

# PulseBlasterUSB™ (v2)

## Owner's Manual

---

---

SpinCore Technologies, Inc.  
[www.spincore.com](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.**

---

© 2010 SpinCore Technologies, Inc. All rights reserved.

SpinCore Technologies, Inc. reserves the right to make changes to the product(s) or information herein without notice. PulseBlasterUSB™, PulseBlaster™, SpinCore, and the SpinCore Technologies, Inc. logos are trademarks of SpinCore Technologies, Inc. All other trademarks are the property of their respective owners.

SpinCore Technologies, Inc. makes every effort to verify the correct operation of the equipment. This equipment version is not intended for use in a system in which the failure of a SpinCore device will threaten the safety of equipment or person(s).

# Table of Contents

<b><u>I. Introduction.....</u></b>	<b><u>5</u></b>
Product Overview .....	5
<b><u>II. System Architecture and Specifications.....</u></b>	<b><u>6</u></b>
<b><u>System Architecture .....</u></b>	<b><u>6</u></b>
<i>Output signals.....</i>	<i>7</i>
<i>Timing characteristics.....</i>	<i>7</i>
<i>Instruction set.....</i>	<i>7</i>
<i>External triggering .....</i>	<i>7</i>
<i>Summary.....</i>	<i>7</i>
<b><u>Specifications.....</u></b>	<b><u>8</u></b>
<i>Output Channels.....</i>	<i>8</i>
<i>Pulse Timing.....</i>	<i>8</i>
<i>Pulse Program Flow Control.....</i>	<i>8</i>
<b><u>III. Installation.....</u></b>	<b><u>8</u></b>
Installing the SpinAPI Driver and Control library.....	8
<b><u>IV. Programming the PulseBlasterUSB.....</u></b>	<b><u>9</u></b>
Programming with the PulseBlaster Interpreter.....	9
Programming with .NET GUI.....	10
Programming in C/C++ using SpinAPI.....	10
Programming with Third-Party GUI Environments.....	11
Using SpinAPI with Linux.....	11
<b><u>V. Connecting to the PulseBlasterUSB.....</u></b>	<b><u>12</u></b>
Power Requirements.....	12
Connector Information.....	12
<i>Power Connectors .....</i>	<i>12</i>
<i>Centronics 50-pin connector (TTL).....</i>	<i>13</i>
<i>DB9 Connector (Trig/Res/Stat).....</i>	<i>14</i>

# *PulseBlasterUSB (v2)*

[JP301 IDC Header \(Trig/Res/Stat\).....16](#)

## [Appendix I. Instruction Set Architecture.....17](#)

[Instruction Set Architecture of the PulseBlaster Processor.....17](#)

[Machine-Word Definition.....17](#)

[Breakdown of 80-bit Instruction Word.....17](#)

[Output Pattern and Control Word.....17](#)

[Data Field and Op Code.....18](#)

[Delay Count.....19](#)

## [Appendix II. Controlling the PulseBlasterUSB with SpinAPI.....20](#)

[About SpinAPI.....20](#)

[Using SpinAPI Functions to Program the PulseBlasterUSB.....20](#)

[Example Use of SpinAPI Functions in C/C++.....22](#)

## [Contact Information.....23](#)

## I. Introduction

### Product Overview

---

The PulseBlasterUSB™ is a programmable multichannel pulse generator that is capable of generating pulses and delays ranging from 50 ns to  $4.5 \times 10^7$  seconds in length (with a 100 MHz input clock) . 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 (e.g., 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*, SpinCore's graphical programming environment for Windows, allows users to easily create, edit, save and run pulse programs.

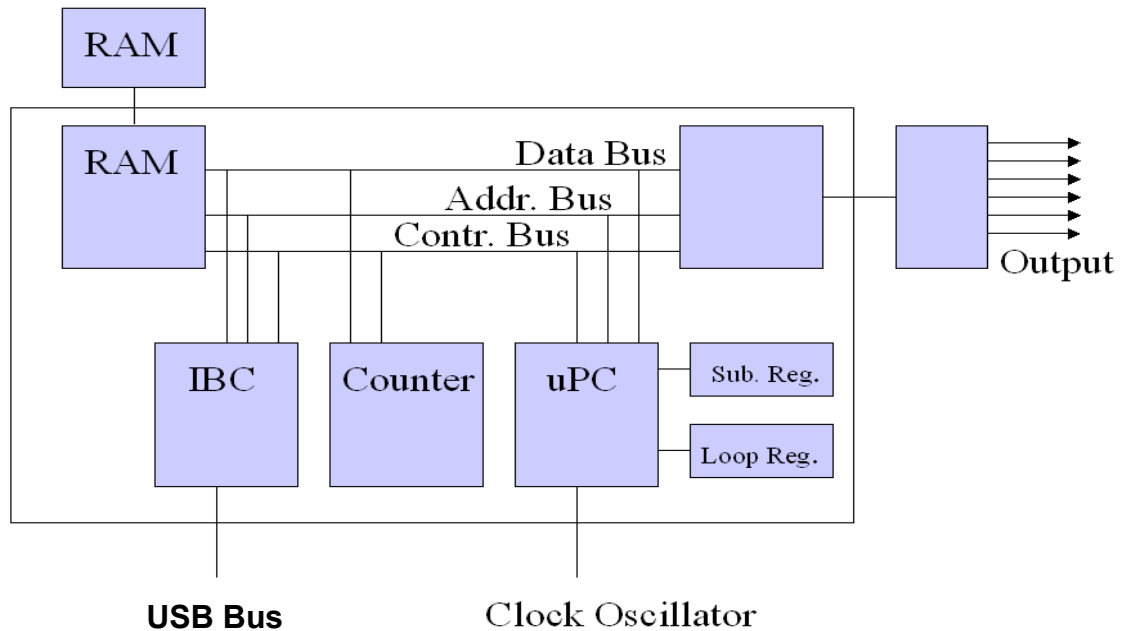
Development environments like LabView, Matlab, and C/C++ can be used to interface with the PulseBlasterUSB through the dedicated Application Programming Interface (API) package called SpinAPI, which is available from SpinCore Technologies at no charge. The SpinAPI package can be used with virtually all 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.

## 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 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.

## **Output signals**

The PulseBlasterUSB has 24 digital output signal lines. These signal lines are routed to a female 50-pin Centronics SCSI connector. All output signal connectors are impedance matched to 50 ohm. The 24 individually controlled digital output bits are 3.3V TTL standard, 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 3.3 V TTL clock signal) of up to 100 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  $4.5 \times 10^7$  seconds 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 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 “active low” (short to ground).

## **Summary**

The PulseBlasterUSB is a versatile, high-performance, programmable TTL signal generator which operates at speeds of up to 100 MHz and which is capable of generating pulses/delays/intervals ranging from 50 ns to  $4.5 \times 10^7$  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<sup>1</sup>, 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*, which is a free graphical programming environment for Windows, or through the SpinAPI driver and control library which is also free of charge and can be integrated into C/C++, Matlab, LabView and other third-party design environments. The SpinAPI control package is also compatible with Linux.

---

<sup>1</sup> No-load voltage level

## Specifications

---

### ***Output Channels***

- 24 individually controlled digital output channels (TTL levels, 3.3 V logical “one”)
- Variable pulses/delays for every output channel
- 25 mA output current per output channel

### ***Pulse Timing***

- 50 ns shortest pulse/interval (at 100 MHz)
- $4.5 \times 10^7$  s longest pulse/interval (at 100 MHz)
- 10 ns pulse/interval resolution (at 100 MHz)
- TTL-level 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  $4.5 \times 10^7$  seconds
- Status polling via hardware and software.
- Up to 4096 instructions

## III. Installation

### **Installing the SpinAPI Driver and Control library**

---

The PulseBlasterUSB requires the SpinAPI driver and control library in order to operate. The latest version of the SpinAPI package can be downloaded from: <http://www.spincore.com/CD/Setup/>. This URL contains both the drivers and the most recent installation guide and example programs. Please refer to the document at the URL above for instructions on how to properly install the drivers.

After installation is completed, it is recommended that at least one of the example programs be executed to verify the proper installation of the SpinAPI drivers and to verify the correct operation of the PulseBlasterUSB. For the testing purposes, any of the PulseBlaster PB24 example pulse programs may be executed, as they are all fully compatible with the PulseBlasterUSB. These example programs can also be called from the Windows command line if necessary.

## IV. Programming the PulseBlasterUSB

There are four major methods of programming the PulseBlasterUSB. The simplest and most intuitive method is the use of SpinCore's *PulseBlaster Interpreter*, which is a graphical programming environment specifically developed for controlling SpinCore's PulseBlaster family of products. 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 like LabView and Matlab which have the ability to access Dynamic Link Libraries (DLLs) like SpinAPI. Finally, programming can be accomplished using C/C++ on Linux computers (the PulseBlaster Interpreter is not yet available for Linux). All four 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 2, below). For more complex bit patterns, the PulseBlaster Interpreter also accepts binary notation and hexadecimal notation (see lines 2, 3, and 4 in Figure 2, below).

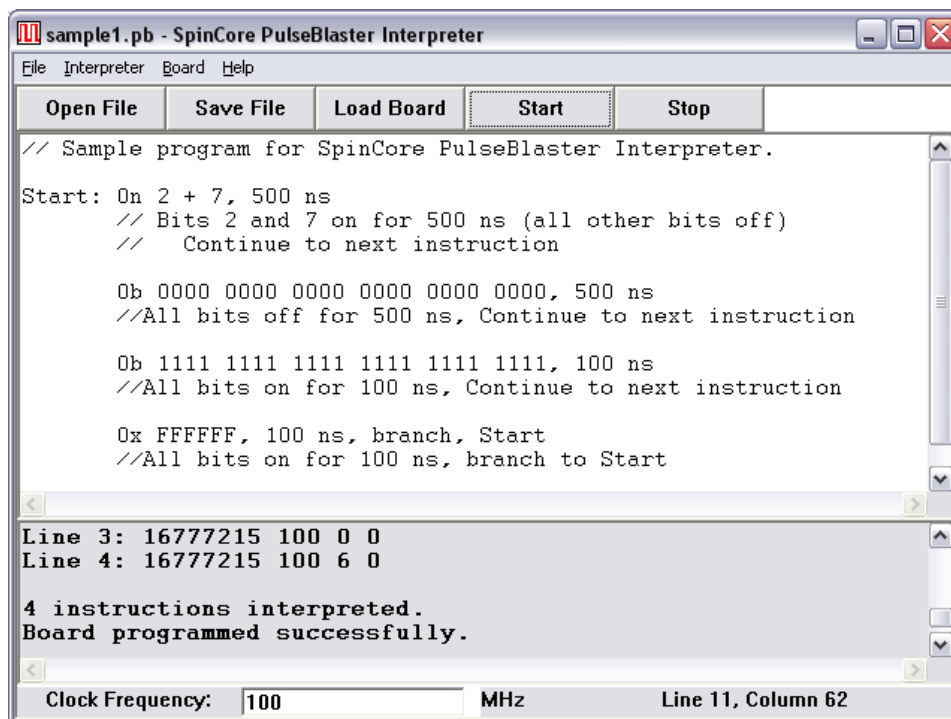


Figure 2: The PulseBlaster Interpreter

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 .NET GUI

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

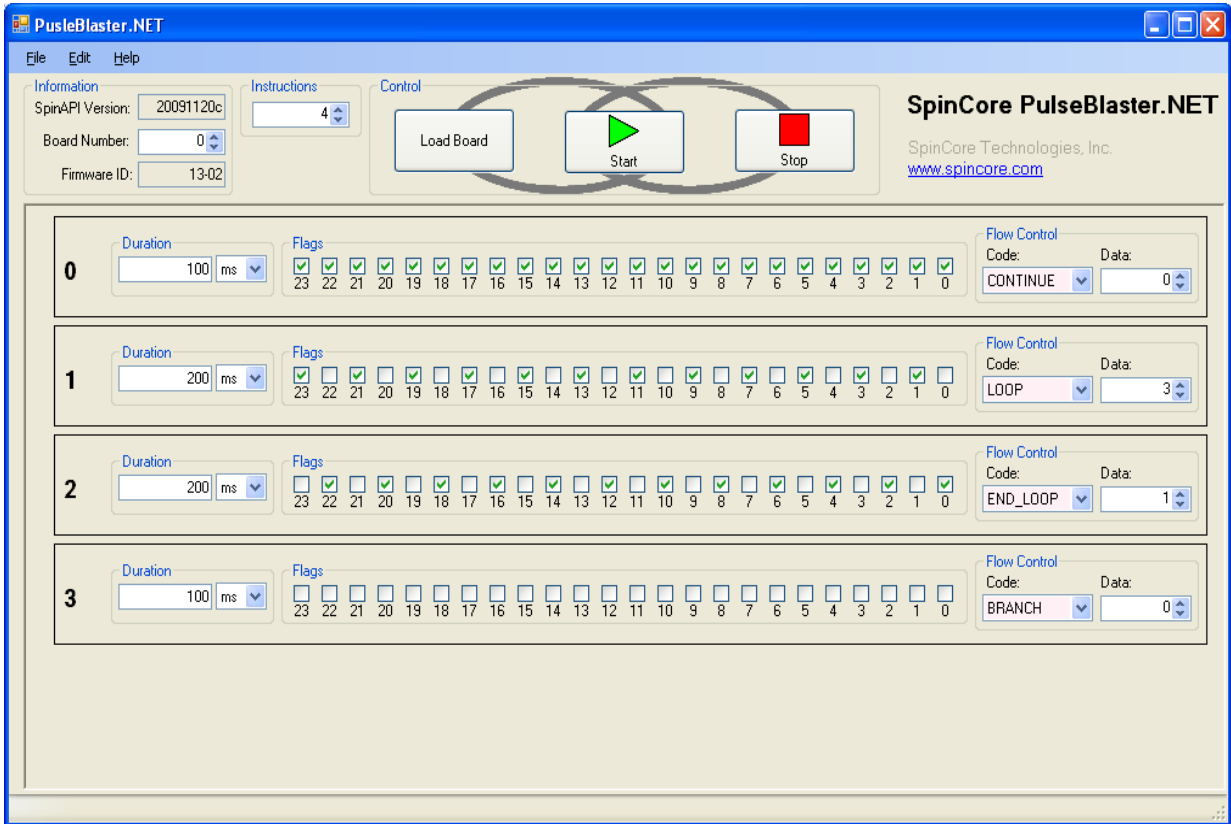


Figure 3: The PulseBlaster .NET GUI

## 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.

One example program is also presented in Appendix II of this manual - *Controlling the PulseBlasterUSB with SpinAPI*. Appendix II also provides a detailed explanation of the program and its resulting output pattern.

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://www.spincore.com/CD/Setup/>

## **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 thing 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, you would use a command like the following one: (assuming that both `libspinapi.a` and `pb24_ex1.c` are in the current directory)

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

This will create an executable called "pb24\_ex1".

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.

## V. Connecting to the PulseBlasterUSB

### Power Requirements

---

Recommended values and maximum currents: +5 V, 3.0 A.

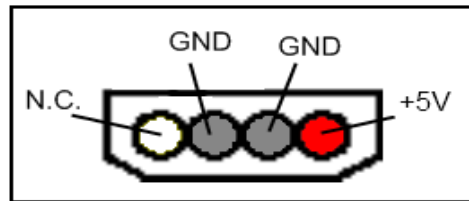
Note that it is suggested that a PC power supply with the connector specified below is used. This is a standard connector type common on most PCs that will satisfy the power requirements and prevent any damage (such as accidentally reversing polarity). The PulseBlasterUSB does not contain overvoltage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

### Connector Information

---

#### *Power Connectors*

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



**Figure 4:** 4 Pin input connector (Molex part 0531090410). Mates with standard PC power supply connector or Molex part 0015244048.

## **Centronics 50-pin connector (TTL)**

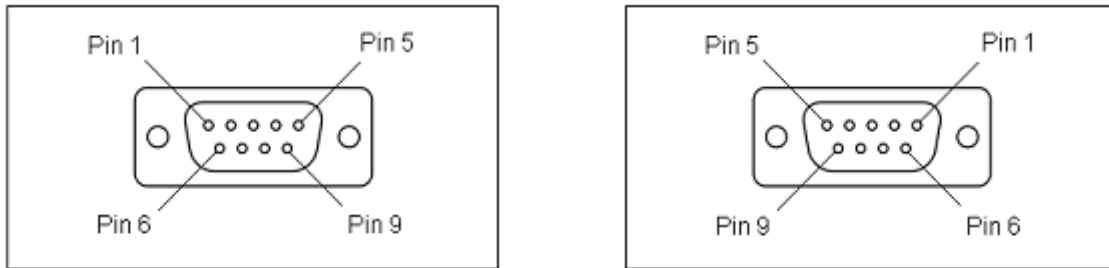
All input and output connectors are mounted at the back of the PulseBlasterUSB enclosure. On the Centronics 50-pin connector, the pins on the top row (1-25) are grounds and the output signals are carried on the pins on the bottom row (26-50). Each pin on the bottom row corresponds to a bit in the output word of an instruction. The association between bits and pins is shown in Table 1.

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 1:** Centronics connector pin assignments

[For mating connectors see Digi-Key part number 1550RMA-ND, T1504A-ND or similar.](#)

## DB9 Connector (Trig/Res/Stat)



**Figure 5:** DB-9 Output Connector. Left: DB-9 Male. Right: DB-9 Female

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

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

**Table 2:** DB-9 connector pin assignments. Left: DB-9 Male. Right: DB-9 Female.

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

**Stopped** – Driven high when the PulseBlasterUSB device has encountered a STOP Opcode during program execution and has therefore entered a stopped state.

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

**Running** – Driven high while the PulseBlasterUSB device is executing a program. This signal is low when the PulseBlasterUSB 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 purpose of the hardware trigger is to bring the PulseBlasterUSB out of the 'idle' state in order to start or resume the execution of a pulse program. If the idle state has been entered by way of the STOP OpCode or by way of hardware or software reset, then applying HW\_Trigger will cause the execution of pulse program to start over from the beginning. When using the STOP OpCode, a software reset or hardware reset must be applied before HW\_Trigger.

When the PulseBlasterUSB encounters a WAIT OpCode in a pulse program, it will wait for a trigger signal before continuing execution. Applying HW\_Trigger when the PulseBlasterUSB is in this waiting state will cause the program to continue to the next instruction.

HW\_Trigger is pulled high using a 10 kohm resistor. When a falling edge is detected (e.g., when this pin is shorted to ground), code execution is triggered.

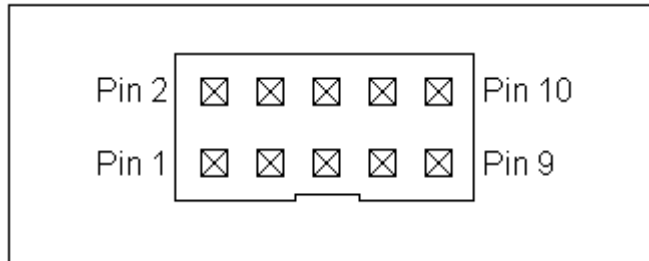
## **Hardware Reset**

The hardware reset line, HW\_Reset, is pulled high by default using a 10 kohm resistor. The execution of a program can be halted by pulling HW\_Reset low (e.g., by shorting the pin to ground). When HW\_Reset is pulled low during the execution of a program, the PulseBlasterUSB resets itself back to the beginning of the pulse program. Program execution can then be initiated by either a software start command or by a hardware trigger.

**NOTE:** *When connecting the HW\_Trigger and HW\_Reset lines to your equipment, please remember that the PulseBlasterUSB expects TTL-level input signals.*

## JP301 IDC Header (Trig/Res/Stat)

For PulseBlasterUSB boards that are not fully enclosed, the trigger, reset and status pins are also available on the IDC Header labeled JP301. The pinout for this header is shown in figure 4 and table 3 below.



**Figure 6:** 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 3:** IDC Header JP301 Pinout.

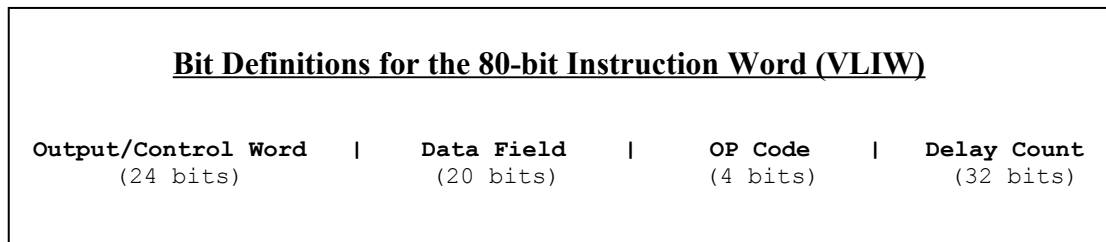
## Appendix I. Instruction Set Architecture

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



**Figure 7:** 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. These 24 signals are routed to the 50-pin Centronics connector as shown in Table 1 in Part V of this manual, entitled *Connecting to the PulseBlasterUSB*.

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 4 (below) describes the available operational codes (hereafter referred to as Op-Codes) and their associated data fields. Note that the data field's function is dependent on the Op-Code.

Op-Code #	Instruction	Data Field	Function
0	CONTINUE	Ignored	Program execution continues to next instruction
1	STOP	Ignored	Stops the execution of program Notes: 1) all TTL outputs remain from previous instruction; 2) the instruction preceding the STOP command must be at least 6 clock cycles in duration
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 4:** 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.

## ***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. Controlling the PulseBlasterUSB with SpinAPI

### About SpinAPI

---

SpinAPI is a control library which allows programs to be written to communicate with the PulseBlasterUSB. The most straightforward way to interface with this library is with a C/C++ program, and the API (Application Programming Interface) definitions are described below in this context. However, virtually all programming languages and software environments (including LabView and Matlab) provide mechanisms for accessing the functionality of standard Dynamic Link Libraries (DLLs) like SpinAPI.

This Appendix contains a list of the most commonly used SpinAPI functions for the PulseBlasterUSB. An example C/C++ program is also included at the end of this Appendix. For more example programs, please download the latest SpinAPI package from <http://www.spincore.com/CD/Setup/>. A complete reference document for the API is also available at that location.

### Using SpinAPI Functions to Program the PulseBlasterUSB

---

In order to use the functions provided in the SpinAPI package, the following files must be in the working directory of your C compiler: the DLL (spinapi.dll), the header file (spinapi.h), and the library file (libspinapi.a for mingw, spinapilibgcc for Borland, and spinapi.lib for MSVC). Some of the most commonly used SpinAPI functions are listed and briefly described below. An example pulse program written in C/C++ follows.

```
int pb_init();
```

Initializes the PulseBlasterUSB, making it ready to begin accepting commands. Initialization must be performed before any other functions can interface with the PulseBlasterUSB, and so this function should be called at the beginning of the pulse program. Returns a negative number in the event of an error and returns 0 upon success.

```
int pb_close();
```

Releases control of the PulseBlasterUSB. Needs to be called as the last command in the pulse program. Returns a negative number in the event of an error and returns 0 upon success.

```
void pb_set_clock(double clock_frequency);
```

Used to tell SpinAPI what the clock frequency of the PulseBlasterUSB is. For example, when using the 100MHz PulseBlasterUSB, the number 100.0 should be entered. Valid units for the clock frequency are MHz, kHz, and Hz. The clock frequency is assumed to be in MHz if no units are entered.

```
int pb_start_programming(int device);
```

Makes the PulseBlasterUSB ready to receive programming information, and should be called after *pb\_init()* and *pb\_set\_clock(...)*. The only valid value for *device* is PULSE\_PROGRAM, which is an internally defined constant. Returns a negative number in the event of an error and returns 0 upon success.

## *PulseBlasterUSB (v2)*

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

Used to send one instruction of a pulse program to the PulseBlasterUSB. Should only be called after *pb\_start\_programming(PULSE\_PROGRAM)* has been called. A negative number is returned in the event of an error, and the instruction number is returned upon success. This function will return -99 if an invalid parameter was passed to it. Instructions are numbered sequentially, starting at 0.

**int flags** – determines the state of the TTL output bits. Valid values are 0x000000 to 0xFFFFFFFF. For example, 0x000010 would turn bit 5 on and turn all other bits off.

**int inst** – determines which type of instruction is to be executed. Please see Table 2 for details.

**int inst\_data** – data to be used by the instruction from the *inst* field. Please see Table 2 for details.

**int length** – duration of this pulse program instruction, specified in nanoseconds (the predefined constants 'us' and 'ms' can be used as multipliers in order to produce longer intervals).

```
int pb_stop_programming();
```

Used to notify the PulseBlasterUSB that programming is complete. Pulse program execution cannot be started if this command has not been invoked. Returns a negative number in the event of an error and returns 0 upon success.

```
int pb_start();
```

Sends a software trigger to the PulseBlasterUSB. This will start the execution of a pulse program. It will also restart (trigger) a program which is currently paused due to a WAIT instruction. Returns a negative number in the event of an error and returns 0 upon success.

```
int pb_stop();
```

Stops execution of the pulse program. This also resets the PulseBlasterUSB so that the pulse program can be run again by calling *pb\_start()* or by applying a hardware trigger to the PulseBlasterUSB. The PulseBlasterUSB's 24 TTL outputs will return to ground when the *pb\_stop()* command is invoked. Returns a negative number in the event of an error and returns 0 upon success.

## **Example Use of SpinAPI Functions in C/C++**

//This program will create an infinite loop consisting of three intervals during //which a) all 24 output bits will be ON for 100ns, b) bit #0 will be OFF and the //remaining 23 output bits will be ON for 200 ns, and c) all output bits will be //OFF for 1us.

```
#include <stdio.h>
#define PBESR
#include "spinapi.h"
#define CLOCK 100.0 //The value of your clock oscillator in MHz

int main(void)
{
    int start;

    if (pb_init() != 0)
    {
        printf("--- Error Initializing PulseBlaster ---\n");
        return -1;
    }
    // Tell driver what clock frequency the PulseBlasterUSB uses
    pb_set_clock(CLOCK);

    // Prepare the PulseBlasterUSB to receive Pulse Program instructions
    pb_start_programming(PULSE_PROGRAM);

    // Instruction #0 - All outputs ON, continue to Instruction #1 after 100ns.
    // Flags = 0xFFFFFFFF (all outputs ON),
    // OPCODE = CONTINUE (proceed to next instruction after specified delay)
    // Data Field = empty. This field is ignored for CONTINUE instructions.
    // Delay Count = 100*ns (other valid units are *us, *ms)
    pb_inst(0xFFFFFFFF, CONTINUE, 0, 100*ns);

    // Instruction #1 - Output bit 0 OFF, all others ON, continue to instruction #2
    // after 200ns.
    // Flags = 0xFFFFFE (all outputs ON except output bit 0)
    // OPCODE = CONTINUE (proceed to next instruction after specified delay)
    // Data Field = empty. This field is ignored for CONTINUE instructions.
    // Delay Count = 200*ns (other valid units are *us, *ms)
    pb_inst(0xFFFFFE, CONTINUE, 0, 200*ns);

    // Instruction #2 - Branch to "start" (Instruction #0) after 1us
    // Flags = 0x000000 (all output bits are OFF),
    // OPCODE = BRANCH
    // Data Field = start (the target branch address)
    // Delay Count = 1*us (other valid units are *ns, *ms)
    pb_inst(0x000000, BRANCH, start, 1*us);

    pb_stop_programming();// Finished Sending Instructions
    pb_start();// Run the Program

    pb_close(); // Release Control of the PulseBlasterUSB
    return 0;
}
```

## Contact Information

**SpinCore Technologies, Inc.**

**Telephone:** +1-352-271-7383

**Web:** <http://www.spincore.com>