DM6620 Driver for Windows 98/NT4/2000/XP

User's Manual

version 4.0

RTD Embedded Technologies, Inc.
"Accessing the Analog World"®

SWM-640010004
Rev. B

ISO9001 and AS9100 Certified

RTD Embedded Technologies, INC.
103 Innovation Blvd.
State College, PA 16803-0906

Phone: +1-814-234-8087
FAX: +1-814-234-5218

E-mail
sales@rtd.com
techsupport@rtd.com

web site
http://www.rtd.com

Revision History

05/29/2001	ver. 3.3 - Initial version released
08/06/2003	revision A issued Documented for ISO9000
09/16/2003	revision B issued Version 4.0 released Windows 2000/XP support added Windows 95 support removed Example program CycleDMA rewritten Interface of the following function changed: + OpenBoard6620

DM6620 Driver for Windows 98/NT4/2000/XP
Published by:

RTD Embedded Technologies, Inc.
103 Innovation Blvd.
State College, PA 16803-0906

Copyright 2003 RTD Embedded Technologies, Inc.
All rights reserved
Printed in U.S.A.

The RTD Logo is a registered trademark of RTD Embedded Technologies. cpuModule and utilityModule are trademarks of RTD Embedded Technologies. PS/2, PC/XT, PC/AT and IBM are trademarks of International Business Machines Inc. MS-DOS, Windows, Windows 95, Windows 98, Windows NT, Windows 2000 and Windows XP are trademarks of Microsoft Corp. PC/104 is a registered trademark of PC/104 Consortium. All other trademarks appearing in this document are the property of their respective owners.

Table of Contents

TABLE OF CONTENTS **4**

INTRODUCTION **5**

INSTALLATION **6**

INSTALLATION OF THE DRIVER AND EXAMPLE PROGRAMS 6
THE DM6620 BOARD DRIVER 7
THE DRIVER CONFIGURATION 7
INSTALLING A SECOND BOARD IN THE SYSTEM 8
WORKING WITH OTHER BOARDS USING THE WINRT DRIVER 8
USING WINDOWS REGISTRY KEYS 9

THE DRIVER API FUNCTIONS **11**

USING THE DRIVER WITHOUT THE WINDOWS-GUI **12**

DM6620 FEATURES **13**

INTERRUPTS 13
DIRECT MEMORY ACCESS (DMA) 13

DRIVER API FUNCTION GROUPS **15**

DRIVER INITIALIZATION FUNCTIONS 15
GENERAL BOARD CONTROL FUNCTIONS 15
FIFO MANIPULATION 15
DMA FUNCTIONS 15
D/A CONVERTER 15
SAMPLE COUNTER FUNCTIONS 16
DIGITAL I/O FUNCTIONS 16
INTERRUPT HANDLING 16
USER TIMER-COUNTER FUNCTIONS 16
ERROR HANDLING FUNCTIONS 16

ALPHABETICAL DRIVER API FUNCTIONS REFERENCE **17**

EXAMPLE PROGRAMS REFERENCE **42**

WIN32 WINDOWS APPLICATIONS 42

LIMITED WARRANTY **43**

Introduction

The DM6620 board driver for Windows 98/NT4/2000/XP was designed for programmers who write Windows-based application programs for the RTD’s DM6620 board.

The driver provides an Application Programming Interface with a lot of function calls to perform all the data acquisition tasks of the board users.

The board driver based on the BlueWater System's WinRT device driver kit.

There are example programs to demonstrate the various board features and the usage of the driver API. The example programs are written in Microsoft Visual C++ ver. 6.0.

Installation

Installation of the Driver and Example Programs

Before installing the driver and example program files, you need to install the DM6620 board in your PC. Please follow the instructions of the manufacturer, how to install the board in a computer.

To install the drivers for the DM6620 under Windows 98/2000/XP you need the installation diskette 0. The readme.txt file on this diskette describes the necessary installation steps.

For Windows NT4.0 the installation diskette 0 is not needed.

On the installation diskette 1 you can found the **Setup.exe** program, which installs on your PC the DM6620 board driver and example programs. The setup program automatically detects your operating system and installs the appropriate files on your PC.
After starting the setup, please follow the instructions on the screen to install the programs. You can select the directory where to install the files.
The setup also adds to the 'Start menu' under the 'Programs' folder of your Windows system the 'RTD Embedded Technologies' folder. It contains shortcuts to the example programs, the driver configuration utility and the readme.txt file.

The example programs will be installed in the target directory under the 'Examples' folder. This folder contains the example program sources for the DM6620 board, and the project files for Microsoft Visual C++ users to rebuild the programs. The example programs are compiled with the Microsoft Visual C++ ver. 6.0. In case of different version of Visual C is installed on your PC, please rebuild the executable files.

Uninstallation: you can uninstall the DM6620 driver and example programs under the 'Control panel' with the 'Add/Remove Programs' tool.

The DM6620 board driver

The RTD's DM6620 board driver handles the hardware through the BlueWater's WinRT driver. The WinRT driver provides the low-level access to the board, and the RTD's Drvr6620.dll provides the device API for the programmers, and communicates with the hardware through the WinRT driver.

The RTD board drivers has multi-board feature. It means that it is possible to use more than one board in the system.

The base address of the DM6620 board can be set by DIP-switch. If you are using multiple boards in your system, you need assign a different base address to each board. The IRQ and DMA channels are set programmatically. When you install a WinRT driver for the board, you must select the IRQ and DMA channel, and during the operating of the board must be used this IRQ and DMA channels.

Each board requires a unique WinRT device to handle the hardware. The maximum number of devices is 32.

During the installation the Setup.exe program set the default driver values:
Board 1: 0x300 base address, DMA channel 5, DMA buffer size 0x4000 and IRQ 11

If these settings are not proper to your PC configuration, you can modify the settings with the **Drvr_cfg.exe** program in Windows NT4.0.
In case of Windows 98/2000/XP the operating system uses a WDM driver and it can be configured in the Control Panel/System/[Hardware]/Device Manager.

The Driver Configuration

As in the previous chapter mentioned, after installing the driver the default settings are used.

If these settings are not proper to your PC configuration, or there are other RTD boards used, you can modify the settings.
Under Windows NT driver can be configured with the **Drvr_cfg.exe** program. This program is installed on your PC during the setup process and it can be started from the Start menu.

Program options:

- Device number: The number of device, which handles the board. Each board has a unique device number between 0 and 9.
- I/O port base address: The I/O base address of the board.
- IRQ number: The IRQ number using of the board.
- DMA channel: The DMA channel using of the board.
- DMA buffer size: The DMA buffer size. It should not be more than 64k.
- Register: Press this button to store the settings in the Windows registry.
- Refresh: Press this button to get the information from the registry about the selected device number.

Under Windows 98/2000/XP same changes can be made using Control Panel/System/[Hardware]/Device Manager/WinRT Devices.

Do not forget, that all boards are required different driver devices, and in your C++ program must be used the DEVICE_NO according to the driver device number.

Installing a second board in the system

It is possible to use more than one board in the system using the RTD's board drivers. Please follow the next steps to install a multi-board – system:

- Install the first board with all software and make sure that everything is working well.
- Place the second board in an empty slot in the PC, and switch on the computer.
  Windows NT4.0: After the system startup start the installation program and answer 'YES' to the next board installation question. Restarting the system with the two-board example program is possible to test the working of both boards. Please note that during the second board installation the driver and example program files will be refreshed thus you need to save the important changes before starting the install.
  Windows 98/2000/XP: You need only the installation diskette 0 to install the next boards. Follow the steps described in the readme.txt file on diskette 0.

Working with Other Boards Using the WinRT Driver

As you may have in your computer other RTD boards, which uses the WinRT driver, or a board from other manufacturer with WinRT, our installation program automatically detects the WinRT settings from the system registry file, and installs the appropriate (next) WinRT device. You can see an informational dialog box about this issue during the setup.
As an application written for the DM6620 board should know which WinRT device is assigned to the board, the installation program generates the file wrtdev.txt in the 'Examples' directory containing the actual device number. The example programs are reading this file at the startup to identify this number.
If the other board in your system, which uses the WinRT driver does not handle the situation when there is other WinRT-based application, install this board firstly, and then install the DM6620 which handles correctly this state.

The RTD board drivers are using the next registry keys:

Using Windows registry keys

The RTD board drivers are using the next registry keys:

Windows NT 4.0:

```
HKEY_LOCAL_MACHINE
|
| -- System
     |
     | -- CurrentControlSet
          |
          | -- Services
               |
               | -- WinRT
                    |
                    | -- WinRTdev0       (one per device)
                    |     |
                    |     | -- Parameters
                    |           |
                    |           | -- Section0

                    | -- WinRTdev1
                    |     |
                    |     | -- Parameters
                    |           |
                    |           | -- Section0
                    |
```

Windows 98 Enum key path:

```
HKEY_LOCAL_MACHINE
|
| -- Enum
```

Windows 98 Class key path:

```
HKEY_LOCAL_MACHINE
|
| -- System
     |
     | -- CurrentControlSet
          |
          | -- Services
               |
               | -- Class
                    |
                    | -- WinRTDevices
```

Windows 2000/XP device class information:

```
HKEY_LOCAL_MACHINE
|
| -- System
     |
     | -- CurrentControlSet
          |
          | -- Control
               |
               | -- Class
                    |
                    | -- {D695ED6A-630D-4D83-D8B-F1F0AC107AD0}
                    |
                    | -- 0001
```

Windows 2000/XP Plug and Play devices are enumerated under the Enum key:

```
HKEY_LOCAL_MACHINE
|
| -- System
   |
   | -- CurrentControlSet
        |
        | -- Enum
             |
             | -- PCI
```

The Driver Api Functions

The resources on the DM6620 board can be accessed from Windows through the driver API (Application Programming Interface) functions. The executable code of these functions is located in the Drvr6620.dll file. To write applications using the API functions you must include the Reg6620.H header file, and link the program with the Drvr6620.lib import library file.
In the example programs you can find different examples how to use the driver.

Using the Driver without the Windows-GUI

This driver is based on the Win32 system, and can run only under Windows. However, the user, who is not familiar with the Windows Graphical User Interface, can use the driver API functions in Win32 console applications, which has an MS-DOS like text-mode interface.

In console applications is not necessary to use the Windows graphical environment, but all the driver's API functions are accessible.
The Microsoft Visual C++ compiler supports writing console applications.

DM6620 Features

Interrupts

Controller can receive interrupt request from up to 16 sources. These 16 sources cover the most important internal signals of the board plus 2 external signals:

- DAC1 FIFO half full
- DAC2 FIFO half full
- DAC3 FIFO half full
- DAC4 FIFO half full
- Sample counter 1
- Sample counter 2
- Sample counter 3
- DMA5 done
- DMA6 done
- DMA7 done
- User Timer/Counter 0 Out
- User Timer/Counter 1 Out
- User Timer/Counter 1 Out, inverted
- User Timer/Counter 2 Out
- Digital interrupt
- External interrupt

To service interrupts with the DM6620 Driver is very simple.

Actually, when you install your interrupt handler, it is not a real Interrupt Service Routine, since you can not do that under Windows. Your handler is an ordinary routine, which is called by the real Interrupt Service Routine implemented by the DM6620 driver. The driver's Interrupt Service Routine schedules your interrupt handler for execution as a separate thread, acknowledges the interrupt to the board's interrupt controller and returns. After returning, the scheduled thread starts executing your interrupt handler.

There are two main interrupt handler types. If you install a counter-type IRQ handler, on interrupt a counter will be incremented to signal your application about the IRQ event number. The other type is the callback-type. In this case on interrupt the interrupt handler function will call your own function, which can contain any processing routines.

Direct Memory Access (DMA)

DMA transfers data between peripheral device and PC memory without using the processor as an intermediate. This method allows very fast data transfer rates.

The DM6620 driver provides an interface for the user to handle the DMA channels of the board.

The following program steps are necessary to use the DMA:

- Initialize the board
- Install the DMA handler.

  InstallDMA6620(DEVICE_NO, &dmasetup, TRUE, 0, 0);

- Setup DMA channel and source
  SetDMASource6620(DEVICE_NO, DMA_CHAN, DA_CHAN);
- Start the DMA
  StartDMA6620(DEVICE_NO, DMA_BUF_SIZE, 0);
- Process data
- Close DMA channel.
  DeInstallDMA6620(DEVICE_NO, 0);

To see how to use the DMA channel on the DM6620 board, see the cycledma example programs.

Driver API Function Groups

Driver Initialization Functions

OpenBoard6620
CloseBoard6620

General Board Control Functions

InitBoard6620
ClearBoard6620
ClearRegister6620
ReadStatus6620
ReadBoardID6620

FIFO Manipulation

ClearDACFIFO6620
ResetDACFIFO6620
IsDAC1FIFOEmpty6620
IsDAC1FIFOHalf6620
IsDAC1FIFOFull6620
IsDAC2FIFOEmpty6620
IsDAC2FIFOHalf6620
IsDAC2FIFOFull6620
IsDAC3FIFOEmpty6620
IsDAC3FIFOHalf6620
IsDAC3FIFOFull6620
LoadDACFIFO6620
LoadDACFIFOWithMarker6620

DMA Functions

InstallDMA6620
DeInstallDMA6620
StartDMA6620
FlushDMA6620
LoadDMARegister6620
SetDMASource6620
IsDMA5Done6620
IsDMA6Done6620
IsDMA7Done6620
ClearDMADone6620

D/A Converter

LoadDACRangeRegister6620
SetDACRange6620
SetDACRange6620
LoadDACControlRegister6620
SetDACUpdate6620
SetDACCycle6620
UpdateAll6620
UpdateDAC6620

Sample Counter Functions

LoadSampleCounter6620
LoadSampleCounterSourceRegister6620
SetSampleCounterSource6620

Digital I/O Functions

DIOSelectRegister6620
DIOClearChip6620
DIOClearIrq6620
DIOEnableIrq6620
DIOSetPort1Direction6620
DIOSetPort0Direction6620
DIOLoadMask6620
DIOLoadCompare6620
DIORead6620
DIOReadCompareRegister6620
DIOSelectClock6620
DIOSelectIrqMode6620
DIOWrite6620
DIOIsChipIRQ6620

Interrupt Handling

InstallCallbackIRQHandler6620
InstallCounterIRQHandler6620
RemoveIRQHandler6620
GetIRQCounter6620
ClearIRQ6620
IsDigitalIRQ6620
LoadIRQRegister6620
SetIRQ6620

User Timer-Counter Functions

SelectTimerCounter6620
ClockMode6620
ClockDivisor6620
SetUserClock6620
ReadTimerCounter6620
DoneTimer6620

Error Handling Functions

GetErrorStatus6620

Alphabetical Driver API Functions Reference

C

ClearBoard6620

void ClearBoard6620(RTDHANDLE hBoard);

Description:

This routine is used to clear the board.

Parameters:

hBoard:	device handle

ClearDACFIFO6620

void ClearDACFIFO6620(RTDHANDLE hBoard, uint16 DAC_Channel);

Description:

This routine is used to clear all the data from the selected DAC's FIFO.

Parameters:

hBoard:	device handle
DAC_Channel:	0, 1, 2, 3

ClearDMADone6620

void ClearDMADone6620(RTDHANDLE hBoard, uint16 DMA_Channel);

Description:

This routine is used to clear the selected DMA done status bit.

Parameters:

hBoard:	device handle
DMA_Channel:	5, 6, 7

ClearIRQ6620

void ClearIRQ6620(RTDHANDLE hBoard, uint16 IRQ_Select);

Description:

This routine is used to clear the selected IRQ circuitry.

Parameters:

hBoard:	device handle
IRQ_Select:	0, 1

ClearRegister6620

void ClearRegister6620(RTDHANDLE hBoard, uint16 ClearValue);

Description:

This routine is used to write the clear register with one command and issue a clear to the board.

Parameters:

hBoard:	device handle
ClearValue:	1 - 65535

ClockDivisor6620

void ClockDivisor6620(RTDHANDLE hBoard, BYTE Timer, uint16 Divisor);

Description:

This routine is used to set the divisor of a designated counter on the 8254 programmable interval timer (PIT). This procedure assumes that the counter has already been set to receive the least significant byte (LSB) of the divisor followed by the most significant byte (MSB).

Parameters:

hBoard:	device handle
Timer:	0,1,2
Divisor:	0 - 65535

ClockMode6620

void ClockMode6620(RTDHANDLE hBoard, BYTE Timer, BYTE Mode);

Description:

This routine is used to set the mode of a designated counter in the 8254 programmable interval timer (PIT).

Parameters:

hBoard:	device handle
Timer:	0,1,2
Mode:	0,1,2,3,4,5

CloseBoard6620

BOOL CloseBoard6620 (LONG DEVICE_NO, LPSTR szBuf);

Description:

This routine is used to close board driver.

Parameters:

hBoard:	device handle
szBuf:	message buffer

Return value:

TRUE on success, FALSE on error

DeInstallDMA6620

BOOL DeInstallDMA6620(RTDHANDLE hBoard, int dma_ch_index);

Description:

This routine is used to remove DMA handler.

Parameters:

hBoard:	device handle
dma_ch_index:	DMA channel index (0/1)

Returns:

0 if success
error code if not success

DIOClearChip6620

void DIOClearChip6620(RTDHANDLE hBoard);

Description:

This routine clears the DIO chip.

Parameters:

hBoard:	device handle

DIOClearIrq6620

void DIOClearIrq6620(RTDHANDLE hBoard);

Description:

This routine clears the DIO chip's IRQ status bit.

Parameters:

hBoard:	device handle

DIOEnableIrq6620

void DIOEnableIrq6620(RTDHANDLE hBoard, BYTE Enable);

Description:

This routine enables the DIO chip's interrupt.

Parameters:

hBoard:	device handle
Enable:	0 = Disabled 1 = Enabled

DIOIsChipIRQ6620

BYTE DIOIsChipIrq6620(RTDHANDLE hBoard);

Description:

This routine checks to see if the DIO chip has generated an interrupt.

Parameters:

hBoard:	device handle

Returns:

1 if IRQ has been generated
0 if no IRQ

DIOLoadCompare6620

void DIOLoadCompare6620(RTDHANDLE hBoard, BYTE Compare);

Description:

This routine loads the DIO chip's compare register.

Parameters:

hBoard:	device handle
Compare:	0 - 255

DIOLoadMask6620

void DIOLoadMask6620(RTDHANDLE hBoard, BYTE Mask);

Description:

This routine loads the DIO chip's mask register.

Parameters:

hBoard:	device handle
Mask:	0 - 255

DIORead6620

BYTE DIORead6620(RTDHANDLE hBoard, BYTE Port);

Description:

This routine reads the DIO chip's port.

Parameters:

hBoard:	device handle
Port:	0, 1

DIOReadCompareRegister6620

BYTE DIOReadCompareRegister6620(RTDHANDLE hBoard);

Description:

This routine reads the DIO chip's compare register.

Parameters:

hBoard:	device handle

DIOSelectClock6620

void DIOSelectClock6620(RTDHANDLE hBoard, BYTE Clock);

Description:

This routine sets the DIO chip's clock source.

Parameters:

hBoard:	device handle
Clock:	0 = 8 MHz 1 = Programmable clock

DIOSelectIrqMode6620

void DIOSelectIrqMode6620(RTDHANDLE hBoard, BYTE IrqMode);

Description:

This routine sets the DIO chip's Irq mode.

Parameters:

hBoard:	device handle
IrqMode:	0 = Event mode 1 = Match mode

DIOSelectRegister6620

void DIOSelectRegister6620(RTDHANDLE hBoard, BYTE Select);

Description:

This routine picks the different registers on the DIO chip.

Parameters:

hBoard:	device handle
Select:	0 = Clear mode
	1 = Port 0 direction
	2 = Port 0 mask
	3 = Port 0 compare

DIOSetPort0Direction6620/ DIOSetPort1Direction6620

void DIOSetPort0Direction6620(RTDHANDLE hBoard, BYTE Direction);
void DIOSetPort1Direction6620(RTDHANDLE hBoard, BYTE Direction);

Description:

This routine sets the DIO chip's port 1 direction.

Parameters:

hBoard:	device handle
Direction:	0 = In
	1 = Out

DIOWrite6620

void DIOWrite6620(RTDHANDLE hBoard, BYTE Port, BYTE Data);

Description:

This routine writes the DIO chip's port.

Parameters:

hBoard:	device handle
Port:	0, 1
Data:	0 - 255

DoneTimer6620

void DoneTimer6620(RTDHANDLE hBoard);

Description:

Initialize the timers for high speed to ensure the immediate load.

Parameters:

hBoard:	device handle

FlushDMA6620

void FlushDMA6620(RTDHANDLE hBoard);

Description:

This routine flushes the DMA buffer.

Parameters:

hBoard: device handle

GetErrorStatus6620

BOOL GetErrorStatus6620 (RTDHANDLE hBoard, LONG *ErrorCode,
LPSTR ErrorString);

Description:

This routine is used to get the error code and string in case of errors.
The function clears the last error code.

Parameters:

hBoard:	device handle
ErrorCode:	Error Code
ErrorString:	Return error string

Return value:

TRUE if the error code not 0
FALSE if the error code 0

GetIRQCounter6620

void GetIRQCounter6620(RTDHANDLE hBoard, IRQSetup *irqsetup);

Description:

This routine is used to get the counter incremented by IRQ.

Parameters:

hBoard:	device handle
irqsetup:	return parameters to caller

InitBoard6620

void InitBoard6620(RTDHANDLE hBoard);

Description:

This Routine Should always be called first.
This clears the board and variables the driver uses.

Parameters:

hBoard: device handle

InstallCallbackIRQHandler6620

BOOL InstallCallbackIRQHandler6620 (RTDHANDLE hBoard, isr_t handler);

Description:

This routine is used to install IRQ handler function.

Parameters:

hBoard: device handle
isrT handler: user callback function

Return value:

TRUE on success, FALSE on error

InstallCounterIRQHandler6620

BOOL InstallCounterIRQHandler6620(RTDHANDLE hBoard, IRQSetup *irqsetup);

This routine is used to install IRQ handler function.

Parameters:

hBoard: device handle
irqsetup: return parameters to caller

Return value:

TRUE on success, FALSE on error

InstallDMA6620

BOOL InstallDMA6620(RTDHANDLE hBoard, DMASetup *dmasetup, BOOL fill,
UCHAR fill_byte, int dma_ch_index);

Description:

This routine is used to install DMA handler.

Parameters:

hBoard:	device handle
dmasetup:	return parameters to caller
fill:	TRUE if buffer fill is necessary
fill_byte:	DMA buffer filler
dma_ch_index:	DMA channel index (0/1)

Return value:

TRUE on success, FALSE on error

IsDAC1FIFOEmpty6620/ IsDAC2FIFOEmpty6620/ IsDAC3FIFOEmpty6620/ IsDAC4FIFOEmpty6620

```
uint16  IsDAC1FIFOEmpty6620(RTDHANDLE hBoard);
uint16 IsDAC2FIFOEmpty6620(RTDHANDLE hBoard);
uint16 IsDAC3FIFOEmpty6620(RTDHANDLE hBoard);
uint16 IsDAC4FIFOEmpty6620(RTDHANDLE hBoard);
```

Description:

This routine checks to see if the DAC FIFO is empty.

Parameters:

hBoard:	device handle

Return value:

1 if FIFO is empty
0 if FIFO is not empty

IsDAC1FIFOHalf6620/ IsDAC2FIFOHalf6620/ IsDAC3FIFOHalf6620/ IsDAC4FIFOHalf6620

```
uint16 IsDAC1FIFOHalf6620(RTDHANDLE hBoard);
uint16 IsDAC2FIFOHalf6620(RTDHANDLE hBoard);
uint16 IsDAC3FIFOHalf6620(RTDHANDLE hBoard);
uint16 IsDAC4FIFOHalf6620(RTDHANDLE hBoard);
```

Description:

This routine checks to see if the DAC FIFO is half full.

Parameters:

hBoard:	device handle

Return value:

1 if FIFO is half full

0 if FIFO is half full

IsDAC1FIFOFull6620/ IsDAC2FIFOFull6620/ IsDAC3FIFOFull6620/ IsDAC4FIFOFull6620

---

uint16 IsDAC1FIFOFull6620(RTDHANDLE hBoard);
uint16 IsDAC2FIFOFull6620(RTDHANDLE hBoard);
uint16 IsDAC3FIFOFull6620(RTDHANDLE hBoard);
uint16 IsDAC4FIFOFull6620(RTDHANDLE hBoard);

Description:

This routine checks to see if the DAC FIFO is full.

Parameters:

hBoard: device handle

Return value:

1 if FIFO is full
0 if FIFO is full

IsDigitalIRQ6620

---

uint16 IsDigitalIRQ6620(RTDHANDLE hBoard);

Description:

This routine checks to see if the digital I/O chip has generated an interrupt.

Parameters:

hBoard: device handle

Return value:

1 if IRQ has been generated
0 if no IRQ

IsDMA5Done6620/ IsDMA6Done6620/ IsDMA7Done6620

---

uint16 IsDMA5Done6620(RTDHANDLE hBoard);
uint16 IsDMA6Done6620(RTDHANDLE hBoard);
uint16 IsDMA7Done6620(RTDHANDLE hBoard);

Description:

This routine checks to see if the DMA transfer is done.

Parameters:

hBoard: device handle

Return value:

1 if DMA is done
0 if DMA is not done

LoadDACControlRegister6620

void LoadDACControlRegister6620(RTDHANDLE hBoard, uint16 Value);

Description:

This routine loads the DAC Control register with one write operation.

Parameters:

hBoard:	device handle
Value:	0 - 65535

LoadDACFIFO6620

void LoadDACFIFO6620(RTDHANDLE hBoard, uint16 Channel, int16 Data);

Description:

This routine loads the specified DAC FIFO.

Parameters:

hBoard:	device handle
Channel:	0 = DAC1
	1 = DAC2
	2 = DAC3
	3 = DAC4
Data:	0 - 4095

LoadDACFIFOWithMarker6620

void LoadDACFIFOWithMarker6620(RTDHANDLE hBoard, uint16 Channel, int16 Data,
uint16 Marker);

Description:

This routine loads the specified DAC FIFO with DAC data and Marker.

Parameters:

hBoard:	device handle
Channel:	0 = DAC1
	1 = DAC2
	2 = DAC3
	3 = DAC4
Data:	0 - 4095
Marker:	0 or 1

LoadDACRangeRegister6620

void LoadDACRangeRegister6620(RTDHANDLE hBoard, uint16 Value);

Description:

This routine loads the DAC Range register with one write operation.

Parameters:

hBoard:	device handle
Value:	0 - 65535

LoadDMARegister6620

void LoadDMARegister6620(RTDHANDLE hBoard, uint16 Value);

Description:

This routine loads the DMA register with one write operation.

Parameters:

hBoard:	device handle
Value:	0 - 65535

LoadIRQRegister6620

void LoadIRQRegister6620(RTDHANDLE hBoard, uint16 Value);

Description:

This routine loads the interrupt register with one write operation.

Parameters:

hBoard:	device handle
Value:	0 - 65535

LoadSampleCounter6620

void LoadSampleCounter6620(RTDHANDLE hBoard, uint16 Counter, uint16 NumOfSamples);

Description:

This routine loads the selected sample counter.

Parameters:

hBoard:	device handle
Counter:	0, 1, 2
NumOfSamples:	05535

void LoadSampleCounterSourceRegister6620(RTDHANDLE hBoard, uint16 Value);

Description:

This routine loads the Sample Counter Source register with one write operation.

Parameters:

hBoard:	device handle
Value:	0 - 65535

OpenBoard6620

BOOL OpenBoard6620(int num, int device_id, LPSTR szBuf,
BoardConfig *boardconfig);

Description:

This routine is used to open the board for use.

Parameters:

num:	the number of the board (a system can have more board)
device_id:	board id equals 0x6620
szBuf:	return string
BoardConfig:	pointer to the structure to return board configuration

Return value:

TRUE on success, FALSE on error

ReadBoardID6620

uint16 ReadBoardID6620(RTDHANDLE hBoard);

Description:

This routine reads the board ID.

Parameters:

hBoard device handle
`Board 0 to (NUMBER_OF_BOARDS - 1)`

Returns:

16 bit unsigned integer

ReadStatus6620

uint16 ReadStatus6620(RTDHANDLE hBoard);

Description:

This routine returns the status byte from the board.

Parameters:

hBoard device handle

Returns:

16 bit unsigned integer

ReadTimerCounter6620

uint16 ReadTimerCounter6620(RTDHANDLE hBoard, BYTE Timer, BYTE Clock);

Description:

This routine is used to read the contents of the selected timer/counter.

Parameters:

hBoard device handle
Timer: 0,1
Clock: 0,1,2

Returns:

Returns the 16-bit count value.

RemoveIRQHandler6620

void RemoveIRQHandler6620(RTDHANDLE hBoard);

Description:

This routine is used to uninstall the IRQ handler function.

Parameters:

hBoard:	device handle

ResetDACFIFO6620

void ResetDACFIFO6620(RTDHANDLE hBoard, uint16 DAC_Channel);

Description:

This routine is used to reset the internal FIFO pointer back to the first location in the FIFO.

Parameters:

hBoard:	device handle
DAC_Channel:	0, 1, 2, 3

SelectTimerCounter6620

void SelectTimerCounter6620(RTDHANDLE hBoard, uint16 Select);

Description:

This routine selects one of the four 8254 timer chips.

Parameters:

hBoard:	device handle
Select:	0 = Clock TC (Pacer & Burst clocks)
	1 = User TC (A/D sample counter & User timer/counters)
	2 = reserved
	3 = reserved

SetDMASource6620

void SetDMASource6620(RTDHANDLE hBoard, uint16 DMA_Channel, uint16 Source);

Description:

This routine sets the source for the DMA channel.

Parameters:

hBoard:	device handle
DMA_Channel:	5 = DMA channel 5
	6 = DMA channel 6
	7 = DMA channel 7
Source:	0 = disabled
	1 = DAC1
	2 = DAC2
	3 = DAC3
	4 = DAC4
	5 = reserved
	6 = reserved
	7 = reserved

SetIRQ6620

void SetIRQ6620(RTDHANDLE hBoard, uint16 IRQ_Select, uint16 Source, uint16 Channel);

Description:

This routine sets the source and channel for the interrupt circuit.

Parameters:

hBoard:	device handle
IRQ_Select:	0 = IRQ 1 circuit
	1 = IRQ 2 circuit
Source:	0 = DAC1 FIFO half full
	1 = DAC2 FIFO half full
	2 = DAC3 FIFO half full
	3 = DAC4 FIFO half full
	4 = Sample counter 1
	5 = Sample counter 2
	6 = Sample counter 3
	7 = DMA 5 done
	8 = DMA 6 done
	9 = DMA 7 done
	10 = User TC out 0
	11 = User TC out 1
	12 = User TC out 1 inverted
	13 = User TC out 2
	14 = Digital chip interrupt
	15 = External interrupt
	16 .. 31 = reserved
Channel:	0 = disabled
	3 = IRQ 3
	5 = IRQ 5
	9 = IRQ 9
	10 = IRQ 10
	11 = IRQ 11
	12 = IRQ 12
	15 = IRQ 15

SelectTimerCounter6620

void SelectTimerCounter6620(RTDHANDLE hBoard, uint16 Select);

Description:

This routine selects one of the two 8254 timer chips.

Parameters:

hBoard:	device handle
Select:	0 = Sample Counters
	1 = User TC

SetDACCycle6620

void SetDACCycle6620(RTDHANDLE hBoard, uint16 DAC_Channel, uint16 Enable);

Description:

This routine enables and disables the cycle mode for the selected DAC.

Parameters:

hBoard:	device handle

DAC_Channel:	0, 1, 2, 3
Enable:	0 = no cycle 1 = cycle

SetDACRange6620

void SetDACRange6620(RTDHANDLE hBoard, uint16 DAC_Channel, uint16 Range);

Description:

This routine sets the output range for selected DAC.

Parameters:

hBoard:	device handle
DAC_Channel:	0, 1, 2, 3
Range:	0 = 0 to +5 volts 1 = 0 to +10 volts 2 = +/-5 volts 3 = +/-10 volts

SetDACUpdate6620

void SetDACUpdate6620(RTDHANDLE hBoard, uint16 DAC_Channel, uint16 Update);

Description:

This routine sets the update source for the selected DAC.

Parameters:

hBoard:	device handle
DAC_Channel:	0, 1, 2, 3
Update:	0 = Software 1 = Software (update all) 2 = User TC out 0 3 = User TC out 1 4 = User TC out 2 5 = External Clock (CN3-39) 6 = External Interrupt (CN3-19) 7 = reserved

SetDACUpdateEnable6620

void SetDACUpdateEnable6620(RTDHANDLE hBoard, uint16 DAC1, uint16 DAC2,
uint16 DAC3, uint16 DAC4);

Description:

This routine is used to enable and disable the DAC update signal.

Parameters:

hBoard:	device handle

DAC1:	0 = disabled
	1 = enabled
DAC2:	0 = disabled
	1 = enabled
DAC3:	0 = disabled
	1 = enabled
DAC4:	0 = disabled
	1 = enabled

SetSampleCounterSource6620

void SetSampleCounterSource6620(RTDHANDLE hBoard, uint16 Counter, uint16 Source);

Description:

This routine sets the source for the selected Sample Counter.

Parameters:

hBoard:	device handle
Counter:	0, 1, 2
Source:	0 = DAC1
	1 = DAC2
	2 = DAC3
	3 = DAC4

SetUserClock6620

float32 SetUserClock6620(RTDHANDLE hBoard, BYTE Timer, float32 InputRate,
float32 OutputRate);

Description:

This routine sets the user timer counters.

Parameters:

Timer:	0, 1, 2
InputRate:	Input clock to selected timer.
OutputRate:	Desired output rate from selected timer.

Returns:

The actual clock frequency that is programmed.

StartDMA6620

void StartDMA6620(RTDHANDLE hBoard, ULONG UserLength, int dma_ch_index);

Description:

This routine starts DMA.

Parameters:

hBoard:	device handle
UserLengt:	user definable dma length
dma_ch_index:	DMA channel index (0/1)

UpdateAll6620

void UpdateAll6620(RTDHANDLE hBoard);

Description:

This routine is used to create a software trigger that will update all of the D/A outputs simultaneously.

Parameters:

hBoard:	device handle

UpdateDAC6620

void UpdateDAC6620(RTDHANDLE hBoard, uint16 Channel);

Description:

This routine is used to create a software trigger that will update only the specified D/A output.

Parameters:

hBoard:	device handle
Channel:	0 = DAC1
	1 = DAC2
	2 = DAC3
	3 = DAC4

Example Programs Reference

Win32 Windows applications

Name	Feature	Remarks
cycle	This program generates waveforms on the analog output.	Sample program that demonstrates how to load the FIFO with data and use the cycle mode to generate a waveform.
cycledma	This program generates waveforms on the analog output.	Sample program that demonstrates how to load the FIFO with DMA to generate a waveform.
Dac	Sample program that demonstrates how to use the analog output.	This program demonstrates how to use the digital to analog converters on the A/D board.
Digital	Sample program that demonstrates how to use the analog output.	Read and write the digital I/O.
Interrupt	This program uses the sample counter to generate an interrupt, which signals the computer to load new data in the FIFO.	Waveform generation.
timers	Demonstration of user timer-counters.	The counters are programmed to count the elapsed time in seconds.

Limited Warranty

RTD Embedded Technologies, Inc. warrants the hardware and software products it manufactures and produces to be free from defects in materials and workmanship for one year following the date of shipment from RTD Embedded Technologies, INC. This warranty is limited to the original purchaser of product and is not transferable.

During the one year warranty period, RTD Embedded Technologies will repair or replace, at its option, any defective products or parts at no additional charge, provided that the product is returned, shipping prepaid, to RTD Embedded Technologies. All replaced parts and products become the property of RTD Embedded Technologies. Before returning any product for repair, customers are required to contact the factory for an RMA number.

THIS LIMITED WARRANTY DOES NOT EXTEND TO ANY PRODUCTS WHICH HAVE BEEN DAMAGED AS A RESULT OF ACCIDENT, MISUSE, ABUSE (such as: use of incorrect input voltages, improper or insufficient ventilation, failure to follow the operating instructions that are provided by RTD Embedded Technologies, "acts of God" or other contingencies beyond the control of RTD Embedded Technologies), OR AS A RESULT OF SERVICE OR MODIFICATION BY ANYONE OTHER THAN RTD Embedded Technologies. EXCEPT AS EXPRESSLY SET FORTH ABOVE, NO OTHER WARRANTIES ARE EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND RTD Embedded Technologies EXPRESSLY DISCLAIMS ALL WARRANTIES NOT STATED HEREIN. ALL IMPLIED WARRANTIES, INCLUDING IMPLIED WARRANTIES FOR MECHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED TO THE DURATION OF THIS WARRANTY. IN THE EVENT THE PRODUCT IS NOT FREE FROM DEFECTS AS WARRANTED ABOVE, THE PURCHASER'S SOLE REMEDY SHALL BE REPAIR OR REPLACEMENT AS PROVIDED ABOVE. UNDER NO CIRCUMSTANCES WILL RTD Embedded Technologies BE LIABLE TO THE PURCHASER OR ANY USER FOR ANY DAMAGES, INCLUDING ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, EXPENSES, LOST PROFITS, LOST SAVINGS, OR OTHER DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PRODUCT.

SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES FOR CONSUMER PRODUCTS, AND SOME STATES DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

RTD Embedded Technologies, Inc.
103 Innovation Blvd.
State College PA 16803-0906
USA
Our website: www.rtd.com