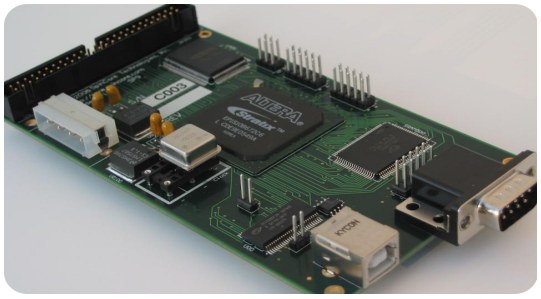
PulseBlasterUSB™ (v2)

Owner's Manual



SpinCore Technologies, Inc.
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 PulseBlasterUSB™ is a programmable multichannel pulse/delay generator that is capable of generating pulses and delays ranging from 50 ns to 4.5×10^7 seconds (~521 days) in length when operating with a 100 MHz clock frequency. The PulseBlasterUSB has 24 output channels, features highly flexible program flow control, and can accommodate pulse programs containing up to 4096 instructions; if more instructions are necessary, please contact SpinCore for more information. The PulseBlasterUSB is capable of generating complex output pulse patterns featuring very short and very long pulses in the same sequence with a resolution of one clock period (i.e., 10 ns at 100 MHz).

The intelligence of the PulseBlasterUSB comes from its proprietary microprocessor core architecture. 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.

User interaction with the PulseBlasterUSB can be accomplished in several different ways. The *PulseBlaster Interpreter* and *PulseBlaster.NET*, SpinCore's graphical programming environments for Windows, allow users to easily create, edit, save and run pulse programs. Graphical User Interfaces (GUI) for MATLAB and LabVIEW have also been developed and are available for download from SpinCore Technologies, Inc. at no charge. PulseBlaster USB can be controlled through user input of either software or hardware triggering.

Development environments like LabVIEW, MATLAB, and C/C++ can be used to interface with the PulseBlasterUSB through the dedicated Application Programming Interface (API) package called SpinAPI. The SpinAPI package can be used with virtually all Windows programming environments and can also be used with other operating systems, including Linux.

The PulseBlasterUSB is a USB 2.0 device, and can be easily integrated into a wide range of computer systems. USB 2.0 provides an easy to use device which can be quickly connected to almost any computer system, while still providing all the functionality given by the PulseBlaster PCI board.

The PulseBlasterUSB is available in four standard options: a) the board-only option with no enclosure (+5 V DC required), b) the single-bay CD-ROM-type enclosure(AC Power Supply included), c) the 2U BNC/TNC enclosure (½ width), and d) the 2U BNC Rackmount enclosure (full width). The ½ width 2U enclosure features BNC connectors for each of the output channels and external clock input, or TNC connectors upon request. All enclosed options include the internal AC/DC power supply.

The 2U BNC Rackmount enclosure (full width) is standard, while the 2U BNC/TNC enclosure (½ width) is available upon request.

II. System Architecture and Specifications

System Architecture

Figure 1 presents the internal architecture of the PulseBlasterUSB system. The major building blocks are the SRAM memory, the microcontroller (uPC), the integrated bus controller (IBC), the counter, and the output buffers. The entire logic design, including output buffers, is contained on a single silicon chip, making it a System-on-Chip design. User control of the system is accomplished via the USB bus.

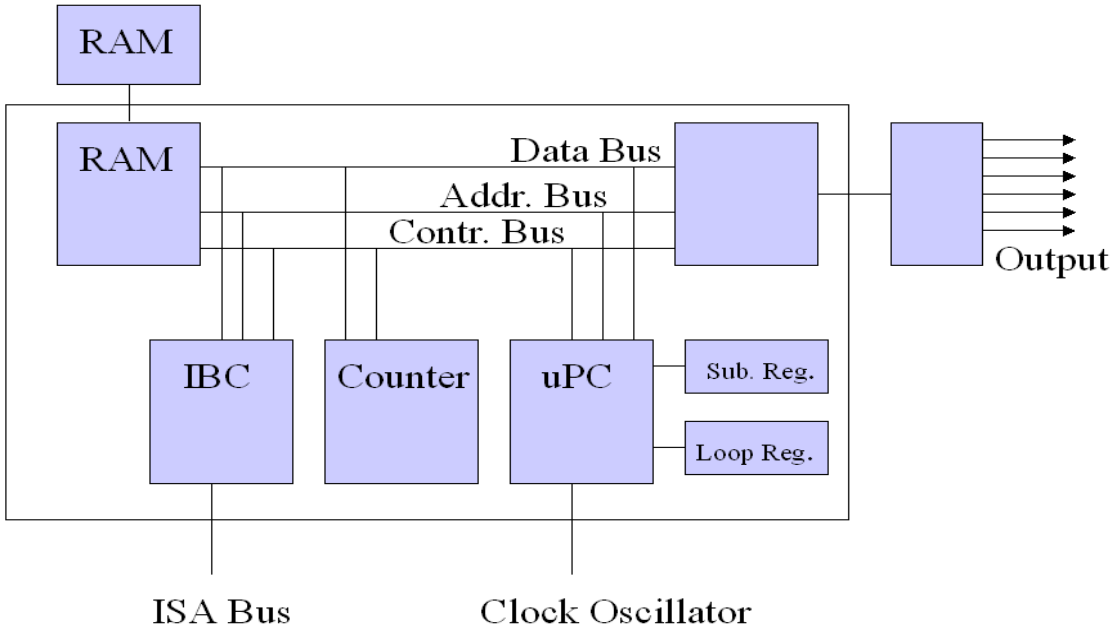


Figure 1: PulseBlasterUSB system architecture. The clock oscillator signal originates from an on-chip PLL circuit with a typical base frequency of 50 MHz.

The next page describes the *general* characteristics of the PulseBlasterUSB. For more detailed information on the *internal* operation of the PulseBlaster processor, please see **Appendix I. Instruction Set Architecture**.

Output signals

The PulseBlasterUSB has 24 digital output signal lines. All output signal lines are impedance matched to 50 ohm. The 24 individually controlled digital output bits are 3.3V TTL standard unterminated, and are capable of delivering in excess of 25 mA per bit/channel.

Timing characteristics

The PulseBlasterUSB's timing controller accepts an on-board crystal oscillator (or externally applied 3.3V TTL clock signal) of 50 MHz. The innovative architecture of the timing controller allows the processing of either simple timing instructions (delays of up to 2^{32} –1 clock cycles, e.g., approximately 43 seconds at 100 MHz), or double-length timing instructions (up to 2^{52} –1 clock cycles, equivalent to 4.5×10^7 seconds, or approximately 521 days, at 100 MHz). Regardless of the type of timing 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 very short minimum delay cycle – only five clock periods. This translates to a 50 ns shortest pulse/delay at 100 MHz.

Instruction set

The PulseBlasterUSB's instruction set allows for highly flexible program flow control. The core microcontroller allows for programs to include branches, subroutines, and loops at up to 8 nested levels – all of this to assist the user in creating dense pulse programs that can cycle through repetitious events, such as those required for multidimensional spectroscopy and imaging applications.

External triggering

The PulseBlasterUSB can be triggered and/or reset externally via dedicated hardware lines. These two hardware lines, Hardware Trigger and Hardware Reset, are pulled high via 10 kOhm resistors and the required control signals are "low-true." This means the control signals will become true on the falling edge and remain true while held at ground. Trigger occurs when the trigger pin is shorted to ground. To protect hardware, do not exceed the recommended voltage levels.

Summary

The PulseBlasterUSB is a versatile, high-performance, programmable TTL signal generator which operates at 100 MHz and is capable of generating pulses/delays/intervals ranging from 50 ns to 4.5×10^7 s in duration. The PulseBlasterUSB has 24 output channels, features highly flexible program flow control, and can accommodate pulse programs containing as many as 4096 instructions.

Each of the PulseBlasterUSB's 24 outputs is independently controlled, has a standard TTL 3.3 V output voltage[1], and can provide in excess of 25 mA of current. User interaction and control can be accomplished in several ways: through SpinCore's *PulseBlaster Interpreter* and *PulesBlaster.NET*, which are free graphical programming environments for Windows, or through the SpinAPI control library which is also free of charge and can be integrated into C/C++, MATLAB, LabVIEW and other third-party design environments.

[1] No-load voltage level

Specifications

Output Channels

- 24 individually controlled digital output channels (LVTTL levels, 3.3 V unterminated high state)
- Variable pulses/delays for every output channel
- 25 mA output current per output channel

Pulse Timing

- 50 ns shortest pulse/interval (at 100 MHz)
- 4.5×10^7 s longest pulse/interval (at 100 MHz)
- 10 ns pulse/interval resolution (at 100 MHz)
- TTL-level (3.3 V max.) external trigger and reset inputs

Pulse Program Flow Control

- Subroutines, nested up to 8 levels deep
- Loops, nested up to 8 levels deep
- 20-bit loop counters (max. 1,048,575 repetitions per loop)
- Wait for trigger with 8 clock cycle latency (80 ns at 100 MHz), can wait up to 4.5×10^7 seconds (~521 days)
- Status polling via hardware and software
- Up to 4096 instructions

III. Installation

Installing the SpinAPI Driver and Control library

SpinAPI is a custom Application Programming Interface developed by SpinCore Technologies, Inc. for use with the PulseBlaster board. SpinAPI can be utilized using C/C++ or graphically by using other methods detailed in section IV.

NOTE: To ensure proper installation of the board, you must uninstall any previous versions of SpinAPI and complete the following:

- Please do not connect the system/ board until SpinAPI has been installed.
- Install the latest version of SpinAPI found at: http://spincore.com/support/spinapi/. Both 32-bit and 64-bit versions are available.
- After installation, shut down the computer, plug the PulseBlasterUSB's USB cable into the computer and then restart.

Now you are ready to run the test C programs that came in the SpinAPI Package. These example files can be found in the “examples” folder that came with the SpinApi Package. Open the “PulseBlaster24” sub-folder to access the pre-compiled examples.

Testing the PulseBlaster

The simplest way to test whether the PulseBlasterUSB has been installed properly and can be controlled as intended is to run a simple test program. The pb24_ex1.exe program, for example, will cause the outputs to turn on and off with a period of 400ms. To test the board, run pb24_ex1.exe and observe each digital output with an oscilloscope or with LEDs..

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 Figure 2 and Figure 3 below).

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

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

Figure 4 shows what you should expect to see on your oscilloscope when running pb24_ex1.exe and using a 50 Ohm resistor to terminate the line. Once this behavior has been verified, the user can be confident the board is installed properly.

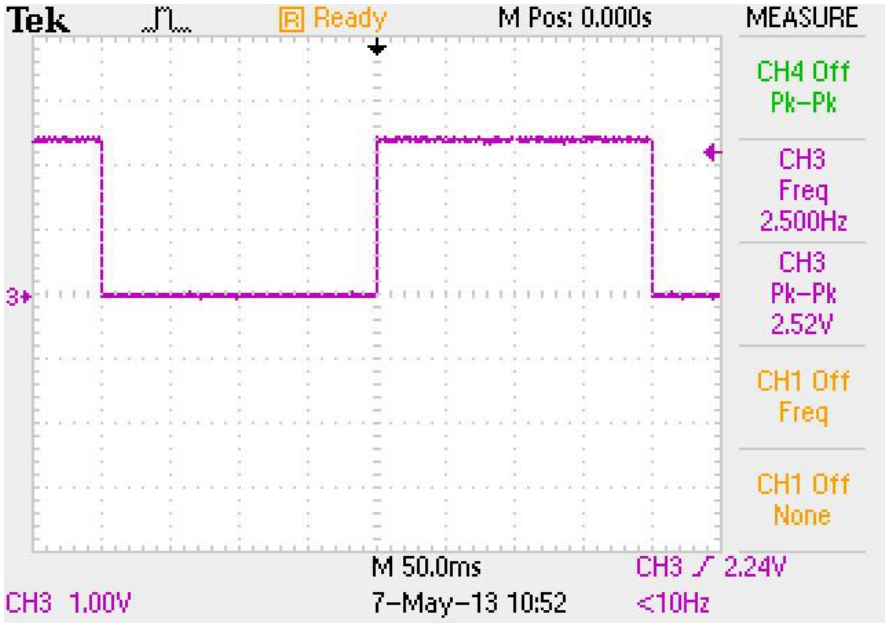


Figure 4: The expected signal from a PulseBlasterUSB running example program pb24_ex1.exe. Notice the amplitude of 2.52 V is due to a 50 Ohm resistor which is terminating the line.

IV. Programming the PulseBlasterUSB

Introduction

There are five major methods of programming the PulseBlasterUSB. The simplest and most intuitive methods of control are the graphical interfaces provided by *PulseBlaster Interpreter* and *PulseBlaster.NET*. More advanced control can be exercised by writing custom C/C++ programs which will interact with the SpinAPI control package.

Programming can also be accomplished through the use of third-party GUI packages such as LabVIEW and MATLAB which have the ability to access Dynamic Link Libraries (DLLs) like SpinAPI. All five programming methods are described in detail below.

Programming with the PulseBlaster Interpreter

The PulseBlasterUSB can be programmed and controlled via the *PulseBlaster Interpreter*, which is an easy to-use editor that allows you create, edit, save and run your pulse sequence.

With the PulseBlaster Interpreter, pulse patterns can be generated using the simple notation of natural numbers (see the first line in Figure 5, below). For more complex bit patterns, the PulseBlaster Interpreter also accepts binary notation and hexadecimal notation (see lines 2, 3, and 4 in Figure 5, below).

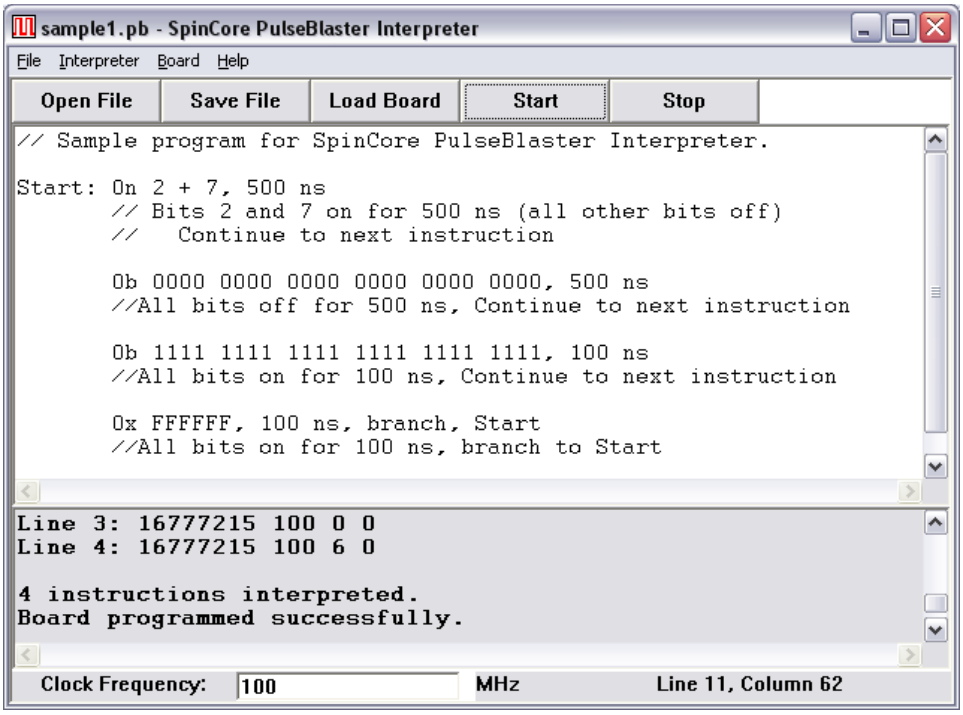


Figure 5: The PulseBlaster Interpreter

Please follow the link http://www.spincore.com/products/spbi/ for the most recent version of the PulseBlaster Interpreter software, installation instructions, documentation, and sample programs.

Programming with PulseBlaster.NET

The PulseBlasterUSB can be programmed and controlled using an intuitive .NET GUI designed in C#.

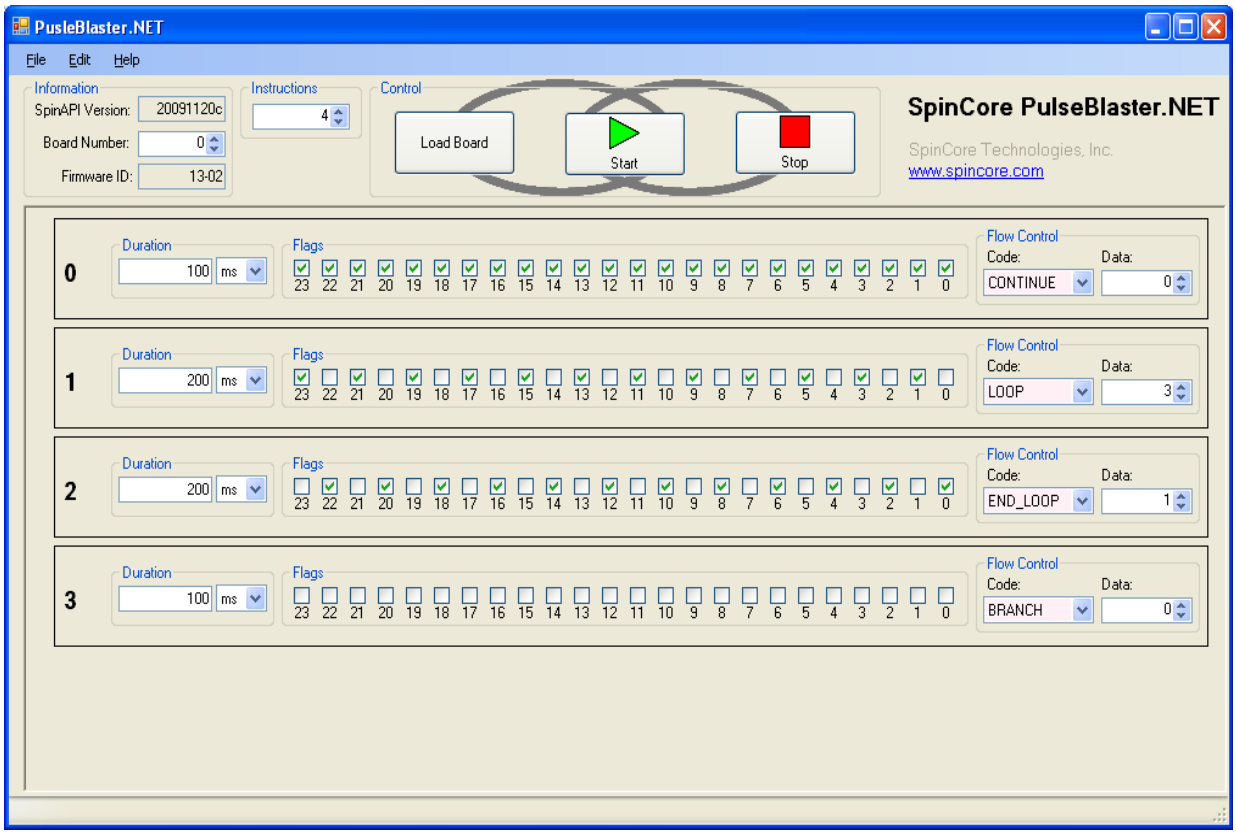


Figure 6: The PulseBlaster.NET Interface

The PulseBlaster.NET interface provides a quick and intuitive method for designing pulse programs for your device. Programming is no longer done in text, but instead presented as a simple visual metaphor. Instructions are presented as a series of checkboxes to represent the TTL values produced.

PulseBlaster.NET is capable of saving and loading files that may be transferred between computers. Best of all, any programming errors (e.g. invalid opcode) are indicated to the user immediately, eliminating much debugging work.

The latest version of the program and supporting documentation is available on our website at http://www.spincore.com/support/net/.

Programming in C/C++ using SpinAPI

The SpinAPI control library allows users to write pulse programs for controlling the PulseBlasterUSB with C/C++ functions. The SpinAPI package also contains several precompiled example pulse programs, along with their source code. These example programs provide an easy way for the user to verify the proper operation of the device and will also provide the user with a jump-start in programming with C/C++ and SpinAPI.

Programming the PulseBlasterUSB in C/C++ uses a simple paradigm in which each generated interval corresponds to a single instruction. Every instruction is programmed with the state of the output channels for that interval, the duration (time) and the program flow information (op code and data). An example is shown below.

```
int start = pb_inst(0xFFFFFF, CONTINUE, 0, 500*ms);
            pb_inst(0x000000, BRANCH, start, 500*ms);
```

Example 1: Code segment to produce a 1Hz square wave on all outputs. The first parameter of pb_inst corresponds to the state of the outputs; the second and third, the op code and data; and the fourth to the duration of the interval.

Use of this programming paradigm is explained in further detail in the document titled “Using SpinAPI in C/C++ for PulseBlaster Programming” which is available from the SpinCore website at http://spincore.com/support/spinapi/using_spin_api_pb.shtml.

The SpinAPI package is available for download free of charge. Detailed explanations of every available API function are available on-line, and the source code of the entire SpinAPI package is also freely available. This may be of value to those users wishing to port the package to other operating systems or platforms. For more information about the SpinAPI package, see http://spincore.com/support/spinapi/.

With the pre-configured compiler that is available on SpinCore's website, changing one of the example programs and running it with the PulseBlasterUSB is as easy as clicking the “Rebuild All” button on the compiler's GUI (see Figure 4). You can download the pre-configured compiler from the SpinCore website at the address http://www.spincore.com/support/spinapi/ by clicking on **Download the Pre-configured Compiler for writing and modifying pulse programs**.

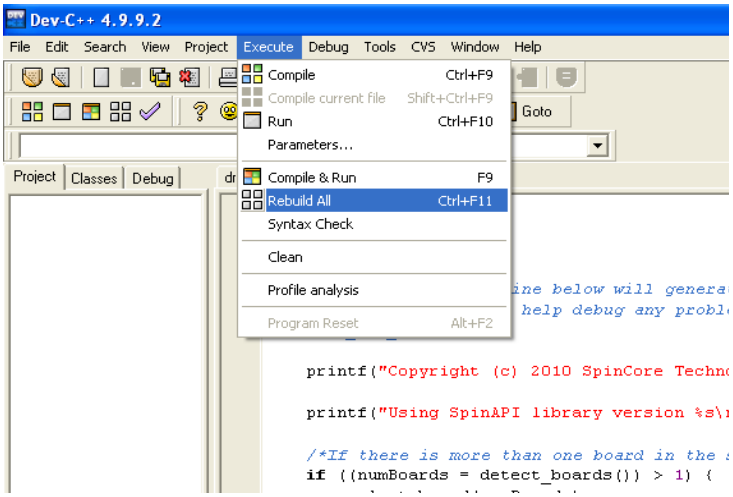


Figure 7: Using the Pre-Configured Compiler

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 to control.

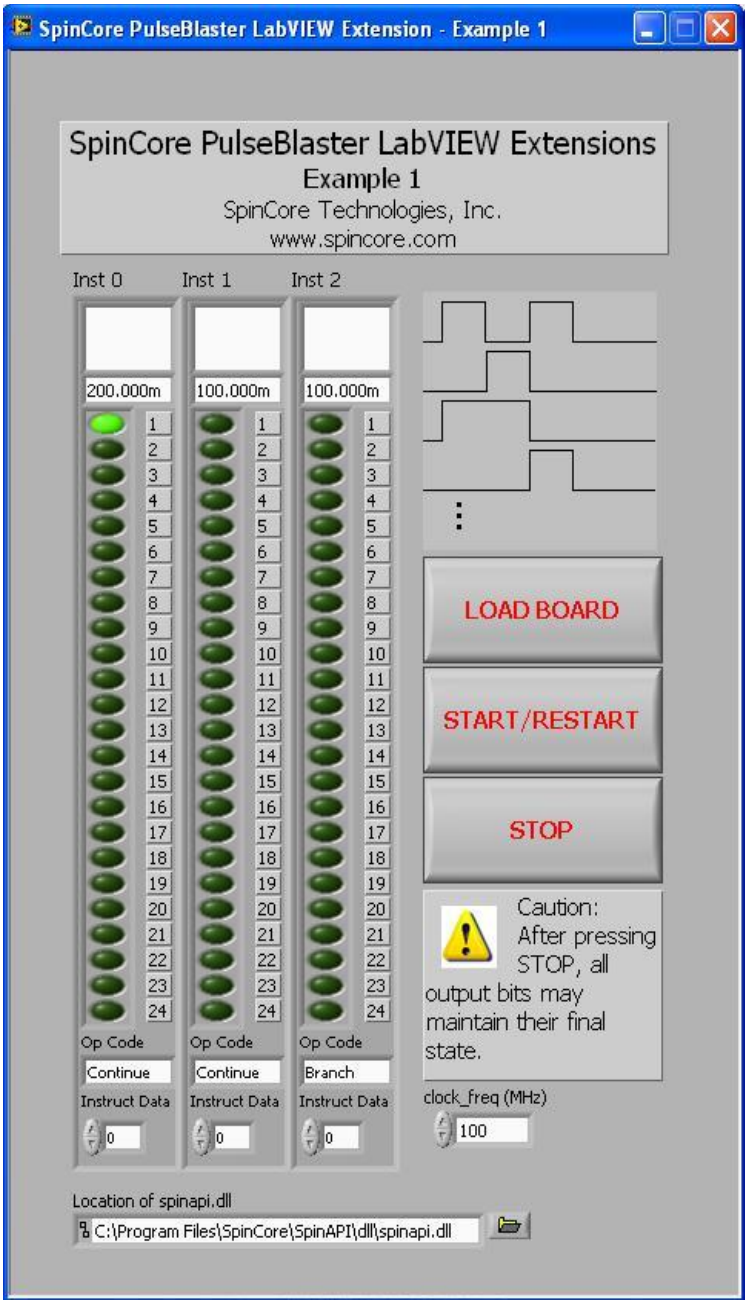


Figure 8: Example of PulseBlaster LabVIEW Extensions User Interface. The example shown has three instructions that toggle TTL bit 1 on for 200 ms and off for 200 ms.

There are two versions of the LabVIEW extensions available free of charge on our 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 5 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/.

Programming with Third-Party GUI Environments

The SpinAPI control library is a standard Windows Dynamic Link Library (DLL), and virtually all programming languages and software environments (including LabVIEW and MATLAB) provide mechanisms for accessing the functionality of libraries like SpinAPI.

Using SpinAPI with Linux

SpinAPI can be used with Linux on x86 processors. To get your programs running on Linux:

1. In the spinapi_source/ directory, type "make linux-direct". This will create a file called libspinapi.a which you can link your programs against in order to use SpinCore hardware. Note that this is not the same as the file in the dll/ directory. The dll/ directory contains files for use only with Windows and its contents are therefore not usable on Linux.
2. To compile your program, you must link with both the SpinAPI and math libraries. For example, when compiling the "pb24_ex1.c" example program into an executable “pb24_ex1”, you would use a command similar to Example 2 (assuming that both libspinapi.a and pb24_ex1.c are in the current directory).
3. When running programs, you MUST have superuser privileges (i.e., be running as root). This is because the SpinAPI library needs to be able to gain access to the system's low-level hardware resources.

```
gcc -opb24_ex1 pb24_ex1.c -L. -lspinapi -lm -lusb
```

Example 2: Compiling SpinAPI example pb24_ex1.

V. Connecting to the PulseBlasterUSB

The PulseBlasterUSB is available in four standard options: a) the board-only option, no enclosure (+5 V DC required), b) the single-bay CD-ROM-type enclosure(AC Power Supply included), c) the 2U BNC/TNC enclosure (½ width), and d) the 2U BNC Rackmount Enclosure. The connectors for each option are explained below.

Connector Information for PulseBlasterUSB board

Power Connector

The PulseBlasterUSB board-only option has a 4-pin Molex-style connector for supplying power. The pin and signal arrangements for this connector is as follows:

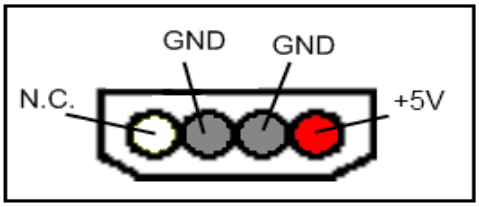


Figure 9: 4 Pin input connector (Molex part 0531090410).

This power connector mates with standard PC power supply connector or Molex part 0015244048.

Note that it is suggested that a PC power supply with the connector specified below is used. This is a standard connector type common on most PCs that will satisfy the power requirements and prevent any damage (such as accidentally reversing polarity). The PulseBlasterUSB board does not contain overvoltage protection, reverse polarity protection, or undercurrent detection. If a PC power supply is not easily accessible it is suggested that a fixed-voltage +5 V power supply with a 3 A maximum current be used.

JP303 IDC Header for PulseBlasterUSB board (Main Output Connector)

The TTL signals are exposed on a shrouded 50-pin male IDC connector (JP303). The shrouded IDC connector labeled JP303 can also be accessed using an SP31 board (Figure 19 on page 31) 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 be identified with a square pin.

Pin Number	Function
2	Bit 0
4	Bit 1
6	Bit 2
8	Bit 3
10	Bit 4
12	Bit 5
14	Bit 6
16	Bit 7
18	Bit 8
20	Bit 9
22	Bit 10
24	Bit 11
26	Bit 12
28	Bit 13
30	Bit 14
32	Bit 15
34	Bit 16
36	Bit 17
38	Bit 18
40	Bit 19
42	Bit 20
44	Bit 21
46	Bit 22
48	Bit 23
50	NC

Table 1: Pin out for 50 pin IDC Connector.

Note: All other pins are ground.

JP301 IDC Header (Trig/Res/Stat) for PulseBlasterUSB board

For PulseBlasterUSB boards that are not fully enclosed, the trigger, reset and status pins are available on the IDC Header labeled JP301 in addition to a DB9 connector. The pinout for the IDC header is shown in Figure 10 and Table 2 below. Please refer to the **DB9 Connector (Trig/Res/Stat)** section on page 20 for more details about the signals and the DB9 pinout.

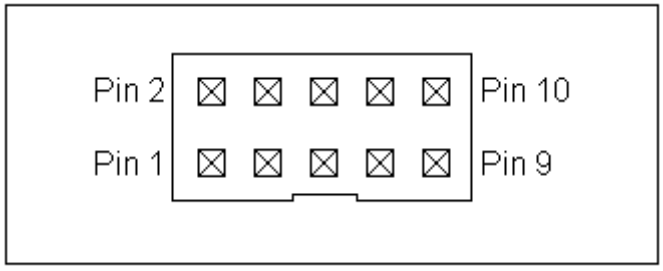


Figure 10: Shrouded IDC Header JP301.

Pin Number	Function
1	Ground
2	Hardware Trigger
3	Ground
4	Hardware Reset
5	Ground
6	WAITING
7	Ground
8	RUNNING
9	STOPPED
10	RESET

Table 2: IDC Header JP301 Pinout.

Connector Information for Single-bay Enclosure

AC Power Connector

The PulseBlasterUSB system in the single-bay enclosure is provided with an internal auto-sensing universal AC power supply. An IEC C14 male connector is exposed at the rear of the enclosure which may be connected to 120/240 V, 50/60 Hz mains power.

DB9 Connector (Trig/Res/Stat)

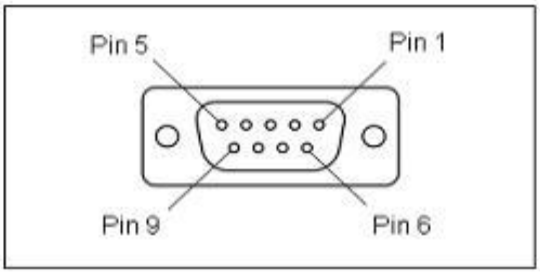


Figure 11: Female DB-9 Output Connector.

Pin Number	Function
1	STOPPED
2	RESET
3	Ground
4	Hardware Reset
5	Hardware Trigger
6	Ground
7	RUNNING
8	WAITING
9	Ground

Table 3: Female DB-9 connector pin assignments.

The DB-9 connector contains both input and output pins: the Hardware Trigger and Hardware Reset control lines (which are inputs), and four status output pins which allow external monitoring of the state of the PulseBlasterUSB. The status bits are defined as follows:

Status Pins Description

Stopped – Driven high when the PulseBlasterUSB device has encountered a STOP Opcode during program execution and has therefore entered a stopped state. The device must be reset before it can start running again.

Reset – In versions up to firmware 13-7, this signal is driven low when the PulseBlasterUSB device is in a RESET state. In versions after 13-7, this signal is driven high when the device is in a RESET state.

Running – Driven high while the PulseBlasterUSB device is executing a program and low when the device enters a reset state or idle state.

Waiting – Driven high when the PulseBlasterUSB device has encountered a WAIT Opcode and is waiting for the next trigger in order to resume operation (trigger may be hardware or software generated).

Hardware Trigger

The hardware trigger is pulled to high voltage (3.3V) on the board and can be triggered by a low pulse (or shorting to ground). When a falling edge is detected (e.g., shorting the pin to ground) 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 input signal for HW_Trigger.* ***Applying voltages to the input pins that are greater than 3.3V or less than 0V will damage the PulseBlasterUSB.***

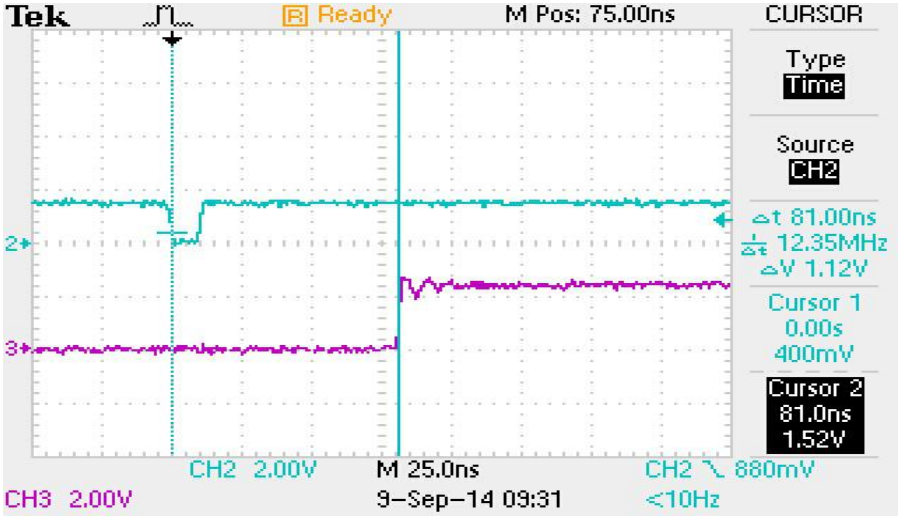


Figure 12: Demonstration of HW_Trigger signal. The blue shows the HW_Trigger signal, the pink shows one of the output flags. A latency of 80 ns is shown in this figure. Please refer to the Instruction Set Architecture section in Appendix I for more details on programming the duration of the WAIT latency. 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. **Caution: applying voltages to the input pins that are greater than 3.3V or less than 0V will damage the board.**

Hardware Reset

The hardware reset signal is pulled to high voltage (3.3V) on the board and can be triggered by a low pulse (or shorting to ground). When a falling edge is detected (e.g., by shorting the pin to ground) the program execution will halt and the controller will reset 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 input signal for HW_Reset.* ***Applying voltages to the input pins that are greater than 3.3V or less than 0V will damage the PulseBlasterUSB.***

Centronics Connector

All input and output connectors are mounted at the back of the PulseBlasterUSB enclosure. The pin layout is shown in Figure 13 and the association between bits and pins is shown in Table 4.

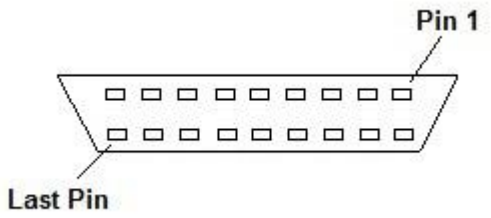


Figure 13: General pin layout for the Centronics connector.

Pin Number	Output Bit
26	0
27	1
28	2
29	3
30	4
31	5
32	6
33	7
34	8
35	9
36	10
37	11
38	12
39	13
40	14
41	15
42	16
43	17
44	18
45	19
46	20
47	21
48	22
49	23
50	NC

Table 4: Centronics connector pin assignments

Note: All other pins are ground.

For mating connectors see Digi-Key part number 1550RMA-ND, T1504A-ND or similar.

Connector Information for 2U BNC/TNC Enclosure

DB9 Connector (Trig/Res/Stat) for 2U BNC/TNC Enclosure

The Trig/Res/Stat DB9 connector information is shown in Figure 14 and Table 5, below. The Hardware Trigger and Hardware Reset are both low-true, so each of these pins would need to be shorted to ground to cause a trigger or reset, respectively. Please refer to the **Status Pins Description**, above, for additional information about each pins functionality.

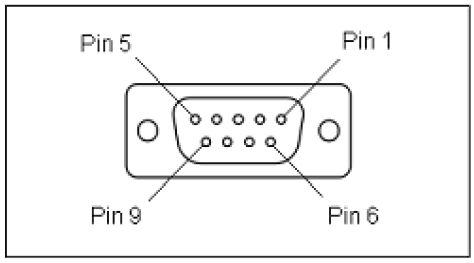


Figure 14: Trig/Res/Stat Female DB9 connector drawing

Pin Number	Function
1	Hardware Trigger
2	Hardware Reset
3	WAITING
4	RUNNING
5	RESET
6	Ground
7	Ground
8	Ground
9	STOPPED

Table 5: Trig/Res/Stat Female DB9 connector pin assignments

Front Panel Connectors for 2U BNC/TNC Enclosure

This enclosure provides 8 output bits through either BNC or TNC connectors, arranged as in Figure 15, below. There is also an external clock signal input provided through a BNC or TNC connector. This clock line is terminated on the PulseBlaster board with a 50 Ω resistor. Please see the **Clock Input Signal Standard** section below for more details. Because the 2U enclosure may be configured with either BNC or TNC connectors, please specify which configuration you require when ordering.

Clock Input Signal Standard

- External clock input available on 2U enclosure (LVTTL level, 3.3V unterminated high state).
- Must be positive voltage only; sinusoidal inputs must be converted to LVTTL levels.
- The clock input connection is internally terminated with a 50 Ω resistor.
- Clock input must be 50 MHz. Please inquire with SpinCore for custom input frequencies.

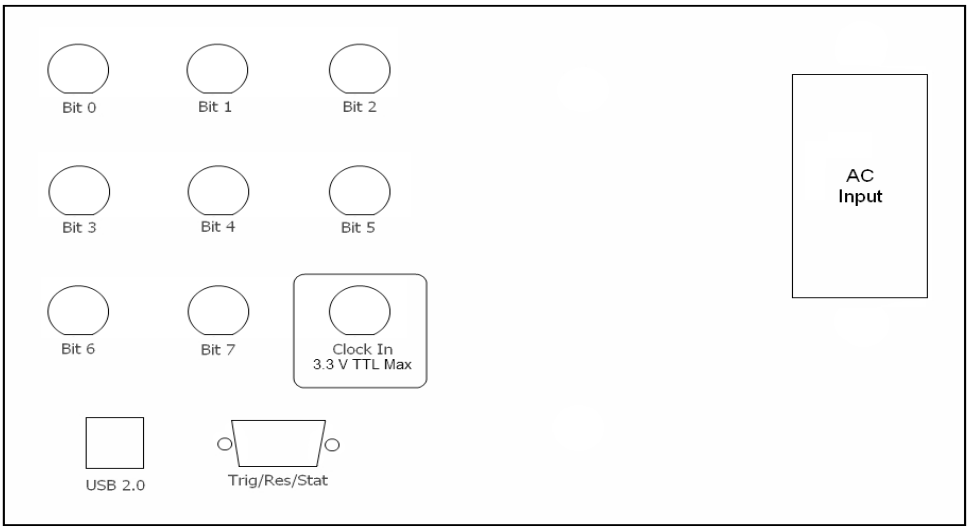


Figure 15: Front-panel connectors for 2U BNC/TNC enclosure

Connector Information for 2U BNC Rackmount Enclosure

Connectors for 2U BNC Rackmount Enclosure

This enclosure provides 24 output bits through BNC connectors. The front panel also contains the USB connector, the Trig/Res/Stat DB9 connector, and the ON/OFF toggle switch which powers on the PulseBlasterUSB. They are arranged in a format which is 3 rows of 8 bits. On the reverse side of the Rackmount enclosure is the AC Input.

DB9 Connector (Trig/Res/Stat) for 2U BNC Rackmount Enclosure

The Trig/Res/Stat DB9 connector information is shown in Figure 16 and Table 6, below. The Hardware Trigger and Hardware Reset are both low-true, so each of these pins would need to be shorted to ground to cause a trigger or reset, respectively. Please refer to the **Status Pins Description**, above, for additional information about each pins functionality.

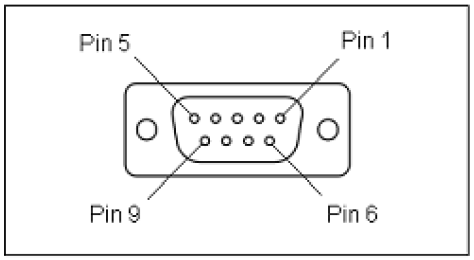


Figure 16: Trig/Res/Stat Female DB9 connector drawing

Pin Number	Function
1	Hardware Trigger
2	Hardware Reset
3	WAITING
4	RUNNING
5	RESET
6	Ground
7	Ground
8	Ground
9	STOPPED

Table 6: Trig/Res/Stat Female DB9 connector pin assignments

Appendix I. Instruction Set Architecture

Instruction Set Architecture of the PulseBlaster Processor

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 into on-board memory varies by model.

The execution time of instructions can be varied and is controlled by one of the fields of the instruction word – the shortest execution time is five clock cycles of the reference clock oscillator and the longest is 2^32 - 1 clock cycles (for standard instructions). All instructions have the same format and bit length, and all bit fields must be filled. Figure 17 shows the fields and bit definitions of the 80-bit instruction word.

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

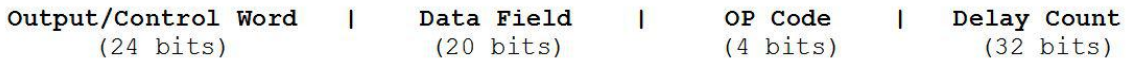


Figure 17: Bit definitions of the 80-bit instruction/memory word

Breakdown of 80-bit Instruction Word

The 80-bit instruction word 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

The first 24 bits of the 80-bit machine word of the PulseBlaster processor are the output bits. The 24 output bits retain their value throughout the entire duration of the instruction represented by the machine word.

Data Field and Op Code

Table 7 (below) describes the available operational codes (referred to as op codes) and their associated data fields. Note that the data field's function is dependent on the op codes.

Op Code #	Instruction	Data Field	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.
2	LOOP	Number of desired loops. This value must be greater than or equal to 1.	Specifies the beginning point of a loop. The Data Field is used to specify number of loops. Execution will continue to next instruction.
3	END_LOOP	Address of beginning of loop	Specifies the 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 the beginning of the desired subroutine
5	RTS	Ignored	Return from subroutine. Execution continues with the instruction following the previous JSR
6	BRANCH	Address of next instruction	Program execution continues with the instruction at the specified address
7	LONG_DELAY	Desired multiplication factor for the “Delay Count” field. Must be greater than or equal to 2.	Used to execute very long instructions/ intervals. Data Field specifies a multiplier of the Delay Field. Execution continues to next instruction.
8	WAIT	Ignored	Program execution stops and waits for hardware trigger. Execution continues to next instruction after receipt of trigger. The exit latency is equal to the delay value entered in the WAIT instruction line plus a fixed delay of 6 clock cycles. The WAIT Op-code may not be used by the first instruction in memory.

Table 7: Op Code and Data Field Description

The execution time of instructions is determined by the content of the Delay field (and the Data field, when applicable), and the output state remains constant during the entire instruction.

NOTE : The following two exceptions apply:

(1) The processor will ignore the delay field of instructions containing the STOP command. Upon encountering an instruction containing the STOP command, the processor will halt execution immediately rather than after the delay specified in the Delay Count field.

(2) The WAIT OpCode may not be used by the first instruction in memory. If you wish to have the processor wait for an external trigger at the beginning of your pulse routine, we suggest that you use two instructions: one with the CONTINUE Op-Code (and with a very short delay) followed by an instruction which uses the WAIT Op-Code.

NOTE : The behavior of the output flags after a STOP command is firmware dependent. With firmware version 13-12, the output flags hold the value given to them in the previous instruction. If a different behavior is desired, please contact SpinCore Technologies, Inc. for alternate firmware options.

Delay Count

The value of the Delay Count field (a 32-bit value) determines the length of time that the current instruction should be executed. The number in this field represents the number of periods of the master clock oscillator that will occur before the execution of the next instruction begins.

Standard time units should be entered into the Delay Field when using the SpinAPI package or the PulseBlaster Interpreter. The allowed units are *ns* (nanoseconds), *us* (microseconds), and *ms* (milliseconds). SpinAPI and the PulseBlaster Interpreter will automatically convert these values into the appropriate number of clock cycles when the PulseBlasterUSB is programmed.

Appendix II. User Programmable Clock Output Design

The PulseBlaster USB design 13-9* features user programmable clock outputs on the pins that would normally have the top 4 flag bits. The Clock Output Channels 0-3 can output pulse patterns such as those seen in Figure 18. Once the clocks are programmed, they remain running at the same rate until they are reprogrammed or the board is turned off. The user can program the period, the clock high time, and the offset as shown in Figure 18. This feature runs on a 50 MHz clock and has 32-bit resolution, meaning the shortest pulse possible is 20 ns and the longest interval is 85 seconds with a resolution of 20 ns.

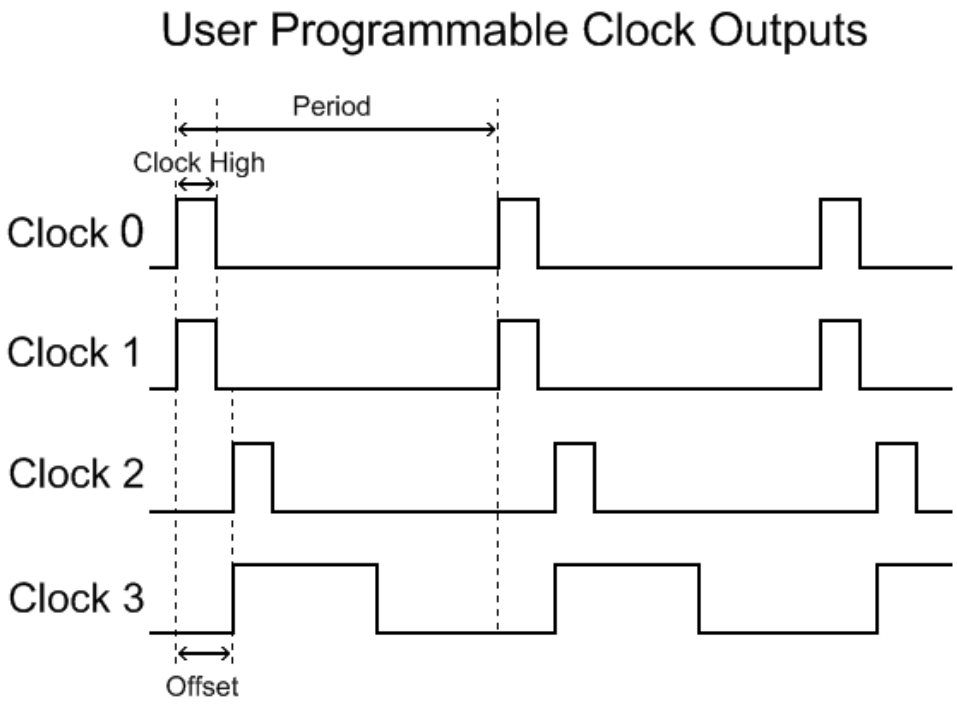


Figure 18: Timing Diagram for User Programmable Clock Outputs

Programming the Clock Feature in C/C++

To program the three parameters into your board, you must use a function in SpinAPI called pb_set_pulse_regs. The function takes four parameters:

pb_set_pulse_regs(unsigned int channel, double period, double clock_high, double offset)

Channel must be an integer between 0 and 3 corresponding to one of the four channels. Period can be as long as 85 seconds or as short as 40 ns. Clock High and Offset must be between 20 ns and 85 seconds. This function needs to be called either before pb_start_programming(PULSE_PROGRAM) or after pb_stop_programming(); it can't be included with a pulse program. Once the function is run, the clock is immediately changed.

*Please contact SpinCore directly about purchasing the PulseBlaster USB board with 13-9 design.

Since the programmable clock outputs are controlled by a 50 MHz clock, only increments of 20 ns can be programmed to the board. An input value that is not a multiple of 20 ns will be rounded down to the closest multiple of 20 ns that is less than it.

Example Use of Programmable Clock Feature in C/C++

There is a sample program for this functionality included with the SpinAPI examples. It uses the code seen in Example 3 to produce outputs on all channels with equal period and clock high times but with varying delays.

```
pb_set_pulse_regs(0, 200.0 * us, 20.0 * us, 20.0 * us);
pb_set_pulse_regs(1, 200.0 * us, 20.0 * us, 40.0 * us);
pb_set_pulse_regs(2, 200.0 * us, 20.0 * us, 60.0 * us);
pb_set_pulse_regs(3, 200.0 * us, 20.0 * us, 80.0 * us);
```

Example 3: Programming the clock outputs.

Related Products and Accessories

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

2. Firmware version 13-9 – Alternate firmware for the PulseBlasterUSB which features user programmable clock outputs on the pins that would normally have the top 4 flag bits. Please see Appendix II. User Programmable Clock Output Design for more details.

3. 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/

4. If you require an Oven Controlled Clock Oscillator (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

5. SpinCore MMCX Adapter Board – This adapter board allows easy access to the individual bits of the PulseBlaster. MMCX cables are available upon request. For ordering information contact SpinCore at http://www.spincore.com/contact.shtml

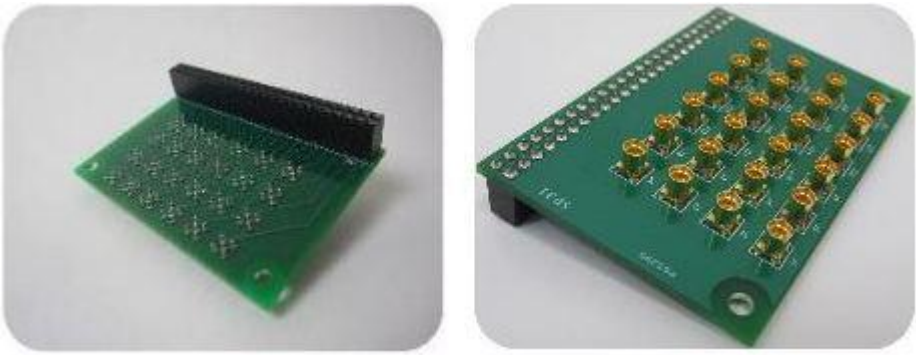


Figure 19: MMCX Adapter Board allows easy access to individual bits. Left: IDC Connector. Right: MMCX Connectors.

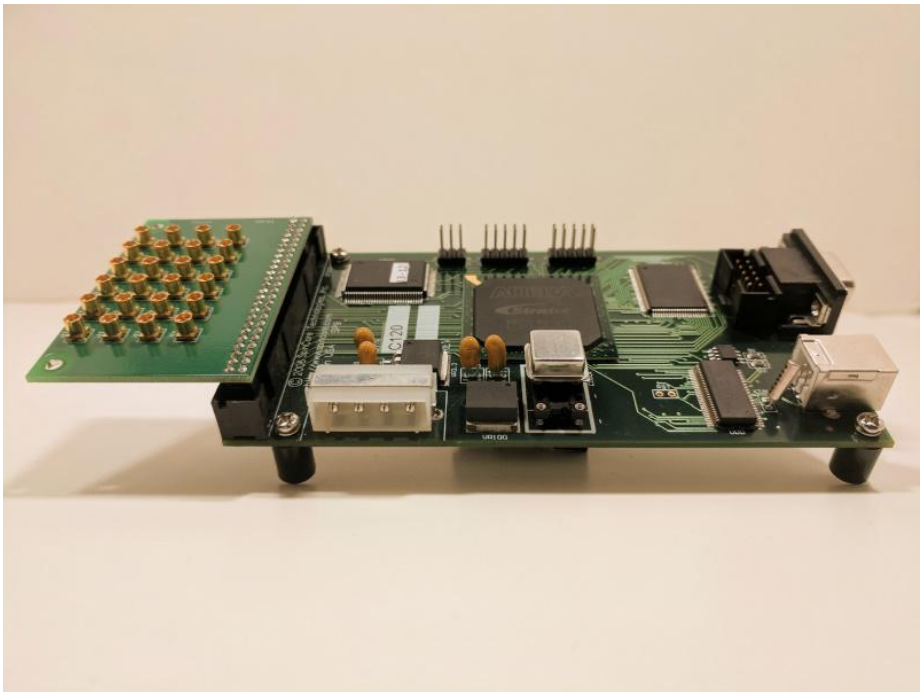


Figure 20: MMCX Adapter Board orientation and configuration with PulseBlasterUSB Board.

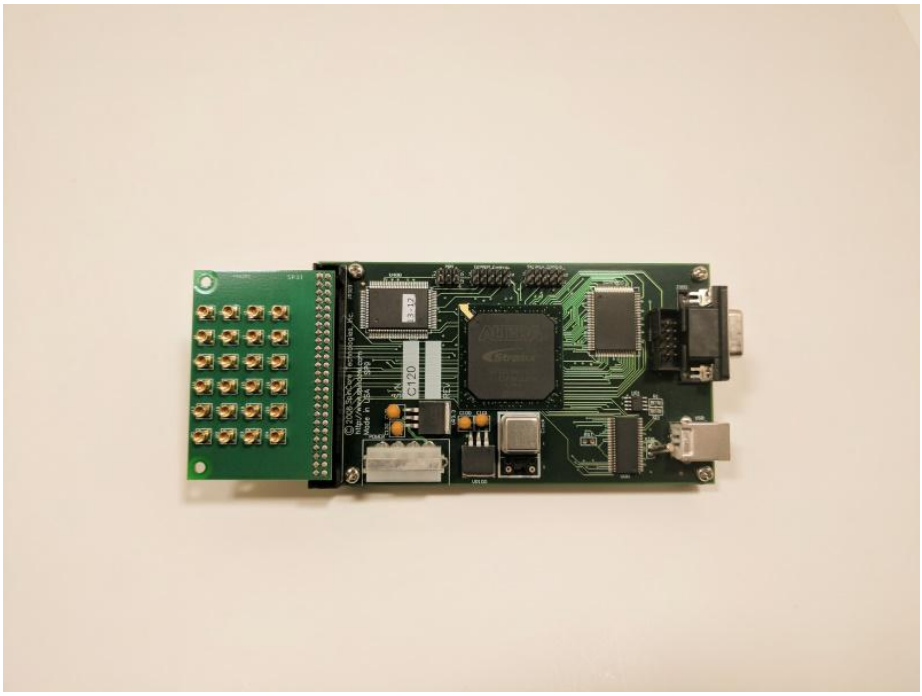


Figure 21: MMCX Adapter Board orientation and configuration with PulseBlasterUSB Board from a different angle.

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

Revision history of this document is maintained at SpinCore.