INT_IEN
VOS_PWM_TOP_INT_IEN
VOS_FIFO_245_INT_IEN
```

Interrupts are not required for drivers which do not directly control hardware interfaces. Therefore,

interrupt enabling and disabling is not required for drivers which are layered on top of hardware drivers. FTDI supplied hardware device drivers control and handle all aspects of interrupts. These functions are documented only to allow advanced use of the FTDI supplied drivers where it may be beneficial to suspend interrupt handling for a driver for a short period of time.

4.1.9.2 Driver Operation

When a driver is opened a handle is obtained:

- Drivers are opened for exclusive access
- The handle is valid until the driver is closed
- Once it is closed it may be reopened

Read, Write and IOCTL calls will return a driver specific status value:

- This is usually zero for success and non-zero for other outcomes

Hardware interrupt handling is taken care of in the FTDI supplied drivers therefore there are no interrupt handlers require to be written. Interrupts cannot be triggered on layered drivers or drivers which do not directly control hardware.

Example

First initialise the driver. This must be done in the main() function before the scheduler starts.

Open the driver to obtain a handle. This must be done after the scheduler starts if the driver generates interrupts.

Setup the driver as required. This is normally accomplished with IOCTL calls.

Send read and write commands to the driver.

Close the driver when it is no longer required.

```
void thread3(void){
    /* handle to UART driver */
    VOS_HANDLE hUart;
    /* UART IOCTL request block */
    uart_ioctl_cb uart_iocb;
    /* string to display (include space for terminating NULL */
    char hello[] = {'H','e','l','l','o','\r'};
    unsigned short len;

    /* find and open UART device */
    hUart = vos_dev_open(VOS_DEV_UART);
    /* set baud rate to 9600 baud */
    uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_BAUD_RATE;
    uart_iocb.buf_in.baud_rate = UART_BAUD_9600;
    vos_dev_ioctl(hUart,&uart_iocb);
    vos_dev_write(hUart, (unsigned)hello, 6, &len);
    vos_dev_close(hUart);
}
```

Do not call any device manager operations until all devices have been initialised with [vos_dev_init\(\)](#). Doing so may result in undefined behaviour.

Avoid using the device manager operations until the scheduler is started running with [vos_start_scheduler\(\)](#).

4.1.9.2.1 vos_dev_open()

Syntax

```
VOS_HANDLE vos_dev_open(uint8 dev_number);
```

Description

Open a device for subsequent access.

Parameters

`dev_number`

The device number allocated to the driver in the `vos_dev_init()` function call.

Returns

`VOS_HANDLE` a handle to the device which must be used for accessing the device.

Comments

When the device has been opened successfully, the caller has exclusive access to the device.

Example

This function returns a handle that must be used in subsequent device accesses.

The following code fragment shows how to open a device.

```
#define VOS_DEV_UART    4
VOS_HANDLE hUart;
hUart = vos_dev_open(VOS_DEV_UART);
```

4.1.9.2.2 `vos_dev_close()`

Syntax

```
void vos_dev_close(VOS_HANDLE h);
```

Description

Close a device.

Parameters

`h`
A `VOS_HANDLE` obtained previously from a call to `vos_dev_open`.

Returns

There is no return value.

Comments

Example

The following code fragment shows how to close a device.

```
#define VOS_DEV_UART    4

VOS_HANDLE hUart;

hUart = vos_dev_open(VOS_DEV_UART);
...
vos_dev_close(hUart);
```

4.1.9.2.3 `vos_dev_read()`

Syntax

```
uint8 vos_dev_read(VOS_HANDLE h,uint8 *buf,uint16 num_to_read,uint16 *num_read);
```

Description

Read data from a device.

Parameters

h
A `VOS_HANDLE` obtained previously from a call to `vos_dev_open`.

buf
Contains a pointer to storage for the data to be read.

num_to_read
Contains the maximum number of bytes to be read.

num_read
A pointer to a location to store the actual number of bytes read.

Returns

0 on success, otherwise a driver specific error code.

Comments

The device manager routes this request to the read function of the device that is represented by the handle.

4.1.9.2.4 `vos_dev_write()`

Syntax

```
uint8 vos_dev_write(VOS_HANDLE h,uint8 *buf,uint16 num_to_write,uint16 *num_written);
```

Description

Read data from a device.

Parameters

h
A `VOS_HANDLE` obtained previously from a call to `vos_dev_open`.

buf
Contains a pointer to the data to be written.

num_to_write
Contains the number of bytes to be written.

num_written
A pointer to a location to store the actual number of bytes written.

Returns

0 on success, otherwise a driver specific error code.

Comments

The device manager routes this request to the write function of the device that is represented by the handle.

4.1.9.2.5 `vos_dev_ioctl()`

Syntax

```
uint8 vos_dev_ioctl(VOS_HANDLE h,void *cb);
```

Description

Send a control request to a device.

Parameters

h
A `VOS_HANDLE` obtained previously from a call to `vos_dev_open`.

cb

Contains a pointer to the control block for the request.

Returns

0 on success, otherwise a driver specific error code.

Comments

The device manager routes this request to the ioctl function of the device that is represented by the handle. The format of the control block is device-specific.

4.1.10 Hardware Information and Control

The kernel provides several functions for obtaining information about the CPU and controlling the behaviour of the CPU.

Default is 48MHz but can be changed by application if required

Allowable values of 48MHz, 24MHz or 12MHz

4.1.10.1 vos_set_clock_frequency() and vos_get_clock_frequency()

Syntax

```
void vos_set_clock_frequency(uint8 frequency);  
uint8 vos_get_clock_frequency(void);
```

Description

Initialise the CPU clock frequency

Parameters

frequency
The new clock frequency for the CPU is specified by the frequency parameter in vos_set_clock_frequency().

Returns

The vos_get_clock_frequency() function returns the current clock frequency of the CPU.

Comments

The only valid values for the frequency are:

```
VOS_48MHZ_CLOCK_FREQUENCY  
VOS_24MHZ_CLOCK_FREQUENCY  
VOS_12MHZ_CLOCK_FREQUENCY
```

Note: If the specified clock frequency is invalid in vos_set_clock_frequency() then it will default to 48MHz.

4.1.10.2 vos_get_package_type()

Syntax

```
uint8 vos_get_package_type(void);
```

Description

Determine the package type of the device.

Parameters

There are no parameters.

Returns

The `vos_get_package_type()` function returns the package type of the device.

Comments

The values returned by the function are:

```
VINCULUM_II_32_PIN  
VINCULUM_II_48_PIN  
VINCULUM_II_64_PIN
```

4.1.10.3 `vos_get_chip_revision()`

Syntax

```
uint8 vos_get_chip_revision(void);
```

Description

Find the revision information for the device.

Parameters

There are no parameters.

Returns

The `vos_get_chip_revision()` function returns a single byte which includes the chip revision in the high nibble and chip ID in the low nibble.

Comments

Currently the only valid value returned by this function is 0x11.

4.1.10.4 `vos_power_down()`

Syntax

```
uint8 vos_power_down(uint8 wakeMask);
```

Description

Power down the CPU into a low power sleep mode. Wait until an event occurs.

Parameters

```
wakeMask  
    Bit mask specifying event or events which will wake the CPU.
```

Returns

0 on success, otherwise 1 for an invalid mask value.

Comments

The valid values of the `wakeMask` are:

```
VOS_WAKE_ON_USB_0  
VOS_WAKE_ON_USB_1  
VOS_WAKE_ON_UART_RI  
VOS_WAKE_ON_SPI_SLAVE_0  
VOS_WAKE_ON_SPI_SLAVE_1
```

4.1.10.5 `vos_halt_cpu()`

Syntax

```
void vos_halt_cpu(void);
```

Description

Halts the CPU. The CPU will cease to process instructions if this function is called.

Parameters

There are no parameters.

Returns

The function does not return any value.

Comments

This function can be useful for debugging. Resetting the VNC2 will run the program from the start again until reaching the `vos_halt_cpu()` function.

4.1.11 Kernel Services

In addition to the core kernel functions, there are kernel services that provide specialised functionality. Kernel services can be used in drivers and applications. The available kernel services are:

[DMA service](#)

This provides an interface for accessing and controlling the on-chip DMA engines. This is used extensively by the VOS device drivers, but the memory-memory mode could be used in user applications.

[IOMux service](#)

The IOMux service provides a simple mechanism for an application to route a specified signal to a particular pin. The IOMux service also provides functions to configure IO cell characteristics.

4.1.11.1 DMA Service

The DMA service provides access to VNC2 direct memory access (DMA) engines. There are 4 on-chip DMAs which this kernel service manages. DMA engines are used extensively by device drivers and are not likely to be used in user applications in modes other than memory-memory.

```
typedef struct _vos_dma_config_t {
    union {
        uint16 io_addr;
        uint8 *mem_addr;
    } src;
    union {
        uint16 io_addr;
        uint8 *mem_addr;
    } dest;
    uint16 bufsiz;
    uint8 mode;
    uint8 fifosize;
    uint8 flow_control;
    uint8 afull_trigger;
} vos_dma_config_t;
```

4.1.11.1.1 DMA Service Return Codes

Calls to the DMA kernel service may return one of the following status codes:

DMA_OK

The DMA request completed successfully.

DMA_INVALID_PARAMETER

An invalid parameter has been passed to the DMA function.

DMA_ACQUIRE_ERROR

Failed to acquire a DMA engine.

DMA_ENABLE_ERROR

Failed to enable the DMA engine.

DMA_CONFIGURE_ERROR

Failed to configure the DMA engine.

DMA_ERROR

Reserved.

DMA_FIFO_ERROR

Failed to retrieve data from the DMA FIFO buffer.

4.1.11.1.2 vos_dma_acquire()

Syntax

```
vos_dma_handle_t vos_dma_acquire(void);
```

Description

Acquire a DMA engine for subsequent use.

Parameters

None.

Returns

A handle to the DMA engine that has been acquired.

Comments

Since there are 4 DMA engines on-chip, they need to be shared between the various drivers and the user application. Where possible, it is recommended that an acquired DMA engine be released by calling [vos_dma_release\(\)](#) when the operation is complete. `vos_dma_acquire()` will block until a DMA engine becomes available to acquire.

4.1.11.1.3 vos_dma_release()

Syntax

```
void vos_dma_release(vos_dma_handle_t h);
```

Description

Release a DMA engine which was previously acquired with a call to [vos_dma_acquire\(\)](#).

Parameters

`h`
A handle to a DMA engine.

Returns

No return code is provided.

Comments

Once a DMA engine has been released, it will be available for acquisition by another module by calling [vos_dma_acquire\(\)](#).

4.1.11.1.4 vos_dma_configure()

Syntax

```
uint8 vos_dma_configure(vos_dma_handle_t h, vos_dma_config_t *cb);
```

Description

Configure a DMA engine which was previously acquired with a call to [vos_dma_acquire\(\)](#).

Parameters

- h**
A handle to a DMA engine.
- *cb**
A pointer to a DMA configuration structure. This specifies the operation that the DMA is intended to perform.

Returns

The return code is one of the [DMA status codes](#).

Comments

A DMA engine must be configured before it can be used for an operation. Once an operation is complete, the DMA engine can be re-configured for another operation by a subsequent call to this function.

4.1.11.1.5 vos_dma_retained_configure()

Syntax

```
uint8 vos_dma_retained_configure(vos_dma_handle_t h, uint8 *mem_addr, uint16 bufsiz);
```

Description

Perform minimal reconfiguration of a DMA engine which was previously acquired with a call to [vos_dma_acquire\(\)](#) and then fully configured with a call to [vos_dma_configure\(\)](#).

Parameters

- h**
A handle to a DMA engine.
- *mem_addr**
A pointer to a RAM data buffer.
- bufsiz**
The size of the RAM data buffer above.

Returns

The return code is one of the [DMA status codes](#).

Comments

Once a DMA operation is complete, the DMA engine may be released with a call to [vos_dma_release\(\)](#) or in some cases an additional DMA operation may be desired.

Provided that the DMA is to be used in the same mode as it has previously been configured for with a call to [vos_dma_configure\(\)](#), the DMA can be quickly re-configured with a new memory address and transfer size using the [vos_dma_retained_configure\(\)](#) function. This function is much faster than performing a full configuration of the DMA using [vos_dma_configure\(\)](#) and can provide performance benefits.

However, the 4 DMA engines in VNC2 are a shared resource and the decision to not release a DMA may have an impact on overall system performance.

4.1.11.1.6 vos_dma_enable()

Syntax

```
uint8 vos_dma_enable(vos_dma_handle_t h);
```

Description

Start a DMA operation which was specified with a call to [vos_dma_configure\(\)](#).

Parameters

h
A handle to a DMA engine.

Returns

The return code is one of the [DMA status codes](#).

Comments

Once a DMA operation has completed, an interrupt is signalled to the CPU. An application can be notified of completion by calling [vos_dma_wait_on_complete\(\)](#) which will block until the specified DMA engine has completed its processing.

4.1.11.1.7 vos_dma_disable()

Syntax

```
uint8 vos_dma_enable(vos_dma_handle_t h);
```

Description

Stop a running DMA operation which was started with [vos_dma_enable\(\)](#).

Parameters

h
A handle to a DMA engine.

Returns

The return code is one of the [DMA status codes](#).

Comments

Terminating a running DMA operation by calling `vos_dma_disable()` does not free the DMA for subsequent acquisition. To free the DMA engine, call [vos_dma_release\(\)](#).

4.1.11.1.8 vos_dma_wait_on_complete()

Syntax

```
void vos_dma_wait_on_complete(vos_dma_handle_t h);
```

Description

Block thread execution until the specified DMA engine has completed its current operation.

Parameters

h
A handle to a DMA engine.

Returns

No return code is provided.

Comments

An application is notified of a DMA operation completing by calling this function. When the function returns, the DMA can either be released by calling [vos_dma_release\(\)](#) or re-configured for a subsequent operation by calling [vos_dma_configure\(\)](#).

4.1.11.1.9 vos_dma_get_fifo_data_register()

Syntax

```
uint16 vos_dma_get_fifo_data_register(vos_dma_handle_t h);
```

Description

Obtain an identifier for the FIFO data register of a DMA engine in FIFO mode.

Parameters

h
A handle to a DMA engine in FIFO mode.

Returns

The return value is the identifier of the FIFO data register for the DMA engine with handle **h**.

Comments

The DMA engine FIFO mode of operation is only intended for use within VNC2 hardware device drivers and under normal circumstances would not ever be required in a user application.

The FIFO data register identifier is used with [vos_dma_configure\(\)](#) as an element of the **cb** structure.

4.1.11.1.10 vos_dma_get_fifo_flow_control()

Syntax

```
uint8 vos_dma_get_fifo_flow_control(vos_dma_handle_t h);
```

Description

Obtain a flow control value for a DMA engine in FIFO mode.

Parameters

h
A handle to a DMA engine in FIFO mode.

Returns

The return value is the flow control value for the DMA engine with handle **h**.

Comments

The DMA engine FIFO mode of operation is only intended for use within VNC2 hardware device drivers and under normal circumstances would not ever be required in a user application.

The FIFO flow control value is used with [vos_dma_configure\(\)](#) as an element of the **cb** structure.

4.1.11.1.11 vos_dma_get_fifo_count()

Syntax

```
uint16 vos_dma_get_fifo_count(vos_dma_handle_t h);
```

Description

Determine the number of bytes in the DMA engine's FIFO.

Parameters

h
A handle to a DMA engine in FIFO mode.

Returns

The return value is the number of bytes currently in the FIFO for the DMA engine with handle h.

Comments

The DMA engine FIFO mode of operation is only intended for use within VNC2 hardware device drivers and under normal circumstances would not ever be required in a user application.

`vos_dma_get_fifo_count()` can be called to determine the number of bytes in the DMA engine's FIFO before calling [vos_dma_get_fifo_data\(\)](#).

4.1.11.1.12 vos_dma_get_fifo_data()

Syntax

```
uint8 vos_dma_get_fifo_data(vos_dma_handle_t h,uint8 *dat);
```

Description

Determine the number of bytes in the DMA engine's FIFO.

Parameters

h

A handle to a DMA engine in FIFO mode.

*dat

A pointer to a variable to receive the data byte from the DMA engine's FIFO.

Returns

The return code is one of the [DMA status codes](#).

Comments

The DMA engine FIFO mode of operation is only intended for use within VNC2 hardware device drivers and under normal circumstances would not ever be required in a user application.

[vos_dma_get_fifo_count\(\)](#) can be called to determine the number of bytes in the DMA engine's FIFO before calling `vos_dma_get_fifo_data()`.

4.1.11.2 IOMux Service

VNC2 features several peripherals. Due to the packages that the IC is provided in, it is not possible to simultaneously route all of the signals for all of the on-chip peripherals to pins for connecting to external electronics.

To allow any of the peripherals to be used in conjunction with external devices, VNC2 uses an IO multiplexer (IOMux) to allow the user to route signals from the IC to the package pins for their specific application. A default configuration is specified for each package, but a simple API is supplied to allow the user to route signals as desired.

Note that there are restrictions on which pins a signal can be routed to.

In addition to signal routing, the IOMux allows an application to control the characteristics of each IO cell.

To prevent unintended reprogramming of the debug pin (pin 11) on VNC2, the pin is mapped to pin 0xC7 (199 decimal) in the IOMux Service. An attempt to route a signal to any other pin above the pin count for the current package will result in an error code being returned (IOMUX_INVALID_PIN_SELECTION).

4.1.11.2.1 IOMux Service Return Codes

All calls to the IOMux kernel service will return one of the following status codes:

IOMUX_OK

The signal routing request completed successfully.

IOMUX_INVALID_SIGNAL

The requested signal is outwith the available range.

IOMUX_INVALID_PIN_SELECTION

The requested pin is outwith the available range.

IOMUX_UNABLE_TO_ROUTE_SIGNAL

The requested signal could not be routed to the requested pin.

IOMUX_INVLAID_IOCELL_DRIVE_CURRENT

The requested IO cell drive current is invalid.

IOMUX_INVLAID_IOCELL_TRIGGER

The requested IO cell trigger value is invalid.

IOMUX_INVLAID_IOCELL_SLEW_RATE

The requested IO cell slew rate is invalid.

IOMUX_INVLAID_IOCELL_PULL

The requested IO cell pull value is invalid.

IOMUX_ERROR

An error occurred.

4.1.11.2.2 vos_iomux_define_input() and vos_iomux_define_output()

Syntax

```
uint8 vos_iomux_define_input(uint8 pin, uint8 signal);  
uint8 vos_iomux_define_output(uint8 pin, uint8 signal);
```

Description

Route the specified input or output signal to the specified pin.

Parameters

pin

The pin number that the requested signal should be routed to.

signal

The requested signal.

Returns

An IOMux request will always return one of the [IOMux status codes](#).

Comments

It is not possible to route every signal to every pin. Any given signal can be routed to every 4th IO pin on a package. The return code indicates if the requested routing has been successful.

4.1.11.2.3 vos_iomux_define_bidi()

Syntax

```
uint8 vos_iomux_define_input(uint8 pin, uint8 input_signal, uint8 output_signal);
```

Description

Route the specified input and output signals to the specified pin.

Parameters

pin

The pin number that the requested signal should be routed to.

input_signal

The requested input signal.

output_signal

The requested output signal.

Returns

An IOMux request will always return one of the [IOMux status codes](#).

Comments

This function is intended for use when routing pins for peripherals with bidirectional signals (FIFO, GPIO and SPI Master). All other signals should be routed as either input or output using the [vos_iomux_define_input](#) and [vos_iomux_define_output](#) functions.

Note that in the case of bidirectional GPIO signals the mask must be changed to input or output as required using the [VOS_IOCTL_GPIO_SET_MASK](#) IOCTL call.

It is not possible to route every signal to every pin. Any given signal can be routed to every 4th IO pin on a package. The return code indicates if the requested routing has been successful.

4.1.11.2.4 vos_iocell_get_config()

Syntax

```
uint8 vos_iocell_get_config(uint8 pin, uint8 *drive_current, uint8 *trigger, uint8 *slew_rate, ui
```

Description

Retrieve the IO cell configuration for the specified pin.

Parameters

pin
The pin number that the requested signal should be routed to.

drive_current
A pointer to the current drive strength setting.

trigger
A pointer to the current trigger setting.

slew_rate
A pointer to the current slew rate setting.

pull
A pointer to the current pull-up/pull-down setting.

Returns

An IOMux request will always return one of the [IOMux status codes](#).

Comments

This function retrieves the current configuration of the IO cell corresponding to the specified pin.

4.1.11.2.5 vos_iocell_set_config()

Syntax

```
uint8 vos_iocell_set_config(uint8 pin, uint8 drive_current, uint8 trigger, uint8 slew_rate, uint8
```

Description

Retrieve the IO cell configuration for the specified pin.

Parameters

pin
The pin number that the requested signal should be routed to.

`drive_current`
The drive strength to request.

`trigger`
The trigger value to request.

`slew_rate`
The slew rate setting to request.

`pull`
The pull-up/pull-down setting to request.

Returns

An IOMux request will always return one of the [IOMux status codes](#).

Comments

This function configures the IO cell corresponding to the specified pin as requested. If the specified pin is not available or an invalid parameter has been passed with the request an appropriate [IOMux status code](#) is returned.

4.2 FTDI Drivers

To facilitate communication between user applications and the hardware peripherals available on the VNC2 IC, FTDI provides device drivers which work with VOS. In addition to the hardware device drivers, FTDI provides function drivers which build upon the basic hardware device driver functionality for a specific purpose.

For example, drivers for standard USB device classes may be created which build upon the USB host hardware driver to implement a BOMS class, CDC, printer class or even a specific vendor class device driver.

4.2.1 Hardware Device Drivers

The VNC2 IC contains several peripheral devices which the CPU has access to. These hardware peripherals are:

- UART
- SPI Slave (x2)
- SPI Master
- Parallel FIFO
- Timers (x3)
- Pulse width modulators (x3)
- GPIOs (x40, spread over 5 ports)
- USB Host (x2)
- USB Slave (x2)

In order for applications to communicate with these peripherals, device drivers are required. Applications will communicate with the device drivers via a device manager.

4.2.1.1 UART, SPI and FIFO Drivers

The UART, SPI and FIFO drivers share a common calling interface. This consists of common IOCTL codes and structures providing a transport neutral method of using these interfaces. IOCTL options specifically targeted at one interface may be sent to the other interfaces without worrying about it getting misinterpreted by the other interface.

The read and write interfaces are identical, allowing data to be read and written in the same way.

Return codes are standardised with identical success codes and common error codes.

There are only sufficient DMA resources available to have DMA enabled for 3 from the 5 interfaces (UART, 2 SPI Slaves, 1 SPI Master and FIFO interface) to be open at the same time. Therefore DMA is not enabled by default on any of these interfaces. It is recommended that is an interface is to be

used then DMA is enabled.

The UART interface cannot be used above 115200 baud without DMA being enabled.

The SPI Slave and SPI Master can operate at frequencies up to one quarter of the CPU clock frequency. The SPI Master can go as low as 1/256th of the CPU clock frequency.

When using the SPI Master the chip select signals SS_0 and SS_1 must be set using the [IOCTL](#) operation. They do not toggle automatically when data is read or written.

Read operations from the SPI Master MUST be preceded by a write operation of exactly the same size as the read operation. The way the SPI Master driver works is that data can only be clocked into the chip only when a write occurs. If not enough data is waiting to be read then the driver will block. Multiple write operations may be performed, up to the driver's buffer size, before the data need be read from the driver. Likewise multiple read operations may be performed until all data in the read buffer is processed.

Driver Hierarchy

UART Driver hierarchy:

UART Driver
VOS Kernel
UART Hardware

The [uart_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

SPI Slave Driver hierarchy:

SPI Slave Driver
VOS Kernel
SPI Slave Hardware

The [spislave_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

SPI Master Driver hierarchy:

SPI Master Driver
VOS Kernel
SPI Master Hardware

The [spimaster_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

FIFO Driver hierarchy:

FIFO Driver
VOS Kernel
FIFO Hardware

The [fifo_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

Library Files

UART.a

SPISlave.a

SPIMaster.a

FIFO.a

Header Files

UART.h

SPISlave.h

SPIMaster.h

FIFO.h

4.2.1.1.1 Common Read and Write Operations

Syntax

```
vos_dev_read(VOS_HANDLE h, unsigned char *buffer,  
             unsigned short len, unsigned short *read);  
vos_dev_write(VOS_HANDLE h, unsigned char *buffer,  
              unsigned short len, unsigned short *written);
```

Description

The UART, SPI and FIFO interfaces present the same read and write interfaces. All read and write operations block until the required number of bytes have been sent or received.

Parameters

h
A handle to the device used for input or output. This device must be initialised and opened.

buffer
Pointer to a buffer from which to send data to the device (read) or to receive data from the device (write).

len
Number of bytes to transfer to or from the buffer. The operation will block until the number of bytes are transferred.

read
written
Optional parameter to inform the calling function how many bytes were read from or written to the device. This may be less than the number of bytes requested in the len parameter if there is an error.

This parameter may be NULL, in which case the value is not updated.

Returns

An interface specific return code. See the return code section of the driver. All success error messages are the same value.

Example

```
// test buffer  
char buf[64];  
unsigned short num_read;  
unsigned short num_written;  
  
while (1)  
{  
    uart_iocb.ioctl_code = VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS;  
    vos_dev_ioctl(hTest, &uart_iocb);  
    num_written = uart_iocb.get.queue_stat;  
  
    // limit to 64 bytes per transaction  
    if (num_written > 64)  
        num_written = 64;  
  
    if (num_written)  
    {  
        if (vos_dev_read(hIn, buf, num_written, &num_read) == UART_OK)
```

```
{
    if (num_read)
    {
        if (vos_dev_write(hOut, buf, num_read, &num_written) == UART_OK)
        {
            // success
        }
    }
}
} while (1);
```

4.2.1.1.2 Common IOCTL Calls

Calls to the IOCTL functions for the UART, SPI and FIFO interfaces take the form:

```
typedef struct _common_ioctl_cb_t {
    unsigned char ioctl_code;
    union {
        unsigned long uart_baud_rate;
        unsigned long spi_master_sck_freq;
        unsigned char param;
        void * data;
    } set;
    union {
        unsigned long spi_master_sck_freq;
        unsigned short queue_stat;
        unsigned char param;
        void * data;
    } get;
} common_ioctl_cb_t;
```

The common codes supported by all interfaces are:

VOS_IOCTL_COMMON_RESET	Reset the interface
VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS	Get the number of bytes in the receive buffer
VOS_IOCTL_COMMON_GET_TX_QUEUE_STATUS	Get the number of bytes in the transmit buffer
VOS_IOCTL_COMMON_ENABLE_DMA	Acquire DMA channels and disable interrupts
VOS_IOCTL_COMMON_DISABLE_DMA	Release DMA channels and enable interrupts

4.2.1.1.2.1 VOS_IOCTL_COMMON_RESET

Description

This IOCTL will perform a hardware reset of the interface.

Parameters

There are no other parameters to set.

Returns

There is no data returned.

The [vos_dev_ioctl\(\)](#) call will always return a code indicating successful transaction.

Example

```
spi_ioctl.ioctl_code = VOS_IOCTL_COMMON_RESET;
vos_dev_ioctl(hSPI, &spi_ioctl);
```

4.2.1.1.2.2 VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS

Description

Returns the number of bytes in the receive queue.

Parameters

There are no parameters to set.

Returns

The number of bytes in the receive buffer is returned in the `queue_stat` member of the get section of the IOCTL structure.

The [vos_dev_ioctl\(\)](#) call will always return a code indicating successful transaction.

Example

```
common_ioctl_cb_t uart_iocb; // UART iocb for getting bytes available.
unsigned int dataAvail = 0; // How much data is available to be read?

uart_iocb.ioctl_code = VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS;
vos_dev_ioctl(hMonitor, &uart_iocb);

dataAvail = uart_iocb.get.queue_stat; // How much data to read?
```

4.2.1.1.2.3 VOS_IOCTL_COMMON_GET_TX_QUEUE_STATUS

Description

Returns the number of bytes in the transmit queue.

Parameters

There are no parameters to set.

Returns

The number of bytes in the transmit buffer is returned in the `queue_stat` member of the get section of the IOCTL structure.

The [vos_dev_ioctl\(\)](#) call will always return a code indicating successful transaction.

Example

```
common_ioctl_cb_t uart_iocb; // UART iocb for getting bytes waiting to be sent.
unsigned int dataAvail = 0; // How much data is waiting in the queue?

uart_iocb.ioctl_code = VOS_IOCTL_COMMON_GET_TX_QUEUE_STATUS;
vos_dev_ioctl(hMonitor, &uart_iocb);

dataAvail = uart_iocb.get.queue_stat; // How much data is there?
```

4.2.1.1.2.4 VOS_IOCTL_COMMON_ENABLE_DMA

Description

This IOCTL will switch the interface from interrupt mode to DMA mode.

Parameters

There are no parameters to set.

Returns

The function returns no data.

The [vos_dev_ioctl\(\)](#) call will return a code indicating successful transaction if there are sufficient DMA resources otherwise it will indicate that DMA was not enabled.

Example

```
uart_iocb.ioctl_code = VOS_IOCTL_COMMON_ENABLE_DMA;  
vos_dev_ioctl(hMonitor,&uart_iocb);
```

4.2.1.1.2.5 VOS_IOCTL_COMMON_DISABLE_DMA

Description

This IOCTL will switch the interface from DMA mode to interrupt mode.

Parameters

There are no parameters to set.

Returns

The function returns no data.

The [vos_dev_ioctl\(\)](#) call will return a code indicating successful transaction if the DMA resources were allocated otherwise it will indicate an invalid parameter.

Example

```
uart_iocb.ioctl_code = VOS_IOCTL_COMMON_DISABLE_DMA;  
vos_dev_ioctl(hMonitor,&uart_iocb);
```

4.2.1.1.3 UART Driver

4.2.1.1.3.1 UART Return Codes

All calls to the UART driver will return one of the following status codes.

- UART_OK
The command completed successfully.
- UART_INVALID_PARAMETER
There was an error or problem with a parameter sent to the driver.
- UART_DMA_NOT_ENABLED
A DMA operation was requested when DMA was not enabled.
- UART_ERROR
An unspecified error occurred.

4.2.1.1.3.2 UART IOCTL Calls

The following IOCTL request codes are supported by the UART driver.

VOS_IOCTL_UART_GET_MODEM_STATUS	Get the modem status bits
VOS_IOCTL_UART_GET_LINE_STATUS	Get the line status
VOS_IOCTL_UART_SET_BAUD_RATE	Set the baud rate
VOS_IOCTL_UART_SET_FLOW_CONTROL	Set flow control
VOS_IOCTL_UART_SET_DATA_BITS	Set the number of data bits
VOS_IOCTL_UART_SET_STOP_BITS	Set the number of stop bits
VOS_IOCTL_UART_SET_PARITY	Set the parity
VOS_IOCTL_UART_SET_RTS	Assert the RTS line

<u>VOS_IOCTL_UART_CLEAR_RTS</u>	Deassert the RTS line
<u>VOS_IOCTL_UART_SET_DTR</u>	Assert the DTR line
<u>VOS_IOCTL_UART_CLEAR_DTR</u>	Deassert the DTR line
<u>VOS_IOCTL_UART_SET_BREAK_ON</u>	Set the line break condition
<u>VOS_IOCTL_UART_SET_BREAK_OFF</u>	Clear the line break condition
<u>VOS_IOCTL_UART_SET_XON_CHAR</u>	Set the XON character
<u>VOS_IOCTL_UART_SET_XOFF_CHAR</u>	Set the XOFF character
<u>VOS_IOCTL_UART_WAIT_ON_MODEM_STATUS_INT</u>	Wait on a transmit status interrupt
<u>VOS_IOCTL_UART_WAIT_ON_LINE_STATUS_INT</u>	Wait on a line status interrupt

Description

Get the modem status. This is the CTS, DSR, RI lines and the DCD function.

Parameters

There are no other parameters to set.

Returns

A bit map of the modem signals in the param member of get is returned:

```
UART_MODEM_STATUS_CTS
UART_MODEM_STATUS_DSR
UART_MODEM_STATUS_DCD
UART_MODEM_STATUS_RI
```

Example

```
//wait for either CTS or DSR to be asserted
do
{
    uart_iocb.ioctl_code = VOS_IOCTL_UART_GET_MODEM_STATUS;
    uart_iocb.get.param = 0;
    vos_dev_ioctl(hMonitor,&uart_iocb);

    uart_iocb.get.param &= (UART_MODEM_STATUS_CTS | UART_MODEM_STATUS_DSR);
    if (uart_iocb.get.param != (UART_MODEM_STATUS_CTS | UART_MODEM_STATUS_DSR))
    {
        break;
    }
} while (1);
```

Description

Get the line status.

Parameters

There are no other parameters to set.

Returns

The line status is returned as a bit map in the param member of get:

```
UART_LINE_STATUS_OE
UART_LINE_STATUS_PE
UART_LINE_STATUS_FE
UART_LINE_STATUS_BI
```

Example

```
uart_iocb.ioctl_code = VOS_IOCTL_UART_GET_LINE_STATUS;
```

```
uart_iocb.get.param = 0;
vos_dev_ioctl(hMonitor,&uart_iocb);

uart_iocb.get.param &= (UART_MODEM_STATUS_CTS | UART_MODEM_STATUS_DSR);
if (uart_iocb.get.param != (UART_MODEM_STATUS_CTS | UART_MODEM_STATUS_DSR))
{
    break;
}
} while (1);
```

Description

Set the baud rate. For non-standard baud rates, the UART driver will calculate the closest possible baud rate.

The baud rate calculation is based on the CPU clock frequency. If the CPU clock frequency is changed after the baud rate has been set then it must be set again to obtain the correct baud rate.

Parameters

Set the desired baud rate in the `baud_rate` member of `set`. No other fields need to be filled out.

Predefined values are available for:

```
UART_BAUD_300
UART_BAUD_600
UART_BAUD_1200
UART_BAUD_2400
UART_BAUD_4800
UART_BAUD_9600
UART_BAUD_19200
UART_BAUD_38400
UART_BAUD_57600
UART_BAUD_115200
UART_BAUD_256000
UART_BAUD_500000
UART_BAUD_1000000
UART_BAUD_1500000
UART_BAUD_2000000
UART_BAUD_3000000
```

Returns

If the baud rate cannot be set within an accuracy of +/-3% then `UART_ERROR` is returned.

Example

```
/* UART setup */
/* set baud rate to 9600 baud */
uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_BAUD_RATE;
uart_iocb.set.uart_baud_rate = UART_BAUD_9600;
vos_dev_ioctl(hMonitor,&uart_iocb);
```

Description

Set the flow control scheme.

Parameters

Set the desired baud rate in the `param` member of `set`. No other fields need to be filled out.

Available flow control methods are:

```
UART_FLOW_NONE
UART_FLOW_RTS_CTS
UART_FLOW_DTR_DSR
UART_FLOW_XON_XOFF
```

Returns

If the parameter is incorrect then UART_INVALID_PARAMETER will be returned.

Example

```
/* set flow control */
uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_FLOW_CONTROL;
uart_iocb.set.param = UART_FLOW_RTS_CTS;
vos_dev_ioctl(hMonitor,&uart_iocb);
```

Description

Set the number of data bits.

Parameters

Set the desired baud rate in the param member of set. No other fields need to be filled out

This can be set to:

```
UART_DATA_BITS_7
UART_DATA_BITS_8
```

Returns

If the parameter is incorrect then UART_INVALID_PARAMETER will be returned.

Example

```
/* set data bits */
uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_DATA_BITS;
uart_iocb.set.param = UART_DATA_BITS_8;
vos_dev_ioctl(hMonitor,&uart_iocb);
```

Description

Set the number of stop bits.

Parameters

Set the desired baud rate in the param member of set. No other fields need to be filled out

This can be set to:

```
UART_STOP_BITS_1
UART_STOP_BITS_2
```

Returns

If the parameter is incorrect then UART_INVALID_PARAMETER will be returned.

Example

```
/* set stop bits */
uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_STOP_BITS;
uart_iocb.set.param = UART_STOP_BITS_1;
vos_dev_ioctl(hMonitor,&uart_iocb);
```

Description

Set the parity.

Parameters

Set the desired baud rate in the param member of set. No other fields need to be filled out.

This can be set to:

```
UART_PARITY_NONE
UART_PARITY_ODD
UART_PARITY_EVEN
UART_PARITY_MARK
UART_PARITY_SPACE
```

Returns

If the parameter is incorrect then UART_INVALID_PARAMETER will be returned.

Example

```
/* set parity */  
uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_PARITY;  
uart_iocb.set.param = UART_PARITY_NONE;  
vos_dev_ioctl(hMonitor,&uart_iocb);
```

Description

Enables the RTS line to be controlled by the flow control if CTS/RTS is selected for flow control.

Parameters

No fields in the ioctl structure need to be filled out.

Returns

If the parameter is incorrect then UART_INVALID_PARAMETER will be returned.

Description

Unconditionally deassert the RTS line.

Parameters

No fields in the ioctl structure need to be filled out.

Returns

If the parameter is incorrect then UART_INVALID_PARAMETER will be returned.

Description

Enables the DTR line to be controlled by the flow control if DTR/DSR is selected for flow control.

Parameters

No fields in the ioctl structure need to be filled out.

Returns

If the parameter is incorrect then UART_INVALID_PARAMETER will be returned.

Description

Unconditionally deassert the DTR line.

Parameters

No fields in the ioctl structure need to be filled out.

Returns

If the parameter is incorrect then UART_INVALID_PARAMETER will be returned.

Description

Set line break condition

Parameters

No fields in the ioctl structure need to be filled out.

Returns

If the parameter is incorrect then UART_INVALID_PARAMETER will be returned.

Description

Clear line break condition

Parameters

No fields in the ioctl structure need to be filled out.

Returns

If the parameter is incorrect then UART_INVALID_PARAMETER will be returned.

Description

Set the XON character to be used for UART_FLOW_XON_XOFF.

Parameters

Set the desired character in the param member of set. No other fields need to be filled out.

Returns

The call does not return any value.

Description

Set the Xoff character to be used with UART_FLOW_XON_XOFF

Parameters

Set the desired character in the param member of set. No other fields need to be filled out

Returns

The call does not return any value.

Description

Wait on a modem status interrupt (CTS, DSR, RI, DCD, BUSY). Note that a call with this IOCTL code will not return until a change in the modem status occurs.

Parameters

No other fields in the ioctl structure need to be filled out.

Returns

A bit map of the modem signals in the param member of get is returned:

```
UART_MODEM_STATUS_CTS
UART_MODEM_STATUS_DSR
UART_MODEM_STATUS_DCD
UART_MODEM_STATUS_RI
```

Description

Wait on a line status interrupt (OE, PE, SE, BI) Note that a call with this IOCTL code will not return until a change in the line status occurs.

Parameters

No other fields in the `ioctl` structure need to be filled out.

Returns

The line status is returned in the `param` member of `get`:

```
UART_LINE_STATUS_OE
UART_LINE_STATUS_PE
UART_LINE_STATUS_FE
UART_LINE_STATUS_BI
```

4.2.1.1.3.3 `uart_init()`

Syntax

```
unsigned char uart_init (
    unsigned char devNum,
    uart_context_t* context
);
```

Description

Initialise the UART driver and registers the driver with the Device Manager.

Parameters

devNum
The device number to use when registering the driver with the Device Manager is passed in the `devNum` parameter.

context
The second parameter, `context`, is used to specify a buffer size for the receive and transmit buffers. If the `context` pointer is NULL then the default buffer size of 64 bytes is used.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The `context` parameter must be of the form of the structure defined below:

```
typedef struct _uart_context_t {
    unsigned char buffer_size;
} uart_context_t;
```

4.2.1.1.4 FIFO Driver

4.2.1.1.4.1 FIFO Return Codes

All calls to the FIFO driver will return one of the following status codes.

FIFO_OK
The command completed successfully.

FIFO_INVALID_PARAMETER
There was an error or problem with a parameter sent to the driver.

FIFO_DMA_NOT_ENABLED
A DMA operation was requested when DMA was not enabled.

FIFO_ERROR
An unspecified error occurred.

4.2.1.1.4.2 FIFO IOCTL Calls

The following IOCTL request codes are supported by the FIFO driver.

<u>VOS_IOCTL_FIFO_GET_STATUS</u>	Get the FIFO status
<u>VOS_IOCTL_FIFO_SET_MODE</u>	Set the FIFO mode

Description

Get the FIFO status.

Parameters

There are no parameters to set.

Returns

This is returned as a bit map of the FIFO status in the param member of get:

```
FIFO_STATUS_READ_NOT_FULL
```

Example

```
//wait for either FIFO status to be not full
do
{
    fifo_iocb.ioctl_code = VOS_IOCTL_FIFO_GET_STATUS;
    fifo_iocb.get.param = 0;
    vos_dev_ioctl(hMonitor,&fifo_iocb);

    if (uart_iocb.get.param != (FIFO_STATUS_READ_NOT_FULL))
    {
        break;
    }
} while (1);
```

Description

Set the FIFO mode to be synchronous or asynchronous.

Parameters

The mode is set in the param member of set:

```
FIFO_MODE_ASYNCHRONOUS
FIFO_MODE_SYNCHRONOUS
```

Returns

If the parameter is invalid then FIFO_INVALID_PARAMETER is returned.

Example

```
fifo_iocb.ioctl_code = VOS_IOCTL_FIFO_SET_MODE;
fifo_iocb.set.param = FIFO_MODE_SYNCHRONOUS;
vos_dev_ioctl(hMonitor,&fifo_iocb);
```

4.2.1.1.4.3 fifo_init()

Syntax

```
unsigned char fifo_init (
    unsigned char devNum,
    fifo_context_t* context
);
```

Description

Initialise the FIFO driver and registers the driver with the Device Manager.

Parameters

`devNum`

The device number to use when registering the driver with the Device Manager is passed in the `devNum` parameter.

`context`

The second parameter, `context`, is used to specify a buffer size for the receive and transmit buffers. If the `context` pointer is NULL then the default buffer size of 64 bytes is used.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The `context` parameter must be of the form of the structure defined below:

```
typedef struct _fifo_context_t {  
    unsigned char buffer_size;  
} fifo_context_t;
```

4.2.1.1.5 SPI Slave Driver

4.2.1.1.5.1 SPI Slave Return Codes

All calls to the SPI Slave driver will return one of the following status codes.

`SPISLAVE_OK`

The command completed successfully.

`SPISLAVE_INVALID_PARAMETER`

There was an error or problem with a parameter sent to the driver.

`SPISLAVE_DMA_NOT_ENABLED`

A DMA operation was requested when DMA was not enabled.

`SPISLAVE_ERROR`

An unspecified error occurred.

4.2.1.1.5.2 SPI Slave IOCTL Calls

The following IOCTL request codes are supported by the SPI Slave driver.

[VOS_IOCTL_SPI_SLAVE_GET_STATUS](#)

Get SPI Slave status

[VOS_IOCTL_SPI_SLAVE_SCK_CPHA](#)

Set the SCK phase

[VOS_IOCTL_SPI_SLAVE_SCK_CPOL](#)

Set the SCK polarity

[VOS_IOCTL_SPI_SLAVE_DATA_ORDER](#)

Set the data transmit order

[VOS_IOCTL_SPI_SLAVE_SET_ADDRESS](#)

Set the SPI slave address

[VOS_IOCTL_SPI_SLAVE_SET_MODE](#)

Set the SPI mode

Description

Not used in the SPI Slave Driver. Always returns zero.

Parameters

There are no parameters to set.

Returns

This returns zero in the param member of get.

Example

Description

Set the clock phase of the SPI Slave. Data can be clocked in on either the rising edge or falling edge of the clock.

Parameters

The phase is set in the param member of set:

```
SPI_SLAVE_SCK_CPHA_0
    Data is latched from SDI on the SPI clk leading edge and loaded onto SDO on the SPI clk
    trailing edge

SPI_SLAVE_SCK_CPHA_1
    Data is latched from SDI on the SPI clk trailing edge and loaded onto SDO on the SPI clk
    leading edge
```

Returns

If the parameter is invalid then SPISLAVE_INVALID_PARAMETER is returned.

Example

```
// set clock phase
spis_iocb.ioctl_code = VOS_IOCTL_SPI_SLAVE_SCK_CPHA;
spis_iocb.set.param = SPI_SLAVE_SCK_CPHA_0;
vos_dev_ioctl(hSpiSlave,&spis_iocb);
```

Description

Set the clock polarity of the SPI Slave. The clock input can be active high or low.

Parameters

The polarity is set in the param member of set:

```
SPI_SLAVE_SCK_CPOL_0
    Active high clk, SCK low in idle.

SPI_SLAVE_SCK_CPOL_1
    Active low clk, SCK high in idle.
```

Returns

If the parameter is invalid then SPISLAVE_INVALID_PARAMETER is returned.

Example

```
// set clock polarity
spis_iocb.ioctl_code = VOS_IOCTL_SPI_SLAVE_SCK_CPOL;
spis_iocb.set.param = SPI_SLAVE_SCK_CPOL_0;
vos_dev_ioctl(hSpiSlave,&spis_iocb);
```

Description

Set the data order of the SPI Slave. Data can be transmitted and received either MSB or LSB first.

Parameters

The data order is set in the param member of set:

```
SPI_SLAVE_DATA_ORDER_MSB
```

MSB transmitted first.

SPI_SLAVE_DATA_ORDER_LSB
LSB transmitted first.

Returns

If the parameter is invalid then SPISLAVE_INVALID_PARAMETER is returned.

Example

```
// set data order
spis_iocb.ioctl_code = VOS_IOCTL_SPI_SLAVE_DATA_ORDER;
spis_iocb.set.param = SPI_SLAVE_DATA_ORDER_MSB;
vos_dev_ioctl(hSpiSlave,&spis_iocb);
```

Description

Set the address of the SPI Slave. This can be in the range 0 to 7.

Parameters

The address is set in the param member of set.

Returns

If the parameter is invalid then SPISLAVE_INVALID_PARAMETER is returned.

Example

```
// set address of slave
spis_iocb.ioctl_code = VOS_IOCTL_SPI_SLAVE_SET_ADDRESS;
spis_iocb.set.param = 1;
vos_dev_ioctl(hSpiSlave,&spis_iocb);
```

Description

Set the operation mode of the SPI slave.

Parameters

The 5 modes of operation available are set in the param member of the set structure.

```
SPI_SLAVE_MODE_FULL_DUPLEX
SPI_SLAVE_MODE_HALF_DUPLEX_4_PIN
SPI_SLAVE_MODE_HALF_DUPLEX_3_PIN
SPI_SLAVE_MODE_UNMANAGED
SPI_SLAVE_MODE_VI_COMPATIBLE
```

Please refer to the data sheet for information about each mode. For compatibility with standard SPI implementations use the "unmanaged" mode. For compatibility with VNC1L applications use "V1 Compatible" mode.

Returns

If the parameter is invalid then SPISLAVE_INVALID_PARAMETER is returned.

Example

```
// set SPI mode
spis_iocb.ioctl_code = VOS_IOCTL_SPI_SLAVE_SET_MODE;
spis_iocb.set.param = SPI_SLAVE_MODE_VI_COMPATIBLE;
vos_dev_ioctl(hSpiSlave,&spis_iocb);
```

4.2.1.1.5.3 spislave_init()

Syntax

```
unsigned char spislave_init (
    unsigned char devNum,
```

```
    spislave_context_t* context  
};
```

Description

Initialise the SPI Slave driver and registers the driver with the Device Manager. There are 2 independent SPI Slaves. A separate driver is required for each SPI Slave.

Parameters

devNum
The device number to use when registering the driver with the Device Manager is passed in the devNum parameter.

context
The second parameter, context, is used to specify a buffer size for the receive and transmit buffers. If the context pointer is NULL then the default buffer size of 64 bytes is used on SPI Slave 0.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The context parameter must be of the form of the structure defined below:

```
typedef struct _spislave_context_t {  
    unsigned char slavenumber;  
    unsigned char buffer_size;  
} spislave_context_t;
```

The slavenumber member can be either:

```
SPI_SLAVE_0  
SPI_SLAVE_1
```

4.2.1.1.6 SPI Master Driver

4.2.1.1.6.1 SPI Master Return Codes

All calls to the SPI Master driver will return one of the following status codes.

SPIMASTER_OK
The command completed successfully.

SPIMASTER_INVALID_PARAMETER
There was an error or problem with a parameter sent to the driver.

SPIMASTER_DMA_NOT_ENABLED
A DMA operation was requested when DMA was not enabled.

SPIMASTER_ERROR
An unspecified error occurred.

4.2.1.1.6.2 SPI Master IOCTL Calls

The following IOCTL request codes are supported by the SPI Master driver.

<u>VOS_IOCTL_SPI_MASTER_GET_STATUS</u>	Get SPI Master status
<u>VOS_IOCTL_SPI_MASTER_SCK_CPHA</u>	Set the SCK phase
<u>VOS_IOCTL_SPI_MASTER_SCK_CPOL</u>	Set the SCK polarity
<u>VOS_IOCTL_SPI_MASTER_DATA_ORDER</u>	Set the data transmit order
<u>VOS_IOCTL_SPI_MASTER_SS_0</u>	Set the SPI master slave select 0

<u>VOS_IOCTL_SPI_MASTER_SS_1</u>	Set the SPI master slave select 1
<u>VOS_IOCTL_SPI_MASTER_SET_SCK_FREQUENCY</u>	Set the SPI master clock frequency
<u>VOS_IOCTL_SPI_MASTER_SET_DATA_DELAY</u>	Set the SPI master data delay between slave select and data transmission in clock cycles

Description

Not used in the SPI Master Driver. Always returns zero.

Parameters

There are no parameters to set.

Returns

This returns zero in the param member of get.

Description

Set the clock phase of the SPI Master. Data can be clocked out on either the rising edge or falling edge of the clock.

Parameters

The phase is set in the param member of set:

```
SPI_MASTER_SCK_CPHA_0
    Data is latched from SDI on the SPI clk leading edge and loaded onto SDO on the SPI clk
    trailing edge

SPI_MASTER_SCK_CPHA_1
    Data is latched from SDI on the SPI clk trailing edge and loaded onto SDO on the SPI clk
    leading edge
```

Returns

If the parameter is invalid then SPIMASTER_INVALID_PARAMETER is returned.

Example

```
// set clock phase
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SCK_CPHA;
spim_iocb.set.param = SPI_MASTER_SCK_CPHA_0;
vos_dev_ioctl(hSpiMaster,&spim_iocb);
```

Description

Set the clock polarity of the SPI Master. The clock can be active high or low.

Parameters

The polarity is set in the param member of set:

```
SPI_MASTER_SCK_CPOL_0
    Active high clk, SCK low in idle.

SPI_MASTER_SCK_CPOL_1
    Active low clk, SCK high in idle.
```

Returns

If the parameter is invalid then SPIMASTER_INVALID_PARAMETER is returned.

Example

```
// set clock polarity
```

```
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SCK_CPOL;
spim_iocb.set.param = SPI_MASTER_SCK_CPOL_0;
vos_dev_ioctl(hSpiMaster,&spim_iocb);
```

Description

Set the data order of the SPI Master. Data can be transmitted and received either MSB or LSB first.

Parameters

The data order is set in the param member of set:

`SPI_MASTER_DATA_ORDER_MSB`
MSB transmitted first.

`SPI_MASTER_DATA_ORDER_LSB`
LSB transmitted first.

Returns

If the parameter is invalid then `SPIMASTER_INVALID_PARAMETER` is returned.

Example

```
// set data order
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_DATA_ORDER;
spim_iocb.set.param = SPI_MASTER_DATA_ORDER_MSB;
vos_dev_ioctl(hSpiMaster,&spim_iocb);
```

Description

Set the slave select line zero or one either active (low) or disabled (high). These signals do not toggle automatically when data is read or written from the interface.

Parameters

The slave select signal is set in the param member of set:

`SPI_MASTER_SS_ENABLE`
Enable slave select signal (active low).

`SPI_MASTER_SS_DISABLE`
Disable the slave select signal.

Returns

If the parameter is invalid then `SPIMASTER_INVALID_PARAMETER` is returned.

Example

```
// set slave select 1 to enable
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SS_1;
spim_iocb.set.param = SPI_MASTER_SS_ENABLE;
vos_dev_ioctl(hSpiMaster,&spim_iocb);

// SS_1 is enabled so can read from device
vos_dev_write(hSpiMaster,&data,dataLen,NULL);

// set slave select 1 to disable
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SS_1;
spim_iocb.set.param = SPI_MASTER_SS_DISABLE;
vos_dev_ioctl(hSpiMaster,&spim_iocb);
```

Description

Set the SPI Master clock frequency. The divisor for the clock generator will be calculated and the actual frequency obtained for the SPI Master will be returned.

The clock frequency calculation is based on the CPU clock frequency. If the CPU clock frequency is changed after the SPI Master clock frequency is set then it must be set again to obtain the correct frequency.

Parameters

The requested clock frequency is set in the `spi_master_sck_freq` member of `set`. A frequency between 1/2 and 1/512 the CPU clock speed is allowed; this is equivalent to an SPI master clock divisor of between 0 and 255 since the maximum SPI master clock frequency is half the CPU clock frequency.

Returns

If the frequency requested is out of range then `SPIMASTER_INVALID_PARAMETER` is returned. The closest obtainable frequency to the requested frequency is returned in the `spi_master_sck_freq` member of the `get` structure.

Example

```
// set slave select 1 to disable
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SET_SCK_FREQUENCY;
spim_iocb.set.spi_master_sck_freq = 2000000;
vos_dev_ioctl(hSpiMaster,&spim_iocb);
```

Description

Set the number of clock periods to delay sending data after the slave select signal is asserted. This is only active when the slave select line is set close to the data send operation.

Parameters

The requested number of cycles to delay is set in the `param` member of `set`. Up to a maximum of 255 cycles.

Returns

If the number of cycles requested is out of range then `SPIMASTER_INVALID_PARAMETER` is returned.

Example

```
// set slave select 1 to disable
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SET_DATA_DELAY;
spim_iocb.set.param = 8;
vos_dev_ioctl(hSpiMaster,&spim_iocb);
```

4.2.1.1.6.3 spimaster_init()

Syntax

```
unsigned char spimaster_init (
    unsigned char devNum,
    spimaster_context_t* context
);
```

Description

Initialise the SPI Master driver and registers the driver with the Device Manager.

Parameters

devNum
The device number to use when registering the driver with the Device Manager is passed in the `devNum` parameter.

context
The second parameter, `context`, is used to specify a buffer size for the receive and transmit buffers. If the `context` pointer is NULL then the default buffer size of 64 bytes is used.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The context parameter must be of the form of the structure defined below:

```
typedef struct _spimaster_context_t {  
    unsigned char buffer_size;  
} spimaster_context_t;
```

4.2.1.2 USB Host Driver

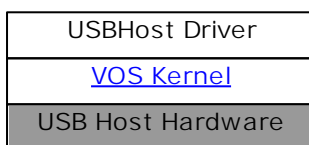
The USB Host driver consists of one driver instance which can control both USB host controllers on the VNC2.

The USB Host driver will build a list of device interfaces available on the configured USB Ports. This is maintained in memory structures and is searchable by an application. Requests to a device interface must be routed through the correct driver handle to the appropriate USB Port.

The [usbhost_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

Driver Hierarchy

USB Host Driver hierarchy:



Library Files

USBHost.a

Header Files

USBHost.h

USB.h (additional definitions for USB device classes and structures)

4.2.1.2.1 USB Host Concepts

Configuration

It can be configured to control either USB Port 1, USB Port 2 or both USB Ports. A unique VOS_DEVICE handle and device number is required for each configured interface. If a port is configured to be a USB Slave then it is not available for the USB Host.

Once the USB Host driver is configured it cannot be reconfigured.

Driver Handles

The USB Host driver will require 2 unique device numbers to register both USB Ports with the device manager. If only one USB Port is configured then only one device number is required.

If both USB Ports are configured then the application will have 2 driver handles when both ports are opened, one for each USB Port and effectively 2 device drivers active. They should be treated separately by the application.

Root Hub

The VNC2 hardware consists of two independent USB Root Hubs each with one port. These are controlled and managed by the USB Host driver. Control begins as soon as the USB Host driver is

configured.

The root hub can turn off or on the USB Ports.

When a device is connected to the root hub it will initiate an enumeration process to discover and configure devices on that USB Port.

If a device is removed from a USB Port then the root hub will remove references to all devices attached to that USB Port.

Downstream Hubs

Can be detected, configured and enumerated by the USB Host driver. Each downstream hub is scanned periodically to check for hub port status changes including device removals and connects.

Device Interfaces

Each USB device which is enumerated may have several interfaces on the device. Each interface is treated separately by the USB Host driver. A maximum number of device interfaces is specified in the USB Host configuration routine.

Endpoints

Device interfaces will have a number of endpoints associated with them although they may share the same control endpoint. A device interface can have control, bulk IN, bulk OUT, interrupt IN, interrupt OUT, isochronous IN and isochronous OUT endpoints.

4.2.1.2.1.1 USB Host Detecting Connections

The USB Host driver will automatically enumerate a USB bus when a device is connected to the Root Hub or a downstream hub.

To detect if there is a device connected to the Root Hub the application must perform an `VOS_IOCTL_USBHOST_GET_CONNECT_STATE` IOCTL call to the appropriate USB Port driver. This will return `PORT_STATE_ENUMERATED` if a device is enumerated. This IOCTL can be polled - with a short delay between polls.

4.2.1.2.1.2 USB Host Finding Device Interfaces

To find a device interface, the application must send one of the following IOCTL calls to the appropriate USB Port driver:

- `VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE` - find the first or next device interface in the list for the USB Port.
- `VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_VID_PID` - search through the device interface list for a device based on the VID and the PID of the device.
- `VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS` - search through the device interface list for a device based on the USB class, subclass or protocol.

These IOCTL calls return a handle to a device interface. Further information about each device interface can be found using the handle and one of these IOCTL calls:

- `VOS_IOCTL_USBHOST_DEVICE_GET_VID_PID` - get the VID and PID information from a device interface handle.
- `VOS_IOCTL_USBHOST_DEVICE_GET_CLASS_INFO` - get the USB class, subclass and protocol information from a device interface handle.
- `VOS_IOCTL_USBHOST_DEVICE_GET_DEV_INFO` - get general device information from a device interface handle.

The device interface handle is valid until a device is removed from the USB Port or the USB Port (or downstream hub) is re-enumerated.

4.2.1.2.1.3 USB Host Finding Endpoints

Starting from device interface handle the list of endpoints associated with the interface can be traversed. The following IOCTLs are used to find an appropriate endpoint type for a device interface:

-
- VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE
 - VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE
 - VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE
 - VOS_IOCTL_USBHOST_DEVICE_GET_INT_IN_ENDPOINT_HANDLE
 - VOS_IOCTL_USBHOST_DEVICE_GET_INT_IN_ENDPOINT_HANDLE
 - VOS_IOCTL_USBHOST_DEVICE_GET_ISO_IN_ENDPOINT_HANDLE
 - VOS_IOCTL_USBHOST_DEVICE_GET_ISO_IN_ENDPOINT_HANDLE

Once the first endpoint of a type is found, subsequent endpoints of that type can be found with the VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_ENDPOINT_HANDLE call.

The IOCTL call VOS_IOCTL_USBHOST_DEVICE_GET_ENDPOINT_INFO can be used to find information concerning the endpoint to establish if it is suitable for use by the application.

4.2.1.2.1.4 USB Host Sending and Receiving Data

When sending data to the read and write interfaces a special transfer structure is used. This structure differs slightly for general endpoints (control, bulk or interrupt) and isochronous endpoints.

The transfer request structure is filled out with the required information about the target endpoint, data location, data size and semaphore for completion notification. This structure is passed to the read or write function for the USB Host driver.

The completion information is returned in the same structure.

4.2.1.2.2 USB Host Return Codes

Status Codes

All calls to the USB Host driver will return one of the following status codes.

USBHOST_OK

No error.

USBHOST_NOT_FOUND

Device or endpoint not found. The device may have been disconnected.

USBHOST_PENDING

Transaction has been started but the result not available.

USBHOST_INVALID_PARAMETER

A parameter is incorrect or has a mistake.

USBHOST_INVALID_BUFFER

The calling function did not specify a valid buffer.

USBHOST_INCOMPLETE_ENUM

Enumeration did not complete.

USBHOST_INVALID_CONFIGURATION

The configuration of the host controller is invalid and cannot support the current operation.

USBHOST_TD_FULL

No more transaction descriptors available.

USBHOST_EP_FULL

No more endpoint descriptors available.

USBHOST_IF_FULL

No more interface descriptors available.

USBHOST_EP_HALTED

An attempt was made to access a halted endpoint.

USBHOST_EP_INVALID

An attempt was made to access an invalid endpoint.

USBHOST_INVALID_STATE

The internal status of the driver or hardware interface is invalid.

USBHOST_ERROR

An unspecified error occurred.

USBHOST_CC_ERROR

A USB transaction failed with a completion code. This is a mask containing the Completion Code of the USB transaction in the lower nibble of the status code.

Completion Codes

USBHOST_CC_CRC

Last data packet from endpoint contained a CRC error.

USBHOST_CC_BITSTUFFING

Last data packet from endpoint contained a bit stuffing violation

USBHOST_CC_DATATOGGLEMISMATCH

Last packet from endpoint had data toggle PID that did not match the expected value.

USBHOST_CC_STALL

Transaction failed because the endpoint returned a STALL PID

USBHOST_CC_DEVICENOTRESPONDING

Device did not respond to token (IN) or did not provide a handshake (OUT)

USBHOST_CC_PIDCHECKFAILURE

Check bits on PID from endpoint failed on data PID (IN) or handshake (OUT)

USBHOST_CC_UNEXPECTEDPID

Receive PID was not valid when encountered or PID value is not defined.

USBHOST_CC_DATAOVERRUN

The amount of data returned by the endpoint exceeded either the size of the maximum data packet allowed from the endpoint or the remaining buffer size.

USBHOST_CC_DATAUNDERRUN

The endpoint returned less than the Maximum Packet Size from the endpoint and that amount was not sufficient to fill the specified buffer.

USBHOST_CC_BUFFEROVERRUN

During an IN, the host controller received data from endpoint faster than it could be written to system memory.

USBHOST_CC_BUFFERUNDERRUN

During an OUT, the host controller could not retrieve data from system memory fast enough to keep up with data USB data rate.

USBHOST_CC_BUFFEROVERRUN_ISO

During an IN on an isochronous endpoint, the host controller received data from endpoint faster than it could be written to system memory.

USBHOST_CC_BUFFERUNDERRUN_ISO

During an OUT on an isochronous endpoint, the host controller could not retrieve data from system memory fast enough to keep up with data USB data rate.

USBHOST_CC_NOTACCESSED

This indicates that the transaction was not processed by the host controller.

4.2.1.2.3 USB Host Read and Write Operations

Description

To read and write to the USB Host a transfer block is used. This is a structure that is sent to [vos_dev_read\(\)](#) and [vos_dev_write\(\)](#) to describe to the USB Host driver how to transfer data to an endpoint.

It specifies the endpoint to target, the buffer for actual transaction data, data length, completion code and a semaphore to use for either signalling the calling application or blocking the transaction.

The structure differs slightly for isochronous and general (control, bulk and interrupt) endpoints.

To perform a USB Host transaction, the application must fill in the data in the transfer block and then send a pointer to the transfer block to the driver using [vos_dev_read\(\)](#) or [vos_dev_write\(\)](#). The driver returns the status of the transaction in the same structure.

Return Values

USBHOST_NOT_FOUND

The device has been disconnected or the endpoint was not found.

USBHOST_EP_HALTED

A previous operation has resulted in the USB Host controller halting the endpoint. The halt state must be cleared with a [VOS_IOCTL_USBHOST_DEVICE_CLEAR_ENDPOINT_HALT](#) call to the endpoint

USBHOST_EP_INVALID

An OUT operation - [vos_dev_write\(\)](#) - has been attempted on an IN endpoint or an IN operation - [vos_dev_read\(\)](#) - has been attempted on an OUT endpoint.

USBHOST_PENDING

No semaphore was provided with the transfer block therefore there will be no notification of completion.

USBHOST_TD_FULL

No internal resources were available to the USB Host controller to complete the transfer.

4.2.1.2.3.1 USB Host General Transfer Block

Syntax

```
typedef struct _usbhost_xfer_t {
    usbhost_ep_handle *ep;
    vos_semaphore_t *s;
    unsigned char cond_code;
    unsigned char *buf;
    unsigned short len;
    unsigned char flags;
    // internal driver use only
    unsigned char resvl;
    // MUST be set to zero
    unsigned char zero;
} usbhost_xfer_t;
```

Description

The structure defined below is used for transactions on control, bulk and interrupt endpoints.

Parameters

ep

The endpoint handle must be first obtained using IOCTL calls. It is mandatory to specify an endpoint in this structure.

s

An optional semaphore pointer can be specified which is supplied by the application to allow either the read or write operation to block until completion or to allow the application to receive a notification of the transfer completing.

If the semaphore is NULL then there is no notification of completion of a transaction. The condition code member must be polled until a valid status is detected.

cond_code

The condition code is set by the completion of the transaction. It can optionally be set to USBHOST_CC_NOTACCESSED to facilitate a manual check that the operation is completed successfully.

buf

This is a pointer to a buffer that either receives the data from the transaction (read) or where data for the transaction is taken from (write).

len

The maximum length (in bytes) of the data to transmit or to receive is set here before the call to [vos_dev_read\(\)](#) or [vos_dev_write\(\)](#). This is the maximum length, however, less data than specified may be transferred.

When the operation completes, the value of len is updated to reflect the actual number of bytes transferred. If an error occurred then the value of len will still be updated with the number of bytes remaining to send.

flags

The following flags may be set for each transaction block:

USBHOST_XFER_FLAG_START_CTRL_ENDPOINT_LIST

Once the transaction block has been processed the USB Host is instructed to start processing control transfers.

USBHOST_XFER_FLAG_START_BULK_ENDPOINT_LIST

Bulk and interrupt transfers will be started when the transaction block has been processed.

USBHOST_XFER_FLAG_NONBLOCKING

If this flag is set then the [vos_dev_read\(\)](#) or [vos_dev_write\(\)](#) operation returns without waiting for the USB transaction to complete. If a semaphore is specified then this can be waited on until completion. If a semaphore is not set then this flag has no effect.

USBHOST_XFER_FLAG_ROUNDING

Not all USB transactions will use the maximum number of bytes in the len field. If a device does return or accept the exact number of bytes in the len member then a data underrun condition will be signalled unless this flag is set.

The flags set for a particular transaction block may affect other transactions by starting their lists. They do not affect the blocking or rounding of other transactions.

resv1

Reserved for future use.

zero

For general transaction blocks this MUST be set to zero.

Remarks

The transfer blocks can be used in a flexible manner to send data to any endpoint on a USB host. It provides a choice of a blocking operation or allowing an application to control when it receives notification of completion.

Example

```
unsigned char sendData(usbhost_ep_handle ep, unsigned char *buf,
    unsigned short len)
{
    usbhost_xfer_t xfer;
    unsigned char status;
    vos_semaphore_t s;

    memset(&xfer, 0, sizeof(usbhost_xfer_t));
    status = MY_OK;
    vos_init_semaphore(&s, 0);

    xfer.buf = buf;
    xfer.len = len;
    xfer.ep = ep;
    xfer.s = s;
    xfer.cond_code = USBHOST_CC_NOTACCESSED;
    xfer.flags = USBHOST_XFER_FLAG_START_BULK_ENDPOINT_LIST;

    status = vos_dev_write(hUSB1, (unsigned char *)&xfer, sizeof(usbhost_xfer_t), NULL);
    if (status != USBHOST_OK) {
        status |= MY_TRANSPORT_ERROR;
        return status;
    }

    status = xfer.cond_code;
    if (status != USBHOST_CC_NOERROR)
    {
        if (status == USBHOST_CC_STALL)
        {

```

```
        // recover endpoint
    }
}
return status;
}
```

4.2.1.2.3.2 USB Host Isochronous Transfer Block

Syntax

```
typedef struct _usbhost_xfer_t {
    usbhost_ep_handle *ep;
    vos_semaphore_t *s;
    unsigned char cond_code;
    unsigned char *buf;
    unsigned short len;
    unsigned char flags;
    // internal driver use only
    unsigned char resvl;
    unsigned char count;
    struct {
        unsigned short size:11;
        unsigned short pad:1;
        unsigned short cond_code:4;
    } len_psw[8];
    unsigned short frame;
} usbhost_xfer_t;
```

Description

The structure defined below is used for transactions on isochronous endpoints.

Parameters

ep

The endpoint handle must be first obtained using IOCTL calls. It is mandatory to specify an endpoint in this structure.

s

An optional semaphore pointer can be specified which is supplied by the application to allow either the read or write operation to block until completion or to allow the application to receive a notification of the transfer completing.

If the semaphore is NULL then there is no notification of completion of a transaction. The condition code member must be polled until a valid status is detected.

cond_code

The condition code is set by the completion of the transaction. It can optionally be set to USBHOST_CC_NOTACCESSED to facilitate a manual check that the operation is completed successfully.

buf

This is a pointer to a buffer that either receives the data from the transaction (read) or where data for the transaction is taken from (write).

len

The maximum length (in bytes) of the data to transmit or to receive is set here before the call to [vos_dev_read\(\)](#) or [vos_dev_write\(\)](#). This is the maximum length, however, less data than specified may be transferred.

When the operation completes, the value of len is updated to reflect the actual number of bytes transferred. If an error occurred then the value of len will still be updated with the number of bytes remaining to send.

flags

The following flags may be set for each transaction block:

USBHOST_XFER_FLAG_NONBLOCKING

If this flag is set then the [vos_dev_read\(\)](#) or [vos_dev_write\(\)](#) operation returns without waiting for the USB transaction to complete. If a semaphore is specified then this can be waited on until completion. If a semaphore is not set then this flag has no effect.

USBHOST_XFER_FLAG_ROUNDING

Not all USB transactions will use the maximum number of bytes in the len field. If a device does return or accept the exact number of bytes in the len member then a data underrun condition will be signalled unless this flag is set.

resv1

Reserved for future use.

count

This specifies the number of consecutive frames to which the isochronous transaction will appear over. This value tells the driver how many len_psw entries are valid.

len_psw

An array containing 8 instances of isochronous frame instructions.

size

For each frame, this is the size of data to be transferred in that frame. In total, all sizes specified in the each len_psw must add up to the len member.

cond_code

This is the condition code for each frame. Due to the nature of isochronous, not all errors in a frame will result in an error in the main cond_code member, but each frame may be checked separately.

Remarks

The configuration descriptor of an isochronous device interface is used to configure the period which the endpoint is interrogated for data when an isochronous transfer is taking place. It is not necessary to configure this in the transfer block.

Example

4.2.1.2.4 USB Host IOCTL Calls

Calls to the IOCTL functions for the USB Host driver take the form:

```
typedef struct _usbhost_ioctl_cb_t {
    unsigned char ioctl_code;
    // hub port number (ignored on root hub)
    unsigned char hub_port;
    union
    {
        // handle of endpoint to use
        usbhost_ep_handle *ep;
        // handle of interface to use
        usbhost_device_handle *dif;
    } handle;
    // read buffer
    void *get;
    // write buffer
    void *set;
} usbhost_ioctl_cb_t;
```

The following IOCTL request codes are supported by the USB Host driver.

[VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#)

Get host controller connect state

[VOS_IOCTL_USBHOST_ENUMERATE](#)

Force host controller enumeration

[VOS_IOCTL_USBHOST_GET_USB_STATE](#)

Get host controller USB state

[VOS_IOCTL_USBHOST_DEVICE_GET_COUNT](#)

Get host controller device count

[VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE](#)

Get handle to device interface on USB host

[VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_VID_PID](#)

Find a device interface by VID and PID

[VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS](#)

Find a device interface by USB class

<u>VOS_IOCTL_USBHOST_DEVICE_GET_VID_PID</u>	Get VID and PID of device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_CLASS_INFO</u>	Get USB class information for device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_DEV_INFO</u>	Get information on device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE</u>	Get first control endpoint for device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE</u>	Get first bulk in endpoint for device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_BULK_OUT_ENDPOINT_HANDLE</u>	Get first bulk out endpoint for device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_INT_IN_ENDPOINT_HANDLE</u>	Get first interrupt in endpoint for device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_INT_OUT_ENDPOINT_HANDLE</u>	Get first interrupt out endpoint for device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_ISO_IN_ENDPOINT_HANDLE</u>	Get first isochronous in endpoint for device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_ISO_OUT_ENDPOINT_HANDLE</u>	Get first isochronous out endpoint for device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_ENDPOINT_HANDLE</u>	Get next endpoint for device interface
<u>VOS_IOCTL_USBHOST_DEVICE_GET_ENDPOINT_INFO</u>	Get endpoint information
<u>VOS_IOCTL_USBHOST_SET_INTERFACE</u>	Set interface for device interface
<u>VOS_IOCTL_USBHOST_DEVICE_CLEAR_ENDPOINT_HALT</u>	Clear endpoint halt on device and USB Host controller
<u>VOS_IOCTL_USBHOST_DEVICE_CLEAR_HOST_HALT</u>	Clear USB Host controller endpoint halt
<u>VOS_IOCTL_USBHOST_DEVICE_SET_HOST_HALT</u>	Set USB Host controller endpoint halt
<u>VOS_IOCTL_USBHOST_DEVICE_SETUP_TRANSFER</u>	Perform a setup transfer to an endpoint
<u>VOS_IOCTL_USBHOST_HW_GET_FRAME_NUMBER</u>	Remove a transaction from a Get host controller current frame number
Hub IOCTL requests are not commonly used but are available in the USB Host driver.	
<u>VOS_IOCTL_USBHUB_HUB_PORT_COUNT</u>	Count of ports on the selected hub
<u>VOS_IOCTL_USBHUB_HUB_STATUS</u>	Status of selected hub
<u>VOS_IOCTL_USBHUB_PORT_STATUS</u>	Status of port on selected hub
<u>VOS_IOCTL_USBHUB_CLEAR_C_HUB_LOCAL_POWER</u>	Clear hub local power state change
<u>VOS_IOCTL_USBHUB_CLEAR_C_HUB_OVERCURRENT</u>	Clear hub overcurrent state change
<u>VOS_IOCTL_USBHUB_CLEAR_PORT_ENABLE</u>	Clear port enable for selected port on hub
<u>VOS_IOCTL_USBHUB_SET_PORT_SUSPEND</u>	Set port suspend for selected port on hub
<u>VOS_IOCTL_USBHUB_CLEAR_PORT_SUSPEND</u>	Clear port suspend for selected port on hub
<u>VOS_IOCTL_USBHUB_SET_PORT_RESET</u>	Set port reset for selected port

[VOS_IOCTL_USBHUB_SET_PORT_POWER](#)

on hub

Set port power for selected port on hub

[VOS_IOCTL_USBHUB_CLEAR_PORT_POWER](#)

Clear port power for selected port on hub

[VOS_IOCTL_USBHUB_CLEAR_C_PORT_CONNECTION](#)

Clear port connection state change for selected port on hub

[VOS_IOCTL_USBHUB_CLEAR_C_PORT_ENABLE](#)

Clear port enable state change for selected port on hub

[VOS_IOCTL_USBHUB_CLEAR_C_PORT_SUSPEND](#)

Clear port suspend state change for selected port on hub

[VOS_IOCTL_USBHUB_CLEAR_C_PORT_OVERCURRENT](#)

Clear port overcurrent state change for selected port on hub

[VOS_IOCTL_USBHUB_CLEAR_C_PORT_RESET](#)

Clear port reset state change for selected port on hub

4.2.1.2.4.1 VOS_IOCTL_USBHOST_GET_CONNECT_STATE

Description

Determines the state of the USB Host controller. This can be unconnected, connected or enumerated.

This feature is used to detect device connection and wait until enumeration of device is completed.

Parameters

There are no parameters to pass to this function.

Returns

The status of the USB Host controller is returned to an unsigned char which is pointed to by the get member of the IOCTL structure.

The return value is one of the following values:

```
PORT_STATE_DISCONNECTED
PORT_STATE_CONNECTED
PORT_STATE_ENUMERATED
```

Example

```
unsigned char waitForEnumeration(VOS_HANDLE hUsbHost)
{
    usbhost_ioctl_cb_t usbhost_iocb;
    unsigned char i;

    // wait until enumeration is complete
    do
    {
        vos_delay_msecs(1);
        usbhost_iocb.ioctl_code = VOS_IOCTL_USBHOST_GET_CONNECT_STATE;
        usbhost_iocb.get = &i;
        vos_dev_ioctl(hUsbHost, &usbhost_iocb);
    } while (i != PORT_STATE_ENUMERATED);

    return i;
}
```

4.2.1.2.4.2 VOS_IOCTL_USBHOST_ENUMERATE

Description

Forces the USB Host controller to re-enumerate from the root hub.

Parameters

There are no parameters to pass to this function.

Returns

There is no return value.

Example

```
void reenumerate(VOS_HANDLE hUsbHost)
{
    usbhost_ioctl_cb_t usbhost_iocb;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHOST_ENUMERATE;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);
}
```

4.2.1.2.4.3 VOS_IOCTL_USBHOST_GET_USB_STATE

Description

Returns the state of the USB Bus. This could be operational, suspended, reset or resuming. A further bit is used to indicate that a change is pending to the state.

Parameters

There are no parameters to pass to this function.

Returns

The USB Bus state can be one of the following values:

```
USB_STATE_RESET
USB_STATE_OPERATIONAL
USB_STATE_RESUME
USB_STATE_SUSPEND
```

An additional bit is set if there is a change in progress.

```
USB_STATE_CHANGE_PENDING
```

Example

```
unsigned char getBusState(VOS_HANDLE hUsbHost)
{
    usbhost_ioctl_cb_t usbhost_iocb;
    unsigned char state;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHOST_GET_USB_STATE;
    usbhost_iocb.get = &state;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);

    return state;
}
```

4.2.1.2.4.4 VOS_IOCTL_USBHOST_DEVICE_GET_COUNT

Description

Returns the number of device interfaces enumerated on the USB bus.

Parameters

There are no parameters to pass to this function.

Returns

The number of interfaces is passed into the variable pointed to by the `get` member of the `IOCTL` function.

Example

```
unsigned char getNumberDevices(VOS_HANDLE hUsbHost)
{
    usbhost_ioctl_cb_t usbhost_iocb;
    unsigned char num_dev;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_COUNT;
    usbhost_iocb.get = &num_dev;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);

    return num_dev;
}
```

4.2.1.2.4.5 VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE

Description

Returns a handle to the first or subsequent device interface on a USB bus.

Parameters

To find the first device interface on a bus, pass `NULL` to the `IOCTL` operation in the `handle.dif` member.

Subsequent device interfaces can be found by passing a handle found with a previous call in the `handle.dif` member.

Returns

A handle to a device interface is returned into the variable pointed to by the `get` member of the `IOCTL` function. If the end of the device interface list is found then `NULL` is returned.

Example

```
usbhost_ioctl_cb_t hc_iocb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
hc_iocb.handle.dif = NULL;
hc_iocb.get = &ifDev;
vos_dev_ioctl(hUsbHost, &hc_iocb);

// find second device interface
hc_iocb.handle.dif = ifDev;
hc_iocb.get = &ifDev;
vos_dev_ioctl(hUsbHost, &hc_iocb);
```

4.2.1.2.4.6 VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_VID_PID

Description

Returns a handle to a device interface which matches a supplied VID and PID.

Parameters

A VID and PID specification is passed to the `IOCTL` operation in the `set` member. This can either have exact VID and PID values or these can match any value using `USB_VID_ANY` or `USB_PID_ANY`.

```
typedef struct _usbhost_ioctl_cb_vid_pid_t {
    unsigned short vid;
    unsigned short pid;
}
```

```
} usbhost_ioctl_cb_vid_pid_t;
```

Common values of VID and PID are in the driver header file "usb.h".

Returns

A handle to a device interface is returned into the variable pointed to by the `get` member of the IOCTL function. If no matching device interface list is found then NULL is returned.

Example

```
usbhost_ioctl_cb_t hc_iocb; // ioctl block
usbhost_ioctl_cb_vid_pid_t usbhost_ioctlVidPid;
usbhost_device_handle *ifDev; // handle to the next device interface

// find VID/PID FT232 (or similar)
usbhost_ioctlVidPid.vid = USB_VID_FTDI;
usbhost_ioctlVidPid.pid = USB_PID_ANY;

usbhost_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_VID_PID;
usbhost_iocb.handle.dif = NULL;
usbhost_iocb.set = &usbhost_ioctlVidPid;
usbhost_iocb.get = &ifDev;

vos_dev_ioctl(hUsbHost1, &usbhost_iocb);
```

4.2.1.2.4.7 VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS

Description

Returns a handle to a device interface which matches a supplied USB class, Subclass and protocol.

Parameters

A class, subclass and protocol specification is passed to the IOCTL operation in the `set` member. This can either have exact class, subclass and protocols values or these can match any subclass and protocol using `USB_SUBCLASS_ANY` or `USB_PROTOCOL_ANY`.

```
typedef struct _usbhost_ioctl_cb_class_t {
    unsigned char dev_class;
    unsigned char dev_subclass;
    unsigned char dev_protocol;
} usbhost_ioctl_cb_class_t;
```

Common values of class, subclass and protocol are in the driver header file "usb.h".

Returns

A handle to a device interface is returned into the variable pointed to by the `get` member of the IOCTL function. If no matching device interface list is found then NULL is returned.

Example

```
usbhost_ioctl_cb_t hc_iocb; // ioctl block
usbhost_ioctl_cb_class_t hc_iocb_class;
usbhost_device_handle *ifDev; // handle to the next device interface

hc_iocb_class.dev_class = USB_CLASS_IMAGE;
hc_iocb_class.dev_subclass = USB_SUBCLASS_IMAGE_STILLIMAGE;
hc_iocb_class.dev_protocol = USB_PROTOCOL_IMAGE_PIMA;

usbhost_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
usbhost_iocb.handle.dif = NULL;
usbhost_iocb.set = &hc_iocb_class;
usbhost_iocb.get = &ifDev;

vos_dev_ioctl(hUsbHost2, &usbhost_iocb);
```

4.2.1.2.4.8 VOS_IOCTL_USBHOST_DEVICE_GET_VID_PID

Description

Returns the VID and PID of a device interface from a device interface handle.

Parameters

The device to query is passed in the `handle.dif` member.

Returns

A structure to receive the VID and PID information for a device is passed in the `get` member of the IOCTL structure.

```
typedef struct _usbhost_ioctl_cb_vid_pid_t {
    unsigned short vid;
    unsigned short pid;
} usbhost_ioctl_cb_vid_pid_t;
```

Common values of VID and PID are in the driver header file "usb.h".

Example

```
usbhost_ioctl_cb_t hc_iocb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface
usbhost_ioctl_cb_vid_pid_t hc_iocb_vid_pid;

hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
hc_iocb.handle.dif = NULL;
hc_iocb.get = &ifDev;
vos_dev_ioctl(hUsbHost, &hc_iocb);

hc_iocb.handle.dif = ifDev;

if (ifDev)
{
    hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_VID_PID;
    hc_iocb.get = &hc_iocb_vid_pid;

    vos_dev_ioctl(hUsbHost, &hc_iocb);

    myVid = hc_iocb_vid_pid.vid;
    myPid = hc_iocb_vid_pid.pid;
}
```

4.2.1.2.4.9 VOS_IOCTL_USBHOST_DEVICE_GET_CLASS_INFO

Description

Returns the class, subclass and protocol of a device interface from a device interface handle.

Parameters

The device to query is passed in the `handle.dif` member.

Returns

A structure to receive the class, subclass and protocol information for a device is passed in the `get` member of the IOCTL structure.

```
typedef struct _usbhost_ioctl_cb_class_t {
    unsigned char dev_class;
    unsigned char dev_subclass;
    unsigned char dev_protocol;
} usbhost_ioctl_cb_class_t;
```

Common values of class, subclass and protocol are in the driver header file "usb.h".

Example

```
usbhost_ioctl_cb_t hc_iocb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface
usbhost_ioctl_cb_class_t hc_iocb_class;

hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
hc_iocb.handle.dif = NULL;
hc_iocb.get = &ifDev;
vos_dev_ioctl(hUsbHost, &hc_iocb);

hc_iocb.handle.dif = ifDev;

if (ifDev)
{
    hc_ioctl.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_CLASS_INFO;
    hc_ioctl.get = &hc_iocb_class;

    vos_dev_ioctl(hUsbHost, &hc_ioctl);

    if ((hc_iocb_class.dev_class != USB_CLASS_IMAGE) ||
        (hc_iocb_class.dev_protocol != USB_PROTOCOL_IMAGE_PIMA))
    {
        return STILLIMAGE_ERROR;
    }
}
```

4.2.1.2.4.10 VOS_IOCTL_USBHOST_DEVICE_GET_DEV_INFO

Description

Returns information about a device interface from a device interface handle.

Parameters

The device to query is passed in the handle.dif member.

Returns

A structure to receive various information about the interface, including device USB device address, speed and the USB port it is connected to.

```
typedef struct _usbhost_ioctl_cb_dev_info_t {
    unsigned char port_number; // USB Host port number
    unsigned char addr; // USB Bus address
    unsigned char interface_number; // interface offset in device
    unsigned char speed; // zero for full speed, 1 for low speed
    unsigned char alt; // alternate setting
} usbhost_ioctl_cb_dev_info_t;
```

Example

```
usbhost_ioctl_cb_t hc_iocb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface
usbhost_ioctl_cb_dev_info_t ifInfo;

hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
hc_iocb.handle.dif = NULL;
hc_iocb.get = &ifDev;
vos_dev_ioctl(hUsbHost, &hc_iocb);

hc_iocb.handle.dif = ifDev;
```

```
if (ifDev)
{
    host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_DEV_INFO;
    host_ioctl_cb.get = &ifInfo;

    vos_dev_ioctl(hUsbHost, &host_ioctl_cb); // send the ioctl to the device manager

    USBAddress = ifInfo.interface_number;
    DeviceSpeed = ifInfo.speed;
    Location = ifInfo.port_number;
}
```

4.2.1.2.4.11

VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE

Description

Finds the first control endpoint for a device interface.

Parameters

The device interface to search is passed in the `handle.dif` member.

Returns

A handle to the first control endpoint is returned to the endpoint handle pointed to by the `get` member of the IOCTL structure.

Example

```
usbhost_ep_handle epHandle; // Handle to our endpoint.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

if (epHandle)
{
    // found control endpoint
}
```

4.2.1.2.4.12

VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE

Description

Finds the first bulk IN endpoint for a device interface.

Parameters

The device interface to search is passed in the `handle.dif` member.

Returns

A handle to the first bulk IN endpoint is returned to the endpoint handle pointed to by the `get` member of the IOCTL structure.

Example

```
usbhost_ep_handle epHandle; // Handle to our endpoint.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

if (epHandle)
{
    // found bulk IN endpoint
}
```

4.2.1.2.4.13

VOS_IOCTL_USBHOST_DEVICE_GET_BULK_OUT_ENDPOINT_HANDLE

Description

Finds the first bulk OUT endpoint for a device interface.

Parameters

The device interface to search is passed in the `handle.dif` member.

Returns

A handle to the first bulk OUT endpoint is returned to the endpoint handle pointed to by the `get` member of the IOCTL structure.

Example

```
usbhost_ep_handle epHandle; // Handle to our endpoint.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_BULK_OUT_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
```

```
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);
```

```
if (epHandle)
{
    // found bulk OUT endpoint
}
```

4.2.1.2.4.14

VOS_IOCTL_USBHOST_DEVICE_GET_INT_IN_ENDPOINT_HANDLE

Description

Finds the first interrupt IN endpoint for a device interface.

Parameters

The device interface to search is passed in the `handle.dif` member.

Returns

A handle to the first interrupt IN endpoint is returned to the endpoint handle pointed to by the `get` member of the IOCTL structure.

Example

```
usbhost_ep_handle epHandle; // Handle to our endpoint.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_INT_IN_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

if (epHandle)
{
    // found interrupt IN endpoint
}
```

4.2.1.2.4.15

VOS_IOCTL_USBHOST_DEVICE_GET_INT_OUT_ENDPOINT_HANDLE

Description

Finds the first interrupt OUT endpoint for a device interface.

Parameters

The device interface to search is passed in the `handle.dif` member.

Returns

A handle to the first interrupt OUT endpoint is returned to the endpoint handle pointed to by the `get` member of the IOCTL structure.

Example

```
usbhost_ep_handle epHandle; // Handle to our endpoint.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_INT_OUT_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

if (epHandle)
{
    // found interrupt OUT endpoint
}
```

4.2.1.2.4.16

VOS_IOCTL_USBHOST_DEVICE_GET_ISO_IN_ENDPOINT_HANDLE

Description

Finds the first isochronous IN endpoint for a device interface.

Parameters

The device interface to search is passed in the `handle.dif` member.

Returns

A handle to the first isochronous IN endpoint is returned to the endpoint handle pointed to by the `get` member of the IOCTL structure.

Example

```
usbhost_ep_handle epHandle; // Handle to our endpoint.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_ISO_IN_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

if (epHandle)
{
    // found isochronous IN endpoint
}
```

4.2.1.2.4.17

VOS_IOCTL_USBHOST_DEVICE_GET_ISO_OUT_ENDPOINT_HANDLE

Description

Finds the first isochronous OUT endpoint for a device interface.

Parameters

The device interface to search is passed in the `handle.dif` member.

Returns

A handle to the first isochronous OUT endpoint is returned to the endpoint handle pointed to by the `get` member of the IOCTL structure.

Example

```
usbhost_ep_handle epHandle; // Handle to our endpoint.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_ISO_OUT_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

if (epHandle)
{
    // found isochronous OUT endpoint
}
```

4.2.1.2.4.18

VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_ENDPOINT_HANDLE

Description

Finds a subsequent endpoint of the same type for a device interface. A starting endpoint handle must be found first.

Parameters

A valid endpoint handle is passed in the `handle.ep` member.

Returns

A handle to the next endpoint of the same type as the starting endpoint is returned to the endpoint handle pointed to by the `get` member of the IOCTL structure.

Example

```
usbhost_ep_handle epHandle; // Handle to our endpoint.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;
```

```
// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_ISO_IN_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

if (epHandle)
{
    // found first isochronous IN endpoint
    host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_ENDPOINT_HANDLE;

    host_ioctl_cb.handle.ep = epHandle;
    host_ioctl_cb.get = &epHandle;
    vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

    if (epHandle)
    {
        // second isochronous IN endpoint found
    }
}
```

4.2.1.2.4.19 VOS_IOCTL_USBHOST_DEVICE_GET_ENDPOINT_INFO

Description

Returns information about an endpoint from an endpoint handle.

Parameters

The endpoint to query is passed in the `handle.ep` member.

Returns

A structure to receive various information about the interface, including device USB endpoint number, speed and the USB port it is connected to.

```
typedef struct _usbhost_ioctl_cb_ep_info_t {
    unsigned char number;
    unsigned short max_size;
    unsigned char speed;
} usbhost_ioctl_cb_ep_info_t;
```

The endpoint number (`number` member) returns the endpoint number and direction bit set. For example, Endpoint 2 IN will return 0x82, endpoint 1 OUT will return 0x01.

The `speed` member is set to zero for Full Speed and one for Low Speed.

Example

```
usbhost_ep_handle epHandle; // Handle to our endpoint.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_ioctl_cb_ep_info_t epInfo; // Structure to store our endpoint data.usbhost_device_handle

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);
```

```
epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

if (epHandle)
{
    host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_ENDPOINT_INFO;
    host_ioctl_cb.handle.ep = epHandle;
    host_ioctl_cb.get = &epInfo;

    vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

    CtrlEndPoint = epInfo.max_size;
}
```

4.2.1.2.4.20 VOS_IOCTL_USBHOST_SET_INTERFACE

Description

Enables a device interface using a USB Set Interface method.

Parameters

A valid device interface handle is passed in the `handle.dif` member.

Returns

There is no data returned from the IOCTL operation.

Example

```
usbhost_ep_handle epHandle; // Handle to our endpoint.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_ISO_IN_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

if (epHandle)
{
    // set this interface to be enabled
    host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_SET_INTERFACE;
    host_ioctl_cb.handle.dif = ifDev;

    vos_dev_ioctl(hUsbHost, host_ioctl_cb);
}
```

4.2.1.2.4.21 VOS_IOCTL_USBHOST_DEVICE_CLEAR_ENDPOINT_HALT

Description

Clears a halt state from an endpoint on a device. This will also clear the halt state from the endpoint on USB Host controller. The endpoint record on the USB Host controller is reset to match the state of the device endpoint.

The control endpoint for a device interface and the endpoint which is halted must be specified in the call to IOCTL operation.

Parameters

A valid control endpoint handle is passed in the `handle.ep` member. A handle to a halted endpoint is passed in the `set` member of the IOCTL structure.

Returns

There is no data returned by this IOCTL operation.

Example

```
usbhost_ep_handle epHandle, epHandleBulkIn; // Handle to our endpoints.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandle;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandleBulkIn;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_CLEAR_ENDPOINT_HALT;
host_ioctl_cb.handle.ep = epHandle;
host_ioctl_cb.set = epHandleBulkIn;

// clear halt state on endpoint
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);
```

4.2.1.2.4.22 VOS_IOCTL_USBHOST_DEVICE_CLEAR_HOST_HALT

Description

Clears a halt state from an endpoint on the USB Host controller. This does not clear the halt state for the endpoint on a device.

The USB Host controller will halt an endpoint in it's own bus status records if a transaction to that endpoint results in an error such as a stall, buffer overrun, buffer underrun; or a pid, data toggle or bit stuffing error.

This error may not actually affect the endpoint on the device so the device itself may not be in a halted state. Therefore it may be possible to clear the halt state from the USB Host controller to continue using the endpoint.

The endpoint to be cleared must be specified in the call to IOCTL operation.

Parameters

A valid endpoint handle to the halted endpoint is passed in the `handle.ep` member.

Returns

There is no data returned by this IOCTL operation.

Example

```
usbhost_ep_handle epHandleBulkIn; // Handle to our endpoints.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandleBulkIn;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_SET_HOST_HALT;
host_ioctl_cb.handle.ep = epHandleBulkIn;

// set halt state on endpoint
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

vos_delay_msecs(10);

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_CLEAR_HOST_HALT;
host_ioctl_cb.handle.ep = epHandleBulkIn;

// clear halt state on endpoint
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);
```

4.2.1.2.4.23 VOS_IOCTL_USBHOST_DEVICE_SET_HOST_HALT

Description

Sets a halt state on an endpoint on the USB Host controller. This does not set the halt state for the endpoint on a device.

The USB Host controller will halt an endpoint in it's own bus status records if a transaction to that endpoint results in an error such as a stall, buffer overrun, buffer underrun; or a pid, data toggle or bit stuffing error. This call will override this and set the halt state.

The endpoint to be set must be specified in the call to IOCTL operation.

Parameters

A valid endpoint handle to the endpoint is passed in the `handle.ep` member.

Returns

There is no data returned by this IOCTL operation.

Example

See example in [VOS_IOCTL_USBHOST_DEVICE_CLEAR_HOST_HALT](#).

4.2.1.2.4.24

VOS_IOCTL_USBHOST_DEVICE_CLEAR_ENDPOINT_TRANSFER

Description

Removes all current transfers scheduled for an endpoint on the USB Host controller.

The USB Host controller will halt an endpoint then clear all the transfers scheduled on the endpoint. Semaphores which trigger when transfers are completed will be signalled.

The endpoint to be set must be specified in the call to IOCTL operation.

Parameters

A valid endpoint handle to the endpoint is passed in the `handle.ep` member.

Returns

There is no data returned by this IOCTL operation.

Example

```
usbhost_ep_handle epHandleBulkIn; // Handle to our endpoints.
usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
usbhost_device_handle *ifDev; // handle to the next device interface

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;

// find first device interface
host_ioctl_cb.handle.dif = NULL;
host_ioctl_cb.get = &ifDev;
vos_dev_ioctl(hUsbHost, host_ioctl_cb);

epHandle = NULL;

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE;

host_ioctl_cb.handle.dif = ifDev;
host_ioctl_cb.get = &epHandleBulkIn;
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);

host_ioctl_cb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_CLEAR_ENDPOINT_TRANSFER;
host_ioctl_cb.handle.ep = epHandleBulkIn;

// clear any transfers active on the endpoint
vos_dev_ioctl(hUsbHost, &host_ioctl_cb);
```

4.2.1.2.4.25 VOS_IOCTL_USBHOST_DEVICE_SETUP_TRANSFER

Description

Performs a setup transaction to a control endpoint on a device interface. A device request descriptor is passed as the SETUP phase of the transaction. This can be followed by a data phase if required. Finally an ACK phase will be performed.

Parameters

A USB device request descriptor is passed to the IOCTL operation in the `set` member of the IOCTL structure. The structure and definitions for the members of the structure can be found in the 'usb.h' driver header file.

```
typedef struct _usb_deviceRequest_t
{
    // D7: Data transfer direction
    //      0 = Host-to-device
    //      1 = Device-to-host
    // D6...5: Type
```

```
//          0 = Standard
//          1 = Class
//          2 = Vendor
//          3 = Reserved
// D4...0: Recipient
//          0 = Device
//          1 = Interface
//          2 = Endpoint
//          3 = Other
unsigned char bmRequestType;
// Table 9-4.
unsigned char bRequest;
unsigned short wValue;
unsigned short wIndex;
unsigned short wLength;

} usb_deviceRequest_t;
```

The control endpoint handle which will receive the SETUP request is passed in the `handle.ep` member.

Returns

If a 'device to host' request is performed then the returned data is placed in the buffer pointed to by the `get` member of the IOCTL structure.

Example

```
// perform a getPortStatus standard request on a printer
desc_dev.bmRequestType = USB_BMREQUESTTYPE_DEV_TO_HOST
                        | USB_BMREQUESTTYPE_CLASS
                        | USB_BMREQUESTTYPE_INTERFACE;
desc_dev.bRequest = GET_PORT_STATUS;
desc_dev.wValue = 0;
desc_dev.wIndex = 0;
desc_dev.wLength = 1; // Sends back 1 byte in the data phase.

hc_ioctl.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_SETUP_TRANSFER;
hc_ioctl.handle.ep = ep;
hc_ioctl.set = &desc_dev;
hc_ioctl.get = printerStatus; // Status returned from the printer

result = vos_dev_ioctl(hUsb, &hc_ioctl);
```

4.2.1.2.4.26 VOS_IOCTL_USBHOST_HW_GET_FRAME_NUMBER

Description

Returns the current USB frame number for a USB Host controller.

Parameters

There are no parameters to pass to this function.

Returns

The frame number is passed into the variable pointed to by the `set` member of the IOCTL function.

Example

```
unsigned short getFrameNumber(VOS_HANDLE hUsbHost)
{
    usbhost_ioctl_cb_t usbhost_iocb;
    unsigned short frame;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHOST_HW_GET_FRAME_NUMBER;
    usbhost_iocb.set = &frame;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);
}
```

```
    return num_dev;  
}
```

4.2.1.2.4.27 VOS_IOCTL_USBHUB_HUB_PORT_COUNT

Description

Returns the number of ports connected to a hub device. This can be the root hub on the VNC2 or a downstream hub.

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

Returns

The port count is returned to an unsigned char which is pointed to by the `get` member of the IOCTL structure.

Example

```
unsigned char getHubPortCount(VOS_HANDLE hUsbHost)  
{  
    usbhost_ioctl_cb_t usbhost_iocb;  
    unsigned char i;  
  
    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_PORT_COUNT;  
    usbhost_iocb.get = &i;  
    // query the root hub  
    usbhost_iocb.handle.dif = NULL;  
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);  
  
    return i;  
}
```

4.2.1.2.4.28 VOS_IOCTL_USBHUB_HUB_STATUS

Description

Returns the status of a hub device. This can be the root hub on the VNC2 or a downstream hub. The status bits returned are described in section 11.24.2.6 of the USB Specification Revision 2.0 "Get Hub Status".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

Returns

The hub status is returned to an unsigned short which is pointed to by the `get` member of the IOCTL structure.

Example

```
unsigned short getHubStatus(VOS_HANDLE hUsbHost,  
                           usbhost_device_handle ifHub)  
{  
    usbhost_ioctl_cb_t usbhost_iocb;  
    unsigned short i;  
  
    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_HUB_STATUS;  
    usbhost_iocb.get = &i;  
    // query a downstream hub  
    usbhost_iocb.handle.dif = ifHub;  
}
```

```
vos_dev_ioctl(hUsbHost, &usbhost_iocb);
```

```
    return i;
}
```

4.2.1.2.4.29 VOS_IOCTL_USBHUB_PORT_STATUS

Description

Returns the status of a port on a hub. This can be the root hub on the VNC2 or a downstream hub. The status bits returned are described in section 11.24.2.7 of the USB Specification Revision 2.0 "Get Port Status".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) is passed in the `hub_port` member of the IOCTL structure.

Returns

The port status is returned to an unsigned char which is pointed to by the `get` member of the IOCTL structure.

Example

```
unsigned short getPortStatus(VOS_HANDLE hUsbHost,
                             usbhost_device_handle ifHub,
                             unsigned char index)
{
    usbhost_ioctl_cb_t usbhost_iocb;
    unsigned char i;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_PORT_STATUS;
    usbhost_iocb.get = &i;
    // query a downstream hub
    usbhost_iocb.handle.dif = ifHub;
    // query port passed as parameter
    usbhost_iocb.hub_port = index;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);

    return i;
}
```

4.2.1.2.4.30 VOS_IOCTL_USBHUB_CLEAR_C_HUB_LOCAL_POWER

Description

Clears the change bit for the hub local power indicator. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.1 of the USB Specification Revision 2.0 "Clear Hub Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure must be zero.

Returns

There is no return value from the hub.

Example

```
void clearHubLocalPowerFeature(VOS_HANDLE hUsbHost,
```

```
        usbhost_device_handle ifHub)
{
    usbhost_ioctl_cb_t usbhost_iocb;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_CLEAR_C_HUB_LOCAL_POWER;
    // clear a downstream hub
    usbhost_iocb.handle.dif = ifHub;
    // clear port must be zero
    usbhost_iocb.hub_port = 0;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);
}
```

4.2.1.2.4.31 VOS_IOCTL_USBHUB_CLEAR_C_HUB_OVERCURRENT

Description

Clears the change bit for the hub overcurrent indicator. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.1 of the USB Specification Revision 2.0 "Clear Hub Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure must be zero.

Returns

There is no return value from the hub.

Example

```
void clearHubOvercurrentFeature(VOS_HANDLE hUsbHost,
                               usbhost_device_handle ifHub)
{
    usbhost_ioctl_cb_t usbhost_iocb;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_CLEAR_C_HUB_OVERCURRENT;
    // clear a downstream hub
    usbhost_iocb.handle.dif = ifHub;
    // clear port must be zero
    usbhost_iocb.hub_port = 0;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);
}
```

4.2.1.2.4.32 VOS_IOCTL_USBHUB_CLEAR_PORT_ENABLE

Description

Clears the port enable feature causing the port to be placed in the disabled state. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.2 of the USB Specification Revision 2.0 "Clear Port Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure.

Returns

There is no return value from the hub.

Example

```
void clearPortEnableFeature(VOS_HANDLE hUsbHost,
                           usbhost_device_handle ifHub)
{
    usbhost_ioctl_cb_t usbhost_iocb;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_CLEAR_PORT_ENABLE;
    // clear a downstream hub
    usbhost_iocb.handle.dif = ifHub;
    // clear port number 1
    usbhost_iocb.hub_port = 1;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);
}
```

4.2.1.2.4.33 VOS_IOCTL_USBHUB_SET_PORT_SUSPEND

Description

Sets the port suspend feature causing the port to be placed in the suspend state. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.13 of the USB Specification Revision 2.0 "Set Port Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure.

Returns

There is no return value from the hub.

Example

```
void setPortSuspendFeature(VOS_HANDLE hUsbHost,
                           usbhost_device_handle ifHub,
                           unsigned char index)
{
    usbhost_ioctl_cb_t usbhost_iocb;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_SET_PORT_SUSPEND;
    // clear a downstream hub
    usbhost_iocb.handle.dif = ifHub;
    // clear port number passed
    usbhost_iocb.hub_port = index;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);
}
```

4.2.1.2.4.34 VOS_IOCTL_USBHUB_CLEAR_PORT_SUSPEND

Description

Clears the port suspend feature causing the port to resume if in the suspend state. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.2 of the USB Specification Revision 2.0 "Clear Port Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure.

Returns

There is no return value from the hub.

Example

```
void clearPortSuspendFeature(VOS_HANDLE hUsbHost,
                             usbhost_device_handle ifHub)
{
    usbhost_ioctl_cb_t usbhost_iocb;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_CLEAR_PORT_SUSPEND;
    // clear a downstream hub
    usbhost_iocb.handle.dif = ifHub;
    // clear port number 2
    usbhost_iocb.hub_port = 2;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);
}
```

4.2.1.2.4.35 VOS_IOCTL_USBHUB_SET_PORT_RESET

Description

Sets the port reset feature causing the port to be placed in the reset state. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.13 of the USB Specification Revision 2.0 "Set Port Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure.

Returns

There is no return value from the hub.

Example

```
void setPortResetFeature(VOS_HANDLE hUsbHost,
                         usbhost_device_handle ifHub,
                         unsigned char index)
{
    usbhost_ioctl_cb_t usbhost_iocb;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_SET_PORT_RESET;
    // clear a downstream hub
    usbhost_iocb.handle.dif = ifHub;
    // clear port number passed
    usbhost_iocb.hub_port = index;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);
}
```

4.2.1.2.4.36 VOS_IOCTL_USBHUB_SET_PORT_POWER

Description

Sets the port power feature causing the port to be powered if this functionality is supported by the hub. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.13 of the USB Specification Revision 2.0 "Set Port Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is

assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure.

Returns

There is no return value from the hub.

Example

```
void setPortPowerFeature(VOS_HANDLE hUsbHost,
                        usbhost_device_handle ifHub,
                        unsigned char index)
{
    usbhost_ioctl_cb_t usbhost_iocb;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_SET_PORT_POWER;
    // clear a downstream hub
    usbhost_iocb.handle.dif = ifHub;
    // clear port number passed
    usbhost_iocb.hub_port = index;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);
}
```

4.2.1.2.4.37 VOS_IOCTL_USBHUB_CLEAR_PORT_POWER

Description

Clears the port power feature causing the port to remove power to a device depending on the functionality of the hub. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.2 of the USB Specification Revision 2.0 "Clear Port Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure.

Returns

There is no return value from the hub.

Example

```
void clearPortPowerFeature(VOS_HANDLE hUsbHost,
                        usbhost_device_handle ifHub)
{
    usbhost_ioctl_cb_t usbhost_iocb;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHUB_CLEAR_PORT_POWER;
    // clear a downstream hub
    usbhost_iocb.handle.dif = ifHub;
    // clear port number 3
    usbhost_iocb.hub_port = 3;
    vos_dev_ioctl(hUsbHost, &usbhost_iocb);
}
```

4.2.1.2.4.38 VOS_IOCTL_USBHUB_CLEAR_C_PORT_CONNECTION

Description

Clears the change bit for the port connection indicator. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.2 of the USB Specification Revision 2.0 "Clear Port Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure must be zero.

Returns

There is no return value from the hub.

4.2.1.2.4.39 VOS_IOCTL_USBHUB_CLEAR_C_PORT_ENABLE

Description

Clears the change bit for the port enable indicator. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.2 of the USB Specification Revision 2.0 "Clear Port Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure must be zero.

Returns

There is no return value from the hub.

4.2.1.2.4.40 VOS_IOCTL_USBHUB_CLEAR_C_PORT_SUSPEND

Description

Clears the change bit for the port suspend indicator. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.2 of the USB Specification Revision 2.0 "Clear Port Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure must be zero.

Returns

There is no return value from the hub.

4.2.1.2.4.41 VOS_IOCTL_USBHUB_CLEAR_C_PORT_OVERCURRENT

Description

Clears the change bit for the port overcurrent indicator. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.2 of the USB Specification Revision 2.0 "Clear Port Feature".

Parameters

The hub device is specified in the `dif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure must be

zero.

Returns

There is no return value from the hub.

4.2.1.2.4.42 VOS_IOCTL_USBHUB_CLEAR_C_PORT_RESET

Description

Clears the change bit for the port reset indicator. This can be the root hub on the VNC2 or a downstream hub. The hub features are described in section 11.24.2.2 of the USB Specification Revision 2.0 "Clear Port Feature".

Parameters

The hub device is specified in the `hif` member of the handle union. If it is NULL then the root hub is assumed.

The index to the port (port number) passed in the `hub_port` member of the IOCTL structure must be zero.

Returns

There is no return value from the hub.

4.2.1.2.5 usbhost_init()

Syntax

```
unsigned char usbhost_init (  
    unsigned char devNum_port_1,  
    unsigned char devNum_port_2,  
    usbhost_context_t *context  
);
```

Description

Initialise the USB Host driver and registers the driver with the Device Manager. There are two USB Host ports, both are controlled from a single instance of the driver.

Three threads are created to manage the USB Host controller. Memory is allocated for these threads from the heap. The size of the memory allocated is defined in the "USBHost.h" file as:

```
USBHOST_MEMORY_REQUIRED
```

There is also a memory requirement for device information storage depending on the the number of interfaces, endpoints and the number of concurrent transfers expected. This is described later in this topic.

The threads are started automatically when the scheduler is started.

One thread manages the root hub, another notifies application of the completion of transactions and the third will monitor downstream hub statuses for changes which require action.

Parameters

`devNum_port_1`

The device number to use when registering the USB Port 1 with the Device Manager is passed in the `devNum_port_1` parameter. If the USB Port 1 device is not to be configured then this parameter should be -1.

`devNum_port_2`

The device number to use when registering the USB Port 2 device with the Device Manager is passed in the `devNum_port_2` parameter. If the USB Port 2 device is not to be configured then this parameter should be -1.

`context`

The last parameter, `context`, is used to specify a number of interfaces available to be

configured on both USB controllers. It can also specify the number of endpoints to be configured, and the number of concurrent general and isochronous transfers allowed. If this is NULL then default values are used.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The context parameter must be of the form of the structure defined below:

```
typedef struct _usbhost_context_t {
    unsigned char if_count;
    unsigned char ep_count;
    unsigned char xfer_count;
    unsigned char iso_xfer_count;
} usbhost_context_t;
```

If any of the parameters are set to zero or the context pointer is NULL then a default value is used for that parameter. The allocations are shared between the two USB Host controllers and sizes should be set accordingly. The parameters are explained as follows:

if_count

Once the number of interfaces set in this parameter is reached then further devices will not be enumerated. A default value of 4 will be used as the maximum number of interfaces available if this parameter is set to zero.

ep_count

The number of endpoints to be reserved for use by all interfaces. This is in addition to a control endpoint for each interface which is automatically allocated. If zero, the default is calculated as the number of interfaces multiplied by 2. This will be sufficient for one IN and one OUT endpoint per interface.

xfer_count

This is the number of concurrent transactions on non-isochronous endpoints (control, bulk and interrupt) expected. If it is set to zero then the default of 2 will be used.

iso_xfer_count

This is the number of concurrent transactions on isochronous endpoints expected. If it is set to zero then the default of 2 will be used.

The following formula will establish the amount of memory required for a given configuration:

$$\text{total} = (\text{if_count} * 37) + (\text{ep_count} * 24) + (\text{xfer_count} * 10) + (\text{iso_xfer_count} * 28)$$

4.2.1.3 USB Slave Driver

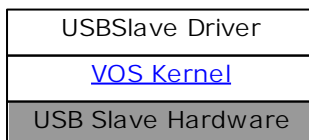
The USB Slave driver consists of one driver instance which can control both USB slave ports on the VNC2.

The USB Slave driver maintains a context for each configured USB port and presents an endpoint-based interface to the application. Requests to an endpoint must be routed through the correct driver handle to the appropriate USB Port.

The [usbslave_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

Driver Hierarchy

USB Slave Driver hierarchy:



Library Files

USBSlave.a

Header Files

USBSlave.h

4.2.1.3.1 USB Slave Concepts

Configuration

The driver can be configured to control either USB Port 1, USB Port 2 or both USB Ports. A unique VOS_DEVICE handle and `usbslave_ep_handle_t` handle is required for each endpoint. If a port is configured for use by the USB Host then it cannot be used by the USB Slave.

Once the USB Slave driver is configured it cannot be reconfigured.

Driver Handles

The USB Slave driver will require 2 unique device numbers to register both USB Ports with the device manager. If only one USB Port is configured then only one device number is required.

If both USB Ports are configured then the application will have 2 driver handles when both ports are opened, one for each USB Port and effectively 2 device drivers active. They should be treated separately by the application.

Endpoints

The interface to a USB port is based on operations on endpoints. Each device has a control endpoint and a variable number of IN and OUT endpoints. The current implementation supports bulk endpoints only.

Endpoints are accessed via a handle of type `usbslave_ep_handle_t`. The control endpoint (EP0) is treated as a special case. It handles SETUP packets and supports IN and OUT transactions, and separate handles are required for EP0 IN and EP0 OUT.

4.2.1.3.1.1 USB Slave Configuration

A device consists of a control endpoint and up to 7 IN or OUT endpoints. The [VOS_IOCTL_USBSLAVE_SET_ENDPOINT_MASKS](#) IOCTL is used to set the endpoint configuration for a device.

Each active endpoint on the slave must be enabled by setting a bit in the mask for either IN or OUT direction.

The control endpoint is always enabled.

4.2.1.3.1.2 USB Slave Obtaining an Endpoint Handle

A handle must be obtained prior to accessing an endpoint. The following IOCTLs are used to obtain a handle to an endpoint:

Control Endpoint	VOS_IOCTL_USBSLAVE_GET_CONTROL_ENDPOINT_HANDLE
IN Endpoint	VOS_IOCTL_USBSLAVE_GET_IN_ENDPOINT_HANDLE
OUT Endpoint	VOS_IOCTL_USBSLAVE_GET_OUT_ENDPOINT_HANDLE

Once a handle is obtained then data can be sent to the endpoint (for an OUT endpoint) and received from the endpoint (for IN endpoints).

4.2.1.3.1.3 USB Slave Enumeration

To support device enumeration, the following algorithm is required. All IOCTLs must be directed to the Control endpoint.

-
- Wait for a SETUP packet to be received ([VOS_IOCTL_USBSLAVE_WAIT_SETUP_RCVD](#))
 - Decode SETUP packet
 - Handle Standard Device Requests - mandatory requests that must be handled are:
 - Set Address, use [VOS_IOCTL_USBSLAVE_SET_ADDRESS](#) to change slave address
 - Set Configuration
 - Get Descriptor for both Device and Configuration descriptors
 - Handle Class Specific Requests (if any)
 - Handle Vendor Specific Requests (if any)
 - If SETUP packet has a data phase read or write further data with [VOS_IOCTL_USBSLAVE_SETUP_TRANSFER](#)
 - Use [VOS_IOCTL_USBSLAVE_SETUP_TRANSFER](#) again to acknowledge transaction with ACK phase

4.2.1.3.1.4 USB Slave Reading and Writing Data

The [VOS_IOCTL_USBSLAVE_TRANSFER](#) IOCTL is used for both IN and OUT endpoints. It must not be used on a Control endpoint.

4.2.1.3.2 USB Slave Return Codes

Status Codes

All calls to the USB Slave driver will return one of the following status codes.

USBSLAVE_OK

No error.

USBSLAVE_INVALID_PARAMETER

A parameter is incorrect or has a mistake.

USBSLAVE_ERROR

An unspecified error occurred.

4.2.1.3.3 USB Slave IOCTL Calls

Calls to the IOCTL functions for the USB Slave driver take the form:

```
typedef struct _usbslave_ioctl_cb_t {
    uint8 ioctl_code;
    uint8 ep;
    usbslave_ep_handle_t handle;
    // read buffer
    void *get;
    // write butter
    void *set;
    union {
        struct {
            uint8 in_mask;
            int16 out_mask;
        } set_ep_masks;
        struct {
            uint8 *buffer;
            int16 size;
            int16 bytes_transferred;
        } setup_or_bulk_transfer;
    } request;
} usbslave_ioctl_cb_t;
```

The following IOCTL request codes are supported by the USB Host driver.

[VOS_IOCTL_USBSLAVE_GET_STATE](#)

Get the state of the USB Slave device

<u>VOS_IOCTL_USBSLAVE_GET_CONTROL_ENDPOINT_HANDLE</u>	Obtain a handle to the control endpoint
<u>VOS_IOCTL_USBSLAVE_GET_IN_ENDPOINT_HANDLE</u>	Get a handle to an IN endpoint
<u>VOS_IOCTL_USBSLAVE_GET_OUT_ENDPOINT_HANDLE</u>	Get a handle to an OUT endpoint
<u>VOS_IOCTL_USBSLAVE_SET_ENDPOINT_MASKS</u>	Set the active endpoints for IN and OUT
<u>VOS_IOCTL_USBSLAVE_WAIT_SETUP_RCVD</u>	Wait for a SETUP packet to be received
<u>VOS_IOCTL_USBSLAVE_SETUP_TRANSFER</u>	Implement the DATA phase or ACK phase of a SETUP transfer
<u>VOS_IOCTL_USBSLAVE_SET_ADDRESS</u>	Set the address of the slave device
<u>VOS_IOCTL_USBSLAVE_TRANSFER</u>	Perform a transfer on an IN or OUT endpoint
<u>VOS_IOCTL_USBSLAVE_ENDPOINT_STALL</u>	Stall an endpoint
<u>VOS_IOCTL_USBSLAVE_ENDPOINT_CLEAR</u>	Clear an endpoint stall
<u>VOS_IOCTL_USBSLAVE_ENDPOINT_STATE</u>	Return the halted status of an endpoint
<u>VOS_IOCTL_USBSLAVE_SET_LOW_SPEED</u>	Set the USB Slave device to low speed

4.2.1.3.3.1 VOS_IOCTL_USBSLAVE_GET_STATE

Description

Returns the current state of the USB Slave hardware interface.

Parameters

There are no parameters.

Returns

The current state is returned to the location whose address is passed in the `get` field of the `usbslave_ioctl_cb_t` struct. It can be one of the following values:

<code>usbsStateNotAttached</code>	Not attached to a host controller.
<code>usbsStateAttached</code>	Attached to a host controller which is not configured.
<code>usbsStatePowered</code>	Attached to a host controller which is configured. Configuration of device can commence.
<code>usbsStateDefault</code>	Default mode where configuration sequence has performed a device reset operation.
<code>usbsStateAddress</code>	Address has been assigned by host.
<code>usbsStateConfigured</code>	Device is fully configured by host.
<code>usbsStateSuspended</code>	Device has been suspended by host.

Example

```
usbslave_ioctl_cb_t iocb;
unsigned char state;

iocb.ioctl_code = VOS_IOCTL_USBSLAVE_GET_STATE;
iocb.get = &state;
vos_dev_ioctl(hA,&iocb);

if (state == usbsStateConfigured)
{
    // device in action
}
```

4.2.1.3.3.2

VOS_IOCTL_USBSLAVE_GET_CONTROL_ENDPOINT_HANDLE

Description

Returns a handle for the control endpoint EP0. Separate handles are required for EP0 IN and EP0 OUT.

Parameters

The control endpoint identifier is passed in the `ep` field of the `usbslave_ioctl_cb_t` struct.

Control endpoint identifiers are defined as follows:

```
enum {
    USBSLAVE_CONTROL_SETUP,
    USBSLAVE_CONTROL_OUT,
    USBSLAVE_CONTROL_IN
};
```

Returns

The control endpoint handle is returned to the location whose address is passed in the `get` field of the `usbslave_ioctl_cb_t` struct.

Example

```
usbslave_ioctl_cb_t iocb;
usbslave_ep_handle_t in_ep0;
usbslave_ep_handle_t out_ep0;

iocb.ioctl_code = VOS_IOCTL_USBSLAVE_GET_CONTROL_ENDPOINT_HANDLE;
iocb.ep = USBSLAVE_CONTROL_IN;
iocb.get = &in_ep0;
vos_dev_ioctl(hA,&iocb);

iocb.ioctl_code = VOS_IOCTL_USBSLAVE_GET_CONTROL_ENDPOINT_HANDLE;
iocb.ep = USBSLAVE_CONTROL_OUT;
iocb.get = &out_ep0;
vos_dev_ioctl(hA,&iocb);
```

4.2.1.3.3.3 VOS_IOCTL_USBSLAVE_GET_IN_ENDPOINT_HANDLE

Description

Returns a handle for an IN endpoint.

Parameters

The endpoint address is passed in the `ep` field of the `usbslave_ioctl_cb_t` struct. Valid endpoint addresses are in the range 1-7.

Returns

The IN endpoint handle is returned to the location whose address is passed in the `get` field of the `usbslave_ioctl_cb_t` struct.

Example

```
usbslave_ioctl_cb_t iocb;
usbslave_ep_handle_t in_ep;

iocb.ioctl_code = VOS_IOCTL_USBSLAVE_GET_IN_ENDPOINT_HANDLE;
iocb.ep = 1;
iocb.get = &in_ep;
```

```
vos_dev_ioctl(hA,&iocb);
```

4.2.1.3.3.4 VOS_IOCTL_USBSLAVE_GET_OUT_ENDPOINT_HANDLE

Description

Returns a handle for an OUT endpoint.

Parameters

The endpoint address is passed in the `ep` field of the `usbslave_ioctl_cb_t` struct. Valid endpoint addresses are in the range 1-7.

Returns

The OUT endpoint handle is returned to the location whose address is passed in the `get` field of the `usbslave_ioctl_cb_t` struct.

Example

```
usbslave_ioctl_cb_t iocb;  
usbslave_ep_handle_t out_ep;  
  
iocb.ioctl_code = VOS_IOCTL_USBSLAVE_GET_OUT_ENDPOINT_HANDLE;  
iocb.ep = 2;  
iocb.get = &out_ep;  
vos_dev_ioctl(hA,&iocb);
```

4.2.1.3.3.5 VOS_IOCTL_USBSLAVE_SET_ENDPOINT_MASKS

Description

Sets the endpoint masks for the device. Endpoint masks represent endpoint addresses and types for the device.

Parameters

The endpoint mask for IN endpoints is passed in the `set_ep_masks.in_mask` field of the request union in the `usbslave_ioctl_cb_t` struct.

The endpoint mask for OUT endpoints is passed in the `set_ep_masks.out_mask` field of the request union in the `usbslave_ioctl_cb_t` struct.

Valid endpoint addresses are in the range 1-7. In the mask fields, bits set to 1 correspond to the addresses of the device's endpoints.

Returns

There is no return value.

Example

```
usbslave_ioctl_cb_t iocb;  
  
iocb.ioctl_code = VOS_IOCTL_USBSLAVE_SET_ENDPOINT_MASKS;  
iocb.request.set_ep_masks.in_mask = 0x02;           // EP1  
iocb.request.set_ep_masks.out_mask = 0x04;          // EP2  
vos_dev_ioctl(hA,&iocb);
```

4.2.1.3.3.6 VOS_IOCTL_USBSLAVE_WAIT_SETUP_RCVD

Description

Receives a SETUP packet. This call blocks until a SETUP packet is received from the host.

Parameters

The address of the buffer to receive the SETUP packet is passed in the `setup_or_bulk_transfer.buffer` field of the request union in the `usbslave_ioctl_cb_t` struct.

The size of the buffer to receive the SETUP packet is passed in the `setup_or_bulk_transfer.size` field of the request union in the `usbslave_ioctl_cb_t` struct. The USB Slave returns a 9-byte SETUP packet.

Returns

The buffer passed in the `setup_or_bulk_transfer.buffer` field of the request union in the `usbslave_ioctl_cb_t` struct contains the SETUP packet.

Example

```
usbslave_ioctl_cb_t iocb;  
unsigned char setup_buffer[9];  
  
iocb.ioctl_code = VOS_IOCTL_USBSLAVE_WAIT_SETUP_RCVD;  
iocb.request.setup_or_bulk_transfer.buffer = setup_buffer;  
iocb.request.setup_or_bulk_transfer.size = 9;  
vos_dev_ioctl(hA,&iocb);
```

4.2.1.3.3.7 VOS_IOCTL_USBSLAVE_SETUP_TRANSFER

Description

Performs a data phase or ACK phase for a SETUP transaction.

Parameters

The handle of the control endpoint on which the transaction is being performed is passed in the `handle` field of the `usbslave_ioctl_cb_t` struct.

The address of the buffer containing data for the transfer is passed in the `setup_or_bulk_transfer.buffer` field of the request union in the `usbslave_ioctl_cb_t` struct.

The size of the buffer containing data for the transfer is passed in the `setup_or_bulk_transfer.size` field of the request union in the `usbslave_ioctl_cb_t` struct.

Returns

There is no return value.

Example

```
usbslave_ioctl_cb_t iocb;  
  
iocb.ioctl_code = VOS_IOCTL_USBSLAVE_SETUP_TRANSFER;  
iocb.handle = in_ep0;  
iocb.request.setup_or_bulk_transfer.buffer = (void *) 0;  
iocb.request.setup_or_bulk_transfer.size = 0;  
vos_dev_ioctl(hA,&iocb);
```

4.2.1.3.3.8 VOS_IOCTL_USBSLAVE_SET_ADDRESS

Description

Sets the USB address for the device. The USB host assigns the device address during enumeration, and this IOCTL is used to set the USB slave port hardware to respond to that address. This IOCTL should be used when processing the USB standard device request, SET_ADDRESS.

Parameters

The address is passed in the `set` field of the `usbslave_ioctl_cb_t` struct.

Returns

There is no return value.

Example

```
void set_address_request(uint8 addr)
{
    usbslave_ioctl_cb_t iocb;

    iocb.ioctl_code = VOS_IOCTL_USBSLAVE_SET_ADDRESS;
    iocb.set = (void *) addr;
    vos_dev_ioctl(hA,&iocb);
}
```

4.2.1.3.3.9 VOS_IOCTL_USBSLAVE_TRANSFER

Description

Performs a transfer to a non-control endpoint. This IOCTL is used for bulk transfers on both IN and OUT endpoints. When used on an OUT endpoint, this IOCTL blocks until data is received from the host.

Parameters

The handle of the endpoint on which the transaction is being performed is passed in the `handle` field of the `usbslave_ioctl_cb_t` struct.

The address of the buffer for the transfer is passed in the `setup_or_bulk_transfer.buffer` field of the request union in the `usbslave_ioctl_cb_t` struct.

The size of the buffer containing data for the transfer is passed in the `setup_or_bulk_transfer.size` field of the request union in the `usbslave_ioctl_cb_t` struct.

Returns

The number of bytes transferred is returned in the `setup_or_bulk_transfer.bytes_transferred` field of the request union in the `usbslave_ioctl_cb_t` struct.

For bulk transfer requests on OUT endpoints, the data is returned in the buffer whose address was passed in the `setup_or_bulk_transfer.buffer` field of the request union in the `usbslave_ioctl_cb_t` struct.

Example

```
usbslave_ioctl_cb_t iocb;
usbslave_ep_handle_t in_ep;
usbslave_ep_handle_t out_ep;
char *str = "hello, world";
uint8 out_buffer[64];

iocb.ioctl_code = VOS_IOCTL_USBSLAVE_BULK_TRANSFER;
iocb.handle = in_ep;
iocb.request.setup_or_bulk_transfer.buffer = (unsigned char *) str;
iocb.request.setup_or_bulk_transfer.size = 12;
vos_dev_ioctl(hA,&iocb);

iocb.ioctl_code = VOS_IOCTL_USBSLAVE_BULK_TRANSFER;
iocb.handle = out_ep;
iocb.request.setup_or_bulk_transfer.buffer = out_buffer;
iocb.request.setup_or_bulk_transfer.size = 64;
iocb.request.setup_or_bulk_transfer.bytes_transferred = 0;
vos_dev_ioctl(hA,&iocb);

while (iocb.request.setup_or_bulk_transfer.bytes_transferred) {

    // process bytes received from host
```

}

4.2.1.3.3.10 VOS_IOCTL_USBSLAVE_ENDPOINT_STALL

Description

Force an endpoint to stall on the USB Slave device. An IN, OUT or control endpoint may be stalled. This may be used on the control endpoint when a device does not support a certain SETUP request or on other endpoints as required. If an endpoint it halted then it will return a STALL to a request from the host.

Parameters

The endpoint identifier is passed in the `ep` field of the `usbslave_ioctl_cb_t` struct.

Returns

There is no return value.

Example

```
usbslave_ioctl_cb_t iocb;  
  
iocb.ioctl_code = VOS_IOCTL_USBSLAVE_ENDPOINT_STALL;  
iocb.ep = 1;  
vos_dev_ioctl(hA,&iocb);
```

4.2.1.3.3.11 VOS_IOCTL_USBSLAVE_ENDPOINT_CLEAR

Description

Remove a halt state on the USB Slave device. An IN, OUT or control endpoint may be stalled but only IN and OUT endpoints can be cleared by this IOCTL.

Parameters

The endpoint identifier is passed in the `ep` field of the `usbslave_ioctl_cb_t` struct.

Returns

There is no return value.

Example

```
usbslave_ioctl_cb_t iocb;  
  
iocb.ioctl_code = VOS_IOCTL_USBSLAVE_ENDPOINT_CLEAR;  
iocb.ep = 1;  
vos_dev_ioctl(hA,&iocb);
```

4.2.1.3.3.12 VOS_IOCTL_USBSLAVE_ENDPOINT_STATE

Description

Returns the halt state of an endpoint on the USB Slave device. If an endpoint it halted then it will return a STALL to a request from the host.

Parameters

The endpoint identifier is passed in the `ep` field of the `usbslave_ioctl_cb_t` struct.

Returns

The return value is zero if the endpoint it not halted and non-zero if it is halted.

Example

```
usbslave_ioctl_cb_t iocb;  
char x;  
  
iocb.ioctl_code = VOS_IOCTL_USBSLAVE_ENDPOINT_STATE;  
iocb.ep = 1;  
iocb.get = &x;  
vos_dev_ioctl(hA,&iocb);
```

4.2.1.3.3.13 VOS_IOCTL_USBSLAVE_SET_LOW_SPEED

Description

Sets the USB Slave device to Low Speed. This is non-reversible. This should be performed as soon as possible after opening the USB Slave device and before host enumeration occurs.

Parameters

There are no parameters.

Returns

There is no return value.

Example

```
usbslave_ioctl_cb_t iocb;  
  
iocb.ioctl_code = VOS_IOCTL_USBSLAVE_SET_LOW_SPEED;  
vos_dev_ioctl(hA,&iocb);
```

4.2.1.3.4 usbslave_init()

Syntax

```
unsigned char usbslave_init (  
    unsigned char slavePort,  
    unsigned char devNum);
```

Description

Initialise the USB Slave driver and registers the driver with the Device Manager. There are two USB Slave ports, both are controlled from a single instance of the driver. However, the `usbslave_init()` function must be called for each slave port used.

Parameters

slavePort
The slave port to initialise to use the device number passed in `devNum`. This can be either `USBSLAVE_PORT_A` or `USBSLAVE_PORT_B`.

devNum
The device number to use when registering this USB Slave port with the Device Manager is passed in the `devNum` parameter.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The function must be called twice to configure both USB Slave ports. If a port is configured by the USB Host then it cannot be used for the USB Slave. The USB Slave has no thread memory

requirements.

4.2.1.4 GPIO Driver

VNC2 provides up to 40 general purpose IO pins (GPIOs). The 40 available signals are arranged in 5 groups of 8 pins referred to as ports. The ports have been assigned identifiers GPIO_PORT_A, GPIO_PORT_B, GPIO_PORT_C, GPIO_PORT_D and GPIO_PORT_E.

Individual pins on each port can be configured as either input or output by setting a mask. These can be switched at runtime if required.

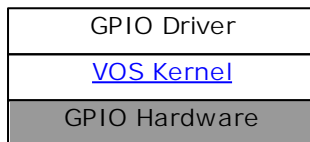
The GPIO block also supports configurable interrupts. GPIO_PORT_A supports a port-level interrupt that will trigger on any pin state change for that port. GPIO_PORT_B allows 4 individually configurable interrupts to be assigned to any of the GPIO_PORT_B pins. The other GPIO ports do not support interrupts.

The GPIO driver supports the [vos_dev_read\(\)](#) and [vos_dev_write\(\)](#) interfaces for reading values from and writing values to the GPIO interface.

The [gpio_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

Driver Hierarchy

GPIO Driver hierarchy:



Library Files

GPIO.a

Header Files

GPIO.h

4.2.1.4.1 GPIO Return Codes

Status Codes

All calls to the GPIO driver will return one of the following status codes.

- GPIO_OK
The command completed successfully.
- GPIO_INVALID_PORT_IDENTIFIER
The requested GPIO port is invalid.
- GPIO_INVALID_PARAMETER
There was an error or problem with a parameter sent to the driver.
- GPIO_INTERRUPT_NOT_ENABLED
The interrupt to wait on is not enabled
- GPIO_ERROR
An unspecified error occurred.

4.2.1.4.2 GPIO Driver Read and Write Operations

Description

The GPIO driver supports read and write operations for each GPIO port via the standard VOS device manager functions. Since each port is 1 byte wide, the buffer for each read operation is a single byte.

In the case of write operations, a buffer of many bytes may be written to the GPIO port and they will

be written in succession to the GPIO port to allow bit-banging functionality.

Example

```
unsigned char portDataOut, portDataIn;
unsigned short num_written, num_read;

for (i = 0; i < 255; i++) {
    portDataOut = i;
    if (vos_dev_write(handle, &portDataOut, 1, &num_written) == GPIO_OK) {
        if (vos_dev_read(handle, &portDataIn, 1, &num_read) == GPIO_OK) {
            // same value read as written
        }
        else {
            // different value read from that written
        }
    }
}
```

4.2.1.4.3 GPIO IOCTL Calls

Calls to the GPIO driver's IOCTL method take the form:

```
typedef struct _gpio_ioctl_cb_t {
    unsigned char ioctl_code;
    unsigned char value;
} gpio_ioctl_cb_t;
```

The following IOCTL request codes are supported by the GPIO driver.

<u>VOS_IOCTL_GPIO_SET_MASK</u>	Set pins to either input (0) or output (1)
<u>VOS_IOCTL_GPIO_SET_PROG_INT0_PIN</u>	Configure programmable interrupt 0 pin
<u>VOS_IOCTL_GPIO_SET_PROG_INT1_PIN</u>	Configure programmable interrupt 1 pin
<u>VOS_IOCTL_GPIO_SET_PROG_INT2_PIN</u>	Configure programmable interrupt 2 pin
<u>VOS_IOCTL_GPIO_SET_PROG_INT3_PIN</u>	Configure programmable interrupt 3 pin
<u>VOS_IOCTL_GPIO_SET_PROG_INT0_MODE</u>	Configure programmable interrupt 0 mode
<u>VOS_IOCTL_GPIO_SET_PROG_INT1_MODE</u>	Configure programmable interrupt 1 mode
<u>VOS_IOCTL_GPIO_SET_PROG_INT2_MODE</u>	Configure programmable interrupt 2 mode
<u>VOS_IOCTL_GPIO_SET_PROG_INT3_MODE</u>	Configure programmable interrupt 3 mode
<u>VOS_IOCTL_GPIO_SET_PORT_INT</u>	Configure port interrupt
<u>VOS_IOCTL_GPIO_WAIT_ON_INT0</u>	Wait on interrupt 0 firing
<u>VOS_IOCTL_GPIO_WAIT_ON_INT1</u>	Wait on interrupt 1 firing
<u>VOS_IOCTL_GPIO_WAIT_ON_INT2</u>	Wait on interrupt 2 firing
<u>VOS_IOCTL_GPIO_WAIT_ON_INT3</u>	Wait on interrupt 3 firing
<u>VOS_IOCTL_GPIO_WAIT_ON_PORT_INT</u>	Wait on port interrupt firing

4.2.1.4.3.1 VOS_IOCTL_GPIO_SET_MASK

Description

Define the pins on the selected GPIO port which are input (0) and those which are output (1).

Parameters

Set the value member of the GPIO IOCTL control block with the desired bit-mask.

Returns

This IOCTL will always return GPIO_OK.

Example

```
/* set pins 0, 1, 2 and 3 to input, all other pins output */
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_MASK;
gpio_iocb.value = 0xF0;
vos_dev_ioctl(hGpioA,&gpio_iocb);
```

4.2.1.4.3.2 VOS_IOCTL_GPIO_SET_PROG_INT0_PIN

Description

Define the GPIO port signal that is associated with configurable interrupt 0.

Parameters

Set the value member of the GPIO IOCTL control block with the number of the GPIO_PORT_B pin to be used with configurable interrupt 0. A valid value is a pin number in the range 0 to 7.

Returns

If the parameter is incorrect then GPIO_INVALID_PARAMETER will be returned.

Example

```
/* associate interrupt 0 with pin 2 of GPIO port B */
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_PROG_INT0_PIN;
gpio_iocb.value = GPIO_PIN_2;
vos_dev_ioctl(hGpioB,&gpio_iocb);
```

4.2.1.4.3.3 VOS_IOCTL_GPIO_SET_PROG_INT1_PIN

Description

Define the GPIO port signal that is associated with configurable interrupt 1.

Parameters

Set the value member of the GPIO IOCTL control block with the number of the GPIO_PORT_B pin to be used with configurable interrupt 1. A valid value is a pin number in the range 0 to 7.

Returns

If the parameter is incorrect then GPIO_INVALID_PARAMETER will be returned.

Example

```
/* associate interrupt 1 with pin 5 of GPIO port B */
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_PROG_INT1_PIN;
gpio_iocb.value = GPIO_PIN_5;
vos_dev_ioctl(hGpioB,&gpio_iocb);
```

4.2.1.4.3.4 VOS_IOCTL_GPIO_SET_PROG_INT2_PIN

Description

Define the GPIO port signal that is associated with configurable interrupt 2.

Parameters

Set the value member of the GPIO IOCTL control block with the number of the GPIO_PORT_B pin to be used with configurable interrupt 2. A valid value is a pin number in the range 0 to 7.

Returns

If the parameter is incorrect then GPIO_INVALID_PARAMETER will be returned.

Example

```
/* associate interrupt 2 with pin 1 of GPIO port B */
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_PROG_INT2_PIN;
gpio_iocb.value = GPIO_PIN_1;
vos_dev_ioctl(hGpioB,&gpio_iocb);
```

4.2.1.4.3.5 VOS_IOCTL_GPIO_SET_PROG_INT3_PIN

Description

Define the GPIO port signal that is associated with configurable interrupt 3.

Parameters

Set the value member of the GPIO IOCTL control block with the number of the GPIO_PORT_B pin to be used with configurable interrupt 3. A valid value is a pin number in the range 0 to 7.

Returns

If the parameter is incorrect then GPIO_INVALID_PARAMETER will be returned.

Example

```
/* associate interrupt 3 with pin 7 of GPIO port B */
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_PROG_INT3_PIN;
gpio_iocb.value = GPIO_PIN_7;
vos_dev_ioctl(hGpioB,&gpio_iocb);
```

4.2.1.4.3.6 VOS_IOCTL_GPIO_SET_PROG_INT0_MODE

Description

Define the condition that will cause configurable interrupt 0 to trigger.

Parameters

Set the value member of the GPIO IOCTL control block with the condition that will cause configurable interrupt 0 to trigger. Valid values are:

```
GPIO_INT_ON_POS_EDGE
GPIO_INT_ON_NEG_EDGE
GPIO_INT_ON_ANY_EDGE
GPIO_INT_ON_HIGH_STATE
GPIO_INT_ON_LOW_STATE
GPIO_INT_DISABLE
```

Returns

If the mode parameter is incorrect then GPIO_INVALID_PARAMETER will be returned.

Example

```
/* generate an interrupt on a positive edge */
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_PROG_INT0_MODE;
gpio_iocb.value = GPIO_INT_ON_POS_EDGE;
vos_dev_ioctl(hGpioB,&gpio_iocb);
```

4.2.1.4.3.7 VOS_IOCTL_GPIO_SET_PROG_INT1_MODE

Description

Define the condition that will cause configurable interrupt 1 to trigger.

Parameters

Set the value member of the GPIO IOCTL control block with the condition that will cause configurable interrupt 1 to trigger. Valid values are:

```
GPIO_INT_ON_POS_EDGE  
GPIO_INT_ON_NEG_EDGE  
GPIO_INT_ON_ANY_EDGE  
GPIO_INT_ON_HIGH_STATE  
GPIO_INT_ON_LOW_STATE  
GPIO_INT_DISABLE
```

Returns

If the mode parameter is incorrect then GPIO_INVALID_PARAMETER will be returned.

Example

```
/* generate an interrupt on a high state */  
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_PROG_INT1_MODE;  
gpio_iocb.value = GPIO_INT_ON_HIGH_STATE;  
vos_dev_ioctl(hGpioB, &gpio_iocb);
```

4.2.1.4.3.8 VOS_IOCTL_GPIO_SET_PROG_INT2_MODE

Description

Define the condition that will cause configurable interrupt 2 to trigger.

Parameters

Set the value member of the GPIO IOCTL control block with the condition that will cause configurable interrupt 2 to trigger. Valid values are:

```
GPIO_INT_ON_POS_EDGE  
GPIO_INT_ON_NEG_EDGE  
GPIO_INT_ON_ANY_EDGE  
GPIO_INT_ON_HIGH_STATE  
GPIO_INT_ON_LOW_STATE  
GPIO_INT_DISABLE
```

Returns

If the mode parameter is incorrect then GPIO_INVALID_PARAMETER will be returned.

Example

```
/* generate an interrupt on any edge */  
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_PROG_INT2_MODE;  
gpio_iocb.value = GPIO_INT_ON_ANY_EDGE;  
vos_dev_ioctl(hGpioB, &gpio_iocb);
```

4.2.1.4.3.9 VOS_IOCTL_GPIO_SET_PROG_INT3_MODE

Description

Define the condition that will cause configurable interrupt 3 to trigger.

Parameters

Set the value member of the GPIO IOCTL control block with the condition that will cause configurable interrupt 3 to trigger. Valid values are:

```
GPIO_INT_ON_POS_EDGE  
GPIO_INT_ON_NEG_EDGE  
GPIO_INT_ON_ANY_EDGE
```

```
GPIO_INT_ON_HIGH_STATE  
GPIO_INT_ON_LOW_STATE  
GPIO_INT_DISABLE
```

Returns

If the mode parameter is incorrect then GPIO_INVALID_PARAMETER will be returned.

Example

```
/* disable interrupt 3 */  
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_PROG_INT3_MODE;  
gpio_iocb.value = GPIO_INT_DISABLE;  
vos_dev_ioctl(hGpioB, &gpio_iocb);
```

4.2.1.4.3.10 VOS_IOCTL_GPIO_WAIT_ON_INT0

Description

Block thread execution until configurable interrupt 0 has triggered. Note that the interrupt is disabled immediately after signalling and must be re-enabled with the [VOS_IOCTL_GPIO_SET_PROG_INT0_MODE](#) IOCTL to detect further interrupts.

Parameters

None.

Returns

If configurable interrupt 0 is not enabled GPIO_INTERRUPT_NOT_ENABLED is returned.

Example

```
/* wait for interrupt 0 to fire */  
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_WAIT_ON_INT0;  
vos_dev_ioctl(hGpioB, &gpio_iocb);
```

4.2.1.4.3.11 VOS_IOCTL_GPIO_WAIT_ON_INT1

Description

Block thread execution until configurable interrupt 1 has triggered. Note that the interrupt is disabled immediately after signalling and must be re-enabled with the [VOS_IOCTL_GPIO_SET_PROG_INT1_MODE](#) IOCTL to detect further interrupts.

Parameters

None.

Returns

If configurable interrupt 1 is not enabled GPIO_INTERRUPT_NOT_ENABLED is returned.

Example

```
/* wait for interrupt 1 to fire */  
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_WAIT_ON_INT1;  
vos_dev_ioctl(hGpioB, &gpio_iocb);
```

4.2.1.4.3.12 VOS_IOCTL_GPIO_WAIT_ON_INT2

Description

Block thread execution until configurable interrupt 2 has triggered. Note that the interrupt is disabled immediately after signalling and must be re-enabled with the

[VOS_IOCTL_GPIO_SET_PROG_INT2_MODE](#) IOCTL to detect further interrupts.

Parameters

None.

Returns

If configurable interrupt 2 is not enabled GPIO_INTERRUPT_NOT_ENABLED is returned.

Example

```
/* wait for interrupt 2 to fire */
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_WAIT_ON_INT2;
vos_dev_ioctl(hGpioB, &gpio_iocb);
```

4.2.1.4.3.13 VOS_IOCTL_GPIO_WAIT_ON_INT3

Description

Block thread execution until configurable interrupt 3 has triggered. Note that the interrupt is disabled immediately after signalling and must be re-enabled with the [VOS_IOCTL_GPIO_SET_PROG_INT3_MODE](#) IOCTL to detect further interrupts.

Parameters

None.

Returns

If configurable interrupt 3 is not enabled GPIO_INTERRUPT_NOT_ENABLED is returned.

Example

```
/* wait for interrupt 3 to fire */
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_WAIT_ON_INT3;
vos_dev_ioctl(hGpioB, &gpio_iocb);
```

4.2.1.4.3.14 VOS_IOCTL_GPIO_SET_PORT_INT

Description

Enable or disable the port interrupt on GPIO port A. This will detect any change in state on any of the 8 pins of port A.

Parameters

Set the value member of the GPIO IOCTL control block to enable or disable the port interrupt on GPIO port A. Valid values are:

```
GPIO_PORT_INT_ENABLE
GPIO_PORT_INT_DISABLE
```

Returns

If the parameter is incorrect then GPIO_INVALID_PARAMETER will be returned.

Example

```
/* enable interrupts on GPIO port A */
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_PORT_INT;
gpio_iocb.value = GPIO_PORT_INT_ENABLE;
vos_dev_ioctl(hGpioA, &gpio_iocb);
```

4.2.1.4.3.15 VOS_IOCTL_GPIO_WAIT_ON_PORT_INT

Description

Block thread execution until the port interrupt on GPIO port A has triggered. Note that the interrupt is disabled immediately after signalling and must be re-enabled with the [VOS_IOCTL_GPIO_SET_PORT_INT](#) IOCTL to detect further interrupts.

Parameters

None.

Returns

If the GPIO port A interrupt is not enabled GPIO_INTERRUPT_NOT_ENABLED is returned.

Example

```
/* wait for port interrupt to fire */
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_WAIT_PORT_INT;
vos_dev_ioctl(hGpioA,&gpio_iocb);
```

4.2.1.4.4 gpio_init()

Syntax

```
unsigned char gpio_init (
    unsigned char devNum,
    gpio_context_t* context
);
```

Description

Initialise the GPIO driver for the port specified in the context and registers the driver with the Device Manager.

Parameters

devNum
The device number to use when registering the driver with the Device Manager is passed in the devNum parameter.

context
The second parameter, context, is used to specify the GPIO port that is being registered with the device manager. If the pointer is NULL then the default port of GPIO_PORT_A is used.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The context parameter must be of the form of the structure defined below:

```
typedef struct _gpio_context_t {
    unsigned char port_identifier;
} gpio_context_t;
```

Valid values for the port_identifier member are GPIO_PORT_A, GPIO_PORT_B, GPIO_PORT_C, GPIO_PORT_D or GPIO_PORT_E.

4.2.1.4.5 GPIO Example

```
// *****
```

```
// GPIO example
// This application uses GPIO port A to output a pattern on LEDs.
// The LEDs illuminate 1 at a time shifting from the least
// significant bit to the most significant bit, then back to the least
// significant bit.
// *****

#include "vos.h"
#include "GPIO.h"

#define SIZEOF_tcb                0x400
#define NUMBER_OF_DEVICES        1
/* Device definitions*/
#define VOS_DEV_GPIO              0

// *****
// Device initialisation
// *****
void init_devices(VOS_HANDLE* handle) {

    gpio_ioctl_cb_t gpio_iocb;
    gpio_context_t gpioCtx;
    unsigned char packageType;

    if (NUMBER_OF_DEVICES != 0) {

        //*****
        // INITIALISE IOMUX
        //*****
        // route GPIO port A signals as required

        packageType = vos_get_package_type();

        if (packageType == VINCULUM_II_48_PIN){
            // GPIO port A bit 0 to pin 45
            vos_iomux_define_output(45,IOMUX_OUT_GPIO_PORT_A_0); //LED7
            // GPIO port A bit 1 to pin 46
            vos_iomux_define_output(46,IOMUX_OUT_GPIO_PORT_A_1); //LED6
            // GPIO port A bit 2 to pin 47
            vos_iomux_define_output(47,IOMUX_OUT_GPIO_PORT_A_2); //LED5
            // GPIO port A bit 3 to pin 48
            vos_iomux_define_output(48,IOMUX_OUT_GPIO_PORT_A_3); //LED4
            // GPIO port A bit 7 to pin 44
            vos_iomux_define_output(44,IOMUX_OUT_GPIO_PORT_A_7); //LED3
            // GPIO port A bit 6 to pin 43
            vos_iomux_define_output(43,IOMUX_OUT_GPIO_PORT_A_6); //LED2
            // GPIO port A bit 5 to pin 42
            vos_iomux_define_output(42,IOMUX_OUT_GPIO_PORT_A_5); //LED1
            // GPIO port A bit 4 to pin 41
            vos_iomux_define_output(41,IOMUX_OUT_GPIO_PORT_A_4); //LED0
        }
        else if (packageType == VINCULUM_II_64_PIN) {
            // GPIO port A bit 0 to pin 61
            vos_iomux_define_output(61,IOMUX_OUT_GPIO_PORT_A_0); //LED7
            // GPIO port A bit 1 to pin 62
            vos_iomux_define_output(62,IOMUX_OUT_GPIO_PORT_A_1); //LED6
            // GPIO port A bit 2 to pin 63
            vos_iomux_define_output(63,IOMUX_OUT_GPIO_PORT_A_2); //LED5
            // GPIO port A bit 3 to pin 64
            vos_iomux_define_output(64,IOMUX_OUT_GPIO_PORT_A_3); //LED4
            // GPIO port A bit 7 to pin 60
            vos_iomux_define_output(60,IOMUX_OUT_GPIO_PORT_A_7); //LED3
            // GPIO port A bit 6 to pin 59
            vos_iomux_define_output(59,IOMUX_OUT_GPIO_PORT_A_6); //LED2
            // GPIO port A bit 5 to pin 58
            vos_iomux_define_output(58,IOMUX_OUT_GPIO_PORT_A_5); //LED1
```

```
// GPIO port A bit 4 to pin 57
vos_iomux_define_output(57,IOMUX_OUT_GPIO_PORT_A_4); //LED0
}

//*****
// INITIALISE GPIO PARAMETERS - use GPIO port A
//*****
gpioCtx.port_identifier = GPIO_PORT_A;
gpio_init(VOS_DEV_GPIO, &gpioCtx);
// open gpio and get a handle
*handle = vos_dev_open(VOS_DEV_GPIO);

// set all pins to output
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_MASK;
gpio_iocb.value = 0xFF;
vos_dev_ioctl(*handle, &gpio_iocb);
}

}

// *****
// Kitt thread
// *****
void kitt(VOS_HANDLE handle) {

    unsigned char portData;
    unsigned char value;
    unsigned char i;
    unsigned char direction;

    unsigned short num_written;

    direction = 0;
    value = 1;
    i = 0;

    portData = value;

    vos_dev_write(handle,&portData,1,&num_written);

    while (1) {
        vos_delay_msecs(50); // wait for a bit
        if (direction == 0) {
            // counting up
            i++;
            if (i == 7)
                direction = 1;
        }
        else {
            // counting down
            i--;
            if (i == 0)
                direction = 0;
        }

        value = (1 << i);

        portData = value;
        vos_dev_write(handle,&portData,1,&num_written);
    }

}

// *****
```

```
// Main application
// *****
void main(void) {

    VOS_HANDLE hGpio = NULL;

    // initialise vos
    vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);

    vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);

    // initialise devices (APPLICATION SPECIFIC)
    init_devices(&hGpio);

    // initialise threads
    // kitt thread
    vos_create_thread(
        31,
        sizeof_tcb,
        &kitt,
        sizeof(VOS_HANDLE),
        (VOS_HANDLE)hGpio
    );

    vos_start_scheduler();

main_loop:
    goto main_loop;
}
```

4.2.1.5 Timer Driver

VNC2 provides 4 timers. One of these timers is reserved for the VOS kernel task scheduler. The remaining timers are accessible via the timer driver. The user timers are identified as TIMER_0, TIMER_1 and TIMER_2.

Timers can be individually configured to have a tick size of 1ms or 1us, count up or count down and be continuous or single-shot. Each timer can be assigned a different initial/final value and independently started or stopped.

The [tmr_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

Driver Hierarchy

Timer Driver hierarchy:

Timer Driver
VOS Kernel
Timer Hardware

Library Files

Timer.a

Header Files

Timer.h

4.2.1.5.1 Timer Return Codes

Status Codes

All calls to the timer driver will return one of the following status codes.

TIMER_OK

The command completed successfully.

TIMER_INVALID_IDENTIFIER

The requested timer is invalid.

TIMER_INVALID_PARAMETER

There was an error or problem with a parameter sent to the driver.

TIMER_ERROR

An unspecified error occurred.

4.2.1.5.2 Timer IOCTL Calls

Calls to the timer driver's IOCTL method take the form:

```
typedef struct _tmr_ioctl_cb_t {  
    unsigned char ioctl_code;  
    unsigned short param;  
} tmr_ioctl_cb_t;
```

The following IOCTL request codes are supported by the timer driver.

VOS_IOCTL_TIMER_SET_TICK_SIZE	Set the base tick for the timer, 1us or 1ms
VOS_IOCTL_TIMER_SET_COUNT	Set initial/final value of timer
VOS_IOCTL_TIMER_SET_DIRECTION	Specify if timer counts up or down
VOS_IOCTL_TIMER_SET_MODE	Specify single-shot or continuous
VOS_IOCTL_TIMER_START	Start the timer
VOS_IOCTL_TIMER_GET_CURRENT_COUNT	Retrieve the current timer count
VOS_IOCTL_TIMER_STOP	Stop the timer
VOS_IOCTL_TIMER_WAIT_ON_COMPLETE	Wait on the timer completing

4.2.1.5.2.1 VOS_IOCTL_TIMER_SET_TICK_SIZE

Description

Specify whether the timer should use 1ms or 1us as the base tick. If using 1ms, the timer is linked to the shared timer prescaler. The prescaler is automatically configured by the timer driver to have a value of 1ms for the current system clock frequency.

Parameters

Set the param member of the timer IOCTL control block with the tick value. Valid values are:

```
TIMER_TICK_US  
TIMER_TICK_MS
```

Returns

If the parameter is incorrect then TIMER_INVALID_PARAMETER will be returned.

Example

```
/* set timer 0 to use 1ms ticks */  
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_SET_TICK_SIZE;  
tmr_iocb.param = TIMER_TICK_MS;  
vos_dev_ioctl(hTimer0,&tmr_iocb);
```

4.2.1.5.2.2 VOS_IOCTL_TIMER_SET_COUNT

Description

Set the timer value at which the timer should generate an interrupt and roll over to its starting value.

Parameters

Set the param member of the timer IOCTL control block with the count value. Valid values are dependent on the system clock frequency and the timer tick size. For example, a VNC2 running at 48MHz which the timer tick specified as TIMER_TICK_US, the maximum count value that can be supported is 0x0555 - this is determined by the maximum unsigned short value (0xFFFF) divided by the number of system clock ticks corresponding to 1us. Similarly, when configured for us ticks at 24MHz the maximum count is 0x0AAA and at 12MHz the maximum count is 0x1555.

The maximum permitted count when configured for TIMER_TICK_MS is 0xFFFF.

Returns

If the parameter is incorrect then TIMER_INVALID_PARAMETER will be returned.

Example

```
/* set timer 2 to have an initial value of 3000ms (previously configured for ms tick) */
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_SET_COUNT;
tmr_iocb.param = 0x3000; // 3 seconds
vos_dev_ioctl(hTimer2,&tmr_iocb);
```

4.2.1.5.2.3 VOS_IOCTL_TIMER_SET_DIRECTION

Description

Specify whether the timer should count up to its count value or count down from it.

Parameters

Set the param member of the timer IOCTL control block with the direction value. Valid values are:

```
TIMER_COUNT_DOWN
TIMER_COUNT_UP
```

Returns

If the parameter is incorrect then TIMER_INVALID_PARAMETER will be returned.

Example

```
/* set timer 0 to count down from its count value */
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_SET_DIRECTION;
tmr_iocb.param = TIMER_COUNT_DOWN;
vos_dev_ioctl(hTimer0,&tmr_iocb);
```

4.2.1.5.2.4 VOS_IOCTL_TIMER_SET_MODE

Description

Specify whether the timer is a single-shot timer or a continuous timer. If in continuous mode, the timer will generate in interrupt at the end of a cycle, reset itself to its initial value and continue counting.

Parameters

Set the param member of the timer IOCTL control block with the mode value. Valid values are:

```
TIMER_MODE_CONTINUOUS
TIMER_MODE_SINGLE_SHOT
```

Returns

If the parameter is incorrect then TIMER_INVALID_PARAMETER will be returned.

Example

```
/* set timer 2 to run in continuous mode */
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_SET_MODE;
tmr_iocb.param = TIMER_MODE_CONTINUOUS;
vos_dev_ioctl(hTimer2,&tmr_iocb);
```

4.2.1.5.2.5 VOS_IOCTL_TIMER_START

Description

Starts a timer. An interrupt will be generated when the timer reaches its final value. If in continuous mode, the timer will reset its count and continue until stopped with the [VOS_IOCTL_TIMER_STOP](#) IOCTL.

Parameters

None.

Returns

This IOCTL will always return `TIMER_OK`.

Example

```
/* start timer 1 */
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_START;
vos_dev_ioctl(hTimer1,&tmr_iocb);
```

4.2.1.5.2.6 VOS_IOCTL_TIMER_GET_CURRENT_COUNT

Description

Retrieves the current value of the timer in units of the base tick. This is either in ms or us depending on how the timer has been configured.

Parameters

There are no parameters to set.

Returns

The current value of the timer in `param`.

Example

```
unsigned short currentCount;
/* get current value of timer 1 */
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_GET_CURRENT_COUNT;
vos_dev_ioctl(hTimer2,&tmr_iocb);
currentCount = tmr_iocb.param;
```

4.2.1.5.2.7 VOS_IOCTL_TIMER_STOP

Description

Stops a timer which was previously started with the [VOS_IOCTL_TIMER_START](#) IOCTL.

Parameters

None.

Returns

This IOCTL will always return `TIMER_OK`.

Example

```
/* stop timer 1 */
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_STOP;
vos_dev_ioctl(hTimer1,&tmr_iocb);
```

4.2.1.5.2.8 VOS_IOCTL_TIMER_WAIT_ON_COMPLETE

Description

Block thread execution until the current timer has finished its cycle and signalled an interrupt.

Parameters

None.

Returns

This IOCTL will always return TIMER_OK.

Example

```
/* wait for timer cycle to complete */
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_WAIT_ON_COMPLETE;
vos_dev_ioctl(hTimer1,&tmr_iocb);
```

4.2.1.5.3 tmr_init()

Syntax

```
unsigned char tmr_init (
    unsigned char devNum,
    tmr_context_t* context
);
```

Description

Initialise the timer driver for the timer specified in the context and registers the driver with the Device Manager.

Parameters

devNum
The device number to use when registering the driver with the Device Manager is passed in the devNum parameter.

context
The second parameter, context, is used to specify the timer that is being registered with the device manager. If the pointer is NULL then the default timer TIMER_0 is used.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The context parameter must be of the form of the structure defined below:

```
typedef struct _tmr_context_t {
    unsigned char timer_identifier;
} tmr_context_t;
```

Valid values for the timer_identifier member are TIMER_0, TIMER_1 or TIMER_2.

4.2.1.5.4 Timer Example

```
// *****
// Timer example
```

```
// This application uses timer 0 with GPIO port A to output a count
// on LEDs. The LEDs display a count of the timer interrupts that
// have occurred since the timer was started.
// *****

#include "vos.h"

#include "Timers.h"
#include "GPIO.h"

#define SIZEOF_tcb                0x400
#define NUMBER_OF_DEVICES        2
/* Device definitions*/
#define TIMER0                    0
#define GPIOA                    1

// *****
// Device initialisation
// *****
void init_devices(VOS_HANDLE* hGpioA, VOS_HANDLE* hTimer0) {

    gpio_iocctl_cb gpio_iocb;
    gpio_context_t gpioCtxA;
    tmr_iocctl_cb_t tmr_iocb;
    tmr_context_t tmrCtx0;
    unsigned char packageType;

    if (NUMBER_OF_DEVICES != 0) {

        // Add device initialisation calls here

        //*****
        // INITIALISE IOMUX
        //*****
        // route GPIO port A signals as required

        packageType = vos_get_package_type();

        if (packageType == VINCULUM_II_48_PIN){
            // GPIO port A bit 0 to pin 45
            vos_iomux_define_output(45,IOMUX_OUT_GPIO_PORT_A_0);
            // GPIO port A bit 1 to pin 46
            vos_iomux_define_output(46,IOMUX_OUT_GPIO_PORT_A_1);
            // GPIO port A bit 2 to pin 47
            vos_iomux_define_output(47,IOMUX_OUT_GPIO_PORT_A_2);
            // GPIO port A bit 3 to pin 48
            vos_iomux_define_output(48,IOMUX_OUT_GPIO_PORT_A_3);
            // GPIO port A bit 7 to pin 44
            vos_iomux_define_output(44,IOMUX_OUT_GPIO_PORT_A_7);
            // GPIO port A bit 6 to pin 43
            vos_iomux_define_output(43,IOMUX_OUT_GPIO_PORT_A_6);
            // GPIO port A bit 5 to pin 42
            vos_iomux_define_output(42,IOMUX_OUT_GPIO_PORT_A_5);
            // GPIO port A bit 4 to pin 41
            vos_iomux_define_output(41,IOMUX_OUT_GPIO_PORT_A_4);
        }
        else if (packageType == VINCULUM_II_64_PIN) {
            // GPIO port A bit 0 to pin 61
            vos_iomux_define_output(61,IOMUX_OUT_GPIO_PORT_A_0);
            // GPIO port A bit 1 to pin 62
            vos_iomux_define_output(62,IOMUX_OUT_GPIO_PORT_A_1);
            // GPIO port A bit 2 to pin 63
            vos_iomux_define_output(63,IOMUX_OUT_GPIO_PORT_A_2);
            // GPIO port A bit 3 to pin 64
            vos_iomux_define_output(64,IOMUX_OUT_GPIO_PORT_A_3);
            // GPIO port A bit 7 to pin 60
```

```
        vos_iomux_define_output(60,IOMUX_OUT_GPIO_PORT_A_7);
        // GPIO port A bit 6 to pin 59
        vos_iomux_define_output(59,IOMUX_OUT_GPIO_PORT_A_6);
        // GPIO port A bit 5 to pin 58
        vos_iomux_define_output(58,IOMUX_OUT_GPIO_PORT_A_5);
        // GPIO port A bit 4 to pin 57
        vos_iomux_define_output(57,IOMUX_OUT_GPIO_PORT_A_4);
    }

//*****
// INITIALISE GPIO PARAMETERS
//*****
// Specify the GPIO port that we wish to open, defined within the GPIO header file.
gpioCtxA.port_identifier = GPIO_PORT_A;
// Initializes our device with the device manager.
gpio_init(GPIOA, &gpioCtxA);
// Open the GPIO and get a handle
*hGpioA = vos_dev_open(GPIOA);

// Set all pins to output using an ioctl.
gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_MASK;
gpio_iocb.value = 0xFF;
// Send the ioctl to the device manager.
vos_dev_ioctl(*hGpioA, &gpio_iocb);

//*****
// INITIALISE Timer PARAMETERS
//*****
// Specify the timer that we wish to open, defined within the timer header file.
tmrCtx0.timer_identifier = TIMER_0;

// Initializes our device with the device manager.
tmr_init(TIMER_0, &tmrCtx0);

// Initializes our device with the device manager.
// Open the timer and get a handle
*hTimer0 = vos_dev_open(TIMER_0);

// set a tick size of 1ms
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_SET_TICK_SIZE;
tmr_iocb.param = TIMER_TICK_MS;
vos_dev_ioctl(*hTimer0, &tmr_iocb);

// set timer to count down from 5 seconds
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_SET_COUNT;
tmr_iocb.param = 5000; // 5s
vos_dev_ioctl(*hTimer0, &tmr_iocb);

tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_SET_DIRECTION;
tmr_iocb.param = TIMER_COUNT_DOWN;
vos_dev_ioctl(*hTimer0, &tmr_iocb);

// set continuous mode
tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_SET_MODE;
tmr_iocb.param = TIMER_MODE_CONTINUOUS;
vos_dev_ioctl(*hTimer0, &tmr_iocb);
}
}

// *****
// tick thread
```

```
// *****
void tick(VOS_HANDLE hGpio, VOS_HANDLE hTimer) {

    tmr_ioctl_cb_t tmr_iocb;

    unsigned char portData = 0;

    tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_START;
    vos_dev_ioctl(hTimer, &tmr_iocb);

    while (1) {

        // block until the timer cycle has completed
        tmr_iocb.ioctl_code = VOS_IOCTL_TIMER_WAIT_ON_COMPLETE;
        vos_dev_ioctl(hTimer, &tmr_iocb);

        portData++;

        // Writes data to the GPIO port.
        vos_dev_write(hGpio, &portData, 1, NULL);

    }

}

// *****
// Main application
// *****
void main(void) {

    VOS_HANDLE hGpioA = NULL;
    VOS_HANDLE hTimer0 = NULL;

    // initialise RTOS
    vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);

    // Sets the CPU frequency of the connected device.
    vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);

    // initialise devices (APPLICATION SPECIFIC)
    init_devices(&hGpioA, &hTimer0);

    // initialise threads
    // tick thread
    vos_create_thread(
        31,
        SIZEOF_tcb,
        &tick,
        2*sizeof(VOS_HANDLE),
        (VOS_HANDLE)hGpioA,
        (VOS_HANDLE)hTimer0
    );

    // Start Scheduler kicks off the created threads.
    vos_start_scheduler();

main_loop:
    goto main_loop;

}
```

4.2.1.6 PWM Driver

VNC2 contains 8 pulse width modulators (PWMs) which can be used to control external devices. The PWM block consists of 8 comparators and 8 PWM outputs. Each PWM output can be individually set to toggle its output state based on any combination of the comparators. The comparators compare

a specified value against a counter and when they are equal this will toggle the value of all the PWM outputs linked to that comparator.

The number of cycles can be set from 1 to 255, or a continuous mode can be set to loop indefinitely. There is an option to preserve the state from the end of the last cycle or reset to the initial state of the PWM before starting the next cycle.

The PWM driver does not support read and write operations.

The [pwm_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

Driver Hierarchy

PWM Driver hierarchy:

PWM Driver
VOS Kernel
PWM Hardware

Library Files

PWM.a

Header Files

PWM.h

4.2.1.6.1 PWM Return Codes

Status Codes

All calls to the PWM driver will return one of the following status codes.

PWM_OK	The command completed successfully.
PWM_INVALID_IOCTL_CODE	The IOCTL code is invalid for this driver.
PWM_INVALID_COMPARATOR_NUMBER	The specified comparator is outwith the valid range.
PWM_INVALID_PWM_NUMBER	The specified PWM output is outwith the valid range.
PWM_INTERRUPT_NOT_ENABLED	The requested operation cannot be performed because the interrupt is not enabled.
PWM_IN_CONTINUOUS_MODE	The requested operation cannot be performed because the PWM is in continuous mode.
PWM_ERROR	An unspecified error occurred.

4.2.1.6.2 PWM IOCTL Calls

Calls to the PWM IOCTL interface use the following structure:

```
typedef struct _pwm_ioctl_cb_t {
    unsigned char ioctl_code;
    union {
        unsigned char pwm_number;
        unsigned char comparator_number;
    } identifier;
    union {
        unsigned char prescaler;
        unsigned short value;
    } count;
}
```

```
union {
    unsigned short value;
} comparator;
union {
    unsigned char enable_mask;
    unsigned char mode;
    unsigned char init_state_mask;
    unsigned char trigger_mode;
    unsigned char restore_state_mask;
} output;
} pwm_ioctl_cb_t;
```

The following IOCTL request codes are supported by the PWM driver.

<u>VOS_IOCTL_PWM_RESET</u>	Reset all PWMs
<u>VOS_IOCTL_PWM_ENABLE_OUTPUT</u>	Enable PWM output
<u>VOS_IOCTL_PWM_DISABLE_OUTPUT</u>	Disable PWM output
<u>VOS_IOCTL_PWM_SET_TRIGGER_MODE</u>	Set trigger mode for all PWMs
<u>VOS_IOCTL_PWM_ENABLE_INTERRUPT</u>	Enable interrupt for all PWMs
<u>VOS_IOCTL_PWM_DISABLE_INTERRUPT</u>	Disable interrupt for all PWMs
<u>VOS_IOCTL_PWM_SET_PRESCALER_VALUE</u>	Set prescaler value for all PWMs
<u>VOS_IOCTL_PWM_SET_COUNTER_VALUE</u>	Maximum counter value, used for end of cycle
<u>VOS_IOCTL_PWM_SET_COMPARATOR_VALUE</u>	Set a comparator value
<u>VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLED</u>	Set which comparators the PWM output should toggle on
<u>VOS_IOCTL_PWM_SET_INITIAL_STATE</u>	Set the initial state of the PWMs hi (1) or low (0)
<u>VOS_IOCTL_PWM_RESTORE_INITIAL_STATE</u>	Restores the initial state of the PWM on completion of cycle
<u>VOS_IOCTL_PWM_SET_NUMBER_OF_CYCLES</u>	Set the number of times the PWM should repeat the output
<u>VOS_IOCTL_PWM_WAIT_ON_COMPLETE</u>	Wait on the specified number of PWM cycles completing

4.2.1.6.2.1 VOS_IOCTL_PWM_RESET

Description

Reset the PWM block.

Parameters

None.

Returns

This IOCTL will always return PWM_OK.

Example

```
/* reset the PWM */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_RESET;
vos_dev_ioctl(hPwm,&pwm_iocb);
```

4.2.1.6.2.2 VOS_IOCTL_PWM_ENABLE_OUTPUT

Description

Enable the PWM outputs.

Parameters

None.

Returns

This IOCTL will always return PWM_OK.

Example

```
/* enable the PWM outputs */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_ENABLE_OUTPUT;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.3 VOS_IOCTL_PWM_DISABLE_OUTPUT

Description

Disable the PWM outputs.

Parameters

None.

Returns

This IOCTL will always return PWM_OK.

Example

```
/* disable the PWM outputs */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_DISABLE_OUTPUT;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.4 VOS_IOCTL_PWM_SET_TRIGGER_MODE

Description

Specify the optional trigger mode for the PWM to use for enabling output. The trigger is provided from any GPIO interrupt.

Parameters

The trigger type to use. Valid values are:

```
PWM_TRIGGER_MODE_DISABLED
PWM_TRIGGER_MODE_POSITIVE_EDGE
PWM_TRIGGER_MODE_NEGATIVE_EDGE
PWM_TRIGGER_MODE_ANY_EDGE
```

Returns

This IOCTL will always return PWM_OK.

Example

```
/* set the PWM trigger mode for negative edge */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_TRIGGER_MODE;
pwm_iocb.output.trigger_mode = PWM_TRIGGER_MODE_NEGATIVE_EDGE;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.5 VOS_IOCTL_PWM_ENABLE_INTERRUPT

Description

Enables an interrupt when the specified PWM operation has completed. If enabled, this allows the PWM driver to signal an application that the PWM operation has completed via the [VOS_IOCTL_PWM_WAIT_ON_COMPLETE](#) IOCTL call.

Parameters

None.

Returns

This IOCTL will always return PWM_OK.

Example

```
/* enable the PWM interrupt */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_ENABLE_INTERRUPT;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.6 VOS_IOCTL_PWM_DISABLE_INTERRUPT

Description

Disables an interrupt when the specified PWM operation has completed.

Parameters

None.

Returns

This IOCTL will always return PWM_OK.

Example

```
/* disable the PWM interrupt */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_DISABLE_INTERRUPT;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.7 VOS_IOCTL_PWM_SET_PRESCALER_VALUE

Description

Set the prescaler value for the PWM counter.

Parameters

The prescaler value to be used is passed in the prescaler member of count.

Returns

This IOCTL will always return PWM_OK.

Example

```
/* set the PWM counter prescaler to 0x70 */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_PRESCALER_VALUE;
pwm_iocb.count.prescaler = 0x70;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.8 VOS_IOCTL_PWM_SET_COUNTER_VALUE

Description

Set the value for the PWM counter. This specifies the value at which the counter will generate a PWM interrupt and restart at.

Parameters

The count value to be used is passed in the value member of count.

Returns

This IOCTL will always return PWM_OK.

Example

```
/* set the PWM counter value to 0x0120 */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COUNTER_VALUE;
pwm_iocb.count.value = 0x0120;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.9 VOS_IOCTL_PWM_SET_COMPARATOR_VALUE

Description

Set the value for a specified PWM comparator. This value is compared with the internal PWM counter value and when equal, will cause PWM outputs which are linked to the comparator to toggle their output.

Parameters

The comparator value to be used is passed in the value member of comparator. The comparator to have its value set should be specified in the comparator_number member of identifier.

Returns

If the comparator_number member of identifier is outwith the permitted range, PWM_INVALID_COMPARATOR_NUMBER will be returned.

Example

```
/* set the PWM comparator 0 value to 0x12 */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
pwm_iocb.identifier.comparator_number = COMPARATOR_0;
pwm_iocb.comparator.value = 0x012;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.10 VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES

Description

Specify which comparators a PWM output should toggle on.

Parameters

A bit-mask of the comparators to be used to toggle the output is passed in the enable_mask member of output. The PWM output to have its output enable mask set should be specified in the pwm_number member of identifier.

Returns

If the pwm_number member of identifier is outwith the permitted range, PWM_INVALID_PWM_NUMBER will be returned.

Example

```
/* set PWM 2 to toggle its output when comparator 0 or comparator 1
values are equal to the internal counter value */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES;
pwm_iocb.identifier.pwm_number = PWM_2;
pwm_iocb.output.enable_mask = (MASK_COMPARATOR_0 | MASK_COMPARATOR_1);
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.11 VOS_IOCTL_PWM_SET_INITIAL_STATE

Description

Control the initial state of the PWM outputs. The initial state can be high (1) or low (0) for each PWM.

Parameters

A bit-mask of the initial state of each PWM output is passed in the `init_state_mask` member of output.

Returns

This IOCTL will always return PWM_OK.

Example

```
/* set PWM 1 initial state to high, all others to low */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_INITIAL_STATE;
pwm_iocb.output.init_state_mask = 0x02;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.12 VOS_IOCTL_PWM_RESTORE_INITIAL_STATE

Description

Control the if a PWM output returns to its initial state when the internal counter expires. A bit value of 1 will restore the PWM output initial state, a bit value of 0 will maintain the state from when the counter last expired.

Parameters

A bit-mask of the initial state of each PWM output is passed in the `init_state_mask` member of output.

Returns

This IOCTL will always return PWM_OK.

Example

```
/* reset PWM 1 and PWM 2 to their initial states when the internal counter expires.
all other PWM outputs will continue from the state they were in when the counter
expired */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_RESTORE_INITIAL_STATE;
pwm_iocb.output.restore_state_mask = (MASK_PWM_1 | MASK_PWM_2);
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.13 VOS_IOCTL_PWM_SET_NUMBER_OF_CYCLES

Description

Set the number of times that the PWM internal counter should run. Each time, the counter will start at 0 and count up to the value specified by [VOS_IOCTL_PWM_SET_COUNTER_VALUE](#).

Parameters

The number of cycles to be used is passed in the `mode` member of output. A value of 0 specifies continuous output.

Returns

This IOCTL will always return PWM_OK.

Example

```
/* set the number of PWM cycles to 8. */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_NUMBER_OF_CYCLES;
pwm_iocb.output.mode = 0x08;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.2.14 VOS_IOCTL_PWM_WAIT_ON_COMPLETE

Description

Block thread execution until the current PWM operation is complete.

Parameters

None.

Returns

If the PWM does not have its interrupt enabled PWM_INTERRUPT_NOT_ENABLED is returned. If the PWM is in continuous mode PWM_IN_CONTINUOUS_MODE is returned.

Example

```
/* wait for PWM complete interrupt to fire */
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_WAIT_ON_COMPLETE;
vos_dev_ioctl(hPwm, &pwm_iocb);
```

4.2.1.6.3 pwm_init()

Syntax

```
unsigned char pwm_init (
    unsigned char devNum
);
```

Description

Initialise the PWM driver and registers the driver with the Device Manager.

Parameters

The device number to use when registering the driver with the Device Manager is passed in the devNum parameter.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

None.

4.2.1.6.4 PWM Example

```
/* *****
// PWM example
// This application uses a single PWM output linked to 2 PWM comparators
// to generate a 50% duty cycle pulse 25 times then sleep for 10 seconds
// repeatedly.
// *****

#include "vos.h"
#include "PWM.h"
```

```
#define SIZEOF_tcb                0x400
#define NUMBER_OF_DEVICES        1
/* Device definitions*/
#define VOS_DEV_PWM                0

// *****
// Device initialisation
// *****
void init_devices(void) {

    unsigned char packageType;

    if (NUMBER_OF_DEVICES != 0) {

        //*****
        // INITIALISE IOMUX PARAMETERS
        //*****

        // route PWM signals as required
        packageType = vos_get_package_type();

        if (packageType == VINCULUM_II_48_PIN){
            // PWM 0 to pin 45
            vos_iomux_define_output(45,IOMUX_OUT_PWM_0); //PWM0
        }
        else if (packageType == VINCULUM_II_64_PIN) {
            // PWM 0 to pin 61
            vos_iomux_define_output(61,IOMUX_OUT_PWM_0); //PWM0
        }

        //*****
        // INITIALISE PWM PARAMETERS
        //*****
        pwm_init(VOS_DEV_PWM);

    }

}

// *****
// Pulse thread
// *****
void pulse() {

    VOS_HANDLE hPwm;
    pwm_ioctl_cb_t pwm_iocb;

    // open pwm and get a handle
    hPwm = vos_dev_open(VOS_DEV_PWM);

    // set counter prescaler value
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_PRESCALER_VALUE;
    pwm_iocb.count.prescaler = 0x0A;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // *****
    // Setting a count value of 0x00A0 with toggles at 0x0010 and 0x0060
    // will give a 50% duty cycle
    // *****

    // set counter value - cycle complete when internal counter reaches this value
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COUNTER_VALUE;
    pwm_iocb.count.value = 0x00A0;
    vos_dev_ioctl(hPwm, &pwm_iocb);
```

```
// set comparator 0 value - toggle output at a value of 0x0010
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
pwm_iocb.identifier.comparator_number = COMPARATOR_0;
pwm_iocb.comparator.value = 0x0010;
vos_dev_ioctl(hPwm, &pwm_iocb);

// set comparator 1 value - toggle output at a value of 0x0060
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
pwm_iocb.identifier.comparator_number = COMPARATOR_1;
pwm_iocb.comparator.value = 0x0080;
vos_dev_ioctl(hPwm, &pwm_iocb);

// enable comparators 0 and 1 for PWM 0
// this will cause PWM output 1 to toggle on comparators 0 and 1
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES;
pwm_iocb.identifier.pwm_number = PWM_0;
pwm_iocb.output.enable_mask = (MASK_COMPARATOR_0 | MASK_COMPARATOR_1);
vos_dev_ioctl(hPwm, &pwm_iocb);

// set initial state - all PWM outputs will be low (0) initially
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_INITIAL_STATE;
pwm_iocb.output.init_state_mask = 0x00;
vos_dev_ioctl(hPwm, &pwm_iocb);

// set restore state - PWM output 0 will return to low state (0)
// at end of cycle
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_RESTORE_INITIAL_STATE;
pwm_iocb.output.restore_state_mask = (MASK_PWM_0);
vos_dev_ioctl(hPwm, &pwm_iocb);

// set mode to 25 cycles
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_NUMBER_OF_CYCLES;
pwm_iocb.output.mode = 0x19;
vos_dev_ioctl(hPwm, &pwm_iocb);

while(1) {
    // enable interrupt - this will fire when the specified number of
    // cycles is complete
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_ENABLE_INTERRUPT;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // enable output
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_ENABLE_OUTPUT;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // wait on interrupt
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_WAIT_ON_COMPLETE;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // When we get to here, we've completed our 25 cycles of 50% duty cycle

    // disable output
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_DISABLE_OUTPUT;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // sleep for 10 seconds
    vos_delay_msecs(10000);
}

}

// *****
// Main application
// *****
void main(void) {
```

```
// initialise rtos
vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);

vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);

// initialise devices (APPLICATION SPECIFIC)
init_devices();

// initialise threads
// pulse thread
vos_create_thread(
    31,
    sizeof_tcb,
    &pulse,
    0
);

// enable PWM interrupts
vos_enable_interrupts(VOS_PWM_TOP_INT_IEN);

vos_start_scheduler();

main_loop:
    goto main_loop;
}
```

4.2.2 Layered Drivers

There is no restriction placed on drivers calling other drivers. Furthermore, drivers need not have any ties to physical hardware devices. These driver are called Layered Drivers because they normally reside logically between an application and a physical or hardware device driver.

On the VNC2 device, the hardware drivers for the USB Host controller, USB Slave controller and SPI Master have a selection of layered drivers.

4.2.2.1 Mass Storage Interface

The Mass Storage Interface provides a common interface from file system drivers and APIs to the underlying hardware. This may be [BOMS Class Disk](#) or other devices. All share the same driver interface.

4.2.2.1.1 Mass Storage Interface Return Codes

All calls to the MSI compliant drivers will return one of the following MSI (Mass Storage Interface) status codes.

MSI_OK
The command completed successfully.

MSI_COMMAND_FAILED
The command to the driver failed.

MSI_NOT_FOUND
The interface supplied to an attach IOCTL could not be found or did not have suitable endpoints. It may also occur if a device was removed unexpectedly.

MSI_INVALID_PARAMETER
A parameter in an IOCTL request outwith the valid range. Or no semaphore was provided in the transfer structure.

MSI_INVALID_BUFFER
The transfer structure passed to a read or write operation was incorrect.

MSI_NOT_ACCESSED
This is used to ensure that the driver has actually modified the return value of the status in the transfer structure.

MSI_ERROR
The device to attach to is invalid.

4.2.2.1.2 Mass Storage Interface IOCTL Calls

Calls to MSI driver IOCTL method take the form:

```
typedef struct _msi_ioctl_cb_t {  
    unsigned char ioctl_code;  
    // read buffer  
    unsigned char *get;  
    // write buffer  
    unsigned char *set;  
} msi_ioctl_cb_t;
```

The following MSI IOCTL request codes are supported by all devices.

MSI_IOCTL_RESET	Resets the device
MSI_IOCTL_GET_MAX_LUN	Gets the maximum LUN
MSI_IOCTL_GET_LUN	Returns the current LUN selected
MSI_IOCTL_SET_LUN	Sets the current LUN
MSI_IOCTL_GET_SECTOR_SIZE	Returns the sector size in bytes
MSI_IOCTL_GET_DEV_INFO	Gets a structure containing driver dependent device information

4.2.2.1.3 Mass Storage Interface Read and Write Operations

Syntax

```
typedef struct _msi_xfer_cb_t {  
    // sector number  
    unsigned long sector;  
    // reference for report completion notification  
    vos_semaphore_t *s;  
    // buffer pointer  
    unsigned char *buf;  
    // buffer length  
    unsigned short buf_len;  
    // transaction total length (not the buffer size when transactions are split)  
    unsigned short total_len;  
    // which command/data/status phases to use  
    unsigned char do_phases;  
  
    // transfer completion status  
    unsigned char status;  
  
    // storage for transport specific transfer structures  
    // transfer structure for USB Host (BOMS specific)  
    union  
    {  
        // transfer structure for USB Host (BOMS specific)  
        usbhost_xfer_t usb;  
    } transport;  
} msi_xfer_cb_t;
```

Description

To read and write to the driver a transfer block is used. This is a structure that is sent to [vos_dev_read\(\)](#) and [vos_dev_write\(\)](#) to describe to the driver how to transfer data to device.

It specifies the sector on the disk, the buffer and length of the buffer, the total size of the transfer and the phases of the operation to perform. Both the size of the buffer and the total size of the transfer are required since a disk operation can be split into several transactions consisting of the Command, one or more Data phases and a Status phase.

All disk operations start with a Command phase, have one or more data phases and must terminate with a Status phase. All data phases must be sized as a multiple of the sector size - this is a dependent on the driver. Multiple data phases can be performed in separate transactions with buffer

sizes smaller than the total size of the transfer.

To perform a read or write operation, the application must fill in the data in the transfer block and then send a pointer to the transfer block to the driver using [vos_dev_read\(\)](#) or [vos_dev_write\(\)](#). The driver returns the status of the transaction in the same structure.

Parameters

`sector`

The sector number to start the operation on the disk. This is also known as the LBA.

`s`

A semaphore pointer can be specified which is supplied by the application to allow either the read or write operation to block until completion.

`buf`

A buffer of multiple of the sector size which is used as the target or source data for the operation.

`buf_len`

The actual length of the buffer in `buf`.

`total_len`

This is used in the Command phase to tell the disk the size of the transfer which will occur. The correct total number of bytes must be sent or received by Data phase operations before a Status phase is performed.

`do_phases`

This bit map instructs the driver to do one or more of the Command Status and Data phases. It can also instruct the transport (USB Host driver) to delay starting a USB transaction or to not block on completion.

`MSI_PHASE_COMMAND_BIT`

`MSI_PHASE_DATA_BIT`

`MSI_PHASE_STATUS_BIT`

`status`

The status of the operation is returned in this member. It will contain a transport (i.e. USB Host driver) error in the lower nibble if the bit `BOMS_HC_CC_ERROR` is set.

`transport`

A storage area for the BOMS driver to build a transfer descriptor for the USB Host controller.

Remarks

Reading and writing sectors directly without the use of a File System driver or application is not recommended.

Example

See the example in [BOMS Read and Write Example](#)

4.2.2.2 USB Host Class Drivers

The USB Host controller is designed to be flexible in order for layered drivers to add the functionality required by devices adhering to both standard classes and user defined classes.

The layered drivers for several standard classes and FTDI's own FT232 style of device are provided.

4.2.2.2.1 BOMS Class Driver

The BOMS class driver provides an interface between a file system and a USB disk. The disk is typically a Flash disk but may also be a hard disk drive.

The interface is described in the document entitled "USB Mass Storage Class Bulk Only Transport" available from the USB Implementers Forum website: <http://usbif.com/>

This class is used by the FAT File System class to access disks. It requires the USBHost class hardware driver.

BOMS is a [Mass Storage Interface](#) (MSI) driver and uses the "msi.h" header file for commonality of [IOCTL Calls](#) and [Return Codes](#) with other similar drivers.

The [boms_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started

with [vos_start_scheduler\(\)](#).

The BOMS Class Driver driver needs to be attached to a suitable USB device to function using the [BOMS MSI_IOCTL_BOMS_ATTACH](#) IOCTL call specifying an interface on the [USB Host Driver](#).

Driver Hierarchy

BOMS Driver hierarchy:

BOMS Driver
USB Host Driver
VOS Kernel
USB Host Hardware

Library Files

USBHost.a

BOMS.a

Header Files

USBHost.h

BOMS.h

MSI.h

4.2.2.2.1.1 BOMS Concepts

The BOMS driver needs to be attached to a suitable USB device to function. A USB device must have USB Class, Subclass and Protocol values for a BOMS class device.

Reading and writing to the BOMS class driver is done through transfer blocks using the read and write methods of the driver. The transfer block structure tells the driver what information to read and how to read it. This provides flexibility in structuring requests to the hardware, allowing reading and writing operations that do not block until completion and streaming operations where the application can receive small amounts of data from a larger request.

BOMS devices have the following properties:

LUN - a Logical Unit Number, essentially a way of having multiple logical devices on one BOMS device. Normally, there will be only one LUN but several may be present. On operations systems these can be shown as multiple drives.

LBA - Logical Block Address, on a disk this is associated with the unique address for a sector.

Sector - the smallest addressable block of storage. All operations must occur on a complete sector.

Cluster - a grouping of 1, 2, 4, 8, 16 or 32 sectors. This can be mapped to the physical storage characteristics of the device whereas a sector can be a small section of a storage unit on a device.

Do not attempt to write directly to a disk using the BOMS driver unless you know what you are doing.

4.2.2.2.1.2 BOMS Return Codes

Return codes are described in [Mass Storage Interface Return Codes](#).

4.2.2.2.1.3 BOMS Read and Write Operations

Refer to the [Mass Storage Interface Read and Write Operations](#) topic for read and write transactions on the BOMS driver.

The sector size, and hence the buffer multiple size, for the BOMS driver is 512 bytes.

4.2.2.2.1.4 BOMS IOCTL Calls

Calls to the BOMS driver's IOCTL method take the form:

```
typedef struct _msi_ioctl_cb_t {
    unsigned char ioctl_code;
    // read buffer
    unsigned char *get;
    // write buffer
    unsigned char *set;
} msi_ioctl_cb_t;
```

The following MSI IOCTL request codes are supported by the BOMS driver.

<u>MSI_IOCTL_RESET</u>	Resets the BOMS device
<u>MSI_IOCTL_GET_MAX_LUN</u>	Gets the maximum LUN
<u>MSI_IOCTL_GET_LUN</u>	Returns the current LUN selected
<u>MSI_IOCTL_SET_LUN</u>	Sets the current LUN
<u>MSI_IOCTL_GET_SECTOR_SIZE</u>	Returns the sector size in bytes
<u>MSI_IOCTL_GET_DEV_INFO</u>	Gets a structure containing device information

The BOMS driver also supports the following transport specific IOCTL requests.

<u>MSI_IOCTL_BOMS_ATTACH</u>	Attaches the BOMS driver to a USB interface device
<u>MSI_IOCTL_BOMS_DETACH</u>	Detaches the BOMS driver

Description

Attaches a BOMS driver to a USB device. The device can be found using the USBHost driver's VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS IOCTL method.

Parameters

The device interface handle and USB Host controller handle must be passed in a structure using the set member of the [msi_ioctl_cb_t](#) IOCTL structure.

```
typedef struct _boms_ioctl_cb_attach_t
{
    VOS_HANDLE hc_handle;
    usbhost_device_handle *ifDev;
} boms_ioctl_cb_attach_t;
```

Returns

There is no data returned by this call although the return value indicates success or otherwise of the attach.

MSI_ERROR if the device is not a valid BOMS device - the USB Class, Subclass and Protocol are checked
MSI_NOT_FOUND if a control, BULK IN or BULK OUT endpoint cannot be found for the device or the Get Max LUN SETUP command failed

Example

```
void BOMSFindDevice()
{
    VOS_HANDLE hUsb2, hBoms;
    usbhost_device_handle *ifDev2;
    usbhost_ioctl_cb_t hc_ioctb;
    usbhost_ioctl_cb_class hc_ioctb_class;
    fat_context fatContext;

    msi_ioctl_cb_t boms_ioctb;
    boms_ioctl_cb_attach_t boms_att;
```

```
// find BOMS class device
hc_iocb_class.dev_class = USB_CLASS_MASS_STORAGE;
hc_iocb_class.dev_subclass = USB_SUBCLASS_MASS_STORAGE_SCSI;
hc_iocb_class.dev_protocol = USB_PROTOCOL_MASS_STORAGE_BOMS;

hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
hc_iocb.handle.dif = NULL;
hc_iocb.set = &hc_iocb_class;
hc_iocb.get = &ifDev2;

if (vos_dev_ioctl(hUsb2, &hc_iocb) != USBHOST_OK)
{
    // no BOMS class found
}

// now we have a device, initialise a BOMS driver for it
hBoms = vos_dev_open(VOS_DEV_BOMS);

// boms_attach
boms_att.hc_handle = hUsb2;
boms_att.ifDev = ifDev2;

boms_iocb.ioctl_code = MSI_IOCTL_BOMS_ATTACH;
boms_iocb.set = &boms_att;
boms_iocb.get = NULL;

if (vos_dev_ioctl(hBoms, &boms_iocb) != MSI_OK)
{
    // could not attach to device
}

// device has been found and opened

// now detach from the device
boms_iocb.ioctl_code = MSI_IOCTL_BOMS_DETACH;

vos_dev_ioctl(hBoms, &boms_iocb)
}
```

Description

Detaches a BOMS driver from a USB device.

Parameters

There are no parameters passed to this IOCTL.

Returns

There is no data returned by this call.

Example

See example in BOMS [BOMS MSI_IOCTL_BOMS_ATTACH](#).

Description

Resets the BOMS device by sending a BOMS Request Reset packet to the device.

Parameters

There are no parameters for this call.

Returns

The return code indicates the status of the USB request sent to the device.

Example

```
boms_iocb.ioctl_code = MSI_IOCTL_RESET;  
vos_dev_ioctl(hBoms,&boms_iocb);
```

Description

Returns the maximum LUN available on the device.

Parameters

The function takes no input parameters.

Returns

The maximum LUN is returned into an unsigned char variable pointed to by the `get` member of the [msi_ioctl_cb_t](#) IOCTL structure.

Example

```
unsigned char max_lun;  
  
boms_iocb.ioctl_code = MSI_IOCTL_GET_MAX_LUN;  
boms_iocb.get = &max_lun;  
vos_dev_ioctl(hBoms,&boms_iocb);  
  
if (max_lun > 1)  
{  
    // option for multiple LUN disks  
}
```

Description

Gets the current LUN selected for the device.

Parameters

The function takes no input parameters.

Returns

The current LUN is returned into an unsigned char variable pointed to by the `get` member of the [msi_ioctl_cb_t](#) IOCTL structure.

Example

```
unsigned char this_lun;  
  
boms_iocb.ioctl_code = MSI_IOCTL_GET_MAX_LUN;  
boms_iocb.get = &this_lun;  
vos_dev_ioctl(hBoms,&boms_iocb);  
  
if (this_lun != 0)  
{  
    // option for multiple LUN disks  
}
```

Description

Sets the current LUN selected for the device.

Parameters

The current LUN is set using a pointer to an unsigned char variable in the `set` member of the [msi_ioctl_cb_t](#) IOCTL structure.

Returns

The function returns no output parameters.

Example

```
unsigned char this_lun;

boms_iocb.ioctl_code = MSI_IOCTL_GET_MAX_LUN;
boms_iocb.get = &this_lun;
vos_dev_ioctl(hBoms, &boms_iocb);

if (this_lun != 0)
{
    this_lun = 0;
    boms_iocb.ioctl_code = MSI_IOCTL_SET_MAX_LUN;
    boms_iocb.set = &this_lun;
    vos_dev_ioctl(hBoms, &boms_iocb);
}
```

Description

Gets the sector size for the device.

Parameters

The function takes no input parameters.

Returns

The sector size returned into an unsigned short variable pointed to by the `get` member of the [msi_ioctl_cb_t](#) IOCTL structure. This can only be 512 or 2048 bytes.

Example

```
unsigned short size;

boms_iocb.ioctl_code = MSI_IOCTL_GET_MAX_LUN;
boms_iocb.get = &size;
vos_dev_ioctl(hBoms, &boms_iocb);

if (size != 512)
{
    // sector size is not supported
}
```

Description

Gets device information and populates an application supplied structure. The device information, although it contains strings, is always of a fixed size.

Parameters

There are no parameters for this call.

Returns

The device information is copied into the structure pointed to by the `get` member of the [msi_ioctl_cb_t](#) IOCTL structure.

```
typedef struct _msi_ioctl_cb_info_t
{
    // device information
```

```
unsigned char vendorId[8];
unsigned char productId[16];
unsigned char rev[4];
unsigned short vid; // BOMS specific
unsigned short pid; // BOMS specific
} msi_ioctl_cb_info_t;
```

The VID and PID members are BOMS specific. Vendor ID, product ID and rev are fixed length strings padded with space characters.

Example

```
void checkDisk(VOS_HANDLE hDisk)
{
    msi_ioctl_cb_t disk_iocb;
    msi_ioctl_cb_info_t disk_iocb_info;

    disk_iocb.ioctl_code = MSI_IOCTL_GET_DEV_INFO;
    disk_iocb.get = &disk_iocb_info;
    vos_dev_ioctl(hDisk,&disk_iocb);

    if (disk_iocb_info.vid == 0x1234)
    {
        // specific operation for this vendor I
    }
}
```

4.2.2.2.1.5 boms_init()

Syntax

```
unsigned char boms_init (
    unsigned char devNum
);
```

Description

Initialise the BOMS driver and registers the driver with the Device Manager.

Parameters

devNum
The device number to use when registering the BOMS driver with the Device Manager.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

Memory is allocated dynamically for an instance of the BOMS driver when this call is made. It is never freed by the driver.

4.2.2.2.1.6 BOMS Examples

This example shows how to attach to BOMS device:

```
void BOMSFindDevice()
{
    VOS_HANDLE hUsb2, hBoms;
    usbhost_device_handle *ifDev2;
    usbhost_ioctl_cb_t hc_iocb;
    usbhost_ioctl_cb_class hc_iocb_class;
    fat_context fatContext;
```

```
msi_ioctl_cb_t boms_iocb;
boms_ioctl_cb_attach_t boms_att;

char buff[512];

// find BOMS class device
hc_iocb_class.dev_class = USB_CLASS_MASS_STORAGE;
hc_iocb_class.dev_subclass = USB_SUBCLASS_MASS_STORAGE_SCSI;
hc_iocb_class.dev_protocol = USB_PROTOCOL_MASS_STORAGE_BOMS;

hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
hc_iocb.handle.dif = NULL;
hc_iocb.set = &hc_iocb_class;
hc_iocb.get = &ifDev2;

if (vos_dev_ioctl(hUsb2, &hc_iocb) != USBHOST_OK)
{
    // no BOMS class found
}

// now we have a device, initialise a BOMS driver for it
hBoms = vos_dev_open(VOS_DEV_BOMS);

// boms_attach
boms_att.hc_handle = hUsb2;
boms_att.ifDev = ifDev2;

boms_iocb.ioctl_code = MSI_IOCTL_BOMS_ATTACH;
boms_iocb.set = &boms_att;
boms_iocb.get = NULL;

if (vos_dev_ioctl(hBoms, &boms_iocb) != MSI_OK)
{
    // could not attach to device
}
else
{
    // initialise FAT file system driver to use BOMS device

    // now detach from the device
    boms_iocb.ioctl_code = MSI_IOCTL_BOMS_DETACH;
    vos_dev_ioctl(hBoms, &boms_iocb)
}
}
```

The following example shows how to read or write a sector (512 bytes) to a flash disk:

```
VOS_DEVICE hBoms;

unsigned char fat_readSector(unsigned long sector, char *buffer)
{
    // transfer buffer
    msi_xfer_cb_t xfer;
    // completion semaphore
    vos_semaphore_t semRead;
    unsigned char status;

    vos_init_semaphore(&semRead, 0);

    xfer.sector = sector;
    xfer.buf = buffer;
    //TODO: 512 byte sector specific
    xfer.total_len = 512;
    xfer.buf_len = 512;
    xfer.status = MSI_NOT_ACCESSED;
    xfer.s = &semRead;
```

```

xfer.do_phases = MSI_PHASE_ALL;

status = vos_dev_read(hBoms, (unsigned char *)&xfer, sizeof(msi_xfer_cb_t ), NULL);
if (status == MSI_OK)
{
    status = FAT_OK;
}
else
{
    status |= FAT_MSI_ERROR;
}

return status;
}

unsigned char fat_writeSector(unsigned long sector, char *buffer)
{
    // transfer buffer
    msi_xfer_cb_t xfer;
    // completion semaphore
    vos_semaphore_t semRead;
    unsigned char status;

    vos_init_semaphore(&semRead, 0);

    xfer.sector = sector;
    xfer.buf = buffer;
    //TODO: 512 byte sector specific
    xfer.total_len = 512;
    xfer.buf_len = 512;
    xfer.status = MSI_NOT_ACCESSED;
    xfer.s = &semRead;
    xfer.do_phases = MSI_PHASE_ALL;

    status = vos_dev_write(hBoms, (unsigned char *)&xfer, sizeof(msi_xfer_cb_t ), NULL);
    if (status == MSI_OK)
    {
        status = FAT_OK;
    }
    else
    {
        status |= FAT_MSI_ERROR;
    }

    return status;
}

```

4.2.2.2.2 Printer Class Driver

A printer class driver will be provided which is layered on top of the USB host driver. Note that not all USB printers fall within the USB Printer device class.

Driver Hierarchy

Printer Driver hierarchy:

Printer Driver
USB Host Driver
VOS Kernel
USB Host Hardware

Library Files

USBHost.a

Printer.a

Header Files

USBHost.h

Printer.h

4.2.2.2.3 Communication Device Class Driver

A communications class driver will be provided which is layered on top of the USB host driver. Note that not all USB CSC class devices fall within the USB CDC class.

Driver Hierarchy

CDC Driver hierarchy:

CDC Driver
USB Host Driver
VOS Kernel
USB Host Hardware

Library Files

USBHost.a

CDC.a

Header Files

USBHost.h

CDC.h

4.2.2.2.4 FT232 USB Host Device Driver

The FT232 USB Host driver uses the same common IOCTL codes and read and write methods as the UART, FIFO and SPI interfaces.

The [usbHostFt232_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

The USB Host FT232 driver needs to be attached to a suitable USB device to function using the [VOS_IOCTL_USBHOSTFT232_ATTACH](#) IOCTL call specifying an interface on the [USB Host Driver](#).

Driver Hierarchy

USBHostFT232 Driver hierarchy:

USBHostFT232 Driver
USB Host Driver
VOS Kernel
USB Host Hardware

Library Files

USBHost.a

USBHostFT232.a

Header Files

USBHost.h

USBHostFT232.h

4.2.2.2.4.1 FT232 USB Host Device Return Codes

All calls to the FT232 USB Host Device driver is designed to return the same codes as the UART driver for common operations. These are listed in the [UART Return Codes](#). An additional set of error codes are also return driver specific codes.

USBHOSTFT232_OK

The command completed successfully.

USBHOSTFT232_INVALID_PARAMETER

There was an error or problem with a parameter sent to the driver.

USBHOSTFT232_DMA_NOT_ENABLED

A DMA operation was requested when DMA was not enabled.

USBHOSTFT232_ERROR

An unspecified error occurred.

USBHOSTFT232_NOT_FOUND

The FT232 device was not found in an attach operation.

USBHOSTFT232_USBHOST_ERROR

An error was reported from the USB Host interface. The USB Host code is returned in the low nibble of the return code.

4.2.2.2.4.2 FT232 USB Host Device IOCTL Calls

The FT232 USB Host Device driver accepts the same IOCTL codes as the UART driver. These are listed in the [UART IOCTL Calls](#) topic.

The structure passed to the IOCTL request is the common structure shown in the topic [Common IOCTL Calls](#).

The following additional IOCTL request codes are supported by the FT232 USB Host Device driver.

[VOS_IOCTL_USBHOSTFT232_SET_LATENCY](#)

Set the device latency timer value.

[VOS_IOCTL_USBHOSTFT232_GET_LATENCY](#)

Get the current latency timer value.

[VOS_IOCTL_USBHOSTFT232_SET_BIT_MODE](#)

Set the mode of the various pins on the device.

[VOS_IOCTL_USBHOSTFT232_GET_BIT_MODE](#)

Get current pin states.

[VOS_IOCTL_USBHOSTFT232_EEPROM_READ](#)

Read a word of data from the device EEPROM.

[VOS_IOCTL_USBHOSTFT232_EEPROM_WRITE](#)

Write a word of data to the device EEPROM.

[VOS_IOCTL_USBHOSTFT232_EEPROM_ERASE](#)

Erase the device EEPROM.

[VOS_IOCTL_USBHOSTFT232_ATTACH](#)

Attach the driver to a USB interface device.

[VOS_IOCTL_USBHOSTFT232_DETACH](#)

Detach the driver from the USB interface device.

[VOS_IOCTL_USBHOSTFT232_START_POLL](#)

Start polling for data from the device.

[VOS_IOCTL_USBHOSTFT232_STOP_POLL](#)

Stop polling for data from the device.

Description

Sets the latency timer of the device.

Parameters

An 8 bit value is taken from the param member of set.

Returns

If the parameters are incorrect then the return code is USBHOSTFT232_INVALID_PARAMETER. An error from the USB Host driver may be returned.

Example

Description

Returns the current latency timer setting of the device.

Parameters

An 8 bit value is returned in the param member of get.

Returns

If the parameters are incorrect then the return code is USBHOSTFT232_INVALID_PARAMETER. An error from the USB Host driver may be returned.

Example

Description

Sets the mode of various functions of the device. Functions and supported modes are device dependent.

Parameters

The following structure is used to set the bit mode.

```
typedef struct _usbhostft232_bitmode_t
{
    unsigned char mode;
    unsigned char mask;
} usbhostft232_bitmode_t;
```

Available bit modes are:

```
USBHOSTFT232_BIT_MODE_RESET
USBHOSTFT232_BIT_MODE_ASYNCHRONOUS_BIT_BANG
USBHOSTFT232_BIT_MODE_MPSSE
USBHOSTFT232_BIT_MODE_SYNCHRONOUS_BIT_BANG
USBHOSTFT232_BIT_MODE_MCU_HOST_BUS_EMULATION
USBHOSTFT232_BIT_MODE_FAST_SERIAL
USBHOSTFT232_BIT_MODE_CBUS_BIT_BANG
USBHOSTFT232_BIT_MODE_SYNCHRONOUS_FIFO
```

Not all devices support all bit modes.

Returns

If the parameters are incorrect then the return code is USBHOSTFT232_INVALID_PARAMETER. An error from the USB Host driver may be returned.

Example

Description

Gets the current pin states of the device. Pins and supported modes are device dependent.

Parameters

An 8bit value is returned in the param member of get. This will be a bitmap of the current pin status - 1 for high and zero for low.

Returns

If the parameters are incorrect then the return code is USBHOSTFT232_INVALID_PARAMETER. An

error from the USB Host driver may be returned.

Example

Description

Read a byte from the EEPROM of a device. The EEPROM addresses and size are device dependent.

Parameters

The following structure is passed in the data member of get.

```
typedef struct _usbhostft232_eeprom_t
{
    unsigned short ee_address;
    unsigned short ee_data;
} usbhostft232_eeprom_t;
```

The address to read from is set in ee_address member and the value returned in ee_data of the same structure.

Returns

If the parameters are incorrect then the return code is USBHOSTFT232_INVALID_PARAMETER. An error from the USB Host driver may also be returned.

Example

Description

Write a byte to the EEPROM of a device. The EEPROM addresses and size are device dependent.

Parameters

The following structure is passed in the data member of set.

```
typedef struct _usbhostft232_eeprom_t
{
    unsigned short ee_address;
    unsigned short ee_data;
} usbhostft232_eeprom_t;
```

The address to write to is set in ee_address member and the value to be written is set in ee_data of the same structure.

Returns

If the parameters are incorrect then the return code is USBHOSTFT232_INVALID_PARAMETER. An error from the USB Host driver may also be returned.

Example

Description

Erase the EEPROM of a device.

Parameters

There are no parameters to set.

Returns

An error from the USB Host driver may be returned.

Example

Description

Attaches the FT232 USBHost Device driver to an interface in the USB Host controller. The host controller handle is that obtained from [vos_dev_open\(\)](#) when the USBHost driver was opened. This function does not check the VID and PID or the class, subclass and protocol values of the device, since these are configurable.

Parameters

The device interface handle and USB Host controller handle must be passed in a structure using the set member of the IOCTL structure. The port number of the interface to address must also be specified.

```
typedef struct _usbhostft232_ioctl_cb_attach_t
{
    VOS_HANDLE hc_handle;
    usbhost_device_handle *ifDev;
    unsigned char ftPort;
} usbhostft232_ioctl_cb_attach_t;
```

Returns

If the parameters are incorrect then the return code is USBHOSTFT232_INVALID_PARAMETER. If the interface does not have control and a bulk IN and bulk OUT endpoint then USBHOSTFT232_NOT_FOUND is returned. If the attach is successful then returns USBHOSTFT232_OK.

Example

```
VOS_HANDLE hFT232, hUsbHost1;

hUsbHost1 = vos_dev_open(VOS_DEV_USBHOST1);
hFT232 = vos_dev_open(VOS_DEV_FT232);

ft232_ioctl.ioctl_code = VOS_IOCTL_USBHOSTFT232_ATTACH;
ft232_ioctl.set.data = &ft232_att;
ft232_att.hc_handle = hUsbHost1;
ft232_att.ifDev = ifDev;
ft232_att.ftPort = USBHOSTFT232_PORTA;

if (vos_dev_ioctl(hFT232, &ft232_ioctl) == USBHOSTFT232_OK)
{
    if (vos_dev_read(hFT232, buf, num_written, &num_read) != USBHOSTFT232_OK)
    {
        return;
    }
}
```

Description

This removes the association of the FT232 USB Host Device driver with the currently connected interface.

Parameters

There are no parameters to set.

Returns

Always returns USBHOSTFT232_OK.

Example

```
ft232_ioctl.ioctl_code = VOS_IOCTL_USBHOSTFT232_DETACH;
vos_dev_ioctl(hFT232, &ft232_ioctl);
```

Description

Signals the driver to start polling the attached device.

Parameters

There are no parameters to set.

Returns

Always returns USBHOSTFT232_OK.

Example

Description

Signals the driver to stop polling the attached device.

Parameters

There are no parameters to set.

Returns

Always returns USBHOSTFT232_OK.

Example

4.2.2.2.4.3 usbHostFt232_init()

Syntax

```
unsigned char usbHostFt232_init(unsigned char devNum)
```

Description

Initialise the USB Host FT232 driver for the port specified in the context and registers the driver with the Device Manager.

Parameters

The device number to use when registering the driver with the Device Manager is passed in the devNum parameter.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The driver can be attached to an FT232 device once the USB Host enumeration has completed.

4.2.2.2.4.4 FT232 USB Host Device Example

```
unsigned short num_written;  
VOS_HANDLE hFT232, hUart;  
common_ioctl_cb_t ft232_ioctl;  
usbhostft232_ioctl_cb_attach_t ft232_att;  
  
hFT232 = vos_dev_open(VOS_DEV_FT232);  
hUart = vos_dev_open(VOS_DEV_UART);
```



```

ft232_ioctl.ioctl_code = VOS_IOCTL_USBHOSTFT232_ATTACH;
ft232_ioctl.set.data = &ft232_att;
ft232_att.hc_handle = hUsbHost1;
ft232_att.ifDev = ifDev;
ft232_att.ftPort = USBHOSTFT232_PORTA;

if (vos_dev_ioctl(hFT232, &ft232_ioctl) == USBHOSTFT232_OK)
{
    // user ioctl to reset device
    ft232_ioctl.ioctl_code = VOS_IOCTL_COMMON_RESET;
    if (vos_dev_ioctl(hFT232, &ft232_ioctl) != USBHOST_OK)
    {
        return;
    }

    // baud rate is calculated as per parameter
    ft232_ioctl.ioctl_code = VOS_IOCTL_USBHOSTFT232_SET_BAUD_RATE;
    ft232_ioctl.set.uart_baud_rate = USBHOSTFT232_BAUD_9600;
    vos_dev_ioctl(hFT232, &ft232_ioctl);

    // wait for data to be received on FT232 device
    while (1)
    {
        ft232_ioctl.ioctl_code = VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS;
        vos_dev_ioctl(hFT232, &ft232_ioctl);

        num_written = ft232_ioctl.get.queue_stat;
        if (num_written > 64)
            num_written = 64;
        if (num_written) break;
    }

    // read data into buffer
    if (vos_dev_read(hFT232, buf, num_written, &num_read) != USBHOSTFT232_OK)
    {
        return;
    }
    if (num_read)
    {
        // retransmit data out on UART
        if (vos_dev_write(hUart, buf, num_read, &num_written) != UART_OK)
        {
            return;
        }
    }
}

vos_dev_close(hFT232);
vos_dev_close(hUart);
}

```

4.2.2.2.5 Still Image Class Driver

A Still Image Class driver is supplied to perform basic operations on a Still Image Class device which supports PTP (Picture Transfer Protocol) and the PIMA command set (USB Class 0x06, Subclass 0x01 and Protocol 0x01).

Driver Hierarchy

Still Image Driver hierarchy:

Still Image Driver
USB Host Driver
VOS Kernel
USB Host Hardware

Library Files

USBHost.a

StillImage.a

Header Files

USBHost.h

StillImage.h

4.2.2.2.5.1 Still Image Return Codes

All calls to the Still Image driver will return one of the following status codes.

STILLIMAGE_OK

The command completed successfully.

STILLIMAGE_NOT_FOUND

The interface supplied to an attach IOCTL could not be found or did not have suitable endpoints. It may also occur if a device was removed unexpectedly.

STILLIMAGE_READ_ONLY

The still image driver does not support writing data to a device. This will be returned for a [vos_dev_write\(\)](#) operation.

STILLIMAGE_PENDING

Not currently used.

STILLIMAGE_INVALID_PARAMETER

A parameter in an IOCTL request outwith the valid range.

STILLIMAGE_INVALID_BUFFER

Not currently used.

STILLIMAGE_INVALID_FILE_TYPE

Not currently used.

STILLIMAGE_EXISTS

Not currently used.

STILLIMAGE_NOT_OPEN

A [vos_dev_read\(\)](#) file operation was attempted when no file was opened.

STILLIMAGE_EOF

Not currently used.

STILLIMAGE_DIRECTORY_TABLE_FULL

Not currently used.

STILLIMAGE_DISK_FULL

Not currently used.

STILLIMAGE_ERROR

The device to attach to is invalid. The USB Class and Protocol do not match the supported types. A command sent to the device failed.

STILLIMAGE_HC_ERROR

A USB Host controller error was encountered. This is returned in the low nibble of the return code.

4.2.2.2.5.2 Still Image Read Operations

Once an object is opened using [STILLIMAGE_IOCTL_OPEN_OBJECT](#) then [vos_dev_read\(\)](#) can be used to stream data from the object. There can be only one object opened at a time and it must be closed when complete.

It is possible to stream directly to a file using the [fat_fileWrite\(\)](#) function. This can take a handle to a device and stream a set amount of data from the device to a file. Alternatively, data can be read into a buffer with [vos_dev_read\(\)](#) and handled accordingly.

There is no support for writing to a Still Image device using [vos_dev_write\(\)](#).

Example

```
stillimage_ioctl_cb_t camera_iocb;
stillimage_ioctl_cb_object_info_t obj_info;
int handle;
file_context_t FILE;

camera_iocb.ioctl_code = STILLIMAGE_IOCTL_OPEN_OBJECT;
camera_iocb.set = &handle;

if (vos_dev_ioctl(hCamera, &camera_iocb) != STILLIMAGE_OK)
{
    break;
}

if (fat_fileOpen(fatContext, &FILE, &obj_info.name, FILE_MODE_WRITE | FILE_MODE_PLUS) != FAT_OK)
{
    break;
}

if (fat_fileWrite(&FILE, obj_info.len, NULL, hCamera, NULL) != FAT_OK)
{
    break;
}

if (fat_fileClose(&FILE) != FAT_OK)
{
    break;
}
```

4.2.2.2.5.3 Still Image Class Driver IOCTL Calls

Calls to Still Image driver IOCTL method take the form:

```
typedef struct _stillimage_ioctl_cb_t {
    unsigned char ioctl_code;
    // read buffer
    unsigned char *get;
    // write buffer
    unsigned char *set;
} stillimage_ioctl_cb_t;
```

The following IOCTL request codes are supported.

<u>STILLIMAGE_IOCTL_ATTACH</u>	Attach a USB device to the driver
<u>STILLIMAGE_IOCTL_GET_FIRST_OBJECT</u>	Get a handle to the first object on the device
<u>STILLIMAGE_IOCTL_GET_OBJECT_INFO</u>	Get an information structure for the given handle
<u>STILLIMAGE_IOCTL_OPEN_OBJECT</u>	Open the object using a handle
<u>STILLIMAGE_IOCTL_CLOSE_OBJECT</u>	Close object from a handle
<u>STILLIMAGE_IOCTL_DELETE_OBJECT</u>	Delete the object referred to from a handle
<u>STILLIMAGE_IOCTL_INITIATE_CAPTURE</u>	If supported by the device, initiate an image capture

Description

Attaches the Still Image driver to an interface in the USB Host controller. The host controller handle is that obtained from [`vos_dev_open\(\)`](#) when the USBHost driver was opened. This function checks the class, subclass and protocol values of the device to ensure they are compatible with the driver.

Parameters

The device interface handle and USB Host controller handle must be passed in a structure using the `set` member of the IOCTL structure.

```
typedef struct _stillimage_ioctl_cb_attach_t
{
    VOS_HANDLE hc_handle;
    usbhost_device_handle *ifDev;
} stillimage_ioctl_cb_attach_t;
```

Returns

STILLIMAGE_OK if the attach is successful

STILLIMAGE_INVALID_PARAMETER if the parameters are incorrect

STILLIMAGE_NOT_FOUND if the interface does not have control and a bulk IN and bulk OUT endpoint

STILLIMAGE_ERROR when the class, subclass and protocol for the device does not match that supported

Example

```
VOS_HANDLE      hUsb1,
                hCamera;
usbhost_device_handle *ifDev1;
usbhost_ioctl_cb_t hc_iocb;
usbhost_ioctl_cb_class_t hc_iocb_class;
stillimage_ioctl_cb_t camera_iocb;
stillimage_ioctl_cb_attach_t camera_att;

// find Still Image class device
hc_iocb_class.dev_class = USB_CLASS_IMAGE;
hc_iocb_class.dev_subclass = USB_SUBCLASS_IMAGE_STILLIMAGE;
hc_iocb_class.dev_protocol = USB_PROTOCOL_IMAGE_PIMA;

// user ioctl to find first hub device
hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
hc_iocb.handle.dif = NULL;
hc_iocb.set = &hc_iocb_class;
hc_iocb.get = &ifDev1;

if (vos_dev_ioctl(hUsb1, &hc_iocb) != USBHOST_OK)
{
    break;
}

// now we have a device, initialise a camera driver for it
hCamera = vos_dev_open(VOS_DEV_STILL_IMAGE);

// boms_attach
camera_att.hc_handle = hUsb1;
camera_att.ifDev = ifDev1;

camera_iocb.ioctl_code = STILLIMAGE_IOCTL_ATTACH;
camera_iocb.set = &camera_att;
camera_iocb.get = NULL;

if (vos_dev_ioctl(hCamera, &camera_iocb) != STILLIMAGE_OK)
{
    break;
}
```

Description

Obtains a handle to the first object in the storage of the device.

Parameters

The handle is a 32 bit value returned in the variable pointed to by the `get` member of the IOCTL structure.

Returns

The handle is updated and one of the following statuses is returned.

STILLIMAGE_OK if the attach is successful

STILLIMAGE_INVALID_PARAMETER if the parameters are incorrect

STILLIMAGE_NOT_FOUND if there are no objects on the device

STILLIMAGE_ERROR the command sent to the device failed

Example

See example in [STILLIMAGE_IOCTL_OPEN_OBJECT](#).

Description

Opens the object in the storage of the device given a handle. Once opened then the data from the object can be read using [vos_dev_read\(\)](#).

Parameters

The handle is a 32 bit value sent in the variable pointed to by the `set` member of the IOCTL structure.

Returns

STILLIMAGE_OK if the attach is successful

STILLIMAGE_INVALID_PARAMETER if the parameters are incorrect

STILLIMAGE_ERROR the command sent to the device failed

Example

```
VOS_HANDLE hCamera;
stillimage_ioctl_cb_t camera_iocb;
int handle;

camera_iocb.ioctl_code = STILLIMAGE_IOCTL_GET_FIRST_OBJECT;
camera_iocb.get = &handle;

if (vos_dev_ioctl(hCamera, &camera_iocb) != STILLIMAGE_OK)
{
    break;
}

camera_iocb.ioctl_code = STILLIMAGE_IOCTL_OPEN_OBJECT;
camera_iocb.set = &handle;

if (vos_dev_ioctl(hCamera, &camera_iocb) != STILLIMAGE_OK)
{
    break;
}

camera_iocb.ioctl_code = STILLIMAGE_IOCTL_CLOSE_OBJECT;

if (vos_dev_ioctl(hCamera, &camera_iocb) != STILLIMAGE_OK)
{
    break;
}

camera_iocb.ioctl_code = STILLIMAGE_IOCTL_DELETE_OBJECT;
camera_iocb.set = &handle;

if (vos_dev_ioctl(hCamera, &camera_iocb) != STILLIMAGE_OK)
{
    break;
}
```

Description

Obtains information about an object from the device. This includes the filename, file length and type of file. The type of file is 0x3801 for JPG files.

Parameters

The handle is a 32 bit value sent in the variable pointed to by the `set` member of the IOCTL structure.

A structure to receive the object information is obtained by passing a pointer to the following structure in the `get` member of the IOCTL structure.

```
typedef struct _stillimage_ioctl_cb_object_info_t
{
    unsigned int len;
    char name[11];
    unsigned short format;
} stillimage_ioctl_cb_object_info_t;
```

Returns

The object information is updated in the structure and the return code is one of the following.

STILLIMAGE_OK if the attach is successful

STILLIMAGE_INVALID_PARAMETER if the parameters are incorrect

STILLIMAGE_ERROR the command sent to the device failed

Example

```
stillimage_ioctl_cb_t camera_iocb;
stillimage_ioctl_cb_object_info_t obj_info;
int handle;
int i;

camera_iocb.ioctl_code = STILLIMAGE_IOCTL_GET_FIRST_OBJECT;
camera_iocb.get = &handle;

if (vos_dev_ioctl(hCamera, &camera_iocb) != STILLIMAGE_OK)
{
    break;
}

camera_iocb.ioctl_code = STILLIMAGE_IOCTL_GET_OBJECT_INFO;
camera_iocb.set = &handle;
camera_iocb.get = &obj_info;

if (vos_dev_ioctl(hCamera, &camera_iocb) != STILLIMAGE_OK)
{
    break;
}

for (i = 0; i < 11; i++) printf(obj_info.name[i];
printf(" length: %d\n", obj_info.len);
```

Description

Closes an object that is open in the storage of the device given a handle.

Parameters

The handle is a 32 bit value sent in the variable pointed to by the `set` member of the IOCTL structure.

Returns

STILLIMAGE_OK if the attach is successful
STILLIMAGE_INVALID_PARAMETER if the parameters are incorrect
STILLIMAGE_ERROR the command sent to the device failed

Example

See example in [STILLIMAGE_IOCTL_OPEN_OBJECT](#).

Description

Deletes an object on the device's storage given a handle. The object must not be opened with [STILLIMAGE_IOCTL_OPEN_OBJECT](#).

Parameters

The handle is a 32 bit value sent in the variable pointed to by the `set` member of the IOCTL structure.

Returns

STILLIMAGE_OK if the attach is successful
STILLIMAGE_INVALID_PARAMETER if the parameters are incorrect
STILLIMAGE_ERROR the command sent to the device failed

Example

See example in [STILLIMAGE_IOCTL_OPEN_OBJECT](#).

Description

If supported by the device, initiate an image capture and return a handle to the object captured in the storage of the device.

Parameters

The handle is a 32 bit value returned in the variable pointed to by the `get` member of the IOCTL structure.

Returns

STILLIMAGE_OK if the attach is successful
STILLIMAGE_INVALID_PARAMETER if the parameters are incorrect
STILLIMAGE_NOT_FOUND if there are no objects on the device
STILLIMAGE_ERROR the command sent to the device failed

Example

```
stillimage_ioctl_cb_t camera_iocb;  
stillimage_ioctl_cb_object_info_t obj_info;  
int handle;  
  
camera_iocb.ioctl_code = STILLIMAGE_IOCTL_INITIATE_CAPTURE;  
camera_iocb.get = &handle;  
  
if (vos_dev_ioctl(hCamera, &camera_iocb) != STILLIMAGE_OK)  
{  
    break;  
}  
  
camera_iocb.ioctl_code = STILLIMAGE_IOCTL_GET_OBJECT_INFO;  
camera_iocb.set = &handle;  
camera_iocb.get = &obj_info;  
  
if (vos_dev_ioctl(hCamera, &camera_iocb) != STILLIMAGE_OK)  
{  
    break;  
}
```

4.2.2.2.5.4 stillimage_init()

Syntax

```
void stillimage_init(unsigned char devNum)
```

Description

Initialise the Still Image driver for the port specified in the context and registers the driver with the Device Manager.

Parameters

The device number to use when registering the driver with the Device Manager is passed in the devNum parameter.

Returns

The function does not return any values.

Comments

The driver can be attached to a Still Image device once the USB Host enumeration has completed.

4.2.2.3 USB Slave Class Drivers

4.2.2.3.1 FT232 USB Slave Device

Driver Hierarchy

USBSlaveFT232 Driver hierarchy:

USBSlaveFT232 Driver
USB Slave Driver
VOS Kernel
USB Slave Hardware

Library Files

USBHost.a

USBSlaveFT232.a

Header Files

USBHost.h

USBSlaveFT232.h

4.2.2.4 File Systems

4.2.2.4.1 FAT File System

The FAT File System is implemented as both an API and as a driver to allow flexibility in an application. Knowledge of the underlying aspects of the file system is not required, as low-level operations are managed by the driver or API.

There are restrictions given relating to the way in which the file system is used and guidance on how to obtain maximum performance from the file system.

The primary target application for the FAT File System is Flash disks, however, there are no design limitations on using it with hard disks with a suitable USB interface.

Driver Hierarchy

FAT Driver hierarchy:

FAT Driver	
FAT API	
BOMS Class Driver	Other MSI Driver
USB Host Driver	
VOS Kernel	
USB Host Hardware	Other Hardware

The [fatdrv_init\(\)](#) function must be called to initialise the driver before the kernel scheduler is started with [vos_start_scheduler\(\)](#).

Library Files

USBHost.a

BOMS.a

FAT.a

Header Files

USBHost.h

BOMS.h

MSI.h

FAT.h

API Hierarchy

FAT API hierarchy:

FAT API	
BOMS Class Driver	Other MSI Driver
USB Host Driver	
VOS Kernel	
USB Host Hardware	Other Hardware

The [fat_init\(\)](#) function must be called before using the FAT file system API. There is no sequential restrictions on when it is called.

Library Files

USBHost.a

BOMS.a

FAT.a

Header Files

USBHost.h

BOMS.h

MSI.h

FAT.h

4.2.2.4.1.1 FAT File System Concepts

The FAT File System supports the following features.

- Method for finding files.
- Stream functions on files:
 - Interface for reading from files to a buffer in memory or another device driver.
 - Interface for writing to a file from a buffer in memory or from another device driver.
 - Moving about in a file, seek, set position.
 - Modifying a file, setting EOF.
- File management routines, delete, modify, rename.
- Directory management routines; create directory, remove directory, change directory.

An API is provided as well as a layered driver interface. Calls to the FAT driver will be enacted by the FAT API.

Other methods of using the file system can be layered on top of the FAT File system. The [stdio](#) library provides a familiar streaming interface to the file system.

If the API is to be used then it is initialised using the [fat_init\(\)](#) call. If the driver is to be used then the [fatdrv_init\(\)](#) call must be made before the kernel scheduler is started. In both cases the device on which the file system is present need not be specified. A connection to a physical device is only made when the relevant attach function is called.

The FAT File System maintains the current directory. Changing directory in the file system will result in all subsequent commands such as file open [FAT_IOCTL_FILE_OPEN](#) or [fat_fileOpen\(\)](#) commands working on files in the current directory. Already opened files will remain working on files in the directories where they were opened.

Sectors

A sector is the smallest unit of data that can be read or written on a disk. It is normally 512 bytes. Data sizes smaller than a sector will require multiple disk operations as a read-modify-write cycle will be used.

This slows data throughput considerably resulting in slower applications.

To benefit from sector sized operations they must be aligned on sector boundaries otherwise the read-modify-write cycle will be required for completing the end of a sector and possibly for the start of the next sector.

Clusters

The natural size for a block of data to be read from or written to a disk is a cluster. A cluster is made up of 1, 2, 4, 8, 16 or 32 sectors. Disks usually have to read in a whole cluster of information to perform a sector operation.

Therefore data read or written in clusters can group together sector operations allowing the disk to work at maximum performance. Again, operations aligned to cluster boundaries will produce the best performance.

Driver or API

There is a very small performance overhead when using the driver interface over the API. This difference is due to the driver decoding the IOCTL code and calling the API commands.

The driver provides an extra level of abstraction by managing certain handles and data allowing a slight simplification of the application. It also presents a standard interface to the application which can be used for layering further drivers on top of the FAT driver. For example, the [stdio](#) runtime library uses the driver to perform file system operations.

It is recommended to use the driver in preference to the API. If possible, the [stdio](#) runtime library should be used where code requires to be written in the style of C programs on other operation systems.

All file names MUST be uppercase.

Filenames are always specified as 11 characters with space characters (ASCII 32 decimal or 0x20) padding. Filename extensions must start on character 9 in the 11 character filename string.

Long file names (on FAT32) are not and will not be supported.

Proprietary extensions to the Bulk-only Mass Storage specification are not supported.

Only disks which have a sector size of 512 bytes are currently supported.

Only FAT16 and FAT32 disk formats are supported at present.

Syntax

```
typedef void *fat_context;
```

Description

The API exposes a handle to a file system when a FAT file system is opened. This is used for certain calls into the API to ensure that the correct instance of the file system is being addressed.

Parameters

The storage structure for fat_context is reserved for use by FTDI.

FAT API File Handles

In the API, a pointer of type fat_context is obtained by the [fat_open\(\)](#) call. This points to storage allocated for the file system handle. This is destroyed when the [fat_close\(\)](#) function is called.

FAT Driver File Handles

The FAT driver manages all aspects of file system handles and does not expose it to the user.

Syntax

```
typedef struct _file_context_t
{
    unsigned char dirEntry[32];
    unsigned char mode;
    unsigned char context[22];
} file_context_t;
```

Description

Both the FAT File System API and Driver use the concept of a file handle, but they are used slightly differently. The file handle is a structure or a pointer to a structure. The file handle structure is called file_context_t and is defined in fat.h.

A pointer to a file handle is used to specify an individual file within the file system. The handle allows multiple files to be open concurrently although the same file may not be opened more than once.

A file handle may be obtained without actually opening a file. This can be used for directory operations where the contents of the file are not to be read or written.

No part of the file system handle may be altered.

Parameters

dirEntry

A copy of the directory entry for the file. This must not be altered.

mode

The current open mode of the file. This can be one of the following values (defined in fat.h) and is set by the [fat_fileOpen\(\)](#) API function or [FAT_IOCTL_FILE_OPEN](#) driver call.

FILE_MODE_HANDLE

Obtain a handle to a file but do not allow read or write

FILE_MODE_READ	operations. Must be used to obtain a handle for a directory. File pointer at Start-of-File, read only access.
FILE_MODE_WRITE	File pointer at Start-of-File, write only access. Truncate to zero length on open.
FILE_MODE_APPEND	File pointer at End-of-File, read only access.
FILE_MODE_READ_PLUS	File pointer at Start-of-File, read or write access.
FILE_MODE_WRITE_PLUS	File pointer at Start-of-File, read or write access. Truncate to zero length on open.
FILE_MODE_APPEND_PLUS	File pointer at End-of-File, read or write access.

context

The context member is reserved for use by FTDI.

FAT API File Handles

In the API, the structure allocated by the application to store information about an open file. A pointer to the file handle is passed to [fat_fileOpen\(\)](#) and this is used to store the information in the structure.

Making the application responsible for allocating memory for file handles allows more control over the number of handles allocated and reuse of existing handles. It also allows an application to preallocate a handle to improve response time when opening a file.

If no updates to a file or directory handle are made then the file handle may be reused without closing the file. If the file or directory has been modified then the [fat_fileClose\(\)](#) function must be called before the file handle can be reused.

FAT Driver File Handles

The FAT driver will allocate memory for a file handle and manage the use of the file handle throughout. The handle will be created when [FAT_IOCTL_FILE_OPEN](#) IOCTL is called in the driver and destroyed when [FAT_IOCTL_FILE_CLOSE](#) is called.

Syntax

```
#define FAT12  0x12
#define FAT16  0x16
#define FAT32  0x32
```

Description

The FAT File system supports FAT32 and FAT16. FAT12 is currently not supported but detected by the file system initialisation process.

The three definitions above are found in fat.h header file and are used to determine the file system type. Any other value indicates an unsupported file system type.

FAT File System dates and times are represented by an encoded 32 bit number for the date and time; or a 16 bit number representing only a date.

Where a date and time are represented, the upper 16 bits are the date and the lower 16 bits the time.

32 Bit Values	16 Bit Values	Description	Allowable Values	Meaning
25:31	9:15	Year	0 – 127	0 = 1980 127 = 2107
21:24	5:8	Months	1 – 12	1 = January 12 = December
16:20	0:4	Days	1 – 31	1 = first day of month
11:15	N/A	Hours	0 – 23	24 hour clock
5:10	N/A	Minutes	0 – 59	

0:4	N/A	Seconds/2	0 – 29	0 = 0 seconds 29 = 58 seconds
-----	-----	-----------	--------	----------------------------------

The default time on the FAT File System is 0x3c210000 (2010-01-01 00:00:00 - 1st January 2010).

The time may be set before a command with a call to [FAT_IOCTL_DIR_SETTIME](#) in the driver or [fat_time\(\)](#) in the API. All subsequent file operations will update the time in the directory table with the new time set by these commands. The driver will not adjust the time and cannot keep track of the time.

4.2.2.4.1.2 FAT Return Codes

Status Codes

All calls to the BOMS driver will return one of the following MSI (Mass Storage Interface) status codes.

- FAT_OK**
The command completed successfully.
- FAT_NOT_FOUND**
A file open, a change directory or a directory search operation did not find the named file.
- FAT_READ_ONLY**
A read only file has been opened for writing or appending, or an attempt has been made to write to a file that is open for reading.
- FAT_PENDING**
Not currently used.
- FAT_INVALID_PARAMETER**
A parameter in an IOCTL request outwith the valid range.
- FAT_INVALID_BUFFER**
Not currently used.
- FAT_INVALID_FILE_TYPE**
An attempt has been made to open either a directory or a volume ID entry as a file. Alternatively an invalid FAT file system was detected (possibly NTFS).
- FAT_EXISTS**
An open file operation reports that the file to open already exists. This is not necessarily an error.

It may be an error response when a create directory operation reports that the target directory name already exists.
- FAT_BPB_INVALID**
The boot partition on a disk is invalid and cannot be used.
- FAT_NOT_OPEN**
An operation on a disk failed because the FAT file system has not been initialised correctly. Alternatively, a read or write operation to a file has failed because the file is not opened.
- FAT_EOF**
The end of file has been reached. This may be because there is no space on the disk to write more data to a file, or a read operation has reached the end of a file.
- FAT_DIRECTORY_TABLE_FULL**
For FAT12 and FAT16 drives there is no space left in the root directory table. This is a fixed size and cannot be extended.
- FAT_DISK_FULL**
There are no free clusters on a disk to store any additional data, create new files or directories.
- FAT_ERROR**
An unexpected or unsupported operation was encountered.
- FAT_MSI_ERROR**
An error was passed from the Mass Storage Interface layer underneath the FAT driver.

4.2.2.4.1.3 FAT File System Driver

Syntax

```
typedef struct _fat_stream_t {  
    // file context  
    file_context_t *file_ctx;  
    // read/write buffer  
    unsigned char *buf;  
    // length of buffer  
    // maximum size of data to read or write  
    unsigned long len;  
    unsigned long actual;  
} fat_stream_t;
```

Description

To read and write to a file using the FAT driver a stream transfer block is used. This is a structure that is sent to [vos_dev_read\(\)](#) and [vos_dev_write\(\)](#) to describe to the FAT driver what file to use on the device and what data to transfer.

It specifies the file context of an open file on the disk, the buffer and length of the buffer, and returns the actual amount of data transferred.

To perform a read or a write operation, the application must fill in the data in the stream transfer block and then send a pointer to the transfer block to the driver using [vos_dev_read\(\)](#) or [vos_dev_write\(\)](#). The driver returns the number of bytes transferred in the transaction in the same structure.

Parameters

<code>file_ctx</code>	The context handle for the open file to be used.
<code>buf</code>	A buffer which is used as the target or source data for the operation.
<code>len</code>	The actual length of the buffer in buf.
<code>actual</code>	A storage area for the BOMS driver to build a transfer descriptor for the USB Host controller.

Remarks

The `num_to_read` and `num_read` parameters in the calls to [vos_dev_read\(\)](#) or [vos_dev_write\(\)](#) are currently not used. For future compatibility please set `num_to_read` to `sizeof(fat_stream_t)` and `num_read` to `NULL`.

Example

```
VOS_HANDLE hFAT; // initialised, attached FAT file system driver  
  
unsigned long readdatafile(char *buf, unsigned long len)  
{  
    // for opening files  
    char *file = "DATA    TXT";  
    fat_ioctl_cb_t fat_ioctl;  
    fatdrv_ioctl_cb_file_t fileInfo;  
    FILE *fd;  
    // for reading files  
    fat_stream_t fst;  
    unsigned char ret;  
    unsigned long actual;  
  
    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_OPEN;  
    fat_ioctl.file_ctx = 0xffff; // dummy value - not required  
    fat_ioctl.set = &fileInfo;
```

```
fileInfo.filename = file;
fileInfo.mode = FILE_MODE_READ;

if (vos_dev_ioctl(hFAT, &fat_ioctl) == FAT_OK)
{
    fd = fat_ioctl.file_ctx;

    // setup a read structure
    fst.buf = buf;
    fst.len = len;
    fst.file_ctx = fd;

    ret = vos_dev_read(hFAT, (char *)&fst, sizeof(fat_stream_t), NULL);

    if (ret == FAT_OK || ret == FAT_EOF)
    {
        actual = fst.actual;
    }

    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_CLOSE;
    fat_ioctl.file_ctx = fd;

    vos_dev_ioctl(hFAT, &fat_ioctl);
}
return actual;
}
```

Calls to the FAT driver's IOCTL method take the form:

```
typedef struct _fat_ioctl_cb_t {
    unsigned char ioctl_code;
    // file context
    file_context_t *file_ctx;
    // read buffer
    unsigned char *get;
    // write buffer
    unsigned char *set;
} fat_ioctl_cb_t;
```

The following IOCTL request codes for a Mass Storage Interface are supported by the FAT driver.

<u>FAT_IOCTL_FS_ATTACH</u>	Attach a Mass Storage Interface to the FAT File System driver
<u>FAT_IOCTL_FS_DETACH</u>	Detach the Mass Storage Interface from the FAT File System driver
<u>FAT_IOCTL_FS_INFO</u>	Return volume information for the Mass Storage Device

The following IOCTL request codes for File Operations are supported by the FAT driver.

<u>FAT_IOCTL_FILE_OPEN</u>	Open a file and return a handle to the file
<u>FAT_IOCTL_FILE_CLOSE</u>	Close a file using a file handle
<u>FAT_IOCTL_FILE_SEEK</u>	Seek to a relative position in a file
<u>FAT_IOCTL_FILE_SETPOS</u>	Set the current position in a file
<u>FAT_IOCTL_FILE_TELL</u>	Return the current position in a file
<u>FAT_IOCTL_FILE_REWIND</u>	Set the current position to the start of a file
<u>FAT_IOCTL_FILE_TRUNCATE</u>	Truncate a file at the current position
<u>FAT_IOCTL_FILE_DELETE</u>	Delete a file
<u>FAT_IOCTL_FILE_RENAME</u>	Rename a file
<u>FAT_IOCTL_FILE_MOD</u>	Modify the attributes of a file

The following IOCTL request codes for Directory Operations are supported by the FAT driver.

<u>FAT_IOCTL_DIR_FIND</u>	Finds a file or directory with a specified name in the current directory
<u>FAT_IOCTL_DIR_FIND_FIRST</u>	Finds the first file or directory in the current directory
<u>FAT_IOCTL_DIR_FIND_NEXT</u>	Finds subsequent files or directories in the current directory
<u>FAT_IOCTL_DIR_CD</u>	Changes the current directory
<u>FAT_IOCTL_DIR_MK</u>	Makes a new directory
<u>FAT_IOCTL_DIR_SIZE</u>	Obtains the size of a file
<u>FAT_IOCTL_DIR_GETTIME</u>	Gets the create, modify and access time information for a file or directory
<u>FAT_IOCTL_DIR_SETTIME</u>	Sets the create, modify and access time information for a file or directory
<u>FAT_IOCTL_DIR_IEMPTY</u>	Tests whether a directory is empty
<u>FAT_IOCTL_DIR_ISVALID</u>	Tests if a file handle is valid
<u>FAT_IOCTL_DIR_ISVOLUMELABEL</u>	Tests if a file handle is a volume label
<u>FAT_IOCTL_DIR_ISREADONLY</u>	Tests if a file handle is read only
<u>FAT_IOCTL_DIR_ISFILE</u>	Tests if a file handle is a valid file
<u>FAT_IOCTL_DIR_ISDIRECTORY</u>	Tests if a file handle is a valid directory

Description

Attaches the FAT driver to a Mass Storage Interface device. This can be a [BOMS Class](#) or other device conforming to the Mass Storage Interface (MSI) defined in header file msi.h. THE MSI device must be opened with `vos_dev_open()` the handle obtained passed to this function The partition number to use on the disk is specified along with the handle of the MSI device in a special structure.

Parameters

The MSI device interface handle disk partition number must be passed in a structure using the `set` member of the [fat_ioctl_cb_t](#) IOCTL structure.

```
typedef struct _fatdrv_ioctl_cb_attach_t
{
    // handle to initialised MSI device
    VOS_HANDLE msi_handle;
    // partition on device to use
    unsigned char partition;
} fatdrv_ioctl_cb_attach_t;
```

Returns

There is no data returned by this call although the return value indicates success or otherwise of the attach.

FAT_OK is successful attachment of the device
 FAT_INVALID_FILE_TYPE if the file system is not FAT32, FAT 16 or FAT12
 FAT_BPB_INVALID if the sector size of not 512 bytes or the boot sector is invalid
 FAT_ERROR if the device cannot be read

Example

```
void BOMSFindDevice()
{
    VOS_HANDLE hUsb2, hBoms, hFat;
    usbhost_device_handle *ifDev2;
    usbhost_ioctl_cb_t hc_iocb;
    usbhost_ioctl_cb_class hc_iocb_class;
    msi_ioctl_cb_t boms_iocb;
    boms_ioctl_cb_attach_t boms_att;
    fat_ioctl_cb_t fat_iocb;
```

```
fatdrv_ioctl_cb_attach_t fat_att;

// find BOMS class device
hc_iocb_class.dev_class = USB_CLASS_MASS_STORAGE;
hc_iocb_class.dev_subclass = USB_SUBCLASS_MASS_STORAGE_SCSI;
hc_iocb_class.dev_protocol = USB_PROTOCOL_MASS_STORAGE_BOMS;

hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
hc_iocb.handle.dif = NULL;
hc_iocb.set = &hc_iocb_class;
hc_iocb.get = &ifDev2;

if (vos_dev_ioctl(hUsb2, &hc_iocb) != USBHOST_OK)
{
    // no BOMS class found
}

// now we have a device, initialise a BOMS driver for it
hBoms = vos_dev_open(VOS_DEV_BOMS);

// boms_attach
boms_att.hc_handle = hUsb2;
boms_att.ifDev = ifDev2;

boms_iocb.ioctl_code = MSI_IOCTL_BOMS_ATTACH;
boms_iocb.set = &boms_att;
boms_iocb.get = NULL;

if (vos_dev_ioctl(hBoms, &boms_iocb) != MSI_OK)
{
    // could not attach to device
}

// now we have a BOMS device, initialise a FAT driver for it
hFat = vos_dev_open(VOS_DEV_FAT);

fat_iocb.ioctl_code = FAT_IOCTL_FS_ATTACH;
fat_iocb.set = &atInfo;
atInfo.msi_handle = bomsDev;
atInfo.partition = 0;

if (vos_dev_ioctl(hFat, &fat_iocb) != FAT_OK)
{
    // could not attach to FAT device
}

// device has been found and opened

// now detach from the device
fat_iocb.ioctl_code = FAT_IOCTL_FS_DETACH;

vos_dev_ioctl(hFat, &fat_iocb)

boms_iocb.ioctl_code = MSI_IOCTL_BOMS_DETACH;

vos_dev_ioctl(hBoms, &boms_iocb)
}
```

Description

Detaches a FAT driver from an MSI device.

Parameters

There are no parameters passed to this IOCTL.

Returns

There is no data returned by this call. The return value will be the following:

FAT_OK successfully received current file pointer

Example

See example in [FAT_IOCTL_FS_ATTACH](#).

Description

Gets file system information about a disk and populates an application supplied structure. The file system information is always of a fixed size.

Parameters

There are no parameters for this call.

Returns

The file system information is copied into the structure pointed to by the `get` member of the [fat_ioctl_cb_t](#) IOCTL structure.

```
typedef struct _fatdrv_ioctl_cb_fs_t
{
    // file system type
    char fsType;
    // free space on disk in bytes
    unsigned int freeSpaceH;
    unsigned int freeSpaceL;
    // capacity of disk in bytes
    unsigned int capacityH;
    unsigned int capacityL;
    // number of bytes in a cluster
    unsigned int bytesPerCluster;
    // number of bytes in a sector
    unsigned short bytesPerSector;
    // volume id
    unsigned long volID;
} fatdrv_ioctl_cb_fs_t;
```

The file system type `fsType` is one of the types defined in [FAT File System Types](#).

The capacity and free space are 64 bit values which are represented as two 32 bit numbers.

A scan of free space on a disk will be performed when this call is made unless a scan has been done previously. This means that all clusters on the disk will be checked to see if they are used or unused. This can take a considerable time on large disks or disks with small cluster sizes. Typically it will complete in 10 to 30 seconds but can take over 60 seconds on some disks.

The volume ID is a 32 bit unique value associated with the file system during a format operation. It must not be confused with the volume label obtained

The return value will be the following:

FAT_OK successfully received current file size
FAT_INVALID_PARAMETER the `get` value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
void diskParams(VOS_HANDLE hDisk)
{
    fat_ioctl_cb_t disk_iocb;
    fatdrv_ioctl_cb_fs_t disk_iocb_info;

    disk_iocb.ioctl_code = FAT_IOCTL_FS_INFO;
    disk_iocb.get = &disk_iocb_info;
```

```
vos_dev_ioctl(hDisk,&disk_iocb);

if (disk_iocb_info.fsType == FAT32)
{
    // FAT32 file system detected
    if (disk_iocb_info.freeSpaceH > 0)
    {
        // more than 4GB available on disk!
    }
}
```

Description

Opens a file or directory by name in the current directory. It can be used for opening files for read, write access or simply obtaining a handle to a file without allowing access to the contents.

Directories may only be opened by mode `FILE_MODE_HANDLE`. This can be used to rename directories (with [FAT_IOCTL_FILE_RENAME](#)) or change the attributes of a directory (with [FAT_IOCTL_FILE_MOD](#)).

Parameters

The call takes a structure containing file information which is pointed to by the `set` member of the [fat_ioctl_cb_t](#) IOCTL structure. The `fatdrv_ioctl_cb_file_t` structure is defined in `fat.h`.

```
typedef struct _fatdrv_ioctl_cb_file_t
{
    // filename
    char *filename;
    // offset within file or size of file
    int offset;
    // access mode of file, seek mode or file attribute
    char mode;
} fatdrv_ioctl_cb_file_t;
```

The `filename` member shall contain a pointer to an 11 character file name. It must use space characters (ASCII 32 decimal or 0x20) as padding. Filename extensions must start on character 9 in the 11 character filename string.

The `offset` member is not used in `FAT_IOCTL_FILE_OPEN`.

The `mode` member is one of the [File Mode Values](#) defined in the [FAT File Handle](#) structure.

Returns

The following value may be returned by this call:

- FAT_OK successful file open
- FAT_NOT_FOUND file system type invalid or file system not attached, open a file for reading which does not exist
- FAT_INVALID_FILE_TYPE attempt to open a volume ID directory entry or directory as a file
- FAT_READ_ONLY opening a read only file with a write or append mode
- FAT_DISK_FULL no free clusters found in which to store file data
- FAT_DIRECTORY_TABLE_FULL root directory on FAT12 and FAT16 disks has no free entries
- FAT_INVALID_PARAMETER the `set` value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

A [FAT File Handle](#) is returned in the `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure. This is used for subsequent access to the file. It must be destroyed using [FAT_IOCTL_FILE_CLOSE](#).

Example

```
FILE *openlog(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_file_t fileInfo;
    char file[11] = "MYBIGDATLOG"; // mybigdat.log
    FILE *fd;
```

```
fat_ioctl.ioctl_code = FAT_IOCTL_FILE_OPEN;
fat_ioctl.file_ctx = 0xffff;
fat_ioctl.set = &fileInfo;
fileInfo.filename = file;
fileInfo.mode = FILE_MODE_APPEND_PLUS; // eof, R/W access

if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
{
    fd = fat_ioctl.file_ctx;
    return fd;
}
else
{
    return NULL;
}

void closelog(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;

    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_CLOSE;

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        // closed
    }
}
```

Description

Closes a file and commits any changes to the file and its directory table entry (size, filename, attributes) to the disk.

Parameters

There are no parameters passed to this IOCTL.

Returns

There is no data returned by this call. The return value will be the following:

FAT_OK successfully received current file pointer

Example

See example in [FAT_IOCTL_FILE_OPEN](#)

Description

Seeks to a specified offset in a file. The offset can be relative to the start, current file pointer or end of a file.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#).

The call takes the same `fatdrv_ioctl_cb_file_t` structure containing file information as [FAT_IOCTL_FILE_OPEN](#). The structure is pointed to by the `set` member of the [fat_ioctl_cb_t](#) IOCTL structure.

The `filename` member of the `fatdrv_ioctl_cb_file_t` structure is not used in `FAT_IOCTL_FILE_SEEK`.

The `mode` member takes a value depending on where the relative position of current file pointer to the `offset` member is calculated. The `offset` member is treated as a signed value and may be

negative. The new position in the file is calculated as follows:

FAT_SEEK_CUR	The offset is added to the current file position
FAT_SEEK_END	The offset must be negative and the new position is the end of file plus the offset value
FAT_SEEK_SET	The offset must be positive and the file position is set to the offset from the start of file

Returns

There is no data returned by this call. However, return values may be one of the following:

FAT_OK successfully moved the file pointer to the new location
FAT_EOF the new file pointer is beyond the EOF of the current file or is negative
FAT_INVALID_PARAMETER the set value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
char skipon(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_file_t fileInfo;

    if (fd == NULL)
    {
        return -1;
    }

    // move 256 bytes further on in a file
    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_SEEK;
    fat_ioctl.file_ctx = fd;
    fat_ioctl.set = &fileInfo;
    fileInfo.offset = 256;
    fileInfo.mode = FAT_SEEK_CUR;

    if (vos_dev_ioctl(hFat, &fat_ioctl) != FAT_OK)
    {
        return -1;
    }

    return 0;
}
```

Description

Moves the current file pointer to a specified offset from the start of a file.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#).

The call takes the same `fatdrv_ioctl_cb_file_t` structure containing file information as [FAT_IOCTL_FILE_OPEN](#). The structure is pointed to by the `set` member of the [fat_ioctl_cb_t](#) IOCTL structure.

The `filename` member of the `fatdrv_ioctl_cb_file_t` structure is not used in `FAT_IOCTL_FILE_SETPOS`.

The `offset` member is treated as an unsigned value and is used as the new location in the file for the current file pointer.

Returns

There is no data returned by this call. However, return values may be one of the following:

FAT_OK successfully moved the file pointer to the new location
FAT_EOF the new file pointer is beyond the EOF

FAT_INVALID_PARAMETER the set value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
char skipheader(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_file_t fileInfo;

    if (fd == NULL)
    {
        return -1;
    }

    // move to an offset of 256 bytes in a file to skip over a header
    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_SETPOS;
    fat_ioctl.file_ctx = fd;
    fat_ioctl.set = &fileInfo;
    fileInfo.offset = 256;

    if (vos_dev_ioctl(hFat, &fat_ioctl) != FAT_OK)
    {
        return -1;
    }

    return 0;
}
```

Description

Obtains the current file pointer for a file.

Parameters

The file_ctx member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#).

The call takes the same fatdrv_ioctl_cb_file_t structure containing file information as [FAT_IOCTL_FILE_OPEN](#). The structure is pointed to by the get member of the [fat_ioctl_cb_t](#) IOCTL structure.

The filename and mode members of the fatdrv_ioctl_cb_file_t structure is not used in FAT_IOCTL_FILE_TELL.

Returns

The offset member of the fatdrv_ioctl_cb_file_t structure obtains the current file pointer. The return value will be one of the following:

FAT_OK successfully received current file pointer
FAT_INVALID_PARAMETER the get value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
unsigned long getposition(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_file_t fileInfo;

    if (fd == NULL)
    {
        return -1;
    }

    // get position in file
    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_TELL;
    fat_ioctl.file_ctx = fd;
    fat_ioctl.get = &fileInfo;
}
```

```
if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
{
    return fileInfo.offset;
}

return -1;
}
```

Description

Moves the current file pointer to the start of a file.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#).

Returns

There is no data returned by this call. The return value will be the following:

FAT_OK successfully moved the file pointer to the new location

Example

```
char back2start(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;

    if (fd == NULL)
    {
        return -1;
    }

    // move to start of file
    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_REWIND;
    fat_ioctl.file_ctx = fd;

    if (vos_dev_ioctl(hFat, &fat_ioctl) != FAT_OK)
    {
        return -1;
    }

    return 0;
}
```

Description

Truncates a file to the position of the current file pointer.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#).

Returns

There is no data returned by this call. The return value will be the following:

FAT_OK successfully truncated the file

Example

```
char cutat64k(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_file_t fileInfo;

    if (fd == NULL)
    {

```

```
        return -1;
    }

    // move to an offset of 65535 bytes
    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_SETPOS;
    fat_ioctl.file_ctx = fd;
    fat_ioctl.set = &fileInfo;
    fileInfo.offset = 65535;

    // truncate at current position (65535 bytes)
    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_TRUNCATE;
    fat_ioctl.file_ctx = fd;

    if (vos_dev_ioctl(hFat, &fat_ioctl) != FAT_OK)
    {
        return -1;
    }

    return 0;
}
```

Description

Deletes a file or directory using a [FAT File Handle](#) obtained from [FAT_IOCTL_FILE_OPEN](#). The delete function does not delete a file or directory based on a name but rather a handle. The file or directory must be opened first and then deleted. The file or directory must be opened with a [file mode](#) of `FILE_MODE_HANDLE` to ensure no changes to the file are made before deletion.

The file handle must be destroyed after the delete by closing the file or directory with [FAT_IOCTL_FILE_CLOSE](#). This will also synchronise the directory table and remove the file or directory from there.

Directories to be deleted must be empty - have no files or sub-directories.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#).

Returns

There is no data returned by this call. The return value will be one of the following:

- FAT_OK successfully deleted the file
- FAT_INVALID_FILE_TYPE file not opened with mode `FILE_MODE_HANDLE`
- FAT_EXISTS the file handle points to a directory that is not empty

Example

```
char delfile(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_file_t fileInfo;
    char file[11] = "MYBIGDATLOG"; // mybigdat.log
    FILE *fd;
    char status = -1;

    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_OPEN;
    fat_ioctl.file_ctx = 0xffff;
    fat_ioctl.set = &fileInfo;
    fileInfo.filename = file;
    fileInfo.mode = FILE_MODE_HANDLE; // no file access

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        // delete file
        fat_ioctl.ioctl_code = FAT_IOCTL_FILE_DELETE;

        if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
        {

```

```
        status = 0;
    }

    // free FILE pointer
    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_CLOSE;

    vos_dev_ioctl(hFat, &fat_ioctl);
}
return status;
}
```

Description

Renames a file or directory using a [FAT File Handle](#) obtained from [FAT_IOCTL_FILE_OPEN](#). The rename function will change the name of a file or directory to the that specified in the `filename` member of `fatdrv_ioctl_cb_file_t` structure. The file must be opened first and then renamed. The file or directory must be opened with a [file mode](#) of `FILE_MODE_HANDLE` to ensure no changes to the file are made before deletion.

Closing the file or directory after the rename with [FAT_IOCTL_FILE_CLOSE](#) is required to synchronise the directory table entry.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#).

The call takes the same `fatdrv_ioctl_cb_file_t` structure containing file information as [FAT_IOCTL_FILE_OPEN](#). The structure is pointed to by the `set` member of the [fat_ioctl_cb_t](#) IOCTL structure.

The `filename` member of the `fatdrv_ioctl_cb_file_t` structure is used for the new name of the file or directory.

The `offset` and `mode` members are not used.

Returns

There is no data returned by this call. The return value will be one of the following:

- FAT_OK successfully deleted the file
- FAT_INVALID_FILE_TYPE file not opened with mode `FILE_MODE_HANDLE`
- FAT_INVALID_PARAMETER the set value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
char renamelog(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_file_t fileInfo;
    char filesrc[11] = "MYBIGDATLOG"; // mybigdat.log
    char filedst[11] = "MYBIGDATBAK"; // mybigdat.bak
    FILE *fd;

    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_OPEN;
    fat_ioctl.file_ctx = 0xffff;
    fat_ioctl.set = &fileInfo;
    fileInfo.filename = filesrc;
    fileInfo.mode = FILE_MODE_HANDLE; // no file access

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        fat_ioctl.ioctl_code = FAT_IOCTL_FILE_RENAME;
        fileInfo.filename = filedst;

        if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
        {
            // rename successful
        }
    }
}
```

```
fat_ioctl.ioctl_code = FAT_IOCTL_FILE_CLOSE;

if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
{
    // closed
}

return 0;
}

return 1;
}
```

Description

The mod function will change the attributes of a file or directory to the that specified in the `mode` member of `fatdrv_ioctl_cb_file_t` structure. The file or directory must be opened first with a [file mode](#) other than `FILE_MODE_READ`.

Closing the file or directory after the mod with [FAT_IOCTL_FILE_CLOSE](#) is required to synchronise the directory table entry.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#).

The call takes the same `fatdrv_ioctl_cb_file_t` structure containing file information as [FAT_IOCTL_FILE_OPEN](#). The structure is pointed to by the `set` member of the [fat_ioctl_cb_t](#) IOCTL structure.

The `mode` member of the `fatdrv_ioctl_cb_file_t` structure is used for the new attribute mask for the file or directory. Bits may be one or more of the following.

FAT_ATTR_READ_ONLY	Read only attribute
FAT_ATTR_HIDDEN	Hidden file attribute
FAT_ATTR_SYSTEM	System file attribute
FAT_ATTR_ARCHIVE	Archive flag attribute
FAT_ATTR_DIRECTORY	Directory bit Must be set if attributes of a directory are changed, must be clear if a file attributes are changed

The `offset` and `filename` members are not used.

Returns

There is no data returned by this call. The return value will be one of the following:

- FAT_OK successfully deleted the file
- FAT_INVALID_FILE_TYPE file not opened with mode `FILE_MODE_HANDLE`
- FAT_INVALID_PARAMETER the `set` value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
char readonlylogfile(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_file_t fileInfo;
    char file[11] = "MYBIGDATLOG"; // mybigdat.log
    FILE *fd;

    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_OPEN;
    fat_ioctl.file_ctx = 0xffff;
    fat_ioctl.set = &fileInfo;
    fileInfo.filename = file;
```

```
fileInfo.mode = FILE_MODE_HANDLE; // no file access

if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
{
    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_MOD;
    fileInfo.mode = FAT_ATTR_READ_ONLY;

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        // read only flag set successful
    }

    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_CLOSE;

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        // closed
    }
    return 0;
}
return 1;
}
```

Description

Searches in the current directory for a file or directory matching the name specified in the parameters of the call. The filename is specified in the `filename` member of the `fatdrv_ioctl_cb_dir_t` structure.

Wildcards are not permitted. To search through filenames and apply search conditions use [FAT_IOCTL_DIR_FIND_FIRST](#) and [FAT_IOCTL_DIR_FIND_NEXT](#).

Parameters

The call takes a structure containing directory information which is pointed to by the `set` member of the [fat_ioctl_cb_t](#) IOCTL structure. The `fatdrv_ioctl_cb_dir_t` structure is defined in `fat.h`.

```
typedef struct _fatdrv_ioctl_cb_dir_t
{
    char *filename;
} fatdrv_ioctl_cb_dir_t;
```

The `filename` member shall contain a pointer to an 11 character file name. It must use space characters (ASCII 32 decimal or 0x20) as padding.

Returns

The return value will be one of the following:

- FAT_OK successfully received current file pointer
- FAT_NOT_FOUND a matching file was not found
- FAT_EOF no matching file was found but directory table is full
- FAT_INVALID_PARAMETER the `set` value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

A [FAT File Handle](#) is returned in the `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure if a matching file is found. This can be used for subsequent access to the file or directory. The file handle is opened with a [file mode](#) of `FILE_MODE_HANDLE`. It must be destroyed using [FAT_IOCTL_FILE_CLOSE](#).

Example

```
char checkforfile(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_dir_t dirInfo;
    char file[11] = "SVNFILES  ";

    fat_ioctl.ioctl_code = FAT_IOCTL_DIR_FIND;
    fat_ioctl.set = dirInfo;
```

```
dirInfo.filename = file;

if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
{
    // file exists - destroy pointer
    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_CLOSE;
    vos_dev_ioctl(hFat, &fat_ioctl);

    return 0;
}

return 1;
}
```

Description

Searches in the current directory for all files and directories. [FAT_IOCTL_DIR_FIND_FIRST](#) initialises a search whereas [FAT_IOCTL_DIR_FIND_NEXT](#) is used to continue searching through the files in the current directory.

Parameters

The call takes the same `fatdrv_ioctl_cb_dir_t` structure containing directory information as [FAT_IOCTL_DIR_FIND](#). The structure is pointed to by the `get` member of the [fat_ioctl_cb_t](#) IOCTL structure.

The `filename` member must point to a buffer of at least 11 bytes.

Returns

The return value will be one of the following:

- FAT_OK successfully received current file pointer
- FAT_NOT_FOUND a matching file was not found
- FAT_EOF no matching file was found but directory table is full
- FAT_INVALID_PARAMETER the filename pointer is NULL
- FAT_INVALID_PARAMETER the set value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

A [FAT File Handle](#) is returned in the `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure if any file is found. This can be used for subsequent access to the file or directory. The file handle is opened with a [file mode](#) of `FILE_MODE_HANDLE`. Subsequent calls to [FAT_IOCTL_DIR_FIND_NEXT](#) will reuse the same file handle. When finished with the handle it must be destroyed using [FAT_IOCTL_FILE_CLOSE](#).

The name of the file found is copied into the buffer pointed to by the `filename` member of the `fatdrv_ioctl_cb_dir_t` structure.

Example

```
char processXfiles(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_dir_t dirInfo;
    char file[11];

    fat_ioctl.ioctl_code = FAT_IOCTL_DIR_FIND_FIRST;
    fat_ioctl.get = dirInfo;
    dirInfo.filename = file;

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        // file exists
        do
        {
            if (file[0] == 'X')
            {
                // process files beginning with X
            }
        }
    }
}
```

```
fat_ioctl.ioctl_code = FAT_IOCTL_DIR_FIND_NEXT;

} while (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK);

// finished with handle, destroy
fat_ioctl.ioctl_code = FAT_IOCTL_FILE_CLOSE;
vos_dev_ioctl(hFat, &fat_ioctl);
}
}
```

Description

Searches in the current directory for subsequent files and directories continuing a [FAT_IOCTL_DIR_FIND_FIRST](#) search.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#) from a successful [FAT_IOCTL_DIR_FIND_FIRST](#) search.

The call takes the same `fatdrv_ioctl_cb_dir_t` structure containing directory information as [FAT_IOCTL_DIR_FIND](#). The structure is pointed to by the `get` member of the [fat_ioctl_cb_t](#) IOCTL structure.

The `filename` member must point to a buffer of at least 11 bytes.

Returns

The return value will be one of the following:

- FAT_OK successfully received current file pointer
- FAT_NOT_FOUND a matching file was not found
- FAT_EOF no matching file was found but directory table is full
- FAT_INVALID_PARAMETER the filename pointer is NULL
- FAT_INVALID_PARAMETER the set value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

The [FAT File Handle](#) in the `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure is updated. This can be used for subsequent access to the file or directory. The file handle is opened with a [file mode](#) of `FILE_MODE_HANDLE`. Subsequent calls to [FAT_IOCTL_DIR_FIND_NEXT](#) will reuse the same file handle. When finished with the handle it must be destroyed using [FAT_IOCTL_FILE_CLOSE](#).

The name of the file found is copied into the buffer pointed to by the `filename` member of the `fatdrv_ioctl_cb_dir_t` structure.

Example

See example in [FAT_IOCTL_DIR_FIND_FIRST](#).

Description

Changes the current directory to a sub-directory specified in the `filename` member of `fatdrv_ioctl_cb_dir_t` structure.

A special case value of `".. "` (2 dots followed by 9 spaces) may be used to move up to a higher level directory or NULL to the top level directory.

Parameters

The call takes the same `fatdrv_ioctl_cb_dir_t` structure containing directory information as [FAT_IOCTL_DIR_FIND](#). The structure is pointed to by the `set` member of the [fat_ioctl_cb_t](#) IOCTL structure.

The `filename` member of the `fatdrv_ioctl_cb_dir_t` structure is used for the destination directory name. The value of NULL will change the current directory to the volume's root directory.

Returns

There is no data returned by this call. The return value will be one of the following:

FAT_OK successfully changed the current directory
FAT_NOT_FOUND directory not changed as destination directory not found
FAT_INVALID_PARAMETER the set value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
char changetoroot(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_dir_t dirInfo;

    fat_ioctl.ioctl_code = FAT_IOCTL_DIR_CD;
    fat_ioctl.set = dirInfo;
    dirInfo.filename = NULL;

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        return 0;
    }

    return 1;
}

char changeup(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_dir_t dirInfo;
    char file[11] = "..    ";

    fat_ioctl.ioctl_code = FAT_IOCTL_DIR_CD;
    fat_ioctl.set = dirInfo;
    dirInfo.filename = file;

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        return 0;
    }

    return 1;
}

char changetosubdir(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_dir_t dirInfo;
    char file[11] = "SVNFILES ";

    fat_ioctl.ioctl_code = FAT_IOCTL_DIR_CD;
    fat_ioctl.set = dirInfo;
    dirInfo.filename = file;

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        return 0;
    }

    return 1;
}
```

Description

Make a new sub-directory in the current directory. The name is specified in the `filename` member of `fatdrv_ioctl_cb_dir_t` structure.

Parameters

The call takes the same `fatdrv_ioctl_cb_dir_t` structure containing directory information as [FAT_IOCTL_DIR_FIND](#). The structure is pointed to by the `set` member of the [fat_ioctl_cb_t](#) IOCTL structure.

The `filename` member of the `fatdrv_ioctl_cb_dir_t` structure is used for the new directory name. This must not exist in the current directory.

Returns

There is no data returned by this call. The return value will be one of the following:

- FAT_OK successfully created the new directory
- FAT_EXISTS directory not created as a directory or file with that name already exists
- FAT_DIRECTORY_TABLE_FULL a FAT16 or FAT12 file system has the root directory table full - this is a fixed size
- FAT_DISK_FULL there were no free clusters found for creating a new directory table
- FAT_INVALID_PARAMETER the `set` value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
char makesub(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_dir_t dirInfo;
    char file[11] = "SVNFILES ";

    fat_ioctl.ioctl_code = FAT_IOCTL_DIR_MK;
    fat_ioctl.set = dirInfo;
    dirInfo.filename = file;

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        return 0;
    }

    return 1;
}
```

Description

Obtains the size of a file.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#).

The call takes the same `fatdrv_ioctl_cb_file_t` structure containing file information as [FAT_IOCTL_FILE_OPEN](#). The structure is pointed to by the `get` member of the [fat_ioctl_cb_t](#) IOCTL structure.

The `filename` and `mode` members of the `fatdrv_ioctl_cb_file_t` structure is not used in `FAT_IOCTL_DIR_SIZE`.

Returns

The `offset` member of the `fatdrv_ioctl_cb_file_t` structure obtains the file size. The return value will be one of the following:

- FAT_OK successfully received current file size
- FAT_INVALID_PARAMETER the `get` value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
unsigned long getsize(VOS_HANDLE hFat, FILE *fd)
{
```

```
fat_ioctl_cb_t fat_ioctl;
fatdrv_ioctl_cb_file_t fileInfo;

if (fd == NULL)
{
    return -1;
}

// get size of file
fat_ioctl.ioctl_code = FAT_IOCTL_DIR_SIZE;
fat_ioctl.file_ctx = fd;
fat_ioctl.get = &fileInfo;

if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
{
    return fileInfo.offset;
}

return -1;
}
```

Description

Obtains the create, modification and last access times of a file or directory. The encoded dates and times are described in the [FAT File System Date and Times](#) topic.

Parameters

The file_ctx member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#).

The call takes a structure containing directory information which is pointed to by the get member of the [fat_ioctl_cb_t](#) IOCTL structure. The fatdrv_ioctl_cb_time_t structure is defined in fat.h.

```
typedef struct _fatdrv_ioctl_cb_time_t
{
    // crtDate and crtTime used for set time and get time methods
    unsigned short crtDate;
    unsigned short crtTime;
    // wrtDate, wrtTime and accDate used for get time method only
    unsigned short wrtDate;
    unsigned short wrtTime;
    unsigned short accDate;
} fatdrv_ioctl_cb_time_t;
```

Returns

The file times are filled out in the fatdrv_ioctl_cb_time_t structure.

The return value will be one of the following:

FAT_OK successfully received current file size

FAT_INVALID_PARAMETER the get value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
unsigned long getmodtime(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_time_t timeInfo;
    unsigned long tmod;

    if (fd == NULL)
    {
        return -1;
    }

    // get time info for file
    fat_ioctl.ioctl_code = FAT_IOCTL_DIR_GETTIME;
```

```
fat_ioctl.file_ctx = fd;
fat_ioctl.get = timeInfo;

if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
{
    tmod = timeInfo.wrtTime;
    tmod |= (timeInfo.wrtDate << 16);

    return tmod;
}

return -1;
}
```

Description

Sets the time value for subsequent create, modify or accesses of files or directories. The encoded date and time is described in the [FAT File System Date and Times](#) topic.

Parameters

The call takes the same `fatdrv_ioctl_cb_time_t` structure containing directory information as [FAT_IOCTL_DIR_GETTIME](#). The structure is pointed to by the `set` member of the [fat_ioctl_cb_t](#) IOCTL structure.

The `crtDate` and `ctrTime` members of the `fatdrv_ioctl_cb_time_t` structure are used for the new file system time. Other members are not used.

Returns

The return value will be one of the following:

FAT_OK successfully received current file size

FAT_INVALID_PARAMETER the `set` value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

Example

```
unsigned long setfstime(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_time_t timeInfo;
    unsigned long tmod;

    if (fd == NULL)
    {
        return -1;
    }

    // get time
    fat_ioctl.ioctl_code = FAT_IOCTL_DIR_GETTIME;
    fat_ioctl.file_ctx = fd;
    fat_ioctl.get = timeInfo;

    // jun 7th 2010 12:00:00 - 0x3cc76000
    timeInfo.crtTime = 0x6000;
    timeInfo.crtDate = 0x3cc7;

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        return 0;
    }

    return -1;
}
```

Description

Returns a flag indicating if a directory is empty.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#) pointing to a directory.

Returns

There is no data returned by this call. The return value will be the following:

- 1 if the directory is empty
- 0 if there are one or more files or directories

Example

```
char logdirexists(VOS_HANDLE hFat)
{
    fat_ioctl_cb_t fat_ioctl;
    fatdrv_ioctl_cb_file_t fileInfo;
    char file[11] = "LOGDIR    "; // logdir
    FILE *fd;
    char status = -1;

    fat_ioctl.ioctl_code = FAT_IOCTL_FILE_OPEN;
    fat_ioctl.file_ctx = 0xffff;
    fat_ioctl.set = &fileInfo;
    fileInfo.filename = file;
    fileInfo.mode = FILE_MODE_HANDLE; // no file access

    if (vos_dev_ioctl(hFat, &fat_ioctl) == FAT_OK)
    {
        // check if directory is empty
        fat_ioctl.ioctl_code = FAT_IOCTL_DIR_ISEMPTY;

        status = vos_dev_ioctl(hFat, &fat_ioctl);

        // free FILE pointer
        fat_ioctl.ioctl_code = FAT_IOCTL_FILE_CLOSE;

        vos_dev_ioctl(hFat, &fat_ioctl);
    }
    return status;
}
```

Description

Returns a flag indicating if a file or directory is valid.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#) pointing to a file or directory.

Returns

There is no data returned by this call. The return value will be the following:

- 1 if the file or directory is valid
- 0 if the file handle points to an entry which is not a file or directory

Example

```
char checkfile(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
```

```
char status;

// check if fd is a valid file
fat_ioctl.ioctl_code = FAT_IOCTL_DIR_ISVALID;
fat_ioctl.file_ctx = fd;

status = vos_dev_ioctl(hFat, &fat_ioctl);

return status;
}
```

Description

Returns a flag indicating if a file handle is a Volume Label entry on FAT32 file system.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#) pointing to a file or directory.

Returns

There is no data returned by this call. The return value will be the following:

- 1 if the file handle is a Volume Label entry
- 0 if the file handle points to an entry which is not a Volume Label

Example

```
char checkvalid(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
    char status;

    // check if fd is a volume label
    fat_ioctl.ioctl_code = FAT_IOCTL_DIR_ISVOLUME LABEL;
    fat_ioctl.file_ctx = fd;

    status = vos_dev_ioctl(hFat, &fat_ioctl);

    return status;
}
```

Description

Returns a flag indicating if a file or directory is read only.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#) pointing to a file or directory.

Returns

There is no data returned by this call. The return value will be the following:

- 1 if the file or directory is read only
- 0 if the file handle points to an entry which is not read only

Example

```
char checkreadonly(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
    char status;
```

```
// check if fd is read only
fat_ioctl.ioctl_code = FAT_IOCTL_DIR_ISREADONLY;
fat_ioctl.file_ctx = fd;

status = vos_dev_ioctl(hFat, &fat_ioctl);

return status;
}
```

Description

Returns a flag indicating if a file handle points to a file rather than a directory or a Volume ID.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#) pointing to a file or directory.

Returns

There is no data returned by this call. The return value will be the following:

- 1 if the file handle is a file
- 0 if the file handle points to an entry which is not a file

Example

```
char checkfile(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
    char status;

    // check if fd is a file
    fat_ioctl.ioctl_code = FAT_IOCTL_DIR_ISFILE;
    fat_ioctl.file_ctx = fd;

    status = vos_dev_ioctl(hFat, &fat_ioctl);

    return status;
}
```

Description

Returns a flag indicating if a file handle points to a directory rather than a file or a Volume ID.

Parameters

The `file_ctx` member of the [fat_ioctl_cb_t](#) IOCTL structure must contain a valid [FAT File Handle](#) pointing to a file or directory.

Returns

There is no data returned by this call. The return value will be the following:

- 1 if the file handle is a directory
- 0 if the file handle points to an entry which is not a directory

Example

```
char checkdir(VOS_HANDLE hFat, FILE *fd)
{
    fat_ioctl_cb_t fat_ioctl;
    char status;
```

```
// check if fd is a file
fat_ioctl.ioctl_code = FAT_IOCTL_DIR_ISDIRECTORY;
fat_ioctl.file_ctx = fd;

status = vos_dev_ioctl(hFat, &fat_ioctl);

return status;
}
```

Syntax

```
unsigned char fatdrv_init (
    unsigned char devNum
);
```

Description

Initialise the FAT driver and registers the driver with the Device Manager.

Parameters

devNum
The device number to use when registering the FAT driver with the Device Manager.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

Memory is allocated dynamically for an instance of the FAT driver when this call is made. It is never freed by the driver.

4.2.2.4.1.4 FAT File System API

The FAT File System API provides direct access to the file system commands.

Initialisation Calls

fat_init()	Initialise the FAT API
fat_time()	Sets the create, modify and access time information for a file or directory
fat_open()	Open the FAT API to use a Mass Storage Interface handle and partition on the device
fat_close()	Close the FAT API

File System Calls

fat_freeSpace()	Obtain the free space available on the device
fat_capacity()	Get the total capacity
fat_bytesPerCluster()	Get the number of bytes per cluster
fat_bytesPerSector()	Return the number of bytes in a sector
fat_getFSType()	Get the file system type
fat_getVolumeID	Find the Volume ID of the file system
fat_getVolumeLabel	Find the Volume Label for FAT12 and FAT16 file system formats

File Functions

fat_fileOpen()	Open a file and return a handle to the file
fat_fileClose()	Close a file using a file handle
fat_fileSeek()	Seek to a relative position in a file
fat_fileSetPos()	Set the current position in a file
fat_fileTell()	Return the current position in a file
fat_fileRewind()	Set the current position to the start of a file
fat_fileTruncate()	Truncate a file at the current position
fat_fileDelete()	Delete a file
fat_fileRename()	Rename a file
fat_fileMod()	Modify the attributes of a file
fat_fileRead()	Read from a file
fat_fileWrite()	Write to a file

Directory Table Functions

fat_dirTableFind()	Finds a file or directory with a specified name in the current directory
fat_dirTableFindFirst()	Finds the first file or directory in the current directory
fat_dirTableFindNext()	Finds subsequent files or directories in the current directory
fat_dirChangeDir()	Changes the current directory
fat_dirCreateDir()	Makes a new directory
fat_dirDirIsEmpty()	Tests whether a directory is empty
fat_dirEntryIsValid()	Tests if a file handle is valid
fat_dirEntryIsVolumeID()	Tests if a file handle is a volume ID
fat_dirEntryIsReadOnly()	Tests if a file handle is read only
fat_dirEntryIsFile()	Tests if a file handle is a valid file
fat_dirEntryIsDirectory()	Tests if a file handle is a valid directory
fat_dirEntrySize()	Obtains the size of a file
fat_dirEntryTime()	Gets the create, modify and access time information for a file or directory

Syntax

```
void fat_init(void)
```

Description

Initialises the FAT API. This need only be called once before any other FAT API call is made. It need only be called after kernel initialisation takes place with [vos_init\(\)](#), but may be called from a user application after the scheduler is started with [vos_start_scheduler\(\)](#).

Parameters

The `fat_init()` function takes no parameters.

Return Value

The `fat_init()` function returns no data.

Syntax

```
fat_context *fat_open(VOS_HANDLE hMsi, unsigned char partition, unsigned char *status)
```

Description

Opens the FAT API and attaches a Mass Storage Interface device to the API. This can be a [BOMS Class](#) or other device conforming to the Mass Storage Interface (MSI) defined in header file msi.h. The MSI device must be initialised and opened with [vos_dev_open\(\)](#). The partition number to use on the disk is specified along with the handle of the MSI device in a special

Parameters

hMsi
Handle to an open, initialised Mass Storage Device.

partition
Partition number on the device to attach to. A value of zero will attach to the first partition (partition 1).

status
The status returned from the function. May be NULL if value not required to be checked.

Return Value

The function returns a `fat_context` pointer. This is a handle to the instance of the FAT API used to address the correct MSI device. This is used to address the FAT API in most calls to the API.

A status value is passed back in the `status` parameter. If this is not used then it may be set to NULL and will be ignored by `fat_open`. The values are

Example

```
fat_context *opendisk(VOS_HANDLE hUsbHost, VOS_HANDLE hBoms)
{
    usbhost_ioctl_cb_t hc_iocb; // ioctl block
    usbhost_ioctl_cb_class_t hc_iocb_class;
    usbhost_device_handle *ifDev; // handle to the next device interface
    msi_ioctl_cb_t boms_cb;
    boms_ioctl_cb_attach_t boms_att;
    fat_context fatContext = NULL;

    hc_iocb_class.dev_class = USB_CLASS_MASS_STORAGE;
    hc_iocb_class.dev_subclass = USB_SUBCLASS_MASS_STORAGE_SCSI;
    hc_iocb_class.dev_protocol = USB_PROTOCOL_MASS_STORAGE_BOMS;

    usbhost_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
    usbhost_iocb.handle.dif = NULL;
    usbhost_iocb.set = &hc_iocb_class;
    usbhost_iocb.get = &ifDev;

    if (vos_dev_ioctl(hUsbHost, &usbhost_iocb) != USBHOST_OK) return NULL;

    boms_cb.ioctl_code = MSI_IOCTL_BOMS_ATTACH;
    boms_cb.set = &boms_att;
    boms_att.hc_handle = hUsbHost;
    boms_att.ifDev = ifDev;

    status = vos_dev_ioctl(hBoms, &boms_cb);
    if (status == MSI_OK)
    {
        fat_init();
        fatContext = fat_open(hBoms, 0, &status);
        if (status != FAT_OK)
        {
            // open failed
            return NULL;
        }
    }
    else

```

```
{
    // BOMS attach failed
    return NULL;
}
// success
return fatContext;
}

closedisk(fat_context *fatContext)
{
    fat_close(fatContext);
}
```

Syntax

```
void fat_close(fat_context *fat_ctx)
```

Description

Closes the FAT API and detaches it from the Mass Storage Interface device.

Parameters

fat_ctx
Pointer to the instance of the FAT API to close.

Return Value

There is no value returned.

Example

See example in [fat_open\(\)](#).

Syntax

```
unsigned char fat_freeSpace(fat_context *fat_ctx, unsigned long *bytes_h, unsigned long *bytes_l,
```

Description

Will scan a disk to calculate the amount of free space available in bytes. A complete scan of the disk will be performed if necessary or requested.

A full scan can take a considerable time on large disks or disks with small cluster sizes. Typically it will complete in 10 to 30 seconds but can take over 60 seconds on some disks.

Parameters

fat_ctx
Pointer to the instance of the FAT API

bytes_h, bytes_l
Two 32 bit long values to receive the 64 bit count of free space on the disk

scan
One of the following values to determine the type of free space scan

FAT_FREESPACE_NO_SCAN	Do not perform a scan but use data from previous scans. If no data is available then free space count is undefined.
FAT_FREESPACE_SCAN	Scan only if a scan has not previously been performed.
FAT_FREESPACE_FORCE_SCAN	Force a scan of the disk to be performed.

Return Value

The 64 bit count of free space is returned in the **bytes_h** and **bytes_l** parameters.

The return value will be the following:

FAT_OK successfully moved the file pointer to the new location

Example

```
unsigned long bytes_l, bytes_h;  
  
fat_freeSpace(fatContext, &bytes_h, &bytes_l, FAT_FREESPACE_SCAN);
```

Syntax

```
unsigned char fat_capacity(fat_context *fat_ctx, unsigned long *bytes_h, unsigned long *bytes_l)
```

Description

This will calculate the total capacity of a disk partition. This is different from the total formatted capacity as it does not take account of reserved areas of the disk.

Parameters

fat_ctx
Pointer to the instance of the FAT API

bytes_h, bytes_l
Two 32 bit long values to receive the 64 bit count of bytes on the disk

Return Value

The 64 bit count of disk capacity is returned in the `bytes_h` and `bytes_l` parameters.

The return value will be the following:

FAT_OK successfully moved the file pointer to the new location

Example

```
unsigned long bytes_l, bytes_h;  
  
fat_capacity(fatContext, &bytes_h, &bytes_l);
```

Syntax

```
unsigned char fat_bytesPerCluster(fat_context *fat_ctx, unsigned long *bytes)
```

Description

This will calculate the number of bytes per cluster for a disk.

Parameters

fat_ctx
Pointer to the instance of the FAT API

bytes
Pointer to variable to receive the calculated number of bytes in a cluster

Return Value

The 32 bit count of of bytes in a cluster in the `bytes` parameter.

The return value will be the following:

FAT_OK successfully moved the file pointer to the new location

Example

```
unsigned long bytes;  
  
fat_bytesPerCluster(fatContext, &bytes);
```

Syntax

```
unsigned char fat_bytesPerSector(fat_context *fat_ctx, unsigned short *bytes)
```

Description

This will calculate the number of bytes per sector for a disk.

Parameters

fat_ctx
Pointer to the instance of the FAT API

bytes
Pointer to variable to receive the calculated number of bytes in a sector

Return Value

The 16 bit count of of bytes in a sector in the **bytes** parameter.

The return value will be the following:

FAT_OK successfully moved the file pointer to the new location

Example

```
unsigned short bytes;  
  
fat_bytesPerSector(fatContext, &bytes);
```

Syntax

```
unsigned char fat_getFSType(fat_context *fat_ctx)
```

Description

This will determine the file system type for the attached disk.

Parameters

fat_ctx
Pointer to the instance of the FAT API

Return Value

The file system type is encoded in the return value. The file system type is one of the types defined in [FAT File System Types](#).

Example

```
if (fat_getFSType(fatContext) == FAT32)  
{  
    // FAT 32  
}
```

Syntax

```
unsigned char fat_getVolumeID(fat_context *fat_ctx, unsigned long *volID)
```

Description

This will return the volume ID for the attached disk. The volume ID is a unique value to identify the file system assigned during a format operation.

Parameters

`fat_ctx`
Pointer to the instance of the FAT API

`volID`
Pointer to a 32 bit variable to receive the volume ID value

Return Value

The 32 bit volume ID is returned in the `volID` parameter.

The return value will be the following:

FAT_OK successfully moved the file pointer to the new location

Example

```
unsigned long volid;

fat_getVolumeID(fatContext, &volid);
if (volid == 0x43234534)
{
    // This is my disk!
}
```

Syntax

```
unsigned char fat_getVolumeLabel(fat_context *fat_ctx, char *volLabel)
```

Description

This will return the volume label for the attached disk. The volume label is an 11 character string stored in the boot sector of a disk file system. It may not be valid on FAT32 file systems being replaced by a volume label entry in the root directory of the disk.

Parameters

`fat_ctx`
Pointer to the instance of the FAT API

`volLabel`
Pointer to an 11 byte buffer which receives the volume label

Return Value

The 11 byte volume label is returned to the buffer pointer to by the `volLabel` parameter.

The return value will be the following:

FAT_OK successfully moved the file pointer to the new location

Example

```
char label[12];

fat_getVolumeLabel(fatContext, label);
label[11] = '\0';
if (strcmp(label, "NO NAME") == 0)
{
    // label not used
}
```

Syntax

```
unsigned char fat_fileOpen(fat_context *fat_ctx, file_context_t *file_ctx, unsigned char *name, u
```

Description

Opens a file or directory by name in the current directory. It can be used for opening files for read, write access or simply obtaining a handle to a file without allowing access to the contents.

Directories may only be opened by mode `FILE_MODE_HANDLE`. This can be used to rename directories (with [fat_fileRename\(\)](#)) or change the attributes of a directory (with [fat_fileMod\(\)](#)).

Parameters

fat_ctx
Pointer to the instance of the FAT API

file_ctx
Pointer to memory allocated to store a file handle

name
The `filename` member shall contain a pointer to an 11 character file name. It must use space characters (ASCII 32 decimal or 0x20) as padding. Filename extensions must start on character 9 in the 11 character filename string.

mode
The `mode` member is one of the [File Mode Values](#) defined in the [FAT File Handle](#) structure.

Returns

The following value may be returned by this call:

FAT_OK successful file open
FAT_NOT_FOUND file system type invalid or file system not attached, open a file for reading which does not exist
FAT_INVALID_FILE_TYPE attempt to open a volume ID directory entry or directory as a file
FAT_READ_ONLY opening a read only file with a write or append mode
FAT_DISK_FULL no free clusters found in which to store file data
FAT_DIRECTORY_TABLE_FULL root directory on FAT12 and FAT16 disks has no free entries
FAT_INVALID_PARAMETER the set value of the [fat_ioctl_cb_t](#) IOCTL structure is NULL

A [FAT File Handle](#) is returned in the `file_ctx` parameter. This is used for subsequent access to the file. The memory is allocated in the calling procedure.

Example

```
file_context_t *openlog(fat_context_t *fatctx)
{
    char file[11] = "MYBIGDATLOG"; // mybigdat.log
    file_context_t *fd;

    fd = malloc(sizeof(file_context_t));

    if (fat_fileOpen(fatctx, fd, file, FILE_MODE_APPEND_PLUS) == FAT_OK)
    {
        return fd;
    }
    else
    {
        return NULL;
    }
}

void closelog(file_context_t *fd)
{
    fat_fileClose(fd);
    free(fd);
}
```

Syntax

```
unsigned char fat_fileClose(file_context_t *file_ctx)
```

Description

Closes a file and commits any changes to the file and it's directory table entry (size, filename, attributes) to the disk.

Parameters

file_ctx
Pointer to file handle of file to close.

Returns

There is no data returned by this call. The return value will be the following:

FAT_OK successfully received current file pointer

Example

See example in [fat_fileOpen\(\)](#)

Syntax

```
unsigned char fat_fileSeek(file_context_t *file_ctx, long offset, unsigned char mode)
```

Description

Seeks to a specified offset in a file. The offset can be relative to the start, current file pointer or end of a file.

Parameters

file_ctx
Pointer to a valid [FAT File Handle](#)

offset
A signed value with the desired number of bytes to seek

mode
Type of seek to perform. Takes a value depending on where the relative position of current file pointer to the `offset` member is calculated. The new position in the file is calculated as follows:

FAT_SEEK_CUR	The offset is added to the current file position
FAT_SEEK_END	The offset must be negative and the new position is the end of file plus the offset value
FAT_SEEK_SET	The offset must be positive and the file position is set to the offset from the start of file

Returns

The return value may be one of the following:

FAT_OK successfully moved the file pointer to the new location

FAT_EOF the new file pointer is beyond the EOF of the current file or is negative

Example

```
char skipon(file_context_t *fd)
{
    // move 256 bytes further on in a file
    if (fat_fileSeek(fd, 256, FAT_SEEK_CUR) != FAT_OK)
    {
```

```
        return -1;
    }

    return 0;
}
```

Syntax

```
unsigned char fat_fileSetPos(file_context_t *file_ctx, unsigned long offset)
```

Description

Sets the current position in a file to a specified offset.

Parameters

file_ctx
Pointer to a valid [FAT File Handle](#)

offset
An unsigned value with the desired number of bytes to seek from the start of the file

Returns

The return value may be one of the following:

FAT_OK successfully moved the file pointer to the new location
FAT_EOF the new file pointer is beyond the EOF of the current file or is negative

Example

```
char skipheader(file_context_t *fd)
{
    // move 256 bytes in to a file
    if (fat_fileSetPos(fd, 256) != FAT_OK)
    {
        return -1;
    }

    return 0;
}
```

Syntax

```
unsigned char fat_fileTell(file_context_t *file_cxt, unsigned long *offset)
```

Description

Gets the current position in a file.

Parameters

file_ctx
Pointer to a valid [FAT File Handle](#)

offset
Pointer to variable to receive the current position in a file

Returns

The *offset* parameter receives the current location in the file as a 32 bit unsigned value.

The return value may be one of the following:

FAT_OK successfully moved the file pointer to the new location

Example

```
unsigned long getposition(file_context_t *fd)
```

```
{
    unsigned long pos;

    // get position in file
    if (fat_fileTell(fd, &pos) == FAT_OK)
    {
        return pos;
    }

    return -1;
}
```

Syntax

```
unsigned char fat_fileRewind(file_context_t *file_ctx)
```

Description

Sets the current position in a file to the start.

Parameters

file_ctx
Pointer to a valid [FAT File Handle](#)

Returns

The return value may be one of the following:

FAT_OK successfully moved the file pointer to the new location

Example

```
char back2start(file_context_t *fd)
{
    // move to start of a file
    if (fat_fileRewind(fd) != FAT_OK)
    {
        return -1;
    }

    return 0;
}
```

Syntax

```
unsigned char fat_fileTruncate(file_context_t *file_ctx)
```

Description

Sets the current position in a file as EOF.

Parameters

file_ctx
Pointer to a valid [FAT File Handle](#)

Returns

The return value may be one of the following:

FAT_OK successfully moved the file pointer to the new location

Example

```
char cutat64k(file_context_t *fd)
{
```

```
// truncate file at 64k
if (fat_fileSetPos(fd, 65535) == FAT_OK)
{
    if (fat_fileTruncate(fd) == FAT_OK)
    {
        return 0;
    }
}
return -1;
}
```

Syntax

```
unsigned char fat_fileDelete(file_context_t *file_ctx)
```

Description

Deletes a file or directory using a [FAT File Handle](#) obtained from [fat_fileOpen\(\)](#). The delete function does not delete a file or directory based on a name but rather a handle. The file or directory must be opened first and then deleted. The file or directory must be opened with a [file mode](#) of `FILE_MODE_HANDLE` to ensure no changes to the file are made before deletion.

The file handle must be closed afterwards with [fat_fileClose\(\)](#). This will also synchronise the directory table and remove the file or directory from there.

Directories to be deleted must be empty - have no files or sub-directories.

Parameters

`file_ctx`
Pointer to a valid [FAT File Handle](#)

Returns

There is no data returned by this call. The return value will be one of the following:

`FAT_OK` successfully deleted the file
`FAT_INVALID_FILE_TYPE` file not opened with mode `FILE_MODE_HANDLE`
`FAT_EXISTS` the file handle points to a directory that is not empty

Example

```
char delfile(fat_context_t *fatctx)
{
    char file[11] = "MYBIGDATLOG"; // mybigdat.log
    file_context_t fd;

    if (fat_fileOpen(fatctx, &fd, file, FILE_MODE_HANDLE) == FAT_OK)
    {
        // delete file
        if (fat_fileDelete(fd) != FAT_OK)
        {
            return -1;
        }

        // free FILE pointer
        fat_fileClose(fd);
    }
    return 0;
}
```

Syntax

```
unsigned char fat_fileRename(file_context_t *file_ctx, char *name)
```

Description

Renames a file or directory using a [FAT File Handle](#) obtained from [fat_fileOpen\(\)](#). The rename function does not rename a file or directory based on a name but rather a handle. The file or directory must

be opened first and then renamed. The file or directory must be opened with a [file mode](#) of `FILE_MODE_HANDLE` to ensure no changes to the file are made before deletion.

The file handle must be closed afterwards with [fat_fileClose\(\)](#). This will also synchronise the directory table and remove the file or directory from there.

Parameters

`file_ctx`
Pointer to a valid [FAT File Handle](#)

`name`
New name of file or directory

Returns

There is no data returned by this call. The return value will be one of the following:

`FAT_OK` successfully renamed the file

`FAT_INVALID_FILE_TYPE` file not opened with mode `FILE_MODE_HANDLE`

Example

```
char renamelog(fat_context_t *fatctx)
{
    char filesrc[11] = "MYBIGDATLOG"; // mybigdat.log
    char filedst[11] = "MYBIGDATBAK"; // mybigdat.bak
    file_context_t fd;
    char status = -1;

    if (fat_fileOpen(fatctx, &fd, filesrc, FILE_MODE_HANDLE) == FAT_OK)
    {
        fat_fileRename(fd, filedst) == FAT_OK)
        {
            // rename successful
            status = 0;
        }

        fat_fileClose(fd);
    }

    return status;
}
```

Syntax

```
unsigned char fat_fileMod(file_context_t *file_ctx, unsigned char attr)
```

Description

The mod function will change the attributes of a file or directory to the that specified in the `attr` parameter. The file or directory must be opened first with a [file mode](#) other than `FILE_MODE_READ`.

Closing the file or directory after the mod with [fat_fileClose\(\)](#) is required to synchronise the directory table entry.

Parameters

`file_ctx`
Pointer to a valid [FAT File Handle](#)

`attr`
The new attribute mask for the file or directory. Bits may be one or more of the following.

<code>FAT_ATTR_READ_ONLY</code>	Read only attribute
<code>FAT_ATTR_HIDDEN</code>	Hidden file attribute
<code>FAT_ATTR_SYSTEM</code>	System file attribute

FAT_ATTR_ARCHIVE	Archive flag attribute
FAT_ATTR_DIRECTORY	Directory bit Must be set if attributes of a directory are changed, must be clear if a file attributes are changed

Returns

There is no data returned by this call. The return value will be one of the following:

FAT_OK successfully changed the attributes of the file
FAT_INVALID_FILE_TYPE file opened with mode FILE_MODE_READ

Example

```
char readonlylogfile(fat_context_t *fatctx)
{
    char file[11] = "MYBIGDATLOG"; // mybigdat.log
    file_context_t fd;

    if (fat_fileOpen(fatctx, &fd, file, FILE_MODE_HANDLE) == FAT_OK)
    {
        // set file readonly
        if (fat_fileMode(fd, FAT_ATTR_READ_ONLY) != FAT_OK)
        {
            return -1;
        }

        // free FILE pointer
        fat_fileClose(fd);
    }
    return 0;
}
```

Syntax

```
unsigned char fat_time(unsigned long time)
```

Description

Sets the time value for subsequent create, modify or accesses of files or directories. The encoded date and time is described in the [FAT File System Date and Times](#) topic.

Parameters

time
Encoded date and time value for file accesses

Return Value

The return value will be the following:

FAT_OK successfully received current file size

Example

```
// jun 7th 2010 12:00:00 - 0x3cc76000
fat_time(0x3cc76000);
```

Syntax

```
unsigned char fat_dirTableFind(fat_context *fat_ctx, file_context_t *file_ctx, char *name)
```

Description

Searches in the current directory for a file or directory matching the name specified in the parameters

of the call. The filename is specified in the `name` parameter.

Wildcards are not permitted. To search through filenames and apply search conditions use [fat_dirTableFindFirst\(\)](#) and [fat_dirTableFindNext\(\)](#).

Parameters

`fat_ctx`
Pointer to the instance of the FAT API

`file_ctx`
Pointer to a [FAT File Handle](#) structure

`name`
Contains a pointer to an 11 character file name. It must use space characters (ASCII 32 decimal or 0x20) as padding.

Returns

The return value will be one of the following:

FAT_OK successfully received current file pointer
FAT_NOT_FOUND a matching file was not found
FAT_EOF no matching file was found but directory table is full

A [FAT File Handle](#) is returned in the `file_ctx` parameter if a matching file is found. This can be used for subsequent access to the file or directory. The file handle is opened with a [file mode](#) of `FILE_MODE_HANDLE`.

Example

```
char checkforfile(fat_context_t *fatctx)
{
    char file[11] = "SVNFILES  ";
    file_context_t fd;

    if (fat_dirTableFind(fatctx, &fd, file)
    {
        // file exists
        return 0;
    }

    return 1;
}
```

Syntax

```
unsigned char fat_dirTableFindFirst(fat_context *fat_ctx, file_context_t *file_ctx)
```

Description

Searches in the current directory for all files and directories. Initialises a search whereas [fat_dirTableFindNext\(\)](#) is used to continue searching through the files in the current directory.

Parameters

`fat_ctx`
Pointer to the instance of the FAT API

`file_ctx`
Pointer to a [FAT File Handle](#) structure

Returns

The return value will be one of the following:

FAT_OK successfully received current file pointer
FAT_NOT_FOUND a matching file was not found
FAT_EOF no matching file was found but directory table is full

A [FAT File Handle](#) is returned in the `file_ctx` parameter if any file is found. This can be used for

subsequent access to the file or directory. The file handle is opened with a [file mode](#) of `FILE_MODE_HANDLE`. Subsequent calls to [fat_dirTableFindNext\(\)](#) must reuse the same file handle.

The name of the file found is the first 11 bytes of the `file_ctx` pointer.

Example

```
char processXfiles(fat_context_t *fatctx)
{
    char file[11];
    file_context_t fd;
    char *file = (char*)&fd;

    if(fat_dirTableFindFirst(fatctx, &fd) == FAT_OK)
    {
        // file exists
        do
        {
            if (file[0] == 'X')
            {
                // process files beginning with X
            }

        } while (fat_dirTableFindNext(fatctx, &fd) == FAT_OK);
    }
}
```

Syntax

```
unsigned char fat_dirTableFindNext(fat_context *fat_ctx, file_context_t *file_ctx)
```

Description

Searches in the current directory for subsequent files and directories continuing a [fat_dirTableFindFirst\(\)](#) search.

Parameters

`fat_ctx`
Pointer to the instance of the FAT API

`file_ctx`
Pointer to a [FAT File Handle](#) structure

Returns

The return value will be one of the following:

`FAT_OK` successfully received current file pointer
`FAT_NOT_FOUND` a matching file was not found
`FAT_EOF` no matching file was found but directory table is full

The [FAT File Handle](#) in the `file_ctx` parameter is updated. This can be used for subsequent access to the file or directory. The file handle is opened with a [file mode](#) of `FILE_MODE_HANDLE`. Subsequent calls to [fat_dirTableFindNext\(\)](#) must reuse the same file handle.

The name of the file found is the first 11 bytes of the `file_ctx` pointer.

Example

See example in [fat_dirTableFindFirst\(\)](#)

Syntax

```
unsigned char fat_dirChangeDir(fat_context *fat_ctx, unsigned char *name)
```

Description

Changes the current directory to a sub-directory specified in the `name` parameter.

A special case value of `".. "` (2 dots followed by 9 spaces) may be used to move up to a higher level directory or NULL to the top level directory.

Parameters

`fat_ctx`

Pointer to the instance of the FAT API

`name`

The destination directory name. The value of NULL will change the current directory to the volume's root directory.

Returns

There is no data returned by this call. The return value will be one of the following:

FAT_OK successfully changed the current directory

FAT_NOT_FOUND directory not changed as destination directory not found

Example

```
char changetoroot(fat_context_t *fatctx)
{
    if (fat_dirChangeDir(fatctx, NULL) == FAT_OK)
    {
        return 0;
    }

    return 1;
}

char changeup(fat_context_t *fatctx)
{
    char file[11] = "..          ";

    if (fat_dirChangeDir(fatctx, file) == FAT_OK)
    {
        return 0;
    }

    return 1;
}

char changetosubdir(fat_context_t *fatctx)
{
    char file[11] = "SVNFILES  ";

    if (fat_dirChangeDir(fatctx, file) == FAT_OK)
    {
        return 0;
    }

    return 1;
}
```

Syntax

```
unsigned char fat_dirCreateDir(fat_context *fat_ctx, unsigned char *name)
```

Description

Make a new sub-directory in the current directory. The name is specified in the `name` parameter.

Parameters

`fat_ctx`

Pointer to the instance of the FAT API

name

The new directory name. This must not exist in the current directory.

Returns

There is no data returned by this call. The return value will be one of the following:

- FAT_OK successfully created the new directory
- FAT_EXISTS directory not created as a directory or file with that name already exists
- FAT_DIRECTORY_TABLE_FULL a FAT16 or FAT12 file system has the root directory table full - this is a fixed size
- FAT_DISK_FULL there were no free clusters found for creating a new directory table

Example

```
char makesub(fat_context_t *fatctx)
{
    char file[11] = "SVNFILES  ";

    if (fat_dirChangeDir(fatctx, file) == FAT_OK)
    {
        return 0;
    }

    return 1;
}
```

Syntax

```
unsigned long fat_dirEntrySize(file_context_t *file_ctx)
```

Description

Obtains the size of a file.

Parameters

file_ctx

Must contain a valid [FAT File Handle](#) pointing to a file.

Returns

The return value is the size of the file in bytes.

Example

```
unsigned long getsize(fat_context_t *fatctx)
{
    char file[11] = "FILETESTDAT";
    file_context_t fd;
    unsigned long size = -1;

    if (fat_fileOpen(fatctx, &fd, file, FILE_MODE_HANDLE) == FAT_OK)
    {
        // check if file is read only
        size = fat_dirEntrySize(&fd);
    }
    return size;
}
```

Syntax

```
unsigned short fat_dirEntryTime(file_context_t *file_ctx, unsigned char offset)
```

Description

Obtains the create, modification or last access dates or times of a file or directory. The encoded dates and times are described in the [FAT File System Date and Times](#) topic.

Parameters

file_ctx
Must contain a valid [FAT File Handle](#) pointing to a file.

offset
The offset parameter is one of the following:

- FAT_DIRENTRYTIME_CREATE_DATE File create date
- FAT_DIRENTRYTIME_CREATE_TIME File create time
- FAT_DIRENTRYTIME_MODIFY_DATE File modify date
- FAT_DIRENTRYTIME_MODIFY_TIME File modify time
- FAT_DIRENTRYTIME_ACCESS_DATE File last access date

Returns

The return value is a 16 bit value containing either a date or a time encoded value described in the [FAT File System Date and Times](#) topic.

Example

```
unsigned long getmodtime(fat_context_t *fatctx)
{
    char file[11] = "FILETESTDAT";
    file_context_t fd;
    unsigned long date = -1;

    if (fat_fileOpen(fatctx, &fd, file, FILE_MODE_HANDLE) == FAT_OK)
    {
        // get modify date and time
        date = fat_dirEntryTime(&fd, FAT_DIRENTRYTIME_MODIFY_DATE);
        date <= 16;
        date |= fat_dirEntryTime(&fd, FAT_DIRENTRYTIME_MODIFY_TIME);
    }
    return date;
}
```

Syntax

```
unsigned char fat_dirDirIsEmpty(file_context_t *file_ctx)
```

Description

Returns a flag indicating if a directory is empty.

Parameters

file_ctx
Must contain a valid [FAT File Handle](#) pointing to a directory.

Returns

There is no data returned by this call. The return value will be the following:

- 1 if the directory is empty
- 0 if there are one or more files or directories

Example

```
char logdirexists(fat_context_t *fatctx)
{
    char file[11] = "LOGDIR    "; // logdir
    file_context_t fd;
    char status = -1;

    if (fat_fileOpen(fatctx, &fd, file, FILE_MODE_HANDLE) == FAT_OK)
    {
        // check if directory is empty
        status = fat_dirDirIsEmpty(&fd);
    }
    return status;
}
```

Syntax

```
unsigned char fat_dirEntryIsValid(file_context_t *file_ctx)
```

Description

Returns a flag indicating if a file or directory is valid.

Parameters

`file_ctx`

Must contain a valid [FAT File Handle](#) pointing to a file or directory.

Returns

There is no data returned by this call. The return value will be the following:

1 if the file or directory is valid

0 if the file handle points to an entry which is not a file or directory

Example

```
char checkfile(fat_context_t *fatctx)
{
    char file[11] = "FILETESTDAT";
    file_context_t fd;
    char status = -1;

    if (fat_fileOpen(fatctx, &fd, file, FILE_MODE_HANDLE) == FAT_OK)
    {
        // check if file is valid
        status = fat_dirDirIsValid(&fd);
    }
    return status;
}
```

Syntax

```
unsigned char fat_dirEntryIsVolumeLabel(file_context_t *file_ctx);
```

Description

Returns a flag indicating if a file handle is a Volume Label entry on FAT32 file system.

Parameters

`file_ctx`

Must contain a valid [FAT File Handle](#) pointing to a file or directory.

Returns

There is no data returned by this call. The return value will be the following:

1 if the file handle is a Volume Label entry

0 if the file handle points to an entry which is not a Volume Label

Example

```
char checkvalid(fat_context_t *fatctx)
{
    char file[11] = "FILETESTDAT";
    file_context_t fd;
    char status = -1;

    if (fat_fileOpen(fatctx, &fd, file, FILE_MODE_HANDLE) == FAT_OK)
    {
        // check if file is a volume label
        status = fat_dirDirIsVolumeID(&fd);
    }
    return status;
}
```

Syntax

```
unsigned char fat_dirEntryIsReadOnly(file_context_t *file_ctx)
```

Description

Returns a flag indicating if a file or directory is read only.

Parameters

file_ctx

Must contain a valid [FAT File Handle](#) pointing to a file or directory.

Returns

There is no data returned by this call. The return value will be the following:

1 if the file or directory is read only

0 if the file handle points to an entry which is not read only

Example

```
char checkreadonly(fat_context_t *fatctx)
{
    char file[11] = "FILETESTDAT";
    file_context_t fd;
    char status = -1;

    if (fat_fileOpen(fatctx, &fd, file, FILE_MODE_HANDLE) == FAT_OK)
    {
        // check if file is read only
        status = fat_dirDirIsReadOnly(&fd);
    }
    return status;
}
```

Syntax

```
unsigned char fat_dirEntryIsFile(file_context_t *file_ctx)
```

Description

Returns a flag indicating if a file handle points to a file rather than a directory or a Volume ID.

Parameters

file_ctx

Must contain a valid [FAT File Handle](#) pointing to a file or directory.

Returns

There is no data returned by this call. The return value will be the following:

1 if the file handle is a file

0 if the file handle points to an entry which is not a file

Example

```
char checkfile(fat_context_t *fatctx)
{
    char file[11] = "FILETESTDAT";
    file_context_t fd;
    char status = -1;

    if (fat_fileOpen(fatctx, &fd, file, FILE_MODE_HANDLE) == FAT_OK)
    {
        // check if handle is a file
        status = fat_dirDirIsFile(&fd);
    }
    return status;
}
```

Syntax

```
unsigned char fat_dirEntryIsDirectory(file_context_t *file_ctx)
```

Description

Returns a flag indicating if a file handle points to a directory rather than a file or a Volume ID.

Parameters

`file_ctx`

Must contain a valid [FAT File Handle](#) pointing to a file or directory.

Returns

There is no data returned by this call. The return value will be the following:

1 if the file handle is a directory

0 if the file handle points to an entry which is not a directory

Example

```
char checkdir(fat_context_t *fatctx)
{
    char file[11] = "FILETESTDAT";
    file_context_t fd;
    char status = -1;

    if (fat_fileOpen(fatctx, &fd, file, FILE_MODE_HANDLE) == FAT_OK)
    {
        // check if handle is a directory
        status = fat_dirDirIsDirectory(&fd);
    }
    return status;
}
```

Syntax

```
unsigned char fat_fileRead(file_context_t *file_ctx, unsigned long length, char *buffer, VOS_HAND
```

Description

Reads a file from a disk to either a buffer or streams to an other device handle.

Parameters

<code>file_ctx</code>	Pointer to file handle of file to read.
<code>length</code>	Number of bytes to read from file.
<code>buffer</code>	Pointer to buffer which receives data from file. This buffer is only used if <code>hOutput</code> is NULL.
<code>hOutput</code>	Handle of device where data from file is sent using vos_dev_write() . The <code>buffer</code> destination is used if this is NULL.
<code>bytes_read</code>	Pointer to variable where the number of bytes read is written. If this is NULL then it is ignored.

Returns

If `hOutput` is NULL, the buffer will be updated with data from the file. Otherwise [vos_dev_write\(\)](#) is used to send the data to a device driver.

The return value will be one the following:

- FAT_OK successfully received current file pointer
- FAT_NOT_OPEN file handle is not an open file
- FAT_ERROR a file system error was encountered
- FAT_EOF the end of the file was reached
- FAT_MSI_ERROR a transport layer error was returned

Example

Stream from a file to a device handle.

```
fat_context fatContext;
file_context_t FILE;
VOS_HANDLE hUart;
char filename = "SOURCE  TXT";
unsigned char status;
unsigned int size;

// Fat handle for the file.
status = fat_dirTableFind(fatContext, &FILE, filename);
if (status != FAT_OK)
{
    return -1;
}
else
{
    // Get the size of the file
    size = fat_dirEntrySize(&FILE);
    // Open the selected file for reading...
    status = fat_fileOpen(fatContext, &FILE, filename, FILE_MODE_READ);
    if (status == FAT_INVALID_FILE_TYPE)
    {
        return -2;
    }
    else
    {
        // Read and redirect output to UART interface
        status = fat_fileRead(&FILE, size, NULL, hUart, NULL);
        if (status != FAT_OK)
        {

```

```
        return -3;
    }
}

fat_fileClose(&FILE); // Close the file after reading.
}
```

Example

Read from a file into a buffer.

```
fat_context fatContext;
file_context_t FILE;
char filename = "BUFFER.TXT";
unsigned char status;
char *buffer;
unsigned int size;
unsigned int actual;

// Fat handle for the file.
status = fat_dirTableFind(fatContext, &FILE, filename);
if (status != FAT_OK)
{
    return -1;
}
else
{
    // Get the size of the file
    size = fat_dirEntrySize(&FILE);
    if (size < 1024)
    {
        buffer = malloc(size);
        if (buffer == NULL)
        {
            return -10;
        }
        // Open the selected file for reading...
        status = fat_fileOpen(fatContext, &FILE, filename, FILE_MODE_READ);
        if (status == FAT_INVALID_FILE_TYPE)
        {
            return -2;
        }
        else
        {
            // Read direct into buffer
            status = fat_fileRead(&FILE, size, buffer, NULL, &actual);
            if (status != FAT_OK)
            {
                return -3;
            }
        }

        fat_fileClose(&FILE); // Close the file after reading.
    }
}
```

Syntax

```
unsigned char fat_fileWrite(file_context_t *file_ctx, unsigned long length, char *buffer, VOS_HANDLE handle);
```

Description

Writes a file to a disk from either a buffer or streams to an other device handle.

Parameters

file_ctx
Pointer to file handle of file to write.

length
Number of bytes to write to the file.

buffer
Pointer to buffer which is used as a source of data for file. This buffer is only used if `hOutput` is NULL.

hOutput
Handle of device where data is received using [vos_dev_read\(\)](#). The `buffer` source is used if this is NULL.

bytes_read
Pointer to variable where the number of bytes read is written. If this is NULL then it is ignored.

Returns

If `hOutput` is NULL, the buffer will be used as a source of data sent to the file. Otherwise [vos_dev_read\(\)](#) is used to receive data from a device driver.

The return value will be one of the following:

- FAT_OK successfully received current file pointer
- FAT_READ_ONLY file is opened for read only
- FAT_NOT_OPEN file handle is not an open file
- FAT_ERROR a file system error was encountered
- FAT_EOF the end of the file was reached
- FAT_MSI_ERROR a transport layer error was returned

Example

See example in [Still Image Read Operations](#).

Example

Write from a buffer into a file.

```
fat_context fatContext;
file_context_t FILE;
char filename = "BUFFER.TXT";
unsigned char status;
char buffer[] = {1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0};
unsigned int size;
unsigned int actual;

// Fat handle for the file.
status = fat_dirTableFind(fatContext, &FILE, filename);
if (status != FAT_OK)
{
    return -1;
}
else
{
    // Get the size of the file
    size = sizeof(buffer);
    // Open the selected file for reading...
    status = fat_fileOpen(fatContext, &FILE, filename, FILE_MODE_WRITE);
    if (status == FAT_OK)
    {
        // Write from buffer direct into file
        status = fat_fileWrite(&FILE, size, buffer, NULL, &actual);
        if (status != FAT_OK)
        {
            return -3;
        }
    }

    fat_fileClose(&FILE); // Close the file after reading.
}
```

}

4.3 FTDI Libraries

A minimal set of C runtime library functions are provided, from general utility functions and macros to input/output functions and dynamic memory management functions.

C runtime libraries supported are :

- [ctype](#) Library
- [stdio](#) Library
- [stdlib](#) Library
- [string](#) Library
- [errno](#) Library

NOTE: Not all the standard library functions have been included.

4.3.1 ctype

The standard Ctype library functions for testing and mapping characters are implemented:

isalnum ,	Alphanumeric character
isalpha ,	An upper or lower case letter.
iscntrl ,	A control character.
isdigit ,	A number.
isgraph ,	Non-space printable character.
islower ,	A lower case character.
isprint ,	A printable character.
ispunct ,	A punctuation mark.
isspace ,	A white space character.
isupper ,	An upper case letter.
isxdigit	A hexadecimal digit.

Library Hierarchy

CType library hierarchy:

CType Library
errno Library (optional)
VOS Kernel

Library Files

ctype.a

Optional Libraries:

errno.a

Header Files

ctype.h

config.h (optional)

errno.h (optional)

4.3.1.1 isalnum

Syntax

```
int isalnum(int c);
```

Description

The isalnum function tests for any character for which isalpha or isdigit. The c argument is an int, the value of which the application shall ensure is representable as an unsigned char or equal to the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

Parameters

^c
c is a character to be tested.

Return Value

The isalnum() function returns non-zero for true and zero for false

4.3.1.2 isalpha

Syntax

```
int isalpha(int c);
```

Description

The isalpha function tests for any character for which isupper or islower, or any character that is one of a locale-specific set of alphabetic characters for which none of iscntrl, isdigit, ispunct, or isspace is true.

Parameters

^c
c is a character to be tested.

Return Value

The isalpha() function returns non-zero for true and zero for false

4.3.1.3 iscntrl

Syntax

```
int iscntrl(int c);
```

Description

The iscntrl function tests for any control character, a control character or non-printing character is a number in a set, that does not in itself represent a written symbol. All entries in the ASCII table below code 32 and 127 are of this kind.

Parameters

^c
c is a character to be tested.

Return Value

The iscntrl() function returns non-zero for true and zero for false. If the parameter is not in the domain of the function, the return result is undefined.

4.3.1.4 isdigit

Syntax

```
int isdigit(int c);
```

Description

The isdigit function tests for any decimal-digit character.

Parameters

^c
c is a character to be tested.

Return Value

The isdigit() function returns non-zero for true and zero for false. If the parameter is not in the domain of the function, the return result is undefined.

4.3.1.5 isgraph

Syntax

```
int isgraph(int c);
```

Description

The isgraph function tests for any printing character except space. All entries in the ASCII table above code 33 and below 126 are of this kind.

Parameters

^c
c is a character to be tested.

Return Value

The isgraph() function returns non-zero for true and zero for false. If the parameter is not in the domain of the function, the return result is undefined.

4.3.1.6 islower

Syntax

```
int islower(int c);
```

Description

The islower function tests for any character that is a lowercase letter or is one of a locale-specific set of characters for which none of isctrl, isdigit, ispunct, or isspace is true. In the "C" locale, islower returns true only for the lowercase letters.

Parameters

^c
c is a character to be tested.

Return Value

The islower() function returns non-zero for true and zero for false.

4.3.1.7 isprint

Syntax

```
int isprint(int c);
```


Description

The isprint function tests for any printing character including space.

Parameters

^c
c is a character to be tested.

Return Value

The isprint() function returns non-zero for true and zero for false.

4.3.1.8 ispunct

Syntax

```
int ispunct(int c);
```

Description

The ispunct function tests for any printing character that is one of a locale-specific set of punctuation characters for which neither is space nor isalnum is true.

Parameters

^c
c is a character to be tested.

Return Value

The ispunct() function returns non-zero for true and zero for false.

4.3.1.9 isspace

Syntax

```
int isspace(int c);
```

Description

The isspace function tests for any character that is a standard white-space character or is one of special characters for which isalnum is false, such as the standard white-space characters are the following: space (' '), form feed ('\f'), new-line ('\n'), carriage return ('\r'), horizontal tab ('\t'), and vertical tab ('\v').

Parameters

^c
c is a character to be tested.

Return Value

The isspace() function returns non-zero for true and zero for false.

4.3.1.10 isupper

Syntax

```
int isupper(int c);
```

Description

The isupper function tests for any character that is an uppercase letter or set of characters for which none of iscntrl, isdigit, ispunct, or isspace is true.

Parameters

^c
c is a character to be tested.

Return Value

The `isupper()` function returns non-zero for true and zero for false.

4.3.1.11 isxdigit

Syntax

```
int isxdigit(int c);
```

Description

The `isxdigit` function tests for any hexadecimal-digit character.

Parameters

^c
c is a character to be tested.

Return Value

The `isxdigit()` function returns non-zero for true and zero for false.

4.3.2 stdio

The standard ANSI STDIO library functions can be used for standard input/output operations. This included file operations, formatted output operation, character operations and direct input and output operations.

File operations work on the FAT file system. Therefore, before using these operations, a device must be set up and attached to the FAT file system driver. Once complete the `fsAttach` function should be called with the handle to the FAT driver to link the stdio library to the device.

Three standard streams are defined: [stdin](#) for input streaming; [stdout](#) for output streaming; and [stderr](#) which is also an output stream.

Initialisation

The formatted output function, [printf](#), requires that an output device is set up for the formatted output. This can be a UART or another interface.

The [stdioAttach](#) function must be called before using [printf](#) to attach to the device. This will initialise the stdio streams: [stdin](#), [stdout](#) and [stderr](#). It is not allowable to set the stdio streams to redirect to a file system.

The stdio streams may be redirected individually to separate devices.

[fsAttach](#) Attach file system handle to stdio library.

[stdioAttach](#) Attach interface to stdin, stdout and stderr streams.

[stdinAttach](#) Attach stdin stream.

[stdoutAttach](#) Attach stdout stream.

[stderrAttach](#) Attach stderr stream.

Operation

Formatted output.

[printf](#) Write formatted data to standard output.

File operations.

[fprintf](#) Write formatted data to a file.
[fopen](#) Open a file.
[fclose](#) Close a file.
[feof](#) Check for end-of-file.
[ftell](#) Get current position in file.

Direct I/O operations.

[fread](#) Read from a file.
[fwrite](#) Write to a file.

Predefined streams.

[stdout](#) Standard output stream
[stdin](#) Standard input stream
[stderr](#) Standard error stream

Library Hierarchy

STDIO library hierarchy:

STDIO Library		
errno Library (Optional)	File system (FAT File System Driver)	Stream (UART, SPI and FIFO Drivers or Layered Driver)
VOS Kernel		

Requirements

FAT and BOMS drivers must be included in a project if file system operations are to be performed, if they are not required then they may be omitted. If stdio streams

Example

1) Attaching to a FAT file system and opening a file.

```
// Here hFAT is a handle to an attached FAT file system driver
fsAttach (hFAT);
```

After attaching the FAT layered device driver file operations as shown below can be performed. If the fsAttach is not called or is successful, operation of the following calls are undefined.

```
fp = fopen ("test.txt", "w+");
```

2) Setting up stdio streams and writing formatted output.

```
// hUART is a handle to the UART interface
stdioAttach(hUART);
printf("%d bytes sent\n", (int)number);
```

Library Files

stdio.a

Optional Libraries:

errno.a

Optional File System Drivers:

FAT.a, BOMS.a

Optional Stream Drivers:

UART.a
FIFO.a
SPIMaster.a
SPISlave.a
USBHostFT232.a

Header Files

stdio.h
config.h (optional)
errno.h (optional)
Optional File System Headers:
FAT.h
Optional Stream Headers:
UART.h
FIFO.h
SPIMaster.h
SPISlave.h
USBHostFT232.h

4.3.2.1 fsAttach

Syntax

```
int fsAttach(VOS_HANDLE h)
```

Description

Attaches a file system driver to the stdio library. Currently only the FAT file system driver is supported. The file system is used to perform all operations containing a FILE pointer.

Parameters

h
Handle for FAT file system driver.

Return Value

Always returns zero.

4.3.2.2 stdioAttach

Syntax

```
int stdioAttach(VOS_HANDLE h)
```

Description

Attaches an I/O interface handle stdio streams, stdin, stdout and stderr. All three streams are attached to the same driver.

Cannot be used with a handle to the FAT file system driver or any driver that requires a structure to be passed in it's read() or write() handlers.

Parameters

h
Handle for stream operations.

Return Value

Always returns zero.

4.3.2.3 stdinAttach

Syntax

```
int stdinAttach(VOS_HANDLE h)
```

Description

Attaches an I/O interface handle stdio stream stdin. This stream is an input only.

Cannot be used with a handle to the FAT file system driver or any driver that requires a structure to be passed in it's read() or write() handlers.

Parameters

h
Handle for stdin input stream.

Return Value

Always returns zero.

4.3.2.4 stdoutAttach

Syntax

```
int stdoutAttach(VOS_HANDLE h)
```

Description

Attaches an I/O interface handle stdio stream stdout. This stream is an output only.

Cannot be used with a handle to the FAT file system driver or any driver that requires a structure to be passed in it's read() or write() handlers.

Parameters

h
Handle for stream output.

Return Value

Always returns zero.

4.3.2.5 stderrAttach

Syntax

```
int stderrAttach(VOS_HANDLE h)
```

Description

Attaches an I/O interface handle stdio streams stderr. This stream is an output only.

Cannot be used with a handle to the FAT file system driver or any driver that requires a structure to be passed in it's read() or write() handlers.

Parameters

h
Handle for stream output.

Return Value

Always returns zero.

4.3.2.6 printf

Syntax

```
int printf (const char * format, ... )
```

Description

Print formatted data to stdout. The output is formatted as a sequence of data specified in the format argument.

After the format parameter, the function expects the correct additional arguments as specified in the format.

Format	Description
%c	Character
%i	Signed decimal
%d	Signed decimal
%o	Signed octal
%s	Null terminated string
%u	Unsigned decimal
%x	Unsigned hexadecimal
%X	Unsigned hexadecimal
%p	Pointer
%r	ROM Pointer

Restrictions

- There is no support for flags -, +, (space), #, 0
- There is no width or precision field support in the printf command.
- There is no support for .precision of .number and .*
- There is no support for length h, l, L

Parameters

format
String that contains the text and formatting to be written to stdout.

Return Value

Not used.

Example

```
// UART is opened with required settings and passed to stdioAttach
// This UART will be used to send print characters
int k = 50;
char m=49;
char *newfile = "hello";
stdioAttach(hUart);
// constant value 100 assumed to be int
printf ("test %d \n", 100);
// variable k is of type int
printf ("test %d \n", k);
// variable k is of type int
printf ("test %c \n", k);
// constant value 'a' is assumed to be int
printf ("test %c\n", 'a');
```

```
// variable k is of type char
printf ("test %d \n", k);
// string newfile is a pointer to a char array
printf ("file %s\n", newfile);
// newfile is also a pointer
printf ("newfile at %p\n", newfile);
// &k is a pointer
printf ("k at %p\n", &k);
```

4.3.2.7 fopen

Syntax

```
FILE *fopen(const char * filename, const char * mode)
```

Description

Opens the file whose name is specified in the parameter filename and associates it with a stream that can be identified in future operations by the FILE object whose pointer is returned.

The operations that are allowed on the stream and how these are performed are defined by the mode parameter.

Parameters

filename
C string containing the name of the file to be opened. This parameter must follow the file name specifications of the running environment and can include a path if the system supports it.

mode
C string containing a file access modes

Return Value

If the file has been successfully opened the function will return a pointer to a FILE object that is used to identify the stream on all further operations involving it. Otherwise, a null pointer is returned.

4.3.2.8 fread

Syntax

```
size_t fread(void * ptr, size_t size, size_t count, FILE * stream)
```

Description

Read block of data from stream

Reads an array of count elements, each one with a size of size bytes, from the stream and stores them in the block of memory specified by ptr.

The position indicator of the stream is advanced by the total amount of bytes read.

The total amount of bytes read is size * count.

Parameters

ptr
Pointer to a block of memory with a minimum size of (size*count) bytes.

size
Size in bytes of each element to be read.

count
Number of elements, each one with a size of size bytes.

stream
Pointer to a FILE objects that specifies an Input stream.

Return Value

The total number of elements successfully read is returned as a `size_t` object, which is an integral data type.

If this number differs from the count parameter, either an error occurred or the End Of File was reached.

Example

```
// FAT file system is opened with required settings and passed to fsAttach
// This file system will be used to receive a buffer from a file
char buf[256];
FILE *f;
fsAttach(hFAT);
f = fopen("TEST.TXT", "r");
if (f)
{
    fread(buf, 256, 1, f);
    fclose(f);
}
```

4.3.2.9 fwrite

Syntax

```
size_t fwrite(const void *ptr, size_t size, size_t count, FILE * stream)
```

Description

Writes an array of count elements, each one with a size of size bytes, from the block of memory pointed by ptr to the current position in the stream.

The position indicator of the stream is advanced by the total number of bytes written.

The underlying type of the objects pointed by both the source and destination pointers are irrelevant for this function (i.e. The result is a binary copy of the data)

The total amount of bytes written is `size * count`.

Parameters

ptr
Pointer to the array of elements to be written .

size
Size in bytes of each element to be written.

count
Number of elements, each one with a size of size bytes.

stream
Pointer to a FILE object that specifies an Output stream.

Return Value

The total number of elements successfully written is returned as a `size_t` object, which is an integer data type.

If this number differs from the count parameter, it indicates an error.

Example

```
// FAT file system is opened with required settings and passed to fsAttach
// This file system will be used to send an output buffer to a file
char buf[256];
int i;
FILE *f;
fsAttach(hFAT);
for (i=0; i < 256; i++)
```

```
{
    buf[i] = i & 0xff;
}
f = fopen("TEST.TXT", "r");
if (f)
{
    fwrite(buf, 256, 1, f);
    fclose(f);
}
```

4.3.2.10 fclose

Syntax

```
int fclose(FILE * stream)
```

Description

Closes the file associated with the stream and disassociates it.

All internal buffers associated with the stream are flushed: the content of any unwritten buffer is written and the content of any unread buffer is discarded.

Even if the call fails, the stream passed as parameter will no longer be associated with the file.

Parameters

stream
Pointer to a FILE object that specifies the stream to be closed.

Return Value

If the stream is successfully closed, a zero value is returned.

On failure, negative number is returned.

4.3.2.11 feof

Syntax

```
int feof(FILE * stream)
```

Description

Check End-of-File indicator.

Checks whether the End-of-File indicator associated with stream is set, returning a value different from zero if it is.

This indicator is generally set by a previous operation on the stream that reached the End-of-File.

Further operations on the stream once the End-of-File has been reached will fail until either rewind, fseek or fsetpos is successfully called to set the position indicator to a new value.

Parameters

stream
Pointer to a FILE object that identifies the stream.

Return Value

A non-zero value is returned in the case that the End-of-File indicator associated with the stream is set.

Otherwise, a zero value is returned.

4.3.2.12 ftell

Syntax

```
int ftell(FILE *)
```

Description

Get current position in stream.

For binary streams, the value returned corresponds to the number of bytes from the beginning of the file.

For text streams, the value is not guaranteed to be the exact number of bytes from the beginning of the file. but the value returned can still be used to restore the position indicator to this position using fseek.

Parameters

stream
Pointer to a FILE object that identifies the stream.

Return Value

On success, the current value of the position indicator is returned.

If an error occurs, -1 is returned, and the global variable errno is set to a positive value. This value can be interpreted by perror.

4.3.2.13 fprintf

Syntax

```
int fprintf (FILE *stream, const char * format, ... )
```

Description

Print formatted data to an open file handle. The output is formatted as a sequence of data specified in the format argument. Refer to the [printf](#) topic for details on formatting and restrictions.

Parameters

stream
File handle for a stream output. May be a file or one of the standard streams: stdin, stdout or stderr.

format
String that contains the text and formatting to be written to stdout.

Return Value

Not used.

Example

```
// FAT file system is opened with required settings and passed to fsAttach
// This file system will be used to send print characters
char *newfile = "hello";
FILE *f;
fsAttach(hFAT);
f = fopen("TEST.TXT", "w");
if (f)
{
    // constant value 100 assumed to be int
    fprintf (f, "test %d \n", 100);
    // newfile is a pointer to a null terminated string
    fprintf (f, "test %s \n", newfile);
    fclose(f);
}
```

4.3.2.14 stdout

Syntax

```
FILE *stdout
```

Description

Default output stream. This handle is used for the output of [printf](#) and is set by the [stdioAttach](#) or [stdoutAttach](#) functions. The definition may be used as a handle for output for [fwrite](#) but may not redirect to a file.

Parameters

N/A

Return Value

N/A

4.3.2.15 stdin

Syntax

```
FILE *stdin
```

Description

Default input stream. This handle is set by the [stdioAttach](#) or [stdinAttach](#) functions. The definition may be used as a handle for input for [fread](#) but may not redirect from a file.

Parameters

N/A

Return Value

N/A

4.3.2.16 stderr

Syntax

```
FILE *stderr
```

Description

Error output stream. This handle is set by the [stdioAttach](#) or [stderrAttach](#) functions. The definition may be used as a handle for output for [fwrite](#) but may not redirect to a file.

Parameters

N/A

Return Value

N/A

4.3.3 stdlib

The standard ANSI STDIO library functions are of general utility.

abs	Get the absolute value of an integer.
strtol	Convert a string to a long value in a specified number base.
atol	Convert a string to a long value.

atoi	Convert a string to an integer
malloc	Allocate dynamic memory.
calloc	Allocate and clear to zero a block of dynamic memory.
free	Free a block of dynamic memory.

Library Hierarchy

STDLIB library hierarchy:

STDLIB Library
errno Library (Optional)
VOS Kernel

Library Files

stdlib.a

Optional Libraries:

errno.a

Header Files

stdlib.h

config.h (optional)

errno.h (optional)

4.3.3.1 abs

Syntax

```
int abs(int val);
```

Description

The abs computes the absolute value of an integer val. If the result cannot be represented, the behaviour is undefined. The absolute value of the most negative number cannot be represented in two's complement.

Parameters

val
Integer value whose absolute value needs to be calculated.

Return Value

The abs() function returns the absolute value.

4.3.3.2 strtol

Syntax

```
long strtol(const char *nptr, char **endptr, int base);
```

Description

The strtol function converts the initial portion of the string pointed to by nptr to long. First, it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by the isspace function), a subject sequence resembling an integer represented in some radix determined by the value of base, and a final string of one or more unrecognized characters, including the terminating null character of the input string. Then, they attempt to convert the subject sequence to an integer, and return the result.

Parameters

- nptr**
Points to a character string for strtol() to convert.
- endptr**
Is a result parameter that, if not NULL
- base**
Is the base of the string, a value between 0 and 36.

Return Value

Return the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, ERANGE is stored in errno.

4.3.3.3 atol

Syntax

```
long atol(const char *nptr);
```

Description

The atol function converts the initial portion of the string pointed to by nptr to long, representation

Parameters

- nptr**
C string containing the representation of an integral number.

Return Value

Functions return the converted value. If no valid conversion could be performed, a zero value is returned.

4.3.3.4 atoi

Syntax

```
int atoi(const char *nptr);
```

Description

The atoi, function converts the initial portion of the string pointed to by nptr to int, representation.

Parameters

- nptr**
C string containing the representation of an integral number.

Return Value

On success, the function returns the converted integral number as an int value.
If no valid conversion could be performed, a zero value is returned.

4.3.3.5 malloc

Syntax

```
void *malloc (size_t size);
```

Description

The malloc function allocates space for an object whose size is specified by size and whose value is indeterminate.

Parameters

size
Size of the memory block, in bytes.

Return Value

A pointer to the memory block allocated by the function. If the function failed to allocate the requested block of memory, a NULL pointer is returned.

4.3.3.6 calloc

Syntax

```
void *calloc(size_t nmemb, size_t size);
```

Description

The calloc function allocates space for an array of nmemb objects, each of whose size is size. The space is initialized to all bits zero.

Parameters

nmemb
Number of elements to be allocated.

size
Size of elements.

Return Value

A pointer to the memory block allocated by the function. If the function failed to allocate the requested block of memory, a NULL pointer is returned.

4.3.3.7 free

Syntax

```
void free(void *ptr);
```

Description

The free function causes the space pointed to by ptr to be deallocated that is, made available for further allocation. If ptr is a null pointer, no action occurs. otherwise, if the argument does not match a pointer earlier returned by the calloc, malloc, or, or if the space has been deallocated by a call to free or realloc, the behaviour is undefined

Parameters

ptr
Pointer to a memory block previously allocated with malloc or calloc to be deallocated.

Return Value

The free function returns no value.

4.3.4 string

The standard ANSI String library functions are useful for manipulating strings and RAM memory.

memcpy	Copy a block of memory.
memset	Set a block of memory to a value.
strcmp	Compare one string with another.
strncmp	Compare a set number of characters in one string to another string.
strcpy	Copy one string to another.

[strcpy](#) Copy a set number of characters from one string to another.
[strcat](#) Concatenate one string onto another.
[strlen](#) Obtain the length of a string.

Library Hierarchy

String library hierarchy:

String Library
errno Library (Optional)
VOS Kernel

Library Files

string.a

Optional Libraries:

errno.a

Header Files

stdlib.h

config.h (optional)

errno.h (optional)

4.3.4.1 memcpy

Syntax

```
void * memcpy (void * destination, const void * source, size_t num );
```

Description

This function is used to copy a block of memory. It copies num of bytes from the location pointed by source to the memory block pointed by destination.

The underlying type of the objects pointed by both the source and destination pointers are irrelevant for this function (i.e. The result is a binary copy of the data).

The function does not check for any terminating null character in source - it always copies exactly num bytes.

To avoid overflows, the size of the arrays pointed by both the destination and source parameters shall be at least num bytes, and should not overlap.

Parameters

destination

Pointer to the destination array where the content is to be copied, type-cast to a pointer of type void*.

source

Pointer to the source of data to be copied, type-cast to a pointer of type void*.

num

Number of bytes to copy.

Return Value

Destination pointer where the source content is copied to is returned.

4.3.4.2 memset

Syntax

```
void * memset ( void * ptr, int value, size_t num );
```

Description

Fill block of memory with given value.

Sets the first num bytes of the block of memory pointed by ptr to the specified value (interpreted as an unsigned char).

Parameters

ptr
Pointer to the block of memory to fill.

value
Value to be set. The value is passed as an int, but the function fills the block of memory using the unsigned char conversion of this value.

num
Number of bytes to be set to the value.

Return Value

Pointer where block of memory is filled with given value is returned.

4.3.4.3 strcmp

Syntax

```
int strcmp ( const char * str1, const char * str2 );
```

Description

Compares the C string str1 to the C string str2.

This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until a terminating null-character is reached.

Parameters

str1
String 1 to be compared.

str2
String 2 to be compared.

Return Value

Returns an integral value indicating the relationship between the strings:

Zero indicates that both strings are equal.

A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2;

A value less than zero indicates the opposite.

4.3.4.4 strncmp

Syntax

```
int strncmp ( const char * str1, const char * str2, size_t num );
```

Description

Compares up to num characters of the C string str1 to those of the C string str2.

This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ, until a terminating null-character is reached, or until num characters match in both strings, whichever happens first.

Parameters

`str1`
String 1 to be compared.

`str2`
String 2 to be compared.

`num`
Maximum number of characters to compare.

Return Value

Returns an integral value indicating the relationship between the strings:

Zero indicates that both strings are equal.

A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2;

A value less than zero indicates the opposite.

4.3.4.5 strcpy

Syntax

```
char * strcpy ( char * destination, const char * source );
```

Description

Copies the C string pointed by source into the array pointed by destination, including the terminating null character.

To avoid overflows, the size of the array pointed by destination shall be long enough to contain the same C string as source (including the terminating null character), and should not overlap in memory with source.

Parameters

`destination`
Pointer to the destination array where the content is to be copied.

`source`
C string to be copied.

Return Value

Destination pointer where the source string is copied is returned.

4.3.4.6 strncpy

Syntax

```
char * strncpy ( char * destination, const char * source, size_t num );
```

Description

Copies the first num characters of source to destination.

If the end of the source C string (which is signaled by a null-character) is found before num characters have been copied, destination is padded with zeros until a total of num characters have been written to it.

No null-character is implicitly appended to the end of destination, so destination will only be null-terminated if the length of the C string in source is less than num.

Parameters

destination
Pointer to the destination array where the content is to be copied.

source
C string to be copied.

num
Maximum number of characters to be copied from source.

Return Value

Destination pointer where two string are copied is returned.

4.3.4.7 strcat

Syntax

```
char * strcat ( char * destination, const char * source );
```

Description

This function is used to concatenate strings

Appends a copy of the source string to the destination string.

The terminating null character in destination is overwritten by the first character of source, and a new null-character is appended at the end of the new string formed by the concatenation of both in destination.

Parameters

destination
Pointer to the destination array, which should contain a C string, and be large enough to contain the concatenated resulting string.

source
C string to be appended. This should not overlap destination.

Return Value

Destination pointer where both the string are concatenated is returned.

4.3.4.8 strlen

Syntax

```
size_t strlen ( const char * str );
```

Description

This function is used to get the length of the str. A C string is as long as the amount of characters between the beginning of the string and the terminating null character.

Parameters

str
C string.

Return Value

The length of str.

4.3.5 errno

The standard ANSI errno library functions are useful for retrieving error codes.

[errno](#) Get last error number.

Library Hierarchy

Errno library hierarchy:

errno Library
VOS Kernel

Library Files

errno.a

Header Files

errno.h

4.3.5.1 errno

Syntax

```
int errno
```

Description

errno is a macro that returns the last error number generated by a library operation.

Parameters

Return Value

The errno macro returns the last error number generated.

5 Sample Firmware Applications

5.1 Sample Firmware Overview

A selection of firmware samples are included in the Vinculum II Toolchain. These are designed to demonstrate the capabilities of the Kernel, Drivers and Libraries.

- [General Samples](#) show small applications which demonstrate a feature of the Kernel, Drivers or Libraries. They are simple, minimal applications that typically perform one small function.
- [USB Host Samples](#) feature the USB Host Controller hardware to demonstrate the use of either the USBHost driver or drivers layered on the USBHost driver.
- [USB Slave Samples](#) show the use of the USBSlave driver and it's associated layered drivers.
- [Firmware Samples](#) are complete applications which may be adapted. These are mainly versions of the VNC1L firmware.

The samples are intended as a guide to writing firmware for the VNC2 and is provided as illustration only. As such it cannot be guaranteed to function correctly under all circumstances nor will support for the code be provided.

5.2 General Samples

The following samples are available in the samples General folder. The table below shows the features demonstrated in each sample.

	Kernel	Drivers	Libraries
Template Sample	Threads, IOMux	UART, GPIO	N/A
GPIOKitt Sample	Threads, IOMux, Delay	GPIO	N/A
PWMBreathe Sample	Threads, IOMux	PWM	N/A
Philosophers Sample	Threads, IOMux, Semaphores, Mutexes	GPIO	N/A
Runtime Sample	Threads, IOMux	UART	stdio
HelloWorld Sample	Threads, IOMux, Delay	GPIO, FAT, BOMS, USBHost	stdio, string

5.2.1 Template Sample

Template sample hierarchy:

Template Application	
UART Driver	GPIO Driver
VOS Kernel	

Description

This sample is a simple project which writes a counter to the UART interface and flashes the LEDs on a V2EVAL board. It is designed to have as few dependencies on drivers and libraries but still provide meaningful output to indicate activity.

Function

The output to the UART is sent at 9600 baud, 8 bits, 1 stop bit, no parity with CTS/RTS flow control enabled. The output is as follows:

```
Hello 00000
Hello 00001
Hello 00002
Hello 00003
Hello 00004
...
```

The GPIO Driver interface is used to output and illuminate the LEDs. The 4 LEDs on the V2EVAL board are flashed in sequence as by an 8 bit counter - a 'zero' in the bit position on the GPIO port will light the LED. The mapping of the counter to the LEDs on the board is dictated by the package type of the VNC2.

32 pin - LED3 bit 1, LED4 bit 2.

48 pin - LED3 bit 1, LED4 bit 2, LED5 bit 5, LED6 bit 4.

64 pin - LED3 bit 1, LED4 bit 2, LED5 bit 5, LED6 bit 6.

Comments

The LEDs will stop flashing if the UART interface is blocked on flow control.

Kernel Functions

[vos_init\(\)](#)

[vos_set_clock_frequency\(\)](#)

[vos_get_package_type\(\)](#)

[vos_iomux_define_input\(\)](#) and [vos_iomux_define_output\(\)](#)

[vos_create_thread\(\)](#)

[vos_start_scheduler\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_read\(\)](#)

[vos_dev_write\(\)](#)

[vos_dev_ioctl\(\)](#)

Driver Functions

[gpio_init\(\)](#)

[VOS_IOCTL_GPIO_SET_MASK](#)

[uart_init\(\)](#)

[VOS_IOCTL_COMMON_ENABLE_DMA](#)

[VOS_IOCTL_UART_SET_BAUD_RATE](#)

[VOS_IOCTL_UART_SET_FLOW_CONTROL](#)

[VOS_IOCTL_UART_SET_DATA_BITS](#)

[VOS_IOCTL_UART_SET_STOP_BITS](#)

[VOS_IOCTL_UART_SET_PARITY](#)

[UART Read and Write Operations](#)

Library Functions

N/A

5.2.2 GPIOKitt Sample

GPIOKitt sample hierarchy:

GPIO Kitt Application
GPIO Driver
VOS Kernel

Description

The GPIOKitt sample is an LED pattern flashing demonstration which maps up-to 4 GPIO pins to the LEDs on a V2EVAL board.

Function

The GPIO Driver interface is used to output and illuminate the LEDs. The 4 LEDs on the V2EVAL board are flashed in sequence by a bit oscillating within an 8 bit register - a 'zero' in the bit position on the GPIO port will light the LED. The mapping of the counter to the LEDs on the board is dictated by the package type of the VNC2.

32 pin - LED3 bit 1, LED4 bit 2.

48 pin - LED3 bit 1, LED4 bit 2, LED5 bit 5, LED6 bit 4.

64 pin - LED3 bit 1, LED4 bit 2, LED5 bit 5, LED6 bit 6.

Comments

The 4 LEDs on the V2EVAL board should flash in sync off and on, with a constant period. For 48 and 64 pin packages there should be three LEDs lit at all times.

Kernel Functions

[vos_init\(\)](#)

[vos_set_clock_frequency\(\)](#)

[vos_get_package_type\(\)](#)

[vos_iomux_define_input\(\) and vos_iomux_define_output\(\)](#)

[vos_create_thread\(\)](#)

[vos_start_scheduler\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_write\(\)](#)

[vos_dev_ioctl\(\)](#)

Driver Functions

[gpio_init\(\)](#)

[VOS_IOCTL_GPIO_SET_MASK](#)

Library Functions

N/A

5.2.3 PWMBreathe Sample

PWMBreathe sample hierarchy:

PWMBreathe Application
<u>PWM Driver</u>
<u>VOS Kernel</u>

Description

The PWMBreathe sample shows the use of the PWM outputs on the VNC2.

Function

The PWM Driver interface is used to illuminate the LEDs. The 4 LEDs on the V2EVAL board are driven with a varying duty cycle by the PWM output. This causes them to 'breathe' between fully-on and fully-off with a period of around 6 seconds.

The mapping of the PWM outputs to the LEDs on the board is dictated by the package type of the VNC2.

32 pin - LED3 to PWM 5 1, LED4 to PWM 6.

48 pin - LED3 to PWM 6, LED4 to PWM5, LED5 to PWM5, LED6 to PWM6.

64 pin - LED3 to PWM 6, LED4 to PWM5, LED5 to PWM5, LED6 to PWM6.

Comments

The 4 LEDs on the V2EVAL board will 'breathe'. LED4 and LED6 are paired and will show the same brightness, LED3 and LED5 will be similarly paired in brightness but out of sync with the other pair of LEDs.

Kernel Functions

[vos_init\(\)](#)
[vos_set_clock_frequency\(\)](#)
[vos_get_package_type\(\)](#)
[vos_iomux_define_input\(\)](#) and [vos_iomux_define_output\(\)](#)
[vos_iocell_set_config\(\)](#)
[vos_create_thread\(\)](#)
[vos_start_scheduler\(\)](#)
[vos_dev_open\(\)](#)
[vos_dev_ioctl\(\)](#)

Driver Functions

[gpio_init\(\)](#)
[VOS_IOCTL_GPIO_SET_MASK](#)
[pwm_init\(\)](#)
[VOS_IOCTL_PWM_SET_PRESCALER_VALUE](#)
[VOS_IOCTL_PWM_SET_COUNTER_VALUE](#)
[VOS_IOCTL_PWM_SET_COMPARATOR_VALUE](#)
[VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES](#)
[VOS_IOCTL_PWM_SET_INITIAL_STATE](#)
[VOS_IOCTL_PWM_SET_NUMBER_OF_CYCLES](#)
[VOS_IOCTL_PWM_ENABLE_INTERRUPT](#)
[VOS_IOCTL_PWM_ENABLE_OUTPUT](#)
[VOS_IOCTL_PWM_WAIT_ON_COMPLETE](#)
[VOS_IOCTL_PWM_DISABLE_OUTPUT](#)

Library Functions

N/A

5.2.4 Philosophers Sample

Philosophers sample hierarchy:

Philosophers Application
GPIO Driver
VOS Kernel

Description

The purpose of this sample is to show how semaphores and mutexes may be employed to synchronise multiple threads and prevent deadlock while allocating resources.

Function

This is an implementation of the classic synchronisation problem in operating system theory http://en.wikipedia.org/wiki/Dining_philosophers_problem. Five independent threads are started which each have access to 2 semaphores from a total of 5 semaphores available. The threads signal and wait on these semaphores. An LED is lit if a thread holds less than 2 semaphores and off if it holds 2 semaphores.

Control over the setting of the LED status variable "gpioData", a resource shared between multiple threads, is achieved using the leds_lock mutex. This will prevent more than one update of the variable's value at any one time. Effectively stopping another thread from reading from or writing to the variable while it is being modified by another thread.

Comments

There are 5 threads which flash LEDs, however, there are only 4 LEDs on the V2EVAL board.

Kernel Functions

[vos_init\(\)](#)
[vos_set_clock_frequency\(\)](#)
[vos_get_package_type\(\)](#)
[vos_iomux_define_input\(\)](#) and [vos_iomux_define_output\(\)](#)
[vos_create_thread\(\)](#)
[vos_start_scheduler\(\)](#)
[vos_dev_open\(\)](#)
[vos_dev_write\(\)](#)
[vos_dev_ioctl\(\)](#)
[vos_init_semaphore\(\)](#)
[vos_wait_semaphore\(\)](#)
[vos_signal_semaphore\(\)](#)
[vos_init_mutex\(\)](#)
[vos_lock_mutex\(\)](#)
[vos_unlock_mutex\(\)](#)
[vos_delay_msecs\(\)](#)

Driver Functions

[gpio_init\(\)](#)
[VOS_IOCTL_GPIO_SET_MASK](#)

Library Functions

N/A

5.2.5 Runtime Sample

Runtime sample hierarchy:

Runtime Application
stdio
UART, SPI and FIFO Drivers
VOS Kernel

Description

Demonstrates the use of the [stdio](#) library. Using [printf](#) and [fwrite](#) to send output to the UART. The

use of the [stdout](#) stream is also demonstrated.

Function

The output to the UART will be as follows:

```
No format
Decimal signed -46
Decimal unsigned 4294967250
Decimal signed -48 unsigned 4294967249
Hex caps FEED lower face
Character A
String here and here!
Pointers 1345 13a2 197
Escape d-quote " s-quote ' tab ^I bslash \
--HELLO--
Month 1 is January
Month 2 is February
Month 3 is March
Month 4 is April
Month 5 is May
Month 6 is June
Month 7 is July
Month 8 is August
Month 9 is September
Month 10 is October
Month 11 is November
Month 12 is December
```

Comments

Variable argument lists sent to the [printf](#) command.

Kernel Functions

[vos_init\(\)](#)

[vos_set_clock_frequency\(\)](#)

[vos_get_package_type\(\)](#)

[vos_iomux_define_input\(\)](#) and [vos_iomux_define_output\(\)](#)

[vos_create_thread\(\)](#)

[vos_start_scheduler\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_ioctl\(\)](#)

Driver Functions

[gpio_init\(\)](#)

[VOS_IOCTL_GPIO_SET_MASK](#)

[uart_init\(\)](#)

[VOS_IOCTL_COMMON_ENABLE_DMA](#)

[VOS_IOCTL_UART_SET_BAUD_RATE](#)

[VOS_IOCTL_UART_SET_FLOW_CONTROL](#)

[VOS_IOCTL_UART_SET_DATA_BITS](#)

[VOS_IOCTL_UART_SET_STOP_BITS](#)

[VOS_IOCTL_UART_SET_PARITY](#)

Library Functions

[stdioAttach](#)

[printf](#)

[fwrite](#)

[fprintf](#)

5.2.6 HelloWorld Sample

HelloWorld sample hierarchy:

HelloWorld Application		
string	stdio	GPIO Driver
	FAT File System	
	BOMS Class Driver	
	USB Host Driver	
VOS Kernel		

Description

Demonstrates the use of the [stdio](#) library. sing [fopen](#), [fwrite](#) and [fclose](#) to append to a file. The [string](#) library is also called to obtain the length of a string with [strlen](#).

Function

A flash disk with a FAT file system is connected to USB port 2. It is detected and attached to the BOMS driver and FAT File System driver. This is then attached to the [stdio](#) library using [fsAttach](#).

The phrase "Hello World! \n" will be appended to the file TEST.TXT on the disk.

Comments

The file is opened in append mode. LEDs are used to signal progress through the code.

Kernel Functions

[vos_init\(\)](#)

[vos_set_clock_frequency\(\)](#)

[vos_get_package_type\(\)](#)

[vos_create_thread\(\)](#)

[vos_start_scheduler\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_close\(\)](#)

[vos_dev_ioctl\(\)](#)

[vos_delay_msecs\(\)](#)

Driver Functions

[gpio_init\(\)](#)

[VOS_IOCTL_GPIO_SET_MASK](#)

[usbhost_init\(\)](#)

[VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#)

[VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS](#)

[boms_init\(\)](#)

[BOMS_MSI_IOCTL_BOMS_ATTACH](#)

[BOMS MSI_IOCTL_BOMS_DETACH](#)

[fatdrv_init\(\)](#)

[FAT_IOCTL_FS_ATTACH](#)

[FAT_IOCTL_FS_DETACH](#)

Library Functions

[fsAttach](#)

[fopen](#)

[fwrite](#)

[fclose](#)

[strlen](#)

5.3 USB Host Samples

The following samples are available in the samples USBHost folder. The table below shows the features demonstrated in each sample.

	Kernel	Drivers	Libraries
StillImageApp Sample	Threads	StillImage, FAT, BOMS, USBHost, GPIO	N/A
USBHostGeneric Sample	Threads, Delay, Drivers	USBHost	N/A
USBHostGPSLogger Sample	Threads, Delay	USBHost, FT232USBHost, FAT, BOMS	N/A
USBHostHID Sample	Threads, Delay, Semaphores, Mutexes	USBHost (Interrupt Endpoints), UART	string
USBHostHID2 Sample	Threads, Delay, Multiple Semaphores	USBHost (Interrupt Endpoints, Non-blocking reads), UART	string
USBMic Sample	Threads, Delay	USBHost (Isochronous Endpoints, Finding Next Endpoint)	string

5.3.1 StillImageApp Sample

StillImageApp sample hierarchy:

StillImageApp Application		
<u>GPIO Driver</u>	<u>FAT File System</u>	<u>Still Image Class Driver</u>
	<u>BOMS Class Driver</u>	
	<u>USB Host Driver</u>	
<u>VOS Kernel</u>		

Description

The Still Image App will communicate with a Still Image Class Camera (supporting PIMA command set). If it supports the InitiateCapture method then it can take a picture on the camera and then download it to a file on a flash disk.

Function

A flash disk with a FAT file system is connected to USB port 2. A suitable digital camera to USB port 1. The firmware will initialise both ports and attach the file system to the flash disk. It will then attach to the digital camera and send an InitiateCapture command. The process of taking a picture can take several seconds depending on the model of camera. It will then transfer the image taken to a file on the flash disk.

LEDs are used to indicate progress while taking pictures or transferring them from the camera to the disk. A delay of approximately 5 seconds is added after a picture is taken and written to the disk. The sample will disconnect from both the camera and the disk.

Comments

The sample has been tested with the Canon Powershot Canon SX 110 IS.

If desired, the macro TAKE_PICTURE may be removed to alter the functionality of the sample. If it is not defined then the sample will copy the first object on the devices storage to the disk and delete the object rather than taking a picture - this mode is intended to be used if the camera does not support the InitiateCapture command.

Kernel Functions

[vos_init\(\)](#)

[vos_set_clock_frequency\(\)](#)

[vos_get_package_type\(\)](#)

[vos_create_thread\(\)](#)

[vos_start_scheduler\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_close\(\)](#)

[vos_dev_write\(\)](#)

[vos_dev_ioctl\(\)](#)

[vos_delay_msecs\(\)](#)

Driver Functions

[gpio_init\(\)](#)

[VOS_IOCTL_GPIO_SET_MASK](#)

[usbhost_init\(\)](#)

[VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#)

[VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS](#)

[stillimage_init\(\)](#)

[STILLIMAGE_IOCTL_ATTACH](#)

[STILLIMAGE_IOCTL_GET_FIRST_OBJECT](#)

[STILLIMAGE_IOCTL_INITIATE_CAPTURE](#)

[STILLIMAGE_IOCTL_GET_OBJECT_INFO](#)

[STILLIMAGE_IOCTL_OPEN_OBJECT](#)

[STILLIMAGE_IOCTL_CLOSE_OBJECT](#)

[STILLIMAGE_IOCTL_DELETE_OBJECT](#)

[boms_init\(\)](#)

[BOMS_MSI_IOCTL_BOMS_ATTACH](#)

[BOMS_MSI_IOCTL_BOMS_DETACH](#)

[fat_init\(\)](#)

[fat_open\(\)](#)

[fat_close\(\)](#)

[fat_fileOpen\(\)](#)

[fat_fileClose\(\)](#)

[fat_fileWrite\(\)](#)

Library Functions

N/A

5.3.2 USBHostGeneric Sample

USBHostGeneric sample hierarchy:

USBHostGeneric Application		
USBHostGeneric Driver	UART Driver	string
USB Host Driver		
VOS Kernel		

Description

The USBHostGeneric sample demonstrates creating and using a layered driver on top of the USBHost.

Function

There are two parts to USBHostGeneric sample.

The first is the layered driver, this is attached to a device on the USBHost Driver which can then be opened using [vos_dev_open\(\)](#). Read and write operations can be sent through [vos_dev_read\(\)](#) and [vos_dev_write\(\)](#) to the driver from an application.

The second is the application which opens the USBHostGeneric driver and receives data from it. This data is sent to the UART interface.

Comments

The driver can be layered on top of any other driver to either provide abstraction or additional functionality to an interface. It is possible to have the driver not layered where it may provide some processing function if that is required.

The UART interface relies on the default settings of the UART driver of 9600 baud, 8 bits, 1 stop bit, no parity.

Kernel Functions

[vos_init\(\)](#)

[vos_set_clock_frequency\(\)](#)

[vos_get_package_type\(\)](#)

[vos_create_thread\(\)](#)

[vos_start_scheduler\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_close\(\)](#)

[vos_dev_write\(\)](#)

[vos_dev_read\(\)](#)

[vos_dev_ioctl\(\)](#)

Driver Functions

[uart_init\(\)](#)

[VOS_IOCTL_COMMON_ENABLE_DMA](#)

[UART Read and Write Operations](#)

[usbhost_init\(\)](#)

[USB Host General Transfer Block](#)

[VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_BULK_OUT_ENDPOINT_HANDLE](#)

Library Functions

[strlen](#)

5.3.3 USBHostGPSLogger Sample

USBHostGPSLogger sample hierarchy:

USBHostGPSLogger Application		
FT232 USB Host Device Driver	FAT File System	string
	BOMS Class Driver	
USB Host Driver		
VOS Kernel		

Description

The USBHostGPSLogger sample demonstrates reading data from an FT232-style device on the USB Host controller and writing it to a flash disk.

Function

The application finds a flash disk on USB port 2 using the [VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS](#) method in the USBHost Driver. It will then attach that to a [BOMS Class Driver](#) and the [FAT File System API](#).

An FT232 (or equivalent) device is found on USB port 1 using [VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_VID_PID](#). This is attached to the [FT232 USB Host Device Driver](#). The baud rate on the FT232 is set to 4800 baud in line with the standard output of GPS units.

A file called "LOG.TXT" is opened and the data from the FT232 will be appended to the file.

Comments

A Rayming TripNav TN-200 GPS was tested with this sample. It utilises an FT232 device to convert the serial output of the GPS to USB. The USBHostFT232 driver was used to interface with the FT232device. The FT232 device on the GPS unit runs at 4800 baud, 8 bits, 1 stop bit, no parity.

Kernel Functions

[vos_init\(\)](#)

[vos_set_clock_frequency\(\)](#)

[vos_get_package_type\(\)](#)

[vos_create_thread\(\)](#)

[vos_start_scheduler\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_close\(\)](#)

[vos_dev_write\(\)](#)

[vos_dev_read\(\)](#)

[vos_dev_ioctl\(\)](#)

Driver Functions

[usbhost_init\(\)](#)

[VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#)

[VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS](#)

[VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_VID_PID](#)

[usbHostFt232_init\(\)](#)

[VOS_IOCTL_USBHOSTFT232_ATTACH](#)

[VOS_IOCTL_UART_SET_BAUD_RATE](#)

[boms_init\(\)](#)

[BOMS_MSI_IOCTL_BOMS_ATTACH](#)

[BOMS_MSI_IOCTL_BOMS_DETACH](#)

[fat_init\(\)](#)

[fat_open\(\)](#)

[fat_close\(\)](#)

[fat_fileOpen\(\)](#)

[fat_fileClose\(\)](#)

[fat_fileWrite\(\)](#)

Library Functions

N/A

5.3.4 USBHostHID Sample

USBHostHID sample hierarchy:

USBHostHID Application		
USB Host Driver	UART Driver	string
VOS Kernel		

Description

The USBHostHID sample demonstrates reading from an interrupt endpoint.

Function

The USBHost driver is connected to USB port 1. A search is made for a HID device by searching using [VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_VID_PID](#). Once the HID device is found then it will poll that device and send any data received to the UART interface. The interrupt endpoint on the device is used.

Comments

The sample is pre-programmed to find a Logitech Wingman Action Pad P/N 863188-0000. This has a USB VID of 0x046d and a PID of 0xc20b. Changing the following lines to use other devices:

```
// find VID/PID of Logitech Wingman Action device
hc_ioctVidPid.vid = 0x046d;
hc_ioctVidPid.pid = 0xc20b;
```

The device will always send a 5 byte status packet when a status change on the buttons or joystick are detected.

Kernel Functions

[vos_init\(\)](#)
[vos_set_clock_frequency\(\)](#)
[vos_get_package_type\(\)](#)
[vos_create_thread\(\)](#)
[vos_start_scheduler\(\)](#)
[vos_init_semaphore\(\)](#)
[vos_wait_semaphore\(\)](#)
[vos_dev_open\(\)](#)
[vos_dev_close\(\)](#)
[vos_dev_write\(\)](#)
[vos_dev_read\(\)](#)
[vos_dev_ioctl\(\)](#)

Driver Functions

[uart_init\(\)](#)
[VOS_IOCTL_COMMON_ENABLE_DMA](#)
[UART Read and Write Operations](#)
[usbhost_init\(\)](#)
[USB Host General Transfer Block](#)
[VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#)
[VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_VID_PID](#)
[VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE](#)
[VOS_IOCTL_USBHOST_DEVICE_GET_INT_IN_ENDPOINT_HANDLE](#)
[VOS_IOCTL_USBHOST_DEVICE_SETUP_TRANSFER](#)

Library Functions

[strlen](#)

5.3.5 USBHostHID2 Sample

USBHostHID2 sample hierarchy:

USBHostHID2 Application		
USB Host Driver	UART Driver	string
VOS Kernel		

Description

The USBHostHID2 sample demonstrates reading from 2 interrupt endpoints simultaneously.

Function

The USBHost driver is connected to both USB ports. The first device on each USB port it used. The interrupt endpoint on each device is found. If it does not have an interrupt endpoint then an error is reported.

When both devices are initialised, both interrupt endpoints are read with the [non-blocking flag set](#). This results in the [vos_dev_read\(\)](#) call not blocking on a response from the devices. The [vos_wait_semaphore_ex\(\)](#) is used to wait on multiple semaphores, i.e. a read from either device to complete.

Sample output from the sample code is as follows:

```
Starting...
```



```
Enumeration complete Port 01
Init complete Port 01
Enumeration complete Port 00
Init complete Port 00
Port 01 Data: 0000000000000000
Port 01 Data: 0000000000000000
Port 01 Data: 0000260000000000
Port 01 Data: 0000000000000000
```

Comments

The sample should work with most HID devices but has only been tested with a selection of keyboards and barcode scanners.

Kernel Functions

[vos_init\(\)](#)
[vos_set_clock_frequency\(\)](#)
[vos_get_package_type\(\)](#)
[vos_create_thread\(\)](#)
[vos_start_scheduler\(\)](#)
[vos_init_semaphore\(\)](#)
[vos_wait_semaphore_ex\(\)](#)
[vos_dev_open\(\)](#)
[vos_dev_close\(\)](#)
[vos_dev_write\(\)](#)
[vos_dev_read\(\)](#)
[vos_dev_ioctl\(\)](#)

Driver Functions

[uart_init\(\)](#)
[VOS_IOCTL_COMMON_ENABLE_DMA](#)
[UART Read and Write Operations](#)
[usbhost_init\(\)](#)
[USB Host General Transfer Block](#)
[VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#)
[VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE](#)
[VOS_IOCTL_USBHOST_DEVICE_GET_INT_IN_ENDPOINT_HANDLE](#)
[VOS_IOCTL_USBHOST_DEVICE_SETUP_TRANSFER](#)

Library Functions

[strlen](#)
[memset](#)

5.3.6 USBMic Sample

USBMic sample hierarchy:

USBMic Application		
USB Host Driver	UART Driver	string
VOS Kernel		

Description

The USBMic sample demonstrates reading from an isochronous endpoint.

Function

The USBHost driver is connected to both USB ports. A search is made on USB port 1 for an Audio Streaming device. This is done with [VOS_IOCTL_USBHOST_DEVICE_GET_COUNT](#), [VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE](#) and [VOS_IOCTL_USBHOST_DEVICE_GET_CLASS_INFO](#) rather than [VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS](#). If the interface does not have an isochronous endpoint then it is ignored.

The [VOS_IOCTL_USBHOST_SET_INTERFACE](#) command is sent to the interface chosen and a SETUP packet to start sampling is sent with [VOS_IOCTL_USBHOST_DEVICE_SETUP_TRANSFER](#). The endpoint number for this SETUP packet is obtained from [VOS_IOCTL_USBHOST_DEVICE_GET_ENDPOINT_INFO](#).

An isochronous transfer is performed using [vos_dev_read\(\)](#) with 4 frames of data per read. The frame number to start the read is obtained by [VOS_IOCTL_USBHOST_HW_GET_FRAME_NUMBER](#) and the start frame set to the following frame. Data from the receive buffer is then sent to the UART interface.

Comments

The sample has been tested with a Prosound USB Microphone. It should also work with some webcams and internet telephone handsets or headsets.

Kernel Functions

[vos_init\(\)](#)

[vos_set_clock_frequency\(\)](#)

[vos_get_package_type\(\)](#)

[vos_create_thread\(\)](#)

[vos_start_scheduler\(\)](#)

[vos_init_semaphore\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_close\(\)](#)

[vos_dev_write\(\)](#)

[vos_dev_read\(\)](#)

[vos_dev_ioctl\(\)](#)

Driver Functions

[uart_init\(\)](#)

[VOS_IOCTL_COMMON_ENABLE_DMA](#)

[UART Read and Write Operations](#)

[usbhost_init\(\)](#)

[USB Host Isochronous Transfer Block](#)

[VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_COUNT](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_CLASS_INFO](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE](#)

[VOS_IOCTL_USBHOST_SET_INTERFACE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_ENDPOINT_INFO.](#)
[_VOS_IOCTL_USBHOST_HW_GET_FRAME_NUMBER](#)
[VOS_IOCTL_USBHOST_DEVICE_SETUP_TRANSFER](#)

Library Functions

[strlen](#)

[memset](#)

5.4 USB Slave Samples

The following samples are available in the samples USBSlave folder. The table below shows the features demonstrated in each sample.

	Kernel	Drivers	Libraries
USBSlaveFT232App Sample	Threads	USBSlave	N/A
USBSlaveHIDKbd Sample	Threads, Drivers	USBSlave	N/A

5.4.1 USBSlaveFT232App Sample

USBSlaveFT232App sample hierarchy:

USBSlaveFT232App Application
FT232 USB Slave Device
USB Slave Driver
VOS Kernel

Description

Demonstrates the use of the FT232 driver for the USB Slave.

Function

The VNC2 will enumerate as an FT232BM Device with a description of "VNC2 USB Serial". It will echo back any data received at the IN endpoint to the OUT endpoint.

Comments

Descriptors cannot be changed.

Kernel Functions

[vos_init\(\)](#)

[vos_set_clock_frequency\(\)](#)

[vos_get_package_type\(\)](#)

[vos_iomux_define_input\(\)](#) and [vos_iomux_define_output\(\)](#)

[vos_create_thread\(\)](#)

[vos_start_scheduler\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_read\(\)](#)

[vos_dev_write\(\)](#)

[vos_dev_ioctl\(\)](#)

Driver Functions

VOS_IOCTL_USBSLAVEFT232_ATTACH
 VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS

Library Functions

N/A

5.5 Firmware Samples

The following samples are available in the samples USBSlave folder. The table below shows the features demonstrated in each sample.

	Kernel	Drivers	Libraries
V2DAP Firmware	Threads, Delay, Semaphores	USBHost, FAT, BOMS, FIFO, UART, SPISlave, GPIO	string
V2DPS Firmware	Threads, Delay, Semaphores	USBHost, FAT, BOMS, UART, GPIO, USBSlave	string

5.5.1 VNC1L Firmware

Sample firmware for emulating the VNC1L command monitor and different firmware builds is provided as source code. The code can be modified to suit applications where VNC1L was used and improve on the supplied firmware.

Please note that the source code for the VNC1L firmware is not available and VNC1L firmware cannot be used on VNC2.

5.5.1.1 V2DAP Firmware

V2DAP application hierarchy:

V2DAP Application				
USB Host Driver	FAT File System	UART, SPI and FIFO Drivers	GPIO Driver	string
	BOMS Class Driver			
	USB Host Driver			
VOS Kernel				

Description

Provides code to emulate a VNC1L device running the V2DAP firmware version. A command monitor is used to send commands to the firmware and receive responses.

Function

Please refer to the "Vinculum Firmware User Manual" which can be obtained from <http://www.vinculum.com/> in the section for Documents.

Comments

Not all features and functions of the VNC1L can be replicated.

Kernel Functions

[vos_init\(\)](#)
[vos_set_clock_frequency\(\)](#)
[vos_get_package_type\(\)](#)
[vos_iomux_define_input\(\) and vos_iomux_define_output\(\)](#)
[vos_create_thread\(\)](#)
[vos_start_scheduler\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_write\(\)](#)

[vos_dev_ioctl\(\)](#)

Driver Functions

[usbhost_init\(\)](#)

[USB Host General Transfer Block](#)

[VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_BULK_OUT_ENDPOINT_HANDLE](#)

[usbHostFt232_init\(\)](#)

[VOS_IOCTL_USBHOSTFT232_ATTACH](#)

[VOS_IOCTL_UART_SET_BAUD_RATE](#)

[boms_init\(\)](#)

[BOMS_MSI_IOCTL_BOMS_ATTACH](#)

[BOMS_MSI_IOCTL_BOMS_DETACH](#)

[fat_init\(\)](#)

[fat_open\(\)](#)

[fat_close\(\)](#)

[fat_fileOpen\(\)](#)

[fat_fileClose\(\)](#)

[fat_fileWrite\(\)](#)

Library Functions

[memset](#)

5.5.1.2 V2DPS Firmware

V2DAP application hierarchy:

V2DAP Application				
FT232 USB Slave Device	FAT File System	UART Drivers	GPIO Driver	string
	BOMS Class Driver			
USB Slave Driver	USB Host Driver			
VOS Kernel				

Description

Provides code to emulate a VNC1L device running the VDPS firmware version. A command monitor is used to send commands to the firmware and receive responses.

Function

Please refer to the "Vinculum Firmware User Manual" which can be obtained from <http://www.vinculum.com/> in the section for Documents.

Comments

Not all features and functions of the VNC1L can be replicated.

Kernel Functions

[vos_init\(\)](#)

[vos_set_clock_frequency\(\)](#)

[vos_get_package_type\(\)](#)

[vos_iomux_define_input\(\) and vos_iomux_define_output\(\)](#)

[vos_create_thread\(\)](#)

[vos_start_scheduler\(\)](#)

[vos_dev_open\(\)](#)

[vos_dev_write\(\)](#)

[vos_dev_ioctl\(\)](#)

Driver Functions

[usbhost_init\(\)](#)

[USB Host General Transfer Block](#)

[VOS_IOCTL_USBHOST_GET_CONNECT_STATE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_CONTROL_ENDPOINT_HANDLE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE](#)

[VOS_IOCTL_USBHOST_DEVICE_GET_BULK_OUT_ENDPOINT_HANDLE](#)

[usbHostFt232_init\(\)](#)

[VOS_IOCTL_USBHOSTFT232_ATTACH](#)

[VOS_IOCTL_UART_SET_BAUD_RATE](#)

[boms_init\(\)](#)

[BOMS MSI_IOCTL_BOMS_ATTACH](#)

[BOMS MSI_IOCTL_BOMS_DETACH](#)

[fat_init\(\)](#)

[fat_open\(\)](#)

[fat_close\(\)](#)

[fat_fileOpen\(\)](#)

[fat_fileClose\(\)](#)

[fat_fileWrite\(\)](#)

Library Functions

[memset](#)

6 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales1@ftdichip.com
E-mail (Support) support1@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com
Web Site URL <http://www.ftdichip.com>
Web Shop URL <http://www.ftdichip.com>

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan, R.O.C.
Tel: +886 (0) 2 8797 1330
Fax: +886 (0) 2 8751 9737

E-mail (Sales) tw.sales1@ftdichip.com
E-mail (Support) tw.support1@ftdichip.com
E-mail (General Enquiries) tw.admin1@ftdichip.com
Web Site URL <http://www.ftdichip.com>

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.support@ftdichip.com
E-Mail (General Enquiries) us.admin@ftdichip.com
Web Site URL <http://www.ftdichip.com>

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 408, 317 Xianxia Road,
Shanghai, 200051
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-Mail (Sales): cn.sales@ftdichip.com
E-Mail (Support): cn.support@ftdichip.com
E-Mail (General Enquiries): cn.admin1@ftdichip.com
Web Site URL: <http://www.ftdichip.com>

Distributor and Sales Representatives

Please visit the Sales Network page of the [FTDI Web site](http://www.ftdichip.com) for the contact details of our distributor(s) and sales representative(s) in your country.

Vinculum is part of Future Technology Devices International Ltd. Neither the whole nor any part of the

information contained in, or the product described in this manual, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. This product and its documentation are supplied on an as-is basis and no warranty as to their suitability for any particular purpose is either made or implied. Future Technology Devices International Ltd will not accept any claim for damages howsoever arising as a result of use or failure of this product. Your statutory rights are not affected. This product or any variant of it is not intended for use in any medical appliance, device or system in which the failure of the product might reasonably be expected to result in personal injury. This document provides preliminary information that may be subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH United Kingdom. Scotland Registered Number: SC136640

7 Revision History

v1.0.2	Initial Vinculum II Toolchain Release	21 Apr 2010
v1.0.4	Update Toolchain and Documentation	12 May 2010
v1.2.0	Add Samples and Getting Started Sections Update Drivers and Libraries Sections	27 July 2010
v1.2.2	Add details to vos_wait_semaphore_ex New IOCTLs for USBSlave Driver Return values for driver init() functions	5th Oct 2010