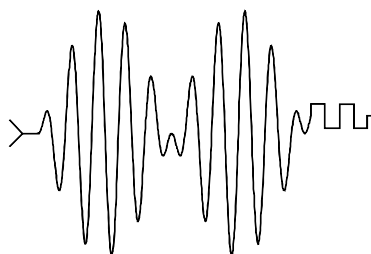


# **PCI4520/DM7520 Data Acquisition Driver for Windows 95/98/ME/NT4/2000/XP**

**Version 5.02**

**User's Manual**

**Real Time Devices USA, Inc.**



**rttd**®  
USA

**Real Time Devices USA, Inc.**  
*"Accessing the Analog World"®*



**REAL TIME DEVICES USA, INC.**

103 Innovation Blvd  
State College, PA 16803

Phone: +1-814-234-8087

FAX: +1-814-234-5218

E-mail

sales@rtdusa.com

techsupport@rtdusa.com

web site

<http://www.rtdusa.com>

## Revision History

---

09.11.2001 Version 4.1 released  
28.11.2001 Version 4.2 released.  
New function: DACClockFreeRun4520  
07.12.2001 Version 4.4 released  
Fully tested on Windows 2000.  
13.12.2001 Version 4.5 released  
The detected board ID is displayed on the programs title bar.  
22.01.2002 Spm\_DM workspace path corrected  
11.04.2002 highspped, control and spm\_dm\_RealTime examples added  
01.05.2002 Windows XP support added  
17.03.2003 McBSP port handling functions added  
24.03.2002 Minor Corrections  
10.04.2003 SetPacerRateF4520, SetPacrClockF4520, SetBurstRateF4520, SetupBurstF4520 function added  
15.04.2003 SetDACRateF4520 function added; InstallCounterIRQHandler4520 and GetIRQCounter4520 functions removed

Publication PCI4520/DM7520 driver manual

Published by:

Real Time Devices USA, Inc.  
103 Innovation Boulevard  
State College, PA 16803

Copyright 2001 by Real Time Devices, Inc.  
All rights reserved  
Printed in U.S.A.

The Real Time Devices USA Logo is a registered trademark of Real Time Devices USA. dspModule, cpuModule, and utilityModule are trademarks of Real Time Devices USA. PC/104 is a registered trademark of PC/104 Consortium. TMS320C62x, VelociTI, and C62x are trademarks of Texas Instruments. All other trademarks appearing in this document are the property of their respective owners.



# Table of Contents

<b>TABLE OF CONTENTS</b> .....	<b>V</b>
<b>INTRODUCTION</b> .....	<b>7</b>
<b>INSTALLATION</b> .....	<b>7</b>
INSTALLATION OF THE DRIVER AND EXAMPLE PROGRAMS .....	7
THE PCI4520 BOARD DRIVER .....	9
INSTALLING A SECOND BOARD IN THE SYSTEM .....	9
WORKING WITH OTHER BOARDS USING THE WINRT DRIVER .....	9
THE DRIVER API FUNCTIONS.....	9
USING THE DRIVER WITHOUT THE WINDOWS GUI.....	10
REGISTRY SETTINGS .....	11
<b>PCI4520 FEATURES</b> .....	<b>14</b>
MEASUREMENT SCENARIOS .....	14
CHANNEL-GAIN CIRCUITRY .....	15
INTERRUPTS .....	18
PRE AND POST TRIGGERING .....	20
ADVANCED DIGITAL TRIGGER .....	23
SYNCHRON BUS .....	23
DATA TRANSFER WITH DMA .....	24
<b>DRIVER API FUNCTION GROUPS</b> .....	<b>25</b>
DRIVER INITIALIZATION FUNCTIONS .....	25
GENERAL BOARD CONTROL FUNCTIONS.....	25
FIFO MANIPULATION .....	25
A/D CONVERTER .....	25
CHANNEL-GAIN TABLE MANIPULATION .....	25
PACER CLOCK HANDLING .....	26
BURST CLOCK HANDLING .....	26
DELAY COUNTER FUNCTIONS .....	26
ABOUT COUNTER FUNCTIONS .....	26
SAMPLE COUNTER FUNCTIONS.....	26
D/A CONVERTER .....	26
DIGITAL I/O FUNCTIONS .....	27
HIGH-SPEED DIGITAL INPUT FUNCTIONS .....	27
USER INPUT/OUTPUT FUNCTIONS.....	27
USER TIMER-COUNTER FUNCTIONS .....	28
INTERRUPT HANDLING .....	28
SYNCHRON BUS CONFIGURATION FUNCTIONS .....	28
EXTERNAL TRIGGER CONFIGURATION FUNCTIONS .....	28
DMA FUNCTIONS.....	28
MCBSP PORT CONTROL FUNCTIONS .....	29
<b>ALPHABETICAL DRIVER API FUNCTIONS REFERENCE</b> .....	<b>31</b>
<b>EXAMPLE PROGRAMS REFERENCE</b> .....	<b>115</b>
W2BOARDS GUI (MFC) EXAMPLE.....	117

WCGT GUI (MFC) EXAMPLE .....	117
WIODMA GUI (MFC) EXAMPLE .....	117
WDAC GUI (MFC) EXAMPLE .....	118
WDAC2 GUI (MFC) EXAMPLE .....	118
WDIGITAL GUI (MFC) EXAMPLE.....	118
WDMAN GUI (MFC) EXAMPLE .....	119
WDMAN_CHAINED GUI (MFC) EXAMPLE.....	119
WDMAN_DEMAND GUI (MFC) EXAMPLE.....	119
WDMAOUT GUI (MFC) EXAMPLE .....	120
WDMAOUT_DEMAND GUI (MFC) EXAMPLE.....	120
WIODMA GUI (MFC) EXAMPLE .....	120
WINTRPTS GUI (MFC) EXAMPLE.....	121
WRANDOM GUI (MFC) EXAMPLE .....	121
WAMLTSCAN GUI (MFC) EXAMPLE .....	121
WAMLTBRST GUI (MFC) EXAMPLE.....	122
WBURSTN GUI (MFC) EXAMPLE .....	122
WSOFTTRIG GUI (MFC) EXAMPLE.....	122
WTIMERS GUI (MFC) EXAMPLE .....	122
SPM_DM GUI (MFC) EXAMPLE .....	123
SPM_DM_REALTIME GUI (MFC) EXAMPLE .....	123
SPM_DM_SERIAL GUI (MFC) EXAMPLE.....	123
DRVR_DEMO GUI (MFC) EXAMPLE .....	123
HIGHSPEED GUI (MFC) EXAMPLE.....	123
CONTROL GUI (MFC) EXAMPLE.....	124
MLSSA GUI (MFC) EXAMPLE .....	124
<b>LIMITED WARRANTY.....</b>	<b>127</b>

## Introduction

The PCI4520 and the DM7520 boards have the same functionality on a different form-factor. The PCI4520 board is a PCI bus board for desktop computers; the DM7520 board is a PCI bus board for *PC/104 Plus* computers.

There are also some minor differences between the PCI4520 and DM7520. In the hardware manual are detailed these differences and you can find a note for some driver functions.

The driver version 4.1 and above intended to use with all type of bus-master PCI data acquisition board of RTD (PCI4520/DM7520/DM7540).

In this manual stands the PCI4520 expression for all boards. The driver functions can be used for any board without modification.

The PCI4520 data acquisition board Windows 95/98/NT driver has designed for programmers who write Windows-based application programs with using the RTD's PCI4520 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 PCI4520 board in your PC. Please follow the instructions of the manufacturer, how to install the board in a computer.

The PCI4520 board is a data acquisition board with PCI bus. The Windows 95/98 operating system and the Windows NT handles differently the plug and play features of a PCI boards, so the installation of the driver is different.

Under the Windows 95/98, Millennium and Windows 2000 operating system after the first installation of the PCI4520 board the system detects the new board and asks for the driver for it. On the installation diskette 0 can be found the registry files to perform the first setup of the board.

Under Windows NT 4.0 you do not need the installation diskette 0.

On the installation diskette 1 you can find the **Setup.exe** program, which installs on your PC the PCI4520 board driver and example programs. The setup program automatically detects your operating system (Windows 95/98/ME/2000 or NT4.0) 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 'Real Time Devices USA' folder. It contains shortcuts to the example programs, the readme.txt file and the manuals in PDF on-line format.

The example programs will be installed in the target directory under the 'Examples' folder. This folder contains the example program sources for the PCI4520 board, and the project files for Microsoft Visual C++ users to rebuild the programs. The example programs are compiled with the Microsoft Visual C++ v. 6.0. In case of different version of Visual C is installed on your PC, you have to rebuild the executable files.

**Uninstallation:** you can uninstall the PCI4520 driver and example programs under the 'Control panel' with the 'Add/Remove Programs' tool.

## **The PCI4520 Board Driver**

The RTD's PCI4520 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 PCI4520.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.

Each board requires a unique WinRT device to handle the hardware. The maximum number of devices is 32, on Windows 95 is 10.

## **Installing a second board in the system**

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

- Install the first board with all the software and make sure that everything is working well.
- Place the second board in an empty PCI slot in the PC, and switch on the computer. If the operating system is Windows 95/98/ME or Windows 2000, then the system automatically recognizes the board and setups the registry.

In case of NT 4.0 the installation program will make the necessary registry settings. After the system startup start the installation program and answer 'YES' to the second 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.

*Do not install a WinRT driven board between two similar RTD boards. The installation program and the two-board example program suppose that these boards will be installed one after the other, and they are using successive driver ID's.*

## **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, the RTD's 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.

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 PCI4520/DM7520 which handles correctly this state.

## **The Driver API Functions**

The resources on the PCI4520 board can be accessed from Windows through the driver API (Application Programming Interface) functions. The executable code of these functions is located in the PCI4520.DLL file.

To write applications using the API functions you must include the PCI4520.H header file, and link the program with the PCI4520.LIB import library file. In the example programs, you can find examples on using 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.

## Registry settings

The RTD board drivers are using different the next registry key settings in the different Windows systems:

### Windows NT4.0:

```
HKEY_LOCAL_MACHINE
|
| -- System
|   |
|   | -- CurrentControlSet
|   |   |
|   |   | -- Services
|   |   |   |
|   |   |   | -- EventLog
|   |   |   |   |
|   |   |   |   | -- System
|   |   |   |   |   |
|   |   |   |   |   | -- WinRT
|   |   |   |   |   |   |
|   |   |   |   |   |   | -- event message file description
|   |   |   |   |   |
|   |   |   |   | -- WinRT
|   |   |   |   |   |
|   |   |   |   |   | -- driver start mode information
|   |   |   |   |   | -- WinRTdev0 (one per device with board parameters)
|   |   |   |   |   |   |
|   |   |   |   |   |   | -- Parameters
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   | -- Section0
|   |   |   |   |   |
|   |   |   |   | -- WinRTdev1
|   |   |   |   |   |
|   |   |   |   |   | -- Parameters
|   |   |   |   |   |   |
|   |   |   |   |   |   | -- Section0
```

## Windows 95:

```
HKEY_LOCAL_MACHINE
| -- System
|   | -- CurrentControlSet
|     | -- Services
|       | -- Class
|         | -- WinRT
|           | -- 0000 (one per board with information about
|             |         the driver VXD)
|             | -- 0001
|             | -- WinRTdev0 (one per device with board
|               |         parameters)
|               |   | -- Parameters
|               |   |   | -- Section0
|               |   |   | -- WinRTdev1
|               |   |   |   | -- Parameters
|               |   |   |   |   | -- Section0
```

## Windows 2000:

```
HKEY_LOCAL_MACHINE
| -- System
|   | -- CurrentControlSet
|     | -- Control
|       | -- Class
|         | -- {D695ED6A-630D-4D83-D8B-F1F0AC107AD0}
|           | -- 0000 (one per device with board parameters)
```

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

## Windows 95 and ME:

HKEY\_LOCAL\_MACHINE

|  
| -- Enum

HKEY\_LOCAL\_MACHINE

|  
| -- System  
| | -- CurrentControlSet  
| | | -- Services  
| | | -- Class  
| | | | -- WinRTDevices  
| | | | | --0000(one per device with board parameters)

## PCI4520 Features

### ***Measurement Scenarios***

Through selecting different options for A/D Conversion Trigger (SetConversionSelect4520), Burst Clock Start Trigger (SetBurstStart4520), Pacer Clock Start/Stop Trigger (SetPacerStart4520/SetPacerStop4520) and creating different Channel-Gain Tables, you have innumerable sampling scenarios. The following bullets try to enumerate only the most frequently used measurement setups.

❑ ***Single Conversion***

In this mode, a single channel is sampled whenever StartConversion4520 is called. The Channel Gain Latch (see WriteCGTLatch4520) specifies the channel to sample. This is the easiest scenario of all. It can be used in a variety of applications, such as sample every time a key is pressed on the keyboard, sample with each iteration of a loop, or watch the system clock and sample every five seconds.

❑ ***Multiple Conversions***

In this mode, conversions are continuously performed at the rate of the Pacer Clock, or other selected A/D Conversion Signal rate. The pacer clock can be internal or external. The maximum rate supported by the board is 1.25MHz. The Pacer Clock can be turned on and off using any of the start and stop triggering modes using the functions SetPacerStart4520 and SetPacerStop4520. If you use the internal pacer clock, you must program it to run at the desired rate (SetPacerClock4520).

This mode is ideal for filling arrays, acquiring data for a specified period, and taking a specified number of samples.

❑ ***Random Channel Scan***

In this mode, the Channel-Gain Table is incrementally scanned through, with each selected A/D Conversion Signal pulse starting a conversion at the channel and gain specified in the current table entry. Before starting a conversion sequence Channel Gain Table, you need to load the table with the desired data. Then make sure that the Channel-Gain Table is enabled by the function EnableCGT4520. This enables the A/D portion of the Channel Gain Table. If you are using the Digital Table as well, you must also enable this using the function EnableCGTDigital4520. Each rising edge of selected A/D Conversion Signal starts a conversion using the current Channel Gain data and then increments to the next position in the table. When the last entry is reached, the next pulse starts the table over again.

❑ ***Programmable Burst***

In this mode, a single trigger initiates a scan of the entire Channel-Gain Table. Before starting a burst of the Channel-Gain Table, you need to load the table with the desired data. Then enable the Channel-Gain Table by EnableCGT4520. This enables the A/D portion of the channel-gain table. If you are using the Digital Table as well, you must also call EnableCGTDigital4520.

Burst is used when you want one sample from a specified number of channels for each trigger. The burst trigger starts the Burst Clock and the Burst Clock initiates each conversion. At high speeds, the burst mode emulates simultaneous sampling of multiple input channels. For time critical simultaneous sampling applications, a simultaneous sample-and-hold board can be used (SS8 eight-channel boards are available from Real Time Devices).

❑ **Programmable Multi-Scan**

This mode - when the A/D Conversion Start Signal is the Burst Clock - lets you scan the Channel Gain Table after a Burst Clock Start Signal. When the Channel Gain Table is empty, the Burst Clock is stopped, and will wait for a new Start Signal.

**Channel-Gain Circuitry**

Channel-Gain Tables are traditionally for implementing random channel scan analog input on boards where a single A/D converter is multiplexed for 8, 16 or more analog input channels. PCI4520 features a more advanced *Channel-Gain Circuitry*, which actually provides synchronous Analog and Digital I/O.

The Channel-Gain Circuitry embeds a 1024x24 bit on-board memory (Channel Scan Memory), called *Channel-Gain Table (CGT)* for historical reasons. Every 24-bit row (entry) in a CGT is an instruction executed by the Channel-Gain circuitry. Execution happens at a programmable rate. Channel-Gain Latch (CGL), provided for easy, single channel analog input, can be perceived as a special, single row CGT for the following description. Unless explicitly indicated, explanation holds for the CGL, as well.

The table below pictures the format of a CGT entry:

DO	Skip	D/A2	D/A1	Pause	Se/Diff	Range	NRSE	Gain	Channel
8 bits	1 bit	1 bit	1 bit	1 bit	1 bit	2 bits	1 bit	3 bits	4 bits

**Channel                      Analog Input Channel**

Specifies the Analog Input channel to sample. Depending on your configuration you may have 8 differential channels (AIN1...AIN8), or 16 Single-Ended channels (AIN1...AIN16)

**Gain                              Analog Input Gain**

This field specifies the gain to apply to the input. Available choices are 1x, 2x, 4x, 8x, 16x, 32x, 64x, 128x. (The 64x and 128x gain are not available on the DM7520 board.)

**Range                            Analog Input Range**

Specifies one of the three supported input ranges: ±5 Volts, ±10 Volts or 0...10Volts.

**Se/Diff                          Analog Input Type**  
**NRSE                            Non-Referenced Single Ended Bit**

These two bits together determine the type of the analog input channel.

**Ground Referenced Single-Ended (GRSE):                      Se/Diff=0, NRSE=0**

This mode is suggested only for floating signal sources to avoid the ground loops. For this input type, the reference signal of Instrumentation Amplifier is the Analog Ground.

To configure a GRSE analog input, connect the high side of the input signal to the selected analog input channel (AIN1...AIN16), and connect the low side to any of the Analog Ground pins of the I/O connector. If you use the channels AIN9...AIN16, switch the appropriate SW1 and SW2 dips off. See Hardware Manual for more.

**Non Referenced Single-Ended (NRSE):                              Se/Diff=0, NRSE=1**

This mode can be used for grounded signal sources and for floating sources as well. For this input type, the reference signal of Instrumentation Amplifier is the AINSENSE signal. In the case of floating sources, an external resistor is needed to ground the AINSENSE signal.

To configure an NRSE analog input, connect the high side of the input signal to the selected analog input channel (AIN1...AIN16), and connect the low side to the AINSENSE pin of the I/O connector. If you use channels AIN9...AIN16, switch the appropriate SW1 and SW2 dips off. See Hardware Manual for more.

**Differential (DIFF):**

**Se/Diff=1, NRSE=x**

This mode can be useful when the shielding of the signal is important. In differential mode, you use two analog input channels of the board for an analog input. AIN<sub>x</sub> and AIN<sub>x+8</sub>, also referred to as AIN<sub>x+</sub> and AIN<sub>x-</sub>, can be used in couple for a differential analog input. This way you may have up to 8 differential analog inputs. For this input type, the reference signal of Instrumentation Amplifier is the AIN-signal.

To configure an NRSE analog input, connect the high side of the analog input to the selected analog input channel (AIN1+...AIN8+), and connect the low side to the corresponding AIN- pin (AIN1-/AIN9...AIN8-/AIN16). See Hardware Manual for more.

**Pause**

**Pause Bit**

If this bit is set and pausing is enabled by the EnableCGTPause4520 function, execution of the Channel-Gain Table stops *after* executing this entry. Execution is resumed with the next CGT entry when the programmed Pacer Clock start trigger occurs.

EXAMPLE: Pause Bit can be used when you have two sequence of entries, each to be executed on a different event (trigger). Suppose that CGT is driven by the Pacer Clock, and the Pacer Clock is started on the External Trigger. The External Trigger comes from a device, whose pulses indicate two different events. Odd pulses indicate an event, on which you want to react by sampling AIN1 and AIN2, on even pulses you want to sample AIN3, AIN4 and AIN5. In this case, you would create a 5 entry CGT:

- Entry #1: AIN1, Pause Bit = 0*
- Entry #2: AIN2, Pause Bit = 1*
- Entry #3: AIN3, Pause Bit = 0*
- Entry #4: AIN4, Pause Bit = 0*
- Entry #5: AIN5, Pause Bit = 1*

In this case, the first pulse on the External Trigger line starts executing the CGT at the rate of the Pacer Clock. After executing the first two entries, execution stops and is waiting for the next External Trigger pulse. The second pulse resumes execution, and entries #3, #4 and #5 are executed at the rate of the Pacer Clock. Execution pauses again, after executing entry #5. A third External trigger pulse continues execution with entry #1, and so on.

NOTE: When the Channel-Gain Latch is used, or in burst mode, Pause Bit is ignored.

**D/A1**

**D/A2**

**D/A Update Bits**

These bits can be used to simultaneously update the D/A converter(s) with the sampling of the appropriate analog input channel. When a D/A Update Bit is set, a D/A update signal is generated when A/D conversion starts.

## **Skip Skip Bit**

When the Skip Bit is set, the entry is skipped, which means that the A/D conversion is performed but the resulting sample is not written into the A/D FIFO. This feature provides a way to sample multiple channels at different rates without saving unwanted data.

**EXAMPLE:** In this example, we want to sample AIN1 in every second and AIN4 in every three seconds. For this end, we must create CGT with six entries:

*Entry #1: AIN1, Skip Bit = 0*  
*Entry #2: AIN4, Skip Bit = 1*  
*Entry #3: AIN1, Skip Bit = 0*  
*Entry #4: AIN4, Skip Bit = 1*  
*Entry #5: AIN1, Skip Bit = 0*  
*Entry #6: AIN4, Skip Bit = 0*

Next, we set the Pacer Clock to run at 2 Hz (0.5 seconds). This allows us to sample each channel once per second, the maximum sampling rate required by one of the channels (pacer clock rate = number of different channels sampled x fastest sample rate).

The first Pacer Clock pulse starts an A/D conversion according to the parameters set in the first entry of the Channel-Gain Table, and each successive clock pulse incrementally steps through the table entries. The first clock pulse takes a sample on AIN1. The second pulse looks at the second entry in the table and sees that the Skip Bit is set. Sample is taken, but is not stored in the FIFO. The third pulse takes a sample on AIN1 again, the fourth pulse skips the next entry, and the fifth pulse takes our third reading on AIN1. On the sixth pulse, the Skip Bit is disabled, AIN4 is sampled and sample is stored to the FIFO. Then the sequence starts over again with entry #1. Samples are not stored when they are not wanted, saving memory and eliminating the need to throw away unwanted data.

**NOTE:** When the Channel-Gain Latch is used, Skip Bit is ignored.

## **DO 8-Bit Digital Table**

The digital portion of the Channel-Gain Table, also referred to as Digital Table, can be used to control input expansion boards such as the TMX32 Analog Input Expansion board. The expansion board is driven at the same speed as the A/D conversions are performed, with no software overhead.

**EXAMPLE:** Let us consider the following simple example on driving an analog input expansion board.

In this example, we have a TMX32 expansion board connected to AIN1 on the PCI4520. We have three signals to sample, one is connected to the first channel of the expansion board (EAIN1), the second is connected to the fourth channel of the expansion board (EAIN4) and the third is connected directly to AIN2 of the PCI4520.

We need to create the following Channel-Gain Table:

```

Entry      #1:      AIN1,      gain=1,      DO=0
Entry      #2:      AIN1,      gain=4,      DO=3
Entry #3: AIN2, gain=1, DO=3

```

Execution, starting with entry #1, samples AIN1 and simultaneously outputs 0 on Digital Port 1. This will cause the expansion board to switch to EAIN1.

Entry #2 will sample AIN1, which is now connected to EAIN1, and simultaneously outputs 3 on Digital Port 1. As a result, the expansion board switches to EAIN4.

Next, entry #3 samples AIN2 and outputs 3 on Digital Port 1, which makes the expansion board to switch (again) to EAIN4.

When executing entry #1 again, AIN1 is sampled which is now connected to EAIN4, and so on.

To load the Channel-Gain Table you need to follow the steps below:

```

SetupCgtRow4520(&entry,0,GAIN1,AIN_BIP5,0,GND_SE,0,0,0,0);
WriteCGTAnalog4520(0,entry);           // entry#1, analog
WriteCGTDigital4520(0,0);               // entry#1, digital
SetupCgtRow4520(&entry,0,GAIN4,AIN_BIP5,0,GND_SE,0,0,0,0);
WriteCGTAnalog4520(0,entry);           // entry#2, analog
WriteCGTDigital4520(0,3);               // entry#2, digital
SetupCgtRow4520(&entry,1,GAIN1,AIN_BIP5,0,GND_SE,0,0,0,0);
WriteCGTAnalog4520(0,entry);           // entry#3, analog
WriteCGTDigital4520(0,3);               // entry#3, digital
EnableCGT4520(0,CSC_CGT);               // enable CGT (vs. CGL)
EnableCGTDigital4520(0,CSC_CGT);        // enable Digital Table

```

**NOTE:** If you only need to use the A/D part of the table, you do not have to program the Digital Table. However, if you only want to use the Digital part of the table, you must program the A/D part of the table.

**NOTE:** When the Channel-Gain Latch is used, Digital Table is ignored.

When using the Channel Gain Table, you should group your entries to maximize the throughput of your module. Low-level input signals and varying gains are likely to drop the throughput rate because low level inputs must drive out high level input residual signals. To maximize throughput:

- Keep channels configured for a certain range grouped together, even if they are out of sequence.
- Use external signal conditioning if you are performing high speed scanning of low level signals. This increases throughput and reduces noise.
- If you have room in the channel-gain table, you can make an entry twice to make sure that sufficient settling time has been allowed and an accurate reading has been taken. Set the skip bit for the first entry so that it is ignored.
- For best results, do not use the channel-gain table when measuring steady-state signals. Use the single convert mode to step through the channels.

## ***Interrupts***

PCI4520 features a Priority Interrupt Controller with interrupt-overrun protection.

Controller can receive interrupt request from up to 15 sources. These 15 sources cover the most important internal signals of the board plus 3 external signals (highest priority interrupt source comes first):

- ❑ A/D FIFO Write  
Interrupt is generated when sample enters the A/D FIFO.  
This interrupt can be used for reading and processing samples real-time.
- ❑ CGT Reset  
Interrupt is generated when the Channel-Gain Table recycles execution to the first table entry.  
This interrupt can be used for reading and processing a burst of samples from different channels real-time.
- ❑ CGT Pause  
Interrupt is generated when Channel-Gain Table execution is paused waiting for a new trigger.
- ❑ About Counter Countdown Interrupt is generated when the About Counter counts down to zero. This interrupt can be used to detect (and react on) the end of sampling when doing Pre/Port Triggering.
- ❑ Delay Counter Countdown  
Interrupt is generated when the Delay Counter counts down to zero.  
This interrupt can be used to detect (and react on) the actual start of sampling when doing Pre/Port Triggering.
- ❑ A/D Sample Counter Countdown
- ❑ Interrupt is generated when the A/D Sample Counter counts down to zero.  
This interrupt can be used to count more than 65535 samples by counting the turnovers of the Sample Counter.  
D/A1 Update Counter Countdown
- ❑ D/A2 Update Counter Countdown  
Interrupt is generated when the D/A1 or D/A2 Update Counter counts down to zero.  
This interrupt can be used to count D/A conversions and do something on a timely basis, e.g., writing new data to the D/A FIFO.
- ❑ User Timer/Counter 1 Out
- ❑ User Timer/Counter 1 Out, inverted
- ❑ User Timer/Counter 2 Out  
Interrupt is generated on the ticks of User T/C1 (i.e., when the counter counts down to zero).  
This interrupt gives you a general-purpose means of measuring real time, frequency, or counting events. It is also intended to use for Pulse output generation.
- ❑ Digital Interrupt  
Interrupt is generated when the Advanced Digital Trigger signals a Digital Interrupt.  
This interrupt can be used to detect (and react on) certain patterns on Digital Input Port 0.
- ❑ External Interrupt
- ❑ Interrupt is generated on the rising edge of the External Interrupt pin of the I/O connector.  
This interrupt is intended for exporting events from an external device.  
External Trigger rising-edge
- ❑ External Trigger falling-edge  
Interrupt is generated on a rising/falling edge of the External Trigger pin.  
This interrupt is intended for the Gated Mode operation of the Pacer Clock.

Because of the several interrupt sources on the board, a Priority Interrupt Controller was built to assure even usage all of the interrupt sources.

When the Interrupt Controller receives an interrupt request, it transmits it to the PC Interrupt Controller (to the IRQ line, which is assigned to the board by the operating system).

In the Interrupt Service Routine, reading the Interrupt Status Register via a call to `GetITStatus4520`, you can identify the causing interrupt source. In this register, there is a single non-zero bit that indicates the signaling interrupt source. If more than one interrupt source generates an interrupt at the same time, this register will indicate the highest priority one. The lower priority request is queued and will appear after acknowledging the higher priority one.

After identifying the source, the interrupt can be serviced. After servicing, the request must be acknowledged by calling `ClearITMask4520`. Failing to do this makes the same interrupt reoccur repeatedly.

In normal operation, the next interrupt request comes later than acknowledging the current one. However, if the next interrupt comes before acknowledging the current one, i.e., interrupt overrun occurs, you must be aware of it. For this end, `PCI4520` has an Interrupt Overrun Register. Reading this register (see `GetITOverrun4520`) after acknowledging the request enables you to determine a possible interrupt overflow.

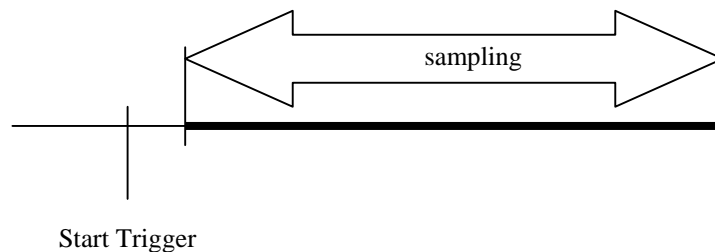
To service interrupts with the `PCI4520` driver you do not have to deal with all these chores. Actually, when you install your interrupt handler, it is not a real Interrupt Service Routine, since you can not do that under Windows 95/98/NT. Your handler is an ordinary routine, which is called by the real Interrupt Service Routine implemented by the `PCI4520` 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.

## ***Pre and Post Triggering***

`PCI4520` offers various triggers for pre/post (about) triggering. The triggering mechanism, built around the Pacer/Burst Clock circuitry, lets you implement various triggering schemes.

### **□ *Pre-Triggering***

Pre-triggering scheme enables you to start sampling with some delay after an event (trigger).



This scheme can be useful when you do not want the first couple of samples after the important event. This can be the case when you know that these samples are noisy or meaningless because of settling issues.

This scheme is realized by using the Delay Counter. Follow the steps below to set up for this scheme:

1. Set up the Pacer Clock to start delayed on the desired event.

```
SetPacerStart4520(hBoard, PCLK_START_D_xxx);
```

2. Initialize the Delay Counter for the required number of samples to wait (Delay Counter counts at the rate of the Pacer Clock).

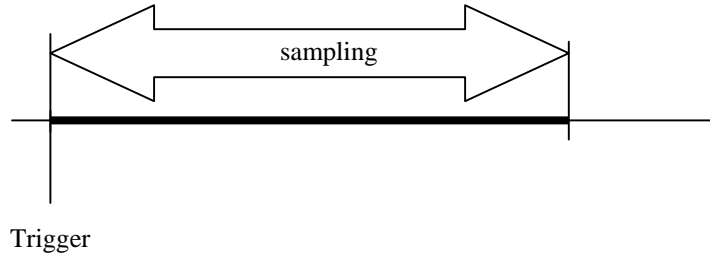
```
SetupDelayCounter4520 (hBoard, samples) ;
```

3. Arm the Pacer Clock to be ready for receiving the start trigger.

```
StartPacer4520 (hBoard) ;
```

□ **Post-Triggering**

Post-triggering scheme enables you to stop sampling with some delay after an event (trigger).



This scheme is for situations where you want to take a known number of samples after an important event.

This scheme is realized by using the About Counter. Follow the steps below to set up for this scheme:

1. Allocate a buffer, which is big enough to hold the desired number of samples (nsamples) after the event.
2. Set up the Pacer Clock to start on the desired event and to stop when About Counter counts down to zero.

```
SetPacerStart4520 (hBoard, PCLK_START_SOFTWARE) ;
```

```
SetPacerStop4520 (hBoard, PCLK_STOP_A_xxx) ;
```

3. Initialize the About Counter for the number of samples you want to be collected after your trigger event.

```
SetupAboutCounter4520 (hBoard, nsamples, TRUE) ;
```

4. Set up a mechanism to copy samples from the A/D FIFO to your sample buffer. For example, install an interrupt handler to copy data by 512 samples:

```
SetupSampleCounter4520 (hBoard, TC_ADC_SCNT,
```

```
ADC_SCNT_FIFO_WRITE,
```

```
512, M8254_RATE_GENERATOR) ;
```

```
InstallCallbackIRQHandler4520 (hBoard, handler) ;
```

```
SetITMask4520 (hBoard, IRQM_AD_SCNT) ;
```

5. Arm the Pacer Clock to be ready for receiving the start trigger.

```
StartPacer4520 (hBoard) ;
```

When the trigger occurs, About Counter will start and sampling stops after nsamples. At this point, your sample buffer will contain the data you needed.

If you want to take more than 65,535 samples after the trigger, you also need to do the following:

3. Change About Counter setup to disable it stopping when it counts down to zero and install an interrupt handler on About Counter Countdown.

```
SetupAboutCounter4520 (hBoard, nsamples, FALSE) ;
```

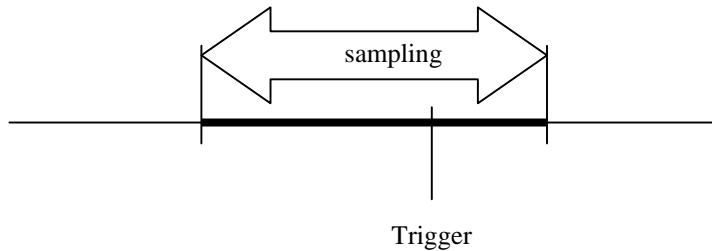
```
InstallCallbackIRQHandler4520 (hBoard, handler) ;
```

6. In the About Counter interrupt handler, count the Countdowns and enable About Counter to stop when the required number of samples has been taken.

```
EnableAcntStop4520 (hBoard, TRUE) ;
```

□ **About Triggering**

About-triggering scheme enables you to take samples around (before and after) an event (trigger). This is also called about triggering. This scheme is very frequently used to take samples around an important event.



This scheme is realized by using the About Counter. Follow the steps below to set up for this scheme:

1. Allocate a buffer, which is big enough to hold samples to take before the event (nsamples1) and after the event (nsamples2).
2. Set up the Pacer Clock to start from software, and to stop on the desired trigger event.

```
SetPacerStart4520 (hBoard, PCLK_START_SOFTWARE) ;
SetPacerStop4520 (hBoard, PCLK_STOP_A_XXX) ;
```

3. Initialize the About Counter for the number of samples you want to be collected after your trigger event.

```
SetupAboutCounter4520 (hBoard, nsamples2, TRUE) ;
```

4. Set up a mechanism to copy samples from the A/D FIFO to your sample buffer. For example, install an interrupt handler to copy data by 512 samples:

```
SetupSampleCounter4520 (hBoard, TC_ADC_SCNT,
                        ADC_SCNT_FIFO_WRITE,
                        512, M8254_RATE_GENERATOR) ;
```

```
InstallCallbackIRQHandler4520 (hBoard, handler) ;
```

```
SetITMask4520 (hBoard, IRQM_AD_SCNT) ;
```

5. Start the Pacer Clock.

```
StartPacer4520 (hBoard) ;
```

When the trigger occurs, About Counter will start and sampling stops after nsamples2. At this point, your sample buffer's last nsamples1+nsamples2 elements will contain the data you needed.

If you want to take more than 65,535 samples after the trigger, you also need to do the following:

3. Change About Counter setup to disable it stopping when it counts down to zero and install an interrupt handler on About Counter Countdown.

```
SetupAboutCounter4520 (hBoard, nsamples2, FALSE) ;
```

```
InstallCallbackIRQHandler4520 (hBoard, handler) ;
```

6. In the About Counter interrupt handler, count the Countdowns and enable About Counter to stop when the required number of samples has been taken.

```
EnableAcntStop4520(hBoard, TRUE);
```

## **Advanced Digital Trigger**

The bit-programmable Digital I/O Port 0 supports two *Advanced Digital Interrupt* modes. Port can be programmed to generate an interrupt when selected port lines match a programmed value (*match mode*) or when any of the selected lines changes (*event mode*). Mode is selected by the *SetDIO0CompareMode4520* function.

When Advanced Digital Interrupt modes enabled (see *EnableDIO0Irq4520*), digital lines are sampled at either at 8 MHz or at the rate of User Timer/Counter 1. Clock is selected by the *SetDIO0Clock4520* function. Only lines enabled in the Mask Register (set by *SetDIO0Mask4520*) take part in monitoring. With each clock pulse, the digital circuitry looks at the state of the next Port 0 bits. To provide noise rejection and prevent erroneous interrupt generation because of noise spikes on the digital lines, a change in the state of any bit must be seen for two edges of a clock pulse to be recognized by the circuit.

In *Event Mode*, the enabled Port 0 input lines are monitored for a change. When any of these lines change, an interrupt is generated and the input pattern is latched into the Compare Register. You can read the contents of this register with *GetDIO0CompareValue4520* to see which bit caused the interrupt to occur.

In *Match Mode*, the enabled Port 0 input lines are monitored for the value programmed in the Compare Register with *SetDIO0CompareValue4520* to occur. When lines match this value, an interrupt is generated.

When Advanced Digital Interrupt mode is not enabled (see *EnableDIO0Irq4520*), the Mask Register can be used to preserve a bit's state, regardless of the digital data written to Port 0. In addition, external data can be strobed into Port 0 by connecting a trigger pulse through the External Pacer Clock pin at the External I/O Connector. This data can be read from the Compare Register with *GetDIO0CompareValue4520*.

## **Synchron Bus**

Synchron Bus (SyncBus, for short) is a three-line bus for synchronized multi-board operation.

A SyncBus line enables you to export/import signals to/from other boards. The other boards do not need to be PCI4520 or compatibles. Actually, you can use SyncBus for any purpose and from any place, which can make use of a digital signal.

A SyncBus line may have a single source and may have one or more users (including the source board). To configure a line you must select a signal (source) to put it on the line. To be more technical, consider the following exemplary situation.

**EXAMPLE:** You have 3 PCI4520 boards, A, B and C. You want board A's Pacer Clock signal to be the D/A Update signal on board B, and the Burst Clock start signal on board C. This case you would do the following steps on the three boards:

### **Board A**

1. Set SyncBus line 0 source to Pacer Clock:  
`SetSbus0Source4520(hBoard, SBUS0_PCLK);`
2. Enable SyncBus line 0:  
`EnableSbus04520(hBoard, TRUE);`

### **Board B**

Set D/A1 Update Source to SyncBus line 0:  
`SetDAC1Start4520(hBoard, DAC_START_SBUS0);`

### **Board C**

Set Burst Clock start trigger to SyncBus line 0:

```
SetBurstStart4520(hBoard, BCLK_START_SBUS0);
```

To deconstruct this buildup (remove the Pacer Clock signal from line 0), you would do the following steps on the three boards:

#### **Board A**

Disable SyncBus line 0.

```
EnableSbus04520(hBoard, FALSE);
```

#### **Board B, C**

Need to do nothing. D/A1 on board B, and Burst Clock on board C automatically seize to receive triggers on SyncBus line 0. Disable SyncBus line 0.

## **Data transfer with DMA**

The PCI4520 board provides two independent DMA channel with various modes to perform fast data transfer from the board to the PC memory or from the PC memory to the board.

At the startup of the PC the PCI4520 driver allocates memory for DMA transfers. The default size of the DMA memory is 0x10000 bytes. To change this value do the following:

- change the **DMACommonBufferSize** value in the registry with the regedit.exe program and reboot the PC.

To setup the DMA transfer there are some driver calls to perform:

- **InstallDMA4520**: setup the driver's DMA handling, get information about the DMA buffer address and length.
- **SetupDMA4520**: setup the direction, transfer byte count, DMA mode and source.
- **StartDMA04520/ StartDMA14520**: start the DMA transfer.
- **DeInstallDMA4520**: remove the DMA handler routine.

There are three basic DMA modes, which can be combined together:

- **Normal mode**: the data transfer initiated with software command (StartDMA04520/ StartDMA14520). After this command the hardware transfers the data from the source to the destination memory area.
- **Demand mode**: the data transfer starts on a programmable hardware event. The data transfer must precede a StartDMA04520/ StartDMA14520 command.
- **Chaining mode**: in this mode the DMA works under the control of chained descriptor blocks. Each descriptor block contains the DMA transfer direction, source and destination address, byte count and the address of the next descriptor block. The DMA continues to work until it reads an end of chain bit in the last descriptor block. This mode is useful to perform DMA transfers with different directions or transfer data from various addresses.

In the example programs can be found examples on normal, demand and chaining mode DMA transfers with different directions.

## **Driver API Function Groups**

### ***Driver Initialization Functions***

OpenBoard4520  
CloseBoard4520  
GetErrorStatus4520

### ***General Board Control Functions***

InitBoard4520  
ClearAllIO4520  
IsBusMaster4520  
ReadFirmwareVersion4520

### ***FIFO Manipulation***

GetFifoStatus4520  
IsADFIFOEmpty4520  
IsADFifoHalfFull4520  
IsADFifoFull4520  
IsDINFifoEmpty4520  
IsDINFifoHalfFull4520  
IsDINFifoFull4520  
IsDAC1FifoEmpty4520  
IsDAC1FifoHalfFull4520  
IsDAC1FifoFull4520  
IsDAC2FifoEmpty4520  
IsDAC2FifoHalfFull4520  
IsDAC2FifoFull4520  
ClearDinFifo4520  
ClearADFIFO4520  
ResetDAC1Fifo4520  
ClearDAC1FIFO4520  
ResetDAC2Fifo4520  
ClearDAC2FIFO4520  
ReadADData4520  
ReadADDataWithMarker4520  
ReadDinFifo4520  
WriteDac1Fifo4520  
WriteDac2Fifo4520  
LoadDAC14520  
LoadDAC24520

### ***A/D Converter***

StartConversion4520  
SetConversionSelect4520

### ***Channel-Gain Table Manipulation***

WriteCGTAnalog4520  
WriteCGTLatch4520  
WriteCGTDigital4520  
EnableCGT4520  
EnableCGTDigital4520

EnableCGTPause4520  
ResetCGT4520  
ClearCGT4520  
SetupCgtRow4520  
SetChannelGain4520

### ***Pacer Clock Handling***

StartPacer4520  
StopPacer4520  
ReadPacer4520  
SetPacerStart4520  
SetPacerStop4520  
SetPacerRepeat4520  
SetPclkSize4520  
SetPacerSource4520  
SetPacerRate4520  
SetPacerRateF4520  
SetPacerClock4520  
SetPacerClockF4520  
SelectPacerClockPrimaryClock4520

### ***Burst Clock Handling***

StartBurst4520  
SetBurstStart4520  
SetBurstRate4520  
SetBurstRateF4520  
SetupBurst4520  
SetupBurstF4520  
SelectBurstClockPrimaryClock4520

### ***Delay Counter Functions***

LoadDcnt4520  
SetupDelayCounter4520

### ***About Counter Functions***

EnableAcntStop4520  
LoadAcnt4520  
GetAcntCount4520  
SetupAboutCounter4520

### ***Sample Counter Functions***

SetAdcntSource4520  
GetAdcntCount4520  
SetupSampleCounter4520  
ReadSampleCounter4520  
LoadAdcnt4520

### ***D/A Converter***

UpdateDAC14520  
UpdateDAC24520  
UpdateAllDAC4520  
SetDACDivisor4520

SetDACRate4520  
SetDACRateF4520  
GetDACDivisor4520  
DACClockStartSelect4520  
DACClockStopSelect4520  
DACClockFreeRun4520  
SelectDACClockPrimaryClock4520  
StartDACClock4520  
StopDACClock4520  
SetDAC1Range4520  
SetDAC1Start4520  
SetDAC1Cycle4520  
SetDAC2Range4520  
SetDAC2Start4520  
SetDAC2Cycle4520  
LoadUcnt14520  
LoadUcnt24520  
SetupDAC4520

### ***Digital I/O Functions***

ClearDIO4520  
ClearDIO0IrqStatus4  
EnableDIO0Irq4520  
GetDIO0Clock4520  
GetDIO0CompareMode4520  
GetDIO0CompareValue4520  
GetDIO0Direction4520  
GetDIO1Direction4520  
GetDIO0Mask4520  
GetDIOStatus4520  
PollDIO04520  
ReadDIO04520  
ReadDIO14520  
SelectDIO0Register4520  
SetDIO0Clock4520  
SetDIO0CompareMode4520  
SetDIO0CompareValue4520  
SetupPort04520  
SetupPort14520  
SetDIO0Direction4520  
SetDIO1Direction4520  
SetDIO0Mask4520  
WriteDIO04520  
WriteDIO14520

### ***High-Speed Digital Input Functions***

StartHdin4520  
SetHdinStart4520

### ***User Input/Output Functions***

ReadUserInput4520  
WriteUserOutput4520  
SetUout0Source4520

SetUout1Source4520

### ***User Timer-Counter Functions***

GetTimerStatus4520  
SetUtc0Clock4520  
SetUtc0Gate4520  
SetUtc1Clock4520  
SetUtc1Gate4520  
SetUtc2Clock4520  
SetUtc2Gate4520  
Set8254Divisor4520  
Set8254Mode4520  
Get8254Mode4520  
Get8254Count4520  
Get8254Status4520  
SetupTimerCounter4520  
ReadTimerCounter4520

### ***Interrupt Handling***

GetITStatus4520  
SetITMask4520  
ClearITMask4520  
ReadITOverrun4520  
ClearITOverrun4520  
SetEintPolarity4520  
InstallCallbackIRQHandler4520  
RemoveIRQHandler4520

### ***Synchron Bus Configuration Functions***

SetSbus0Source4520  
EnableSbus04520  
SetSbus1Source4520  
EnableSbus14520  
SetSbus2Source4520  
EnableSbus24520  
SetupSbus4520

### ***External Trigger Configuration Functions***

SetEtrgPolarity4520

### ***DMA functions***

InstallDMA4520  
DeInstallDMA4520  
SetupDMA4520  
SetupDirectDMA4520  
ShareDMABuffer4520  
SetupChainedDMA4520  
SetFirstDescriptorBlock4520  
SetFirstDirectDescriptorBlock4520  
SetNextDescriptorBlock4520  
SetNextDirectDescriptorBlock4520

SetLastDescriptorBlock4520  
SetLastDirectDescriptorBlock4520  
StartDMA04520  
StartDMA14520  
StartChainedDMA04520  
StartChainedDMA14520  
AbortDMA04520  
AbortDMA14520  
GetDMA0Status4520  
GetDMA1Status4520  
GetDMADoneChannel4520  
IsDMA0Done4520  
IsDMA1Done4520  
ResetDMA0State4520  
ResetDMA1State4520

### ***McBSP Port Control Functions***

McBSPADControl4520  
McBSPDACControl4520



## Alphabetical Driver API Functions Reference

---

AbortDMA04520  
AbortDMA14520

---

**Syntax**

```
void AbortDMA04520 ( RTDHANDLE hBoard );
```

**Description**

Aborts (stops) the current DMA data transfer.

**Parameters**

hBoard:                    device handle

---

**ClearADFIFO4520**

---

**Syntax**

```
void ClearADFIFO4520 (RTDHANDLE hBoard);
```

**Description**

Dispose all samples from the A/D FIFO.

NOTE: Reading from an empty FIFO (e.g., with ReadADDData4520, ReadADDDataWithMarker4520) results in deterministic data.

**Parameters**

hBoard: device handle

---

**ClearAllIO4520**

---

**Syntax**

```
void ClearAllIO4520 (RTDHANDLE hBoard);
```

**Description**

Software reset of the board.

**Parameters**

hBoard: device handle

---

**ClearCGT4520**

---

**Syntax**

```
void ClearCGT4520 (RTDHANDLE hBoard);
```

**Description**

Clear A/D Channel-Gain Table.

This function removes all entries from the CGT. Use this function before reprogramming the CGT. This is the only way to remove entries from the CGT. There is no way to insert /remove a single CGT entry or change it in place.

NOTE: This function clears the Digital part of the CGT.

**Parameters**

hBoard: device handle

**See Also**

Channel-Gain Circuitry

---

## ClearDAC1FIFO4520

## ClearDAC2FIFO4520

---

### Syntax

```
void ClearDAC1FIFO4520 (RTDHANDLE hBoard);  
void ClearDAC2FIFO4520 (RTDHANDLE hBoard);
```

### Description

Remove all data from D/A1 or D/A FIFO.

This function sets the update and the write pointer of the FIFO to the beginning of the FIFO. This means that the FIFO is ready to be filled with new data.

NOTE: Updating a D/A Converter (e.g., with UpdateDAC14520 ,UpdateDAC24520, UpdateAllDAC4520) when its FIFO is empty has no effect.

### Parameters

hBoard: device handle

---

## ClearDinFifo4520

---

### Syntax

```
void ClearDinFifo4520 (RTDHANDLE hBoard);
```

### Description

Remove all samples from the High Speed Digital Input FIFO.

NOTE: Reading from an empty FIFO (e.g., with ReadDinFifo4520) results in undeterministic data.

### Parameters

hBoard: device handle

---

## ClearDIO4520

---

### Syntax

```
void ClearDIO4520 (RTDHANDLE hBoard);
```

### Description

Reset the Digital I/O chip.

It programs all I/O lines for input and programs the Advanced Digital Trigger for Event Mode.

NOTE: It does not clear a pending interrupt. You must make a call to ClearDIO0IrqStatus4520 explicitly.

### Parameters

hBoard: device handle

---

## ClearDIOIrqStatus4520

---

### Syntax

```
void ClearDIOIrqStatus4520 (RTDHANDLE hBoard);
```

### Description

Clears the Digital IRQ status flag. You must acknowledge Digital Interrupts by calling this function. Failing to do this, no more Digital Interrupts will be generated by the Digital I/O chip.

### Parameters

hBoard:                device handle

### See Also

Advanced Digital Trigger

---

## ClearITMask4520

---

### Syntax

```
void ClearITMask4520 (RTDHANDLE hBoard, uint16 mask);
```

### Description

Acknowledge interrupts from the specified interrupt sources.

NOTE:                Usually there is no need to call this function, because it is automatically called when you install an interrupt handler. However, if you do not install a handler for an interrupt source, but interrupts are enabled from that source (see SetITMask4520), you may need this function to acknowledge interrupts.

### Parameters

hBoard:                device handle

mask:                    mask of interrupt sources for which to acknowledge interrupts:

D0	(0x0001) ADC FIFO Write
D1	(0x0002) Reset CGT
D2	(0x0004) reserved
D3	(0x0008) Pause CGT
D4	(0x0010) About counter out
D5	(0x0020) Delay counter out
D6	(0x0040) ADC Sample Counter
D7	(0x0080) DAC1 Update Counter
D8	(0x0100) DAC2 Update Counter
D9	(0x0200) User TC1 out
D10	(0x0400) User TC1 out, inverted
D11	(0x0800) User TC2 out
D12	(0x1000) Digital Interrupt
D13	(0x2000) External Interrupt
D14	(0x4000) External Trigger rising-edge

---

## ClearITOverrun4520

---

### Syntax

```
void ClearITOverrun4520 (RTDHANDLE hBoard);
```

### Description

Clear Interrupt Overrun Register.

The Interrupt Overrun Register is for detecting interrupt overrun conditions. If an interrupt occurs from a source and another interrupt occurs from that source before the first interrupt is acknowledged, the appropriate bit in the Interrupt Overrun Register goes to 1 indicating overrun.

**NOTE:** This function is automatically called on every interrupt if you install an interrupt handler.

### Parameters

hBoard: device handle

---

## CloseBoard4520

---

### Syntax

```
BOOL CloseBoard4520 (RTDHANDLE hBoard, LPSTR szBuf);
```

### Description

This routine is used to close a board.

Call this function when you finished to use a board. CloseBoard4520 releases no longer needed system resources and the PCI4520 itself and makes them available for other applications.

**NOTE:** After closing a board, no driver functions may be called but OpenBoard4520.

### Parameters

hBoard: device handle.

szBuf: Message buffer to return error message if any error occurs during closing the board.

### Return Value

TRUE Board is closed successfully.  
FALSE There was an error while closing the board.  
See GetErrorStatus4520 for the error occurred.

---

**DACClockFreeRun4520**

---

**Syntax**

```
void DACClockFreeRun4520(RTDHANDLE hBoard, BOOL runmode);
```

**Description**

DAC Clock Run Mode selection.

On power-up the board starts with free running DAC clock. For the start/stop mode which is used with the one-cycle generation program, the run mode should be set to start/stop mode.

NOTE: This function works only on DM7520 with EPLD version 11 or greater.

**Parameters**

hBoard:	device handle
runmode:	TRUE = free run mode (power-up default) FALSE = start/stop mode

---

**DACClockStartSelect4520**

---

**Syntax**

```
void DACClockStartSelect4520(RTDHANDLE hBoard, uint16 data);
```

**Description**

DAC Clock Start Select.

NOTE: This function works only on DM7520 with EPLD version 9 or greater.

**Parameters**

hBoard:	device handle
data :	0x0 = Software Pacer Start (RD_LAS0+028h) 0x1 = External Trigger 0x2 = Digital Interrupt 0x3 = User TC2 out 0x4 = SyncBus 0 0x5 = SyncBus 1 0x6 = SyncBus 2 0x7 = Software D/A Clock Start (RD_LAS0+0ch)

---

**DACClockStopSelect4520**

---

**Syntax**

```
void DACClockStopSelect4520(RTDHANDLE hBoard, uint16 data);
```

**Description**

DAC Clock Stop Select.

NOTE: This function works only on DM7520 with EPLD version 9 or greater.

### Parameters

hBoard: device handle  
data :  
0x0 = Software Pacer Stop (WR\_LAS0+028h)  
0x1 = External Trigger  
0x2 = Digital Interrupt  
0x3 = User TC2 out  
0x4 = SyncBus 0  
0x5 = SyncBus 1  
0x6 = SyncBus 2  
0x7 = Software D/A Clock Stop (WR\_LAS0+0ch)  
0x8 = D/A1 update counter  
0x9 = D/A2 update counter

---

## DeInstallDMA4520

---

### Syntax

```
BOOL DeInstallDMA4520(RTDHANDLE hBoard, int dma_channel );
```

### Description

This routine is used to remove the DMA handler.

### Parameters

hBoard: device handle  
dma\_channel: DMA channel number (0/1)

### Return Value

TRUE Board is closed successfully.  
FALSE There was an error while closing the board.  
See GetErrorStatus4520 for the error occurred.

---

**EnableAcntStop4520**

---

**Syntax**

```
void EnableAcntStop4520 (RTDHANDLE hBoard, uint16 enable);
```

**Description**

Enable/disable About Counter to stop when it counts down to zero. Default setting is enabled. The About Counter is a 16-bit down counter that stops or reloads the initial count value after counting down to zero. The About Counter, counting at the rate of the Pacer Clock, can be used to count samples. When About Counter stop enabled, the counter stops when it counts down to zero. If you disable About Counter Stop, the counter keeps reloading the initial counter value when it reaches zero. In both cases, you may program the board to generate an interrupt. When stop disabled, you can install your own interrupt handler for the About Counter countdown event and keep track of the turnovers. Use this mode to extend the counting capability highest number than 65535. In the interrupt handler, when the counter counted the desired number of samples, you have to reprogram the counter to the fractional number of samples and enable Sample Counter stop.

**Example**

Suppose you want to count 500,000 samples with the About Counter. In this case first you load the initial count of 65,535, disable About Counter stop and install an interrupt handler for About Counter Count-Down. In the interrupt handler, you maintain a counter for the turnovers. After  $500,000/65,535 = 7$ , you load the fractional count  $500,000 - 7*65,535 = 41,255$  and enable About Counter stop.

**Parameters**

hBoard:	device handle
enable:	0 = Stop enabled
	1 = Stop disabled

---

**EnableCGT4520**

---

**Syntax**

```
void EnableCGT4520 (RTDHANDLE hBoard, uint16 enable);
```

**Description**

Select Channel Gain Table or Channel-Gain Latch to use for A/D sampling.

**Parameters**

hBoard:	device handle
enable:	CSC_LATCH (0) Channel Gain Table disabled Channel Gain Latch enabled.
	CSC_CGT (1) Channel Gain Table enabled Channel Gain Latch disabled.

**See Also**

Channel-Gain Circuitry, EnableCGTDigital4520.

---

**EnableCGTDigital4520**

---

**Syntax**

```
void EnableCGTDigital4520 (RTDHANDLE hBoard, uint16 enable);
```

**Description**

Enable/disable Digital Table.

NOTE: This function has no effect if you use the Channel-Gain Latch.

**Parameters**

hBoard:	device handle
enable:	FALSE (0) Digital Table disabled, Digital I/O P1 port enabled.
	TRUE (1) Digital Table enabled, Digital I/O P1 port disabled.

**See Also**

Channel-Gain Circuitry, EnableCGT4520

---

**EnableCGTPause4520**

---

**Syntax**

```
void EnableCGTPause4520 (RTDHANDLE hBoard, uint16 enable);
```

**Description**

Enable/disable using the Channel-Gain Table Pause Bit.

NOTE: Set the Pause Bit if you want to stop an entry in the table and wait for the next trigger. In burst mode, the pause bit is ignored.

**Parameters**

hBoard:	device handle
enable:	FALSE (0)      Disable interpreting CGT Pause Bit.
	TRUE (1)        Enable interpreting CGT Pause Bit.

**See Also**

Channel-Gain Circuitry

---

**EnableDIO0Irq4520**

---

**Syntax**

```
void EnableDIO0Irq4520 (RTDHANDLE hBoard, BOOL enable);
```

## Description

Enable/disable Bit Programmable Digital I/O to generate interrupts.

## Parameters

hBoard:	device handle	
enable:	FALSE (0)	disable
	TRUE (1)	enable

## See Also

Advanced Digital Trigger

---

EnableSbus04520  
EnableSbus14520  
EnableSbus24520

---

## Syntax

```
void EnableSbus04520 (RTDHANDLE hBoard, uint16 enable);  
void EnableSbus14520 (RTDHANDLE hBoard, uint16 enable);  
void EnableSbus24520 (RTDHANDLE hBoard, uint16 enable);
```

## Description

Enable Synchron Bus line 0/1/2.

This function enables the programmed Synchron Bus source to appear on the line. Calling it with enable=FALSE has the effect of removing the signal from the line.

NOTE: A Synchron Bus line may be enabled on one board only.

## Parameters

hBoard:	device handle
enable:	0x0 = enable
	0x1 = disable

## See Also

Synchron Bus.



---

**Get8254Count4520**

---

**Syntax**

```
uint16 Get8254Count4520 (RTDHANDLE hBoard, int counter);
```

**Description**

Read current counter value of the 8254 Timer/Counter.

**Parameters**

hBoard: device handle  
counter: One of the 16-bit timer/counters of the board.  
Use TC\_XXXX for argument 'counter'.

**Return Value**

Current counter value of the 8254 Timer/Counter.

---

**Get8254Mode4520**

---

**Syntax**

```
uint8 Get8254Mode4520 (RTDHANDLE hBoard, int counter);
```

**Description**

Read back programmed operation mode of a 8254 Timer/Counter.

**Parameters**

hBoard: device handle  
counter: One of the 16-bit timer/counters of the board.  
Use TC\_XXXX for argument 'counter'.

**Return Value**

Current operation mode of the 8254 Timer/Counter.

---

**Get8254Status4520**

---

**Syntax**

```
uint8 Get8254Status4520 (RTDHANDLE hBoard, int counter);
```

**Description**

Read status of a 8254 Timer/Counter.

**Parameters**

hBoard: device handle  
counter: One of the 16-bit timer/counters of the board.

Use TC\_XXXX for argument 'counter'.

### **Return Value**

Current status of the 8254 Timer/Counter.

---

## **GetAcntCount4520**

---

### **Syntax**

```
uint16 GetAcntCount4520 ( RTDHANDLE hBoard );
```

### **Description**

Read the About Counter value.

### **Parameters**

hBoard:                device handle

### **Return Value**

The current about counter value.

---

## **GetAdcntCount4520**

---

### **Syntax**

```
uint16 GetAdcntCount4520 ( RTDHANDLE hBoard );
```

### **Description**

Read the A/D Sample Counter value.

### **Parameters**

hBoard:                device handle

### **Return Value**

The current A/D sample counter value.

---

## **GetDACDivisor4520**

---

### **Syntax**

```
uint32 GetDACDivisor4520 (RTDHANDLE hBoard);
```

### **Description**

Read the 16 bit wide DAC clock value. (PCI4520)

Read the 24 bit wide DAC clock value. (DM7520)

### **Parameters**

hBoard:                device handle

### **Return Value**

The actual DAC clock value.

---

## GetDIO0Clock4520

---

### Syntax

```
uint8 GetDIO0Clock4520 (RTDHANDLE hBoard);
```

### Description

Read Clock set for the bit programmable Digital I/O (port 0).

### Parameters

hBoard:                device handle

### Return Value

0                      - 8 MHz On-board Oscillator  
1                      - User TC Counter 1

---

## GetDIO0CompareMode4520

---

### Syntax

```
uint8 GetDIO0CompareMode4520 (RTDHANDLE hBoard);
```

### Description

Read port 0 compare mode.

### Parameters

hBoard:                device handle

### Return Value

0                      - Event Mode  
1                      - Match Mode

---

## GetDIO0CompareValue4520

---

### Syntax

```
uint8 GetDIO0CompareValue4520 (RTDHANDLE hBoard);
```

### Description

Read pattern of Port 0 I/O lines for Digital Interrupt generation.

### Parameters

hBoard:                device handle

### Return Value

pattern

---

## GetDIO0Direction4520

---

### Syntax

```
uint8 GetDIO0Direction4520 (RTDHANDLE hBoard);
```

### Description

Return programmed Digital I/O Port 0 line directions.

### Parameters

hBoard:                device handle

### Return Value

8-bit direction mask:

Bit = 0 – line is programmed for input

Bit = 1 – line is programmed for output

---

## GetDIO0Mask4520

---

### Syntax

```
uint8 GetDIO0Mask4520 (RTDHANDLE hBoard);
```

### Description

Return Digital Interrupt mask for Port 0 I/O lines.

### Parameters

hBoard:                device handle

### Return Value

8-bit interrupt mask:

Bit = 0 - line is disabled to take part in Digital Interrupt generation

Bit = 1 - line is enabled to take part in Digital Interrupt generation

### See Also

Advanced Digital Trigger

---

## GetDIO1Direction4520

---

### Syntax

```
uint8 GetDIO1Direction4520 (RTDHANDLE hBoard);
```

### Description

Return Digital I/O Port 1 direction.

### Parameters

hBoard:                device handle

### Return Value

Bit 0	port direction
	0     input
	1     output
Bit 1..7	0

---

## GetDIOStatus4520

---

### Syntax

```
uint8 GetDIOStatus4520 (RTDHANDLE hBoard);
```

### Description

Read Digital I/O status.

### Parameters

hBoard:                device handle

### Return Value

D0..D1	register opened for programming
D2	Port 1 direction (0 - input, 1 - output)
D3	Digital IRQ Mode (0 - Event, 1 - Match)
D4	Digital IRQ Enable (0 - disabled, 1 - enabled)
D5	Digital Sample Clock Select (0 - 8MHz oscillator, 1 - User TC Counter 1)
D6	Digital IRQ Status (0 - no IRQ, 1 - IRQ)
D7	Strobe Status

### See Also

Advanced Digital Trigger

---

## GetDMA0Status4520

## GetDMA1Status4520

---

### Syntax

```
uint8 GetDMA0Status4520 ( RTDHANDLE hBoard );
```

```
uint8 GetDMA1Status4520 ( RTDHANDLE hBoard );
```

### Description

Read DMA status register.

### Parameters

hBoard:                device handle

### Return Value

Value of status register (DMA channel 0/1 command/status register).

---

## GetDMADoneChannel4520

---

### Syntax

```
unsigned char GetDMADoneChannel4520(RTDHANDLE hBoard);
```

### Description

This routine is used to get which DMA channel generated the DMA done interrupt.  
(See the wiodma example program how to use this function.)

### Parameters

hBoard:                device handle

### Return Value

1 if DMA channel 0 done  
2 if DMA channel 1 done  
3 if DMA channel 0 and DMA channel 1 done

---

## GetErrorStatus4520

---

### Syntax

```
BOOL GetErrorStatus4520 (RTDHANDLE hBoard, LONG *ErrorCode);
```

### Description

Return last error code for a given board and clears the internal error status variable (i.e., a second call to this function will return FALSE, indicating no error).

### Parameters

hBoard: device handle  
ErrorCode: pointer to a location to return the error code.

### Return Value

TRUE If there is an error.  
FALSE If there is no error.

---

## GetFifoStatus4520

---

### Syntax

```
uint16 GetFifoStatus4520 (RTDHANDLE hBoard);
```

### Description

Return the status of all board FIFO buffers.

### Parameters

hBoard: device handle

### Return Value

Bit 0 (0x0001) DAC1 FIFO not empty  
Bit 1 (0x0002) DAC1 FIFO not half empty  
Bit 2 (0x0004) DAC1 FIFO not full  
Bit 3 (0x0008) 0 (reserved)  
Bit 4 (0x0010) DAC2 FIFO not empty  
Bit 5 (0x0020) DAC2 FIFO not half empty  
Bit 6 (0x0040) DAC2 FIFO not full  
Bit 7 (0x0080) 0 (reserved)  
Bit 8 (0x0100) ADC FIFO not empty  
Bit 9 (0x0200) ADC FIFO not half empty  
Bit 10 (0x0400) ADC FIFO not full  
Bit 11 (0x0800) 0 (reserved)  
Bit 12 (0x1000) DIN FIFO not empty  
Bit 13 (0x2000) DIN FIFO not half empty  
Bit 14 (0x4000) DIN FIFO not full  
Bit 15 (0x8000) 0 (reserved)

See FS\_XXXX symbolic constants.

---

## GetITStatus4520

---

### Syntax

```
uint16 GetITStatus4520 (RTDHANDLE hBoard);
```

### Description

Read Interrupt Status Register.

### Parameters

hBoard:                    device handle

### Return Value

Return a mask of interrupt sources which have a pending interrupt request (bit = 0-inactive, bit = 1-active):

Bit 0	(0x0001) ADC FIFO Write
Bit 1	(0x0002) Reset CGT
Bit 2	(0x0004) reserved
Bit 3	(0x0008) Pause CGT
Bit 4	(0x0010) ADC FIFO half-full
Bit 5	(0x0020) ADC FIFO full
Bit 6	(0x0040) ADC Sample Counter
Bit 7	(0x0080) DAC1 Update Counter
Bit 8	(0x0100) DAC2 Update Counter
Bit 9	(0x0200) User TC1 out
Bit 10	(0x0400) User TC1 out, inverted
Bit 11	(0x0800) User TC2 out
Bit 12	(0x1000) Digital Interrupt
Bit 13	(0x2000) External Interrupt
Bit 14	(0x4000) External Trigger rising-edge
Bit 15	(0x8000) External Trigger falling-edge

---

## GetTimerStatus4520

---

### Syntax

```
uint16 GetTimerStatus4520 (RTDHANDLE hBoard);
```

### Description

Read Timer Counters Status.

### Parameters

hBoard:                    device handle

### Return Value

Bit 0	Pacer Clock Gate status: 0 – clock is gated (not running) 1 – clock is running
Bit 1	Burst Clock Gate status: 0 – clock is gated (not running) 1 – clock is running
Bit 2	Pacer Clock Delayed Start Trigger status: 0 - delay over 1 - delay in progress

Bit 3        Pacer Clock About Trigger status:  
             0 – completed  
             1 – in progress  
Bit 4        Pacer Clock Shutdown Flag status:  
             0 - Pacer Clock can be start triggered only by Software Pacer Start Command  
             1 - Pacer Clock can be start triggered  
See TS\_XXXX symbolic constants.

---

## InitBoard4520

---

### Syntax

```
void InitBoard4520 (RTDHANDLE hBoard);
```

### Description

Resets the board to power up defaults and clears all FIFO buffers and the CGT.

NOTE: It is a good practice to call this function after opening a board for use.

### Parameters

hBoard: device handle

---

## InstallCallbackIRQHandler4520

---

### Syntax

```
void InstallCallbackIRQHandler4520 (RTDHANDLE hBoard,  
                                     isr_t handler);
```

### Description

This function installs interrupt handler function.

NOTE: The handler parameter is not a real interrupt handler. It is a callback function that is scheduled for execution by the interrupt service routine. Your handler function will run in a separate thread after the completion of the interrupt service routine.

### Parameters

hBoard: device handle  
isrT handler: user callback function

---

## InstallDMA4520

---

### Syntax

```
BOOL InstallDMA4520(RTDHANDLE hBoard, DMASetup4520 *dmasetup,  
                   BOOL fill, UCHAR fill_byte, int dma_channel );
```

### Description

This routine is used to install DMA handler. It must be called before any DMA operations.

NOTE: Call **DeInstallDMA4520** after finishing DMA operation.

### Parameters

hBoard: device handle  
dmasetup: structure to return parameters to caller

fill: TRUE if buffer fill is necessary  
fill\_byte: DMA buffer filler  
dma\_channel: DMA channel index (0/1)

**Return Value**

TRUE If there is no error.  
FALSE If there is an error.

---

**IsADFifoEmpty4520**

---

**Syntax**

```
BOOL IsADFifoEmpty4520 (RTDHANDLE hBoard);
```

**Description**

This routine checks to see if the A/D FIFO is empty.

**Parameters**

hBoard: device handle

**Return Value**

TRUE (non-zero) if the A/D FIFO is empty.

---

**IsADFifoFull4520**

---

**Syntax**

```
BOOL IsADFifoFull4520 (RTDHANDLE hBoard);
```

**Description**

This routine checks to see if the A/D FIFO is full.

**Parameters**

hBoard: device handle

**Return Value**

TRUE (non-zero) if the A/D FIFO is full.

---

**IsADFifoHalfFull4520**

---

**Syntax**

```
BOOL IsADFifoHalfFull4520 (RTDHANDLE hBoard);
```

**Description**

This routine checks to see if the A/D FIFO is half-full.

**Parameters**

hBoard: device handle

**Return Value**

TRUE (non-zero) if the A/D FIFO is half-full.

---

## IsBusMaster4520

---

### Syntax

```
BOOL IsBusMaster4520 (RTDHANDLE hBoard);
```

### Description

This routine checks the target/bus-master jumper setting on the board. The function is useful to detect the board mode and disable to run the DMA related functions.

### Parameters

hBoard:                    device handle

### Return Value

TRUE if the board is jumpered as bus-master device.  
FALSE if the board is jumpered as target device.

---

## IsDAC1FifoEmpty4520

---

### Syntax

```
BOOL IsDAC1FifoEmpty4520 (RTDHANDLE hBoard);
```

### Description

This routine checks to see if the DAC1 FIFO is empty.

### Parameters

hBoard:                    device handle

### Return Value

TRUE (non-zero) if the DAC1 FIFO is empty.

---

## IsDAC1FifoFull4520

---

### Syntax

```
BOOL IsDAC1FifoFull4520 (RTDHANDLE hBoard);
```

### Description

This routine checks to see if the DAC1 FIFO is full.

### Parameters

hBoard:                    device handle

### Return Value

TRUE (non-zero) if the DAC1 FIFO is full.

---

## IsDAC1FifoHalfFull4520

---

### Syntax

```
BOOL IsDAC1FifoHalfFull4520 (RTDHANDLE hBoard);
```

**Description**

This routine checks to see if the DAC1 FIFO is half full.

**Parameters**

hBoard:                device handle

**Return Value**

TRUE (non-zero) if the DAC1 FIFO is half full.

---

**IsDAC2FifoEmpty4520**

---

**Syntax**

```
BOOL IsDAC2FifoEmpty4520 (RTDHANDLE hBoard);
```

**Description**

This routine checks to see if the DAC2 FIFO is empty.

**Parameters**

hBoard:                device handle

**Return Value**

TRUE (non-zero) if the DAC2 FIFO is empty.

---

**IsDAC2FifoFull4520**

---

**Syntax**

```
BOOL IsDAC2FifoFull4520 (RTDHANDLE hBoard);
```

**Description**

This routine checks to see if the DAC2 FIFO is full.

**Parameters**

hBoard:                device handle

**Return Value**

TRUE (non-zero) if the DAC2 FIFO is full.

---

**IsDAC2FifoHalfFull4520**

---

**Syntax**

```
BOOL IsDAC2FifoHalfFull4520 (RTDHANDLE hBoard);
```

**Description**

This routine checks to see if the DAC2 FIFO is half full.

**Parameters**

hBoard:                device handle

**Return Value**

TRUE (non-zero) if the DAC2 FIFO is half full.

---

## IsDINFifoEmpty4520

---

### Syntax

```
BOOL IsDINFifoEmpty4520 (RTDHANDLE hBoard);
```

### Description

This routine checks to see if the High Speed Digital Input FIFO is empty.

### Parameters

hBoard:                device handle

### Return Value

TRUE (non-zero) if the High Speed Digital Input FIFO is empty.

---

## IsDINFifoFull4520

---

### Syntax

```
BOOL IsDINFifoFull4520 (RTDHANDLE hBoard);
```

### Description

This routine checks to see if the High Speed Digital Input FIFO is full.

### Parameters

hBoard:                device handle

### Return Value

TRUE (non-zero) if the High Speed Digital Input FIFO is full.

---

## IsDINFifoHalfFull4520

---

### Syntax

```
BOOL IsDINFifoHalfFull4520 (RTDHANDLE hBoard);
```

### Description

This routine checks to see if the High Speed Digital Input FIFO is half full.

### Parameters

hBoard:                device handle

### Return Value

TRUE (non-zero) if the High Speed Digital Input FIFO is half full.

---

## IsDMA0Done4520

## IsDMA1Done4520

---

### Syntax

```
BOOL IsDMA0Done4520 ( RTDHANDLE hBoard );
```

```
BOOL IsDMA1Done4520 ( RTDHANDLE hBoard );
```

**Description**

Read DMA done status bit and return TRUE if done.

**Parameters**

hBoard:                    device handle

**Return Value**

TRUE if DMA done bit is 1 (the DMA data transfer is complete).  
FALSE if DMA data transfer is not complete.

---

**LoadAcnt4520**

---

**Syntax**

```
void LoadAcnt4520 (RTDHANDLE hBoard, uint16 divisor);
```

**Description**

Load A/D About Counter divisor.

**Parameters**

hBoard:                device handle  
divisor:                divisor value

---

**LoadAdcnt4520**

---

**Syntax**

```
void LoadAdcnt4520 (RTDHANDLE hBoard);
```

**Description**

Load A/D Sample Counter divisor.

After setting the A/D Sample Counter divisor, call this function to properly load the divisor.

NOTE:                The output of the A/D sample counter can be an interrupt source, so at the end of the count-down process an interrupt may be generated.

**Parameters**

hBoard:                device handle

---

**LoadDAC14520****LoadDAC24520**

---

**Syntax**

```
void LoadDAC14520 (RTDHANDLE hBoard, int Data);
```

```
void LoadDAC24520 (RTDHANDLE hBoard, int Data);
```

**Description**

Write a sample (12-bit) to D/A1 or D/A2 FIFO.

NOTE:                This function does not update the D/A. Depending on the configuration (see SetupDAC4520) of the D/A channel, the appropriate event must occur to cause an update.

**Parameters**

hBoard:                device handle  
Data:                    12-bit data to write

---

---

## LoadDcnt4520

---

### Syntax

```
void LoadDcnt4520 (RTDHANDLE hBoard);
```

### Description

Load Delay Counter divisor.

After setting the Delay Counter divisor, call this function to properly load the divisor.

### Parameters

hBoard:                device handle

---

## LoadUcnt14520

## LoadUcnt24520

---

### Syntax

```
void LoadUcnt14520 (RTDHANDLE hBoard);
```

```
void LoadUcnt24520 (RTDHANDLE hBoard);
```

### Description

Load D/A1 or D/A2 Update Counter divisor.

After setting the Update Counter divisor, call this function to properly load the divisor.

NOTE:                The output of the Update Counter can be an interrupt source, so at the end of the count-down process an interrupt may be generated.

### Parameters

hBoard:                device handle

---

**McBSPADControl4520**

---

**Syntax**

```
void McBSPADControl4520 ( RTDHANDLE hBoard, BOOL enable );
```

**Description**

This function enables or disables the sending of the A/D converted samples to the serial DSP port (McBSP).

**Parameters**

hBoard:                    device handle  
enable:                    Boolean value to enable or disable the serial port

---

**McBSPDAControl4520**

---

**Syntax**

```
void McBSPDAControl4520 ( RTDHANDLE hBoard, BOOL enable );
```

**Description**

This function enables or disables the receiving of the serial DSP port (McBSP) samples to the D/A converter of the PCI4520/DM7520 board. The D/A channel can be selected with the Digital Input Data Marker bit 0.

**Parameters**

hBoard:                    device handle  
enable:                    Boolean value to enable or disable the serial port

---

## OpenBoard4520

---

### Syntax

```
int OpenBoard4520    (int DeviceId, int Number,  
                    BoardConfig4520 boardconfig);
```

### Description

This routine is used to open the board with ordinal number 'Number' in the system. This function must be called before any API function call. At the end of the application program, the board must be closed with the CloseBoard4520 function.

### Parameters

DeviceId:	The PCI device ID of the board in hexadecimal form. It is 0x4520 in case of PCI4520 and 0x7520 in case of DM7520.
Number:	The ordinal number of board to put in operation. There may be more than one board in the system, they are numbered from 1.
boardconfig:	Pointer to a BoardConfig4520 structure to return configuration information about the board. Use this argument if you need to know the Operating System resources allocated to the board. Pass NULL for this argument if you do not need this information.

### Return Value

0	if no error occurred
ERROR_XX	otherwise

---

**PollDIO04520**

---

**Syntax**

```
void PollDIO04520 ( RTDHANDLE hBoard, uint8 *direction,  
                  uint8 *mask, uint8 *mode, uint8 *compare,  
                  uint8 *clock, uint8 *irq,  
                  uint8 *itstatus );
```

**Description**

Return Digital I/O Port 0 status.

**Parameters**

hBoard:	device handle
direction:	Returns the programmed direction mask for the 8 I/O lines (see GetDIO0Direction4520): Bit = 0 – line is programmed for input. Bit = 1 – line is programmed for output.
mask:	Returns the Digital Interrupt mask for Port 0 I/O lines (see GetDIO0Mask4520): Bit = 0 - line is disabled to take part in Digital Interrupt generation. Bit = 1 - line is enabled to take part in Digital Interrupt generation.
mode:	Returns Advanced Digital Trigger mode (see GetDIO0CompareMode4520): 0 – Event mode 1 – Mask mode
compare:	Returns the programmed value of the Compare Register (see GetDIO0CompareValue4520).
clock:	Returns the programmed Advanced Digital Trigger clock (see GetDIO0Clock4520): 0 – 8 MHz On-board Oscillator 1 – User TC Counter 1
irq:	Returns the Digital I/O Port 0 IRQ Status (see GetDIO0Status4520): 0 - no IRQ 1 - IRQ
itstatus:	(return value)



---

**ReadADDData4520**

---

**Syntax**

```
int ReadADDData4520 (RTDHANDLE hBoard);
```

**Description**

Read sample (12-bit) from the A/D FIFO.

**Parameters**

hBoard: device handle

**Return Value**

Signed 12 bit A/D Data.

**See Also**

ReadADDDataWithMarker4520

---

**ReadADDDataWithMarker4520**

---

**Syntax**

```
int16 ReadADDDataWithMarker4520 (RTDHANDLE hBoard);
```

**Description**

Read A/D FIFO data and data marker bits.

**Parameters**

hBoard: device handle

**Return Value**

Signed 12 bit AD Data and 3 bit Data Marker.

**See Also**

ReadADDData4520

---

**ReadDinFifo4520**

---

**Syntax**

```
uint16 ReadDinFifo4520 (RTDHANDLE hBoard);
```

**Description**

Read a block of samples from the High Speed Digital Input FIFO.

NOTE: You must ensure that the FIFO contains at least *nSamples* samples.

**Parameters**

hBoard: device handle

**Return Value**

16 bit data from FIFO (the upper byte is undefined).

**See Also**

ReadDinFifoBlock4520

---

**ReadDIO04520**

---

**Syntax**

```
uint8 ReadDIO14520 (RTDHANDLE hBoard);
```

**Description**

Read Digital I/O Port 0 digital input lines.

NOTE: For lines programmed for output, the current value of the line is returned.

**Parameters**

hBoard: device handle

**Return Value**

8-bit port value (bit mask).

---

**ReadDIO14520**

---

**Syntax**

```
uint8 ReadDIO14520 (RTDHANDLE hBoard);
```

**Description**

Read Port 1 digital input lines.

**Parameters**

hBoard: device handle

**Return Value**

8-bit port value.

---

**ReadFirmwareVersion4520**

---

**Syntax**

```
BYTE ReadFirmwareVersion4520 (RTDHANDLE hBoard);
```

**Description**

This routine reads the 4-bit firmware version code.

**Parameters**

hBoard: device handle

### Return Value

The firmware version of the board. The lower 4 bits are containing the information.

---

## ReadITOverrun4520

---

### Syntax

```
uint16 ReadITOverrun4520 (RTDHANDLE hBoard);
```

### Description

Read the Interrupt Overrun Register.

### Parameters

hBoard:                    device handle

### Return Value

Returns a mask of IT sources which have been overrun.

Bit 0	(0x0001) ADC FIFO Write
Bit 1	(0x0002) Reset CGT
Bit 2	(0x0004) reserved
Bit 3	(0x0008) Pause CGT
Bit 4	(0x0010) About counter out
Bit 5	(0x0020) Delay counter out
Bit 6	(0x0040) ADC Sample Counter
Bit 7	(0x0080) DAC1 Update Counter
Bit 8	(0x0100) DAC2 Update Counter
Bit 9	(0x0200) User TC1 out
Bit 10	(0x0400) User TC1 out, inverted
Bit 11	(0x0800) User TC2 out
Bit 12	(0x1000) Digital Interrupt
Bit 13	(0x2000) External Interrupt
Bit 14	(0x4000) External Trigger rising-edge
Bit 15	(0x8000) External Trigger falling-edge

### See Also

ClearITOverrun4520

---

## ReadPacer4520

---

### Syntax

```
void ReadPacer4520 (RTDHANDLE hBoard);
```

### Description

Reads the actual Pacer Clock counter value.

### Parameters

hBoard:                    device handle

### Return Value

32 bit value.

---

## ReadSampleCounter4520

---

### Syntax

```
void Read SampleCounter4520 (RTDHANDLE hBoard,  
                             int counter);
```

### Description

Read one of the 3 sample counters.

### Parameters

hBoard: device handle  
counter Use TC\_XXXX for argument 'counter'.

---

## ReadTimerCounter4520

---

### Syntax

```
void ReadTimerCounter4520 (RTDHANDLE hBoard, int counter,  
                           uint8 *mode, uint16 *count );
```

### Description

Read back programmed operation mode and current counter value of a 8254 Timer/Counter.

### Parameters

hBoard: device handle  
counter: One of the 16-bit timer/counters of the board.:

TC_PCLK16	Pacer Clock 0
TC_PCLK32	Pacer Clock 1
TC_BCLK	Burst Clock
TC_ADC_SCNT	A/D Sample counter
TC_DAC1_UCNT	D/A1 Update counter
TC_DAC2_UCNT	D/A2 Update counter
TC_DCNT	Delay Counter
TC_ACNT	About Counter
TC_DAC_CLK	DAC Clock
TC.UTC0	User TC 0
TC.UTC1	User TC 1
TC.UTC2	User TC 2

mode: Pointer to location to return counter mode.  
count: Pointer to location to return current count  
NOTE: This is not the initial count (or divisor) of the counter. There is no way to read initial count from a 8254 Timer/Counter.

---

## ReadUserInput4520

---

### Syntax

```
uint16 ReadUserInput4520 (RTDHANDLE hBoard);
```

### Description

Read User Input Line 0 & 1.

### Parameters

hBoard:                device handle

### Return Value

Bit 0                User input 0 state.

Bit 1                User input 1 state.

All the other bits are zeros.

---

## RemoveIRQHandler4520

---

### Syntax

```
void RemoveIRQHandler4520 (RTDHANDLE hBoard);
```

### Description

This routine is used to uninstall the IRQ handler function.

NOTE:                As a side effect, this function disables all interrupt sources enabled when the interrupt handler was installed.

### Parameters

hBoard:                device handle

---

## ResetCGT4520

---

### Syntax

```
void ResetCGT4520 (RTDHANDLE hBoard);
```

### Description

Reset the A/D Channel Gain Table pointer to the beginning of the table.

After calling this function, the next channel sampled is that of specified by the first entry in the Channel-Gain Table.

NOTE:                Calling this function will cause a Channel-Gain Table Reset event. Consider this if you have a sample counter (A/D Sample Counter or the About Counter) configured to count these events or you enabled this event to be an interrupt source.

### Parameters

hBoard:                device handle

## See Also

Channel-Gain Circuitry

---

## ResetDAC1Fifo4520 ResetDAC2Fifo4520

---

### Syntax

```
void ResetDAC1Fifo4520 (RTDHANDLE hBoard);  
void ResetDAC2Fifo4520 (RTDHANDLE hBoard);
```

### Description

Reset the DAC1/DAC2 update pointer to the beginning of the FIFO.

After calling this function, the next value output on the D/A converter is the first value in the FIFO, i.e., the one, which was entered first in the FIFO.

### Parameters

hBoard:                    device handle

---

## ResetDMA0State4520 ResetDMA1State4520

---

### Syntax

```
void ResetDMA0State4520 ( RTDHANDLE hBoard );  
void ResetDMA1State4520 ( RTDHANDLE hBoard );
```

### Description

Reset DMA state machine (enable next cycle).

The function is used in demand mode DMA operation.

### Parameters

hBoard:                    device handle

---

## SelectDIO0Register4520

---

### Syntax

```
void SelectDIO0Register4520 ( RTDHANDLE hBoard, int src );
```

### Description

Program digital control register.

### Parameters

hBoard: device handle

aReg: 0 - Clear Command  
1 - Direction Register  
2 - Mask Register  
3 - Compare Register

---

## SelectBurstClockPrimaryClock4520

---

### Syntax

```
void SelectBurstClockPrimaryClock4520 ( RTDHANDLE hBoard,  
uint16 src );
```

### Description

Select burst clock primary clock frequency.

NOTE: This function works only on DM7520 with EPLD version 10 or greater.

### Parameters

hBoard: device handle

src: 0 - 8 MHz clock selected  
1 - 20 MHz clock selected

---

## SelectDACClockPrimaryClock4520

---

### Syntax

```
void SelectDACClockPrimaryClock4520 ( RTDHANDLE hBoard, uint16  
src );
```

### Description

Select DAC clock primary clock frequency.

NOTE: This function works only on DM7520 with EPLD version 10 or greater.

## Parameters

hBoard: device handle  
src: 0 - 8 MHz clock selected  
1 - 20 MHz clock selected

---

## SelectPacerClockPrimaryClock4520

---

### Syntax

```
void SelectPacerClockPrimaryClock4520 ( RTDHANDLE hBoard,  
uint16 src );
```

### Description

Select pacer clock primary clock frequency.

NOTE: This function works only on DM7520 with EPLD version 10 or greater.

## Parameters

hBoard: device handle  
src: 0 - 8 MHz clock selected  
1 - 20 MHz clock selected

---

## Set8254Divisor4520

---

### Syntax

```
void Set8254Divisor4520 (RTDHANDLE hBoard, int counter,  
uint16 divisor);
```

### Description

Load an initial count in a Timer/Counter .

## Parameters

hBoard: device handle  
counter: One of the 16-bit timer/counters of the board.  
Use TC\_XXXX for argument 'counter'.  
divisor: 16-bit divisor to divide clock input frequency.

## See Also

Timer/Counters

---

## Set8254Mode4520

---

### Syntax

```
void Set8254Mode4520 (RTDHANDLE hBoard, int counter,
```

```
uint8 mode);
```

### Description

Load count in Timer/Counter 0 (8-bit).

### Parameters

hBoard:	device handle
counter:	One of the 16-bit timer/counters of the board. Use TC_XXXX for argument ' <i>counter</i> '. TC_PCLK16           Pacer Clock 0 TC_PCLK32           Pacer Clock 1 TC_BCLK             Burst Clock TC_ADC_SCNT         A/D Sample counter TC_DAC1_UCNT        D/A1 Update counter TC_DAC2_UCNT        D/A2 Update counter TC_DCNT             Delay Counter TC_ACNT             About Counter TC_DAC_CLK          D/A Clock TC_UTC0             User TC 0 TC_UTC1             User TC 1 TC_UTC2             User TC 2
mode	One of the 6 8254 operation modes. Use M8254_XXXX for argument ' <i>mode</i> ': M8254_EVENT_COUNTER   Event Counter M8254_HW_ONE_SHOT     Hardware-Retriggerable One-Shot M8254_RATE_GENERATOR   Rate Generator M8254_SQUARE_WAVE     Square Wave Mode M8254_SW_STROBE       Software Triggered Strobe M8254_HW_STROBE       Hardware Triggered Strobe (Retriggerable)

---

## SetAdcntSource4520

---

### Syntax

```
void SetAdcntSource4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

Select A/D Sample Counter source.

### Parameters

hBoard:	device handle
src:	Use ADC_SCNT_XXXX for argument ' <i>src</i> ': ADC_SCNT_CGT_RESET (0) Count Channel Gain Table resets. Counter counts down by one when the Channel-Gain table turns over to the first table entry.

## ADC\_SCNT\_FIFO\_WRITE (1)

Count A/D FIFO writes.

Counter counts down by one when a new sample is entered in the FIFO.

---

## SetBurstRate4520

---

### Syntax

```
void SetBurstRate4520 ( RTDHANDLE hBoard, float rate,  
int clock );
```

### Description

Program the burst clock rate. This is a deprecated function for legacy compatibility. Use the SetBurstRateF4520 function.

### Parameters

hBoard: device handle  
rate: Frequency value (Hertz) between 122Hz and 1,25MHz.  
clock: Burst Clock Primary Clock Frequency select  
0 = 8 MHz  
1 = 20 MHz  
This argument is ignored if the EPLD version is less than 10.

---

## SetBurstRateF4520

---

### Syntax

```
float SetBurstRateF4520 ( RTDHANDLE hBoard, float rate, int  
clock );
```

### Description

Program the burst clock rate. This function sets the rate by calculating a divisor of the Primary Clock (that equals either 8 MHz or 20 MHz). Since this divisor is rounded to an integer value, the actual burst clock frequency can be different from 'rate' parameter.

### Parameters

hBoard: device handle  
rate: Frequency value (Hertz) between 122Hz and 1,25MHz.  
clock: Burst Clock Primary Clock Frequency select  
0 = 8 MHz  
1 = 20 MHz  
This argument is ignored if the EPLD version is less than 10.

### Return Value

The actual Burst Clock rate.

---

## SetBurstStart4520

---

### Syntax

```
void SetBurstStart4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

Select Burst Clock start trigger.

NOTE: There is no programmable stop trigger for the Burst Clock. The Burst Clock automatically stops when the Channel-Gain Table resets to its first entry.

### Parameters

hBoard: device handle

src: Use BCLK\_START\_XXXX for argument 'src'.  
BCLK\_START\_SOFTWARE (0x0)  
Calling the StartBurst4520 function starts Burst Clock.  
BCLK\_START\_PCLK (0x1)  
Burst Clock is started on every Pacer Clock tick.  
BCLK\_START\_ETRIG (0x2)  
Burst Clock is started by the External Trigger line.  
BCLK\_START\_DIGITAL\_IT (0x3)  
Burst Clock is started when a Digital Interrupt occurs.  
BCLK\_START\_SBUS0 (0x4)  
BCLK\_START\_SBUS1 (0x5)  
BCLK\_START\_SBUS2 (0x6)  
Burst Clock is started by a rising edge on Synchron Bus line 0/1/2.

### See Also

Channel-Gain Circuitry

---

## SetChannelGain4520

---

### Syntax

```
void SetChannelGain4520 (RTDHANDLE hBoard, uint16 Channel, uint16 Gain, uint16 Range, uint16 Se_Diff, uint16 NrSe );
```

### Description

This routine loads the channel/gain latch.

### Parameters

hBoard: device handle

Channel: One of the 16 Analog Input channels (0 – AIN1, 1 – AIN2, etc.).

Gain: Gain value to apply on the Analog Input channel.  
Valid gains are 1, 2, 4, 8, 16, 32, 64, 128.  
(The 64x and 128x gain are not available on the DM7520 board.)

Range: Analog Input range.  
Valid values are:  
0 to +5 Volts  
0 to +10 Volts  
-5 to +5 Volts  
-10 to +10 Volts

Se\_Diff: Analog Input channel type:  
Single-Ended  
Differential

NrSe: Referencing mode:  
Analog ground referenced  
Non-ground referenced

---

## SetConversionSelect4520

---

### Syntax

```
void SetConversionSelect4520 (RTDHANDLE hBoard, uint16 Select);
```

### Description

Select A/D Conversion start signal.

### Parameters

hBoard: device handle

Select: ADC\_START\_SOFTWARE (0x0)  
One A/D conversion is started by a call to StartConversion4520.

ADC\_START\_PCLK (0x1)  
A/D conversions are started by the Pacer Clock.

ADC\_START\_BCLK (0x2)  
A/D conversions are started by the Burst Clock.

ADC\_START\_DIGITAL\_IT (0x3)  
One A/D conversion is started when a Digital Interrupt occurs.  
NOTE: Remember to acknowledge the Digital Interrupt by calling ClearDIO0IrqStatus4520. Otherwise, no newer Digital Interrupt is generated.

ADC\_START\_DAC1\_MARKER1 (0x4)

ADC\_START\_DAC2\_MARKER1 (0x5)  
An A/D conversion is started when D/A 1/2 Data Marker 1 becomes one. This trigger is for simultaneous A/D and with D/A conversions.

ADC\_START\_SBUS0 (0x6)

ADC\_START\_SBUS1 (0x7)

ADC\_START\_SBUS2 (0x8)  
The A/D converter is controlled by the signal on Synchron Bus line 0/1/2. Conversion started on a rising edge.

---

**SetDAC1Cycle4520**  
**SetDAC2Cycle4520**

---

**Syntax**

```
void SetDAC1Cycle4520 (RTDHANDLE hBoard, uint16 cycle);  
void SetDAC2Cycle4520 (RTDHANDLE hBoard, uint16 cycle);
```

**Description**

Program D/A1 or D/A2 Cycle Mode.

By setting cycle mode, D/A will continuously output data from the D/A FIFO. When the Update Pointer reaches the end of the FIFO, it resets to the beginning of the FIFO.

This mode can be used for generating periodic signals (i.e., waveform generation) without any processor intervention.

No cycle mode is the normal (default) mode of operation.

**Parameters**

hBoard:	device handle
cycle:	0x0 = no cycle
	0x1 = cycle

---

**SetDAC1Range4520**  
**SetDAC2Range4520**

---

**Syntax**

```
void SetDAC1Range4520 (RTDHANDLE hBoard, uint16 rng);  
void SetDAC2Range4520 (RTDHANDLE hBoard, uint16 rng);
```

**Description**

Program D/A 1 or D/A 2 output range.

**Parameters**

hBoard:	device handle
rng:	0x0 = unipolar 0..+5V
	0x1 = unipolar 0..+10V
	0x2 = bipolar 5V
	0x3 = bipolar 10V

---

**SetDAC1Start4520**  
**SetDAC2Start4520**

---

**Syntax**

```
void SetDAC1Start4520 (RTDHANDLE hBoard, uint16 src);  
void SetDAC2Start4520 (RTDHANDLE hBoard, uint16 src);
```

**Description**

Select D/A 1 or D/A 2 update signal.

## Parameters

hBoard:	device handle
src:	DAC_START_SOFTWARE (0) DAC1/DAC2 Update initiated by a call to UpdateDAC14520/UpdateDAC24520 or UpdateDACAll4520.
	DAC_START_CGT (1) = CGT controlled DAC1/DAC2 Update If the D12 (D13 for D/A2) bit of a Channel-Gain Table entry is set, D/A1 (D/A2) is updated simultaneously with the sampling of the Analog Input channel defined by the entry.
	DAC_START_DAC_CLK (2) D/A updates are triggered by the dedicated D/A Clock.
	DAC_START_EPCLK (3) D/A updates are triggered by the External Pacer Clock. External Clock must produce 100ns pulses, at least. Update occurs on the rising edge. NOTE: External Pacer Clock can be used independently from the Pacer Clock if Pacer Clock source is programmed to Internal. External Pacer Clock is only a pin on the I/O connector that you may use to for input to the Pacer Clock circuitry.
	DAC_START_SBUS0 (4)
	DAC_START_SBUS1 (5)
	DAC_START_SBUS2 (6) D/A update started on a rising edge on Synchron Bus line 0/1/2.

---

## SetDACDivisor4520

---

### Syntax

```
void SetDACDivisor4520 (RTDHANDLE hBoard, uint32 divisor);
```

### Description

Load the 16 bit wide DAC clock divisor value (PCI4520).

Load the 24 bit wide DAC clock divisor value (DM7520).

The 8 MHz input clock frequency is divided with this value generating the output signal.

### Parameters

hBoard:	device handle
divisor:	divisor value

---

## SetDACRate4520

---

### Syntax

```
void SetDACRate4520 (RTDHANDLE hBoard, float rate,
```

```
int clock);
```

### Description

Setup the D/A clock rate. If the argument clock = 0, the 8 MHz primary clock will be selected for the DAC clock, is clock = 1 the 20 MHz primary clock selected.

The function works with EPLD version 10 or greater.

### Parameters

hBoard: device handle  
rate: clock rate to setup  
clock: DAC Clock Primary Clock Frequency select  
0 = 8 MHz  
1 = 20 MHz  
This argument is ignored if the EPLD version is less than 10.

---

## SetDACRate4520

---

### Syntax

```
float SetDACRate4520 (RTDHANDLE hBoard, float rate,  
int clock);
```

### Description

Setup the D/A clock rate. If the argument clock = 0, the 8 MHz primary clock will be selected for the DAC clock, is clock = 1 the 20 MHz primary clock selected.

The function works with EPLD version 10 or greater.

### Parameters

hBoard: device handle  
rate: clock rate to setup  
clock: DAC Clock Primary Clock Frequency select  
0 = 8 MHz  
1 = 20 MHz  
This argument is ignored if the EPLD version is less than 10.

### Return Value

The actual DAC Clock rate.

---

## SetDIO0Clock4520

---

### Syntax

```
void SetDIO0Clock4520 (RTDHANDLE hBoard, uint8 src);
```

### Description

Select clock for the Advanced Digital Trigger.

### Parameters

hBoard: device handle

src:                   0 - 8 MHz On-board Oscillator  
                      1 - User TC Counter 1

**See Also**

Advanced Digital Trigger

---

## SetDIO0CompareMode4520

---

**Syntax**

```
void SetDIO0CompareMode4520 (RTDHANDLE hBoard, uint8 mode);
```

**Description**

Programs port 0 compare mode. This register is used for the Advanced Digital Interrupt modes.

**Parameters**

hBoard:               device handle  
mode:                 0 - Event Mode  
                      1 - Match Mode

**See Also**

Advanced Digital Trigger

---

## SetDIO0CompareValue4520

---

**Syntax**

```
void SetDIO0CompareValue4520 (RTDHANDLE hBoard, uint8 compare);
```

**Description**

Program Port 0 compare value.  
Pattern of Port 0 I/O lines for Digital Interrupt generation.

**Parameters**

hBoard:               device handle  
compare:              compare value

**See Also**

Advanced Digital Trigger

---

## SetDIO0Direction4520

---

**Syntax**

```
void SetDIO0Direction4520 (RTDHANDLE hBoard, uint8 direction);
```

**Description**

Program Port 0 direction (bit programmable).

**Parameters**

hBoard:               device handle

direction:            bit=0 - input  
                         bit=1 - output

---

## SetDIO0Mask4520

---

### Syntax

```
void SetDIO0Mask4520 (RTDHANDLE hBoard, uint8 mask);
```

### Description

Program Digital Interrupt mask for Port 0 I/O lines.

### Parameters

hBoard:            device handle  
mask:              bit mask to enable/disable I/O lines taking part in Digital Interrupt generation.

### See Also

Advanced Digital Trigger

---

## SetDIO1Direction4520

---

### Syntax

```
void SetDIO1Direction4520 (RTDHANDLE hBoard, uint8 direction);
```

### Description

Program Port 1 direction (byte programmable).

### Parameters

hBoard:            device handle  
direction:         0 - input  
                         1 - output

---

## SetEintPolarity4520

---

### Syntax

```
void SetEintPolarity4520 (RTDHANDLE hBoard, uint16 polarity);
```

### Description

Select External Interrupt polarity.

### Parameters

hBoard:            device handle  
polarity:         0x0 = Interrupts are generated on the positive edge of the External Interrupt line.  
                         0x1 = Interrupts are generated on the negative edge of the External Interrupt line.

---

## SetEtrgPolarity4520

---

### Syntax

```
void SetEtrgPolarity4520 (RTDHANDLE hBoard, uint16 polarity);
```

### Description

Select External Trigger polarity.

NOTE: External Trigger polarity also defines the behavior of the Pacer Clock in Gated mode.

### Parameters

hBoard: device handle  
polarity: 0x0 = External Trigger occurs on the positive edge of the External Trigger line.  
          0x1 = External Trigger occurs on the negative edge of the External Trigger line.

---

## SetFirstDescriptorBlock4520

---

### Syntax

```
BOOL SetFirstDescriptorBlock4520(RTDHANDLE hBoard, int  
dma_channel, uint32 dma_local, uint32 direction, uint32 offset,  
uint32 length);
```

### Description

Setup the first chained mode descriptor block. The first descriptor block starts at the beginning of the allocated DMA buffer in case of channel 0, and at the address of the allocated DMA buffer+0x100. The length of one descriptor block element is 16 bytes, the maximal length of the chaining table is 0x100 bytes for both channel.

### Parameters

hBoard: device handle  
dma\_channel: DMA channel index (0/1)  
dma\_local: local (onboard) DMA source/destination  
direction: transfer direction  
offset: the offset from the DMA buffer+0x200 address.  
          the DMA transfer starts from/to here  
length: DMA transfer length in bytes

### Return Value

TRUE If there is no error.  
FALSE If there is an error.

---

## SetFirstDirectDescriptorBlock4520

---

### Syntax

```
BOOL SetFirstDirectDescriptorBlock4520(RTDHANDLE hBoard, int  
dma_channel, uint32 dma_local, uint32 direction, uint32  
direct_address, uint32 length);
```

### Description

Setup the first chained mode descriptor block with direct physical target address.

Starting a DMA cycle to improper address may hangup your system!

### Parameters

hBoard:	device handle
dma_channel:	DMA channel index (0/1)
dma_local:	local DMA source/destination address
direction:	direction
direct_address:	physical DMA address
length:	DMA transfer byte count

### Return Value

TRUE	If there is no error.
FALSE	If there is an error.

---

## SetHdinStart4520

---

### Syntax

```
void SetHdinStart4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

Select sampling signal for High Speed Digital Input.

### Parameters

hBoard:	device handle
src:	0x0 = Software Trigger Port is sampled when StartHDin4520 is called. 0x1 = A/D Conversion Signal Port is sampled in synchrony with the Analog Input. 0x2 = User TC out 0 0x3 = User TC out 1 0x4 = User TC out 2 Port is sampled when User T/C 0/1/2 counts down to zero.

0x5 = External Pacer Clock

Port is sampled by the External Pacer Clock.

0x6 = External Trigger

Port is sampled when an External Trigger occurs.

Use HDIN\_XXXX for argument 'src'.

---

## SetITMask4520

---

### Syntax

```
void SetITMask4520 (RTDHANDLE hBoard, uint16 mask);
```

### Description

Write Interrupt Enable Mask Register.

### Parameters

hBoard: device handle

mask: IT source bit mask.

(Bit = 0 - source is disabled, bit = 1 – source is enabled)

IRQM\_ADC\_FIFO\_WRITE (0x0001)

Enable generating interrupt when sample entered to the A/D FIFO.

IRQM\_CGT\_RESET (0x0002)

Enable generating interrupt when Channel-Gain Table execution resets to the beginning of the table.

IRQM\_CGT\_PAUSE (0x0008)

Enable generating interrupt when a Channel-Gain Table entry has its Pause Bit set and CGT Pause is enabled (see EnableCGTPause4520).

IRQM\_ADC\_ABOUT\_CNT (0x0010)

Enable generating interrupt when About Counter counts down to zero and About Counter Stop is enabled (see EnableAcntStop4520).

IRQM\_ADC\_DELAY\_CNT (0x0020)

Enable generating interrupt when Delay Counter counts down to zero.

IRQM\_ADC\_SCNT (0x0040)

Enable generating interrupt when A/D Sample Counter counts down to zero.

IRQM\_DAC1\_UCNT (0x0080)

IRQM\_DAC2\_UCNT (0x0100)

Enable generating interrupt when D/A1 / D/A2 Update Counter counts down to zero.

IRQM\_UTC1 (0x0200)

Enable generating interrupt on User TC1 ticks.

IRQM\_UTC1\_INV (0x0400)

Enable generating interrupt when User TC1 out goes low.

IRQM\_UTC2 (0x0800) User TC2 out

Enable generating interrupt on User TC2 ticks.

IRQM\_DIGITAL\_IT (0x1000)

Enable generating interrupt when a Digital Interrupt occurs.

IRQM\_EXTERNAL\_IT (0x2000)

Enable generating interrupt on the External Interrupt pin. Interrupt will be generated on the rising or falling edge, depending on the programmed External Interrupt polarity (see SetEIntPolarity4520).

IRQM\_ETRIG\_RISING (0x4000)

IRQM\_ETRIG\_FALLING (0x8000)

Enable generating interrupt on External Trigger rising/falling-edge.

### See Also

Interrupts, InstallCallbackIrqHandler4520

---

## SetLastDescriptorBlock4520

---

### Syntax

```
BOOL SetLastDescriptorBlock4520(RTDHANDLE hBoard, int  
dma_channel, uint32 dma_local, uint32 direction, uint32 offset,  
uint32 length);
```

### Description

Setup the next chained mode descriptor block. The maximum number of blocks is 16.

### Parameters

hBoard:	device handle
dma_channel:	DMA channel index (0/1)
dma_local:	local (onboard) DMA source/destination
direction:	transfer direction
offset:	the offset from the DMA buffer+0x200 address. the DMA transfer starts from/to here
length:	DMA transfer length in bytes

### Return Value

TRUE	If there is no error.
FALSE	If there is an error.

---

## SetLastDescriptorBlock4520

---

### Syntax

```
BOOL SetLastDescriptorBlock4520(RTDHANDLE hBoard, int  
dma_channel, uint32 local, uint32 direction,  
uint32 direct_address, uint32 length);
```

### Description

Setup the last chained mode descriptor block with direct physical target address.

Starting a DMA cycle to improper address may hangup your system!

### Parameters

hBoard:	device handle
dma_channel:	DMA channel index (0/1)
dma_local:	local DMA source/destination address
direction:	direction
direct_address:	physical DMA address
length:	DMA transfer byte count

### Return Value

TRUE	If there is no error.
FALSE	If there is an error.

---

## SetNextDescriptorBlock4520

---

### Syntax

```
BOOL SetNextDescriptorBlock4520(RTDHANDLE hBoard, int  
dma_channel, uint32 dma_local, uint32 direction, uint32 offset,  
uint32 length);
```

### Description

Setup the next chained mode descriptor block. The maximum number of blocks is 16.

### Parameters

hBoard:	device handle
dma_channel:	DMA channel index (0/1)
dma_local:	local (onboard) DMA source/destination
direction:	transfer direction
offset:	the offset from the DMA buffer+0x200 address. the DMA transfer starts from/to here

length: DMA transfer length in bytes

### Return Value

TRUE If there is no error.  
FALSE If there is an error.

---

## SetNextDirectDescriptorBlock4520

---

### Syntax

```
SetNextDirectDescriptorBlock4520(RTDHANDLE hBoard, int  
dma_channel, uint32 local, uint32 direction,  
uint32 direct_address, uint32 length);
```

### Description

Setup the next chained mode descriptor block with direct physical target address.

Starting a DMA cycle to improper address may hangup your system!

### Parameters

hBoard: device handle  
dma\_channel: DMA channel index (0/1)  
dma\_local: local DMA source/destination address  
direction: direction  
direct\_address: physical DMA address  
length: DMA transfer byte count

### Return Value

TRUE If there is no error.  
FALSE If there is an error.

---

## SetPacerClock4520

---

### Syntax

```
unsigned SetPacerClock4520 (RTDHANDLE hBoard, uint16 start,  
uint16 stop, uint16 repeat,  
uint16 src, float rate, int clock);
```

### Description

Configure the Pacer Clock.

Select start/stop trigger, single or repeat mode, internal or external pacer clock source, rate. (See the individual SetPacerxxx functions.) This is a deprecated function for legacy compatibility. Use the SetPacerClockF4520 function returning float value.

## Parameters

hBoard:	device handle
start:	Pacer Start trigger (see SetPacerStart4520).
stop:	Pacer Stop trigger (see SetPacerStop4520).
repeat:	Pacer Clock retriggering mode (see SetPacerRepeat4520).
src:	Pacer Clock select (see SetPacerSource4520).
rate:	Pacer Clock rate in Hertz. Ignored if Pacer Clock source ( <i>src</i> ) is External Pacer Clock.
clock:	Pacer Clock Primary Clock Frequency select 0 = 8 MHz 1 = 20 MHz This argument is ignored if the EPLD version is less than 10.

## Return Value

The pacer clock counter value set by the function. This value could be different as the rate parameter due rounding.

---

## SetPacerClockF4520

---

### Syntax

```
float SetPacerClockF4520 (RTDHANDLE hBoard, uint16 start,  
                          uint16 stop, uint16 repeat,  
                          uint16 src, float rate, int clock);
```

### Description

Configure the Pacer Clock.

Select start/stop trigger, single or repeat mode, internal or external pacer clock source, rate. (See the individual SetPacerxxx functions.)

### Parameters

hBoard:	device handle
start:	Pacer Start trigger (see SetPacerStart4520).
stop:	Pacer Stop trigger (see SetPacerStop4520).
repeat:	Pacer Clock retriggering mode (see SetPacerRepeat4520).
src:	Pacer Clock select (see SetPacerSource4520).
rate:	Pacer Clock rate in Hertz. Ignored if Pacer Clock source ( <i>src</i> ) is External Pacer Clock.
clock:	Pacer Clock Primary Clock Frequency select 0 = 8 MHz 1 = 20 MHz This argument is ignored if the EPLD version is less than 10.

### Return Value

The actual Pacer Clock rate.

---

## SetPacerRate4520

---

### Syntax

```
unsigned SetPacerRate4520 ( RTDHANDLE hBoard, float rate, int  
clock );
```

### Description

Program Pacer Clock rate. This is a deprecated function for legacy compatibility. Use the SetPacerRateF4520 function.

### Parameters

hBoard: device handle  
rate: Clock frequency (Hertz). Valid values are between 0.0018Hz and 1.25MHz.  
clock: Pacer Clock Primary Clock Frequency select  
0 = 8 MHz  
1 = 20 MHz  
This argument is ignored if the EPLD version is less than 10.

### Return Value

The pacer clock counter value set by the function. This value could be different as the rate parameter due rounding.

---

## SetPacerRateF4520

---

### Syntax

```
float SetPacerRateF4520 ( RTDHANDLE hBoard, float rate, int  
clock );
```

### Description

Program Pacer Clock rate. This function sets the rate by calculating a divisor of the Primary Clock (that equals either 8 MHz or 20 MHz). Since this divisor is rounded to an integer value, the actual pacer clock frequency can be different from 'rate' parameter.

### Parameters

hBoard: device handle  
rate: Clock frequency (Hertz). Valid values are between 0.0018Hz and 1.25MHz.  
clock: Pacer Clock Primary Clock Frequency select  
0 = 8 MHz  
1 = 20 MHz  
This argument is ignored if the EPLD version is less than 10.

## Return Value

The actual Pacer Clock rate.

---

## SetPacerRepeat4520

---

### Syntax

```
void SetPacerRepeat4520 (RTDHANDLE hBoard, uint16 repeat);
```

### Description

Pacer Start Trigger Mode select.

### Parameters

hBoard: device handle  
repeat: 0x0 = Single Cycle Mode  
          New cycle is possible only after a Software Pacer Start command (StartPacer4520).  
          0x1 = Trigger Repeat Mode  
          Pacer Clock can be restarted by the selected Pacer Start Trigger.  
          Use PCLK\_NO\_REPEAT/PCLK\_REPEAT for argument 'repeat'.

---

## SetPacerSource4520

---

### Syntax

```
void SetPacerSource4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

Select Pacer Clock Source.

### Parameters

hBoard: device handle  
src: 0x0 = External Pacer Clock  
      0x1 = Internal Pacer Clock  
      Use PCLK\_XXXX for argument 'src'.

---

## SetPacerStart4520

---

### Syntax

```
void SetPacerStart4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

Select Pacer Clock start trigger.

### Parameters

hBoard: device handle  
          PCLK\_START\_SOFTWARE (0x0)  
          Pacer Clock is started by a call to StartPacer4520.

#### PCLK\_START\_ETRIG (0x1)

A positive- or negative-going edge (depending on the setting of the trigger polarity, SetEtrgPolarity4520) on the external TRIGGER INPUT line, will start the Pacer Clock. The pulse duration should be at least 100 nanoseconds.

NOTE: If the Pacer Clock is already running when an External Trigger occurs, behavior depends on the programmed Pacer Clock Repeat Mode (see SetPacerRepeat4520).

#### PCLK\_START\_DIGITAL\_IT (0x2)

Pacer Clock is started when a Digital Interrupt occurs.

#### PCLK\_START\_UTC2 (0x3)

Pacer Clock is started by User T/C 2 (when counter reaches 0).

#### PCLK\_START\_SBUS0 (0x4)

#### PCLK\_START\_SBUS1 (0x5)

#### PCLK\_START\_SBUS2 (0x6)

Pacer Clock is started by a positive edge of Synchron Bus line 0/1/2, respectively.

#### PCLK\_START\_D\_SOFTWARE (0x8)

#### PCLK\_START\_D\_ETRIG (0x9)

#### PCLK\_START\_D\_DIGITAL\_IT (0xA)

#### PCLK\_START\_D\_UTC2 (0xB)

#### PCLK\_START\_D\_SBUS0 (0xC)

#### PCLK\_START\_D\_SBUS1 (0xD)

#### PCLK\_START\_D\_SBUS2 (0xE)

These start trigger sources provide delayed triggering (see Pre/Post Triggering).

When the selected trigger occurs, the A/D Delay Counter starts the Pacer Clock, but Pacer Clock is not yet enabled to drive the A/D converter. When Delay Counter counts down to zero, Pacer Clock starts driving the A/D converter.

#### PCLK\_START\_ETRIG\_GATED (0xF)

Pacer Clock is started/stopped by the External Trigger line.

In this mode, the External Trigger line serves as the gate of the Pacer Clock. Pacer Clock runs when line goes high and stops when the line goes low, or vice-versa, depending on the programmed External Trigger polarity (see SetEtrgPolarity4520).

In this mode of operation, programming Pacer Clock stop trigger (SetPacerStop4520) has no effect.

---

## SetPacerStop4520

---

### Syntax

```
void SetPacerStop4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

Select Pacer Clock Stop Trigger.

## Parameters

hBoard:	device handle
src:	PCLK_STOP_SOFTWARE (0x0) Pacer Clock is stopped by a call to StopPacer4520.
	PCLK_STOP_ETRIG (0x1) Pacer Clock stops when an External Trigger occurs. A positive- or negative-going edge (depending on the trigger polarity programmed by SetEtrgPolarity4520) on the external TRIGGER INPUT line, will stop the Pacer Clock. The pulse duration should be at least 100 nanoseconds.
	PCLK_STOP_DIGITAL_IT (0x2) Pacer Clock stops when a Digital Interrupt occurs.
	PCLK_STOP_ACNT (0x3) Pacer Clock is stopped by the About Counter. Pacer Clock stops when About Counter counts down to zero and About Counter Stop is enabled (see EnableAcntStop4520). About Counter counts the samples that are actually entering the A/D FIFO Use this mode for post triggering (see Pre/Post Triggering).
	PCLK_STOP.UTC2 (0x4) Pacer Clock stops when User T/C2 counts down to zero.
	PCLK_STOP_SBUS0 (0x5)
	PCLK_STOP_SBUS1 (0x6)
	PCLK_STOP_SBUS2 (0x7) Pacer Clock is stopped when a rising edge occurs on Synchron Bus line 0/1/2, respectively.
	PCLK_STOP_A_SOFTWARE (0x8)
	PCLK_STOP_A_ETRIG (0x9)
	PCLK_STOP_A_DIGITAL_IT (0xA)
	PCLK_STOP_A.UTC2 (0xC)
	PCLK_STOP_A_SBUS0 (0xD)
	PCLK_STOP_A_SBUS0 (0xE)
	PCLK_STOP_A_SBUS0 (0xF) These stop trigger sources are to provide about triggering (see Pre/Port Triggering). When the selected stop trigger occurs, About Counter starts counting the A/D samples entering the A/D FIFO. When About Counter counts down to zero, it stops the Pacer Clock.

---

## SetPclkSize4520

---

### Syntax

```
void SetPclkSize4520 (RTDHANDLE hBoard, uint16 size);
```

### Description

Select Pacer Clock size.

NOTE: Pacer Clock size selection is ignored if Pacer Clock source is External Pacer Clock (see SetPacerSource4520).

### Parameters

hBoard: device handle  
size: PCLK\_16BIT (0) = 16 bit divider  
A/D converter connects to Pacer Clock 0.  
PCLK\_32BIT (1) = 32 bit divider  
A/D converter connects to Pacer Clock 1 that is cascaded with Pacer Clock 0

---

## SetSbus0Source4520

---

### Syntax

```
void SetSbus0Source4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

Select Synchron Bus line 0 signal.

NOTE: To export a signal to the Synchron Bus it is not enough to set the source by this function. You also have to enable the selected source signal to appear on the bus by calling EnableSBus04520.

### Parameters

hBoard: device handle  
src: SBUS0\_START\_ADC\_COMMAND (0)  
Synchron Bus line 0 will be equivalent with the signal caused by the execution of StartConversion4520.  
SBUS0\_PCLK (1)  
Put the Out line of the Pacer Clock on the bus.  
SBUS0\_BCLK (2)  
Put the Out line of the Burst Clock on the bus.  
SBUS0\_CGT\_DAC1\_UPDATE (3)  
Bus line 0 will be equivalent with the signal of the CGT controlled D/A1 updates.  
SBUS0\_ETRG (4)  
Put External Trigger signal on the bus.  
SBUS0\_START\_DACS\_COMMAND  
Bus line 0 will be equivalent with the signal caused by a simultaneous D/A1 and D/A2 update by UpdateDACAll4520.  
SBUS0\_DAC\_CLK (6)  
Put the Out line of the D/A Clock on the bus.  
SBUS0.UTC2 (7)  
Put the Out line of the User TC2 on the bus.

### See Also

Synchron Bus

---

## SetSbus1Source4520

---

### Syntax

```
void SetSbus1Source4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

Select Synchron Bus line 1 signal.

**NOTE:** To export a signal to the Synchron Bus it is not enough to set the source by this function. You also have to enable the selected source signal to appear on the bus by calling EnableSBus14520.

### Parameters

**hBoard:** device handle

**src:** SBUS1\_START\_ADC\_COMMAND (0)  
Synchron Bus line 1 will be equivalent with the signal caused by the execution of StartConversion4520.

SBUS1\_PCLK (1)  
Put the Out line of the Pacer Clock on the bus.

SBUS1\_BCLK (2)  
Put the Out line of the Burst Clock on the bus.

SBUS1\_CGT\_DAC2\_UPDATE (3)  
Bus line 1 will be equivalent with the signal of the CGT controlled D/A2 updates.

SBUS1\_ETRG (4)  
Put External Trigger signal on the bus.

SBUS1\_START\_DACS\_COMMAND  
Bus line 1 will be equivalent with the signal caused by a simultaneous D/A1 and D/A2 update by UpdateDACAll4520.

SBUS1\_DAC\_CLK (6)  
Put the Out line of the D/A Clock on the bus.

SBUS1.UTC2 (7)  
Put the Out line of the User TC2 on the bus.

### See Also

Synchron Bus

---

## SetSbus2Source4520

---

### Syntax

```
void SetSbus2Source4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

Select Synchron Bus line 2 signal.

**NOTE:** To export a signal to the Synchron Bus it is not enough to set the source by this function. You also have to enable the selected source signal to appear on the bus by calling EnableSBus24520.

## Parameters

hBoard:	device handle
src:	SBUS2_START_ADC_COMMAND (0) Synchron Bus line 2 will be equivalent with the signal caused by the execution of StartConversion4520.
	SBUS2_START_PCLK_COMMAND (1) Synchron Bus line 2 will be equivalent with the signal caused by the execution of StartPacer4520.
	SBUS2_STOP_CLK_COMMAND (2) Synchron Bus line 2 will be equivalent with the signal caused by the execution of StopPacer4520.
	SBUS2_UPDATE_DAC1 (3)
	SBUS2_UPDATE_DAC2 (4) Bus line 2 will be equivalent with the signal caused by a D/A1 and D/A2 update by UpdateDAC14520/UpdateDAC24520.
	SBUS2_EPCLK (5) Put External Pacer Clock signal on the bus.
	SBUS2_ETRG (6) Put External Trigger signal on the bus.
	SBUS1_UTC2 (7) Put the Out line of the User TC2 on the bus.

## See Also

Synchron Bus

---

SetUout0Source4520

SetUout1Source4520

---

## Syntax

```
void SetUout0Source4520 (RTDHANDLE hBoard, uint16 src);  
void SetUout1Source4520 (RTDHANDLE hBoard, uint16 src);
```

## Description

Select User Output 0/1 signal.

## Parameters

hBoard:	device handle
src:	0x0 = A/D Conversion Signal 0x1 = D/A1 Update 0x2 = D/A2 Update 0x3 = Software Programmable by WriteUserOutput4520. Use UOUT_XXXX for argument 'src'.

---

## SetupAboutCounter4520

---

### Syntax

```
void SetupAboutCounter4520 (RTDHANDLE hBoard,  
                             uint16 samples,  
                             uint16 enable );
```

### Description

Program the about counter. The function programs the rate and enables or disables the about counter.

NOTE: This function is a short for the following calls:

```
Set8254Mode4520 (hBoard, TC_ACNT, M8254_RATE_GENERATOR);  
Set8254Divisor4520 (hBoard, TC_ACNT, samples);  
EnableAcntStop4520 (hBoard, enable);
```

### Parameters

hBoard:	device handle
samples:	initial count
enable:	0: stop disabled 1: stop enable

---

## SetupBurst4520

---

### Syntax

```
void SetupBurst4520 (RTDHANDLE hBoard, uint16 start,  
                     float rate, int clock);
```

### Description

Setup the burst clock. This is a deprecated function for legacy compatibility. Please use function SetupBurstF4520.

NOTE: This function is a short for the following calls:

```
SetBurstStart4520 (hBoard, start);  
SetBurstRate4520 (hBoard, rate);
```

### Parameters

hBoard:	device handle
start:	Burst Trigger source.
rate:	Burst clock rate (in Hertz).
clock:	Burst Clock Primary Clock Frequency select 0 = 8 MHz 1 = 20 MHz This argument is ignored if the EPLD version is less than 10.

---

## SetupBurstF4520

---

### Syntax

```
float SetupBurstF4520 (RTDHANDLE hBoard, uint16 start,  
                      float rate, int clock);
```

### Description

Setup the burst clock.

(See the SetBurstStart4520 and the SetBurstRateF4520 functions).

NOTE: This function is a short for the following calls:

```
SetBurstStart4520(hBoard, start);  
SetBurstRateF4520(hBoard, rate);
```

### Parameters

hBoard:	device handle
start:	Burst Trigger source.
rate:	Burst clock rate (in Hertz).
clock:	Burst Clock Primary Clock Frequency select 0 = 8 MHz 1 = 20 MHz This argument is ignored if the EPLD version is less than 10.

### Return Value

The actual Burst Clock rate.

---

## SetupCgtRow4520

---

### Syntax

```
void SetupCgtRow4520 (ioctl_t *cscRow,  
                     int aChannel, int aGain, int aRange,  
                     int aNRSE, int aGround, int aPause,  
                     int aDAC1, int aDAC2, int aSkip );
```

### Description

Setup a Channel-Gain Table entry.

### Parameters

ioctl_t *cscRow	structure for CGT rows
aChannel	Analog Input channel to sample (0...15 for AIN1...AIN16).
aGain	Analog Input gain to apply on the analog input (0...7 for gains 1x, 2x, 4x, 8x, 16x, 32x, 64x, and 128x, respectively).

	(The 64x and 128x gain are not available on the DM7520 board.)
aRange	Analog Input range: AIN_BIP5 (0) ±5 Volts AIN_BIP10 (1) ±10 Volts AIN_UNIP10 (2) 0...+10 Volts
aNRSE	Ground referencing mode: NRSE_AGND (0) Non-Ground Referenced Single-Ended NRSE_AINS (1) Ground Referenced Single-Ended
aGround	Analog Input Type: GND_SE (0) Single-Ended GND_DIFF (1) Differential
aPause	Pause Bit 0 do not pause CGT execution 1 pause CGT execution
aDAC1	
ADAC2	D/A1 / D/A2 Update Bit: 0 do not update D/A1 / D/A2 1 update D/A1 / D/A2
aSkip	Skip Bit: 0 store converted A/D sample to FIFO 1 discard converted A/D sample

### See Also

Channel-Gain Circuitry, WriteCGTAnalog4520

---

## SetupChainedDMA4520

---

### Syntax

```
BOOL SetupChainedDMA4520(RTDHANDLE hBoard, int dma_channel,
uint16 demand_source, uint32 mode, BOOL it_enable);
```

### Description

Setup the chained mode DMA operation.

### Parameters

hBoard:	device handle
dma_channel:	DMA channel index (0/1)
demand_source:	source of demand mode DMA request
mode:	mode
it_enable:	enable DMA done interrupt

### Return Value

TRUE	If there is no error.
FALSE	If there is an error.

---

## SetupDAC4520

---

### Syntax

```
void SetupDAC4520 (RTDHANDLE hBoard, int idx, uint16 rng,
                  uint16 cycle, uint16 src );
```

### Description

Setup DAC.

NOTE: This function is a short for the following calls:

```
SetDACXRange4520(hBoard, rng);
SetDACXStart4520(hBoard, src);
SetDACXCycle4520(hBoard, cycle);
```

### Parameters

hBoard:	device handle
idx:	0: DAC1, 1: DAC2
rng:	range
cycle:	cycle mode
src:	start source

---

## SetupDelayCounter4520

---

### Syntax

```
void SetupDelayCounter4520 (RTDHANDLE hBoard, uint16 samples);
```

### Description

Program the delay counter.

NOTE: This function is a short for the following calls:

```
Set8254Mode4520(hBoard, TC_DCNT, M8254_RATE_GENERATOR);
Set8254Divisor4520(hBoard, TC_DCNT, samples);
```

### Parameters

hBoard:	device handle
samples:	number of samples to delay

---

## SetupDirectDMA4520

---

### Syntax

```
BOOL SetupDirectDMA4520(RTDHANDLE hBoard, int dma_channel,  
uint32 dma_source, uint16 demand_source, uint32 mode, int  
direction, uint32 direct_address, uint32 length, BOOL  
it_enable);
```

### Description

Setup DMA channel with direct physical target address.

Starting a DMA cycle to improper address may hangup your system!

### Parameters

hBoard:	device handle
dma_channel:	DMA channel index (0/1)
dma_source:	address of DMA data source
demand_source:	hardware signal for demand mode DMA
mode:	DMA mode
direction:	data flow direction
direct_address:	physical address to DMA
length:	number of bytes to transfer
enable:	DMA done interrupt enable

### Return Value

TRUE	If there is no error.
FALSE	If there is an error.

---

## SetupDMA4520

---

### Syntax

```
BOOL SetupDMA4520(RTDHANDLE hBoard, int dma_channel, uint32  
dma_source, uint16 demand_source, uint32 mode, int direction,  
uint32 offset, uint32 length, BOOL it_enable);
```

### Description

Setup DMA channel before start data transfer.

### Parameters

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

dma\_source: address of DMA data source  
demand\_source: hardware signal for demand mode DMA  
mode: DMA mode  
direction: data flow direction  
offset: data offset from the beginning of the driver allocated buffer  
length: number of bytes to transfer  
enable: DMA done interrupt enable

### Return Value

TRUE If there is no error.  
FALSE If there is an error.

---

## SetupPort04520

---

### Syntax

```
void SetupPort04520 (RTDHANDLE hBoard, uint8 direction,  
                    uint8 mask, uint8 mode, uint8  
                    compare, uint8 clock, BOOL irq );
```

### Description

Configure Digital I/O Port 0.

NOTE: This function is a short for the following calls:

```
SetDIO0Direction4520 (hBoard, direction);  
SetDIO0Mask4520 (hBoard, mask);  
SetDIO0CompareMode4520 (hBoard, mode);  
SetDIO0CompareValue4520 (hBoard, compare);  
SetDIO0Clock4520 (hBoard, clock);  
EnableDIO0Irq4520 (hBoard, irq);
```

### Parameters

hBoard: device handle  
direction: 8-bit mask specifying the direction of the individual lines (see SetDIO0Direction4520)  
mask: Digital Interrupt generation mask (see SetDIO0Mask4520)  
mode: Triggering Mode (see SetDIO0CompareMode4520)  
compare: Compare value for Match mode (see SetDIO0CompareValue4520)  
clock: Clock for Digital Triggering (see SetDIO0Clock4520)  
irq: Enable/disable Digital Interrupt generation (see EnableDIO0Irq4520)

---

## SetupPort14520

---

### Syntax

```
void SetupPort14520 (RTDHANDLE hBoard, uint8 direction,  
                    uint8 mask, uint8 mode,  
                    uint8 compare, uint8 clock,  
                    BOOL irq );
```

## Description

Configure Digital I/O Port 0.

NOTE: This function is a short for the following calls:

```
SetDIO1Direction4520(hBoard,direction);
```

### Parameters

hBoard:	device handle
direction:	port direction (see SetDIO1Direction4520)
mask:	N/A, set to zero
mode:	N/A, set to zero
compare:	N/A, set to zero
clock:	N/A, set to zero
irq:	N/A, set to zero

---

## SetupSampleCounter4520

---

### Syntax

```
void SetupSampleCounter4520 (RTDHANDLE hBoard,  
                             int counter,uint16 src,  
                             uint16 divisor);
```

### Description

Program one of the 3 sample counters.

NOTE: This function is a short for the following calls:

```
switch ( counter )  
{  
case TC_ADC_SCNT:  
    SetAdcntSource4520(hBoard,src);  
    LoadAdcnt4520(hBoard,divisor);  
    break;  
case TC_DAC1_UCNT:  
    LoadUcnt14520(hBoard,divisor);  
    break;  
case TC_DAC2_UCNT:  
    LoadUcnt24520(hBoard,divisor);  
    break;  
}
```

### Parameters

hBoard:	device handle
counter	Use TC_XXXX for argument 'counter'.
source	Use ADC_SCNT_XXXX for argument 'src'.
divisor	16-bit initial count

---

## SetupSbus4520

---

### Syntax

```
void SetupSbus4520 (RTDHANDLE hBoard, int idx, uint16 src,
                  uint16 enable);
```

### Description

Setup Synchron bus.

NOTE: This function is a short for the following calls:

```
SetSbusXSource4520(hBoard, src);
EnableXbus04520(hBoard, enable);
```

### Parameters

hBoard: device handle  
idx: 0: Synchron Bus line 0  
1: Synchron Bus line 1  
2: Synchron Bus line 2  
src: source  
enable: enable/disable

---

## SetupTimerCounter4520

---

### Syntax

```
void SetupTimerCounter4520 (RTDHANDLE hBoard, int counter,
                            uint8 mode, uint16 divisor );
```

### Description

Program a 8254 Timer/Counter.

NOTE: This function is a short for the following calls:

```
Set8254Mode4520(hBoard, counter, mode);
Set8254Divisor4520(hBoard, counter, divisor);
```

### Parameters

hBoard: device handle  
counter: One of the 16-bit timer/counters of the board.  
Use TC\_XXXX for argument 'counter'.  
mode: One of the 6 8254 operation modes.  
Use M8254\_XXXX for argument 'mode'.  
divisor: 16-bit divisor to divide the clock input frequency.

---

## SetUtc0Clock4520

---

### Syntax

```
void SetUtc0Clock4520 (RTDHANDLE hBoard, uint16 src);
```

## Description

Select User Timer/Counter 0 clock input.

## Parameters

hBoard:	device handle
src:	CUTC0_8MHZ (0) Connect User T/C0 clock input to the 8MHz on-board crystal.
	CUTC0_EXT_TC_CLOCK1 (1) Connect User T/C0 clock input to the External TC Clock 1 pin on the I/O connector.
	CUTC0_EXT_TC_CLOCK2 (2) Connect User T/C0 clock input to the External TC Clock 2 pin on the I/O connector.
	CUTC0_EXT_PCLK (3) Connect User T/C0 clock input to the External Pacer Clock pin on the I/O connector.

---

## SetUtc0Gate4520

---

### Syntax

```
void SetUtc0Gate4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

User Timer/Counter 0 Gate Select.

### Parameters

hBoard:	device handle
src:	GUTC0_NOT_GATED (0) Connect User T/C0 Gate to +5Volts (let clock run).
	GUTC0_GATED (1) Connect User T/C0 Gate to the ground (stop clock).
	GUTC0_EXT_TC_GATE1 (2) Connect User T/C0 Gate to the External TC Gate 1 pin on the I/O connector.
	GUTC0_EXT_TC_GATE2 (3) Connect User T/C0 Gate to the External TC Gate 2 pin on the I/O connector.

---

## SetUtc1Clock4520

---

### Syntax

```
void SetUtc1Clock4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

User Timer/Counter 1 Clock Select.

## Parameters

hBoard:	device handle
src:	CUTC1_8MHZ (0) Connect User T/C1 clock input to the 8MHz on-board crystal.
	CUTC1_EXT_TC_CLOCK1 (1) Connect User T/C1 clock input to the External TC Clock 1 pin on the I/O connector.
	CUTC1_EXT_TC_CLOCK2 (2) Connect User T/C1 clock input to the External TC Clock 2 pin on the I/O connector.
	CUTC1_EXT_PCLK (3) Connect User T/C1 clock input to the External Pacer Clock pin on the I/O connector.
	CUTC1_UTC0_OUT (4) Connect (cascade) User T/C1 clock input to User T/C0 out.
	CUTC1_DIN_SIGNAL (5) Connect User T/C1 clock input to the High-Speed Digital Input sampling signal.

---

## SetUtc1Gate4520

---

### Syntax

```
void SetUtc1Gate4520 (RTDHANDLE hBoard, uint16 src);
```

### Description

User Timer/Counter 1 Gate Select.

### Parameters

hBoard:	device handle
src:	GUTC1_NOT_GATED (0) Connect User T/C1 Gate to +5Volts (let clock run).
	GUTC1_GATED (1) Connect User T/C1 Gate to the ground (stop clock).
	GUTC1_EXT_TC_GATE1 (2) Connect User T/C1 Gate to the External TC Gate 1 pin on the I/O connector.
	GUTC1_EXT_TC_GATE2 (3) Connect User T/C1 Gate to the External TC Gate 2 pin on the I/O connector.
	GUTC1_UTC0_OUT (4) Connect User T/C1 Gate to User T/C 0 out.

---

## SetUtc2Clock4520

---

### Syntax

```
void SetUtc2Clock4520 (RTDHANDLE hBoard, uint16 src);
```

## Description

User Timer/Counter 2 Clock Select.

## Parameters

hBoard:	device handle
src:	CUTC2_8MHZ (0) Connect User T/C2 clock input to the 8MHz on-board crystal.
	CUTC2_EXT_TC_CLOCK1 (1) Connect User T/C2 clock input to the External TC Clock 1 pin on the I/O connector.
	CUTC2_EXT_TC_CLOCK2 (2) Connect User T/C2 clock input to the External TC Clock 2 pin on the I/O connector.
	CUTC2_EXT_PCLK (3) Connect User T/C2 clock input to the External Pacer Clock pin on the I/O connector.
	CUTC2_UTC1_OUT (4) Connect (cascade) User T/C2 clock input to User T/C1 out.

---

## SetUtc2Gate4520

---

### Syntax

```
void SetUtc2Gate4520 (RTDHANDLE hBoard, uint16 src);
```

## Description

User Timer/Counter 2 Gate Select.

## Parameters

hBoard:	device handle
src:	GUTC2_NOT_GATED (0) Connect User T/C2 Gate to +5Volts (let clock run).
	GUTC2_GATED (1) Connect User T/C2 Gate to the ground (stop clock).
	GUTC2_EXT_TC_GATE1 (2) Connect User T/C2 Gate to the External TC Gate 1 pin on the I/O connector.
	GUTC2_EXT_TC_GATE2 (3) Connect User T/C2 Gate to the External TC Gate 2 pin on the I/O connector.
	GUTC2_UTC1_OUT (4) Connect User T/C2 Gate to User T/C1 out.

---

## ShareDMABuffer4520

---

### Syntax

```
void ShareDMABuffer4520 (RTDHANDLE hBoard, int dma_channel);
```

### **Description**

Share the DMA memory buffer allocated by the driver for DMA transfers between two DMA channels. This function is used when both DMA channel works at the same time.

Before sharing a channel buffer, the channel must initialize with the InstallDMA4520 function.

### **Parameters**

hBoard:                device handle  
dma\_channel:         channel to share

---

## **StartConversion4520**

---

### **Syntax**

```
void StartConversion4520(RTDHANDLE hBoard);
```

### **Description**

This routine is used to generate a software trigger or enable hardware triggers depending on the programmed A/D conversion signal (SetConversionSelect4520).

### **Parameters**

hBoard:                device handle

---

## **StartBurst4520**

---

### **Syntax**

```
void StartBurst4520 (RTDHANDLE hBoard);
```

### **Description**

Generate a software Burst Clock start.

NOTE:                This function is needed and has only effect if Burst Clock is programmed for software start (see SetBurstStart4520).

### **Parameters**

hBoard:                device handle

---

## **StartChainedDMA04520** **StartChainedDMA14520**

---

### **Syntax**

```
void StartChainedDMA04520 ( RTDHANDLE hBoard );  
void StartChainedDMA14520 ( RTDHANDLE hBoard );
```

### **Description**

Start the chained DMA operation in normal or demand mode.

### **Parameters**

hBoard:                device handle

---

---

## StartDACClock4520

---

### Syntax

```
void StartDACClock4520 (RTDHANDLE hBoard);
```

### Description

Software DAC Clock Start.

In the DACClockStartSelect4520 function can be selected the SW DAC clock start.

NOTE: This function works only on DM7520 with EPLD version 9 or greater.

### Parameters

hBoard: device handle

---

## StartDMA04520

## StartDMA14520

---

### Syntax

```
void StartDMA04520 ( RTDHANDLE hBoard );
```

```
void StartDMA14520 ( RTDHANDLE hBoard );
```

### Description

Start the DMA operation in normal or demand mode.

### Parameters

hBoard: device handle

---

## StartHdin4520

---

### Syntax

```
void StartHdin4520 (RTDHANDLE hBoard);
```

### Description

Generate a High-Speed Digital Input start trigger.

NOTE: This function is needed and has only effect if High Speed Digital Input is programmed for software start (see SetHdinStart4520).

### Parameters

hBoard: device handle

---

## StartPacer4520

---

### Syntax

```
void StartPacer4520 (RTDHANDLE hBoard);
```

### Description

Generate a software Pacer Clock start trigger.

NOTE: This function is needed and has only effect if Pacer Clock is programmed for software start (see SetPacerStart4520).

### Parameters

hBoard: device handle

---

## StopDACClock4520

---

### Syntax

```
void StopDACClock4520 (RTDHANDLE hBoard);
```

### Description

Software DAC Clock Stop.

In the DACClockStopSelect4520 function can be selected the SW DAC clock stop.

NOTE: This function works only on DM7520 with EPLD version 9 or greater.

### Parameters

hBoard: device handle

---

## StopPacer4520

---

### Syntax

```
void StopPacer4520 (RTDHANDLE hBoard);
```

### Description

Generate a software Pacer Clock stop trigger.

NOTE: This function is needed and has only effect if Pacer Clock is programmed for software stop (see SetPacerStop4520).

### Parameters

hBoard: device handle



---

**UpdateAllDAC4520**

---

**Syntax**

```
void UpdateAllDAC4520 (RTDHANDLE hBoard);
```

**Description**

Software Simultaneous D/A1 and D/A2 Update.

NOTE: This function is needed and has only effect if D/As are programmed for software update (see SetDAC1Update4520 and SetDAC2Update4520).

**Parameters**

hBoard: device handle

---

**UpdateDAC14520****UpdateDAC24520**

---

**Syntax**

```
void UpdateDAC14520 (RTDHANDLE hBoard);
```

```
void UpdateDAC24520 (RTDHANDLE hBoard);
```

**Description**

Software D/A 1 or D/A 2 Update.

NOTE: This function is needed and has only effect if the D/A is programmed for software update (see SetDAC1Update4520 and SetDAC2Update4520).

**Parameters**

hBoard: device handle



---

## WriteCGTAnalog4520

---

### Syntax

```
void WriteCGTAnalog4520 (RTDHANDLE hBoard, uint16 entry);
```

### Description

Write entry to the Analog part of the Channel-Gain Table.

**NOTE:** If you have cleared the Channel-Gain Table (by the ClearCGT4520 function), the first byte written will be placed in the first entry of the table. The second byte will be placed in the second entry, and so on. If you are adding to an existing table, the new data written will be added at the end.

**NOTE:** Remember to select Channel-Gain Table to control sampling (see EnableCGT4520).

### Parameters

hBoard:                device handle  
entry:                 table entry

### See Also

Channel-Gain Circuitry, WriteCGTDigital4520, ClearCGT4520

---

## WriteCGTDigital4520

---

### Syntax

```
void WriteCGTDigital4520 (RTDHANDLE hBoard, uint8 entry);
```

### Description

Channel Gain / Digital Output Write Digital table (To control external MUX).

**NOTE:** The first entry made into the Digital Table lines up with the first entry made into the Analog Table, the second entry made into the Digital Table lines up with the second entry made into the Analog Table, and so on. Make sure that, if you add to an existing table and did not program the Digital Table portion when you made your Analog Table entries previously, you fill those entries with digital data first before entering the desired added data. Since the first digital entry you make always lines up with the first analog entry made, failure to do this will cause the analog and digital control data to be misaligned in the table.

**NOTE:** You cannot turn the digital control lines off for part of a conversion sequence and then turn them on for the remainder of the sequence. Note that the digital data programmed here is sent out on the Port 1 digital I/O lines whenever this portion of the table is enabled by the EnableCGTDigital4520.

## Parameters

hBoard: device handle  
entry: table entry

## See Also

Channel-Gain Circuitry, WriteCGTAnalog4520, ClearCGT4520

---

## WriteCGTLatch4520

---

### Syntax

```
void WriteCGTLatch4520 (RTDHANDLE hBoard, uint16 entry);
```

### Description

Write ADC channel gain latch (Single-channel mode).

NOTE: Remember to select Channel-Gain Latch to control sampling (see EnableCGT4520).

## Parameters

hBoard: device handle  
entry: latch entry

## See Also

Channel-Gain Circuitry

---

## WriteDac1Fifo4520 WriteDac2Fifo4520

---

### Syntax

```
void WriteDac1Fifo4520 (RTDHANDLE hBoard, uint16 value);  
void WriteDac2Fifo4520 (RTDHANDLE hBoard, uint16 value);
```

### Description

Write D/A 1 or D/A 2 FIFO.

## Parameters

hBoard: device handle  
value: 16-bit value to enter the FIFO.

---

## WriteDIO04520

---

### Syntax

```
void WriteDIO04520 (RTDHANDLE hBoard, uint8 value);
```

### Description

Set Digital Port 0 output lines.

NOTE: Bits in argument 'value' are ignored for lines, which are programmed for input.

**Parameters**

hBoard: device handle  
value: 8-bit value to write to port.

---

**WriteDIO14520**

---

**Syntax**

```
void WriteDIO14520 (RTDHANDLE hBoard, uint8 Data);
```

**Description**

Write to Digital I/O Port 1.

NOTE: This function has no effect if Digital I/O Port 1 is programmed for input.

**Parameters**

hBoard: device handle  
Data: data out

---

**WriteUserOutput4520**

---

**Syntax**

```
void WriteUserOutput4520 (RTDHANDLE hBoard, uint16 data);
```

**Description**

This routine is used to write User Output Line 0 & 1.

**Parameters**

hBoard: device handle  
data: output value (only bits 0 and 1 are used)  
Bit 0 Value for User Output Line 0.  
Bit 1 Value for User Output Line 1.



## Example Programs Reference

All example programs are Win32 applications.

<b>Board Feature</b>	<b>Example Program to demonstrate feature</b>
Analog Input	WSOFTTRIG, WINTRPTS, WCGT, W2BOARDS
Analog Output	WDAC
Digital I/O	WDIGITAL
Pacer Clock	WINTRPTS, WCGT, W2BOARDS
Sample Counters	WDMAIN
User Timer/Counters	WTIMERS
Channel-Gain Table	WCGT
Interrupts	WINTRPTS, WCGT, W2BOARDS, WDMAIN, WDMAIN_CHAINED, WDMAIN_DEMAND
DMA	WDMAIN, WDMAIN_DEMAND, WDMAIN_CHAINED, WDMAOUT, WDMAOUT_DEMAND
Using multiple boards	W2BOARDS
Waveform Generation	WDMAOUT, WDMAOUT_DEMAND
Working with the SPM6020 DSP board	SPM_DM
Real-time application with the SPM6020 DSP board	SPM_DM_REALTIME
Network Analyzer	MLSSA
High Speed DAQ	HIGHSPEED
Real-time Control	CONTROL



## **W2BOARDS**

## **GUI (MFC) Example**

---

Example on using two PCI4520 boards in an application.

This example samples AIN1 on two PCI4520s. Sampling on the two boards can be started/stopped independently.

This example uses the first two installed PCI4520 boards.

Both boards are configured in the same way.

A/D converters are set up for Pacer Clock controlled conversions.

The Internal Pacer Clocks are programmed for 5/10Hz, software start/stop.

Channel-Gain Latches are set up for sampling AIN1, single-ended channels, in 5Volts bipolar range.

Interrupts are enabled on A/D FIFO Write.

When a Start button is pressed, an interrupt handler is installed for the corresponding board, its Pacer Clock is started, and a Windows timer is programmed to provide the a polling loop.

The interrupt handlers read 1 sample from the A/D FIFO, which is displayed in the polling loop.

## **WCGT**

## **GUI (MFC) Example**

---

Example on using the Channel-Gain Table.

This example samples two Analog Input channels and saves data to disk. On the screen, it monitors number of samples saved.

This example uses the first installed PCI4520.

A/D converter is set up for Pacer Clock controlled conversions.

The Internal Pacer Clock is programmed for 1kHz, software start/stop.

Channel-Gain Table filled with two entries to sample AIN1 and AIN2 as single-ended channels, in 5Volts bipolar range.

Interrupts are enabled on About Counter countdown.

When the Start button is pressed, an interrupt handler is installed, the Pacer Clock is started, and a Windows timer is programmed to provide the main polling loop.

The interrupt handler reads 512 samples from the A/D FIFO and saves them on disk.

In the polling loop the interrupt counter is monitored.

## **WIODMA**

## **GUI (MFC) Example**

---

Example on how to use two DMA channels simultaneously.

The program shows how to program the PCI4520 to output samples through a DMA channel and read analog data simultaneously on the other DMA channel.

This example uses the first installed PCI4520.

The pacer clock is programmed to perform sampling at the rate of 48kHz. The DMA channel 0 is programmed in demand mode, requesting DMA transfer when the A/D FIFO is half full.

The DMA transfer is continued until the programmed number of samples not transferred. There is an interrupt service routine activated on DMA-done.

In the interrupt routine an IRQ counter is incremented and the sampling is stopped.

The samples are graphically displayed on the screen, and then the sampling is restarted.

The second DMA channel is programmed to transfer data from the PCI memory to the DAC FIFO. The analog output 1 is programmed on bipolar 5 V. A sine-wave is generated and the calculated samples are placed in the DMA buffer.

In this program the D/A 1 update counter is programmed to initiate a DMA cycle in the demand mode data transfer.

If the analog input 1 is connected to the analog output 1, the generated sine-wave is graphed on the screen.

## **WDAC**

## **GUI (MFC) Example**

---

Example on doing simple Digital/Analog conversions.

The output voltage on D/A1 can be changed by an on-screen slider. If D/A1 is hooked back to Analog Input channel 1 (AIN1), this example program monitors the voltage on D/A1.

This example uses the first installed PCI4520.

D/A1 is programmed for 5V bipolar range, software update mode.

AIN1 is sampled via the Channel-Gain Latch with software started conversions.

A Windows timer is programmed to provide the main program loop. In every iteration, an A/D conversion is initiated to sample AIN1 and display voltage on the screen. D/A1 is updated from software when slider is moved.

## **WDAC2**

## **GUI (MFC) Example**

---

Example on how to generate a single cycle on the analog output.

The program demonstrates the using of the DAC clock start/stop trigger selection function and the software DAC clock starting.

This example uses the first installed PCI4520.

D/A1 is programmed for 5V bipolar range, DAC Clock update mode.

When the user presses a button, a single sine wave cycle is generated on the AOUT1.

## **WDIGITAL**

## **GUI (MFC) Example**

---

Example on reading/writing the Digital I/O ports.

Digital I/O Port 1 can be written by entering a value and pressing the Start button. Port 0 is polled on a timely basis. Hook back Port 1 to Port 0 to read back the value written to Port 1 on Port 0

This example uses the first installed PCI4520.

Digital I/O Port 0 is programmed for input, Port 1 is programmed for output.

A Windows timer is programmed to provide the main program loop. In every iteration Port 0 is read and displayed on screen.

Pressing the Start button initiates writing to Port 1.

---

**WDMMAIN****GUI (MFC) Example**

---

Example on normal mode DMA input.

The program shows how to program the PCI4520 to read samples through the DMA channel.

This example uses the first installed PCI4520.

The pacer clock is programmed to perform sampling at 10 kHz. The sample counter is programmed to generate an interrupt when the A/D FIFO is half-full.

In the interrupt routine an IRQ counter is incremented and the DMA is restarted.

The samples are graphically displayed on the screen.

---

**WDMMAIN\_CHAINED****GUI (MFC) Example**

---

Example on normal chained mode DMA input.

The program shows how to program the PCI4520 to perform chained mode DMA transfers.

This example uses the first installed PCI4520.

The pacer clock is programmed to perform sampling at the rate of 10 kHz.

The sample counter is programmed to generate interrupts when the A/D FIFO is half full. In the IRQ routine the DMA is started and the data is read from the FIFO.

The DMA programmed in chained mode and the DMA transfer can be initiated with software start.

In this example program are four chaining mode descriptor block defined. In every descriptor block programmed the DMA controller to read samples from the A/D FIFO and transfer the samples to different PCI memory locations.

Both of the descriptor blocks are setup to transfer 128 samples, thus on interrupt the A/D FIFO is emptied.

The samples are graphically displayed on the screen. The four DMA transfers are good visible on the screen with the gaps between them filled with the default character used by the InstallDMA4520 routine.

---

**WDMMAIN\_DEMAND****GUI (MFC) Example**

---

Example on demand mode DMA input.

The program shows how to program the PCI4520 to read samples through the DMA channel in demand mode.

This example uses the first installed PCI4520.

The pacer clock is programmed to perform sampling at the rate of 1.25 MHz. The DMA programmed in demand mode, requesting DMA transfer when the A/D FIFO is half full.

This program shows how to operate on the maximum (1.25 MHz) sample rate.

The board can operate on the maximum sampling rate and transfers the data to the allocated PCI memory (DMA buffer).

The DMA transfer is continued until the programmed number of samples not transferred. There is an interrupt service routine activated on DMA-done.

In the interrupt routine an IRQ counter is incremented and the sampling is stopped.

The samples are graphically displayed on the screen, and then the sampling is restarted.

## ***WDMAOUT***

## ***GUI (MFC) Example***

---

Example on normal mode DMA output.

The program shows how to program the PCI4520 to output samples through the DMA channel.

This example uses the first installed PCI4520.

The analog output 1 is programmed on bipolar 5 V. A sine wave is generated and the calculated samples are placed in the DMA buffer.

The D/A update counter is programmed to generate an interrupt on countdown and restart the DMA transfer to load new samples to the DAC FIFO.

A wave counter is displayed on the screen during the running of the program.

## ***WDMAOUT\_DEMAND***

## ***GUI (MFC) Example***

---

Example on demand mode DMA output.

The program shows how to program the PCI4520 to output samples through the DMA channel initiated by a hardware event.

This example uses the first installed PCI4520.

The analog output 1 is programmed on bipolar 5 V. A sine wave is generated and the calculated samples are placed in the DMA buffer.

In this program the D/A 1 FIFO half empty bit is programmed to initiate a DMA cycle in the demand mode data transfer.

The DMA done bit is programmed to generate an interrupt to restart the DMA transfer and load new samples in the DAC FIFO.

A wave counter is displayed on the screen during the running of the program.

## ***WIODMA***

## ***GUI (MFC) Example***

---

Example on how to use two DMA channels at the same time.

The program shows how to program the PCI4520 to read samples from an analog input through the first DMA channel and generate a waveform through the second DMA channel at the same time.

This example uses the first installed PCI4520.

The pacer clock is programmed to perform sampling at the rate of 5 kHz. The DMA channel 0 is programmed in demand mode, requesting DMA transfer when the A/D FIFO is half full.

The DMA done bit is programmed to generate an interrupt to restart the DMA transfer and read new samples from the DMA buffer.

The analog output 1 is programmed on bipolar 5 V. A sine wave is generated and the calculated samples are placed in the DMA channel 1 buffer.

In this program the D/A 1 FIFO half empty bit is programmed to initiate a DMA cycle in the demand mode data transfer.

The DMA done bit is programmed to generate an interrupt to restart the DMA transfer and load new samples in the DAC FIFO.

Wave counters are displayed on the screen during the running of the program.

---

**WINTRPTS****GUI (MFC) Example**

---

Example program to demonstrate the using of the interrupt handler routine.

The board is programmed to sample the analog input 1 with the pacer clock.

On A/D conversion, the board generates an interrupt.

The samples are read and displayed on the screen after an interrupt occurs.

The example uses the first installed PCI4520.

---

**WRANDOM****GUI (MFC) Example**

---

Example on doing multiple-channel Analog/Digital conversions using the Channel Gain Table.

This example uses the first installed PCI4520.

The Channel-Gain Table is programmed to sample AIN1-AIN16 channels in 5Volts bipolar, single-ended mode. A/D converter is programmed for pacer clock triggered conversions. When pressing the Start button, an A/D conversion is initiated and the samples are displayed on the screen as Voltage.

There is a slider in the window to change the pacer clock frequency.

The AOUT1 and AOUT2 are programmed to generate an output voltage of the 90% of full scale input values. The gain is selectable in the program between 1-128. The AOUT values are changed according to the gain, because in case of higher gain the input range is changing.

---

**WAMLTSCAN****GUI (MFC) Example**

---

Sample program that demonstrates how to use the pacer clock, about counter and channel gain table to scan groups of channels, multiple times.

This example uses the first installed PCI4520.

This program will show how to use the about counter and the pacer clock features to do MULTI-SCAN sampling. The board will be set up to sample a set of channels, every time the board receives a trigger from the on board 8254.

While the measurement is running the acquired data on the A/D channels is displayed on the screen.

## **WAMLTBRST**

## **GUI (MFC) Example**

---

Sample program that demonstrates how to use the pacer clock, burst clock, about counter and channel gain table to scan groups of channels, multiple times.

This example uses the first installed PCI4520.

This program will show how to use the about counter, burst clock and the pacer clock features to do MULTI-BURST sampling. The board will be set up to sample a set of channels, every time the board receives a trigger from the on board 8254.

While the measurement is running the acquired data on the A/D channels is displayed on the screen.

## **WBURSTN**

## **GUI (MFC) Example**

---

Sample program that demonstrates how to perform an analog to digital conversion on multiple channels using the channel gain table. Burst mode means that all the channels in the table are sampled once for each trigger. The time between channels is set by the burst clock. If the burst clock is set to the highest rate, this mode simulates simultaneous sampling.

This example uses the first installed PCI4520.

Sample program that uses the Burst Clock, Pacer clock and the channel gain table to acquire data on several channels. The burst clock will start the conversions and the burst clock will be triggered by the pacer clock.

While the measurement is running the acquired data on the A/D channels is displayed on the screen.

## **WSOFTTRIG**

## **GUI (MFC) Example**

---

Example on doing simple Analog/Digital conversions.

This example program does an A/D conversion when a button is pressed.

This example uses the first installed PCI4520.

Channel-Gain Latch is programmed for AIN1, 5Volts bipolar, single-ended. A/D converter is programmed for software triggered conversions. When pressing the Start button, an A/D conversion is initiated and the sample is displayed on screen as Voltage.

## **WTIMERS**

## **GUI (MFC) Example**

---

Example on programming the User Timer/Counters.

This example counts seconds real-time.

This example uses the first installed PCI4520.

User Timer/Counter 0 and 1 are cascaded. Counter 0 is programmed for 200 Hertz. When pressing the Start button a Windows timer is programmed for 200 msec to provide the main program loop. In every iteration User Timer/Counter 1 count is read back and converted to seconds.

### ***SPM\_DM***

### ***GUI (MFC) Example***

---

This example program demonstrates how can be used the DM7520 DataModule with the RTD's SPM6020 DSP board to sample analog inputs with the analog board and pass the samples to the DSP board to process them.

To run this example an installed DSP board is required in the system.

The detailed description of the example can be found in the documentation of the DSP board.

### ***SPM\_DM\_RealTime***

### ***GUI (MFC) Example***

---

This example program demonstrates how can be used the DM7520 DataModule with the RTD's SPM6020 DSP board to make real-time data processing.

To run this example an installed DSP board is required in the system.

The detailed description of the example can be found in the documentation of the DSP board.

### ***SPM\_DM\_Serial***

### ***GUI (MFC) Example***

---

This example program demonstrates the using of the McBSP serial port. This port can be used to connect the DataModule with a DSP board and make high speed data transfer between them. The program starts sampling of the analog input channel 1 and sends the samples to the McBSP port. If you have an SPM6020 board connected to the system, a DSP program will be downloaded to the DSP. This program simple sends back the samples to the D/A converter of the DataModule through the McBSP port. The D/A channel 1 / 2 can be selected with the Digital Input Data Marker bit 0.

### ***DRVR\_DEMO***

### ***GUI (MFC) Example***

---

This is a complex example on using the various driver functions to build a multifunctional program.

This example uses the first installed PCI4520.

### ***HIGHSPEED***

### ***GUI (MFC) Example***

---

This program demonstrates how can you make a high-speed data acquisition for a long time. In this program the speed is 128 kByte/s.

The program allocates a large memory buffer (2 Mbytes) for the temporary storage of the samples. This buffer could be as large as your system memory allows for you.

The CGT will be programmed to sample 8 channels. The conversion initiated by the burst clock. The burst clock starts on pacer clock. The samples will be transferred from the A/D FIFO to the DMA buffer with demand mode DMA. On DMA done interrupt in the ISR routine the samples will be copied to the memory buffer.

If the memory buffer is full, the data acquisition will be stopped and the content of the buffer is transferred to the hard disk.

The speed of the DAQ rate in this program depends on your computer's speed. The program is using CPU resources to transfer the samples from the DMA buffer to the memory buffer in the DMA done interrupt routine.

This example uses the first installed PCI4520.

## **CONTROL**

## **GUI (MFC) Example**

---

This program demonstrates how can you perform with the PCI4520/DM7520 board a real-time control.

The program is a simple "stereo volume control" application. The channel gain table is programmed to sample two analog channel with burst clock. The rate of burst clock is programmed to 1 MHz and the CGT is started with pacer clock. The pacer clock rate is variable with using of a scrollbar control.

Two analog outputs are defined. The sampled analog inputs are transferred to the DMA buffer with demand mode DMA started on A/D FIFO half full. On DMA done interrupt an ISR is called. Here can be done the data processing on the analog samples. In this example we simple multiply the analog values with a variable gain and the new values are transferred to the D/A FIFO.

As the D/A update source is the CGT (D/A is updated when a new CGT row is processed), the analog input and the analog output will be synchronized and they are jitter-free.

However, there is a phase shift between the real analog input and output due the programmatic data transfer from the DMA buffer to the D/A FIFO. This delay depends on the pacer clock rate.

This example uses the first installed PCI4520.

## **MLSSA**

## **GUI (MFC) Example**

---

Sample system analyzer application that demonstrates various features of the board.

The program generates a special signal called MLS, outputs it to the linear system to be measured, and simultaneously samples the response of the system. Then computes the impulse response and the transfer function of the system using the measure and response signal and display them.

Theoretical basics:

The basic idea of the measurement is the use of maximum-length sequences (MLS). The MLS is a periodic binary sequence generated by a shift register with feedback. The meaning of the naming is that this sequence is the longest in the set of the sequences generated by an n bit shift register, so its length is  $2^n - 1$ . (This example program uses 13 bit shift register, and the MLS is 8191 samples long.) Because the auto-correlation of the MLS is a Dirac-impulse, the impulse response of the measured system can be computed as the cross correlation of the measure signal (MLS) and the response signal. Thus, the sampling frequency chosen for a measurement cannot be too high, because the length of the MLS signal in time will be too short, and the impulse response won't fit in this time interval. Note, that the program uses a fast algorithm in computing the cross correlation based on a method called fast Hadamard-transform, then the transfer function is computed with the Fast Fourier Transform.

(more details in Douglas D. Rife - John Vanderkooy: Transfer-Function Measurement with Maximum-Length Sequences in Journal of the Audio Engineering Society, Volume 37, Number 6, 1989 June)

**Implementation:**

This example uses the first installed PCI4520.

The measurement begins with a gain calibration to use optimally the range of the A/D converter in the real measurement. One period of the MLS is outputted by the ADC, and sampled by the DAC1 with 1x gain. After this the gain is adjusted using the maximum of the response. One more measurement is performed, and if the maximum of the response with the new gain is greater than the half of the A/D range, the real measurement begins.

Both DMA channels are programmed to work in chained demand mode. The DMA channel 0 transfers the data of the DAC, the other channel the ADC. The synchronization of the D/A and A/D converting is done by setting the SyncBus 0 source to the pacer clock, and setting the D/A and A/D source to the SyncBus 0.

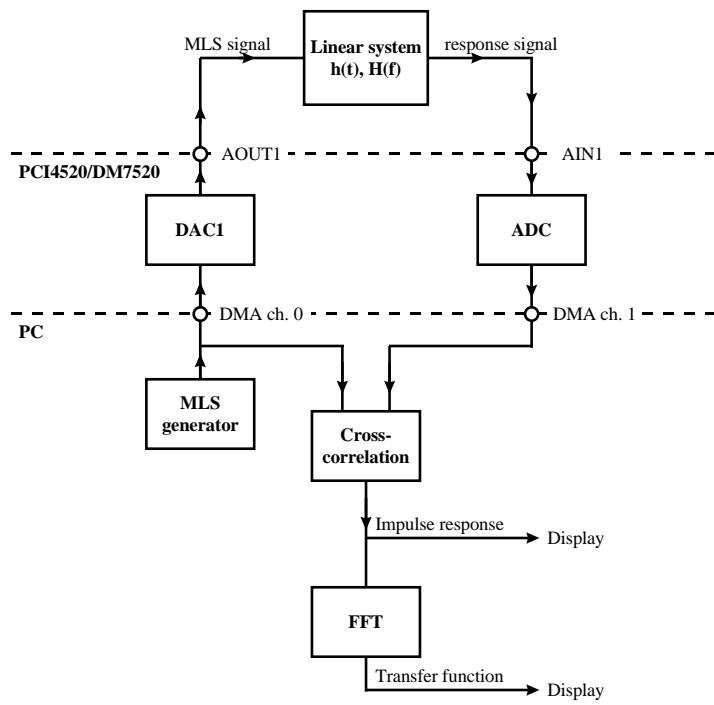
The measurement is executed one or more times (can be set in the Setup), and the average of the results is computed.

**Using the program:**

The system to be measured must be connected between AOUT1 (pin 21) and AIN1 (pin 1).

The program window consists of the display area, the message box and the buttons. You can start the measurement with the Start button. The format of the display can be chosen with the radio buttons on the bottom of the window. When the result is displayed, you can query the coordinates of any points on the diagrams by pressing the left mouse button in the display area. By pressing the Setup button, you can set the conditions of the measurement (sampling frequency, output level, number of measurements to the average) and the attributes of the display (time and frequency range). You can adjust the precision of the results to an optimum by modifying the output level and the number of measurements.

Block diagram of the measurement:



## Limited Warranty

Real Time Devices USA, 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 REAL TIME DEVICES USA, INC. This warranty is limited to the original purchaser of product and is not transferable.

During the one year warranty period, REAL TIME DEVICES USA 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 REAL TIME DEVICES USA. All replaced parts and products become the property of REAL TIME DEVICES USA. 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 REAL TIME DEVICES USA, "acts of God" or other contingencies beyond the control of REAL TIME DEVICES USA), OR AS A RESULT OF SERVICE OR MODIFICATION BY ANYONE OTHER THAN REAL TIME DEVICES USA. 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 REAL TIME DEVICES USA EXPRESSLY DISCLAIMS ALL WARRANTIES NOT STATED HEREIN. ALL IMPLIED WARRANTIES, INCLUDING IMPLIED WARRANTIES FOR MERCHANTABILITY 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 REAL TIME DEVICES USA 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.

Real Time Devices USA, Inc.

P.O. Box 906

103 Innovation Blvd.

State College PA 16803

USA

Our website: [www.rtdusa.com](http://www.rtdusa.com)