FUSE C/C++ API Developers Guide

NT107-0068- Issue 8

Contacting Nallatech:

Support:

WWW:

Go to www.nallatech.com and click 'support'.

Email:

support@nallatech.com

Headquarters

Europe and Asia-Pacific:
Nallatech
Boolean House
One Napier Park
Cumbernauld
Glasgow G68 0BH
UK

North America:
Nallatech Inc
12565 Research Parkway
Orlando
Florida 32826
USA

Western Sales Office and Design Center
Nallatech Inc
226 Airport Parkway
Suite 470
San Jose 95110-1004
USA

Phone/Fax

Europe and Asia-Pacific:
Phone: +44 (0)1236 789500
Fax: +44 (0)1236 789599

North America:
Phone: +1 407 384 9255
Fax: +1 407 384 8555

Western Sales Office and Design Center
Phone: +1 408 467 5075
Fax: +1 408 436 5402

WWW:

www.nallatech.com

Document Name: FUSE C/C++ API Developers Guide

Document Number: NT107-0068

Issue Number: Issue 8

Date of Issue: 14/09/04

Revision History:

Date	Issue Number	Revision
14/09/2004	8	New template

Trademark Information

The Nallatech Logo, the DIME logo, the DIME-II logo, FUSE, FIELD Upgradeable Systems Environment, DIME, DIME-II, TCL Plug-In for FUSE, and the "Bally", Ben and "Strath" product name prefixes are all trademarks of Nallatech Limited. "The Algorithms to Hardware Company", "Making Hardware Soft", "FPGA-Centric Systems", "the only logical solution" and "software defined systems" are Service Marks of Nallatech Limited.

All products or brand names mentioned herein are used for identification purposes only and are trademarks, registered trademarks, or service marks of their respective owners.

Copyright Information

Contents

About this Developers Guide

This Developers Guide provides detailed information on the FUSE C/C++ API. It allows you to become acquainted with the API, its features and the functionality it provides. After reading the introduction you should proceed with installation instructions then work through the examples which provide a basic starting point for working with the API. The reference section provides a list of functions which can be used in conjunction with the FUSE C/C++ API.

Symbols Used

Throughout this Guide there are symbols to draw attention to important information:

▼ **The red arrow symbol indicates a set of procedures to follow, such as installing software or setting up hardware.**

ℹ The blue ‘i’ symbol indicates useful or important information.

❗ The red ‘!’ symbol indicates a warning, which requires special attention.

Reference Guide Format

The Reference Guide is divided into **Sections**, which are grouped into **Parts**. The parts divide the document as follows:

- Introduction: Provides a brief introduction to the Developers Guide and the FUSE C/C++ API. Installation instructions are also described here
- Implementation: Provides details on how to use the FUSE C/C++ API and example applications
- Reference Information: Provides reference information on the FUSE API functions. Use this section as a 'quick reference' to the API functions

Related Documentation

Nallatech	FUSE System Software User Guide

Abbreviations

API:	Application Program Interface
DIME:	DSP and Image Processing Modules for Enhanced FPGAs
DIMESDL:	DIME Software Development Library
DMA:	Direct Memory Access
FPGA:	Field Programmable Gate Array
FUSE:	Field Upgradeable System Environment
GUI:	Graphical User Interface
I/O:	Input/Output
PPS:	Programmable Power Supplies
SRAM:	Static Random Access Memory

Typographical Conventions

The following typographical convention are used in this manual:

- Red text indicates a cross-reference to information within the document set you are currently reading. Click the red text to go to the referenced item. To return to the original page, right-click anywhere on the current page and select **Go To Previous View**.
- Blue underlined text indicates a link to a Web page. Click blue-underlined text to browse the specified Web site.
- *Italics* denotes the following items:
  - References to other documents:

    See the *FUSE System Software User Guide* for more information.
  - Emphasis in text:

    Enable Loopback should *not* be enabled until all other registers have been set up.

FUSE Naming Conventions

Please note that the FUSE C/C++ API clocks are named differently in the FUSE System Software compared to this Developers Guide. The clock naming conventions are shown in Table 1 on page xii.

Clock Names in FUSE	Clock Names in Documentation
System Clock (SYSCLK)	Clock A (CLK A)
DSP Clock (DSPCLK)	Clock B (CLK B)
Pixel Clock (PIXCLK)	Clock C (CLK C)

Table 1: FUSE Naming Conventions

Comments and Suggestions

At the back of this book, you will find a remarks form. We welcome any comments you may have on our product or its documentation. Your remarks will be examined thoroughly and taken into account for future versions of Nallatech products.

Part I:Introduction

This part of the Reference Guide provides an introduction to the FUSE C/C++ API and outlines its key features and functionality. Step-by-step installation instructions are also included in the Getting Started section.

Section 1

FUSE C/C++ API Overview

In this section:

- FUSE C/C++ API Key Features

1.1 Key Features

The FUSE API is a pure software product that allows the Nallatech hardware to be easily integrated with software removing any interfacing issues. Developers can create their own applications, using the FUSE API in addition to their own code, to interface directly with their Nallatech hardware. The key features of the FUSE C/C++ API include:

- Fast and simple device configuration
- Multiple card support
- Multiple interface support
- Multiple operating system support
- Interfacing & control of Nallatech hardware features
- Generic function interface

Figure 1 on page 4 shows the layered API approach to interfacing with Nallatech hardware.

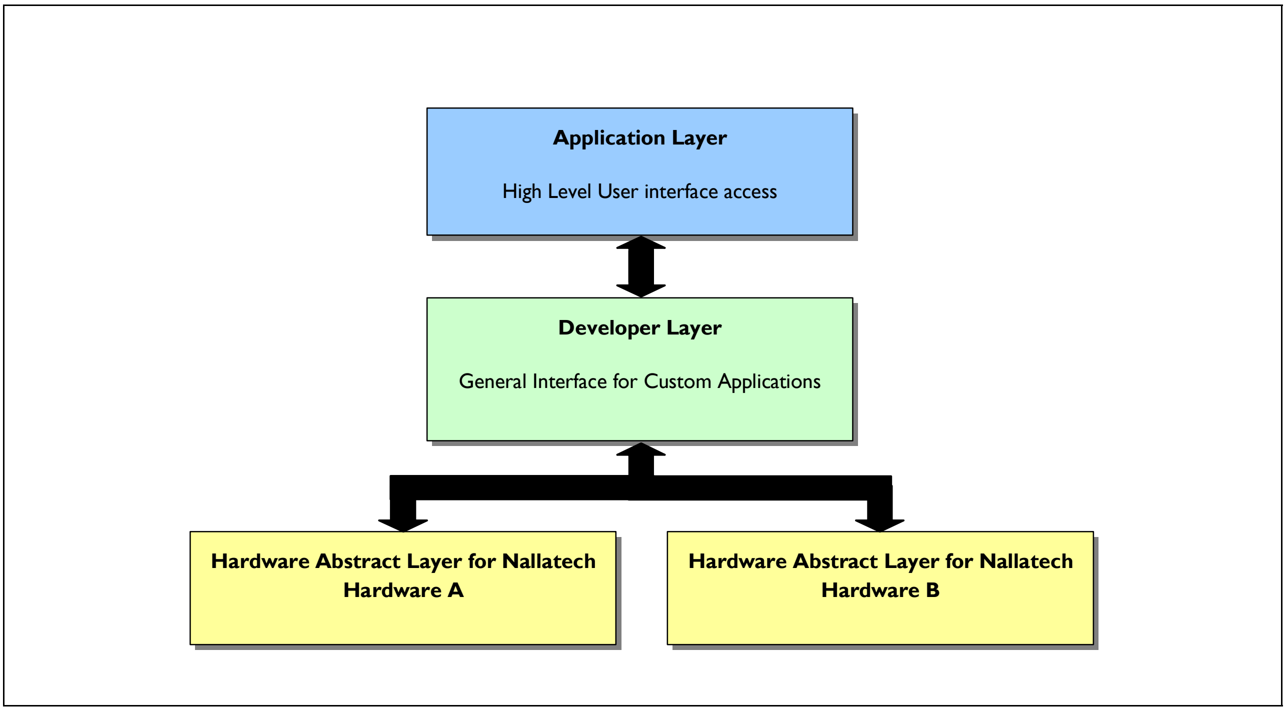

Application Layer
High Level User interface access
Developer Layer
General Interface for Custom Applications
Hardware Abstract Layer for Nallatech Hardware A
Hardware Abstract Layer for Nallatech Hardware B

Figure 1: FUSE API Layers

The hardware abstract layer interfaces with the custom Nallatech hardware and cannot be accessed by developers. Access to this layer is only possible indirectly through the developer layer, which effectively removes all hardware interfacing issues. The interface to the hardware abstract layer is therefore not provided and is only used for internal development by Nallatech. The developer layer is the main layer used by developers when interfacing with the board for custom applications. It consists of a library called DIMESDL (DIME Software Development Library), which contains functions that are detailed later in this Developers Guide. The application layer, intended for high level user interface access, uses functions from the developer layer to communicate with the hardware. An application such as the FUSE Probe Tool in which the user can control the hardware via a GUI is an example of an application layer product.

Section 2

Getting Started

In this section:

- Installation

2.1 Installation

2.1.1 Host System Requirements

The FUSE API runs under the following operating systems:

- Microsoft Windows XP Professional
- Microsoft Windows 2000
- Microsoft Windows Millennium Edition
- Microsoft Windows NT Service Pack 4
- Microsoft Windows 98

2.1.2 Software Installation

Windows

▼ **To install FUSE software in Windows NT/2000/XP use the following procedures:**

1. Insert the supplied FUSE System Software CD into your system's CD-ROM drive and wait for the CD to autorun. If autorun does not start click 'Start->Run' from the taskbar and run the following program: CD_Drive:\ autorun.exe.

2. In the FUSE Main Menu click on 'Install FUSE Application Software'.

3. The installation process begins. Work through the series of dialog boxes until the 'Finish' box is reached.

4. Click 'Finish' to install the software.

5. Restart the PC to complete the installation.

Linux

▼ **To install FUSE software in Linux RedHat use the following procedures:**

1. Insert the supplied FUSE System Software CD into your system's CD-ROM drive.

2. Mount the installation CD:

```
>mount /dev/cdrom /mnt/cdrom
```

3. Change directory as follows:

```
cd/mnt/cdrom
```

4. Type ‘rpm -ihv’ followed by the full name of the rpm (RedHat Package Format) file.

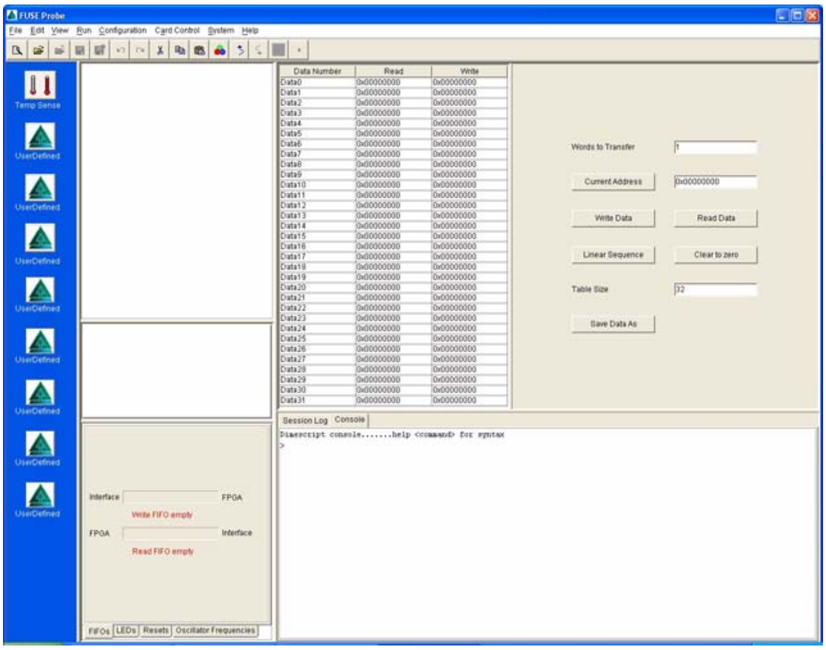
```
>rpm -ihv fuse-1-7.rpm
```

5. This installs FUSE to the following location:

```
>/usr/local/nallatech/FUSE
```

2.1.3 Initial Confidence Test

After rebooting the machine and installing the DIME-II hardware as described in the *DIME-II Installation Guide* (located on your product CD) run the FUSE Probe Tool from the 'Start Menu' - 'Programs' - 'FUSE' - 'Software'. If the software and drivers have been installed correctly and the hardware is present in the PC, a screen similar to that shown in Figure 2 on page 6 appears.



Data Number	Read	Write
Data0	0x00000000	0x00000000
Data1	0x00000000	0x00000000
Data2	0x00000000	0x00000000
Data3	0x00000000	0x00000000
Data4	0x00000000	0x00000000
Data5	0x00000000	0x00000000
Data6	0x00000000	0x00000000
Data7	0x00000000	0x00000000
Data8	0x00000000	0x00000000
Data9	0x00000000	0x00000000
Data10	0x00000000	0x00000000
Data11	0x00000000	0x00000000
Data12	0x00000000	0x00000000
Data13	0x00000000	0x00000000
Data14	0x00000000	0x00000000
Data15	0x00000000	0x00000000
Data16	0x00000000	0x00000000
Data17	0x00000000	0x00000000
Data18	0x00000000	0x00000000
Data19	0x00000000	0x00000000
Data20	0x00000000	0x00000000
Data21	0x00000000	0x00000000
Data22	0x00000000	0x00000000
Data23	0x00000000	0x00000000
Data24	0x00000000	0x00000000
Data25	0x00000000	0x00000000
Data26	0x00000000	0x00000000
Data27	0x00000000	0x00000000
Data28	0x00000000	0x00000000
Data29	0x00000000	0x00000000
Data30	0x00000000	0x00000000
Data31	0x00000000	0x00000000

Words to Transfer: 1

Current Address: 0x00000000

Table Size: 32

Interface → FPGA: Write FIFO empty

FPGA → Interface: Read FIFO empty

```
Dimescript console.......help <command> for syntax
>
```

Figure 2: FUSE Probe Tool

Part II:Implementation

This part of the Reference Guide provides detailed information on how to use the FUSE C/C++ API and its key components.

Section 3

Using FUSE C/C++ API

In this section:

- General Implementation Information

3.1 General Implementation Information

When developing with the FUSE API certain files must be added to your project to gain access to the API functions. These files are:

- dimesdl.h
- dimesdl.lib (coff version of the library for inclusion in Microsoft projects)
- dimesdlomf.lib (omf version of the library for inclusion in Borland projects)

The above files can all be found in the include directory within the FUSE install.

For some motherboards there will be an additional header file that allows access to specific card functions. Details of this header file will be included in the *FUSE Compatibility* section of your *Motherboard Reference Guide*. Please refer to this for further information.

The example designs detailed in this part of the Guide have all been produced using Microsoft Visual Version 6 and can be found in the FUSE API Examples directory with the FUSE install.

Section 4

Examples

In this section:

- Fundamental Steps
- Locating and Opening Examples
- Device Configuration Examples
- DMA Transfer Examples
- Interrupt Example

4.1 Fundamental Steps

There are two key steps that are required to enable all the API functions. The first step is to locate all the cards over a certain interface and the second is to open a selected card. Once this has been achieved the FPGAs can be configured, DMA transfers are possible, LEDs can be flashed etc. To locate a card DIME_LocateCard must be called. After this has returned successfully other locate functions can be called to find out details on what was located and the card can be opened using DIME_OpenCard. Once the card has been opened the other API functions become available. Finally the card and the locate must be closed using DIME_CloseCard and DIME_CloseLocate.

4.2 Locating and Opening Examples

There are three example programs included in the FUSE install that specifically deal with locating and opening cards. These are:

- Opening a single card
- Opening multiple cards over the same interface
- Opening multiple cards over different interfaces

All the above programs open one or more cards and flash LEDs to prove the cards have been opened before closing the cards down.

4.3 Device Configuration Examples

Once a card has been opened the next step is the configuration of a device - typically an FPGA. This device may be on the motherboard itself or may be on a module. To configure the device the developer simply needs to use one of the device configuration functions detailed on page 15 such as DIME_ConfigOnBoardDevice or DIME_ConfigControl.

There are two example programs included in the FUSE install that demonstrate configuration of both the on-board FPGA and a module FPGA with the LED snake design. Details on the design are included in the *Motherboard or Module Reference Guide*. The two example programs are:

- Configuring the on board FPGA with ledsnake

- Configuring a module with ledsnake

4.4 DMA Transfer Examples

Once a card has been opened and a design placed into the FPGA, data is normally required to be transferred between the card and the software application via DMA transfers. Firstly if the transfer is to occur over the PCI then the memory needs to be locked down prior to the transfer. This is achieved using the DIME_LockMemory function. Now a DMA channel needs to be opened between the card and the PC. This can be achieved using the DIME_DMAOpen function. Now the actual transfer of data can take place. Functions such as DIME_DMAReadToLockedMem and DIME_DMAWriteFromLockedMem perform this data transfer. Once all the data has been transferred the DMA channel can be closed and the locked memory can be unlocked.

In the examples detailed below the Nallatech ping design is used to 'turn the data around' on the card. Details of this design can be found in the *PCI To User FPGA Interface Core Application Note* on the FUSE CD at the location: '<CDROM Drive>:\ApplicationNotes\NT302-0000 Spartan to Virtex Interface'.

It is worth noting that in the examples the memory is locked down at the start of the program and unlocked at the end rather than locked and unlocked for each transfer. This improves the speed to the transfers. The two example programs are:

- DMA transfers
- DMA transfers with two cards

In the second example data is first written and then read back from the first card as in the DMA transfers example and then the same data is written to and read back from the second card. Data is never sent from one card directly to the other card.

4.5 Non-threaded Interrupts Example

An FPGA design may need to communicate with the PC using interrupts. In order to use interrupts they must firstly be enabled using the DIME_InterruptControl function. Once enabled whenever the FPGA produces an interrupt then a genuine hardware interrupt is produced. For the software to find out if an interrupt has occurred the DIME_InterruptControl function can be used with the dintWAIT command mode. Once the software has finished dealing with interrupts and wants to disable then again DIME_InterruptControl is used with the dintDISABLE command mode. The non-threaded interrupts example program is:

- Non-threaded interrupts

Once interrupts have been enabled only polling DMA transfers should be used. Once interrupts have been enabled all interrupts that occur are logged. When waiting for interrupts if an interrupt has been logged then the function will return immediately.

For a threaded example and further information on how to use interrupts please refer to the *Interrupts in FUSE Application Note* on the FUSE CD at the location: '<CDROM Drive>:\ApplicationNotes\NT302-0026 Using Interrupts\documentation'.

Part III:Reference Information

This part of the Guide provides reference information on the FUSE API, including a list of functions

Section 5

Functions By Category

In this section:

- FUSE C/C++ API Functions by Category

5.1 Locating Cards

These functions are used to locate the cards within your system and to retrieve simple information about these located cards to help determine which card should be opened.

- DIME_LocateCard
- DIME_CloseLocate
- DIME_LocateStatus
- DIME_LocateStatusPtr

5.2 Opening and Closing the Card

These functions are used to de/allocate system resources for the chosen card and to create/destroy a handle for the card. This handle is required for all functions that interface with the card.

- DIME_OpenCard
- DIME_CloseCard

5.3 Oscillator

These functions are used for controlling cards oscillators.

- DIME_SetOscillatorFrequency

5.4 LEDs

These functions are used to read or write to the interface LEDs.

- DIME_ReadLEDs
- DIME_WriteLEDs

5.5 Reset

These functions are used to control the cards various resets.

- DIME_CardResetControl
- DIME_CardResetStatus

5.6 Interrupts

These functions are for controlling the cards various interrupts.

- DIME_InterruptStatus
- DIME_InterruptControl

5.7 DMA Transfers

These functions are for dealing with transfer of data between the card(s) and the PC.

- DIME_LockMemory
- DIME_UnLockMemory
- DIME_DMAOpen
- DIME_DMAClose
- DIME_DMAStatus
- DIME_DMAControl
- DIME_DMARead
- DIME_DMAWrite
- DIME_DMAReadToLockedMem
- DIME_DMAWriteFromLockedMem
- DIME_DataWriteSingle
- DIME_DataReadSingle
- DIME_DataRead
- DIME_DataWrite
- DIME_AddressWriteSingle

5.8 Device Configuration

These functions are used to configure devices such as FPGAs and for assigning designs to devices.

- DIME_ConfigOnBoardDevice
- DIME_MemConfigOnBoardDevice
- DIME_ConfigDevice
- DIME_MemConfigDevice
- DIME_ConfigModule
- DIME_ConfigCard
- DIME_ConfigSetBitsFilename
- DIME_ConfigSetBitsMemory
- DIME_ConfigSetBitsFilenameAndConfig
- DIME_ConfigSetBitsMemoryAndConfig
- DIME_ConfigGetBitsFilename

5.9 Card/System Definition Files

These functions allow the user to create card and system definition files that allow faster loading of both cards and systems.

- DIME_SaveCardDefinition
- DIME_LoadCardDefinition
- DIME_SaveSystemDefinition
- DIME_LoadSystemDefinition

5.10 JTAG

These functions allow control over the JTAG chain.

- DIME_JTAGStatus
- DIME_JTAGControl

5.11 System Information and Control

These functions enable the FUSE API to be controlled to suit the developers needs and for system information to be obtained.

- DIME_SystemStatus
- DIME_SystemControl
- DIME_SystemStatusPtr
- DIME_GetError
- DIME_GetLocateVersionNumber
- DIME_GetVersionNumber

5.12 Card/Module/Device Information

These functions allow control over various aspects of the cards, modules and devices.

- DIME_CardStatus
- DIME_CardControl
- DIME_CardStatusPtr
- DIME_CardControlPtr
- DIME_ModuleStatus
- DIME_ModuleControl
- DIME_ModuleStatusPtr
- DIME_ModuleControlPtr
- DIME_DeviceStatus
- DIME_DeviceControl
- DIME_DeviceStatusPtr
- DIME_DeviceControlPtr

5.13 Programmable Power Supplies (PPS)

These functions allow control of the programmable power supplies for certain motherboards.

- DIME_PPSStatus
- DIME_PPSControl

5.14 All I/O

These functions deal with all I/O such as digital, peripheral and miscellaneous I/O.

- DIME_ReadPIO
- DIME_WritePIO

5.15 Multiple Configuration GUI

These functions are for linking the example multiple configuration GUI to your application.

- DIME_MConfigGUI
- DIME_MConfigGUIExit
- DIME_ShowMConfigGUI

Section 6

Functional Reference

In this section:

- Alphabetical listing of each function within the FUSE API with full details of the function.

6.1 DIME_AddressWriteSingle

Syntax DWORD DIME_AddressWriteSingle(DIME_HANDLE handle, DWORD *Data, volatile DWORD *Terminate, DWORD Timeout)

Arguments handle is a valid handle to a DIME carrier card.

Data is a pointer to the PC memory that holds the address to be written.

Terminate points to a memory location that enables termination of a transfer. This memory location is 0 under normal conditions. A non-zero value terminates the transfer. This argument can be NULL if not used.

Timeout is the maximum time in milliseconds that the transfer is allowed to take. If all the data has not been transferred within this time period the transfer is terminated and an error condition is returned. If this is set to zero then the timeout is effectively infinite.

Return There are several possible returns for the DMA transfer. See Table 18 on page 41.

Description This handles the transfer of a single 32-bit word from the PC memory pointed to by 'Data' to the Interface connected to the on-board FPGA of the DIME Motherboard.

In this case the 'AS/DS' line of the FPGA interface is asserted to indicate that address data is being passed on the Data lines.

Notes This function should only be used in single card systems. In systems with more than one card it is strongly advised that the DIME_DMA functions are used instead.

6.2 DIME_CardControl

Syntax DWORD DIME_CardControl(DIME_HANDLE handle, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

CmdMode: This argument is used to specify what particular aspect of a card is to be controlled. There are no current command modes for this function.

Value: This argument is used to specify the action for a command mode.

Return Returns -1 on error.

Description This function is used to control certain aspects of the card.

6.3 DIME_CardControlPtr

Syntax DWORD DIME_CardControlPtr(DIME_HANDLE handle, DWORD CmdMode, void *pValue)

Arguments handle is a valid handle to a DIME carrier card.

CmdMode: This argument is used to specify what particular aspect of a card is to be controlled. There are no current command modes for this function.

pValue: This argument is used to specify the action for a command mode.

Return Returns -1 on error.

Description This function is used to control certain aspects of the card that cannot be controlled using DIME_CardControl.

6.4 DIME_CardResetControl

Syntax DWORD DIME_CardResetControl (DIME_HANDLE handle, DWORD ResetNum, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

ResetNum: This argument is used to specify which reset is to be controlled. Table 2 on page 20 gives details of available resets.

ResetNum	Description
drINTERFACE	This is the reset for the interface FPGA. Toggling this reset causes an internal reset of the cards interface (PCI, USB etc.) that is primarily used to clear the internal interface FIFOs of the interface FPGA.

Table 2: DIME_CardResetControl ResetNum Argument Options

ResetNum	Description
drSYSTEM	This is the reset for the DIME system reset. This signal is typically connected to the User FPGA(s) and all module sites. This reset is designed to allow a software controlled reset of your implemented design. Toggling this reset has a minimum pulse width of 100ns. Please consult your *Motherboard Reference Guide* for more details on this signal.
drONBOARDFPGA	This is commonly a single bit signal that is provided as part of the communications signals between the interface FPGA and the on board FPGA. This provides an active-low software controllable reset to on board FPGA. Toggling this reset has a minimum pulse width of 100ns.

Table 2: DIME_CardResetControl ResetNum Argument Options

CmdMode: This argument is used to specify the command on the selected reset. Table 3 on page 21 gives details of the available commands.

CmdMode	Description
drDISABLE	This de-asserts the reset line for the selected reset.
drENABLE	This asserts the reset line for the selected reset.
drTOGGLE	This toggles the reset line for the selected reset.

Table 3: DIME_CardResetControl CmdMode Argument Options

The value argument is not used in this function and is only included for consistency.

Return Returns 0 on success. Non-zero on error.

Description This function controls the software resets. For more details on these resets please consult your *Motherboard Reference Guide*.

Example

```
//Enable the OnBoardFPGA reset.
DIME_CardResetControl(handle,drONBOARDFPGA,drENABLE,0);
//Disable the OnBoardFPGA reset.
DIME_CardResetControl(handle,drONBOARDFPGA,drDISABLE,0);
//Toggle the OnBoardFPGA reset.
DIME_CardResetControl(handle,drONBOARDFPGA,drTOGGLE,0);

// Enable the System reset.
DIME_CardResetControl(handle,drSYSTEM, drENABLE,0);
// Disable the System reset.
DIME_CardResetControl(handle,drSYSTEM, drDISABLE,0);
// Toggle the System reset.
DIME_CardResetControl(handle,drSYSTEM,drTOGGLE,0);

// Toggle the interface FPGA reset.
DIME_CardResetControl(handle,drINTERFACE,drToggle,0);
```

Figure 3: Examples of Using DIME_CardResetControl

6.5 DIME_CardResetStatus

Syntax DWORD DIME_CardResetStatus(DIME_HANDLE handle, DWORD ResetNum, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

ResetNum: This argument is used to specify which reset status information is to be retrieved. See Table 2 on page 20 for details.

CmdMode: This argument is used to specify the command on the selected reset. See Table 3 on page 21 for details.

Return Returns 1 on an invalid handle, 0 on error and drCONTROLABLE or drTOGGLEONLY depending on the resets capabilities. A return of drCONTROLABLE means that the reset can be toggled, enabled or disabled. A return of drTOGGLEONLY means that the reset can only be toggled.

Description This function allows the user to determine the capability of the selected reset.

Notes PCI resets are toggle only. System and on board FPGA resets are controllable.

6.6 DIME_CardStatus

Syntax DWORD DIME_CardStatus(DIME_HANDLE handle, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

CmdMode: This argument is used to specify what particular aspect of card status information is to be returned. Table 4 on page 23 gives details of the available command modes.

CmdMode	Description
dinfMULTICONFIGLICENCE	This function returns whether the multiple configuration licence is valid on this system. A return value of 1 indicates that the card has a multiple configuration licence. A return value of 0 indicates that it does not.
dinfNUMBERMODULES	Returns the number of modules installed in the card. Note the card itself counts as an on-board module.
dinfNUMBERSLOTS	Returns the number of module slots there are for the card.
dinfSLOTSUSED	This returns a bit wise value to indicate if a module is plugged into a particular DIME slot. A '1' in a particular bit location indicates that a module is present otherwise the slot is free. Bit 0 represents slot 0, bit 1 represents slot 1 etc.
dinfMOTHERBORDTYPE	Returns the motherboard type of the card. See Table 29 on page 55 for details.
dinfCARDMAXJTAGSPEED	Returns the maximum speed that the cards JTAG chain can be driven. See Table 26 on page 53 for details.
dinfSERIALNUMBER	Returns the serial number of the card.
dinfDESCRIPTION	Returns a description of the motherboard

Table 4: DIME_CardStatus CmdMode Argument Options

Return The return value is dependant upon the command mode. Returns -1 on error.

Description This function returns card status information.

6.7 DIME_CardStatusPtr

Syntax void *DIME_CardStatusPtr(DIME_HANDLE handle, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

CmdMode: This argument is used to specify what particular aspect of card status information is to be returned. The Table 5 on page 24 gives details of the available command modes.

CmdMode	Description
dinfDEFAULTIMAGE	This command mode returns a string (char *), which is the default image filename for the card.
dinfDEFAULTICON	This command mode returns a string (char *), which is the default icon filename for the card.
dinfFAILEDMDF	This command mode returns a string (char *), which is the last failed mdf.
dinfIMAGEFILENAME	This command mode returns a string (char *), which is the image filename for the card.
dinfICONFILENAME	This command mode returns a string (char *), which is the icon filename for the card.
dinfDESCRIPTION	This command mode returns a string (char *), which is a short description of the card.

Table 5: DIME_CardStatusPtr CmdMode Argument Options

Return The return value is dependant upon the command mode. Returns NULL on error.

Description This function returns card status information that cannot be returned using DIME_CardStatus.

Notes If a pointer to a string is returned this string is only valid until the next call is made into the library. It is therefore advised that either the string is used directly or that it is copied for later use.

6.8 DIME_CloseCard

Syntax void DIME_CloseCard(DIME_HANDLE CardHandle)

Arguments handle is a valid handle returned from DIME_OpenCard.

Return N/A

Description This function closes down the handle returned from DIME_OpenCard. It de-allocates all system resources that where used when interfacing with the card.

Notes Call this function after your application has finished using the card to ensure that the card is closed properly and all systems resources that where used are available for other applications.

Example

```
#include <dimesdl.h> //This is held in the include directory
                        within FUSE.
DIME_HANDLE hCard1;
LOCATE_HANDLE hLocate;
DWORD LEDs;
//Locate the Cards on the PCI interface
hLocate=DIME_LocateCard(dlPCI,mbtALL,NULL,dldrDEFAULT,dlDEFAULT);

//Open the first card found in the locate.
hCard1=DIME_OpenCard(hLocate,1,dccOPEN_DEFAULT);

//Change the LEDs
LEDs=DIME_ReadLEDs(hCard1);
DIME_WriteLEDs(hCard1,(LEDs-1));

//Close the card down.
DIME_CloseCard(hCard1);

//Finally close the locate down.
DIME_CloseLocate(hLocate);
```

Figure 4: Locating, Opening and Closing a Card

6.9 DIME_CloseLocate

Syntax void DIME_CloseLocate(LOCATE_HANDLE LocateHandle)

Arguments handle is a valid handle returned from DIME_LocateCard.

Return N/A

Description This function closes down the handle returned from DIME_LocateCard.

Notes This function should be the final function called and should only be used after all the cards that were opened using this locate handle have been closed down.

Example See Figure 4 on page 25.

6.10 DIME_ConfigCard

Syntax DWORD DIME_ConfigCard (DIME_HANDLE handle, DWORD *ModuleProgress, DWORD *DeviceProgress, DWORD *ConfigProgress)

Arguments handle is a valid handle to a DIME carrier card.

ModuleProgress is the progress through the modules.

DeviceProgress is the progress through the device.

ConfigProgress is the progress through a configuration.

Return The function simply returns the last value returned from the configuration of a module. A return of '-1' indicates an invalid carrier card.

Description The function simply iterates through each of the detected modules and calls the DIME_ConfigModule function. The function continues to iterate through the configuration of each module until a module returns a boot result that is not equal to dcfgOK_NOSTATUS, dcfgDL_IH_NOCRC or dcfgOK_STATUS. When this happens the invalid boot result is returned.

6.11 DIME_ConfigControl

Syntax DWORD DIME_ConfigControl (DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the module you want to target.

DeviceNumber is the device in that module you want to target.

CmdMode is control command to implement.

Return	Description
dcfgFRAMEADDR	This command sets the Frame Address. The frame address is only used in the Readback functions and is used to set what Frame is to be read back. By default the address is 0x00000000

Table 6: ConfigControl Commands

Value is the command specific.

Return Return is zero on success

Description This function is used to control some of the aspects of configuration

6.12 DIME_ConfigDevice

Syntax DWORD DIME_ConfigDevice(DIME_HANDLE handle, const char *FileName, DWORD ModuleNumber, DWORD ModuleDeviceNumber, DWORD *Progress, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

FileName is the filename of a bitfile for configuring the FPGA.

ModuleNumber is the Module that is being addressed.

ModuleDeviceNumber is the selected device within the selected module.

Progress should point to a variable, which will be updated with the actual position in the configuration. The position in the configuration file is expressed as a percentage (0 - 100). This is only useful in multi-threaded applications and may point to a valid location or NULL in single threaded applications.

Flags: This argument is used to control the configuration of the on-board device.

Return	Description
dcfgPROGSECURE	Program the device in secure mode.
dcfgPROGKEYS	This is used to program the triple des keys for that specific device. Note, a file with extension nky that is output from ISE is used instead of the bitfile. Note when programming the encryptions keys the pc has to be turned off and back on before the encrypted bitstream works.

Table 7: Configuration Flags

Return This function has several possible returns. Table 8 on page 27 gives details.

Return	Description
dcfgOK_NOSTATUS	Configured completed successfully although no post configuration checking carried out.
dcfgOK_STATUS	Configuration completed successfully as indicated by read back of FPGA Status register. DONE high, INIT high, No CRC errors detected.
dcfgINVALID_CARD	The handle argument is invalid.
dcfgBIT_FILE	Returned when the specified bitfile could not be successfully opened.
dcfgINTEG_FAIL	Indicates that the JTAG integrity scan check has failed and the chain is apparently incomplete.
dcfgDL_IL_NOCRC	Configuration Status - DONE Low, INIT Low, No CRC errors detected.
dcfgDL_IL_CRC	Configuration Status - DONE Low, INIT Low, CRC errors detected.
dcfgDL_IH_NOCRC	Configuration Status - DONE Low, INIT high, No CRC errors detected.

Table 8: Configuration Function Return Values

Return	Description
dcfgDL_IH_CRC	Configuration Status - DONE Low, INIT high, CRC errors detected.
dcfgDH_IL_NOCRC	Configuration Status - DONE high, INIT low, No CRC errors detected.
dcfgDH_IL_CRC	Configuration Status - DONE high, INIT low, CRC errors detected.
dcfgDH_IH_CRC	Configuration Status - DONE high INIT high, CRC errors detected.
dcfgUNKNOWN	Unidentifiable configuration result.
dcfgNOLIC	Multiple Configuration Licence not available.
dcfgDEV_BIT_MIS	The bitfile is incorrect for the FPGA device type.
dcfgERROR	An unspecified error has occurred.

Table 8: Configuration Function Return Values

Description This function configures the specified device with the specified bitfile.

Notes The bitfile must be configured to use the JTAG clock for configuration rather than the default of the CCLK.

6.13 DIME_ConfigGetBitsFilename

Syntax const char *DIME_ConfigGetBitsFilename (DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the number of the selected module.

DeviceNumber is the number of the selected device.

Return Returns a pointer to the filename that has been assigned to the selected device. If no filename has been set for the device then a NULL pointer is returned.

Description This function returns the filename that is assigned to a device.

6.14 DIME_ConfigGetBitsMemory

Syntax DWORD *DIME_ConfigGetBitsMemory (DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber, DWORD *ByteLength)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the number of the selected module.

DeviceNumber is the number of the selected device.

ByteLength is the address of the location that the byte length of the assigned bit-stream will be placed.

Return Returns a pointer to the start of the bit-stream that has been assigned to the selected device. If no bit-stream has been set for the device then a NULL pointer is returned.

Description This function returns the pointer to the start of the bit-stream that is assigned to a device.

6.15 DIME_ConfigModule

Syntax DWORD DIME_ConfigModule (DIME_HANDLE handle, DWORD ModuleNumber, DWORD *DeviceProgress, DWORD *ConfigProgress)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the Number of the selected module.

DeviceProgress is the progress through the device.

ConfigProgress is the progress through a configuration.

Return The function simply returns the last value returned from the configuration of a device. A return of '-1' indicates a non configuration error.

Description The function iterates through each of the devices in a specified module and will configure each device using DIME_ConfigDevice or DIME_MemConfigDevice provided that the device has been defined in the MDF as bootable and that a bitfile has actually been assigned to the device.

‘Note that it will stop iterating through the devices at the first device that is unsuccessfully configured. An unsuccessful configuration is indicated by a returned configuration result that is not equal to dcfgOK_NOSTATUS, dcfgDL_IH_NOCRC or dcfgOK_STATUS.

6.16 DIME_ConfigOnBoardDevice

Syntax DWORD DIME_ConfigOnBoardDevice(DIME_HANDLE handle, const char *FileName, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

FileName is the filename of the bitfile that is to be used for booting the on board FPGA.

Flags: This argument is used to control the configuration of the on board device. See Table 7 on page 27.

Return This function has several possible returns. Please see Table 8 on page 27 for details.

Description This function configures the cards on board FPGA with the specified bitfile. Configuration is carried out using the cards JTAG chain.

Notes The bitfile must be configured to use the JTAG clock for configuration rather than the default of the CCLK.

6.17 DIME_ConfigSetBitsFilename

Syntax DWORD DIME_ConfigSetBitsFilename (DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber, const char *Filename, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the number of the selected module.

DeviceNumber is the number of the selected device.

Filename is the filename of the bitfile that is to be assigned to this device.

Flags: This argument allows the developer to configure the assignment of bitfiles to devices to suit the development requirements. Table 9 on page 30 gives details for the Flags.

Flags	Description
dcfgFREEEMBEDBITS	It is possible to embed a bitstream with a particular device instead of a filename. If this is the case then when assigning a Filename to a device that already has a bitstream assigned then the existing assigned bitstream may no longer be required. In this situation this flag should be used. This will de-allocate the system resources that were assigned to the embedded bit-stream.

Table 9: DIME_ConfigSetBitsFilename Flags Argument Options

Return Returns 0 upon success. Returns non-zero otherwise.

Description This function assigns a bitfile to the specified device. Once a device has a bitfile assigned this information is stored in any card definition file that is saved. Furthermore when either DIME_ConfigCard or DIME_ConfigModule is called this assigned bitfile is used to configure the device.

Notes This function only assigns the name of the bitfile to the device. No configuration is carried out.

6.18 DIME_ConfigSetBitsFilenameAndConfig

Syntax DWORD DIME_ConfigSetBitsFilenameAndConfig (DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber, const char *Filename, DWORD SetFlags, DWORD *Progress, DWORD ConfigFlags)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the number of the selected module.

DeviceNumber is the number of the selected device.

Filename is the filename of the bitfile that is to be assigned to this device.

SetFlags: These are the flags for the setting of the filename. Please refer to DIME_ConfigSetBitsFilename on page 30 for further details.

Progress: This is a pointer to a memory location that holds the configuration completion percentage. Please refer to DIME_ConfigControl on page 26 for further details.

ConfigFlags: These are the flags for the configuration of the device. Again please refer to DIME_ConfigControl for further details.

Return Returns the result of DIME_ConfigSetBitsFilename if there is an error. If no error occurs in this function then it returns the result of the device configuration.

Description This function assigns a bitstream filename to a particular device and then configures the device using the assigned bitstream file.

6.19 DIME_ConfigSetBitsMemory

Syntax DWORD DIME_ConfigSetBitsMemory (DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber, DWORD *Bits, DWORD ByteLength, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the number of the selected module.

DeviceNumber is the number of the selected device.

Bits is a pointer to the bitstream that is to be assigned to the device.

ByteLength is the byte length of the bitstream.

Flags: This argument allows the developer to configure the assignment of bitstreams to devices to suit the development requirements. Table 10 on page 31 gives details for the Flags.

Flags	Description
dcfgFREEEMBEDBITS	If the device already has a bitstream assigned and this bit-stream is no longer valid then using this flag will de-allocate the system resources that where assigned to the embedded bit-stream.

Table 10: DIME_ConfigSetBitsMemory Flags Argument Options

Returns 0 upon success. Returns non-zero otherwise.

Description This function assigns a bitstream to the specified device. Once a device has a bit-stream assigned this information is stored in any card definition file that is saved. Furthermore when either DIME_ConfigCard or DIME_ConfigModule is called this assigned bitstream is used to configure the device.

Notes This function only assigns the memory location of the bitstream to the device. No configuration is carried out. If the bitstream is moved or altered after it has been assigned then the bitstream will need to be re-assigned to the device.

6.20 DIME_ConfigSetBitsMemoryAndConfig

Syntax DWORD DIME_ConfigSetBitsMemoryAndConfig (DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber, DWORD *Bits, DWORD ByteLength, DWORD SetFlags, DWORD *Progress, DWORD ConfigFlags)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the number of the selected module.

DeviceNumber is the number of the selected device.

Bits is a pointer to the bitstream that is to be assigned to the device.

ByteLength is the byte length of the bitstream.

SetFlags: These are the flags for the setting of the filename. Please refer to DIME_ConfigSetBitsMemory on page 31 for further details.

Progress: This is a pointer to a memory location that holds the configuration completion percentage. Please refer to DIME_MemConfigDevice on page 61 for further details.

ConfigFlags: These are the flags for the configuration of the device. Again please refer to DIME_MemConfigDevice for further details.

Return Returns the result of DIME_ConfigSetBitsMemory if there is an error. If no error occurs in this function then it returns the result of the device configuration.

Description This function assigns a bitstream in memory to a particular device and then configures the device using the assigned bitstream.

6.21 DIME_ConfigStatus

Syntax DWORD DIME_ConfigStatus(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the module you wish to target.

DeviceNumber is the device in that module you wish to target.

CmdMode is control command to implement.

Return	Description
dcfgFRAMEADDR	This command sets the Frame Address. The frame address is only used in the Readback functions and is used to set which Frame is read back. By default the address is 0x00000000

Table 11: ConfigControl Commands

Return The return is the status information from the specific command set in CmdMode.

Description This function is used to return configuration status information.

6.22 DIME_DataRead

Syntax DWORD DIME_DataRead(DIME _HANDLE handle, DWORD *Data, DWORD WordCount, volatile DWORD *Terminate, DWORD *CurrCount, DWORD Timeout)

Arguments handle is a valid handle to a DIME carrier card.

Data is a pointer to the PC memory which receives data from the card. This should be 32-bit aligned.

WordCount is the number of 32-bit words to transfer.

Terminate is not used.

CurrCount points to memory location that holds the current total of words transferred, which is useful for feedback to the application. This argument can be NULL if not used. This argument can be used in multi-threaded applications to monitor the progress of the data transfer. In single threaded applications it can be used to return the total number of words transferred.

Timeout is the maximum time in milliseconds that the transfer is allowed to take. If all the data has not been transferred within this time period the transfer is terminated and an error condition is returned. If this is set to zero then the timeout is effective infinite.

Return There are several possible returns for the data transfer. See Table 19 on page 42 for details.

Description This handles the transfer of 'WordCount' 32-bit words from the Interface with the on board FGPA of the DIME Motherboard to the PC memory pointed to by 'Data'. The memory pointed to by 'Data' does not need to be contiguous, as this function will handle the internal transfer and memory management.

Notes This function should only be used in single card systems. In systems with more than one card it is strongly advised that the DIME_DMA functions are used instead.

6.23 DIME_DataReadSingle

Syntax DWORD DIME_DataReadSingle(DIME _HANDLE handle, DWORD *Data, volatile DWORD *Terminate, DWORD Timeout)

Arguments handle is a valid handle to a DIME carrier card.

Data is a pointer to the PC memory which receives data from the card. This should be 32-bit aligned.

Terminate is not used.

Timeout is the maximum time in milliseconds that the transfer is allowed to take. If all the data has not been transferred within this time period the transfer is terminated and an error condition is returned. If this is set to zero then the timeout is effectively infinite.

Return There are several possible returns for the DMA transfer. See Table 19 on page 42 for details.

Description This handles the transfer of a single 32-bit word from the Interface with the on board FGPA of the DIME Motherboard and places this 32-bit word into the PC memory pointed to by 'Data'.

Notes This function should only be used in single card systems. In systems with more than one card it is strongly advised that the DIME_DMA functions are used instead.

6.24 DIME_DataWrite

Syntax DWORD DIME_DataWrite(DIME _HANDLE handle, DWORD *Data, DWORD WordCount, volatile DWORD *Terminate, DWORD *CurrCount, DWORD Timeout)

Arguments handle is a valid handle to a DIME carrier card.

Data is a pointer to the PC memory which holds the data to be written. This should be 32-bit aligned.

WordCount is the number of 32-bit words to transfer.

Terminate is not used.

CurrCount points to memory location which holds the current total of words transferred, which is useful for feedback to the application. This argument can be NULL if not used. This argument can be used in multi-threaded applications to monitor the progress of the data transfer. In single threaded applications it can be used to return the total number of words transferred.

Timeout is the maximum time in milliseconds that the transfer is allowed to take. If all the data has not been transferred within this time period the transfer is terminated and an error condition is returned. If this is set to zero then the timeout is effectively infinite.

Return There are several possible returns for the data transfer. See Table 19 on page 42 for details.

Description This handles the transfer of 'WordCount' 32-bit words from the PC memory pointed to by 'Data' to the Interface connected to the on-board FGPA of the DIME Motherboard. The memory pointed to by 'Data' does not need to be contiguous, as this function will handle the internal transfer and memory management.

Notes This function should only be used in single card systems. In systems with more than one card it is strongly advised that the DIME_DMA functions are used instead.

6.25 DIME_DataWriteSingle

Syntax DWORD DIME_DataWriteSingle(DIME_HANDLE handle, DWORD *Data, volatile DWORD *Terminate, DWORD Timeout)

Arguments handle is a valid handle to a DIME carrier card.

Data is a pointer to the PC memory which holds the data to be written. This should be 32-bit aligned.

Terminate is not used.

Timeout this is the maximum time in milliseconds that the transfer is allowed to take. If all the data has not been transferred within this time period the transfer is terminated and an error condition is returned. If this is set to zero then the timeout is effectively infinite.

Return There are several possible returns for the DMA transfer. See Table 19 on page 42 for details.

Description This handles the transfer of a single 32-bit word from the PC memory pointed to by 'Data' to the Interface connected to the on-board FGPA of the DIME Motherboard.

Notes This function should only be used in single card systems. In systems with more than one card it is strongly advised that the DIME_DMA functions are used instead.

6.26 DIME_DeviceControl

Syntax DWORD DIME_DeviceControl(DIME_HANDLE handle, DWORD ModuleNum, DWORD DeviceNum, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNum is the Module that is being addressed. Note modules are numbered from 0.

DeviceNum is the number of the device that is being addressed.

CmdMode: This argument is used to specify what particular aspect of device is to be controlled. There are no current command modes for this function.

Value: This argument is used to specify the action for a command mode.

Return Returns -1 on error.

Description This function is used to control certain aspects of the selected device.

6.27 DIME_DeviceControlPtr

Syntax DWORD DIME_DeviceControlPtr(DIME_HANDLE handle, DWORD ModuleNum, DWORD DeviceNum, DWORD CmdMode, void *pValue)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNum is the Module that is being addressed. Note modules are numbered from 0.

DeviceNum is the number of the device that is being addressed.

CmdMode: This argument is used to specify what particular aspect of the device is to be controlled. There are no current command modes for this function.

Value: This argument is used to specify the action for a command mode.

Return Returns NULL on error.

Description This function is used to control certain aspects of the card that cannot be controlled using DIME_DeviceControl.

6.28 DIME_DeviceStatus

Syntax DWORD DIME_DeviceStatus(DIME_HANDLE handle, DWORD ModuleNum, DWORD DeviceNum, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNum is the Module that is being addressed. Note that modules are numbered from 0.

DeviceNum is the number of the device that is being addressed.

CmdMode: This argument is used to specify what particular aspect of device status information is to be returned. Table 12 on page 36 gives details of the available command modes.

CmdMode	Description
dinfDEVICEIDCODE	This command mode returns the 32-bit hex-decimal JTAG device ID code for the device. See Table 14 on page 37 for details.
dinfDEVICETYPE	This command mode returns the device type for the device. See Table 13 on page 36 for details.
dinfXOFFSET	This command mode returns the x co-ordinate from the left most edge of the module image, which is used to position the device image correctly on the module image. This is useful when wishing to display an image of the device on the module.
dinfYOFFSET	This command mode returns the y co-ordinate from the bottom most edge of the module image, which is used to position the device image correctly on the module image. This is useful when wishing to display an image of the device on the module.
dinfWIDTH	This command mode returns the width of the device image. This is useful when wishing to display an image of the device on the module.
dinfHEIGHT	This command mode returns the height of the device image. This is useful when wishing to display an image of the device on the module.

Table 12: DIME_DeviceStatus CmdMode Argument Options

Return The return value is dependant upon the command mode. Returns -1 on error.

Description This function returns device status information.

Notes

Device Type	Description
dinfDEVICEBOOTABLE	The device is configurable.
dinfDEVICEBYPASS	The device is a bypass device.
dinfDEVICENOBOOT	The device cannot be configured.

Table 13: Device Types

JTAG device ID code	Description	JTAG device ID code	Description
0	Invalid arguments	didXCV50E	Xilinx Virtex E device V50
didXCV50	Xilinx Virtex device V50	didXCV100E	Xilinx Virtex E device V100
didXCV100	Xilinx Virtex device V100	didXCV200E	Xilinx Virtex E device V200
didXCV150	Xilinx Virtex device V100	didXCV300E	Xilinx Virtex E device V300
didXCV200	Xilinx Virtex device V200	didXCV400E	Xilinx Virtex E device V400
didXCV300	Xilinx Virtex device V300	didXCV600E	Xilinx Virtex E device V600
didXCV400	Xilinx Virtex device V400	didXCV1000E	Xilinx Virtex E device V1000
didXCV600	Xilinx Virtex device V600	didXCV1600E	Xilinx Virtex E device V1600
didXCV800	Xilinx Virtex device V800	didXCV2000E	Xilinx Virtex E device V2000
didXCV1000	Xilinx Virtex device V1000	didXCV2600E	Xilinx Virtex E device V2600
didXCV405EM	Xilinx Virtex EM device V405	didXCV3200E	Xilinx Virtex E device V3200
didXCV812EM	Xilinx Virtex EM device V812	didXC18V01	Xilinx 1800 PROMs V01
didXC2S50	Xilinx Spartan 2 S50	didXC18V02	Xilinx 1800 PROMs V02
didXC2S100	Xilinx Spartan 2 S100	didXC18V04	Xilinx 1800 PROMs V04
didXC2S150	Xilinx Spartan 2 S150	didXC18V256	Xilinx 1800 PROMs V256
didXC2S200	Xilinx Spartan 2 S200	didXC18V512	Xilinx 1800 PROMs V512
XC9536	Xilinx 9536 CPLD	XC9536XL	Xilinx 9536XL CPLD
XC9572	Xilinx 9572 CPLD	XC9572XL	Xilinx 9572XL CPLD
XC95108	Xilinx 95108 CPLD	XC95108XL	Xilinx 95108XL CPLD
XC95144	Xilinx 95144 CPLD	XC95144XL	Xilinx 95144XL CPLD
XC95216	Xilinx 95216 CPLD	XC95216XL	Xilinx 95216XL CPLD
XC95288	Xilinx 95288 CPLD	didXC2V40	Xilinx Virtex2 V40
didXC2V80	Xilinx Virtex2 V80	didXC2V250	Xilinx Virtex2 V250
didXC2V500	Xilinx Virtex2 V500	didXC2V1000	Xilinx Virtex2 V1000
didXC2V1500	Xilinx Virtex2 V1500	didXC2V2000	Xilinx Virtex2 V2000
didXC2V3000	Xilinx Virtex2 V3000	didXC2V4000	Xilinx Virtex2 V4000
didXC2V6000	Xilinx Virtex2 V6000	didXC2V8000	Xilinx Virtex2 V8000
didXC2V10000	Xilinx Virtex2 V10000	didXC2VP2	Xilinx Virtex2 Pro P2
didXC2VP4	Xilinx Virtex2 Pro P4	didXC2VP7	Xilinx Virtex2 Pro P7
didXC2VP20	Xilinx Virtex2 Pro P20	didXC2VP30	Xilinx Virtex2 Pro P30
didXC2VP40	Xilinx Virtex2 Pro P40	didXC2VP50	Xilinx Virtex2 Pro P50
didXC2VP70	Xilinx Virtex2 Pro P70	didXC2VP100	Xilinx Virtex2 Pro P100
didXC2VP125	Xilinx Virtex2 Pro P125		

Table 14: JTAG Device ID Codes

6.29 DIME_DeviceStatusPtr

Syntax void *DIME_DeviceStatusPtr(DIME_HANDLE handle, DWORD ModuleNum, DWORD DeviceNum, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNum is the Module that is being addressed. Note modules are numbered from 0.

DeviceNum is the number of the device that is being addressed.

CmdMode: This argument is used to specify what particular aspect of device status information is to be returned. Table 15 on page 38 gives details of the available command modes.

CmdMode	Description
dinfICONFILENAME	This command mode returns a string (char *), which is the icon filename for the device.
dinfDESCRIPTION	This command mode returns a string (char *), which is a short description of the device.

Table 15: DIME_DeviceStatusPtr CmdMode Argument Options

Return The return value is dependant upon the command mode. Returns NULL on error.

Description This function returns card status information that cannot be returned using DIME_DeviceStatus.

Notes If a pointer to a string is returned this string is only valid until the next call is made into the library. It is therefore advised that either the string is used directly or that it is copied for later use.

6.30 DIME_DMAClose

Syntax DWORD DIME_DMAClose(DIME_HANDLE handle, DIME_DMAHANDLE DMAhandle, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

DMAhandle is a valid DMA handle that was returned from DIME_DMAOpen.

Flags: This argument allows the DMA close process to be customised to suit your development requirements. Table 16 on page 39 gives details for the Flags.

Flags	Description
ddmaCLOSETERMINATE	This will immediately terminate any DMA transfer using this DMAhandle and then close the handle.
ddmaCLOSEWAITFORFINISH	This will wait for any active DMA transfer to finish before closing down the handle.

Table 16: DIME_DMAClose Flags Argument Options

Return Returns zero on success, non-zero otherwise.

Description This function closes down a valid DMAhandle. In doing this it de-allocates system resources. This function should be called after all DMA transfers have finished using this DMAhandle.

Notes After successfully calling this function the DMAhandle is no longer valid.

6.31 DIME_DMAControl

Syntax DWORD DIME_DMAControl(DIME_HANDLE handle, DIME_DMAHANDLE DMAChannel, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

DMAChannel is a valid DMA handle that was returned from DIME_DMAOpen.

CmdMode: This argument is used to specify which aspect of the open DMA channel is to be controlled. The available command modes are given in Table 17 on page 39.

CmdMode	Description
ddmaLOCALNOINC	Sets the DMA channel to have no incrementing for the local address during transfers. Returns 0 on success.
ddmaREMOTENOINC	Sets the DMA channel to have no incrementing for the remote address during transfers. Returns 0 on success.
ddmaLOCALINC	Sets the DMA channel to have incrementing for the local address during transfers. Returns 0 on success.

Table 17: DIME_DMAControl CmdMode Argument Options

CmdMode	Description
ddmaREMOTEINC	Sets the DMA channel to have incrementing for the remote address during transfers. Returns 0 on success.
ddmaTERMINATE	If the DMAChannel is a valid handle, the current DMA transfer for this channel is terminated. If DMAChannel=NULL. All DMA transfers with this card are terminated. Returns 0 on success.
ddmaTIMEOUT	Sets the Timeout value for the DMA channel to the number of milli-seconds specified by the value argument. The Timeout specifies the maximum period of inactivity that is acceptable during a DMA transfer. If this value is exceeded then the DMA transfer is deemed to have failed and is terminated. If the timeout value is set to zero then the timeout is effectively infinite. The default value for DMA timeouts is 5000milli-seconds which is set every time a DMA handle is created using DIME_DMAOpen. Returns 0 on success.
ddmaCURRCOUNT	This is used to set the address of where the current total of words transferred is to be stored.
ddmaPOLLED	This is used to set all future DMA transfers on this channel to be polled rather than use interrupts. When the DMA channel is opened the default state is that all transfers are polling transfers and not interrupt based. Returns 0 on success.
ddmaINTERRUPTS	This is used to set all future DMA transfers on this channel to be interrupt based rather than polling. When the DMA channel is opened the default state is that all transfers are polling transfers and not interrupt based. Note that polling transfers are in general faster than interrupt based transfers although interrupt based transfers use less CPU time. Returns 0 on success.

Table 17: DIME_DMAControl CmdMode Argument Options

Return Returns are dependant on the selected command mode.

Description This function is used to control DMA transfers for a particular channel.

6.32 DIME_DMAOpen

Syntax DIME_DMAHANDLE DIME_DMAOpen(DIME_HANDLE handle, DWORD DMAChannel, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

DMAChannel is used to specify the actual DMA channel on your hardware that is to be addressed. This is a bit wise argument. Please refer to the *Motherboard Reference Guide* for more details on what DMA channels are available for the particular hardware.

Flags: Used to specify special modes of opening the DMA channel. Currently there are none and this argument is not used.

Return This handle is used to control the desired DMA channel. Returns a valid DMA handle. Returns NULL otherwise.

Description DMA transfers are fast and efficient ways of transferring large quantities of data between the host PC and the card. This function checks the availability of the selected DMA channel and if suitable creates a handle that is required when performing the DMA data transfers.

6.33 DIME_DMARead

Syntax DWORD DIME_DMARead(DIME_HANDLE handle, DIME_DMAHANDLE DMAChannel, DWORD *DestData, DWORD SrcAddr, DWORD WordCount, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

DMAChannel: A valid DMA handle that was returned from DIME_DMAOpen.

DestData: This should be a pointer to the PC memory that receives data from the card. This should be 32-bit aligned.

SrcAddr: This is the source address of the data on the card that is to be read. If this argument is 0 the interface FPGAs FIFOs are to be read.

WordCount: This is the number of 32-bit words to be read.

Flags: This argument allows the DMA transfer to be customised to suit your development requirements. The flags are bit wise so multiple flags can be combined. Table 18 on page 41 gives details for the Flags. Obviously directly conflicting flags such as ddmaBLOCKING and ddmaNONBLOCKING should not be combined. In this case the flag that sets the bit will be taken. In this case the transfer would be set to be non blocking.

Flags	Description
ddmaBLOCKING	DIME_DMARead will not return until the DMA transfer is complete.
ddmaNONBLOCKING	DIME_DMARead will return before the DMA transfer is complete.

Table 18: DMA Transfer Functions Flags Argument Options

Flags	Description
ddmaPHYSICALADDR	This is only valid in multiple card systems where DMA transfer is between two cards. In this case the local address is actually a physical address and this flag should be used. Please check your *Motherboard Reference Guide* for details.
ddmaINITWORDADDR	This flag should be used when the initial word of the data is an address.

Table 18: DMA Transfer Functions Flags Argument Options

Return There are several possible returns for the DMA transfer. Table 19 on page 42 provides details.

DMA transfer function return values	Description
ddmaOK	The DMA transfer was a success.
ddmaINVALID_HANDLE	The DMA transfer could not begin since one of the handles where invalid.
ddmaMEM_ERROR	There are insufficient system resources to carry out the transfer. Please free some resources by closing down other applications and retry.
ddmaTIMEDOUT	The DMA transfer was terminated due to the transfer exceeding the specified timeout value.
ddmaINPROGRESS	The DMA transfer is in progress.
ddmaINTERRUPT_ERROR	The DMA transfer failed due to an error relating to the interrupts.
ddmaINVALID_MEM_HANDLE	The DMA transfer failed due to the locked down memory handle being invalid.

Table 19: DMA Transfer Functions Return Values

Description This handles the transfer of 'WordCount' 32-bit words from the Interface with the on-board FGPA of the DIME Motherboard to the memory pointed to by 'DestData'. The memory pointed to by 'DestData' does not need to be contiguous, as this function will handle the internal transfer and memory management.

Notes To achieve this first it locks down DestData and then performs the DMA transfer. On completion of the transfer irrespective of the result it then unlocks the DestData. So if your application performs several DMA reads then it is more efficient to use the DIME_DMAReadToLockedMem function.

6.34 DIME_DMAReadToLockedMem

Syntax DWORD DIME_DMAReadToLockedMem (DIME_HANDLE handle, DIME_DMAHANDLE DMAChannel, DIME_MEMHANDLE DestData, DWORD SrcAddr, DWORD WordCount, DWORD Flags);

Arguments handle is a valid handle to a DIME carrier card.

DMAChannel: A valid DMA handle that was returned from DIME_DMAOpen.

DestData: This should be a valid memory handle returned from DIME_LockMemory. This should be the 32-bit aligned location of the memory for the data is going to read from the card and stored in.

SrcAddr: This is the source address of the data on the card that is to be read. If this argument is 0 the interface FPGAs FIFOs are to be read.

WordCount: This is the number of 32-bit words to be read.

Flags: This argument allows the DMA transfer to be customized to suit your development requirements. The flags are bit wise so multiple flags can be combined. See Table 18 on page 41 for valid flags.

Return There are several possible returns for the DMA transfer. See Table 19 on page 42 for details.

Description This handles the transfer of 'WordCount' 32-bit words from the Interface with the on-board FPGA of the DIME Motherboard to the memory pointed to by 'DestData'. The memory pointed to by 'DestData' does not need to be contiguous, as this function will handle the internal transfer and memory management.

6.35 DIME_DMAStatus

Syntax DWORD DIME_DMAStatus(DIME_HANDLE handle, DIME_DMAHANDLE DMAChannel, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

DMAChannel is a valid DMA handle that was returned from DIME_DMAOpen or DMAChannel is ddmaALLDMACHANNELS. This is used under certain command modes to return card specific information on all DMA channels.

CmdMode: This argument is used to specify which aspect of the open DMA channel is being addressed. The available command modes are given inTable 20 on page 43.

CmdMode	Description
ddmaNUMCHANNELS	Used with ddmaALLDMACHANNELS. Returns the number of DMA channels available for this card.
ddmaREADFLAGS	Used with ddmaALLDMACHANNELS. Returns the number of readable DMA channels for this card. The return is a bit wise DWORD with a '1' in a particular bit position indicates that that channel is readable. E.g. a return of 0x00000003 means that channel numbers 1 and 2 are both readable channels.
ddmaWRITEFLAGS	Used with ddmaALLDMACHANNELS. Returns the number of writeable DMA channels for this card. The return is a bit wise DWORD with a '1' in a particular bit position indicates the channel is writeable. E.g. a return of 0x00000004 means that channel number 3 is a writeable channel.
ddmaREADANDWRITE	Used with ddmaALLDMACHANNELS. Returns the number of DMA channels that are both writeable and readable for this card. The return is a bit wise DWORD with a '1' in a particular bit position indicates the channel is both writeable and readable. E.g. a return of 0x00000001 means that channel number 1 is both a readable and writeable channel.

Table 20: DIME_DMAStatus CmdMode Argument Options

CmdMode	Description
ddmaREADABLE	Returns a 'I' if the DMAChannel is readable. Returns a '0' if not.
ddmaWRITABLE	Returns a 'I' if the DMAChannel is writeable. Returns a '0' if not.
ddmaACTIVE	Returns a 'I' if the DMAChannel is active. Returns a '0' if not.
ddmaINTERUPTABLE	Returns a 'I' if the DMAChannel is interruptible. Returns a '0' if not.
ddmaNONBLOCKINGSUPPORT	Returns a 'I' if the DMAChannel supports non-blocking DMA transfers. Returns a '0' if not.
ddmaLOCALINCFLAG	Returns a 'I' if the DMAChannel at the local side (the PC in the majority of cases) supports incremental addressing. Return a '0' if not.
ddmaREMOTEINCFLAG	Returns a 'I' if the DMAChannel at the remote side (the card in the majority of cases) supports incremental addressing. Returns a '0' if not.
ddmaLOCALNOINC	Returns a 'I' if the DMAChannel can do no incrementing on local addresses. Returns a '0' if not.
ddmaREMOTENOINC	Returns a 'I' if the DMAChannel can do no incrementing on remote addresses. Returns a '0' if not.
ddmaTIMEOUT	Returns the DMA timeout value for DMA transfers. This is defaulted to 5000 milli-seconds.
ddmaCURRCOUNT	Returns the current word count of any DMA transfer. This is useful feedback to the application. This can be used in multi-threaded applications to monitor the progress of the data transfer. In single threaded applications it can be used to return the total number of words transferred.
ddmaEMPTYFLAG	When EMPTY is high it indicates that there is no data waiting to be transferred to the FPGA, i.e. the FPGA application has read all the available data that has been transferred via DMA write operation. Returns the status of the EMPTY signal that is on the interface FPGA to the on-board FPGA Interface. When the EMPTY signal is high this function returns a 'I' otherwise it returns a '0'.
ddmaBUSYFLAG	When BUSY is high it indicates that the internal transfer buffer from the on board FPGA to the interface FPGA is full and cannot accept any more data. The user application should initiate a DMA read at this stage. Returns the status of the BUSY signal that is on the interface FPGA to the on-board FPGA Interface. When the BUSY signal is high this function returns a 'I' otherwise it returns a '0'.
ddmaREADEMPTY	Returns the status of the internal buffer between the on board FPGA and the interface FPGA interface. If there is no data in the read buffer to be accessed this will return I, however 0 will be returned if there is data waiting to be read.

Table 20: DIME_DMAStatus CmdMode Argument Options

CmdMode	Description
ddmaWRITEFULL	This returns a '1' when the internal buffer from the interface FPGA to the on-board FPGA is full and hence cannot accept any more data from the user application. The user application must therefore wait until the FPGA reads data before any more data will be transferred.
ddmaWAITFORFINISH	This returns only when the current DMA transfer over this channel is finished.

Table 20: DIME_DMAStatus CmdMode Argument Options

Return Returns are dependant on the selected command mode.

Description This function returns status information on various DMA operations for the selected DMA channel.

6.36 DIME_DMAWrite

Syntax DWORD DIME_DMAWrite(DIME_HANDLE handle, DIME_DMAHANDLE DMAChannel, DWORD *SrcData, DWORD DestAddr, DWORD WordCount, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

DMAChannel: A valid DMA handle that was returned from DIME_DMAOpen.

SrcData: This should be a pointer to the PC memory that contains the data to be written to the card. This should be 32-bit aligned.

DestAddr: This is the destination address on the card that the data is to be written to. If this argument is 0 the interface FPGAs FIFOs are assumed.

WordCount: This is the number of 32-bit words to be written.

Flags: This argument allows the DMA transfer to be customized to suit your development requirements. The flags are bit wise so multiple flags can be combined. See Table 18 on page 41 for valid flags.

Return There are several possible returns for the DMA transfer. See Table 19 on page 42 for details.

Description This handles the transfer of 'WordCount' 32-bit words from the PC memory pointed to by SrcData to the Interface connected to the on board FPGA of the DIME Motherboard. The memory pointed to by SrcData does not need to be contiguous, as this function will handle the internal transfer and memory management.

Notes To achieve this first it locks down SrcData and then performs the DMA transfer. On completion of the transfer irrespective of the result it then unlocks the SrcData. So if your application performs several DMA writes then it is more efficient to use the DIME_DMAWriteFromLockedMem function.

6.37 DIME_DMAWriteFromLockedMem

Syntax DWORD DIME_DMAWriteFromLockedMem (DIME_HANDLE handle, DIME_DMAHANDLE DMAChannel, DIME_MEMHANDLE SrcData, DWORD DestAddr, DWORD WordCount, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

DMAChannel: A valid DMA handle that was returned from DIME_DMAOpen.

SrcData: This should be a valid memory handle returned from DIME_LockMemory. This should be the 32-bit aligned location of the memory of the data that is to be written to the card.

DestAddr: This is the destination address on the card that the data is to be written to. If this argument is 0 the interface FPGAs FIFOs are assumed.

WordCount: This is the number of 32-bit words to be written.

Flags: This argument allows the DMA transfer to be customized to suit your development requirements. The flags are bit wise so multiple flags can be combined. See Table 18 on page 41 for valid flags.

Return There are several possible returns for the DMA transfer. See Table 19 on page 42 for details.

Description This handles the transfer of 'WordCount' 32-bit words from the PC memory pointed to by SrcData to the Interface connected to the on board FPGA of the DIME motherboard. The memory pointed to by SrcData does not need to be contiguous, as this function will handle the internal transfer and memory management.

6.38 DIME_GetCurrentHandle

Syntax DIME_HANDLE DIME_GetCurrentHandle(void)

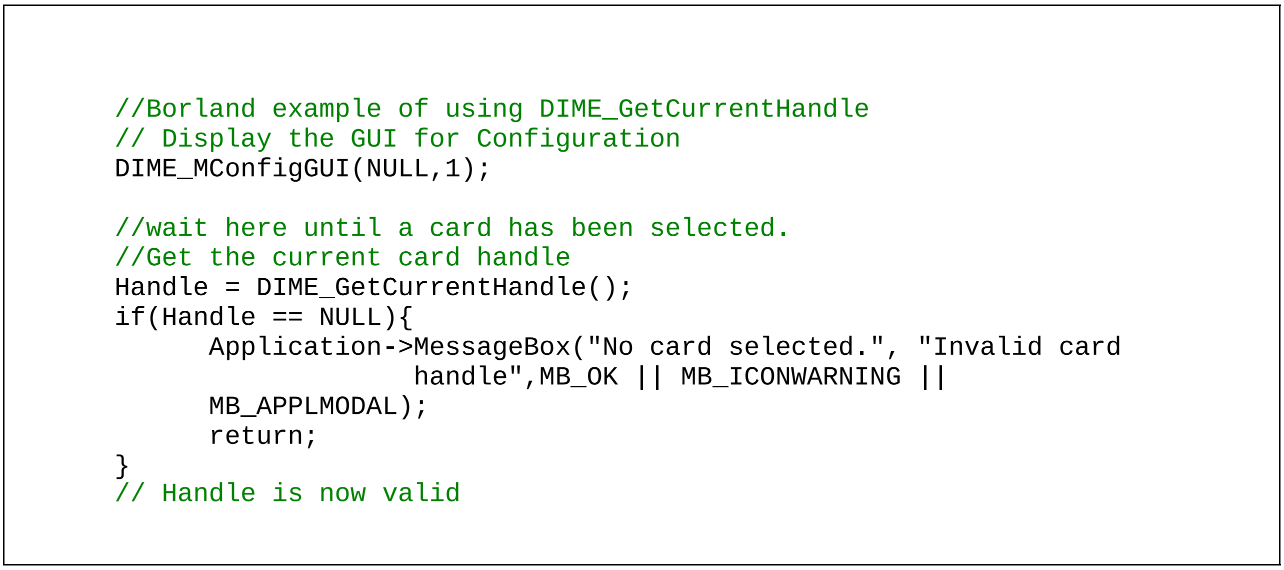
Arguments N/A

Return Returns the currently selected card handle from the probe utility. If no card is selected in the probe utility then the return will be NULL.

Description This function gets the current card handle from the probe utility. Note that the probe utility must be launched and a card selected.

Notes The nueym.lib (nueymomf.lib for Borland projects) needs to included in your design.

Example



```
//Borland example of using DIME_GetCurrentHandle
// Display the GUI for Configuration
DIME_MConfigGUI(NULL,1);

//wait here until a card has been selected.
//Get the current card handle
Handle = DIME_GetCurrentHandle();
if(Handle == NULL){
      Application->MessageBox("No card selected.", "Invalid card
                   handle",MB_OK || MB_ICONWARNING ||
      MB_APPLMODAL);
      return;
}
// Handle is now valid
```

Figure 5: DIME_GetCurrentHandle

6.39 DIME_GetError

Syntax void DIME_GetError(DIME_HANDLE handle, DWORD *ErrNumber, char* ErrString)

Arguments handle is a valid handle to a DIME carrier card.

ErrNumber is a pointer to the memory where the error number will be returned.

ErrString is a pointer to the start of the memory where the error string will be written.

Return N/A

Description This function allows you to gather information relating to the last error that occurred in the system. This is particularly useful if the system dialogue boxes have been disabled using DIME_SystemControl.

Notes You can allocate any memory required by either the ErrNumber or the ErrString arguments. 1000 characters is the maximum error string length.

Example

```
//Example of using DIME_GetError
//Note DIME_CardStatusPtr is deliberately getting called with an
//invalid command mode to generate an error.
if( DIME_CardStatusPtr(hCard,dinfYOFFSET)==NULL)
{
     DWORD ErrorNumber;
     char ErrorString[1000];
     DIME_GetError(hCard,&ErrorNumber,ErrorString);
     printf("Error number: %d.\n",ErrorNumber);
     printf(ErrorString);
}
```

Figure 6: DIME_GetError Example

6.40 DIME_GetLocateVersionNumber

Syntax DWORD DIME_GetLocateVersionNumber(void)

Arguments N/A

Return Returns the version number of the locate unit within the FUSE software. Returns a 0 on an error.

Description Gets the version number of the locate unit within the FUSE software.

6.41 DIME_GetVersionNumber

Syntax DWORD DIME_GetVersionNumber(DIME_HANDLE CardHandle)

Arguments CardHandle: If this argument is NULL then the version number for the FUSE SDL software is returned. If this is a valid card handle then the returned version number is the version number for the FUSE card driver.

Return Returns a version number.

Description Gets the version number of the installed software.

6.42 DIME_InterruptControl

Syntax DWORD DIME_InterruptControl(DIME_HANDLE handle, DWORD InterruptFlags, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

InterruptFlags: This is used to specify which interrupt the function is to address. See Table 23 on page 51 for details of the available InterruptFlags.

Note: Currently the only parameter available is dintONBOARDFPGA except for the BenONE. If a BenONE is used then either dintONBOARDFPGA or dintMODULE1 may be used. Both of these address the interrupt on the module FPGA.

CmdMode: This argument is used to specify what function is to be performed. Table 21 on page 49 gives details of the available command modes.

CmdMode	Description
dintENABLE	This is enable the selected interrupt. Returns zero on success, non-zero otherwise.
dintDISABLE	This is used to disable the selected interrupt. Returns zero on success, non-zero otherwise.
dintWAIT	This causes the function to wait for the selected interrupt to occur providing that the interrupt has been enabled. Returns zero on success, non-zero otherwise.

Table 21: DIME_InterruptControl CmdMode Argument Options

Value: This is used in the control to specify the type of interrupts to be used. This is only used in the dintWAIT command mode. Table 22 on page 49 gives details of the available Values.

Value	Description
dintBLOCKING	Blocks the threads execution until the interrupt occurs.

Table 22: DIME_InterruptControl Value Argument Options

Return Returns all '-1' on error, otherwise the return is dependant on the command mode.

Description This function is used to control the interrupts on the card. It allows the interrupts to be enabled, disabled and allows your program to wait and act on interrupts generated by your design. All interrupts are genuine hardware interrupts.

Example

```
//Sample interrupt code. Obviously your design needs to use this
pin.
//enable the interrupt
if( (Answer=DIME_InterruptControl(hCard, dintONBOARDFPGA,
      dintENABLE, dintBLOCKING)) == 0)
      printf("On board FPGA Interrupt Enabled.\n");
else
      printf("failed to enable the on baord FPGA interrupt.\n");

//Wait for the interrupt
if( (Answer=DIME_InterruptControl(hCard,dintONBOARDFPGA,dintWAIT,
      dintBLOCKING))==0)
      printf("On board interrupt received.\n");
else
      printf("Error while waiting for the on board FPGA
      interrupt.\n");

//Disable the interrupt
if((Answer=DIME_InterruptControl(hCard,dintONBOARDFPGA,
      dintDISABLE, dintBLOCKING)) ==0)
      printf("Disabled the on board FPGA interrupt.\n");
else
      printf("Failed to disable the on board FPGA interrupt.\n");
```

Figure 7: Interrupt Example using DIME_InterruptControl

6.43 DIME_InterruptStatus

Syntax DWORD DIME_InterruptStatus(DIME_HANDLE handle, DWORD InterruptFlags, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

InterruptFlags: This is used to specify which interrupt the function is to address. Table 23 on page 51 gives details of the available InterruptFlags.

InterruptFlags	Description
dintONBOARDFPGA	This is on board FPGA interrupt pin.
dintMODULE1	This is the interrupt for Module 1.
dintMODULE2	This is the interrupt for Module 2.
dintMODULE3	This is the interrupt for Module 3.
dintMODULE4	This is the interrupt for Module 4.
dintALL	This is the flag for all the above interrupts.

Table 23: InterruptFlags Argument Options

Note: Currently the only parameter available is dintONBOARDFPGA except for the BenONE. If a BenONE is used then either dintONBOARDFPGA or dintMODULE1 may be used. Both of these address the interrupt on the module FPGA.

CmdMode: This argument is used to specify what function is to be performed. Table 24 on page 51 gives details of the available command modes.

Condoned	Description
dintFLAGS	This is used to determine if an interrupt has occurred. Returns 1 if an interrupt has occurred. 0 otherwise.
dintAVAILABLE	Allows the user to check whether an interrupt for this card exists. Returns a one in the relevant bit position if the interrupt is available. Returns a zero in the relevant bit position otherwise.
dintPINVALUE	This returns the value on the interrupt pin. If the interrupt pin for the selected interrupt is high then this returns 1 in the relevant bit position otherwise it returns 0.

Table 24: DIME_InterruptStatus CmdMode Argument Options

Return Returns all '-1' on error, otherwise the return is dependant on the command mode.

Description This function is used to get the status of the interrupts on the card. There are three command modes available. The first dintFLAGS is used to determine if an interrupt has occurred since the specified interrupt was enabled. dintAVAILABLE is used to check whether the specified interrupt is valid for this particular card. Finally dintPINVALUE is used to return the current value of the chosen interrupt pin.

Example

```
//Example Interrupt code
if((Answer=DIME_InterruptStatus(hCard,dintALL,dintAVAILABLE))==0)
      printf("No Interrupts are available for this card.\n");
if((Answer=DIME_InterruptStatus(hCard,dintONBOARDFPGA,
      dintAVAILABLE))==dintONBOARDFPGA)
printf("The on-board FPGA interrupt is available for this
      card.\n");
if((Answer=DIME_InterruptStatus(hCard,dintMODULE1,dintAVAILABLE))
      ==dintMODULE1)
printf("The MODULE 1 interrupt is available for this card.\n");
if((Answer=DIME_InterruptStatus(hCard,dintMODULE2,dintAVAILABLE))
       == dintMODULE2)
printf("The MODULE 2 interrupt is available for this card.\n");
if((Answer=DIME_InterruptStatus(hCard,dintMODULE3,dintAVAILABLE))
      == dintMODULE3)
printf("The MODULE 3 interrupt is available for this card.\n");
if((Answer=DIME_InterruptStatus(hCard,dintMODULE4,dintAVAILABLE))
      == dintMODULE4)
printf("The MODULE 4 interrupt is available for this card.\n");

printf("The current state of the on-board FPGA interrupt pin is
%d\n",DIME_InterruptStatus(hCard,dintONBOARDFPGA,dintPINVALUE));

printf("A 0 indicates that an interrupt has not occured, 1
indicates that one has: %d\n",
DIME_InterruptStatus(hCard,dintONBOARDFPGA,dintFLAGS));

printf("The value on the Virtex Int Pin is %d.\n",
DIME_VirtexIntPin(hCard));
```

Figure 8: Examples of using DIME_InterruptStatus

6.44 DIME_JTAGControl

Syntax DWORD DIME_JTAGControl(DIME_HANDLE handle, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

CmdMode: This argument is used to specify which particular aspect of the JTAG chain is to be returned. Table 25 on page 53 gives details.

CmdMode	Description
djtagCONFIGSPEED	The will set the current speed that the cards JTAG chain is running at. The value argument is then taken and the cards chain is set as close to the specified value as possible. This will return a zero on success. Please consult your *Motherboard Reference Guide* for further details on the JTAG chain.

Table 25: DIME_JTAGControl CmdMode Argument Options

Value is used to specify the value that the chain is to be driven at. Table 26 on page 53 provides details.

Value	Description
djtagDEFAULTSPEED	This is the default speed for the JTAG chain on the card. Please consult your motherboards used guide for more details.
djtagMAXSPEED1	This will attempt to set the cards JTAG chain to 1MHz.
djtagMAXSPEED2	This will attempt to set the cards JTAG chain to 2MHz.
djtagMAXSPEED5	This will attempt to set the cards JTAG chain to 5MHz.
djtagMAXSPEED10	This will attempt to set the cards JTAG chain to 10MHz.
djtagMAXSPEED20	This will attempt to set the cards JTAG chain to 20MHz.
djtagMAXSPEED30	This will attempt to set the cards JTAG chain to 30MHz.
djtagMAXSPEED40	This will attempt to set the cards JTAG chain to 40MHz.
djtagMAXSPEED50	This will attempt to set the cards JTAG chain to 50MHz.
djtagMAXSPEED60	This will attempt to set the cards JTAG chain to 60MHz.
djtagMAXSPEED70	This will attempt to set the cards JTAG chain to 70MHz.
djtagMAXSPEED80	This will attempt to set the cards JTAG chain to 80MHz.
djtagMAXSPEED90	This will attempt to set the cards JTAG chain to 90MHz.
djtagMAXSPEED100	This will attempt to set the cards JTAG chain to 100MHz.

Table 26: JTAG Return Descriptions

Return This function returns '-1' on failure. The function returns 0 on success.

Description This controls the JTAG chain on the card.

Notes Once this function has been called calling the DIME_JTAGStatus function will return the speed at which the JTAG chain has been set.

6.45 DIME_JTAGStatus

Syntax DWORD DIME_JTAGStatus(DIME_HANDLE handle, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

CmdMode: This argument is used to specify which particular of the JTAG chain is to be returned. Table 27 on page 54 provides details.

CmdMode	Description
djtagCONFIGSPEED	The will return the current speed that the cards JTAG chain is running at. E.g. If the JTAG chain is being driven at its default speed it will return djtagDEFAULTSPEED. Otherwise it will return the closest maximum speed of the chain. For example with the Ballynuey this would return djtagMAXSPEED10. See Table 26 on page 53 for details. Please consult your *Motherboard Reference Guide* for further details on the JTAG chain.

Table 27: DIME_JTAGStatus CmdMode Argument Options

Return This function returns '-1' on failure. Otherwise the return is dependant upon the command mode.

Description This returns status information regarding the JTAG chain on the card.

6.46 DIME_LoadCardDefinition

Syntax DWORD DIME_LoadCardDefinition (DIME_HANDLE handle, const char *Filename, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

Filename is the filename that the card definition is to be saved to.

Flags: This argument is used to customize the loading of the card definition file. Currently there are no flags and this argument is not used. 0 Should be used.

Return Returns 0 on success. Returns non-zero otherwise.

Description This function loads a valid card definition file. By doing this it assigns bitfiles/streams to devices.

Notes If the dcfgEMBEDALLBITS flag was used when creating the card definition file then the bitfiles that where embedded into the card file will be loaded up into memory.

6.47 DIME_LoadSystemDefinition

Syntax DIME_HANDLE *DIME_LoadSystemDefinition (const char *SysFilename, DWORD *CardCount, LOCATE_HANDLE **Locate_handles, DWORD Flags)

Arguments SysFilename: This is the name of the system definition file that is to be loaded intomemory.

CardCount: This is a memory location to which the number of cards that the loaded system contains is returned.

Locate_handles is a pointer to an array that will return with the locate handles for each card that the loaded system has opened.

Flags: This argument is used to customize the loading of the system definition file. Currently there are no flags and this argument is not used. 0 Should be used.

Return Returns a pointer to an array containing the card handles for the cards that have been opened. Also via the Locate_handles argument the locate handles are returned and via the CardCount argument the number of cards in the system is returned. On error returns NULL.

Description This function takes a valid system definition file and loads it up. In doing this it locates all the cards in the system and opens each of these cards. Once opened the bitfiles/streams are assigned to every device in every card.

Notes To successfully load a system definition file the card definition files for each card in the system must be in the same location as when the system definition file was created.

6.48 DIME_LocateCard

Syntax LOCATE_HANDLE DIME_LocateCard(int LocateType, DWORD MBType, void* LocateTypeArgs, DWORD DriverVersion, DWORD Flags)

Arguments LocateType: This is the interface that the locate is to be performed over. The Table 28 on page 55 provides further details.

LocateType	Description
dlPCI	Searches for all Nallatech cards over the PCI interface.
dlUSB	Searches for all Nallatech cards over the USB interface.
dlTCPIP	Searches for all Nallatech cards over a network. (note that a server application has to be running on the destination machine)
dlCITRIX	Searches for all Nallatech cards over a Citrix environment
dlETHERNET	Searches for all Nallatech cards that have an Ethernet module.
dlVME	Searches for all Nallatech cards over the VME bus.

Table 28: Locate Types

MBType: This argument is used to specify which particular Nallatech motherboard is to be located (i.e. the motherboard type). Table 29 on page 55 gives details for the MBType.

MBType	Description
mbtALL	All Nallatech motherboards.
mbtNONE	No motherboard type not recognized. Not valid for this function.
mbtTHEBALLYINX	The Ballyinx
mbtTHEBALLYNUEY	The Ballynuey
mbtTHEBALLYNUEY2	The Ballynuey2
mbtTHEBALLYNUEY3	The Ballynuey3

Table 29: Motherboard Types

MBType	Description
mbtTHEBENERA	The BenERA
mbtTHESTRATHNUEY	The Strathnuey

Table 29: Motherboard Types

LocateTypeArgs: This argument is used to provide any specific additional information that is required to locate a card over a specified interface. Table 30 on page 56 details what information should be provided dependant on the interface.

LocateType	LocateTypeArgs
dlPCI	NULL
dlUSB	NULL
dlTCPIP	DIME_TCPIP
dlCITRIX	DIME_CITRIX
dlETHERNET	DIME_ETHERNET
dlVME	DIME_VME

Table 30: DIME_LocateCard LocateType Argument Options

DriverVersion: This argument is used to specify a particular software driver that is to be used when controlling the particular card. This is only required for advanced users. If the specific driver version number is known then this number can be used. Otherwise an option from Table 31 on page 56 should be used.

DriverVersion	Description
dldrDEFAULT	This locates the latest driver installed on your system for each card found.
dldrALL	This locates all drivers installed on your system for each card found.

Table 31: DIME_LocateCard DriverVersion Argument Options

Flags: This argument allows the locate process to be customized to suit your development requirements. Table 32 on page 56 gives details for the Flags.

Flags	Description
dlDEFAULT	This is the default option for the locate. It does not get the serial number from the cards.
dlSERIALNUM	Since getting the serial number from all the cards is a lengthy (approximately a second per card) process this information is not requested in the default option. If the serial number is required then specifying this flag will bring back the serial number for all cards.

Table 32: DIME_LocateCard Flags Argument Options

Return Returns a handle to information pertaining to the detected cards. Returns NULL on failure. The return type LOCATE_HANDLE is defined as a void pointer.

Description This function must be called before all other functions. It searches the specified interface for the specified Nallatech motherboards and returns a handle, which is subsequently used to open a chosen card.

Example See Figure 4 on page 25.

6.49 DIME_LocateStatus

Syntax DWORD DIME_LocateStatus(LOCATE_HANDLE handle, DWORD CardNumber, DWORD CmdMode)

Arguments handle is a valid locate handle.

CardNumber is the selected cards index. This can be NULL for certain command modes.

CmdMode is the command mode for the status function. This is used to specify what particular piece of information is required. Table 33 on page 57 gives details for the CmdMode argument.

CmdMode	Description
dINUMCARDS	This command mode returns the number of cards found by the locate. No card number is required when this command mode is used.
dIMBTYPE	Returns the motherboard type of the selected card.
dIINTERFACE	Returns the interface type for the selected card.
dISERIALNUMBER	Returns the serial number for the selected card.
dIDRIVERVERSION	Returns the software driver version number that will be used to control the selected card when it is opened (DIME_OpenCard).

Table 33: DIME_LocateStatus CmdMode Argument Options

Return Returns a DWORD that is dependant on the CmdMode argument. Returns 0xFFFFFFFF on error.

Description This function is used by the developer upon a successful return from a DIME_LocateCard function call to gather information on what has been located. This is normally required for systems that contain multiple cards over various interfaces. This information is then used to ensure that the desired card is opened and interfaced with.

```
#include <dimesdl.h> //This is held in the include directory
                        within FUSE.
#include <stdio.h>
int main(int argc, char* argv[])
{
LOCATE_HANDLE hLocate;
DWORD NumOfCards,LoopCntr;
//Locate the Cards on the PCI interface
hLocate=DIME_LocateCard(dlPCI,mbtALL,NULL,dldrDEFAULT,dlDEFAULT);

//Determine how many Nallatech cards have been found.
NumOfCards = DIME_LocateStatus(hLocate,0,dlNUMCARDS);
printf("%d Nallatech card(s) found.\n", NumOfCards);

//Get the details for each card detected.
for (LoopCntr=1; LoopCntr<=NumOfCards; LoopCntr++){
printf("Details of card number %d, of %d:\n",LoopCntr,NumOfCards);
printf("\tThe card driver for this card is a%s.\n",
      (char*)DIME_LocateStatusPtr(hLocate,LoopCntr,
      dlDESCRIPTION));
printf("\tThe cards motherboard type is %d.\n",
      DIME_LocateStatus(hLocate,LoopCntr,dlMBTYPE));
}
//Finally close the locate down.
DIME_CloseLocate(hLocate);
return 0;
}
```

Figure 9: Getting Information on the Located Cards

6.50 DIME_LocateStatusPtr

Syntax void* DIME_LocateStatusPtr(LOCATE_HANDLE handle, DWORD CardNumber, DWORD CmdMode)

Arguments handle is a valid locate handle.

CardNumber is the selected cards index. This can be NULL for certain command modes.

CmdMode is the command mode for the status function. This is used to specify what particular piece of information is required. Table 34 on page 59 provides details for the CmdMode argument.

CmdMode	Description
dIDESCRIPTION	This returns a pointer of type CHAR to a string that is a short description of the software driver for the chosen card.

Table 34: DIME_LocateStatusPtr CmdMode Argument Options

Return See Table 34 on page 59 since the return dependant on CmdMode.

Description This function is used by the developer upon a successful return from a DIME_LocateCard function call to gather information on what has been located. This is normally required for systems that contain multiple cards over various interfaces. This information is then used to ensure that the desired card is opened and interfaced with.

Notes Copy the string into your own programs memory space immediately after the function returns since the pointer may only be valid until the next call into the library.

Example See Figure 9 on page 58.

6.51 DIME_LockMemory

Syntax DIME_MEMHANDLE DIME_LockMemory(DIME_HANDLE handle, unsigned int *Data, DWORD Length)

Arguments handle is a valid handle to a DIME carrier card.

Data: This a pointer to the start of the data that is required to be locked down.

Length: This is the byte size of the memory that is to be locked down.

Return Returns a null pointer on error. On success returns a memory handle.

Description When performing PCI DMA transfers the physical memory that your data resides in must be locked down by the kernel. This function does exactly this and returns a handle to the locked down memory that can then be passed into the DMA functions.

Notes Locking down and unlocking memory takes a significant amount of time and should therefore be minimized. Please refer to the DMA examples for further information on efficient memory usage.

When dealing with transferring data between multiple cards, one of which is connected via the USB interface, the memory should be locked down using the PCI card handle. This allows the one memory handle to be used with both cards and saves needless data transfers.

Locking memory prevents the OS kernel from moving the memory around and hence decreases the kernel efficiency. Therefore locking down one large segment of memory is better practice than locking down several smaller segments.

Example

```
DWORD DataArray[256];
DIME_MEMHANDLE hMem;
DWORD i;

//Put data in the array
for(i=0;i<(sizeof(DataArray)/sizeof(DWORD));i++)
      DataArray[i]=i;

//Lock down the memory
if((hMem=DIME_LockMemory(hCard,DataArray,(sizeof(DWORD)*
      sizeof(DataArray))))==NULL)
      printf("Unable to lock down the data.\n");
else
      printf("Data locked down.\n");

//Do DMA transfer

//Unlock the memory
if(DIME_UnLockMemory(hCard,hMem)==0)
      printf("Data unlocked.\n");
else
      printf("Unable to unlock the memory.\n");
```

Figure 10: Locking and Unlocking Memory for DMA Transfers

6.52 DIME_MConfigGUI

Syntax void DIME_MConfigGUI(void * handle, DWORD ShowFlag)

Arguments handle is a valid handle to a DIME carrier card.

ShowFlag is a flag for initial condition of the form.

Return N/A

Description Calling this function opens the FUSE Probe Tool provided. It is displayed as a separate window and can handle all reconfiguration operations.

When 'ShowFlag' is 0 then the window is created but not visible, otherwise the window will be displayed when this function is called.

Notes The nueym.lib (for Microsoft Visual C++) or nueyomf.lib (for Borland C++ builder) needs to be included in your design when using this function. These libraries are installed in the include directory of the FUSE software.

6.53 DIME_MConfigGUIExit

Syntax void DIME_MConfigGUIExit()

Arguments N/A

Return N/A

Description This function is used to clean up any memory used in the DIME_MConfigGUI.

Notes This function should only be used when you have finished using the Multi config GUI.

6.54 DIME_MemConfigDevice

Syntax DWORD DIME_MemConfigDevice (DIME_HANDLE handle, DWORD *Bitstream, DWORD ByteLength, DWORD ModuleNumber, DWORD ModuleDeviceNumber, DWORD *Progress, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

Bitstream is the start of the Bitstream that is used to configure the device held in PC memory.

ByteLength: This argument specifies the length of the Bitstream in bytes.

ModuleNumber is the Module that is being addressed.

ModuleDeviceNumber is the selected device within the select module.

Progress should point to a variable that will be updated with the actual position in the configuration. The position in the configuration file is expressed as a percentage (0 - 100). This is only useful in multi-threaded applications and may point to a valid location or NULL in single threaded applications.

Flags: This argument is used to control the configuration of the on board device. Currently there are no flags and this argument is ignored.

Return This function has several possible returns. Please see Table 8 on page 27 for details.

Description This function configures the specified device with the specified bitfile.

Notes The bitfile must be configured to use the JTAG clock for configuration rather than the default of the CCLK.

6.55 DIME_MemConfigOnBoardDevice

Syntax DWORD DIME_MemConfigOnBoardDevice (DIME_HANDLE handle, DWORD *Bitstream, DWORD ByteLength, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

Bitstream is the start of the bitstream that is used to configure the device held in PC memory.

ByteLength: This argument specifies the length of the bitstream in bytes.

Flags: This argument is used to control the configuration of the on-board device. Currently there are no flags and this argument is ignored.

Return This function has several possible returns. Please see Table 8 on page 27 for details.

Description This function configures the on-board FPGA of the targeted card in the same manner as DIME_ConfigOnBoardDevice. The difference is that this function takes the Bitstream used to configure the device directly from memory and not from a file.

Notes The bitfile must be configured to use the JTAG clock for configuration rather than the default of the CCLK.

6.56 DIME_MemReadbackDevice

Syntax DWORD DIME_MemReadbackDevice (DIME_HANDLE handle, DWORD *Bitstream, DWORD ByteLength, DWORD ModuleNumber, DWORD DeviceNumber, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

Bitstream is a pointer to where you want the readback bitstream to be stored.

ByteLength: This argument specifies the length of the Bitstream to readback.

ModuleNumber is the specific module you want to target.

DeviceNumber is the specific device you want to target.

Flags: This argument is used to control the configuration of the on-board device. Currently there are no flags and this argument is ignored.

Return This function returns zero on success.

Description This function is used to either readback the configuration registers of the FPGA or readback the state of the CLBs, IOBs etc. Note, to readback the state of all CLBs, IOBs etc you must use the CAPTURE_XXX where XXX is the type of FPGA.

6.57 DIME_MemReadbackOnBoardDevice

Syntax DWORD DIME_MemReadbackOnBoardDevice (DIME_HANDLE handle, DWORD *Bitstream, DWORD ByteLength, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

Bitstream is a pointer to where you want the readback bitstream to be stored.

ByteLength: This argument specifies the length of the Bitstream to readback.

Flags: This argument is used to control the configuration of the on-board device. Currently there are no flags and this argument is ignored.

Return This function returns zero on success.

Description This function is used to either readback the configuration registers of the FPGA or readback the state of the CLBs, IOBs etc. Note, to readback the state of all CLBs, IOBs etc you must use the CAPTURE_XXX where XXX is the type of FPGA.

6.58 DIME_Miscioctl

Syntax DWORD DIME_Miscioctl(DIME_HANDLE handle, DWORD CMD, DWORD *Arg1, DWORD *Arg2, DWORD *Arg3, DWORD *Arg4, void *Arg5)

Arguments handle is a valid handle to a DIME carrier card.

CMD: The command to be performed. Currently there are no commands.

Arg1, Arg2, Arg3, Arg4 and Arg5 are all dependant upon the command.

Return The return is dependant upon the selected command.

Description This function is used to control and return status information for the miscellaneous I/O.

6.59 DIME_ModuleControl

Syntax DWORD DIME_ModuleControl(DIME_HANDLE handle, DWORD ModuleNum, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNum is the module that is being addressed. Note that modules are numbered from 0.

CmdMode: This argument is used to specify what particular aspect of module is to be controlled.

CmdMode	Description
dinfPRIMTEMPALERTMAX	This command mode is used to set the maximum temperature level for the temperature alert signal on the primary FPGA. Once set if the FPGA die temperature exceeds this temperature the temperature alert signal is triggered. Note that the power on default setting for this temperature is 255 degrees Celsius. Value should be the integer value that the maximum alert should be set to in degrees Celsius. NOTE: This function sets the maximum temperature for the alert signal to trigger. The alert signal is connected to the FPGA on the module, so if the user wants to trigger an event at a certain temperature then the user will have to use the alert signal in their design.
dinfSECTEMPALERTMAX	This command mode is used to set the maximum temperature level for the temperature alert signal on the secondary FPGA. Once set if the FPGA die temperature exceeds this temperature the temperature alert signal is triggered. Note that the power on default setting for this temperature is 255 degrees Celsius. Value should be the integer value that the maximum alert should be set to in degrees Celsius. NOTE: This function sets the maximum temperature for the alert signal to trigger. The alert signal is connected to the FPGA on the module, so if the user wants to trigger an event at a certain temperature then the user will have to use the alert signal in their design.
dinfPRIMTEMPALERTMIN	This command mode is used to set the minimum temperature level for the temperature alert signal on the primary FPGA. Once set if the FPGA die temperature falls below this temperature the temperature alert signal is triggered. Note that the power on default setting for this temperature is 0 degrees Celsius. Value should be the integer value that the minimum alert should be set to in degrees Celsius. Note: Not all temperature sense devices have this capability. Consult your *Module Reference Guide* or the *Temperature Sense Device Datasheet* to confirm if your module has this capability.
dinfSECTEMPALERTMIN	This command mode is used to set the minimum temperature level for the temperature alert signal on the secondary FPGA. Once set if the FPGA die temperature falls below this temperature the then the temperature alert signal is triggered. Note that the power on default setting for this temperature is 0 degrees Celsius. Value should be the integer value that the minimum alert should be set to in degrees Celsius. Note: Not all temperature sense devices have this capability. Consult your module user guide or the temperature sense device data sheet to confirm if your module has this capability.
dinfTEMPALERTCLEAR	This clears the temperature alert signal if set. Note that if either the maximum or minimum temperature limits are still exceeded then the alert signal will immediately be set. Value should to set to 0.

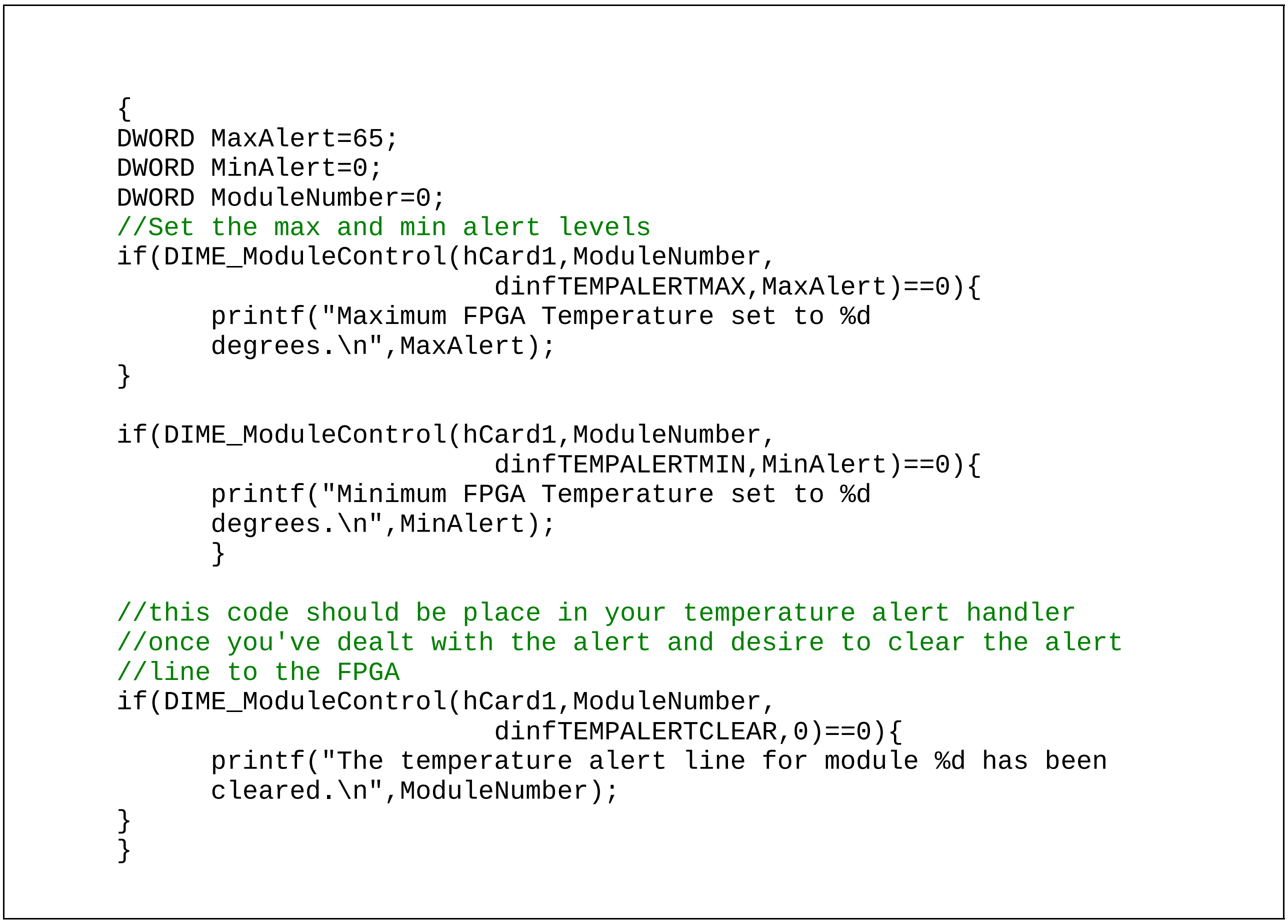
Table 35: DIME_ModuleControl CmdMode Argument Options

Value: This argument is command mode specific.

Return Returns -1 on error.

Description This function is used to control certain aspects of the selected module. Note, for the temperature functions, not all motherboards support this feature. Check your *Motherboard Reference Guide* if you are unsure if your hardware supports this.

Example



```
{
DWORD MaxAlert=65;
DWORD MinAlert=0;
DWORD ModuleNumber=0;
//Set the max and min alert levels
if(DIME_ModuleControl(hCard1,ModuleNumber,
                        dinfTEMPALERTMAX,MaxAlert)==0){
      printf("Maximum FPGA Temperature set to %d
      degrees.\n",MaxAlert);
}

if(DIME_ModuleControl(hCard1,ModuleNumber,
                        dinfTEMPALERTMIN,MinAlert)==0){
      printf("Minimum FPGA Temperature set to %d
      degrees.\n",MinAlert);
      }

//this code should be place in your temperature alert handler
//once you've dealt with the alert and desire to clear the alert
//line to the FPGA
if(DIME_ModuleControl(hCard1,ModuleNumber,
                        dinfTEMPALERTCLEAR,0)==0){
      printf("The temperature alert line for module %d has been
      cleared.\n",ModuleNumber);
}
}
```

Figure 11: Setting the Maximum and Minimum Temperature Alert Limits and Clearing an Alert

6.60 DIME_ModuleControlPtr

Syntax DWORD DIME_ModuleControlPtr(DIME_HANDLE handle, DWORD ModuleNum, DWORD CmdMode, void *pValue)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNum is the module that is being addressed. Note modules are numbered from 0.

CmdMode: This argument is used to specify what particular aspect of module is to be controlled. There are no current command modes for this function.

Value: This argument is used to specify the action for a command mode.

Return Returns NULL on error.

Description This function is used to control certain aspects of the card that cannot be controlled using DIME_ModuleControl.

6.61 DIME_ModuleStatus

Syntax DWORD DIME_ModuleStatus(DIME_HANDLE handle, DWORD ModuleNum, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNum is the module that is being addressed. Note modules are numbered from 0.

CmdMode: This argument is used to specify what particular aspect of module status information is to be returned. Table 36 on page 66 gives details of the available command modes.

CmdMode	Description
dinfDIMECODE	This command mode returns the 32-bit hex-decimal DIME Code (User Code) for the module. Please check your *Module Reference Guide* for further details.
dinfNUMDEVICES	This command mode returns the number of devices on the module.
dinfPRIMFPGATEMP	This command mode returns the die temperature of the primary FPGA in degrees Celsius. Temperatures are accurate to +/- 1 degree.
dinfSECFPGATEMP	This command mode returns the die temperature of the secondary FPGA in degrees Celsius. Temperatures are accurate to +/- 1 degree.
dinfMODULETEMP	This command mode returns the temperature of the module in degrees Celsius. Temperatures are accurate to +/- 1 degree. Note that this temperature is measured next to the User FPGA and hence usually follows the FPGA temperature. It shows the temperature of the module as a whole but not one specific device.

Table 36: DIME_ModuleStatus CmdMode Argument Options

CmdMode	Description
dinfPRIMTEMPALERTMAX	This command mode is used to read the maximum temperature level for the temperature alert signal on the primary FPGA. Once set if the FPGA die temperature exceeds this temperature the temperature alert signal is triggered. Note that the power on default setting for this temperature is 255 degrees Celsius. NOTE: The TempAlertMax temperature only sets the trigger for the alert signal. For an event to happen at that temperature the user will have to use the alert signal in their design.
dinfSECTEMPALERTMAX	This command mode is used to read the maximum temperature level for the temperature alert signal on the secondary FPGA. Once set if the FPGA die temperature exceeds this temperature the temperature alert signal is triggered. Note that the power on default setting for this temperature is 255 degrees Celsius. NOTE: The TempAlertMax temperature only sets the trigger for the alert signal. For an event to happen at that temperature the user will have to use the alert signal in their design.
dinfPRIMTEMPALERTMIN	This command mode is used to set the minimum temperature level for the temperature alert signal on the primary FPGA. Once set if the FPGA die temperature falls below this temperature the temperature alert signal is triggered. Note that the power on default setting for this temperature is 0 degrees Celsius. Note: not all temperature sense devices used have this capability. Consult your *Module Reference Guide* or the *Temperature Sense Datasheet* to find out if the temperature sense device on your module supports this.
dinfSECTEMPALERTMIN	This command mode is used to set the minimum temperature level for the temperature alert signal on the secondary FPGA. Once set if the FPGA die temperature falls below this temperature the temperature alert signal is triggered. Note that the power on default setting for this temperature is 0 degrees Celsius. Note: not all temperature sense devices used have this capability. Consult your *Module Reference Guide* or the *Temperature Sense Datasheet* to find out if the temperature sense device on your module supports this.
dinfSERIALNUMBER	This command mode is used to return the serial number of the module.

Table 36: DIME_ModuleStatus CmdMode Argument Options

Return The return value is dependant upon the command mode. Returns -1 on error.

Description This function returns module status information. Note, for the temperature functions, not all motherboards support this feature. Check your *Motherboard Reference Guide* if you are unsure if your hardware supports this.

Example

```
//read the temperature alert levels and both the module and FPGA
//temperatures
{
DWORD ModuleNumber=0;
DWORD FPGATemp,ModuleTemp,MaxAlert,MinAlert;
//Read the FPGA temperature (degrees c)
FPGATemp=DIME_ModuleStatus(hCard1,ModuleNumber,dinfFPGATEMP);
//Read the module temperature (degrees c)
ModuleTemp=DIME_ModuleStatus(hCard1,ModuleNumber,dinfMODULETEMP);
//Read the maximum alert threshold temperature (degrees c)
MaxAlert=DIME_ModuleStatus(hCard1,ModuleNumber,dinfTEMPALERTMAX);
//Read the minimum alert threshold temperature (degrees c)
MinAlert=DIME_ModuleStatus(hCard1,ModuleNumber,dinfTEMPALERTMIN);
}
```

Figure 12: Reading the Temperature Alert Levels and the FPGA and module temperatures

6.62 DIME_ModuleStatusPtr

Syntax void *DIME_ModuleStatusPtr(DIME_HANDLE handle, DWORD ModuleNum, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNum is the module that is being addressed. Note modules are numbered from 0.

CmdMode: This argument is used to specify what particular aspect of module status information is to be returned. The Table 37 on page 68 gives details of the available command modes.

CmdMode	Description
dinfIMAGEFILENAME	This command mode returns a string (char *), which is the image filename for the module.
dinfICONFILENAME	This command mode returns a string (char *), which is the icon filename for the module.
dinfDESCRIPTION	This command mode returns a string (char *), which is a short description of the module.

Table 37: DIME_ModuleStatusPtr CmdMode Argument Options

Return The return value is dependant upon the command mode. Returns NULL on error.

Description This function returns card status information that cannot be returned using DIME_ModuleStatus.

Notes If a pointer to a string is returned this string is only valid until the next call is made into the library. It is therefore advised that either the string is used directly or that it is copied for later use.

6.63 DIME_OpenCard

Syntax DIME_HANDLE DIME_OpenCard(LOCATE_HANDLE LocateHandle, int CardNumber, DWORD Flags)

Arguments LocateHandle is a valid handle returned from the DIME_LocateCard function.

CardNumber is the index of the card within the locate handle that the developer wishes to open.

Flags: This argument allows the open process to be customised to suit the development requirements. The Table 38 on page 69 gives details for the Flags.

Flags	Description
dccOPEN_DEFAULT	This is the default option for opening the card. With this option the on-board oscillators will get set to their default frequencies if this is appropriate for the card. See your *Motherboard Reference Guide* for card specific details.
dccOPEN_NO_OSCILLATOR_SETUP	This option opens the card as in the default mode except that the on-board oscillators are not set to their default frequencies.

Table 38: DIME_OpenCard Flags Argument Options

Return Returns a handle that is used when calling other functions for this card. Returns NULL on error.

Description Calling this function opens the motherboard and performs all the required set up so that the motherboard can be interfaced with. Once this function has been called all other functions are available.

Example See Figure 4 on page 25.

6.64 DIME_Peripheralioctl

Syntax DWORD DIME_Peripheralioctl(DIME_HANDLE handle, DWORD CMD, DWORD *Arg1, DWORD *Arg2, DWORD *Arg3, DWORD *Arg4, void *Arg5)

Arguments handle is a valid handle to a DIME carrier card.

CMD: The command to be performed. Currently there are no commands.

Arg1, Arg2, Arg3, Arg4 and Arg5 are all dependant upon the command.

Return The return is dependant upon the selected command.

Description This function is used to control and return status information for the peripheral I/O.

6.65 DIME_PPSControl

Syntax DWORD DIME_PPSControl(DIME_HANDLE handle, DWORD ModuleNum, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNum: This is the module number.

CmdMode: This argument is used to specify what particular aspect of programmable power supplies information is required. There are no current command modes for this function.

Value: This argument is used to specify the action for a command mode.

Return The return is dependant upon the selected command mode.

Description Allows control of the programmable power supplies.

6.66 DIME_PPSStatus

Syntax DIME_PPSStatus(DIME_HANDLE handle, DWORD ModuleNum, DWORD SupplyNum, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNum: This is the module number.

SupplyNum: This argument is used to specify what particular power supply is targeted. Valid supply numbers are provided Table 39 on page 70.

SupplyNum	Description
dppsSUPPLYA	Power Supply A is selected.
dppsSUPPLYB	Power Supply B is selected.
dppsSUPPLYC	Power Supply C is selected.
dppsSUPPLYD	Power Supply D is selected.
dppsALLSUPPLYS	All supplies are selected.

Table 39: DIME_PPSStatus Supply Number Options

CmdMode: This argument is used to specify what particular aspect of programmable power supplies information is required.

CmdMode	Description
dppsVOLTAGE	This command mode selects that only voltage information is returned. The voltage returned is given in millivolts and has an error of +/- 100millivolts.

Table 40: DIME_PPSStatus Command Mode Options

Return The return is dependant upon the selected command mode.

Description Returns status information for the programmable power supplies.

Note The voltage and current capabilities are only applicable to DIME-II systems.

Example

```
//Get the Voltage and Currents for module 0 power supply C.
{
DWORD Current,Voltage;
Voltage=DIME_PPSStatus(hCard1,dppsMODULE0,
                        dppsSUPPLYC, dppsVOLTAGE);
printf("Core Voltage is %d millivolts.\n", Voltage);

//Get the Voltage and current for all of module 0.
Voltage=DIME_PPSStatus(hCard1,dppsMODULE0,
                        dppsALLSUPPLIES, dppsVOLTAGE);
printf("The total Voltage for Module 0 is %d.\n", Voltage);
}
```

Figure 13: Getting Information on Power Supply Voltages and Currents

6.67 DIME_ReadLEDs

Syntax DWORD DIME_ReadLEDs(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return This returns the current setting of the LEDs which are controlled via the PCI interface.

Description The values are stored in the least significant bits with a value of '0' indicating that the LED is illuminated. If a valid card has not been opened then a 0 is returned by the function. The number of active bits depends on the number of LEDs on the motherboard.

Please check your *Motherboard Reference Guide* for details on the LEDs.

Example See Figure 4 on page 25.

6.68 DIME_ReadPIO

Syntax DWORD DIME_ReadPIO(DIME_HANDLE handle, DWORD Bank)

Arguments handle is a valid handle to a DIME carrier card.

Bank is used to indicate which bank of periphery I/O is to be read. Table 41 on page 72 gives details of the available banks.

Bank	Description
dpioDIGITAL	This is the digital I/O. Data will be read from the digital I/O connector.

Table 41: Periphery I/O Bank Argument Options

Please check your *Motherboard Reference Guide* for details on available periphery I/O.

Return Returns the value of the requested pins. Returns all ones on error.

Description This function reads the values on the pins of the periphery I/O.

6.69 DIME_ReadPIODirection

Syntax DWORD DIME_ReadPIODirection(DIME_HANDLE handle, DWORD Bank)

Arguments handle is a valid handle to a DIME carrier card.

Bank is used to indicate which bank of periphery I/O direction is to be read. See Table 41 on page 72 for details of the available banks.

Please check your *Motherboard Reference Guide* for details on available periphery I/O.

Return A '1' in a particular bit location indicates that the pin is an input, otherwise the pins is an output. Returns -1 on error.

Description This returns the setting of the direction for the specified bank.

6.70 DIME_SaveCardDefinition

Syntax DWORD DIME_SaveCardDefinition (DIME_HANDLE handle, const char *Filename, DWORD Flags)

Arguments handle is a valid handle to a DIME carrier card.

Filename is the filename that the card definition is to be saved to.

Flags: This argument allows the developer to configure the information held within the card definition files to suit their development needs. Table 42 on page 73 gives details for the Flags.

Flags	Description
dcfgEMBEDALLBITS	It is possible not only to save the name of the assigned bitfiles into the card definition but also to save the bitstreams themselves. If this is desired then this flag should be used. This flag saves the bitstreams from the assigned bitfiles or the assigned bitstreams for each device on the card into the card definition file. This file can then be used to completely configure the card. It is the only file required. Note: If there are several bitstreams associated to large devices then the saved card definition file will become very large.

Table 42: DIME_SaveCardDefinition Flags Argument Options

Return Returns a pointer to the start of the bit-stream that has been assigned to the selected device. If no bit-stream has been set for the device then a NULL pointer is returned.

Description This function should be used to save information regarding the current configuration of your card to a file. Information such as the modules, devices and the assigned bitfiles/bitstreams to particular devices is all saved.

Notes By using the dcfgEMBEDALLBITS flag a complete system 'snapshot' is created. This can be very useful for back-up purposes or when porting the set-up to different PCs.

6.71 DIME_SaveSystemDefinition

Syntax DWORD DIME_SaveSystemDefinition (DIME_HANDLE **handles, char **CardFilenames, const char *SysFilename, DWORD NumOfCards, DWORD Flags)

Arguments handles is a pointer to an array containing the valid card handles for the system.

CardFilenames is a pointer to an array containing the card definition filenames that correspond with the card handle array.

SysFilename is a pointer to the system definition filename to be created.

NumOfCards is the number of cards that are to be used to create this system. This number should correspond with the number of elements in the handles array and the CardFilenames array.

Flags: This argument is used to customize the saving of the system definition file. Currently there are no flags and this argument is not used. 0 Should be used.

Return Returns 0 on success. Returns non-zero otherwise.

Description This function takes a group of card handles and card definition files and creates one system definition file. This file can then be used to load the complete system without the requirement of locating or opening the cards.

Notes The system definition file that is created does not incorporate the information held within the card definition files. Hence to successfully load a system definition file the card definition files for each card in the system must be in the same location as when the system definition file was created.

6.72 DIME_SetOscillatorFrequency

Syntax DWORD DIME_SetOscillatorFrequency(DIME_HANDLE handle, DWORD OscillatorNum, double DesiredFrequency, double *ActualFrequency)

Arguments handle is a valid handle to a DIME carrier card.

OscillatorNum determines which clock is changed where:

0 = All Clocks

1 = SYSCLK

2 = DSPCLK

3 = PIXCLK

DesiredFrequency is the desired frequency that the oscillator should be changed to. Note not all frequencies are achievable precisely and some error may result, this is where the ActualFrequency argument can be used to provide the actual frequency obtained.

ActualFrequency points to a memory location which is loaded with the actual frequency programmed. This last argument can be set to NULL if the returned value is not required.

Return A return value of 0 indicates success, 1 means that a NULL handle has been given, 2 means that the Oscillator Number is out of range, 3 indicates a invalid frequency requested.

Description This function is used to control frequency of the programmable oscillators. The frequency is given in Mhz and the frequency change is carried out glitch free.

Notes Please check your *Motherboard Reference Guide* for further details on the programmable oscillators.

Example

```
//Change the oscillators.
{
double ActualFrequency;
//Try and set oscillator 1, the system clock to 41.23456MHz
DIME_SetOscillatorFrequency(hCard1,1,41.23456,&ActualFrequency);
printf("Actual frequency is %f.\n",ActualFrequency);
}
```

Figure 14: Getting Information on the Located Cards

6.73 DIME_ShowMConfigGUI

Syntax void DIME_ShowMConfigGUI(void * handle, DWORD ShowFlag)

Arguments handle is a valid handle to a DIME carrier card.

ShowFlag is a flag for the visibility of the form.

Return N/A

Description This function changes whether the Multiple DIME configuration window is visible or not.

Notes The nueym.lib (for Microsoft Visual C++) or nueyomf.lib (for Borland C++ builder) needs to be included in your design when using this function. These libraries are installed in the include directory of the FUSE software.

6.74 DIME_SystemControl

Syntax DWORD DIME_SystemControl(DIME_HANDLE handle, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

CmdMode: This argument is used to specify what particular aspect of system information is to be controlled. Table 43 on page 75 provides details of the available command modes.

CmdMode	Description
dinfDIME_MODE_GUI	Controls whether dialog boxes are displayed within the SDL or whether no dialog boxes are displayed so that the calling applications can control any error message boxes. If the Value argument is dinfDISABLE then dialogue boxes will not be displayed. If the Value argument is dinfENABLE then dialogue boxes will be displayed. Note that the actual error will not be altered by this function and the can be used to return the error information. Returns 0 on success.

Table 43: DIME_SystemControl CmdMode Argument Options

Value: This argument is used to specify the action for a command mode. Table 44 on page 75 provides details.

Value	Description
dinfDISABLE	Disables the selected command mode feature.
dinfENABLE	Enables the selected command mode feature.

Table 44: DIME_SystemControl Value Argument Options

Return The return is dependant upon the command mode. A return of '-1' indicates an error.

Description This function is used to control system features.

6.75 DIME_SystemStatus

Syntax DWORD DIME_SystemStatus(DIME_HANDLE handle, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

CmdMode: This argument is used to specify what particular aspect of system status information is to be returned. Table 45 on page 76 provides details of the available command modes.

CmdMode	Description
dinfDIME_MODE_GUI	Returns whether dialog boxes are displayed within the SDL or whether no dialog boxes are displayed so that the calling applications can control any error message boxes. A return of '1' indicates that the dialogue boxes will appear and a return of '0' indicates that the dialogue boxes will not appear. Note that the actual error will not be altered by this function and can be used to return the error information.

Table 45: DIME_SystemStatus CmdMode Argument Options

Return The return is dependant upon the command mode. A return of '-1' indicates an error.

Description This function is used to return system status information.

6.76 DIME_SystemStatusPtr

Syntax void *DIME_SystemStatusPtr(DIME_HANDLE handle, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card.

CmdMode: This argument is used to specify what particular aspect of system status information is to be returned. Table 46 on page 76 provides details of the available command modes.

CmdMode	Description
dinfDIME_SWMBTS	This command mode does not require a card to be opened. The Handle argument is not used. This command mode is used to return a structure that contains the system information on what motherboard types it can detect and a brief description of each of these motherboard types.

Table 46: DIME_SystemStatusPtr CmdMode Argument Options

Return Return NULL on error.

For the command mode dinfDIME_SWMBTS the return will be a pointer of type SWMBInfo.

Type	Name	Description
DWORD	NumTypes	The number of motherboard types returned.
CardInfo	pCardInfo	Pointer to an array containing the motherboard information. The number of elements in this array corresponds with NumTypes.

Table 47: Type SWMBInfo Members

Type	Name	Description
DWORD	MotherBoardType	The motherboard type. See Table 29 on page 55 details.
char	MotherBoardDesc[200]	A short description of the motherboard. E.g. "Ballynuey2".

Table 48: Type CardInfo members

Description This function is used to return system status information that cannot be returned using DIME_SystemStatus.

Example

```
DWORD i;
SWMBInfo* pSWMBInfo;
//Using the DIME_SystemStatusPtr function

if( (pSWMBInfo=(SWMBInfo*)DIME_SystemStatusPtr(0,dinfDIME_SWMBTS))
      !=NULL)
{
      printf("The software detects %d motherboards.\n",
            pSWMBInfo->NumTypes);
      for (i=0; i<pSWMBInfo->NumTypes; i++)
      {
            printf("Details of mothreboard number %d of %d
                  follows:\n",(1+i),pSWMBInfo->NumTypes);
            printf("\tMotherboard type: %d.\n",pSWMBInfo
                  ->pCardInfo[i].MotherBoardType);
            printf("\tMotherboard description: %s.\n",pSWMBInfo
            ->pCardInfo[i].MotherBoardDesc);
      }
}
```

Figure 15: DIME_SystemStatusPtr Example

6.77 DIME_UnLockMemory

Syntax DWORD DIME_UnLockMemory(DIME_HANDLE handle, DIME_MEMHANDLE MemHandle)

Arguments handle is a valid handle to the DIME carrier card that performed the DIME_LockMemory.

MemHandle is the valid memory handle that needs to be unlocked.

Return Returns zero on success, non-zero otherwise.

Description This unlocks memory and gives control of the memory back to the OS kernel.

Example See Figure 10 on page 60.

6.78 DIME_WriteLEDs

Syntax void DIME_WriteLEDs(DIME_HANDLE handle, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

Value is the value that is to be written to the LEDs.

Return N/A

Description This function sets the status of the LEDs that are controlled via the PCI interface. The values are stored in the least significant bits with a value of '0' indicating that the LED is to be illuminated. The function checks that a valid card has been opened before setting the LEDs status.

Please check your *Motherboard Reference Guide* for details on the LEDs.

Example See Figure 4 on page 25.

6.79 DIME_WritePIO

Syntax DWORD DIME_WritePIO(DIME_HANDLE handle, DWORD Bank, DWORD Data)

Arguments handle is a valid handle to a DIME carrier card.

Bank is used to indicate which bank of periphery I/O is to be written to. See Table 41 on page 72 for details of the available banks.

Please check your *Motherboard Reference Guide* for details of available periphery I/O.

Data is the data to be written to the periphery I/O.

Return Returns 0 on success. Returns non-zero on error.

Description This function writes the values to the pins of the periphery I/O.

6.80 DIME_WritePIODirection

Syntax DWORD DIME_WritePIODirection(DIME_HANDLE handle, DWORD Bank, DWORD Data)

Arguments handle is a valid handle to a DIME carrier card.

Bank is used to indicate which bank of periphery I/O direction is to be written to. See Table 41 on page 72 for details of the available banks

Please check your *Motherboard Reference Guide* for details of available periphery I/O.

Return Returns zero on success, non zero on error.

Description This function sets the direction of individual pins for the specified bank. A '1' in a particular bit location sets that pin to an input, otherwise the pins is an output.

Section 7

Legacy Functions

In this section:

- The FUSE API is backward compatible with Nallatech's DIME system software. These legacy functions are from the DIME system software library and, although still supported under the FUSE API, you should convert to the new FUSE API functions. This section contains full details of all obsolete functions with alternative FUSE API function suggestions.

7.1 CloseDIMEBoard

Syntax void CloseDIMEBoard(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME Carrier Card that was returned from the OpenDIMEBoard function.

Return N/A

Description This function should be called at the end of the program. This closes access to the DIME Carrier card and frees the resources used by the card and the software library.

Notes Should only be used when a card has been opened using the OpenDIMEBoard function.

Alternative DIME_CloseCard providing OpenDIMEBoard was not used to open the card.

7.2 OpenDIMEBoard

Syntax DIME_HANDLE OpenDIMEBoard(void)

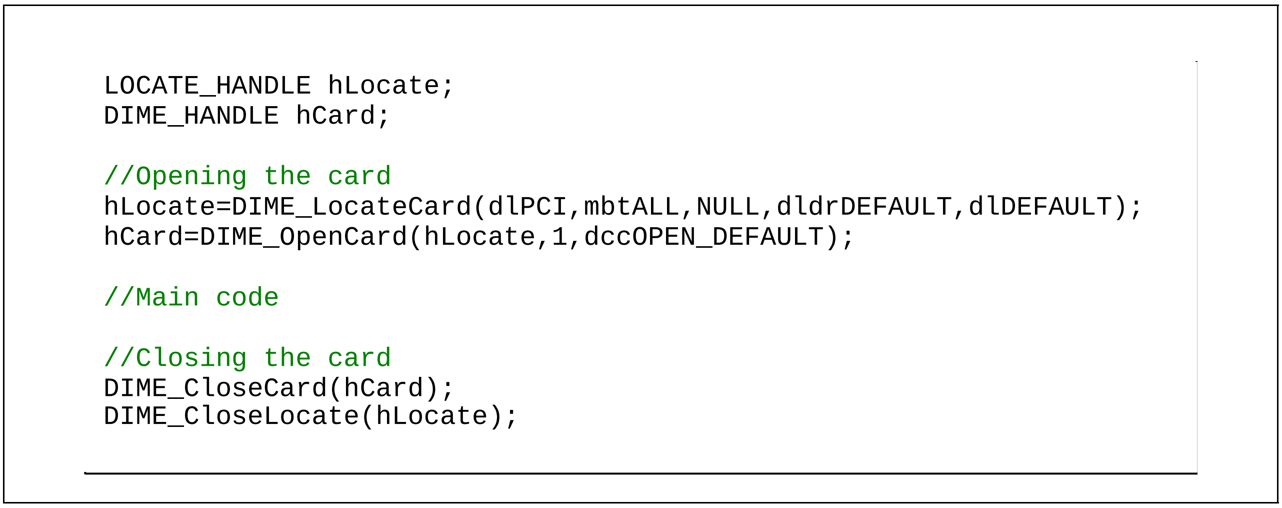
Arguments N/A

Return Returns a handle for the card, otherwise NULL is returned. The return type DIME_HANDLE is defined as a void pointer.

Description This function will search the PCI interface for any Nallatech motherboard and then call DIME_OpenCard for the first Nallatech motherboard found.

Notes Included only for backward compatibility. When this function is used to provide a handle for a card the CloseDIMEBoard function must be used to close down the card. This function cannot be used in conjunction with the DIME_LocateCard or DIME_OpenCard functions.

Alternative It is strongly advised that DIME_LocateCard then DIME_OpenCard is used as shown in Figure 16 on page 80 as an alternative.



```
LOCATE_HANDLE hLocate;
DIME_HANDLE hCard;

//Opening the card
hLocate=DIME_LocateCard(dlPCI,mbtALL,NULL,dldrDEFAULT,dlDEFAULT);
hCard=DIME_OpenCard(hLocate,1,dccOPEN_DEFAULT);

//Main code

//Closing the card
DIME_CloseCard(hCard);
DIME_CloseLocate(hLocate);
```

Figure 16: Alternative to OpenDIMECard

7.3 GetDIMEHandle

Syntax DIME_HANDLE GetDIMEHandle(void)

Arguments N/A

Return Always returns NULL.

Description This function previously returned the handle that was last returned by OpenDIMEBoard. This cannot be achieved now since it is possible that multiple cards and hence handles have been generated. It is now up to the developer to store all valid handles returned. This function has now been made fully obsolete and will now simply return NULL.

7.4 DIME_SmartScan

Syntax DWORD DIME_SmartScan(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return ssOK SmartScan has been successfully completed.

Description This function previously carried out a scan of all the hardware modules and devices that were present in the cards JTAG chain. However this scan is now incorporated into the DIME_OpenCard function. So if a card is open then a successful SmartScan has already been carried out. For this reason this function does nothing except return ssOK.

7.5 DIME_VirtexReset

Syntax void DIME_VirtexReset(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return N/A.

Description Simply calls DIME_CardResetControl (handle, drONBOARDFPGA, drTOGGLE, 0).

Functionality not changed. Enables the reset for the on-board FPGA.

Alternative DIME_CardResetControl (handle, drONBOARDFPGA, drTOGGLE, 0).

7.6 DIME_VirtexResetEnable

Syntax void DIME_VirtexResetEnable(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return N/A.

Description Simply calls DIME_CardResetControl (handle, drONBOARDFPGA, drENABLE, 0).

Functionality not changed. Enables the reset for the on-board FPGA.

Alternative DIME_CardResetControl (handle, drONBOARDFPGA, drENABLE, 0).

7.7 DIME_VirtexResetDisable

Syntax void DIME_VirtexResetDisable (DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return N/A.

Description Simply calls DIME_CardResetControl (handle, drONBOARDFPGA, drDISABLE, 0).

Functionality not changed. Disables the reset for the on-board FPGA.

Alternative DIME_CardResetControl (handle, drONBOARDFPGA, drDISABLE, 0).

7.8 DIME_SystemReset

Syntax void DIME_SystemReset(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return N/A.

Description Simply calls DIME_CardResetControl (handle, drSYSTEM, drTOGGLE, 0).

Functionality not changed. Toggles the reset for the system.

Alternative DIME_CardResetControl (handle, drSYSTEM, drTOGGLE, 0).

7.9 DIME_SystemResetEnable

Syntax void DIME_SystemResetEnable(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return N/A.

Description Simply calls DIME_CardResetControl (handle, drSYSTEM, drENABLE, 0).

Functionality not changed. Enables the reset for the system.

Alternative DIME_CardResetControl (handle, drSYSTEM, drENABLE, 0).

7.10 DIME_SystemResetDisable

Syntax void DIME_SystemResetDisable(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return N/A.

Description Simply calls DIME_CardResetControl (handle, drSYSTEM, drDISABLE, 0).

Functionality not changed. Disables the reset for the system.

Alternative DIME_CardResetControl (handle, drSYSTEM, drDISABLE, 0).

7.11 DIME_PCIReset

Syntax void DIME_PCIReset(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return N/A.

Description Simply calls DIME_CardResetControl(handle, drINTERFACE, drTOGGLE, 0)

Functionality not changed. Disables the reset for the system.

Alternative DIME_CardResetControl(handle, drINTERFACE, drTOGGLE, 0).

7.12 DIME_ReadDigitalIO

Syntax DWORD DIME_ReadDigitalIO(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return Returns the value of the digital I/O pins.

Description Simply calls DIME_ReadPIO(handle, dpioDIGITAL).

Functionality not changed. Reads the pins on the digital I/O connector.

Alternative DIME_ReadPIO(handle, dpioDIGITAL).

7.13 DIME_WriteDigitalIO

Syntax DWORD DIME_WriteDigitalIO(DIME_HANDLE handle, DWORD Data)

Arguments handle is a valid handle to a DIME carrier card.

Data is the values to be written to the pins on the digital I/O connector.

Return Returns 0 on success. Returns non-zero on error.

Description Simply calls DIME_WritePIO(handle, dpioDIGITAL, Data).

Functionality not changed. Writes the pins on the digital I/O connector.

Alternative DIME_WritePIO(handle, dpioDIGITAL, Data).

7.14 DIME_ReadDigitalIODirection

Syntax DWORD DIME_ReadDigitalIODirection(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return A '1' in a particular bit location indicates that the pin is an input, otherwise the pins is an output. Returns -1 on error.

Description Simply calls DIME_ReadPIODirection(handle, dpioDIGITAL)

Functionality not changed. Reads the pins on the digital I/O connector.

Alternative DIME_ReadPIODirection(handle, dpioDIGITAL).

7.15 DIME_WriteDigitalIODirection

Syntax DWORD DIME_WriteDigitalIODirection(DIME_HANDLE handle,DWORD Data)

Arguments handle is a valid handle to a DIME carrier card.

Data is the data used to set the direction of individual pins.

Return Returns zero on success, non zero on error.

Description Simply calls DIME_WritePIODirection(handle, dpioDIGITAL, Data). This function sets the direction of individual pins of the Digital I/O connector. A '1' in a particular bit location sets that pin to an input, otherwise the pin is an output.

Functionality not changed.

Alternative DIME_WritePIODirection(handle, dpioDIGITAL, Data).

7.16 DIME_VirtexIntPin

Syntax DWORD DIME_VirtexIntPin(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return If the interrupt pin on the FGPA is high then this returns 1, otherwise it returns 0. Returns -1 on error.

Description This function returns the value on the Interrupt pin from the FPGA. Simply calls DIME_InterruptStatus(handle, dintONBOARDFPGA, dintPINVALUE).

Alternative DIME_InterruptStatus(handle, dintONBOARDFPGA, dintPINVALUE).

7.17 DIME_InterfaceFlagBusy

Syntax DWORD DIME_InterfaceFlagBusy(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return Returns the status of the BUSY signal that is on the PCI to FPGA Interface. When the BUSY signal is high this function returns a 1 otherwise it returns a 0.

Description When BUSY is high it indicates that the internal transfer buffer from the FPGA to the PCI is full and cannot accept any more data. The user application should initiate a data read at this stage.

Simply calls DIME_DMAStatus(handle, ddmaALLDMACHANNELS, ddmaBUSYFLAG).

Alternative DIME_DMAStatus(handle, ddmaALLDMACHANNELS, ddmaBUSYFLAG).

7.18 DIME_InterfaceFlagEmpty

Syntax DWORD DIME_InterfaceFlagEmpty(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return Returns the status of the EMPTY signal that is on the PCI to FPGA Interface. When the EMPTY signal is high this function returns a 1 otherwise it returns a 0.

Description When EMPTY is high it indicate that there is no data waiting to be transferred to the FPGA, i.e. the FPGA application has read all the available data that has been transferred via a PCI write operation.

Calls DIME_DMAStatus(handle, ddmaALLDMACHANNELS, ddmaEMPTYFLAG).

Alternative DIME_DMAStatus(handle, ddmaALLDMACHANNELS, ddmaEMPTYFLAG).

7.19 DIME_InterfaceFlagVirtexReadEmpty

Syntax DWORD DIME_InterfaceFlagVirtexReadEmpty(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return If there is no data in read buffer to be accessed this function will return 0, however 1 will be returned if there is data waiting to be read.

Description This function returns the status of the internal buffer between the FPGA and the PCI interface. Therefore when the function returns a 0 the data is available to be read DIME_DataRead or DIME_DataReadSingle.

Calls DIME_DMAStatus(handle, ddmaALLDMACHANNELS, ddmaREADEMPTY).

Alternative DIME_DMAStatus(handle, ddmaALLDMACHANNELS, ddmaEMPTYFLAG).

7.20 DIME_InterfaceFlagVirtexWriteFull

Syntax DWORD DIME_InterfaceFlagVirtexWriteFull(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return This function returns a 1 when the internal buffer from the PCI to the FPGA is full and hence cannot accept any more data from the user application.

Description The user application must therefore wait until the FPGA reads data before any more data will be transferred.

Calls DIME_DMAStatus(handle, ddmaALLDMACHANNELS, ddmaWRITEFULL).

Alternative DIME_DMAStatus(handle, ddmaALLDMACHANNELS, ddmaWRITEFULL).

7.21 DIME_JTAGTurboDisable

Syntax void DIME_JTAGTurboDisable(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return N/A.

Description This sets the cards JTAG chain to run at its default speed.

Alternative DIME_JTAGControl(handle, djtagCONFIGSPEED, djtagDEFAULTSPEED).

7.22 DIME_JTAGTurboEnable

Syntax void DIME_JTAGTurboEnable(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return N/A.

Description This sets the cards JTAG chain to run faster than its default speed.

Alternative DIME_JTAGControl(handle, djtagCONFIGSPEED, djtagMAXSPEED20).

7.23 DIME_BootVirtexSingle

Syntax DWORD DIME_BootVirtexSingle(DIME_HANDLE handle, const char *FileName)

Arguments handle is a valid handle to a DIME carrier card.

FileName is the filename of the bitfile that is to be used for booting the on-board FPGA.

Return If successful this function returns a 0, otherwise a non-zero result is returned on error and the device was not configured properly.

The return values are as listed for the function DIME_BootDevice.

Description This function boots the on-board FPGA device using the bitfile given by 'filename'. Note that the bitfile must be configured to use the JTAG clock for configuration rather than the default of the CCLK.

Alternative DIME_ConfigOnBoardDevice(handle, FileName, 0).

7.24 DIME_BootDevice

Syntax DWORD DIME_BootDevice(DIME_HANDLE handle, const char *FileName, DWORD ModuleNumber, DWORD ModuleDeviceNumber, DWORD *Progress)

Arguments handle is a valid handle to a DIME carrier card.

FileName is the filename of a bit for loading into the FPGA.

ModuleNumber is the Module that is being addressed.

ModuleDeviceNumber is the selected device within the select module.

Progress should point to a variable which will be updated with the actual position in the configuration. The position in the configuration file is expressed as a percentage (0 - 100). This is only useful in multi-threaded applications and may point to a valid location or NULL in single threaded applications.

Return

cfgINVLAID_CARD	Indicates a valid card has not been detected.
cfgOK_NOSTATUS	Indicates that a bitfile has been successfully shifted into the chain, with no post configuration checking carried out.
cfgBIT_FILE	If a the specified bitfile could not be successfully opened.
cfgINTEG_FAIL	Indicates that the JTAG integrity scan check has failed and the chain is apparently incomplete.
cfgDL_IL_NOCRC	Configuration Status - DONE Low, INIT Low, No CRC errors detected.
cfgDL_IL_CRC	Configuration Status - DONE Low, INIT Low, CRC errors detected.
cfgDL_IH_NOCRC	Configuration Status - DONE Low, INIT high, No CRC errors detected.

cfg_DL_IH_CRC	Configuration Status - DONE Low, INIT high, CRC errors detected.
cfgDH_IL_NOCRC	Configuration Status - DONE high, INIT low, No CRC errors detected.
cfgDH_IL_CRC	Configuration Status - DONE high, INIT low, CRC errors detected.
cfgOK_STATUS	Configuration completed successfully as indicated by read back of FPGA Status register. DONE high, INIT high, No CRC errors detected.
cfgDH_IH_CRC	Configuration Status - DONE high INIT high, CRC errors detected.
cfgNOLIC	Multiple Configuration Licence not available.
cfg_UNKNOWN	Unidentifiable configuration result.

Description This is the main function used for carrying out the actual configuration sequence of a specific single Programmable Logic device. The main arguments apart from handle, to denote if a card has actually been detected, are ModuleNumber and ModuleDeviceNumber. These are used to identify a particular device in the JTAG chain and provides enough information to configure the selected device.

Alternative DIME_ConfigDevice(handle, FileName, ModuleNumber, ModuleDeviceNumber, Progress, 0).

7.25 DIME_SetFilename

Syntax DWORD DIME_SetFilename(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber, const char *Filename)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the Module that is being addressed.

DeviceNumber is the selected device within the select module.

Filename is the filename of a bit for loading into the FPGA.

Return Returns 0 upon success. Returns non-zero otherwise.

Description This function can be used to set the filename for the specified device on the specified module. The handle for the particular board also needs to be passed to the function The specified filename is stored in internal data structures for later use.

Alternative DIME_ConfigSetBitsFilename(handle, ModuleNumber, DeviceNumber, Filename,0).

7.26 DIME_GetFilename

Syntax const char *DIME_GetFilename(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

DeviceNumber is the selected Device number.

Return Returns a pointer to the filename that has been assigned to the selected device. If no filename has been set for the device then a NULL pointer is returned.

Description This function is used to read the filename that has been set for a particular device on a module. The handle for the particular board also needs to be passed to the function.

Alternative DIME_ConfigGetBitsFilename(handle, ModuleNumber, DeviceNumber).

7.27 DIME_SetFilenameAndConfig

Syntax DWORD DIME_SetFilenameAndConfig (DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber, const char *Filename, DWORD *Progress)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the Number of the selected module.

DeviceNumber is the selected device.

Filename is the filename used to assign for configuration.

Progress is the progress through a configuration.

Return Returns the result of DIME_ConfigSetBitsFilename if there is an error. If no error occurs in this function then it returns the result of the device configuration.

Description This function calls the function, DIME_ConfigSetBitsFilenameAndConfig to assign the filename to the device and then to configure the device.

Alternative DIME_ConfigSetBitsFilenameAndConfig(handle, ModuleNumber, DeviceNumber, Filename, 0, Progress, 0).

7.28 DIME_SaveSystemConfig

Syntax DWORD DIME_SaveSystemConfig (DIME_HANDLE handle, const char *Filename)

Arguments handle is a valid handle to a DIME carrier card.

Filename is the file to save the system configuration to.

Return If the information is successfully written to a file then the function returns a 0 otherwise a non-zero result indicates an unsuccessful configuration.

Description This function can be used to save existing configuration information such as module, device information and which bitfiles have been assigned to particular devices. The information can simply be written to the file specified.

Alternative DIME_SaveCardDefinition(handle,Filename,0).

7.29 DIME_LoadSystemConfig

Syntax DWORD DIME_LoadSystemConfig (DIME_HANDLE handle, const char *Filename)

Arguments handle is a valid handle to a DIME carrier card.

Filename is the file to load the system configuration from.

Return A successful load is indicated by a return value of 0.

Description This function can be used to load back into the program existing information previously saved to a file.

Alternative DIME_LoadCardDefinition(handle,Filename,0).

7.30 DIME_GlobalMode

Syntax DWORD DIME_GlobalMode(DIME_HANDLE handle, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card.

CmdMode is the mode to change.

Value is the value to set the mode to.

Return Returns non-zero value on error.

Description This is a function to add greater global control over the functionality of the development libraries. The details are given in the **Notes** below.

The general operation of this functions is to pass the global mode operation to be changed in the 'CmdMode argument and to pass its state in the 'Value' argument.

Notes

CmdMode	Function
DIME_MODE_GUI	Sets whether dialog boxes are displayed within the SDL or whether no dialog boxes are displayed so that the calling applications can control any error message boxes. This does not affect the return values of functions. When 'Value' is TRUE, dialog boxes are displayed, when FALSE they are not displayed. Default is on, i.e. Value = TRUE.

Table 49: DIME_GlobalMode

CmdMode	Function
DIME_JTAG_CHECK	When a Configuration of the FPGA is performed it is possible for a post configuration check to be done on the configuration of the FPGA itself. Essentially, this reads the contents out of the 32-bit internal FPGA status register. When 'Value' is TRUE post configuration status checking is carried out, otherwise when 'Value' is FALSE no checking is done and the FPGA is still sent the selected bit-stream. Default is on in multiple configuration, i.e. Value = TRUE.

Table 49: DIME_GlobalMode

Alternative DIME_SystemControl(handle, CmdMode, Value).

7.31 DIME_GetMotherBoardType

Syntax DWORD DIME_GetMotherBoardType(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME Card.

Return See Table 29 on page 55 for details.

Description The DIME API is made as generic as possible for all DIME Carrier cards and this function returns the type of Motherboard installed. This enables an application to take advantage of any special facilities for a particular card.

Alternative DIME_CardStatus(handle, dinfMOTHERBORDTYPE).

7.32 DIME_GetMultiConfigLicence

Syntax DWORD DIME_GetMultiConfigLicence(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return A return value of 0 indicates that no multiple configuration licence is available and a value of 1 indicates that a multiple configuration licence is available.

Description This function returns whether the multiple configuration licence is valid on this system.

Alternative DIME_CardStatus(handle, dinfMULTICONFIGLICENCE).

7.33 DIME_ReadSlotUsed

Syntax DWORD DIME_ReadSlotUsed(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return A '1' in a particular bit location indicates that a Module is present, otherwise the slot is free. Bit 0 represents slot 0, bit 1 represents slot 1 etc.

Description This returns flags to indicate if a module is plugged into a particular DIME slot.

Alternative DIME_CardStatus(handle, dinfSLOTSUSED).

7.34 DIME_GetNumberOfModules

Syntax DWORD DIME_GetNumberOfModules(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return Returns the number of modules installed in the card

Description The function first checks that a valid card has in fact been successfully opened, if not a 0 is returned. The function accesses the internal data structure of board information and returns the value for the number of modules detected in the current board set-up.

Notes The on-board FGPA device is counted as a module itself.

Alternative DIME_CardStatus(handle, dinfNUMBERMODULES).

7.35 DIME_GetFailedMDFFileName

Syntax const char *DIME_GetFailedMDFFileName(DIME_HANDLE handle)

Arguments handle is a valid handle to a DIME carrier card.

Return The filename can be returned through the use of this function. If a card has not successfully been opened then NULL is returned, otherwise the MDF filename is returned.

Description When opening the card a number of associated Module Definition Files are read into the program. If there is a problem with a required MDF file then the card will not be opened.

Alternative DIME_CardStatusPtr(handle, dinfFAILEDMDF).

7.36 DIME_GetModuleDIMECode

Syntax DWORD DIME_GetModuleDIMECode(DIME_HANDLE handle, DWORD ModuleNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

Return The function accesses the internal data structure of board information and returns the 32-bit hexadecimal DIME Code (User Code) for the specified module. On error 0 is returned.

Please refer to the *Module Reference Guide* for the code details.

Description The function also checks that the specified module number is not greater than the total number of modules detected on the board.

Alternative DIME_ModuleStatus(handle, ModuleNumber, dinfDIMECODE).

7.37 DIME_GetNumberOfDevices

Syntax DWORD DIME_GetNumberOfDevices(DIME_HANDLE handle, DWORD ModuleNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

Return The function accesses the internal data structure of board information and returns the number of devices for the specified module.

Description The function first checks that a valid card has in fact been successfully opened, if not 0 is returned. The function also checks that the specified module number is not greater than the total number of modules detected on the board.

Alternative DIME_ModuleStatus(handle, ModuleNumber, dinfNUMDEVICES).

7.38 DIME_GetModuleDescription

Syntax const char *DIME_GetModuleDescription (DIME_HANDLE handle, DWORD ModuleNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

Return Returns a pointer to a string that describes the module selected. Returns NULL on error.

Description The function first checks that a valid card has in fact been successfully opened, if not a NULL is returned. The function also checks that the specified module number is not greater than the total number of modules detected on the board. The function accesses the internal data structure of board information and returns the description for the specified module.

Alternative DIME_ModuleStatusPtr(handle, ModuleNumber, dinfDESCRIPTION).

7.39 DIME_GetModuleIconFilename

Syntax const char *DIME_GetModuleIconFilename(DIME_HANDLE handle, DWORD ModuleNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

Return The function accesses the internal data structure of board information and returns the complete path and filename for the icon representing the specified module.

Description The function first checks that a valid card has in fact been successfully opened, if not a NULL is returned. The function also checks that the specified module number is not greater than the total number of modules detected on the board.

Note If an icon has not been specified in the MDF associated with the module a default icon filename is loaded.

Alternative DIME_ModuleStatusPtr(handle, ModuleNumber, dinfICONFILENAME).

7.40 DIME_GetModuleImageFilename

Syntax const char *DIME_GetModuleImageFilename(DIME_HANDLE handle, DWORD ModuleNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

Return The function accesses the internal data structure of board information and returns the complete path and filename of the Image representing the specified module.

Description The function first checks that a valid card has in fact been successfully opened, if not a NULL is returned. The function also checks that the specified module number is not greater than the total number of modules detected on the board.

Alternative DIME_ModuleStatusPtr(handle, ModuleNumber, dinfIMAGEFILENAME).

7.41 DIME_GetDeviceIDCode

Syntax DWORD DIME_GetDeviceIDCode(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

DeviceNumber is the selected Device number.

Return The function accesses the internal data structure of board information and returns the 32-bit ID code for the specified module device. See Table 26 on page 53 for all possible returns values.

Description The function first checks that a valid card has in fact been successfully opened, if not a 0 is returned. The function also checks that the specified module and device numbers are not greater than the total number of modules detected on the board and the total number of devices for the module respectively.

Alternative DIME_DeviceStatus(handle, ModuleNumber, DeviceNumber, dinfDEVICEIDCODE).

7.42 DIME_GetDeviceType

Syntax DWORD DIME_GetDeviceType(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

DeviceNumber is the selected Device number.

Return This function returns the type of the specified device on the specified module. See Table 13 on page 36 for details.

Description As part of the MDF file format each device is classified as a particular type corresponding to whether it can be configured or not.

Alternative DIME_DeviceStatus(handle, ModuleNumber, DeviceNumber, dinfDEVICETYPE).

7.43 DIME_GetDeviceXOffset

Syntax DWORD DIME_GetDeviceXOffset(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

DeviceNumber is the selected Device number.

Return The function accesses the internal data structure of board information and returns the X-Offset for the specified module device in the module image.

Description The function first checks that a valid card has in fact been successfully opened, if not a 0 is returned. The function also checks that the specified module and device numbers are not greater than the total number of modules detected on the board and the total number of devices for the module respectively.

Alternative DIME_DeviceStatus(handle, ModuleNumber, DeviceNumber, dinfXOFFSET).

7.44 DIME_GetDeviceYOffset

Syntax DWORD DIME_GetDeviceYOffset(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

DeviceNumber is the selected Device number.

Return The function accesses the internal data structure of board information and returns the Y-Offset for the specified module device in the module image.

Description The function first checks that a valid card has in fact been successfully opened, if not a 0 is returned. The function also checks that the specified module and device numbers are not greater than the total number of modules detected on the board and the total number of devices for the module respectively.

Alternative DIME_DeviceStatus(handle, ModuleNumber, DeviceNumber, dinfYOFFSET).

7.45 DIME_GetDeviceWidth

Syntax DWORD DIME_GetDeviceWidth(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

DeviceNumber is the selected Device number.

Return The function accesses the internal data structure of board information and returns the width for the specified module device in the module image.

Description The function first checks that a valid card has in fact been successfully opened, if not a 0 is returned. The function also checks that the specified module and device numbers are not greater than the total number of modules detected on the board and the total number of devices for the module respectively.

Alternative DIME_DeviceStatus(handle, ModuleNumber, DeviceNumber, dinfWIDTH).

7.46 DIME_GetDeviceHeight

Syntax DWORD DIME_GetDeviceHeight(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

DeviceNumber is the selected Device number.

Return The function accesses the internal data structure of board information and returns the height for the specified module device in the module image.

Description The function first checks that a valid card has in fact been successfully opened, if not a 0 is returned. The function also checks that the specified module and device numbers are not greater than the total number of modules detected on the board and the total number of devices for the module respectively.

Alternative DIME_DeviceStatus(handle, ModuleNumber, DeviceNumber, dinfHEIGHT).

7.47 DIME_GetDeviceDescription

Syntax const char *DIME_GetDeviceDescription(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

DeviceNumber is the selected Device number.

Return The function accesses the internal data structure of board information and returns the description for the specified module device. A pointer to a string that describes the device is returned.

Description The function first checks that a valid card has in fact been successfully opened, if not a NULL is returned. The function also checks that the specified module and device numbers are not greater than the total number of modules detected on the board and the total number of devices for the module respectively.

Alternative DIME_DeviceStatusPtr(handle, ModuleNumber, DeviceNumber, dinfDESCRIPTION).

7.48 DIME_GetDeviceIconFilename

Syntax const char *DIME_GetDeviceIconFilename(DIME_HANDLE handle, DWORD ModuleNumber, DWORD DeviceNumber)

Arguments handle is a valid handle to a DIME carrier card.

ModuleNumber is the selected Module number.

DeviceNumber is the selected Device number.

Return The function accesses the internal data structure of board information and returns the complete path and filename representative of the specified module device. If an icon filename has not been specified in the MDF then a default device icon filename is returned.

Description The function first checks that a valid card has in fact been successfully opened, if not a NULL is returned. The function also checks that the specified module and device numbers are not greater than the total number of modules detected on the board and the total number of devices for the module respectively.

Alternative DIME_DeviceStatusPtr(handle, ModuleNumber, DeviceNumber, dinfICONFILENAME).

Section 8

Version History List

In this section:

- Version History of FUSE C/C++ API.

8.1 New in version 1.9

"Enhancement: DIME_GetCurrentHandle added.

"Enhancement: Added the following command modes to DIME_ModuleStatus: dinfFPGATEMP, dinfMODULETEMP, dinfTEMPALERTMAX, dinfTEMPALERTMIN

"Enhancement: Added the following command modes to DIME_ModuleControl: dinfTEMPALERTMAX, dinfTEMPALERTMIN, dinfTEMPALERTCLEAR

"Enhancement: Added the following command modes to DIME_PPSStatus: dppsVOLTAGE, dppsCURRENT

8.2 New in version 1.6

"Fix: Several minor bug fixes to the API

"Fix: When using DIME_DataRead and DIME_DataWrite for transfers greater than 32768 words the transfer will return a timeout error. Transfers greater than 32K words are now allowed.

"Fix: DIME_DMARead and DIME_DMAWrite where previously only locking down ¼ of the required memory for the transfer. This has been fixed so the correct amount of memory is locked down.

"Enhancement: DIME_PPSStatus and DIME_PPSControl now have a supply number argument added.

Standard Terms and Conditions

GENERAL

These Terms and Conditions shall apply to all contracts for goods sold or work done by Nallatech Limited. (hereinafter referred to as the "company" or Nallatech) and purchased by any customer (hereinafter referred to as the customer).

Nallatech Limited trading in the style Nallatech (the company), submits all quotations and price lists and accepts all orders subject to the following conditions of contract which apply to all contracts for goods supplied or work done by them or their employees to the exclusion of all other representations, conditions or warranties, express or implied.

The buyer agrees to execute and return any license agreements as may be required by the company in order to authorize the use of those licensable items. If the licensable item is to be resold this condition shall be enforced by the re-seller on the end customer.

Each order received by the company will be deemed to form a separate contract to which these conditions apply and any waiver or any act of non-enforcement or variation of these terms or part thereof shall not bind or prejudice the company in relation to any other contract.

The company reserves the right to re-issue its price list at any time and to refuse to accept orders at a price other than at the price stated on the price list in force at the time of order.

The company reserves the right to vary the specification or withdraw from the offer any of its products without prior warning.

The company reserves the right to refuse to accept any contract that is deemed to be contrary to the companies policies in force at the time.

PRICING

All prices shown on the company's price list, or on quotations offered by them, are based upon the acceptance of these conditions. Any variation of these conditions requested by the buyer could result in changes in the offered pricing or refusal to supply.

All quoted pricing is in Pounds Sterling and is exclusive of Value Added Tax (VAT) and delivery. In addition to the invoiced value the buyer is liable for all import duty as may be applicable in the buyer's location. If there is any documentation required for import formalities, whether or not for the purposes of duty assessment, the buyer shall make this clear at the time of order.

Quotations are made by Nallatech upon the customer's request but there is no obligation for either party until Nallatech accepts the customer's order.

Nallatech reserves the right to increase the price of goods agreed to be sold in proportion to any increase of costs to Nallatech between the date of acceptance of the order and the date of delivery or where the increase is due to any act or default of the customer, including the cancellation or rescheduling by the customer of part of any order.

Nallatech reserves the right (without prejudice to any other remedy) to cancel any uncompleted order or to suspend delivery in the event of any of the customer's commitment with Nallatech not being met.

DELIVERY

All delivery times offered by the company are to be treated as best estimates and no penalty can be accepted for non compliance with them.

Delivery shall be made by the company using a courier service of its choice. The cost of the delivery plus a nominal fee for administration will be added to the invoice issued. Payment of all inward customs duties and fees are the sole responsibility of the buyer. If multiple shipments are requested by the buyer, multiple delivery charges will be made. In the case of multiple deliveries separate invoices will be raised.

If requested at the time of ordering an alternative delivery service can be used, but only if account details are supplied to the company so that the delivery can be invoiced directly to the buyer by the delivery service.

The buyer accepts that any 'to be advised' scheduled orders not completed within twelve months from the date of acceptance of the original order, or orders held up by the buyers lack of action regarding delivery, can be shipped and invoiced by the company and paid in full by the buyer, immediately after completion of that twelve month period.

INSURANCE

All shipments from the company are insured by them. If any goods received by the buyer are in an unsatisfactory condition, the following courses of action shall be taken.

If the outer packaging is visibly damaged, then the goods should not be accepted from the courier, or they should be signed for only after noting that the packaging has sustained damage.

If the goods are found to be damaged after unpacking, the company must be informed immediately.

Under no circumstances should the damaged goods be returned, unless expressly authorized by the company.

If the damage is not reported within 48 hours of receipt, the insurers of the company shall bear no liability.

Any returns made to the company for any reason, at any time shall be packaged in the original packaging, or its direct equivalent and must be adequately insured by the buyer.

Any equipment sent to the company for any purpose, including but not limited to equipment originally supplied by the company must be adequately insured by the buyer while on the premises of the company.

PAYMENT

Nallatech Ltd. terms of payment are 30 days net.

Any charges incurred in making the payment, either

currency conversion or otherwise shall be paid by the buyer.

The company reserves the right to charge interest at a rate of 2% above the base rate of the Bank of Scotland PLC on any overdue accounts. The interest will be charged on any outstanding amount from said due date of payment, until payment is made in full, such interest will accrue on a daily basis.

TECHNICAL SUPPORT

The company offers a dedicated technical support via telephone and an E-mail address. It will also accept faxed support queries.

Technical support will be given free of charge for 90 days from the date of invoice, for queries regarding the use of the products in the system configuration for which they were sold. Features not documented in the user manual or a written offer of the company will not be supported. Interfacing with other products other than those that are pre-approved by the company as compatible will not be supported. If the development tools and system hardware is demonstrably working, no support can be given with application level problems.

WARRANTY

The company offers as part of a purchase contract 12 months warranty against parts and defective workmanship of hardware elements of a system. The basis of this warranty is that the fault be discussed with the companies technical support staff before any return is made. If it is agreed that a return for repair is necessary then the faulty item and any other component of the system as requested by those staff shall be returned carriage paid to the company. Insurance terms as discussed in the INSURANCE Section will apply.

Returned goods will not be accepted by the company unless this has been expressly authorized.

After warranty repair, goods will be returned to the buyer carriage paid by the company using their preferred method.

Faults incurred by abuse of the product (as defined by the company) are not covered by the warranty.

Attempted repair or alteration of the goods as supplied by the company, by another party immediately invalidates the warranty offered.

The said warranty is contingent upon the proper use of the goods by the customer and does not cover any part of the goods which has been modified without Nallatech's prior written consent or which has been subjected to unusual physical or electrical stress or on which the original identification marks have been removed or altered. Nor will such warranty apply if repair or parts required as a result of causes other than ordinary authorized use including without limitation accident, air conditioning, humidity control or other environmental conditions.

Under no circumstances will the company be liable for any incidental or consequential damage or expense of any kind, including, but not limited to, personal injuries and loss of profits arising in connection with any contract or with the use, abuse, unsafe use or inability to use the companies goods. The company's maximum liability shall not exceed and the customers remedy is limited to, either:

i. repair or replacement of the defective part or product or at the companies option.

ii. return of the product and refund of the purchase price and such remedy shall be the customer's entire and exclusive remedy.

Warranty of the software written by the company shall be limited to 90 days warranty that the media is free from defects and no warranty express or implied is given that the computer software will be free from error or will meet the specification requirements of the buyer.

The terms of any warranty offered by a third party whose software is supplied by the company will be honoured by the company exactly. No other warranty is offered by the company on these products.

Return of faulty equipment after the warranty period has expired, the company may at its discretion make a quotation for repair of the equipment or declare that the equipment is beyond repair.

PASSING OF RISK AND TITLE

The passing of risk for any supply made by the company shall occur at the time of delivery. The title however shall not pass to the buyer until payment has been received in full by the company. And no other sums whatever shall be due from the customer to Nallatech.

If the customer (who shall in such case act on his own account and not as agent for Nallatech) shall sell the goods prior to making payment in full for them, the beneficial entitlement of Nallatech therein shall attach to the proceeds of such sale or to the claim for such proceeds.

The customer shall store any goods owned by Nallatech in such a way that they are clearly identifiable as Nallatech's property and shall maintain records of them identifying them as Nallatech's property. The customer will allow Nallatech to inspect these records and the goods themselves upon request.

In the event of failure by the customer to pay any part of the price of the goods, in addition to any other remedies available to Nallatech under these terms and conditions or otherwise, Nallatech shall be entitled to repossess the goods. The customer will assist and allow Nallatech to repossess the goods as aforesaid and for this purpose admit or procure the admission of Nallatech or its employees and agents to the premises in which the goods are situated.

INTELLECTUAL PROPERTY

The buyer agrees to preserve the Intellectual Property Rights (IPR) of the company at all times and that no contract for supply of goods involves loss of IPR by the company unless expressly offered as part of the contract by the company.

GOVERNING LAW

This agreement and performance of both parties shall be governed by Scottish law.

Any disputes under any contract entered into by the company shall be settled in a court if the company's choice operating under Scottish law and the buyer agrees to attend any such proceedings. No action can be brought arising out of any contract more than 12 months after the completion of the contract.

INDEMNITY

The buyer shall indemnify the company against all claims made against the company by a third party in respect of the goods supplied by the company.

SEVERABILITY

If any part of these terms and conditions is found to be illegal, void or unenforceable for any reason, then such clause or Section shall be severable from the remaining clauses and Sections of these terms and conditions which shall remain in force.

NOTICES

Any notice to be given hereunder shall be in writing and shall be deemed to have been duly given if sent or delivered to the party concerned at its address specified on the invoice or such other addresses as that party may from time to time notify in writing and shall be deemed to have been served, if sent by post, 48 hours after posting.

SOFTWARE LICENSING AGREEMENT

Nallatech Ltd. software is licensed for use by end users under the following conditions. By installing the software you agree to be bound by the terms of this license. If you do not agree with the terms of this license, do not install the Software and promptly return it to the place where you obtained it.:

1. **License**: Nallatech Ltd. grants you a licence to use the software programs and documentation in this package(“Licensed materials”). If you have a single license, on only one computer at a time or by only one user at a time;

   if you have acquired multiple licenses, the Software may be used on either stand alone computers or on computer networks, by a number of simultaneous users equal to or less than the number of licenses that you have acquired; and, if you maintain the confidentiality of the Software and documentation at all times.

2. **Restrictions**: This software contains trade secrets in its human perceivable form and, to protect them, except as permitted by applicable law, you may not reverse engineer, disassemble or otherwise reduce the software to any human perceivable form. You may not modify, translate, rent, lease, loan or create derivative works based upon the software or part thereof without a specific run-time licence from Nallatech Ltd.

3. **Copyright**: The Licensed Materials are Copyrighted. Accordingly, you may either make one copy of the Licensed Materials for backup and/or archival purposes or copy the Licensed Materials to another medium and keep the original Licensed Materials for backup and/or archival purposes. Additionally, if the package contains multiple versions of the Licensed Materials, then you may only use the Licensed Materials in one version on a single computer. In no event may you use two copies of the Licensed Materials at the same time.

4. **Warranty**: Nallatech Ltd. warrants the media to be free from defects in material and workmanship and that the software will substantially conform to the related documentation for a period of ninety (90) days after the date of your purchase. Nallatech Ltd. does not warrant that the Licensed Materials will be free from error or will meet your specific requirements.

5. **Limitations**: Nallatech Ltd. makes no warranty or condition, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding the Licensed Materials.

   Neither Nallatech Ltd. nor any applicable Licenser will be liable for any incidental or consequential damages, including but not limited to lost profits.

6. **Export Control**: The Software is subject to the export control laws of the United States and of the United Kingdom. The Software may not be shipped, transferred, or re-exported directly or indirectly into any country prohibited by the United States Export Administration Act 1969 as amended, and the regulations there under, or be used for any purpose prohibited by the Act.

DEVELOPERS GUIDE CONDITIONS

Information in this Developers Guide is subject to change without notice. Any changes will be included in future versions of this document. Information within this manual may include technical, typing or printing inaccuracies or errors and no liability will arise therefrom.

This Developers Guide is supplied without warranty or condition, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding the information provided herein.

Under no circumstances will Nallatech Limited be liable for any incidental or consequential damage or expense of any kind, including, but not limited to, loss of profits, arising in connection with the use of the information provided herein.

Index

Remarks Form

We welcome any comments you may have on our product and its documentation. Your remarks will be examined thoroughly and taken into account for future versions of this product.

FUSE C/C++ API Developers Guide NT107-0068 Issue 8	**14/09/04**

Errors Detected

Suggested Improvement

Please send this completed form to:

Nallatech
Boolean House
One Napier Park
Cumbernauld
Glasgow G68 0BH
United Kingdom

If you prefer you may send your remarks via E-mail to support@nallatech.com or by fax to +44 (0) 1236 789599.

If you want Nallatech to reply to your comments, please include your name, address and telephone number.