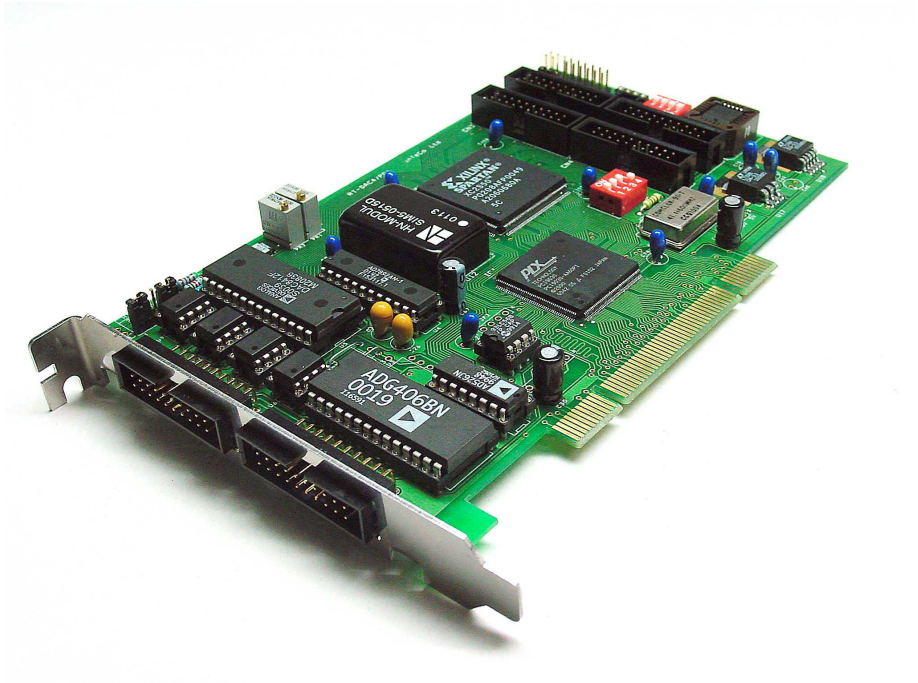
RT-DAC4/PCI

Multi I/O Board

Board version 1.11



User's Manual

Kraków 2013

Table of contents

NOTES

1. GENERAL INFORMATION

1.1 Introduction

The RT-DAC4/PCI is a multifunction analog and digital timing I/O board dedicated to real-time data acquisition and control in the Windows 95/98/NT/2000 environment. The board uses a PCI bus and supports real-time operations without introducing latencies caused by the Windows default timing system. The board contains a Xilinx® FPGA chip that can be reprogrammed to introduce a new functionality of digital inputs/outputs without any hardware modification.

The default configuration of the FPGA chip accepts signals from incremental encoders and generates PWM outputs, typical for mechatronic control applications.

1.2 Specification

Analog section

Analog Inputs

Channels:	16 single-ended, multiplexed
Resolution:	12 bit
Input ranges:	±10V, programmable gain (x1, x2, x4, x8, x16)
Conversion time:	1.6µs
Trigger:	software, hardware
Reference voltage:	on-board

Analog Outputs

Channels:	4
Resolution:	12 bit
Output range:	0V÷+10V, -10V÷0V, ±10V
Settling time:	6µs (to 0.01%)
Reference voltage:	on-board

Digital section* (version 1.11)

Digital Input/ Output

Channels:	32 bi-directional, direction setting
Direction:	bi-directional, direction is individual software programmable
Input voltage:	V_{IH} = 2.0V÷3.6V, V_{IL} = - 0.5V÷0.8V
Output voltage:	V_{OH} = 2.4V (min), V_{OL} = 0.4V (max)
Output current:	2mA÷24mA per channel
Standard:	LVTTL

Digital Timer/Counter

16 bit counter:	2 channels, counts external signal
32 bit timer:	2 channels, counts internal clock signal (frequency depends on a current version of the board)

Digital Signal Generator

Channels:	2
Resolution:	28 bits of the H state; 28-bits of the L state
Max frequency:	20 MHz
Duty cycle	Software configurable

PWM Outputs

Channels:	4
Resolution:	8/12 bits (software selected)
Base Frequency:	programmable, depends on a current version of the board

Incremental encoders

Channels:	4
Output:	32 bit counter

Interrupts

PCI interrupt:	INTA#
Interrupt sources:	2 external inputs, software, timer, arbitrary change of state
Pulse width generating COS interrupt	25ns min.

PCI features

Supports PCI v2.2 compliant
Supports both version of PCI slots (3.3V and 5V) and can work with all computers equipped with PCI buses.

*Digital section with the default FPGA chip configuration

Software support:

The included Testing Software allows initial tests of the board under Windows 95/98/NT/2000. Advanced users can access all the functions of the board using the standard programming languages supported with optionally included *DLL* library.
The board is compatible with the RT-CON real-time development toolbox distributed by INTECO. The toolbox integrates input/output RT-DAC4/PCI board capabilities to MATLAB/Simulink functionality and creates an ideal design and application environment.

Fig. 1 presents the block diagram of the RT-DAC4/PCI board. The board contains the analog input multiplexer connected to 16 single-ended analog input channels. Voltage ranges are defined from –10V to 10V (bipolar). The RT-DAC4/PCI board includes the software-programmable gain amplifier that can be configured for the voltage gains 1,2,4,8,16 to accommodate low-level and high level analoge input signals.

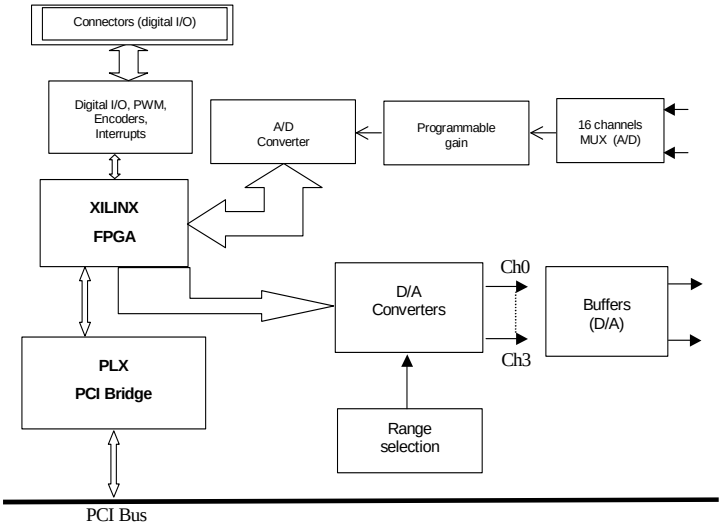


Fig. 1. General block diagram of the RT-DAC4/PCI board

The board is equipped with 12-bit successive approximation A/D converters that give the 5 mV resolution within input range ±10V. Finer resolution can be achieved by the gain definition using gain. The A/D conversion time of the RT-DAC4/PCI board is equal to 1.6 μs. This means that Troughoutput Rate is greater than 500kSPS.

The board contains four 12-bits D/A converters connected to four analog output channels. All channels can be hardware configured to operate in the unipolar or bipolar mode. Each analog output channel can sink up to 10 mA.

There are 32 digital I/O lines at the RT-DAC4/PCI board. Digital I/O lines are LVTTL compatible. The direction of each digital I/O can be configured separately.

The default configuration of the RT-DAC4/PCI includes four PWM outputs and four input channels of the incremental encoders. The PWM outputs and encoders inputs turn the PC into a digital controller to be used in control of manipulators, servo systems, etc.

Two digital signal generators can be applied to generate signals with an arbitrary duty cycle.

PCI interrupts are born in the interrupt generation block. They come out from different sources: software, timer, two external signals and change-of-state at thirty two digital inputs.

Reprogramming the XILINX FPGA chip can change functions of the digital section of the board. The interior of the FPGA chip is presented in Fig.2. Beside digital I/O and interrupts, there are the FPGA logic which implements A/D and D/A control functions and the board controller. The FPGA chip is connected to a hardware oscillator, which gives a high counting resolution. Different versions of the board operate at different frequencies (typically 40 MHz).

The board is equipped with six 20-pin ribbon cable connectors. The detailed block diagram of the RT-DAC4/PCI board (including connectors) is given in sections 3 and 4.

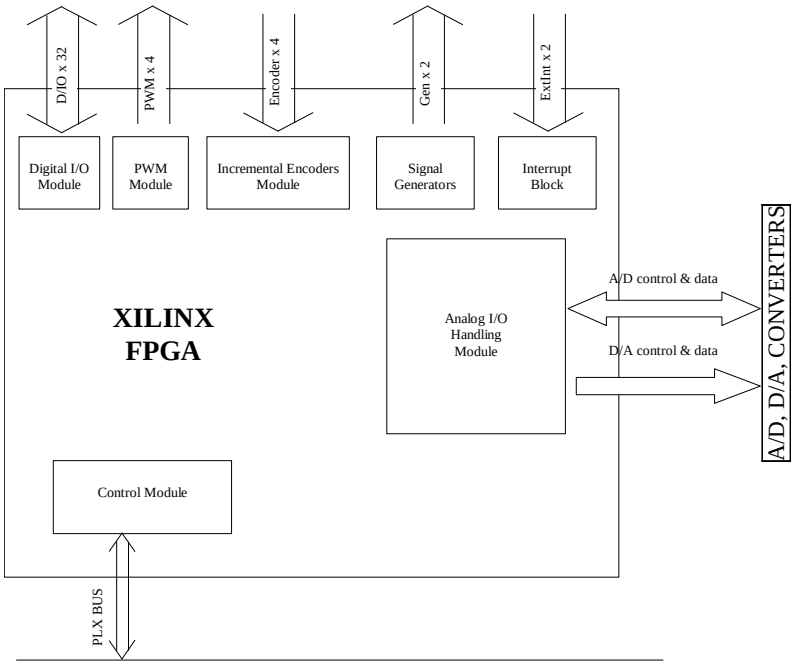


Fig. 2. The default structure of the Xilinx FPGA chip

The next section contains the basic information necessary to install and test the board. The information and specification how to reprogram XILINX FPGA is not included in this guide. Please, relate to *RT-DAC4/PCI FPGA Programming Guide* distributed by INTECO separately.

2. BOARD INSTALLATION

The RT-DAC4/PCI setup contains:

- RT-DAC4/PCI board,
- RT-DAC4/PCI Testing Software,
- 2 ribbon cables,
- *RT-DAC4/PCI User's Manual* - this manual,
- terminal wiring board (optional).

The RT-DAC4/PCI board contains sensitive electric components which can be easily damaged by static electricity, therefore the board should be kept in its original anti-static packing until it is installed. During installation the board should be handled carefully only by the edges to avoid static electric discharge.

To install the board:

- turn off the computer and remove the cover,
- find an empty 32-bit PCI slot and remove the metal bracket,
- check jumper settings for your configuration (see section 4),
- insert the RT-DAC4/PCI board into the expansion slot firmly and evenly, then secure the board with the bracket screw and install the cover,
- turn on the computer,
- install driver for the board (see section 3 or CD:\DRIVER\readme*.txt),
- install the Testing Software or
- install RT-CON package,
- test the board.

3. DRIVER INSTALLATION

The driver for RTDAC4/PCI board has to be installed because the board is of the PCI type. The way of driver installation depends on operating system. The user with administrator privileges must install the drivers for Windows XP and Windows 7.

3.1 Installation

Administrator privileges are required for driver installation.

- Start Windows XP/7
- System detects *new PCI device*
- Select Next, then Display a list....
- Select Other Devices, then Next
- Select Have a disk, then Browse
- Select path CD:\driver\WinW7x86\RTDAC4_PCI9030.inf, then *Open*
- Select *OK* and *Next*, and *Next*
- Select *Finish*
  If Windows propose to restart the computer select *Yes*.

4. JUMPER SETTINGS

The RT-DAC4/PCI board is equipped with two jumpers for configuration setting. The board layout is shown in Fig. 3.

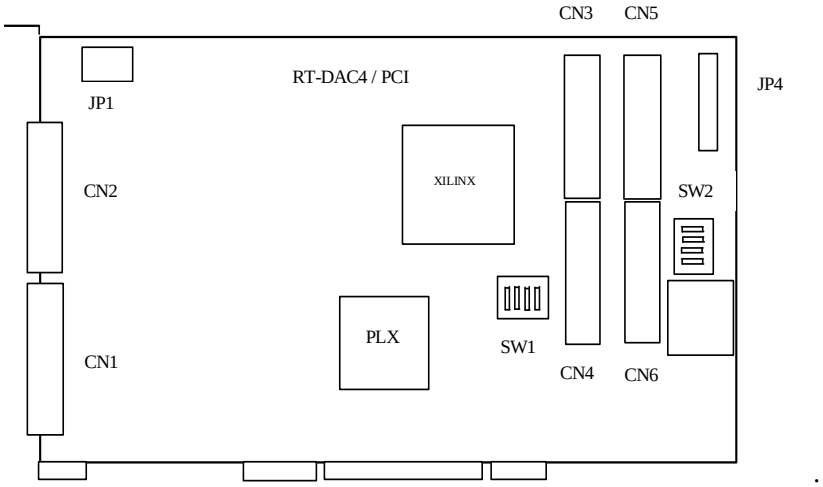


Fig. 3. The layout of the RT-DAC4/PCI board

4.1 Output range selection – JP1

RT-DAC4/PCI is equipped with four analogue output channels. The user can set the output range for all channels. The JP1 jumper supports all channels (Table 1).

Table 1. D/A range selection

Pin	0 ÷10 V	-10 ÷ 10V	-10 ÷ 0V
1-2 closed 3-4 closed	X		
1-2 closed 4-5 closed		X	
2-3 closed 4-5 closed			X
All others N/A			

4.2 Dip switches for FPGA programming

The SW1 and SW2 switches allow a user to choose a programming method of XILINX FPGA. **If you are using the default XILINX FPGA configuration do not change the settings of these switches.** The default settings are defined as follows.

Dipswitch SW1	Settings	Dipswitch SW2	Settings
SW1 – 1	close	SW2 – 1	Open
SW1 – 2	close	SW2 – 2	Open
SW1 – 3	close	SW2 – 3	Open
SW1 – 4	open	SW2 – 4	Open

5. CONNECTOR PIN ASSIGNMENTS

RT-DAC4/PCI is equipped with two 20-pin I/O connectors CN1, CN2 accessible from the rear bracket, and four 20-pin I/O connectors CN3, CN4, CN5 and CN6 on the board (see Fig. 3). Fig. 4 shows the pin assignment of each connector.

CN1
A/I – Analog Input
GND A - Analog Ground

A/I 0	1	2		GND A
A/I 1	3	4		GND A
A/I 2	5	6		GND A
A/I 3	7	8		GND A
A/I 4	9	10		GND A
A/I 5	11	12		GND A
A/I 6	13	14		GND A
A/I 7	15	16		GND A
A/I 8	17	18		GND A
A/I 9	19	20		GND A

CN2
A/I – Analog Input
A/O – Analog Output

A/I 10	1	2		GND A
A/I 11	3	4		GND A
A/I 12	5	6		GND A
A/I 13	7	8		GND A
A/I 14	9	10		GND A
A/I 15	11	12		GND A
A/O 0	13	14		GND A
A/O 1	15	16		GND A
A/O 2	17	18		GND A
A/O 3	19	20		GND A

CN3
D I/O Digital Input/Output
GND - Digital ground

D I/O 0	1	2		D I/O 1
D I/O 2	3	4		D I/O 3
D I/O 4	5	6		D I/O 5
D I/O 6	7	8		D I/O 7
D I/O 8	9	10		D I/O 9
D I/O 10	11	12		D I/O 11
D I/O 12	13	14		D I/O 13
D I/O 14	15	16		D I/O 15
GND	17	18		GND
+5V	19	20		+3.3V

CN4
D I/O Digital Input/Output
GND - Digital ground

D I/O 16	1	2		D I/O 17
D I/O 18	3	4		D I/O 19
D I/O 20	5	6		D I/O 21
D I/O 22	7	8		D I/O 23
D I/O 24	9	10		D I/O 25
D I/O 26	11	12		D I/O 27
D I/O 28	13	14		D I/O 29
D I/O 30	15	16		D I/O 31
GND	17	18		GND
+5V	19	20		+3.3V

CN5
PWM and COUNTER outputs

PWM0	1	2		
PWM1	3	4		
PWM2	5	6		
PWM3	7	8		
GEN0	9	10		GND
GEN1	11	12		All pins
ExtInt0	13	14		
ExtInt1	15	16		
CNT0	17	18		
CNT1	19	20		

CN6
Encoder inputs

ENC 0 - A	1	2		
ENC 0 - B	3	4		
ENC 1 – A	5	6		
ENC 1 – B	7	8		
ENC 2 - A	9	10		GND
ENC 2 - B	11	12		All pins
ENC 3 – A	13	14		
ENC 3 – B	15	16		
Reserved	17	18		
Reserved	19	20		

Fig. 4. RT-DAC4/PCI I/O connectors

The further information included in section 5 can be used by advanced users.

6. REGISTER STRUCTURE AND FORMAT

Table 2 shows the offset of each registers and the control words relative to the I/O base address of the RT-DAC4/PCI board. **This information is necessary to configure RT-DAC4/PCI board to serve ones own system.**

It is assumed that each offset in Table 2 reserves 4 bytes in the I/O address space. Some functions do not require all of 32 information bits. In such a case the most significant bits **have to be neglected**. The functionality description given below corresponds only to meaningful bits. The convention applied is that the most significant bit is denoted D31 and the least significant bit D0.

The RT-DAC4/PCI access functions are defined in the *rtdacpci.*c file. This file contains the API macro definitions (see Table 2) and C API functions referred to the description of the board functions.

Notice: The *rtdacpci.c* file is accessible after installation of the RT-CON toolbox or installation of *Testing Software*.

In the examples of the next sections it is assumed that the *BaseAddr* variable is the base I/O space address of the board. The RT-DAC4/PCI board is located in the I/O address space of the microprocessor. The offset can vary because the PCI bus controller determines its value automatically. The current base address of the board can be detected by *Testing Software* or by the MATLAB *mex_baseaddress* function.

Table 2. I/O address space map

Byte offset		**Description**
Decimal	**Hexadecimal**	**API macro definition**
Version management		
0	00	XILINX bitstream version (read only) *RTDACPCI_BITSTREAM_VERSION*
4	04	Number of digital signal generators, counters, timers, PWM and encoders (read only) *RTDACPCI_NO_OF_CHANNELS*
8	08	Application name (read only) *RTDACPCI_APPLICATION_NAME*
Counter/timer		
32	20	Counter index *RTDACPCI_COUNTER_IDX*
36	24	Load a new counter value *RTDACPCI_COUNTER_LOAD*
40	28	Counter value *RTDACPCI_COUNTER*
44	2C	Timer index *RTDACPCI_TIMER_IDX*
48	30	Load a new timer value *RTDACPCI_TIMER_LOAD*
52	34	Timer value *RTDACPCI_TIMER*
Digital I/O		
64	40	Digital I/O directions of 16 least significant digital I/O lines *RTDACPCI_DIG_IO_DIR_L*
68	44	Digital I/O directions of 16 most significant digital I/O lines *RTDACPCI_DIG_IO_DIR_H*
72	48	Digital input/output values of 16 least significant digital I/O lines *RTDACPCI_DIG_IO_VALUE_L*
76	4C	Digital input/output values of 16 most significant digital I/O lines *RTDACPCI_DIG_IO_VALUE_H*

PWM		
96	60	PWM index *RTDACPCI_PWM_IDX*
100	64	PWM mode *RTDACPCI_PWM_MODE*
104	68	PWM prescaler *RTDACPCI_PWM_PRESCALER*
108	6C	PWM channel width *RTDACPCI_PWM_WIDTH*
Digital signal generators		
112	70	Generator index *RTDACPCI_WAVE_IDX*
116	74	Duration of the H state *RTDACPCI_WAVE_L*
120	78	Duration of the L state *RTDACPCI_WAVE_H*
Encoders		
128	80	Encoder index *RTDACPCI_ENCODER_IDX*
132	84	Reset encoder counters *RTDACPCI_RESET*
136	88	Encoder counter *RTDACPCI_ENCODER*
A/D conversion		
160	A0	A/D channel and gain *RTDACPCI_AD_MUX*
164	A4	A/D control signals (ADSTART, CS, RD) *RTDACPCI_AD_CONTROL*
168	A8	Conversion result *RTDACPCI_AD_RESULT*
D/A conversion		
192	C0	D/A control signals (LDAC, CS, A1, A0, WR) *RTDACPCI_DA_CONTROL*
196	C4	D/A channel *RTDACPCI_DA*
Interrupts		
224	E0	Interrupt flags *RTDACPCI_INTR_FLAGS*
228	E4	Interrupt status *RTDACPCI_INTR_STATUS*
232	E8	Interrupt timer period *RTDACPCI_INTR_PERIOD*
236	EC	Change-of-state LSW mask *RTDACPCI_COS_MASK_L*
240	F0	Change-of-state MSW mask *RTDACPCI_COS_MASK_H*
244	F4	Change-of-state “before” value *RTDACPCI_COS_BEFORE*
248	F8	Change-of-state “after” value *RTDACPCI_COS_AFTER*

6.1 Before using a C API functions

To use C API functions in a user application the following statements have to be included at the beginning of the C-source file:

```
#define RTDAC_PCI_VERSION_API
#define RTDAC_PCI_COUNTER_API
#define RTDAC_PCI_TIMER_API
#define RTDAC_PCI_DIGITALDIRECTIONS_API
#define RTDAC_PCI_DIGITAL_IO_API
#define RTDAC_PCI_PWM_API
#define RTDAC_PCI_GENERATOR_API
#define RTDAC_PCI_ENCODER_API
#define RTDAC_PCI_AD_API
#define RTDAC_PCI_DA_API
#define RTDAC_PCI_INTR_API
#define RTDAC_PCI_FIFO_API
#define RTDAC_PCI_FREQM_API

// For PLC API I/O access functions
#define _inp(A)     ReadByte(A)
#define _inpw(A)    ReadWord(A)
#define _inpd(A)    ReadDWord(A)
#define _outp(A,B)  WriteByte(A,B)
#define _outpw(A,B) WriteWord(A,B)
#define _outpd(A,B) WriteDWord(A,B)

#include "rtdacpci.c"
```

The statements define the macro definitions required by the rtdacpci.c file. The API functions are implemented in the rtdacapi.c.

To be able to establish the communication with the RT-DAC/PCI board there are required functions to access the I/O address space of the board. The communication is performed by the functions from the PLX library (the PlxApi631.dll file).

The RT-DAC/PCI functions are implemented in the rtdacapi.dll library. For convenience also the rtdacapi.lib and rtdacapi.h files are available.

The names of the I/o access functions are:
ReadByte(address) - read a single byte from the address location
ReadWord(address) - read a single word from the address location
ReadDWord(address) - read a double word from the address location
WriteByte(address,value) - writes a single byte value to the address location
WriteWord(address,value) - writes a single word value to the address location
WriteDWord(address,value) - writes a double word value to the address location

The following Visual Studio projects are given as examples:

- RT-DAC PCI Test - VS 2010 project of the test program
- Common/RTDAC_PCI_API_PLX_IO - VS 2010 project to build the RTDACAPI.dll

6.2 Version management

The RT-DAC4/PCI board is equipped with XILINX FPGA. All functions of the board are implemented as the FPGA project (except the PCI communication functions). **The FPGA logic can be easily changed and tailored to the user requirements**. The bitstream version function (I/O address offset 0) enables one to distinguish different logic versions (Table 2). The I/O address offset equal to 4 allows one to read current number of digital signal generators, counter, timer, PWM and encoder channels.

For example, to read the number of available channels relating to: digital signal generators, counters, timers, PWM and encoders the following C-statements can be executed:

```
NoOfChans = _inpd( BaseAddr ) & 0xFFFFF; // read no of channels
NoOfGenerators = (NoOfChans >> 16) & 0xF;
NoOfCounters = (NoOfChans >> 12) & 0xF;
NoOfTimers = (NoOfChans >> 8) & 0xF;
NoOfPWM = (NoOfChans >> 4) & 0xF;
```

```
NoOfEncoders = NoOfChans & 0xF;
```

or the following API functions can be called:

```
NoOfGenerators = RtdacPCI_ReadNoOfGenerators( BaseAddr );
NoOfCounters = RtdacPCI_ReadNoOfCounters( BaseAddr );
NoOfTimers = RtdacPCI_ReadNoOfTimers( BaseAddr );
NoOfPWM = RtdacPCI_PWMNoOfChans( BaseAddr );
NoOfEncoders = RtdacPCI_ReadNoOfEncoders( BaseAddr ) ;
```

Version management functions

I/O space byte offset: 0

Function:	read version of FPGA bitstream.
Used bits:	D15-D0
C API function name:	*RtdacPCI_BitstreamVersion*

I/O space byte offset: 4

Function:	read number of: digital signal generators, counters, timers, PWM outputs and encoders.
Used bits:	D19-D16 – number of generators D15-D12 – number of counters D11-D8 – number of timers D7-D4 – number of PWM outputs D3-D0 – number of encoders
C API function name:	*RtdacPCI_ReadNoOfGenerators* *RtdacPCI_ReadNoOfCounters* *RtdacPCI_ReadNoOfTimers* *RtdacPCI_PWMNoOfChans* *RtdacPCI_ReadNoOfEncoders*

I/O space byte offset: 8

Function:	read application name. Returns four-characters logic name.
Used bits:	D31-D0
C API function name:	*RtdacPCI_AppName*

6.3 Counter/timer

RT-DAC4/PCI includes 32-bit timer and 16-bit channels counter. The channel timer counts pulses of the internal board clock. The frequency of the clock depends on the board version (40 MHz is the default value). The channel counter counts external pulses.

To access the appropriate timer or counter the channel must be selected by setting an appropriate value to the I/O offset. The offset equal to 32 is used for counters, the offset 44 is used for timers. The I/O offset equal to 36 is used to reset counters. The I/O offset equal to 48 is used to reset timers. The I/O offset 40 is used to read current counter value. The I/O offset 52 is used to read current timer value.

For example, to reset timer 0 and start counting of the internal clock impulses the following C-statements can be executed:

```
_outpd( BaseAddr + 44, 0 );     // select timer 0
_outpd( BaseAddr + 48, 1 ); // reset current timer
_outpd( BaseAddr + 48, 0 );    // return to counting mode
…
TmrValue = _inpd( BaseAddr + 52 ); // read current timer value
```

or the following API functions can be called:

```
RtdacPCI_ResetTimer( BaseAddr, 0, 1 ); // reset timer 0
RtdacPCI_ResetTimer( BaseAddr, 0, 0 ); // set timer 0 to counting mode
…….
TmrValue = RtdacPCI_ReadTimer( BaseAddr, 0 );
```

Counter/timer functions

I/O space byte offset: 32
- Function: read/write current counter number
- Used bits: D3-D0 – counter number
- C API function name: *RtdacPCI_ReadCounter*

I/O space byte offset: 36
- Function: reset the counter value.
- Used bits: D0 –when equal to ‘0’ the counter counts input impulses. When equal to ‘1’ the current counter is set to zero.
- C API function name: *RtdacPCI_ReadCounter*

I/O space byte offset: 40
- Function: reads the current counter value.
- Used bits: D16-D0 – the counter value.
- C API function name: *RtdacPCI_ReadCounter*

I/O space byte offset: 44
- Function: read/write the current timer number.
- Used bits: D3-D0 – the timer number.
- C API function name: *RtdacPCI_ReadTimer*

I/O space byte offset: 48
- Function: reset the timer value.
- Used bits: D0 –when equal to ‘0’ the timer counts the input clock impulses. When equal to ‘1’ the current timer is set to zero.
- C API function name: *RtdacPCI_ReadTimer*

I/O space byte offset: 52
- Function: reads the current timer value.
- Used bits: D31-D0 – the timer value.
- C API function name: *RtdacPCI_ReadTimer*

6.4 Digital I/O

The RT-DAC4/PCI board contains 32 digital input/output lines. The digital I/O lines are connected to the D I/O0 – D I/O31 pins of the CN3 and CN4 connectors.

The direction of each line can be set separately. The direction is determined by the value written to the I/O offsets 64 and 68. The I/O offset 64 determines the direction of D I/O0 to D I/O15. The I/O offset 68 determines the direction of D I/O16 to D I/O31. When a bit of the direction control word is set to '0' the appropriate digital line is configured as output. The value '1' sets the digital line as input.

The input lines can be read and output lines can be set by the I/O offset 72 and 76. The I/O offset 72 determines the state of D I/O0 to D I/O15. The I/O offset 68 determines the state of D I/O16 to D I/O31.

For example, to set lines D I/O0-D I/O15 as outputs and to set D I/O16 to D I/O31 as inputs, set all output lines to logic state '1' and read all 16 inputs, the following C-statements can be executed:

```
_outpd( BaseAddr + 64, 0x0000 );            // set directions of D I/O0 to D I/O15
_outpd( BaseAddr + 68, 0xFFFF );   // set directions of D I/O16 to D I/O 31
_outpd( BaseAddr + 76, 0xFFFF );   // set all outputs to '1'
DigInp = _inpd( BaseAddr + 72 ) & 0xFFFF; // read digital inputs
```

or the following API functions can be called:

```
RtdacPCI_WriteDigIOConfig( BaseAddr, 0xFFFF0000 );
RtdacPCI_WriteDigIO( BaseAddr, 0xFFFF0000 );
DigInp = RtdacPCI_ReadDigIO( BaseAddr );
```

Digital I/O functions

I/O space byte offset: 64
- Function: read/write the directions of the D I/O0 to D I/O15 digital I/O signals.
- Used bits: D15-D0 – define the directions of the 16 least significant digital I/O lines. Each bit defines the direction of the appropriate digital line. When set to '0' the line works as output, when set to '1' the line is an input.
- C API function name: *RtdacPCI_WriteDigIOConfig*, *RtdacPCI_ReadDigIOConfig*

I/O space byte offset: 68
- Function: read/write the directions of the D I/O16 to D I/O31 digital I/O signals.
- Used bits: D15-D0 – define the directions of 16 most significant digital I/O lines. Each bit defines the direction of the appropriate digital line. When set to '0' the line works as output, when set to '1' the line is input.
- C API function name: *RtdacPCI_WriteDigIOConfig*, *RtdacPCI_ReadDigIOConfig*

I/O space byte offset: 72
- Function: read/write the state of the D I/O0 to D I/O15 digital I/O signals.
- Used bits: D15-D0 – reads the state of 16 least significant digital inputs or sets the state of digital outputs.
- C API function name: *RtdacPCI_WriteDig*, *RtdacPCI_ReadDig*

I/O space byte offset: 76
- Function: read/write the state of the D I/O16 to D I/O31 digital I/O signals.
- Used bits: D15-D0 – reads the state of 16 most significant digital inputs or sets the state of digital outputs.
- C API function name: *RtdacPCI_WriteDig*, *RtdacPCI_ReadDig*

6.5 PWM

The RT-DAC4/PCI board includes four output PWM channels denoted PWM0 to PWM3. The base PWM period and the period of the "H" state of each channel are selected separately (see Fig. 5). The counters of the base PWM period and the "H" state period can work in the 12 or 8-bit mode. The 8-bit mode allows PWM to operate in high speed and the 12-bit mode allows PWM to achieve high accuracy of the output.

In the 12-bit mode a single PWM period contains 4095 impulses of the output prescaler frequency. The time of logic output '1' is set by a number from 0 to 4095. In the 8-bit mode a PWM period contains 255 impulses of the output prescaler frequency. The time of logic '1' is set by a number from 0 to 255. The 8-bit mode is used for high speed. The 12-bit mode gives a high accuracy.

The input (base) frequency of the PWM channels is set by default to: 40MHz, 30MHz or 20MHz. It depends on board version. This frequency is divided by the counter (called prescaler), which creates the PWM base period and the period of the "H" state. The valid prescaler value is a number taken from the range [0 - 65535].

The frequency of the PWM wave is calculated by the formula:

$$f_{PWM} = \frac{f_{base}}{(prescaler + 1) * 255} \quad \text{for 8-bit mode}$$

$$f_{PWM} = \frac{f_{base}}{(prescaler + 1) * 4095} \quad \text{for 12-bit mode}$$

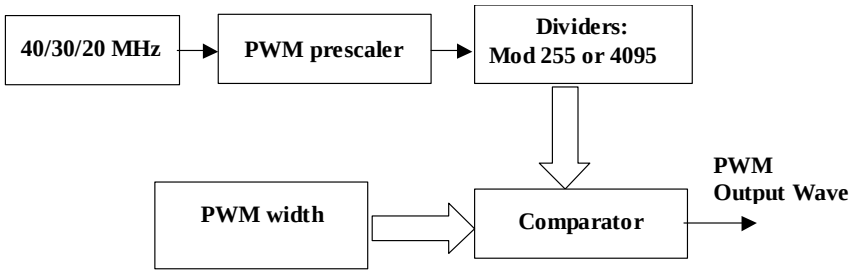


Fig. 5. Block diagram of the PWM generator

To set the correct wave of a PWM channel the number of channel have to be written to the I/O space offset 96. The I/O offset equal to 100 sets the PWM mode, the I/O offset equal to 104 sets the prescaler value and the I/O offset equal to 108 determines the PWM duty cycle.

For example, to set: PWM2 to 12-bit mode, prescaler to 1200 and duty cycle to 50% the following C-statements can be executed:

```
_outpd( BaseAddr + 96, 2 );      // select PWM2 as the current PWM channel
_outpd( BaseAddr + 100, 1 );     // select the 12-bit mode of the current channel
_outpd( BaseAddr + 104, 1200 );  // set prescaler to 1200
_outpd( BaseAddr + 108, 2047 );  // set duty cycle to 50 % (2047 is 50% of 4095)
```

or the following API function can be called:

```
RtdacPCI_PWMWrite( BaseAddr, 2, 1, 1200, 2047 );
```

PWM functions

I/O space byte offset: 96

Function:	read/write the current PWM channel.
Used bits:	D3-D0 – PWM channel number.
C API function name:	*RtdacPCI_PWMChannel* *RtdacPCI_PWMWrite*

I/O space byte offset: 100

Function:	read/write the mode of the current PWM channel.
Used bits:	D0 – '0' defines the 8-bit mode, '1' defines the 12-bit PWM mode.
C API function name:	*RtdacPCI_PWMMode*

RtdacPCI_PWMWrite

I/O space byte offset: 104
Function: read/write the prescaler for the current PWM channel
Used bits: D15-D0 – the divider value.
C API function name: *RtdacPCI_PWMPrescaler*
RtdacPCI_PWMWrite

I/O space byte offset: 108
Function: read/write the width value of the current PWM channel.
Used bits: D7-D0 – for the 8-bit mode
D11-D0 – for the 12-bit mode.
C API function name: *RtdacPCI_PWMWidth*
RtdacPCI_PWMWrite

6.6 Digital signal generators

The RT-DAC4/PCI board includes two outputs of digital signal generators denoted as GEN0 and GEN1. The output waves are generated on the basis of the default (40 MHz frequency) wave signal. The software sets the durations of the L and H states of the generated waves independently.

To set the correct output wave the channel number must be selected by setting an appropriate value to the I/O space offset equal to 112. The I/O offset equal to 116 sets the duration of the L state and the I/O offset equal to 120 sets the duration of the H state of the generated output.

For example, to set the GEN1 to generate the wave kept at the L state 100 time periods of the base wave long and kept at the H state 300 time periods long the following C-statements have to be executed:

```
_outpd( BaseAddr + 112, 1 );     // select GEN1 channel
_outpd( BaseAddr + 116, 100 );    // set duration of the L state
_outpd( BaseAddr + 120, 300 );// set duration of the H state
```

or the following API function can be called:

```
RtdacPCI_WriteGeneratorL ( BaseAddr, 1, 100 );
RtdacPCI_WriteGeneratorH ( BaseAddr, 1, 300 );
```

If the base wave frequency on the board is 40MHz then GEN1 generates the 100kHz wave with the duty cycle equal to 75%.

Digital signal generator functions

I/O space byte offset: 112
Function: read/write the current wave generator channel.
Used bits: D3-D0 – wave generator channel number.
C API function name: *RtdacPCI_ ReadGeneratorL*
RtdacPCI_ ReadGeneratorH
RtdacPCI_ WriteGeneratorL
RtdacPCI_ WriteGeneratorH

I/O space byte offset: 116
Function: read/write the duration of the L state.
Used bits: D27-D0 – defines the number of periods of the base wave when the output is L.
C API function name: *RtdacPCI_ ReadGeneratorL*
RtdacPCI_ WriteGeneratorL

I/O space byte offset: 120
Function: read/write the duration of the H state.
Used bits: D27-D0 – defines the number of periods of the base wave when the output is H.
C API function name: *RtdacPCI_ ReadGeneratorH*
RtdacPCI_ WriteGeneratorH

6.7 Encoders

The RT-DAC4/PCI board includes four 32-bit incremental encoder input channels denoted as ENC0-ENC3. Each channel counts the changes of two input waves. The initial value of each encoder counter can be set to zero in a programmable way.

The appropriate encoder input channel has to be selected by writing a given value to the I/O offset equal to128. The I/O space offset equal to 132 is used to reset encoder counters. The bits of the 132 I/O offset reset the corresponding encoders. A single write action to the 132 offset can reset all the encoder counters. The I/O space offset equal to 136 is used to read the current encoder counter value.

For example: to reset encoder ENC2, start counting and read data the following C-statements can be executed:

```
_outpd( BaseAddr + 132, 4 );    // set the third bit – only ENC2 is reset l
_outpd( BaseAddr + 132, 0 );    // set reset flags to zero for all encoders –
                                // normal operation of encoder counters
……..
_ outpd( BaseAddr + 128, 2 );   // select ENC2 as the current encoder
Counter = _inpd( BaseAddr + 136 ); // read current counter value
```

or the following API functions can be called:

```
RtdacPCI_ResetEncoder( BaseAddr, 2, 1 );
RtdacPCI_ResetEncoder( BaseAddr, 2, 0 );
……..
Counter = RtdacPCI_ReadEncoder( BaseAddr, 2 );
```

Encoder functions

I/O space byte offset: 128

Function:	read/write the current encoder channel.
Used bits:	D3-D0 – the encoder channel number.
C API function name:	*RtdacPCI_ReadEncoder*

I/O space byte offset: 132

Function:	reset encoder counters.
Used bits:	D3 – D0 – four bits responsible for resetting four encoder counters. If a bit is set, the corresponding encoder counter is set to zero.
C API function name:	*RtdacPCI_EncoderReset*

I/O space byte offset: 136

Function:	reads the current encoder counter value.
Used bits:	D31-D0 – for the 8-bit mode
C API function name:	*RtdacPCI_ReadEncoder*

6.8 A/D Conversion

The RT-DAC4/PCI is equipped with 16 multiplexed analog inputs. The output of the analog multiplexer is connected to the input of the digital programmable analog amplifier. The 160 I/O offset is used to select an input channel and an amplifier gain. The Table 3 and the Table 4 show the setting for the D6÷D4 and D3÷D0 bits.

Table 3. Gain setting

D6	D5	D4	Amp. Gain
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	X	X	16

Table 4. Channel selection

D3	D2	D1	D0	Channel no.
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

The RT-DAC4/PCI board is equipped with the parallel 12-bit A/D converter. The A/D converter control is performed by the I/O offset 164. The *end-of-conversion* (EOC) flag is accessible at offset 164. The A/D conversion results are available at I/O offset 168.

The A/D control word (offset 164) contains three signals: RD# (D2), CS# (D1) and CONVST# (D0). The low state of the CS# signal is used to enable the A/D converter. When the CS# signal is high the converter is disabled. The RD# signal is used to read the last A/D conversion results. The A/D conversion starts if the rising edge of the CONVST signal occurs. The conversion can be started when the CS signal is high. The EOC signal, available to read at I/O offset 164 (D0 when read offset 164), detects the termination of the A/D conversion. When the EOC is equal to '0' the conversion results are ready to be read.

For example, to start A/D conversion of the analog input 3, set the gain to 1 and read the conversion results, the following C-statements can be executed:

```
_outpd( BaseAddr+160, 0x03 ); // set gain and channel number
_outpd( BaseAddr+164, 0x6 ); // set CONVST to 0 (D0 set to 0)
_outpd( BaseAddr+164, 0x7 ); // set CONVST to 1
_outpd( BaseAddr+164, 0x5 ); // set CS to 0 (D1 set to 0)
_outpd(BaseAddr+164, 0x1 ); // set CS to 0 and RD to 0 (D1 and D2 set to 0)
while( (_inpd(BaseAddr+164) & 1 ) ~= 0 ) ; // wait for EOC equal to 0
ADResult = _inpd( BaseAddr+168); // read A/D conversion result
ADResult = ADResult & 0xFFF; // mask only 12 bits
_outpd(BaseAddr+164), 0x7 ); // set CONVST, RD, and CS to 1
```

or the following API function can be called:

```
ADResult = RtdacPCI_AD( BaseAddr, 3, 1 );
```

A/D converter functions

I/O space byte offset: 160
Function: select the channel and gain.
Used bits: D3-D0 – the analog input channel (see Table 4)
D6-D4 – the analog amplifier gain (see Table 3)
C API function name: *RtdacPCI_AD*

I/O space byte offset: 164
Function: sets the A/D control or reads the A/D end of conversion signal.
Used bits: when write:
D0 – the CONVST# signal,
D1 – the CS# signal,
D2 – the RD# signal,
when read:
D0 – the EOC signal.
C API function name: *RtdacPCI_AD*

I/O space byte offset: 168
Function: reads the A/D conversion results.
Used bits: D11-D0 – the result of the last A/D conversion
C API function name: *RtdacPCI_AD*

6.9 D/A Conversion

The RT-DAC4/PCI board is equipped with four-channel 12-bit D/A converter. The converter is controlled by the A1, A0, LDAC#, R/W# and CS# signals. The signals are controlled by the I/O offset 192 (A1 at D4, A0 at D3, LDAC# at D2, R/W# at D1 and CS# at D0). The A1 and A0 signals select a appropriate output channel. The R/W# signal is used to store data in the D/A converter buffer. The rising edge of the LDAC# signal moves data from the D/A converter buffer to the D/A converter. The input data for the D/A converter are written to the I/O offset 196. The CS# signal enables the D/A converter. The data movement controlled by the LDAC# signal can be performed even if CS# is disabled.

To set a new output voltage equivalent to the digit 1500 at the D/A channel 2, the following C-statements can be executed:

```
_outpd( BaseAddr+192, 0x14 ); // set 'A1A0' to '10' (2), LDAC to 1,
                              // R/W to 0 and CS to 0
_outpd( BaseAddr+196), 1500 ); // set data to the D/A buffer

// Update D/A converter – the new analog voltage will appear
_outpd( BaseAddr+192, 0x11 ); // set 'A1A0' to '10' (2), LDAC to 0,
                              // R/W to 0 and CS to 1
_outpd( BaseAddr+192, 0x15 ); // set 'A1A0' to '10' (2), LDAC to 1,
                              // R/W to 0 and CS to 1
```

or the following API function can be called:

```
RtdacPCI_DA( BaseAddr, 2, 1500 );
```

D/A converter functions

I/O space byte offset: 192
Function: writes the D/A converter control signals.
Used bits: D4-D3 – the A1 and A0 signals,
D2 – the LDAC# signal,
D1 – the R/W# signal,
D0 – the CS# signal.

C API function name: *RtdacPCI_DA*

I/O space byte offset: 196
Function: sets D/A data.
Used bits: D11-D0 – data for the D/A converter.
C API function name: *RtdacPCI_DA*

6.10 Interrupts

The RT-DAC4/PCI board is able to generate the INTA# PCI interrupt. The parameters of the interrupts are set in two I/O locations. The first one is the internal PLX9030 I/O space. The second is RT-DAC4/PCI logic I/O space. The base address of the internal PLX9030 I/O space is returned by the *mex_pcibar1* MATLAB function (as well as by the call to the *BoardLocationEx* basic API function). The base address of the RT-DAC4/PCI logic I/O space is returned by the *mex_baseaddress* MATLAB function (or by the call to the *BoardLocation* basic API function).

The internal PLX9030 I/O space is responsible for the parameters of the PCI bridge, including the PCI interrupt generation. The RT-DAC4/PCI logic generates interrupt signal which triggers the PLX9030 chip. This signal is denoted as the local interrupt. If local interrupt is active the PLX9030 can generate the INTA# PCI interrupt.

The parameters of the PCI INTA# interrupt are set by the location, which offset in the internal PLX9030 I/O space is equal to 4C hex. The name of this location is INTCSR. The bits of the INTCSR have the following meaning:

0 – **Local interrupt enable**. Value of 1 indicates enabled interrupt generation by the RT-DAC4/PCI logic. Value of 0 indicates disabled,
1 – **Local interrupt polarity**. Value of 1 indicates active high. Value of 0 indicates active high,
2 – **Local interrupt status**. Value of 1 indicates interrupt active. Value of 0 indicates interrupt not active,
3-5 – reserved,
6 – **PCI interrupt enable**. Value of 1 enables PCI interrupt,
7 **– Software interrupt**. Value of 1 generates software interrupt,
8 - **Local interrupt select enable**. Value of 1 indicates enabled edge triggerable interrupt. Value of 0 indicates enabled level trigerrable interrupt. Operates only in high polarity mode,
9 – reserved,
10 – **Local edge trigerrable interrupt clear**. Writing 1 to this bit clears local interrupt,
11-15 – reserved.

The interrupts can be level or edge triggered. The RT-DAC4/PCI logic uses only edge triggering. The *IntrInit* procedure from the board basic API enables local interrupt (bit 0 set to 1), sets high polarity mode (bit 1 set to 1), enables PCI interrupts (bit 6 set to 1) and enables edge local interrupt trigger (bit 8 set to 1). The *IntrResponse* procedure clears interrupt by setting the bit 10 to 1.

Each interrupt requests are stored in the RT-DAC4/PCI register. Only one of the requested local interrupts can generate the PCI interrupt. The PCI handling procedure must be able to distinguish which local interrupt source has generated the PCI interrupt and which interrupt requests are queued.

The I/O offset equal to 224 activates the interrupt sources and clears the interrupt requests. The I/O offset equal to 228 is used to read which local interrupt source has generated the current PCI interrupt and the queued interrupts. The I/O offset 232 defines the period of the interrupt timer. The timer is applied to periodically generation of interrupts. The period of the timer is defined in 25ns units. Be sure not to define too short interrupt period, because it may degrade the performance of the computer system. The I/O offsets 236 and 240 are used to define which digital inputs are considered when the change-of-state interrupt is generated. The I/O location 236 is responsible for digital signals from the CN3 connector and the location 240 is responsible for the CN4 signals. Only digital signals defined as input are applied to detect the COS. When the COS interrupt is generated the state of digital inputs before and after the interrupt generation moment can be read from the I/O locations 244 and 248 respectively.

Let us configure the interrupt block to generate interrupts from all available interrupt sources. The frequency of the timer interrupt will be set to 1kHz. The following steps have to be performed:

- clear of all interrupt sources,

- thread creation. To implement the interrupt handling procedure it is recommended to create a thread. The thread operates as interrupt handling procedure,
- enabling required interrupt sources. Setting interrupt timer period if timer interrupt source enabled.

To simplify the programming the basic API functions will be applied (see section 7 for details). The following statements perform all actions required by interrupt services:

```
HANDLE hThread;
ULONG IDThread;
char Str[200];
int ret;
int BusNo, SlotNo, VendorID, DeviceID, BaseAddress;
int RTDAC_BaseAddress;

if( NoOfDetectedBoards() < 1 )
  return;
ret = BoardLocation( 1, &BusNo, &SlotNo,
                        &VendorID, &DeviceID, &BaseAddress );
if( ret != 0 )
  return;
RTDAC_BaseAddress = BaseAddress;

// clear all interrupt requests
WriteByte( BaseAddress+224, 0 );

// Call API interrupt initialisation procedure
if( (ret=IntrInit( 1 )) < 0 ) {
  ErrorAction( );
}

// enable local interrupt generation and enable local timer interrupt
WriteByte( BaseAddress+224, 0x88 );
// set timer period to 1kHz – 40000 impulses of 25ns period
WriteDWord( BaseAddress+232, 40000 );

hThread = CreateThread(NULL, // no security attributes
        0,                          // use default stack size
        (LPTHREAD_START_ROUTINE) ThreadFunc, // thread function
        NULL,                       // no thread function argument
        0,                          // use default creation flags
        &IDThread);                 // returns thread identifier
if (hThread == NULL)
  ErrorAction( );
```

The body of the thread is presented below.

```
DWORD WINAPI ThreadFunc(VOID)
{
  HANDLE eventHandle;
  DWORD val;
  int IntrSource;
  int iAux;

  // Set priority of the current process and thread - optional
  SetPriorityClass( GetCurrentProcess(),
                    REALTIME_PRIORITY_CLASS );
  SetThreadPriority( GetCurrentThread(),
                     THREAD_PRIORITY_TIME_CRITICAL );
  // Clear all interrupt requests and set active interrupt sources
  iAux = ReadWord(BaseAddress + 0xE0);
  WriteWord( BaseAddress + 0xE0, 0 );
  WriteWord( BaseAddress + 0xE0, iAux );

  for(;;) {
    if( IntrAttach( BoardNo + 1, &eventHandle ) < 0 ) AfxMessageBox("Interrupt Attach Error");//ErrorAction( );
    __try {
      val = WaitForSingleObject( eventHandle, 10000 );
    }
    __except(EXCEPTION_EXECUTE_HANDLER) {
      ;
    }
```

```
    if( (val == WAIT_TIMEOUT) || (val==WAIT_FAILED) )
        AfxMessageBox( "__try Timeout" );
    else {
      if( IntrResponse(  BoardNo + 1 ) < 0 )
        AfxMessageBox("Intrrupt Response Error");;
        ResetEvent(eventHandle);
        IntrSource = ReadWord( BaseAddress + 0xE4 ) & 0x1F;
        if( IntrSource & 0x10 ) { // Software interrupt
            AfxMessageBox("Software Interrupt Source");
            iAux = ReadWord(BaseAddress + 0xE0);
            iAux &= 0xFFEF;
            WriteWord(BaseAddress + 0xE0, iAux);  // Clear software intr. request
            iAux |= 0x0010;
            WriteWord(BaseAddress + 0xE0, iAux);  // Reinit software interrupt
        }
        if( IntrSource & 0x08 ) { // Timer interrupt
            AfxMessageBox("Timer Interrupt Source" );
            iAux = ReadWord(BaseAddress + 0xE0);
            iAux &= 0xFFF7;
            WriteWord(BaseAddress + 0xE0, iAux);  // Clear timer intr. request
            iAux |= 0x0008;
            WriteWord(BaseAddress + 0xE0, iAux);  // Reinit software interrupt
        }
        if( IntrSource & 0x04 ) { // COS interrupt
            AfxMessageBox ("COS Interrupt Source" );
            iAux = ReadWord(BaseAddress + 0xE0);
            iAux &= 0xFFFB;
            WriteWord(BaseAddress + 0xE0, iAux);  // Clear COS intr. request
            iAux |= 0x0004;
            WriteWord(BaseAddress + 0xE0, iAux);  // Reinit software interrupt
        }
        if( IntrSource & 0x04 ) { // Ext1
            AfxMessageBox ("External 1 Interrupt Source" );
            iAux = ReadWord(BaseAddress + 0xE0);
            iAux &= 0xFFFD;
            WriteWord(BaseAddress + 0xE0, iAux);  // Clear Ext1 intr. request
            iAux |= 0x0002;
            WriteWord(BaseAddress + 0xE0, iAux);  // Reinit software interrupt
        }
        if( IntrSource & 0x04 ) { // Ext0
            AfxMessageBox ("External 0 Interrupt Source" );
            iAux = ReadWord(BaseAddress + 0xE0);
            iAux &= 0xFFFE;
            WriteWord(BaseAddress + 0xE0, iAux);  // Clear Ext0 intr. request
            iAux |= 0x0001;
            WriteWord(BaseAddress + 0xE0, iAux);  // Reinit software interrupt
        }
    }
   }
   return 0;
}
```

Interrupt functions

I/O space byte offset: 224

Function:	reads/writes interrupt flags.
Used bits:	D7 – value of 1 enables local interrupt generation. Value of 0 disables, D6 – not used D5 – value of 1 generates local software interrupt, D4 – value of 1 enables local software interrupt generation. If local software interrupt occurred previously setting this bit to 0 clears the interrupt request, D3 – value of 1 enables local timer interrupt generation. If local timer interrupt occurred previously setting this bit to 0 clears the interrupt request, D2 – value of 1 enables local COS interrupt generation. If local software COS occurred previously setting this bit to 0 clears the interrupt request, D1 – value of 1 enables local interrupt generation requested by the external ExtInt1 signal. If local ExtInt1 interrupt occurred previously setting this bit to 0 clears the interrupt request, D0 – value of 1 enables local interrupt generation requested by the external ExtInt0 signal. If local ExtInt0 interrupt occurred previously setting this bit to 0 clears the interrupt request.

C API function name: *RtdacPCI_ReadIntrFlags*
RtdacPCI_WriteIntrFlags

I/O space byte offset: 228
Function: reads interrupt status.
Used bits: D11 – value of 1 indicates that local interrupt has occurred,
D10 – not used
D9 – value of 1 indicates that the software interrupt request is queued,
D8 – value of 1 indicates that the timer interrupt request is queued,
D7 – value of 1 indicates that the COS interrupt request is queued,
D6 – value of 1 indicates that the ExtInt1 interrupt request is queued,
D5 – value of 1 indicates that the ExtInt0 interrupt request is queued,
D4 – value of 1 indicates that the current PCI interrupt is caused by the software interrupt source,
D3 – value of 1 indicates that the current PCI interrupt is caused by the timer interrupt source,
D2 – value of 1 indicates that the current PCI interrupt is caused by the COS interrupt source,
D1 – value of 1 indicates that the current PCI interrupt is caused by the ExtInt1 interrupt source,
D0 – value of 1 indicates that the current PCI interrupt is caused by the ExtInt0 interrupt source,
C API function name: *RtdacPCI_ ReadIntrStatus*

I/O space byte offset: 232
Function: read/write period of the interrupt timer.
Used bits: D27-D0 – define the period of the interrupt timer. The period is defined in units equal to 25ns.
C API function name: *RtdacPCI_WriteIntrPeriod*
RtdacPCI_ReadIntrPeriod

I/O space byte offset: 236
Function: read/write COS mask.
Used bits: D15-D0 – defines which inputs from the CN3 connector are used to detect change-of-state and to generate the interrupt. Value of 1 means that the respective signal is applied to generate the COS interrupt. Value of 0 means that the respective signal does not influence the COS block.
C API function name: *RtdacPCI_ReadIntrCOSMask*
RtdacPCI_WriteIntrCOSMask

I/O space byte offset: 240
Function: read/write COS mask.
Used bits: D15-D0 – defines which inputs from the CN4 connector are used to detect change-of-state and to generate the interrupt. Value of 1 means that the respective signal is applied to generate the COS interrupt. Value of 0 means that the respective signal does not influence the COS block.
C API function name: *RtdacPCI_ReadIntrCOSMask*
RtdacPCI_WriteIntrCOSMask

I/O space byte offset: 244
Function: read COS before state.
Used bits: D31-D0 – reads which state at the CN3 and the CN4 connectors was active immediately before the COS generation. The D31-D16 bits store the state of the CN4 and the D15-D0 bits store the state of the CN3 connector.
C API function name: *RtdacPCI_ReadIntrCOSBefore*

I/O space byte offset: 248
Function: read COS after state.

Used bits:	D31-D0 – reads which state at the CN3 and the CN4 connectors was active immediately after the COS generation. The D31-D16 bits store the state of the CN4 and the D15-D0 bits store the state of the CN3 connector.
C API function name:	*RtdacPCI_ReadIntrCOSAfter*

7. LOW-LEVEL API FUNCTIONS

It was developed the DLL library which contain functions used to detect the RT-DAC4/PCI location and to allow the access to the board resources. The library is distributed as three files: *RTDACAPI.DLL*, *RTDACAPI.LIB* and *RTDACAPI.H*. The first file contains the functions. The second file is used during static DLL linking. The last file contains the declarations of exported functions.

The API DLL allows to detect the location of the RT-DAC4/PCI boards available in the system and allows access to the I/O address space. **The access to the I/O address space is prohibited in the Windows NT/2000/XP operating systems and the API DLL allows to exceed these limitations.** A special kernel-mode device driver called by the API DLL functions performs operations which are forbidden in user-mode applications.

The API interface contains the following functions (see the *RTDACAPI.H* file):

int NoOfDetectedBoards(void);

Returns the number of RT-DAC4/PCI boards detected in the system.

```
int BoardLocation(  int BoardIdx,
                    int *BusNo, int *SlotNo,
                    int *VendorID,
                    int *DeviceID,
                    int *BaseAddress );
```

Determines the bus number, slot number, vendor ID, device ID and base address of the RT-DAC4/PCI board given by the *BoardIdx* input argument. The *BoardIdx* can vary from 1 to the number of detected boards returned by the *NoOfDetectedBoards* function. The function returns 0 value if succeed or –1 if failed.

```
int BoardLocationEx(  int BoardIdx,
                      int *BusNo, int *SlotNo,
                      int *VendorID,
                      int *DeviceID,
                      int *BaseAddress,
                      int *PCIBAR1 );
```

Determines the bus number, slot number, vendor ID, device ID, base address and the PCIBAR1 location of the RT-DAC4/PCI board given by the *BoardIdx* input argument. The *BoardIdx* can vary from 1 to the number of detected boards returned by the *NoOfDetectedBoards* function. The function returns 0 value if succeed or –1 if failed.

```
int WriteByte( int port, int value );
unsigned short WriteWord( int port, unsigned int value );
unsigned long WriteDWord( int port, unsigned int value );
```

Output the *value* byte (*WriteByte*), word(*WriteWord*), or double word (*WriteDWord*) at the *port* port. The functions return the data output.

```
int ReadByte( int port );
unsigned int ReadWord( int port );
unsigned long ReadDWord( int port );
```

Input a byte (*ReadByte*), a word (*ReadWord*), or a double word (*ReadDWord*) from the *port* port.

int IntrInit(int BoardIdx)

Interrupt initialisation function. The *BoardIdx* can vary from 1 to the number of detected boards returned by the *NoOfDetectedBoards* function and defines which board initialises interrupts. The function enables interrupt generation and defines that the interrupt trigger is rising edge of the LINTi1 local bus signal. The function returns 0 value if succeed or –1 otherwise.

*int IntrAttach(int BoardIdx, HANDLE *eventHandle)*

This function attaches the object *eventHandle* to the interrupt handling procedure. The *BoardIdx* can vary from 1 to the number of detected boards returned by the *NoOfDetectedBoards* function and defines which board is considered. The function enables interrupt generation and defines that the interrupt trigger is rising edge of the LINTi1 local bus signal. The function returns 0 value if succeed r a negative value otherwise.

int IntrResponse(int BoardIdx)

This function clears the internal PLX9030 interrupt request flag. It is called by the user interrupt handling procedure. The *BoardIdx* can vary from 1 to the number of detected boards returned by the *NoOfDetectedBoards* function and defines which board is considered. The function returns 0 value if succeed or -1 value otherwise.

IntrClose(int BoardIdx)

This function terminates interrupt generation by the board. The *BoardIdx* can vary from 1 to the number of detected boards returned by the *NoOfDetectedBoards* function and defines which board is considered. The function returns 0 value if succeed or -1 otherwise.

8. XILINX FPGA CHIP PROGRAMMING

The RT-DAC4/PCI board is equipped with a XILINX FPGA chip. The user can reprogram the logic design of the FPGA chip. A new logic can perform quite different functions. For example the user can build a new logic, which can perform:

- 32 PWM outputs, or
- hardware implemented digital filters, or
- hardware implemented FFT algorithm, or
- fast data acquisition from A/D converters, or
- fast analog signal generators using D/A converters, or
- data encryption and decryption, or
- finite state machines, or
- microprocessor cores and
- much more.

The design of a new logic requires a more extended RT-DAC4/PCI board description. The description is included in the "*RT-DAC4/PCI FPGA Programming Guide*" distributed separately.