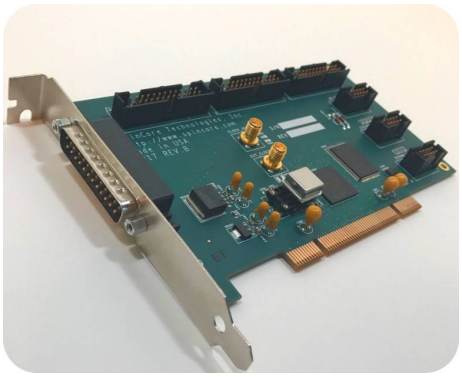
PulseBlaster - Programmable Pulse and Delay Generator

(PCI Board SP17)

(PCIe Boards SP35, SP40, SP41, SP44, SP46)

Models: PB12-100-4k, PB24-100-4k, PB24-100-32k, PB24-100-64k

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/delay generator producing up to 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, Op Codes, and will execute them much the same way as a general-purpose microprocessor does. Unlike general-purpose processors, the PulseBlaster processor features a highly optimized instruction set that has been specifically designed for timing applications. 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[1] 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.

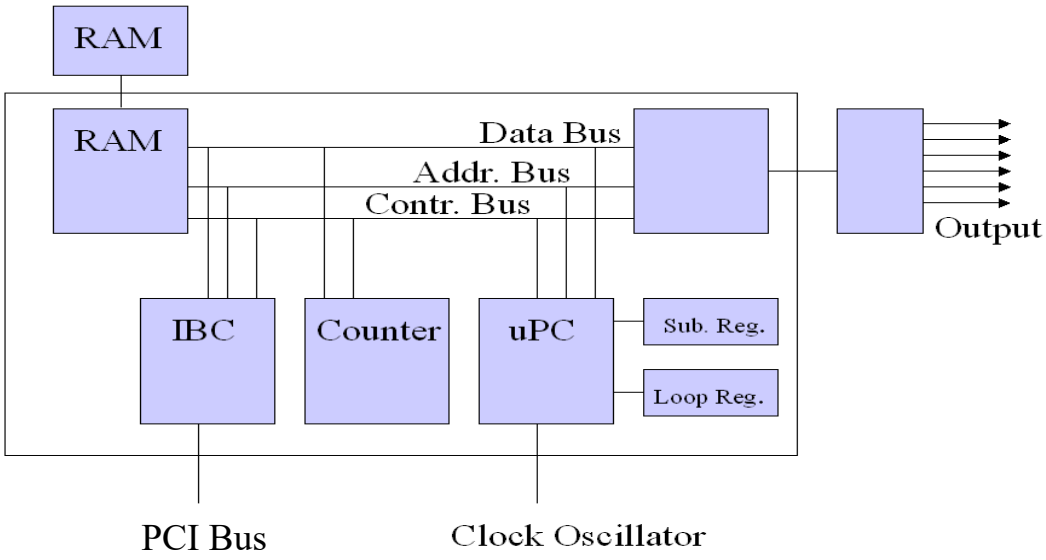


Figure 1: PulseBlaster board architecture. The clock oscillator signal is derived from an on-chip PLL circuit typically using a 50 MHz on-board reference clock.

Key Features

Output Signals

The PulseBlaster PB24 models allow for 24 digital output signal lines. Sixteen output lines are routed to a DB25 bracket-mounted connector. On the SP17 and the PCIe boards, all 24 output lines are for routed to IDCs. The PB12 models allow for 12 digital output signal lines (bits 0 to 11 as describe in the Pin Assignments). The individually controlled digital output lines comply with the transistor-transistor logic (TTL) levels' standard, and are capable of delivering up to ±25 mA per bit/channel. The number of output channels and current output are dependent on the board and firmware, so make sure to see Firmware Designs. If the load being driven is less than 132 Ohms, the

[1] SP46 boards do not have external SRAM.

voltage will drop below the TTL limit of 3.3 V. If more current is required for lower loads, users can boost power using the SpinCore TTL Line Driver.

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 and PB24-100-4k and a minimum delay cycle of nine clock periods for PB24-100-32k and PB24-100-64k. 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 and PB24-100-64k models.

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.

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 18) for more detail about the hardware lines and Appendix I (Controlling the PulseBlaster with SpinAPI, page 25) for more detail about the C function pb_read_status().

Summary

The PulseBlaster 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 or PCIe port and can accommodate pulse programs with highly flexible control commands of up to 64k (i.e., 65,536) program words (Model PB24-100-64k). Its high-current output logic bits are individually controlled with a voltage of 3.3 V.

Specifications

Pulse Parameters

- Up to 24 individually controlled digital output lines (TTL levels, 3.3 V logical “one”)
- Variable pulses/delays for every TTL line
- Up to 25 mA output current per TTL line (depends on board and firmware, see Firmware Designs)
- 50 ns shortest pulse/interval for internal memory models: PB12-100-4k and PB24-100-4k
- 90 ns shortest pulse/interval for external memory models: PB24-100-32k, PB24-100-64k
- 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 64k pulse program memory words/instructions (Model PB24-100-64k)
- 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
- Latency after trigger (WAIT state) – 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

Whenever installing or uninstalling the PulseBlaster, always have it disconnected from the computer initially. Uninstall any previous version of SpinAPI.

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 and most of SpinCore's other products. 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. Shut down the computer, unplug the power cord, insert the PulseBlaster card into an appropriate slot (PCI for PCI boards and PCIe for PCIe boards) and fasten the PC bracket securely with a screw.

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

Testing the PulseBlaster

The simplest way to test whether the PulseBlaster has been installed properly and can be controlled as intended is to run a simple test program. These example files can be found in the PulseBlaster24 folder in the examples folder of the SpinAPI.

The pb24_ex1.exe program will produce a square wave, on all digital outputs, with a logical high time of 200 ms and logical low time of 200 ms. To test the board, run pb24_ex1.exe and observe each digital output with an oscilloscope.

If using a high input impedance oscilloscope to monitor the PulseBlaster's output, place a resistor that matches the characteristic impedance of the transmission line in parallel with the coaxial transmission line at the oscilloscope input. (e.g., a 50 Ω resistor with a 50 Ω transmission line, see Figures 2 and 3 below).

Figure 2: Left: BNC T-Adapter and Right: BNC 50 Ohm resistor.

Figure 3: BNC T-Adapter on the oscilloscope input channel with coaxial transmission line connected on the left and BNC 50 Ohm resistor connected to the right to terminate the line.

Figure 4 below shows a typical pattern displayed by an oscilloscope when running pb24_ex1.exe with the above described connections. Verifying this behavior confirms the board is installed properly.

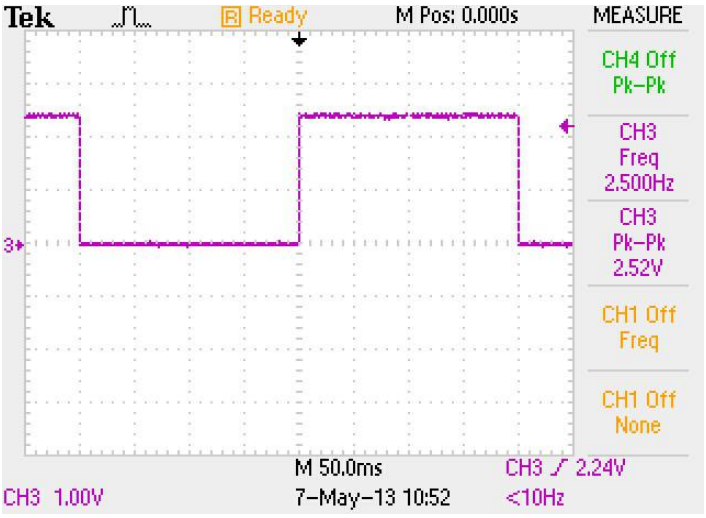


Figure 4: Expected signal from a PulseBlaster output running pb24_ex1.exe.

You may also run the remaining example programs available for this board to observe different output patterns and pulse durations. Keep in mind that pb24_programmable_clock.exe is only compatible with PulseBlasters with the programmable clock feature which is available upon request.

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*, 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 5, below, shows the PulseBlaster Interpreter being used with one of the example programs.

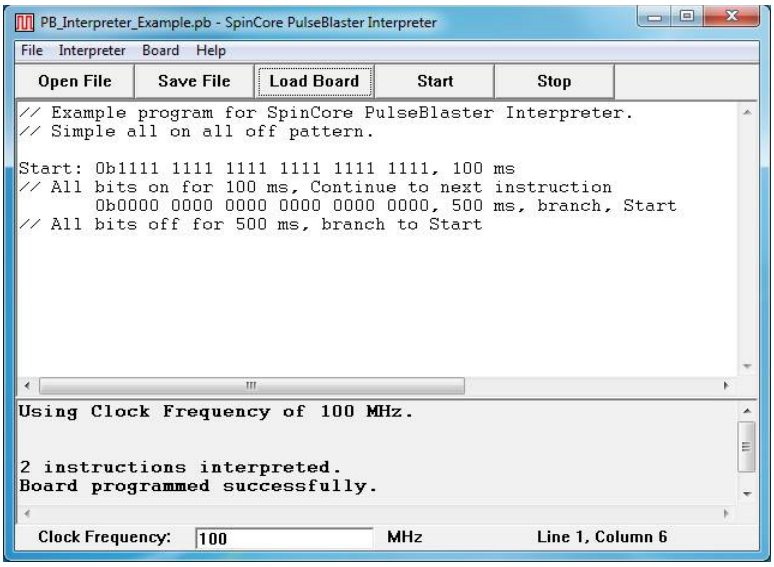


Figure 5: Graphical Interface of the PulseBlaster Interpreter. The example shown creates a pulse that toggles all TTL bits on for 100 ms, and all off for 500 ms.

The PulseBlaster Interpreter is available for download on the SpinCore website and can be found in the following location: http://www.spincore.com/support/SPBI/Interpreter_Main.shtml.

Example programs, such as the one above, are installed to C:\SpinCore\SpinAPI\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 6 shows an example instance of the program.

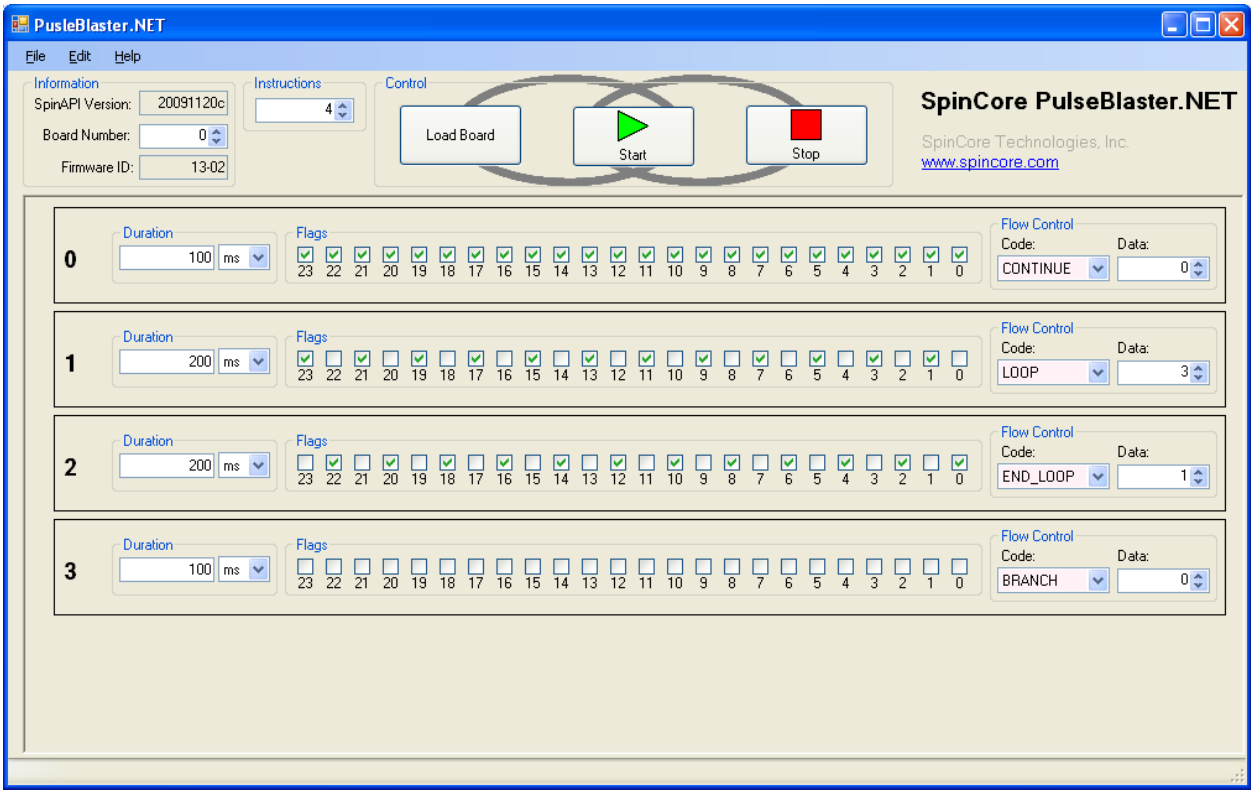


Figure 6: An example pulse program in PulseBlaster.NET. This example creates a pulse that has all TTL bits on for 100 ms, alternating bits on for 400 ms (looping three times), and then all bits off for 100 ms.

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.

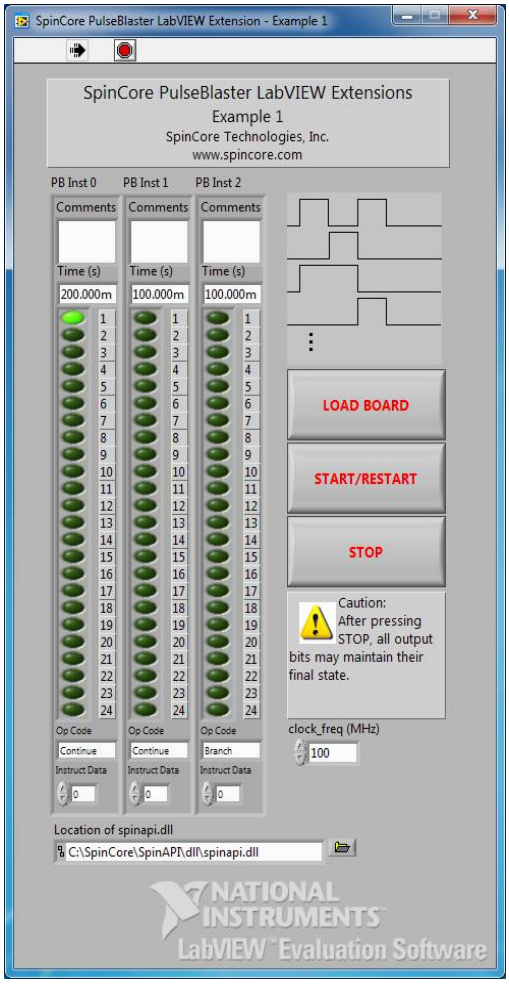


Figure 7: Example of PulseBlaster LabVIEW Extensions User Interface. The example shown has three instructions that toggle TTL bit 1 on for 200 ms and off for 200 ms.

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 7 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 8 shows an example instance of the program.

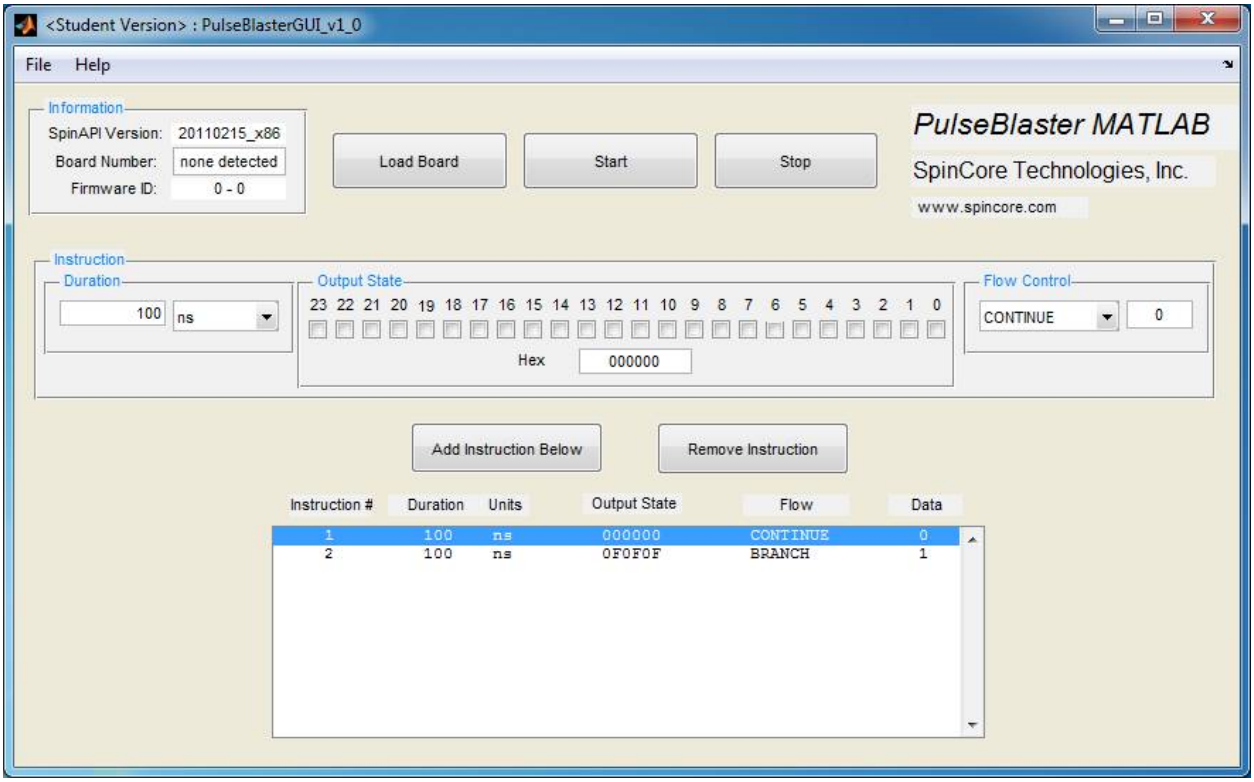


Figure 8: An example pulse program in PulseBlaster MATLAB GUI.

PulseBlaster MATLAB GUI is available at:

http://spincore.com/support/PulseBlasterMATLABGUI/pbmgui_main.shtml

C/C++ Programming

The most dynamic and flexible way to program the PulseBlaster board is with C/C++ using the SpinAPI package. 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. The instructions to compile on Windows can be found at http://www.spincore.com/support/spinapi/Windows_Help.shtml. After configuring the compiler, 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 9 below).

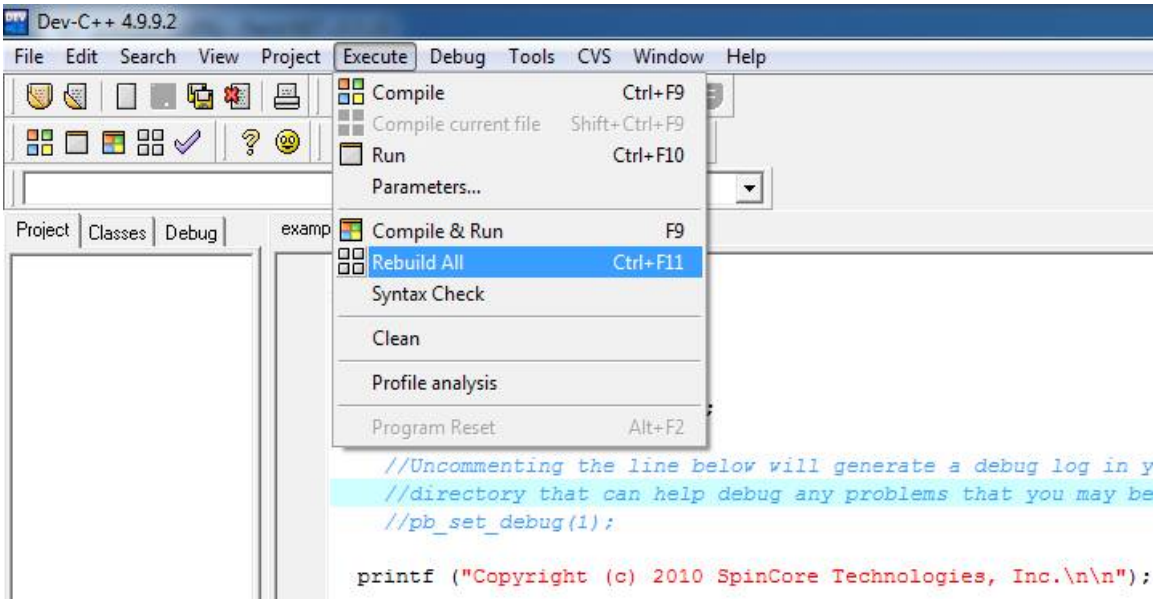


Figure 9: Compiling a C program to run the PulseBlaster board is easy!

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\SpinAPI\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

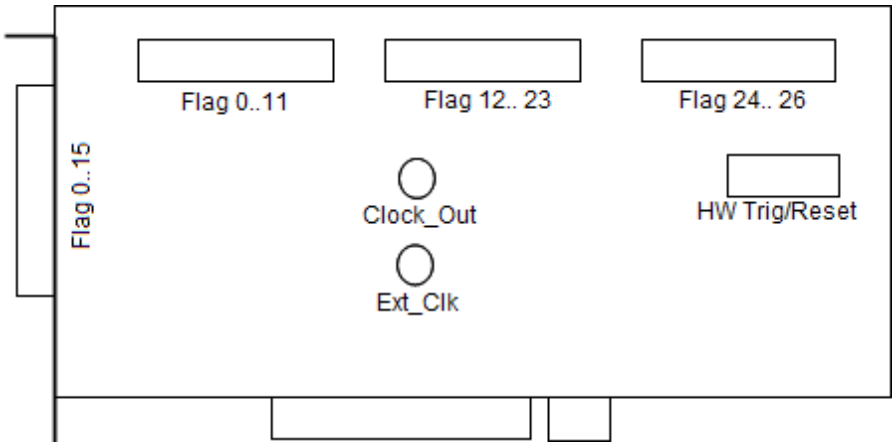


Figure 10: Sketch of PulseBlaster, Board Version SP17

On the SP17 board, the Clock_Out and Ext_Clk are SMA connectors, Flag0..15 is a DB-25 connector, Flag0..11, Flag12..23, Flag24..26 and HW Trig/Reset are shrouded IDC header connectors. Other versions of the board may exist, but main connector features are typically preserved.

General Pin Assignments

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 the PulseBlaster PCI and PCIe boards use 50 MHz as the reference clock frequency and that clock is internally multiplied to provide that actual PulseBlaster Core frequency[2]. 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, the Ext_Clk connector for certain boards is not terminated on the printed circuit board, and a 50 ohm terminating resistor should be used externally via a T connector placed directly at the SMA Ext_Clk connector. If the Ext_Clk is terminated, there will be a 50 ohm resistor on one the Ext_Clk pads: R401 pads for SP2 board, R200 pads for SP17 board, or R001 for PCIe boards. Soldering a 50 ohm resistor to these pads, if not already populated, is an alternative to using a T connector with a 50 ohm resistor.

[2] Custom firmware may use a different speed reference clock and may not be internally multiplied.

SP17 and PCIe Boards 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 2: Lower 12 output bits and 12 ground lines on the 24 pin IDC connector.

The shrouded IDC connector labeled Flag 0..11 can be connected to IDC-MMCX adapter boards (Figure 17, page 36) which allow 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 be connected to IDC-MMCX adapter boards (Figure 17, page 36) 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 3: Higher 12 output bits and 12 ground lines on the 24 pin IDC connector.

Shrouded IDC Connector Flag24..26 - Pin Assignments

The shrouded IDC connector labeled Flag 24..26 outputs three status signals: Reset, Running, and Waiting. 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 4: 3 status signals and 21 ground lines on the 24 pin IDC connector.

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).

CAUTION: ***Applying voltages to the input pins that are greater than 3.3V or less than 0V will damage the PulseBlaster***.

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 5: Pinout for HW_TRIG/RESET IDC Connector on SP17 and PulseBlaster PCIe boards.

Note that for all board models the IDC pins are enumerated in the manner shown by Figure 11, below. 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).

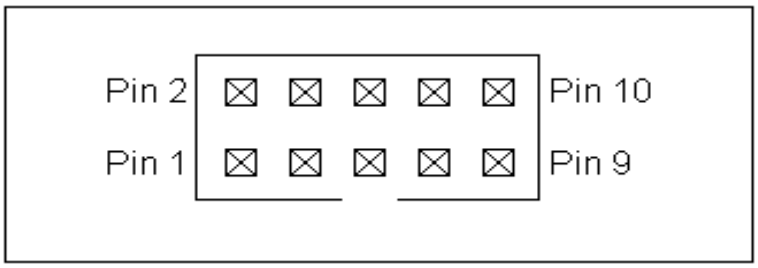


Figure 11: IDC connector pin enumeration.

SP17 and PulseBlaster PCIe boards come with three hardware trigger pins. **HW_Trigger** is pulled to high voltage (3.3V) on the board and can be triggered by a low pulse (or shorting to GND, e.g., pin 9). The two **HW_Trigger_H** pins are pulled to low voltage (ground) on the board and can be triggered with a high voltage pulse (to 3.3V). When the falling edge is detected (or rising edge on HW_Trigger_H), and the program is idle, code execution is triggered. If the program is idle due to a WAIT instruction, the HW_Trigger will cause the program to continue to the next instruction. If the program is idle due to a STOP instruction or a HW_Reset signal, the HW_Trigger will start execution from the beginning of the program. If the STOP instruction was used, a HW_Reset or software reset (pb_reset()) needs to be applied prior to the HW_Trigger.

NOTE: *The PulseBlaster requires a 3.3V input signal for HW_Trigger.* ***Applying voltages to the input pins that are greater than 3.3V or less than 0V will damage the PulseBlaster****.*

Figure 12, below, shows an example of the HW_Trigger signal with a latency of 80 ns. Please refer to the Instruction Set Architecture section in Appendix I for more details on programming the duration of the WAIT latency. To trigger once, the trigger signal must begin at high voltage (between 2V and 3.3V), then must be pulled low (to ground) and stay low for at least 10 ns before returning to high voltage. The PulseBlaster will continue to trigger or reset for as long as the HW_Trigger or HW_Reset signals stay at ground. If using a long TTL cable, make sure it is terminated and a buffer is used. If necessary, use an inverter or program the triggering device to match the high-low-high HW_Trigger signal. The input impedance of the HW_Trigger pin is 10 kOhms.

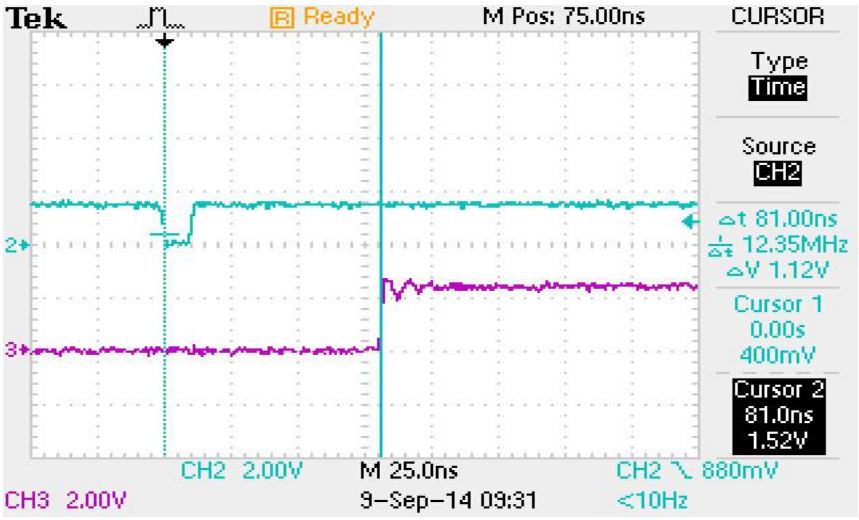


Figure 12: Demonstration of HW_Trigger high-low-high signal. The blue shows the HW_Trigger signal, the pink shows one of the output flags **Caution: applying voltages to the input pins that are greater than 3.3V or less than 0V will damage the PulseBlaster.**

SP17 and PulseBlaster PCIe models have two hardware reset pins. **HW_Reset** is pulled to high voltage (3.3V) on the board and can be activated by a low voltage pulse (or shorting to GND, e.g., pin 7). **HW_Reset_H** is pulled to low voltage on the board (ground) and can be activated by a high voltage pulse (to 3.3V). 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 started from the beginning by either a software start command (pb_start()) or by a hardware trigger.

NOTE: *The PulseBlaster requires a 3.3V input signal for HW_Reset.* ***Applying voltages to the input pins that are greater than 3.3V or less than 0V will damage the PulseBlaster****.*

Clock Oscillator Header

The PulseBlaster comes with a crystal oscillator mounted on the oscillator socket to provide a timing signal for the board. If required, it is possible to remove the oscillator that comes standard, and instead drive the PulseBlaster with an external clock signal. The oscillator module can be removed from the board, and an external signal can be input through the header pins. Do not attempt to drive a PulseBlaster board with an external clock while an oscillator module is also connected. The standard clock oscillator's orientation should be noted - if the clock oscillator is reconnected, it must be inserted in the same orientation or board damage may occur. The external clock signal must be a TTL square wave, i.e. a digital signal of no more than 3.3 V. This is the absolute maximum allowable voltage, typically a voltage of 1.5-2 V is sufficient. Be aware that the TTL signal must be a positive-only signal, any negative voltage will damage the programmable-logic chip.

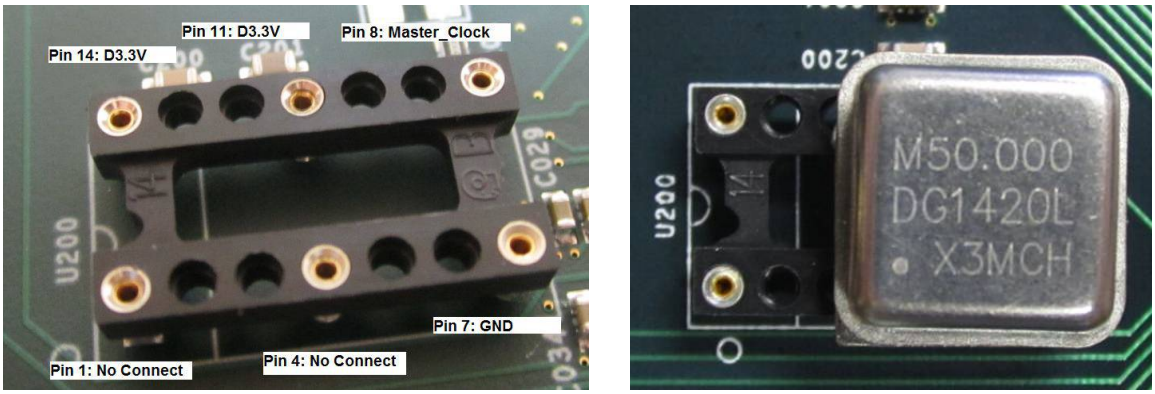


Figure 13: Both the bare header socket and the installed clock module are shown above. Please note the proper orientation of the 50 MHz clock.

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 14 shows the fields and bit definitions of the 80-bit instruction word.

Breakdown of 80-bit Instruction Word

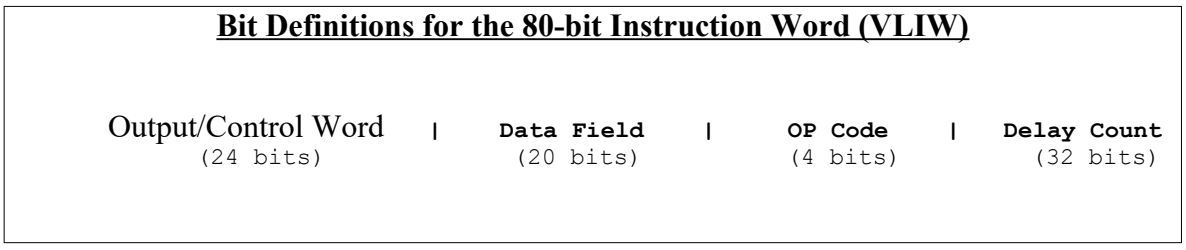


Figure 14: Bit definitions of the 80-bit instruction/memory word.

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 6 for output pattern and control bit assignments of the 24-bit output/control word.

Bit #	Output Connector Label
23	Flag12..23 Out pin 23
22	Flag12..23 Out pin 21
21	Flag12..23 Out pin 19
20	Flag12..23 Out pin 17
19	Flag12..23 Out pin 15
18	Flag12..23 Out pin 13
17	Flag12..23 Out pin 11
16	Flag12..23 Out pin 9
15	Flag12..23 Out pin 7
14	Flag12..23 Out pin 5
13	Flag12..23 Out pin 3
12	Flag12..23 Out pin 1

Bit #	Output Connector Label
11	Flag0..11 Out pin 23
10	Flag0..11 Out pin 21
9	Flag0..11 Out pin 19
8	Flag0..11 Out pin 17
7	Flag0..11 Out pin 15
6	Flag0..11 Out pin 13
5	Flag0..11 Out pin 11
4	Flag0..11 Out pin 9
3	Flag0..11 Out pin 7
2	Flag0..11 Out pin 5
1	Flag0..11 Out pin 3
0	Flag0..11 Out pin 1

Table 6: Output Pattern and Control Word Bits.

Data Field and Op Code

Please refer to Table 7 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. (Please see note below)
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	Delay multiplier. This value must be greater than or equal to 2.	For long interval instructions. Executes length of pulse given in the time field multiplied by the value in the data field.
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 7: Op Code and Data Field Description.

NOTE: For SP17 boards model PB12-100-4k, the output can be set and held by the control word. The behavior of the STOP OpCode maybe different based on the firmware version. If you have any questions, please contact SpinCore.

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:

http://www.spincore.com/support/spinapi/reference/production/2013-09-25/spinapi_8h.html

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[3].

```
int pb_init();
```

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

```
int pb_close();
```

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

[3] 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 MHz board which is reflected in the model (e.g. PB24-100-4k). 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 7 for details.

int inst_data – data to be used with the previous inst field. Please see Table 7 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, e.g. 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 5 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;
}
```

Appendix III: Available Firmware Designs

The following table contains information about the various firmware designs available on the PulseBlaster series of boards.

Firmware Revision	Board	Clock Speed (MHz)	Number of Output Bits	Memory Depth (words)	Output Current Strength (mA)
19-9	SP17	100	12	4k	21
19-15	SP17	100	12	4k	21
19-16	SP17	100	24	64k	8[4]
19-17	SP17	100	24	4k	21
21-2	SP35	100	12	4k	21
22-1	SP40	100	24	4k	21
23-1	SP41	100	4	4k	21
25-1	SP44	100	24	4k	21
26-1	SP46	100	24	4k	21
26-4	SP46	100	12	4k	21

Table 8: Firmware Designs.

[4] SpinCore's TTL Line Driver can be used if higher current is required

Related Products and Accessories

1. Ribbon Cable with 2x13 IDC plug and DB-25 (Parallel port style*) connector on PC bracket. For more information, please visit http://www.spincore.com/products/InterfaceCable/

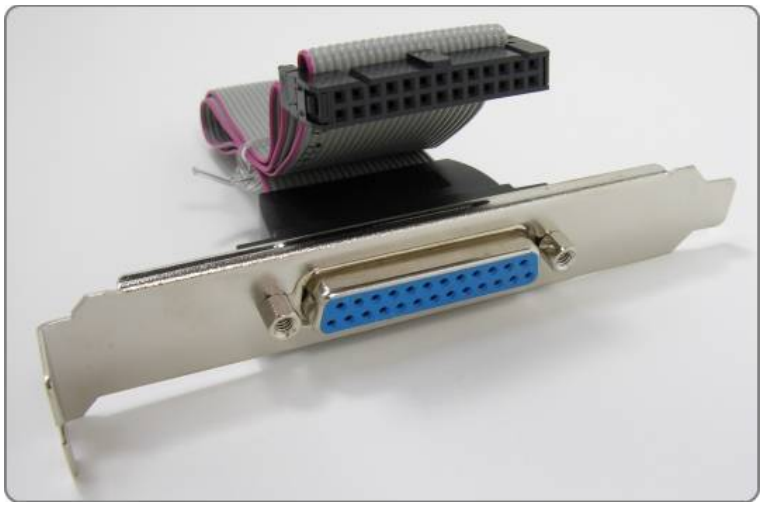


Figure 15: PulseBlaster Parallel Port Interface Cable.

***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-PRO – Alternate version of the PulseBlaster that are capable of Higher Clock Frequencies (currently up to 500 MHz). For more information, please visit http://www.spincore.com/products/PulseBlasterESR-PRO/

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 visit http://spincore.com/products/OCXO/ or inquire with SpinCore Technologies through our contact form, which is available at http://www.spincore.com/contact.shtml

Figure 16: 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 17 – This adapter board allows easy access to the individual bits of the PulseBlaster. 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, please visit http://spincore.com/products/Adapters/ or contact SpinCore at http://www.spincore.com/contact.shtml.

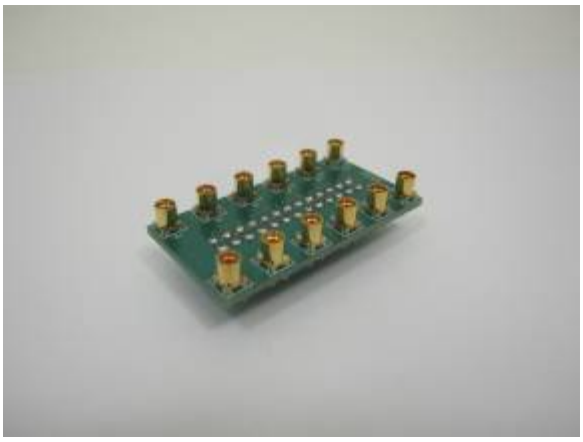


Figure 17: MMCX Adapter Board allows easy access to individual bits.

The SMA-BNC Adapter Board, shown in Figure 18, provides easy access to four additional output signals from the back panel of your computer. SMA-SMA cables are available from SpinCore upon request. For ordering information, please visit http://spincore.com/products/Adapters/ or contact SpinCore at http://www.spincore.com/contact.shtml.

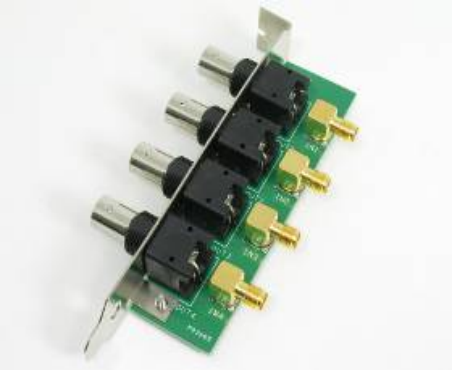


Figure 18: SMA-BNC Adapter Board is available to access additional flag bits.

IDC to BNC Adapter Set-Up on an SP4B Board Figure 19 – Additional BNC output signals can be accessed using a set-up consisting of an IDC-MMCX adapter board (SP32), MMCX-SMA cables, and an SMA-BNC adapter board (SP29P).

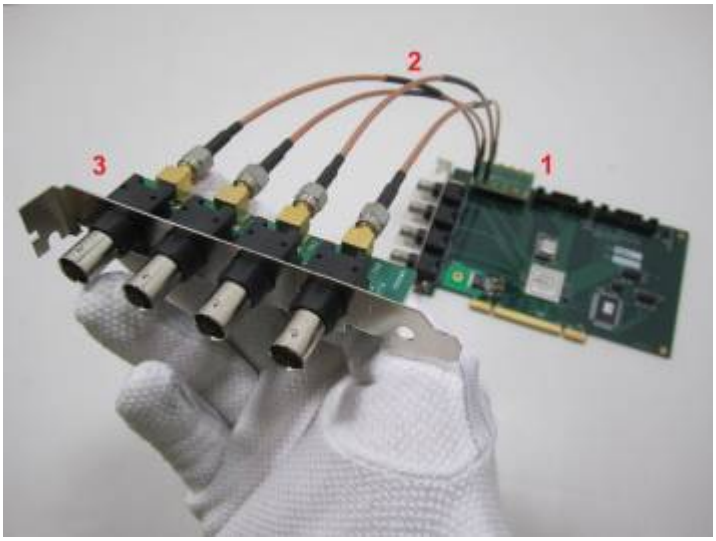


Figure 19: IDC to BNC Adapter Set-Up on an SP4B Board.

These three components correspond to Number 1 (SP32, MMCX-SMA Adapter Board), Number 2 (MMCX-SMA cables), and Number 3 (SP29P, SMA-BNC Adapter Board) below.

7. SpinCore TTL Line Driver Figure 20 - A USB-powered device with four input channels and 8 output lines. Each output line is equipped with current driving capabilities to insure TTL voltage level over 50 Ohm loads. The SpinCore TTL Line Driver is the perfect tool to accompany any TTL device. Additional specifications, ordering information, and the manual for the TTL Line Driver are available at http://www.spincore.com/products/SpinCoreTTLLineDriver/SpinCoreTTLLineDriver.shtml.

Figure 20: TTL Line Driver assures TTL levels over 50 Ohm loads.

8. If you require a custom design, custom interface cables, or other custom features, please inquire with SpinCore Technologies through our contact form, which is available at http://www.spincore.com/contact.shtml.

Contact Information

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

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

Document Information Page

Revision history available at SpinCore.