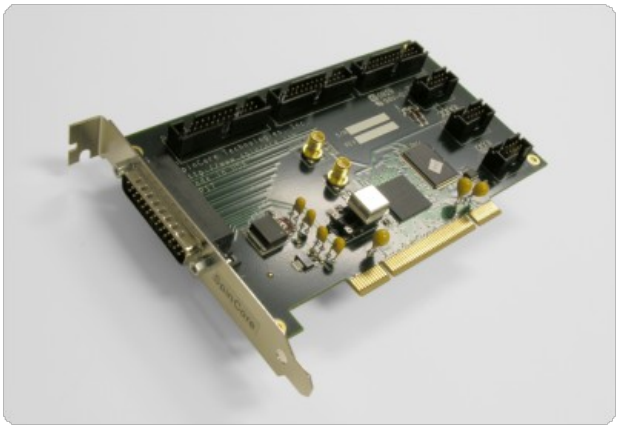
PulseBlaster - Programmable Pulse Generator

(PCI Boards SP2 and SP17)

SP2 Model: PB24-100-32k

SP17 Models: PB12-100-4k and PB24-100-4k

Owner's Manual



SpinCore Technologies, Inc.
http://www.spincore.com

Congratulations and *thank you* for choosing a design from SpinCore Technologies, Inc.

We appreciate your business!

At SpinCore we aim to fully support the needs of our customers. If you are in need of assistance, please contact us and we will strive to provide the necessary support.

Table of Contents

I. Introduction

Product Overview

The PulseBlaster™ device is an intelligent pulse/word/pattern generation unit producing 24 precisely timed, individually controlled digital output signals.

The intelligence of the PulseBlaster timing processor comes from an embedded microprogrammed control core (uPC). The PulseBlaster processor is able to execute instructions that allow it to control program flow. This means that the PulseBlaster processor understands Operational Control Codes, OpCodes, and will execute them much the same way as a general-purpose microprocessor does. The PulseBlaster's microcontroller is different from the general-purpose microprocessor in that it does not contain an arithmetic logic unit (ALU) and is, therefore, incapable of doing mathematical or logical calculations. However, a unique and distinguishing feature of the PulseBlaster processor is that the execution time of instructions is user programmable. This feature makes the PulseBlaster capable of executing complex output timing patterns at greatly varying update rates, ranging from nanoseconds to years, with a constant setting accuracy of just one clock period (e.g., a 10 ns setting accuracy at a 100 MHz clock frequency).

Board Architecture

Block Diagram

Figure *1* presents the general architecture of the PulseBlaster system. The major building blocks are the SRAM memory (both internal and external to the processor), the microcontroller (uPC), the integrated bus controller (IBC), the counter, and the output buffers. The entire logic design, excluding output buffers, is contained on a single silicon chip, making it a System-on-a-Chip design. User control to the system is provided through the IBC over the peripheral component interconnect (PCI) bus.

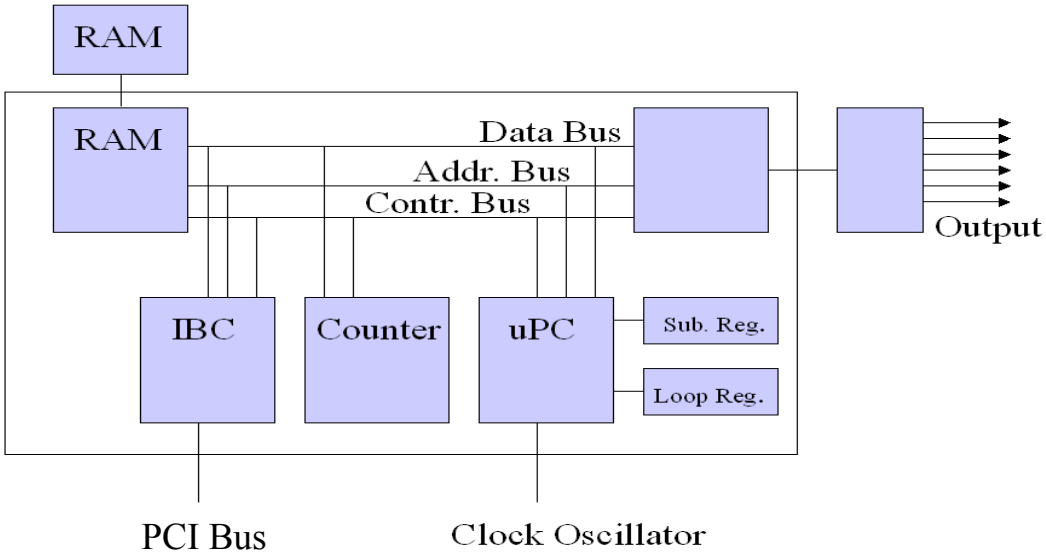


Figure 1: PulseBlaster board architecture. The clock oscillator signal is derived from an on-chip PLL circuit typically using a 50 MHz on-board reference clock.

Key Features

Output Signals

The PulseBlaster PB24 PCI board allows for 24 digital output signal lines. Sixteen output lines are routed to a DB25 bracket-mounted connector. The remaining 8 output lines are routed to an insulation displacement connector (IDC). The 24 individually controlled digital output lines comply with the transistor-transistor logic (TTL) levels' standard, and are capable of delivering ±25 mA per bit/channel. If more output current is necessary, the individual bits/channels can be driven in parallel.

Timing Characteristics

The PulseBlaster's timing controller accepts an internal (on-board) crystal oscillator up to 100 MHz. The innovative architecture of the timing controller allows the processing of either simple timed instructions (with delays of up to 2^{32} or 4,294,967,296 clock cycles), or double-length timed instructions (up to 2^{52} clock cycles long – nearly 2 years with a 100 MHz clock!). Regardless of the type of instruction, the timing resolution remains constant for any delay – just one clock period (e.g., 10 ns at 100 MHz).

The core-timing controller has a minimum delay cycle of five clock periods for the PB12-100-4k (SP17) and PB24-100-4k (SP17) and a minimum delay cycle of nine clock periods for PB24-100-32k (SP2). For a 100 MHz clock, this translates to a 50.0 ns pulse/delay/update for the PB12-100-4k and PB24-100-4k models and a 90.0 ns pulse/delay/update for the PB24-100-32k model.

Instruction Set

The PulseBlaster's design features a set of commands for highly flexible program flow control. The micro-programmed controller allows for programs to include branches, subroutines, and loops at up to 8 nested levels – all this to assist the user in creating dense pulse programs that cycle through repetitious events, especially useful in numerous multidimensional spectroscopy and imaging applications.

External Triggering

The PulseBlaster can be triggered and/or reset externally via dedicated hardware lines. These lines combine the convenience of triggering (e.g., in cardiac gating) with the safety of the "stop/reset" line. The required control signals are active-low (or short to ground) for SP2 boards, with additional active-high pins available on SP17 models.

Status Readback

The status of the pulse program can be read in hardware or software. The hardware status output signals consist of five IDC connector pins labeled "Status". The same output can be read through software using C. See section IV (Connecting to the PulseBlaster Board, page 16) for more detail about the hardware lines and Appendix I (Controlling the PulseBlaster with SpinAPI, page 24) for more detail about the C function pb_read_status().

Summary

PB24 is a versatile, high-performance, programmable pulse/pattern TTL signal generator operating at speeds of 100 MHz (or more!) and capable of generating pulses/delays/intervals ranging from 50 ns to two years per instruction. It is connected via PCI port and can accommodate pulse

programs with highly flexible control commands of up to 32k (i.e.,32,768) program words (Model PB24-100-32k). Its high-current output logic bits are independently controlled with a voltage of 3.3 V.

Specifications

Pulse Parameters

- 24 individually controlled digital output lines (TTL levels, 3.3 V logical “one”)
- variable pulses/delays for every TTL line
- 25 mA output current per TTL line
- 50 ns shortest pulse/interval (at 100 MHz, Internal Memory Model, PB24-100-4k)
- 2 years longest pulse interval (at 100 MHz, with the use of the “Long Delay” instruction)
- 10 ns pulse/interval resolution (at 100 MHz)
- up to 32k pulse program memory words/instructions (Model PB24-100-32k)
- external triggering and reset – TTL levels

Pulse Program Control Flow

- loops, nested 8 levels deep
- 20 bit loop counters (max. 1,048,576 repetitions)
- subroutines, nested 8 levels deep
- wait for trigger – 8 clock cycle latency (80 ns at 100 MHz), adjustable to 40 seconds in duration
- 5 MHz max. re-triggering frequency (at 100 MHz clock frequency)

Note on Related Boards Compatible with this Manual

Much of the programming information provided in this manual is nearly universal to SpinCore's lines of boards. More complex boards such as the PulseBlasterESR, PulseBlaster-DDS, and RadioProcessor lines of boards still rely on the same PulseBlaster core for TTL pulse generation. Therefore, the basic example programs for the PulseBlaster will be able to produce the same results on any of the more complex boards. The exception is the PulseBlaster-DDS-II board which uses a 96-Bit or 124-Bit instruction word, depending on the firmware, instead of an 80-Bit instruction word and is currently not compatible with PulseBlaster methods of programming the board.

II. Installation

Installing the PulseBlaster

To install the board you must uninstall any previous versions of SpinAPI and complete the following:

1. Install the latest version of SpinAPI found at: http://www.spincore.com/support/spinapi/.
    - SpinAPI is a custom Application Programming Interface developed by SpinCore Technologies, Inc. for use with the PulseBlaster board. It can be utilized using C/C++ or graphically using the options in the next section below. The API will also install the necessary drivers.

2. There is an additional package available to download and install that offers example programs to run on your PulseBlaster. The installer for these examples can be found at http://spincore.com/CD/spinapi_examples/Installers/. However, if only some example programs are needed then they can be individually downloaded from http://spincore.com/CD/spinapi_examples/Individual%20Files/.

3. Shut down the computer, unplug the power cord, insert the PulseBlaster card into an available PCI slot and fasten the PC bracket securely with a screw.

4. Plug the power cord back in, turn on the computer and follow the installation prompts.

We recommend running the example programs when you first receive a PulseBlaster to verify that your device is functional. If the example programs were installed using the installer provided by SpinCore then the example programs for the PulseBlaster can be found in the Windows Start Menu at **"Start > Programs > SpinCore > Examples"**. This will open the Examples directory; the file PulseBlaster24 will hold all of the examples for the PulseBlaster. If the files were downloaded individually then the test programs will be found in the user defined directory.

III. Programming the PulseBlaster

There are several ways of programming the PulseBlaster board. In this section the PulseBlaster Interpreter, LabVIEW extensions, .NET GUI, MATLAB GUI, and C/C++ methods of programming will be introduced. In addition to these, the PulseBlaster is programmable using nearly any higher level programming software that lets you utilize a C language API package, in this case SpinCore's SpinAPI.

The PulseBlaster Interpreter

The PulseBlaster board is programmable via the *PulseBlaster Interpreter*, which is a programming utility provided by SpinCore for writing pulse programs. This easy-to-use editor allows you to create, edit, save and run your pulse sequence. Figure 2 below shows the PulseBlaster Interpreter being used with one of the example programs.

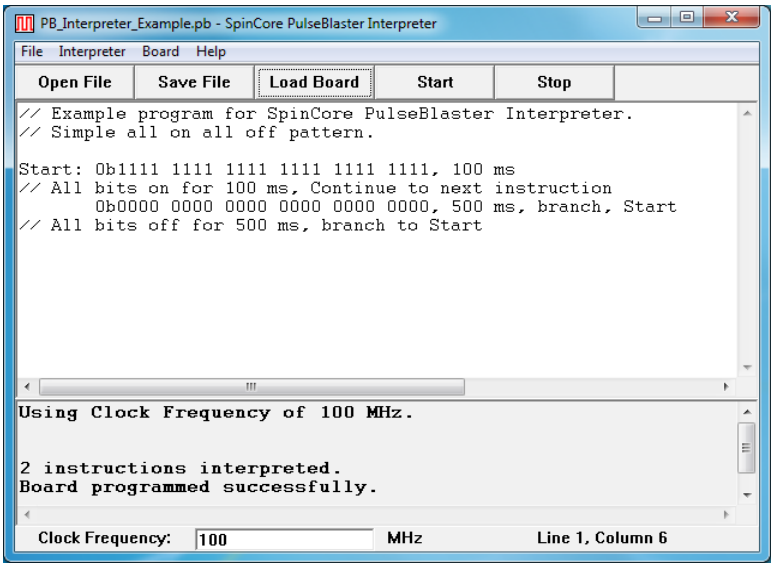


Figure 2: Graphical Interface of the PulseBlaster Interpreter. The example shown creates a pulse that toggles all TTL bits on for 100 ms, and all off for 500 ms.

The PulseBlaster Interpreter is available as part of the SpinCore Driver Suite and will be installed automatically during the setup process described above in Section II. Example programs, such as the one above, are installed to C:\SpinCore\PulseBlaster Interpreter\Examples by default. For convenience, a shortcut to the PulseBlaster Interpreter will be added to your desktop. For more information on programming using the PulseBlaster Interpreter, see the manual located at http://www.spincore.com/support/SPBI/Doc/

PulseBlaster.NET

PulseBlaster.NET is a graphical interface for creating pulse programs and loading them to the PulseBlaster board. PulseBlaster.NET currently provides the simplest interface possible to pulse control. Figure 3 shows an example instance of the program.

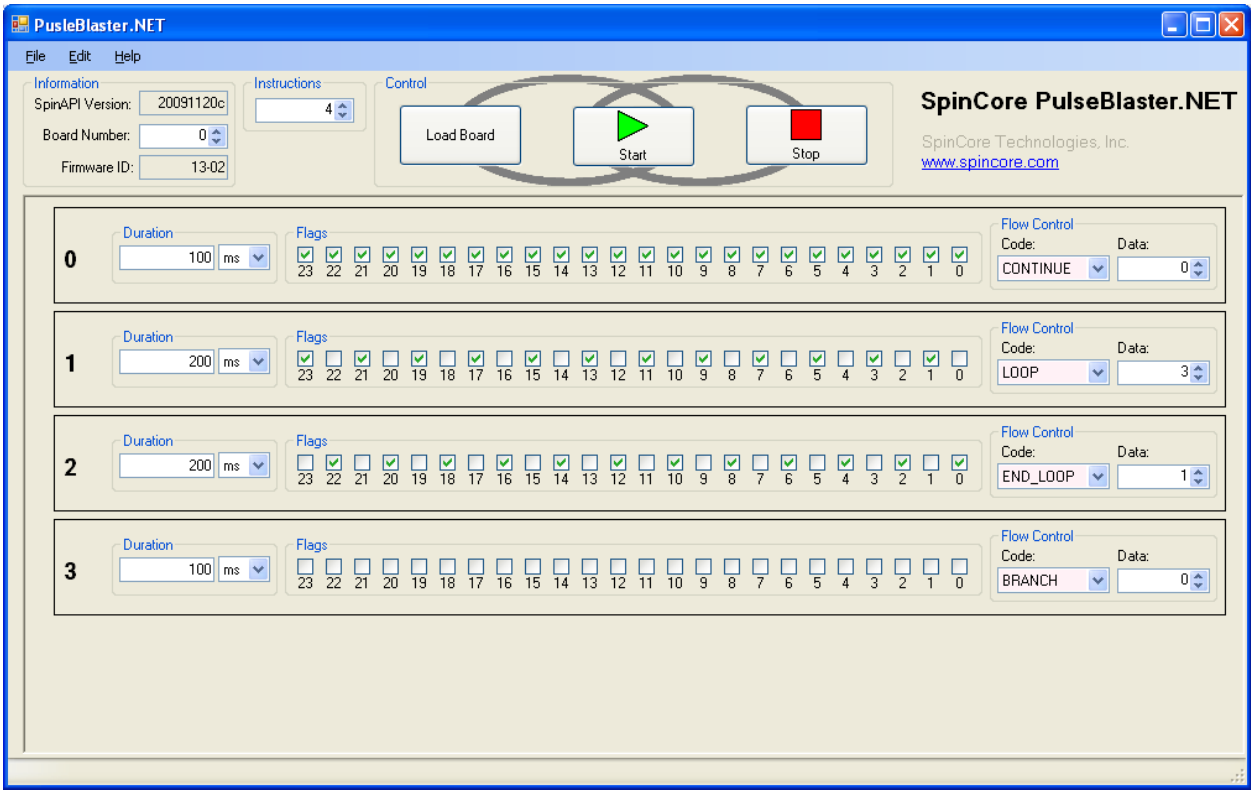


Figure 3: An example pulse program in PulseBlaster.NET. This example creates a pulse that has all TTL bits on for 100 ms, alternating bits on for 400 ms (looping three times), and then all bits off for 100 ms.

PulseBlaster.NET is available on the web from http://www.spincore.com/support/net/.

LabVIEW Extensions

The SpinCore PulseBlaster LabVIEW Extensions (PBLV) provide the ability to program and control the functionality of PulseBlaster boards using the simple National Instruments (NI) LabVIEW graphical programming interface. The package contains basic subVIs that can be used to include PulseBlaster interaction from your own LabVIEW programs, as well as some complete example VIs. Additionally, all of the examples are available as stand-alone applications, so that no programming is necessary for use.

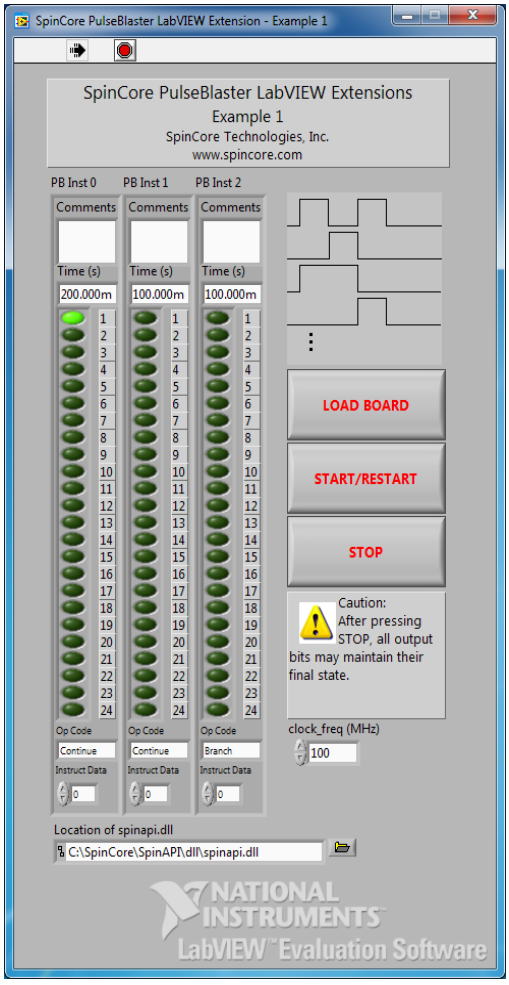


Figure 4: Example of PulseBlaster LabVIEW Extensions User Interface. The example shown has three instructions that toggle TTL bit 1 on for 200 ms and off for 200 ms.

There are two versions of the LabVIEW extensions available free of charge on SpinCore's website. The first is for those who do not have LabVIEW or who are not familiar with LabVIEW programming. This option is a stand-alone GUI (see Figure 4 above) that comes in executable form and utilizes the LabVIEW runtime environment. The second is for those who have LabVIEW and

would like to make a custom interface for the PulseBlaster board. For more information and downloads please visit:

http://www.spincore.com/support/PBLV/TTL.shtml

PulseBlaster Matlab GUI

PulseBlaster Matlab GUI is a graphical interface for creating pulse programs and loading them to the PulseBlaster board. PulseBlaster Matlab GUI currently provides the simplest interface possible to pulse control. Figure 5 shows an example instance of the program.

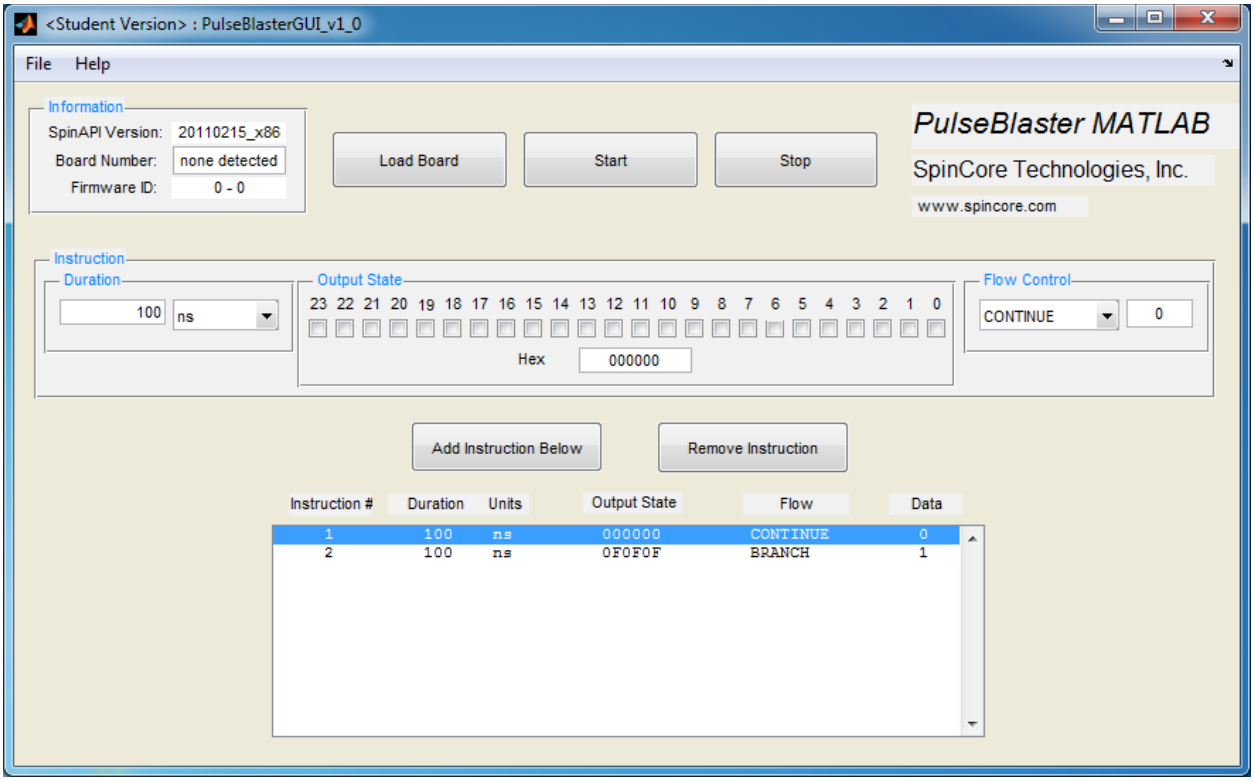


Figure 5: An example pulse program in PulseBlaster Matlab GUI.

PulseBlaster Matlab GUI is available at: http://www.spincore.com/support/PulseBlasterMatlabGUI/.

C/C++ Programming

The most dynamic and flexible way to program the PulseBlaster board is with C/C++ using the SpinAPI package. The GUI based approaches to programming the board are designed for simplicity so they can be used by someone with no programming experience. While GUI's are easier to use, coding in C/C++ allows you to better utilize all features of the board, and in some cases it may be easier to copy and paste lines of code than to make 100 instructions on a GUI. With the pre-configured compiler package available on our website, changing one of our example programs and recompiling the executable file for use with your PulseBlaster board is as easy as clicking “Rebuild All” (see Figure 6 below). The free pre-configured compiler package is available at the following URL, under the “Windows Pre-configured Compiler” heading:

http://www.spincore.com/support/spinapi/

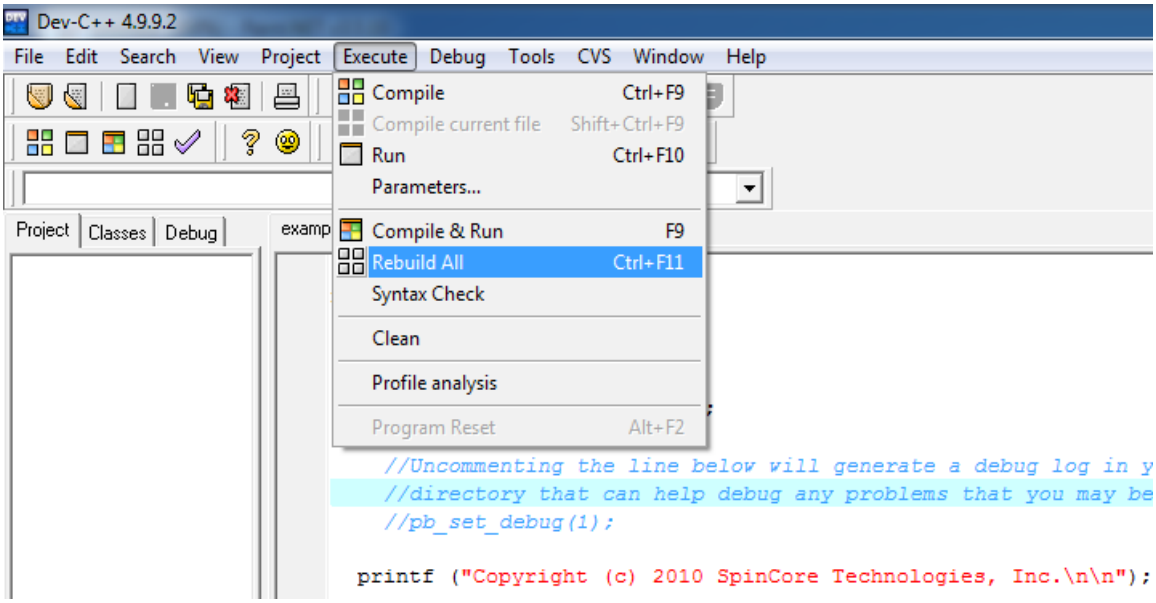


Figure 6: Compiling a C program to run the PulseBlaster board is easy!

Making changes to an example program requires understanding of only a few lines of code. The most important is the following line from pb24_ex1.c (found in C:\SpinCore\Examples\PulseBlaster24 if the examples were installed in the default directory):

```
pb_inst(0xFFFFFF, CONTINUE, 0, 200.0*ms);
```

This line of code produces a high output on all the TTL bits lasting for 200 ms and then continues on to the next instruction. This is accomplished using the four parameters in the function call (parameters are located between parentheses and are separated by commas).

- The first is the hexadecimal 0xFFFFFF which corresponds to setting the 24 output bits to a logical high since it translates to a binary string of 24 1's.

- The second parameter is CONTINUE which means to proceed on to the next instruction after this one completes. Other examples for what this parameter could be are BRANCH or LOOP.

- The third parameter is the instruction data field which, for a CONTINUE instruction, is ignored because it is unnecessary for that particular instruction. In the event of another instruction, such as BRANCH, this parameter would correspond to the target of the BRANCH instruction.

- The fourth parameter is 200.0*ms which means that this instruction will last for 200 ms.

A simple program to generate a square wave signal on all 24 output bits will have two intervals (as in the GUI Interpreter described earlier), as shown below:

```
start= pb_inst(0xFFFFFF, CONTINUE, 0, 200.0*ms);
pb_inst(0x000000, BRANCH, start, 200.0*ms);
```

The first line of the code above corresponds to the logical "one" on all output bits. The second line corresponds to the logical "zero," after which the program branches (jumps) back to the beginning, thus resulting in a continuous generation of a square wave on all outputs.

A complete C program will have, in addition to the two lines above, the initialization section, the closing section and, optionally, the (software) trigger to start the execution immediately upon launch of the program. For more detailed information on programming the board using C/C++ see the appendices.

IV. Connecting to the PulseBlaster Board

Connector Information

Sketch of PulseBlaster, Board Version SP2

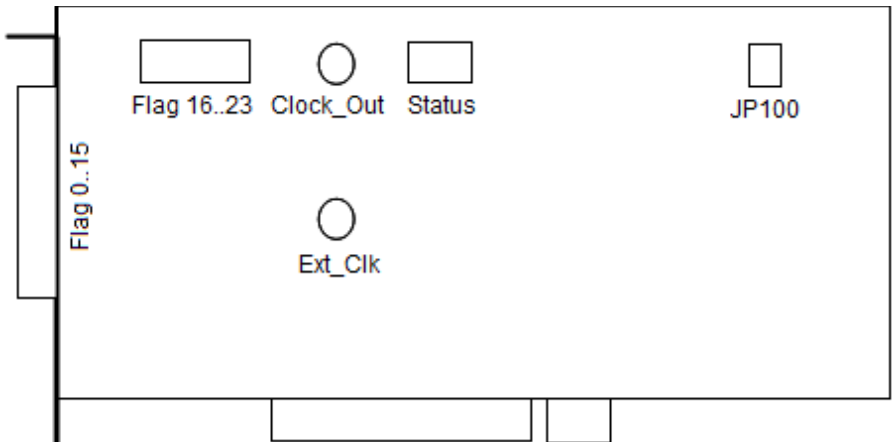


Figure 7: On the board the Clock_Out and Ext_Clk are SMA connectors, Flag0..15 is a DB-25 connector, Flag16..23, Status, and JP100 are IDC header connectors.

Sketch of PulseBlaster, Board Version SP17

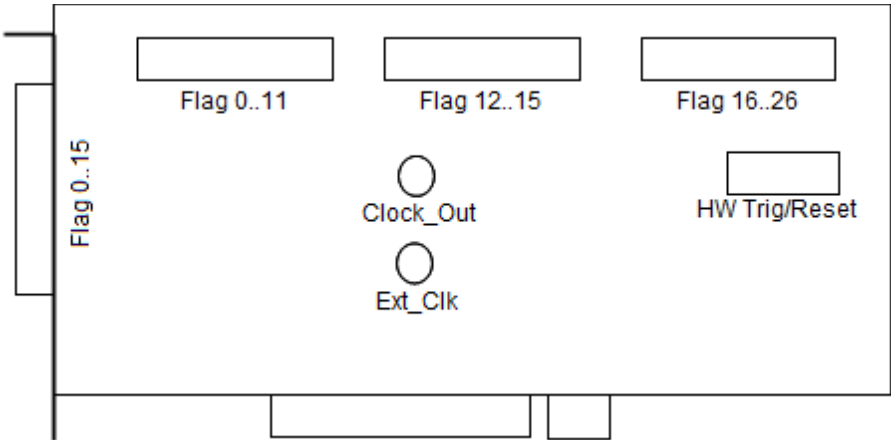


Figure 8: On the board the Clock_Out and Ext_Clk are SMA connectors, Flag0..15 is a DB-25 connector, Flag0..11, Flag12..15, Flag16..26 and HW Trig/Reset are shrouded IDC header connectors.

Note: Other versions of the board may exist, but main connector features are typically preserved.

General Pin Assignments (Common to SP2 and SP17)

DB25 Bracket Connector Flag 0..15 - Pin Assignments

Outputs 16 TTL signals generated by the user's program. Please consult the table below for bit assignments.

Pin Assignments			
Pin#	Bit#	Pin#	Bit#
1	GND	14	GND
2	Bit 15	15	Bit 14
3	GND	16	Bit 13
4	Bit 12	17	GND
5	Bit 11	18	Bit 10
6	GND	19	Bit 9
7	Bit 8	20	GND
8	Bit 7	21	Bit 6
9	GND	22	Bit 5
10	Bit 4	23	GND
11	Bit 3	24	Bit 2
12	GND	25	Bit 1
13	Bit 0		

Table 1: Lower 16 output bits and 9 ground lines on the bracket-mounted DB25 connector.

SMA Connector Clock_Out

This SMA connector outputs the reference clock as a 3.3 V TTL signal, i.e., it generates positive-only voltage. Note that SP17 models use 50 MHz as the reference clock frequency and that clock is internally multiplied to provide that actual PulseBlaster Core frequency. The output resembles a square wave if properly terminated. This signal can be measured with an oscilloscope using either a high impedance probe at the SMA connector or a 50 ohm coaxial line that is terminated.

SMA Connector Ext_Clk

This SMA connector can be used to input an external clock signal. Extreme care should be exercised, and certain conditions have to be met prior to using this connector. First, before attaching any external clock source, the internal clock oscillator must be removed from its socket. The internal clock oscillator's orientation should be noted - if the internal clock is reconnected, it must be inserted in the same orientation or board damage may occur. Second, the external clock signal must be 3.3 V TTL, i.e., a positive-only voltage - any negative voltage at the Ext_Clk connector will damage the programmable-logic processor chip. Third, as the Ext-Clk connector is not terminated on the printed circuit board, a 50 ohm terminating resistor should be used externally via a T connector placed directly

at the SMA Ext_Clk connector. Alternatively, a 50 ohm resistor could be soldered on the board on R401 pads (R200 pads for SP17 models).

SP2-Specific Pin Assignments

IDC Connector Flag16..23 - Pin Assignments

The IDC connector labeled Flag 16..23 outputs TTL signals generated by the user's program. Please consult the table below for pin assignments.

Pin Assignments			
Pin#		Pin#	
1	Bit 16	9	Bit 20
2	GND	10	GND
3	Bit 17	11	Bit 21
4	GND	12	GND
5	Bit 18	13	Bit 22
6	GND	14	GND
7	Bit 19	15	Bit 23
8	GND	16	GND

Table 2: Upper 8 output bits and 8 ground lines on the 16 pin IDC connector (SP2 board revision).

The IDC connector labeled Flag 16..23 can also be accessed using an SP32 board (Figure 13) which allows the use of MMCX cables. This enables the individual bits of the PulseBlaster to be more easily accessed. Pin 1 on the MMCX adapter board can identified with a square pin.

IDC Connector Status - Pin Assignments

The IDC connector labeled Status outputs TTL signals based on status of the user's program. Please consult the table below for pin assignments.

Pin Assignments			
Pin#		Pin#	
1	Stopped	6	GND
2	GND	7	Waiting
3	Reset	8	GND
4	GND	9	Reserved
5	Running	10	GND

Table 3: 5 status signals and 5 ground lines on the 10 pin IDC connector (SP2 board revision).

The status pins correspond to the current state of the pulse program and are defined as follows:

Stopped – Driven high when the PulseBlaster device has encountered a STOP Op Code during program execution and has entered a stopped state.

Reset – Driven high when the PulseBlaster device is in a RESET state and must be reprogrammed before code execution can begin again.

Running – Driven high when the PulseBlaster device is executing a program. It is low when the PulseBlaster enters either a reset or idle state.

Waiting – the PulseBlaster device has encountered a WAIT Op Code and is waiting for the next trigger (either hardware or software) to resume operation.

Header JP100

This is an input connector, for hardware triggering (HW_Trigger) and hardware resetting (HW_Reset) on SP2 boards.

HW_Trigger (active-low) is pulled high by default using a 10 kΩ resistor, and pin 1 is active (pin 2 = GND). When a falling edge is detected (e.g., when shorting pins 1-2) and the program is idle, code execution is triggered. If the program is idle due to a WAIT Op Code, then the HW_Trigger will cause the program to continue to the next instruction. If the program is idle due to a STOP Op Code or a HW_Reset signal, then the HW_Trigger starts execution from the beginning of the program. When using the STOP Op Code, a HW_Reset or software reset (pb_reset()) needs to be applied prior to the HW_Trigger.

NOTE: *The PulseBlasterUSB requires a 3.3V TTL input signal for HW_Trigger.* ***A signal that is more than 3.3V or less than 0V will damage the device****.*

HW_Reset (active-low) is pulled high by default using a 10 kΩ resistor, and pin 3 is active (pin 4 = GND). It can be used to halt the execution of a program by pulling it low (e.g., by shorting pins 3-4). When the signal is pulled low during the execution of a program, the controller resets itself back to the beginning of the program. Program execution can be resumed by either a software start command (pb_start()) or by a hardware trigger.

NOTE: ***The*** *PulseBlasterUSB requires a 3.3V TTL input signal for HW_Trigger.* ***A signal that is more than 3.3V or less than 0V will damage the device****.*

SP17-Specific Pin Assignments

Shrouded IDC Connector Flag0..11 - Pin Assignments

The shrouded IDC connector labeled Flag 0..11 outputs TTL signals generated by the user's program. Please consult the table below for pin assignments.

Pin Assignments			
Pin#		Pin#	
1	Bit 0	13	Bit 6
2	GND	14	GND
3	Bit 1	15	Bit 7
4	GND	16	GND
5	Bit 2	17	Bit 8
6	GND	18	GND
7	Bit 3	19	Bit 9
8	GND	20	GND
9	Bit 4	21	Bit 10
10	GND	22	GND
11	Bit 5	23	Bit 11
12	GND	24	GND

Table 4: Lower 12 output bits and 12 ground lines on the 24 pin IDC connector (SP17 board revision).

The shrouded IDC connector labeled Flag 0..11 can also be accessed using an SP32 board (Figure 13) which allows the use of MMCX cables. This enables the individual bits of the PulseBlaster to be more easily accessed. Pin 1 on the MMCX adapter board can identified with a square pin.

Shrouded IDC Connector Flag12..23 - Pin Assignments

The shrouded IDC connector labeled Flag 12..23 outputs TTL signals generated by the user's program. Please consult the table below for pin assignments.

The shrouded IDC connector labeled Flag 12..23 can also be accessed using an SP32 board (Figure 13) which allows the use of MMCX cables. This enables the individual bits of the PulseBlaster to be more easily accessed. Pin 1 on the MMCX adapter board can identified with a square pin.

Pin Assignments			
Pin#		Pin#	
1	Bit 12	13	Bit 18
2	GND	14	GND
3	Bit 13	15	Bit 19
4	GND	16	GND
5	Bit 14	17	Bit 20
6	GND	18	GND
7	Bit 15	19	Bit 21
8	GND	20	GND
9	Bit 16	21	Bit 22
10	GND	22	GND
11	Bit 17	23	Bit 23
12	GND	24	GND

Table 5: Higher 12 output bits and 12 ground lines on the 24 pin IDC connector (SP17 board revision).

Shrouded IDC Connector Flag24..26 - Pin Assignments

The shrouded IDC connector labeled Flag 0..11 outputs TTL signals generated by the user's program. Please consult the table below for pin assignments.

Pin Assignments			
Pin#		Pin#	
1	Reset	13	GND
2	GND	14	GND
3	Running	15	GND
4	GND	16	GND
5	Waiting	17	GND
6	GND	18	GND
7	GND	19	GND
8	GND	20	GND
9	GND	21	GND
10	GND	22	GND
11	GND	23	GND
12	GND	24	GND

Table 6: 3 status signals and 21 ground lines on the 24 pin IDC connector (SP17 board revision).

The status pins correspond to the current state of the pulse program and are defined as follows:

Reset – Driven high when the PulseBlaster device is in a RESET state and must be reprogrammed before code execution can begin again.

Running – Driven high when the PulseBlaster device is executing a program. It is low when the PulseBlaster enters either a reset or idle state.

Waiting – the PulseBlaster device has encountered a WAIT Op Code and is waiting for the next trigger (either hardware or software) to resume operation. Note that the Running bit will also be high during a WAIT state.

Shrouded IDC Connector HW Trig/Reset

This is an input connector, for hardware triggering (HW_Trigger) and resetting (HW_Reset).

Pin Assignments			
Pin#		Pin#	
1	GND	2	HW_Trigger_H
3	GND	4	HW_Trigger_H
5	GND	6	HW_Reset_H
7	GND	8	HW_Reset
9	GND	10	HW_Trigger

Table 7: Pinout for HW_TRIG/RESET IDC Connector on SP17 boards.

In addition to the standard active-low **HW_Trigger** pin, SP17 models have two active-high **HW_Trigger_H** pins. The functionality of these three pins are identical, however, the HW_Trigger pin is pulled high on board and therefore can be triggered by a low pulse (or shorting to GND, i.e., pin 9). The HW_Trigger_H pins are pulled low on board and can be triggered by a high pulse. This trigger will restart execution of a program from the beginning of the code if it is detected after the design has reached an idle state. The idle state could have been created either by reaching the STOP Op Code of a program, or by the detection of the HW_Reset signal. If the STOP Op Code is used, a HW_Reset or software reset (pb_reset()) needs to be applied prior to the HW_Trigger. When the WAIT Op Code is used in the pulse program, the HW_Trigger will cause the program to continue to the next instruction.

In addition to the standard active-low **HW_Reset** pin, SP17 models have one active-high **HW_Reset_H** pin. The functionality of these two pins are identical, however, the HW_Reset pin is pulled high on board and therefore can be activated by a low pulse (or shorting to GND, i.e., pin 7). The HW_Reset_H pin is pulled low on board and can be activated by a high pulse. When the signal is activated during the execution of a program, the controller resets itself back to the beginning of the program. Program execution can be resumed by either a software start command or by a hardware trigger.

Note that for all board models the IDC pins are enumerated in the manner shown by Figure 9. Pin 1 is marked on the board and the rest of the pins follow in this fashion (for the 26 pin IDC connectors, the pin numbers simply continue in this pattern until pin 26).

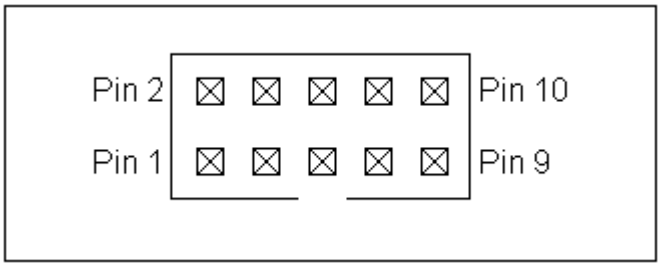


Figure 9: IDC connector pin enumeration.

Appendix I: Controlling the PulseBlaster with SpinAPI

Introduction

This section provides detailed descriptions of the instruction set for the processor on the PulseBlaster board and the C functions in SpinAPI that utilize them. The information on the instruction set is very in depth and knowledge of this is essential to be able to properly operate the board. Details of the instruction set architecture are provided first so that the user can understand the functionality of the PulseBlaster. The second part provides information about SpinCore's Application Programming Interface (API) package, called SpinAPI.

Instruction Set Architecture

Machine-Word Definition

The PulseBlaster pulse timing and control processor implements an 80-bit wide Very Long Instruction Word (VLIW) architecture. The VLIW memory words have specific bits/fields dedicated to specific purposes, and every word should be viewed as a single instruction of the micro-controller. The maximum number of instructions that can be loaded to on-chip memory is equal to the memory size described in the model number (i.e., 4k memory words for Model PB24-100-4k, 32k memory words for Model PB24-100-32k, etc.). The execution time of instructions can be varied and is under (self) control by one of the fields of the instruction word – the shortest being five clock cycles for the "Internal Memory Model" and nine clock cycles for the "External Memory Model." All instructions have the same format and bit length, and all bit fields have to be filled. Figure 10 shows the fields and bit definitions of the 80-bit instruction word.

Breakdown of 80-bit Instruction Word

Bit Definitions for the 80-bit Instruction Word (VLIW)

Output/Control Word (24 bits)	Data Field (20 bits)	OP Code (4 bits)	Delay Count (32 bits)

Figure 10: Bit definitions of the 80-bit instruction/memory word.

The 80-bit VLIW is broken up into 4 sections

1. Output Pattern and Control Word - 24 bits
2. Data Field - 20 bits
3. OP Code - 4 bits
4. Delay Count - 32 bits.

Output Pattern and Control Word

Please refer to Table 8 for output pattern and control bit assignments of the 24-bit output/control word.

Bit #	Output Connector Label	Bit #	Output Connector Label
23	Flag16..23 Out pin 15	11	Flag0..15 Out pin 5
22	Flag16..23 Out pin 13	10	Flag0..15 Out pin 18
21	Flag16..23 Out pin 11	9	Flag0..15 Out pin 19
20	Flag16..23 Out pin 9	8	Flag0..15 Out pin 7
19	Flag16..23 Out pin 7	7	Flag0..15 Out pin 8
18	Flag16..23 Out pin 5	6	Flag0..15 Out pin 21
17	Flag16..23 Out pin 3	5	Flag0..15 Out pin 22
16	Flag16..23 Out pin 1	4	Flag0..15 Out pin10
15	Flag0..15 Out pin 2	3	Flag0..15 Out pin 11
14	Flag0..15 Out pin 15	2	Flag0..15 Out pin 24
13	Flag0..15 Out pin 16	1	Flag0..15 Out pin 25
12	Flag0..15 Out pin 4	0	Flag0..15 Out pin 13

Table 8: Output Pattern and Control Word Bits.

Data Field and Op Code

Please refer to Table 9 for information on the available operational codes (OpCode) and the associated data field functions (the data field's function is dependent on the OpCode).

Op Code #	Inst	Inst_data	Function
0	CONTINUE	Ignored	Program execution continues to next instruction.
1	STOP	Ignored	Stop execution of program. Aborts the operation of the micro-controller with no control of output states (all TTL values remain from previous instruction). It is recommended that prior to the STOP op-code a short interval should be added to set the output states as desired.
2	LOOP	Number of desired loops. This value must be greater than or equal to 1.	Specify beginning of a loop. Execution continues to next instruction. Data used to specify number of loops
3	END_LOOP	Address of beginning of loop	Specify end of a loop. Execution returns to beginning of loop and decrements loop counter.
4	JSR	Address of first subroutine instruction	Program execution jumps to beginning of a subroutine
5	RTS	Ignored	Program execution returns to instruction after JSR was called
6	BRANCH	Address of next instruction	Program execution continues at specified instruction
7	LONG_DELAY	Number of desired loops. This value must be greater than or equal to 3.	For long interval instructions. Data field specifies a multiplier of the delay field. Execution continues to next instruction
8	WAIT	Ignored	Program execution stops and waits for software or hardware trigger. Execution continues to next instruction after receipt of trigger. A WAIT instruction must be preceded by an instruction lasting longer than the minimum instruction time.

Table 9: Op Code and Data Field Description.

Delay Count

The value of the Delay Count field (a 32-bit value) determines how long the current instruction should be executed. The allowed minimum value of this field is 0x00000002 for the 4k and 0x00000006 for the 32k models, and the allowed maximum is 0xFFFFFFFF. The timing controller has a fixed delay of three clock cycles and the value that one enters into the Delay Count field should account for this inherent delay. (NOTE: the pb_inst() family of functions in SpinAPI and the PulseBlaster Interpreter automatically account for this delay.)

About SpinAPI

SpinAPI is a control library which allows programs to be written to communicate with the PulseBlaster board. The most straightforward way to interface with this library is with a C/C++ program, and the API definitions are described in this context. However, virtually all programming languages and software environments (including software such as LabView and Matlab) provide mechanisms for accessing the functionality of standard libraries such as SpinAPI.

Please see the example programs for an an explanation of how to use SpinAPI. A reference document for all SpinAPI functions is available online at the following URL under the "Windows Drivers and Example Programs" heading, at the "API Reference" link:

http://www.spincore.com/support/spinapi/

For a pre-configured compiler for writing and modifying pulse programs download Dev-C++ with MinGW from our website at the URL above under the "Windows Pre-configured Compiler" heading.

Using C Functions to Program the PulseBlaster

A series of functions have been written to control the board and facilitate the construction of pulse program instructions.

In order to use these functions, the DLL (spinapi.dll), the library file (libspinapi.a for mingw, spinapilibgcc for Borland, and spinapi.lib for MSVC), the header file (spinapi.h), must be in the working directory of your C compiler[1].

```
int pb_init();
```

Initializes PulseBlaster board. Needs to be called before calling any functions using the PulseBlaster. Returns a negative number on an error or 0 on success.

```
int pb_close();
```

Releases PulseBlaster board. Needs to be called as last command in pulse program. Returns a negative number on an error or 0 on success.

[1] These functions and library files have been generated and tested with MinGW (www.mingw.com), Borland 5.5 (www.borland.com), MS Visual Studio 2003 (msdn.microsoft.com) compilers.

```
int pb_core_clock(double clock_freq);
```

Used to set the clock frequency of the board. The variable **clock_frequency** is specified in MHz when no units are entered. Valid units are MHz, kHz, and Hz. The default clock value is 50MHz. You only need to call this function if you are not using a –50 board. Please contact SpinCore for more information if needed.

```
int start_programming(int device);
```

Used to initialize the system to receive programming information. It accepts a parameter referencing the target for the instructions. The only valid value for **device** is `PULSE_PROGRAM`, It returns a 0 on success or a negative number on an error.

```
int pb_inst(int flags, int inst, int inst_data, double length);
```

Used to send one instruction of the pulse program. Should only be called after `start_programming(PULSE_PROGRAM)` has been called. It returns a negative number on an error, or the instruction number upon success. If the function returns –99, an invalid parameter was passed to the function. Instructions are numbered starting at 0.

int flags – determines state of each TTL output bit. Valid values are 0x0 to 0xFFFFFF. For example, 0x010 would correspond to bit 4 being on, and all other bits being off.

int inst – determines which type of instruction is to be executed. Please see Table 9 for details.

int inst_data – data to be used with the previous inst field. Please see Table 9 for details.

double length – duration of this pulse program instruction, specified in nanoseconds (ns).

```
int stop_programming();
```

Used to tell that programming the board is complete. Board execution cannot start until this command is received. It returns a 0 on success or a negative number on an error.

```
int pb_start();
```

Once board has been programmed, this instruction will start execution of pulse program. It returns a 0 on success or a negative number on an error.

```
int pb_stop();
```

Stops output of board. Analog output will return to ground, and TTL outputs will remain in the state they were in when stop command was received. It returns a 0 on success or a negative number on an error.

```
int pb_read_status();
```

Read status from the board. Each bit of the returned integer indicates whether the board is in that state. Bit 0 is the least significant bit.

- Bit 0 – Stopped
- Bit 1 - Reset
- Bit 2 – Running
- Bit 3 – Waiting
- Bit 4 - Scanning (RadioProcessor boards only)

Note on Bit 1: Bit 1 will be high, '1', as soon as the board is initialized. It will remain high until a hardware or software reset occurs. At that point, it will stay low, '0', until the board is triggered again.

Bits 5-31 are reserved for future use. It should not be assumed that these will be set to 0.

```
char* pb_get_version();
```

Returns the version of SpinAPI in the form YYYYMMDD, i.e. 20090209. This function should be used to make sure you are using an up to date version of SpinAPI.

```
int pb_select_board(int board_num);
```

If multiple boards from SpinCore Technologies are present in your system, this function allows you to select which board to communicate with. Once this function is called, all subsequent commands (such as pb_init(), pb_core_clock(), etc.) will be sent to the selected board. You may change which board is selected at any time. If you have only one board, it is not necessary to call this function. All PCI slot boards are numbered before any USB boards, starting with the number 0. This function returns a 0 upon success, and a negative number upon failure.

Example Use of C Functions

```
/*
 * PulseBlaster example 1
 * This program will cause the outputs to turn on and off with a period
 * of 400ms
 */
#include <stdio.h>
#define PB24
#include "spinapi.h"

int main(){

      int start, status;

      printf ("Using spinapi library version %s\n", pb_get_version());

      if(pb_init() != 0) {
                  printf ("Error initializing board: %s\n", pb_get_error());
            return -1;
      }

      // Tell the driver what clock frequency the board has (in MHz)
      pb_core_clock(100.0);

      pb_start_programming(PULSE_PROGRAM);

      // Instruction 0 - Continue to instruction 1 in 200ms
      // Flags = 0xFFFFFF, OPCODE = CONTINUE
      start = pb_inst(0xFFFFFF, CONTINUE, 0, 200.0*ms);

      // Instruction 1 - Continue to instruction 2 in 100ms
      // Flags = 0x0, OPCODE = CONTINUE
      pb_inst(0x0, CONTINUE, 0, 100.0*ms);

      // Instruction 2 - Branch to "start" (Instruction 0) in 100ms
      // 0x0, OPCODE = BRANCH, Target = start
      pb_inst(0x0, BRANCH, start, 100.0*ms);

      pb_stop_programming();

      // Trigger the pulse program
      pb_start();

      //Read the status register
      status = pb_read_status();
      printf("status: %d", status);

      pb_close();

      return 0;
}
```

A more complex program using C Functions is provided in Appendix II.

Appendix II: Sample C Program

```
 //*
 * PulseBlaster example 2
 * This example makes use of all instructions (except WAIT).
 */
#include <stdio.h>
#define PB24
#include <spinapi.h>

int main(int argc, char **argv){
      int start, loop, sub;
      int status;

      printf ("Using spinapi library version %s\n", pb_get_version());
      if(pb_init() != 0) {
            printf ("Error initializing board: %s\n", pb_get_error());
            return -1;
      }

      // Tell the driver what clock frequency the board has (in MHz)
      pb_core_clock(100.0);

      pb_start_programming(PULSE_PROGRAM);

      // Since we are going to jump forward in our program, we need to
      // define this variable by hand.  Instructions start at 0 and count up
      sub = 5;

      // Instruction format
      // int pb_inst(int flags, int inst, int inst_data, int length)

      // Instruction 0 - Jump to Subroutine at Instruction 4 in 1s
      start =     pb_inst(0xFFFFFF,JSR, sub, 1000.0 * ms);

      // Loop. Instructions 1 and 2 will be repeated 3 times
      // Instruction 1 - Beginning of Loop (Loop 3 times).  Continue to next
      // instruction in 1s
      loop =      pb_inst(0x0,LOOP,3,150.0 * ms);

      // Instruction 2 - End of Loop.  Return to beginning of loop or
      // continue to next instruction in .5 s
      pb_inst(0xFFFFFF,END_LOOP,loop,150.0 * ms);

      // Instruction 3 - Stay here for (5*100ms) then continue to Instruction
      // 4
      pb_inst(0x0,LONG_DELAY,5, 100.0 * ms);

      // Instruction 4 - Branch to "start" (Instruction 0) in 1 s
      pb_inst(0x0,BRANCH,start,1000.0*ms);

      // Subroutine
      // Instruction 5 - Continue to next instruction in 1 * s
      pb_inst(0x0,CONTINUE,0,500.0*ms);

      // Instruction 6 - Return from Subroutine to Instruction 1 in .5*s
      pb_inst(0xF0F0F0,RTS,0,500.0*ms);
```

```
        // End of pulse program
        pb_stop_programming();

        // Trigger the pulse program
        pb_start();

        //Read the status register
        status = pb_read_status();
        printf("status = %d", status);

        pb_close();

        return 0;
}
```

Related Products and Accessories

1. Ribbon Cable with 2x13 IDC plug and DB-25 (Parallel port style*) connector on PC bracket – can be used to route the upper 8 output bits to the back panel of the computer. For more information, please visit http://www.spincore.com/products/InterfaceCable/

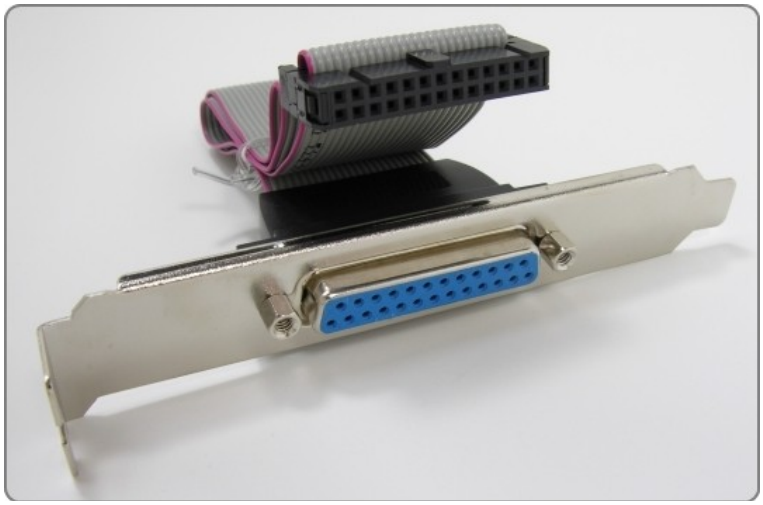


Figure 11: PulseBlaster Parallel Port Interface Cable for SP17.

***Note: This is NOT a parallel port and will not work with a PC printer or other such peripheral devices! This cable uses the parallel type DB-25 connector to easily access the TTL bits of the PulseBlaster Board.**

2. PulseBlasterESR, PulseBlasterESR-PRO, and PulseBlasterESR-PRO-II – Alternate versions of the PulseBlaster that are capable of Higher Clock Frequencies (currently up to 500 MHz). For more information, please visit the individual Product URLs of the aforementioned products at http://www.spincore.com/products.shtml

3. PulseBlasterUSB – The portable, stand-alone version of the PulseBlaster. For more information, please visit http://www.spincore.com/products/PulseBlasterUSB

4. PulseBlasterDDS – Built upon the PulseBlaster, the PulseBlasterDDS features programmable TTL outputs and RF Pulse Generation. For more information, please visit http://www.spincore.com/products/PulseBlasterDDS-300/

5. If you require an Oven Controlled Clock Oscillator (with sub-ppm stability) or other custom features, please inquire with SpinCore Technologies through our contact form, which is available at http://www.spincore.com/contact.shtml

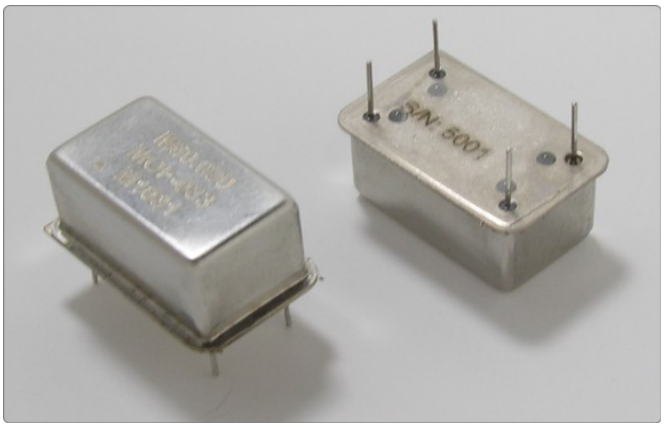


Figure 12: An Oven Controlled Clock Oscillator (or OCXO) with sub-ppm frequency stability is available for the PulseBlaster upon request.

6. SpinCore MMCX Adapter Board Figure 16 – This adapter board allows easy access to the individual bits of the PulseBlaster (only for the SP17 board). This adapter board can be part of a package that includes 12 MMCX to BNC cables and three SMA to BNC adapters. This package can be changed to include any number of cables and any number of adapter boards. For ordering information contact SpinCore at http://www.spincore.com/contact.shtml.

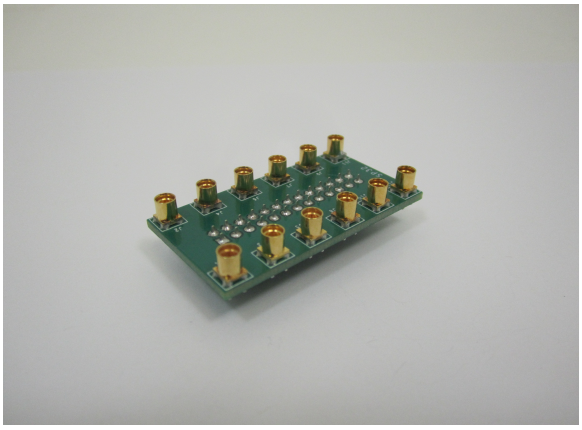


Figure 13: MMCX Adapter Board allows easy access to individual bits

Contact Information

SpinCore Technologies, Inc.
4631 NW 53rd Avenue, SUITE 103
Gainesville, FL 32653
USA

Telephone (USA):	**352-271-7383**
Fax (USA):	**352-371-8679**
Website:	http://www.spincore.com
Web Form:	http://spincore.com/contact.shtml

Document Information Page

Revision history available at SpinCore.