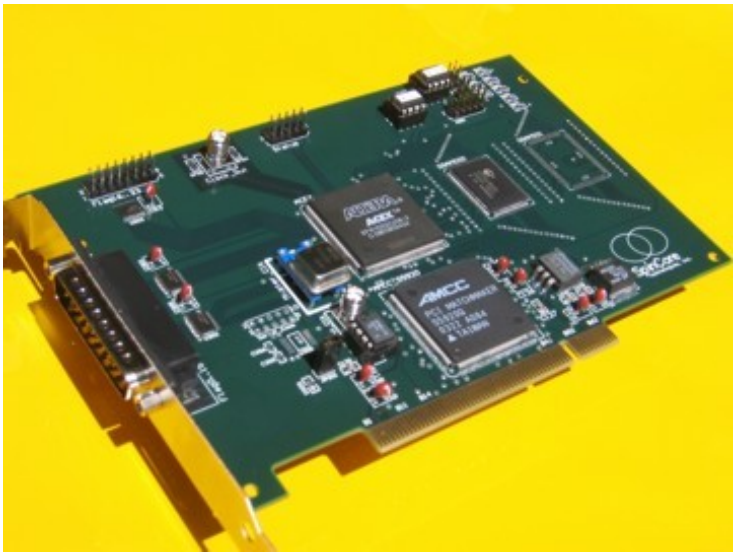
PulseBlaster: Programmable Pulse Generator

Models PB24-512 and PB24-32k

(PCI Board SP2)

Owner's Manual



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

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

We appreciate your business!

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

Table of Contents

I. Introduction

Product Overview

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

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

Board Architecture

Block Diagram

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

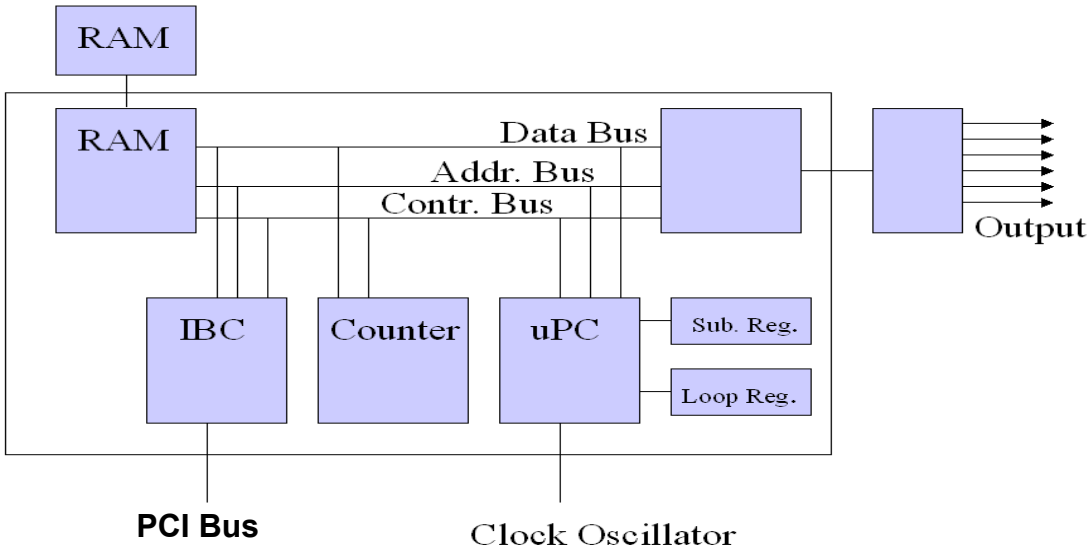


Figure 1: PulseBlaster board architecture

Output Signals

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

Timing Characteristics

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 timing instructions (delays of up to 2^{32} = 4,294,967,296 clock cycles), or double-length timing instructions (up to 2^{52} clock cycles long – nearly 2 years with a 100 MHz clock!). 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 minimum delay cycle of five clock periods for PB24-512 and a minimum delay cycle of nine clock periods for PB24-32k. For a 100 MHz clock, this translates to a 50 ns pulse/delay/update for PB24-512 and 90.0 ns pulse/delay/update for PB24-32k.

Instruction Set

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

PulseBlaster can be triggered and/or reset externally via dedicated hardware lines. The two separate lines combine the convenience of triggering (e.g., in cardiac gating) with the safety of the "stop/reset" line. The required control signals are "active low" (or short to ground).

Status Read Back

The status of the pulse program can be read in hardware or software. The hardware status output signals consist of five IDC connector pins labeled "Status". The same output can be read through software using C. See section IV (Connecting to the PulseBlaster Board, page 16) for more detail about the hardware lines and section III (Programming the PulseBlaster, page 11) for more detail about the C function status_readback().

Summary

PB24 is a versatile, high-performance, programmabe pulse/pattern TTL signal generator operating at speeds of up to 100 MHz and capable of generating pulses/delays/intervals ranging from 50 ns to two years per instruction. It can accommodate pulse programs with highly flexible control commands of up to 512 program words (Model PB24-512) or 32k program words (Model PB24-32k). Its high-current output logic bits are independently controlled with a voltage of 3.3 V.

Specifications

Pulse Parameters

- 24 individually controlled digital output lines (TTL levels, 3.3 V logical “one”)
- variable pulses/delays for every TTL line
- 25 mA output current per TTL line
- 50 ns shortest pulse/interval (at 100 MHz, Internal Memory Model, PB24-512)
- 2 years longest pulse interval (at 100 MHz, with the use of the “Long Delay” instruction)
- 10 ns pulse/interval resolution (at 100 MHz)
- up to 32k pulse program memory words/instructions (Model PB24-32k)
- external triggering and reset – TTL levels

Pulse Program Control Flow

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

II. Installation

Installing the PulseBlaster Driver

The PulseBlaster uses the SpinAPI driver and control library. This is part of the SpinCore Driver Suite, which can be downloaded from:

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

This URL also contains a link to the SpinAPI documentation and installation instructions. Please refer to the above URL for instructions on how to properly install the drivers for your PulseBlaster.

III. Programming the PulseBlaster

The PulseBlaster Interpreter

The PulseBlaster board is now programmable via the *PulseBlaster Interpreter*, which is a free programming utility provided by SpinCore for writing pulse programs. This easy-to-use editor allows you create, edit, save and run your pulse sequence.

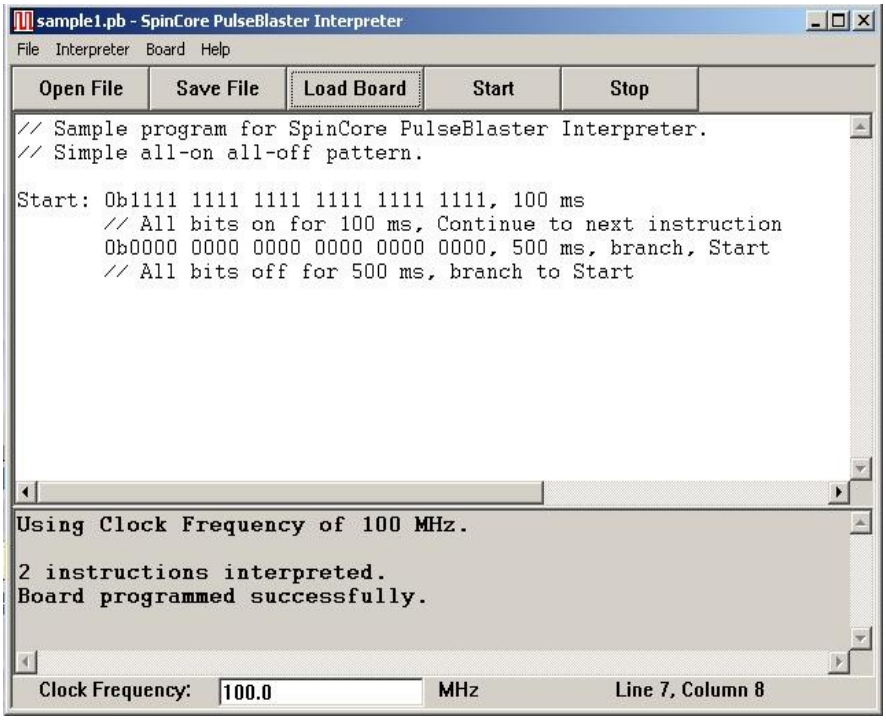


Figure 2: The PulseBlaster Interpreter

The PulseBlaster Interpreter is available as part of the SpinCore Driver Suite and will be installed automatically during the setup process described above in Section II.

It is also possible to write pulse programs and control the PulseBlaster with C functions. For more information on this, please see Appendix I.

IV. Connecting to the PulseBlaster Board

Connector Information

DB25 Bracket Connector Flag 0..15 - Pin Assignments

Outputs 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 3: Output bits and signals of the PulseBlaster

IDC Connector Flag16..23 - Pin Assignments

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

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

Table 4: Output bits and signals of the PulseBlaster

IDC Connector Status - Pin Assignments

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

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

Table 5: Status signals of the PB24

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

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

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

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

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

Header JP100

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

HW_Trigger is pulled high by default, and pin 1 is active (pin 2 = GND). When a falling edge is detected (e.g., when shorting pins 1-2), it initiates code execution. This trigger will also restart execution of a program from the beginning of the code if it is detected after the design has reached an idle state. The idle state could have been created either by reaching the STOP Op Code of a program, or by the detection of the HW_Reset signal. When the WAIT Op Code is used in the pulse program, the HW_Trigger will cause the program to continue to the next instruction.

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

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. 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 excercised, and certain conditions have to be met prior to using this connector. First, before attaching any external clock source, the internal clock oscillator must be removed from its socket. The internal clock oscillator's orientation should be noted - if the internal clock is reconnected, it must be inserted in the same orientation or board damage may occur. Second, the external clock signal must be 3.3 V TTL, i.e., a positive-only voltage - any negative voltage at the Ext_Clk connector will damage the programmable-logic processor chip. Third, as the Ext-Clk connector is not terminated on the printed circuit board, a 50 ohm terminating resistor should be used externally via a T connector placed directly at the SMA Ext_Clk connector. Alternatively, a 50 ohm resistor could be soldered on the board on R401 pads.

Appendix I: Controlling the PulseBlaster with 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 512 (Model PB24-512, also called the "Internal Memory Model"); the maximum number of instructions that can be loaded to the on-board memory is 32k (Model PB24-32k, also called the "External Memory Model). 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 2 shows the fields and bit definitions of the 80-bit instruction word.

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

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

Figure 2: Bit definitions of the 80-bit instruction/memory word

Breakdown of 80-bit Instruction 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 1 for output pattern and control bit assignments of the 24-bit output/control word.

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

Table 1: Output Pattern and Control Word Bits

Data Field and Op Code

Please refer to Table 2 for information on the available operational codes (OpCode) and the associated data field functions (the data field's function is dependent on the Op Code)

Op Code #	Inst	Inst_data	Function
0	CONTINUE	Ignored	Program execution continues to next instruction
1	STOP	Ignored	Stop execution of program (*Note all TTL values remain from previous instruction, and analog outputs turn off)
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 begging of loop and decrements loop counter.
4	JSR	Address of first subroutine instruction	Program execution jumps to beginning of a subroutine
5	RTS	Ignored	Program execution returns to instruction after JSR was called
6	BRANCH	Address of next instruction	Program execution continues at specified instruction
7	LONG_DELAY	Number of desired loops. This value must be greater than or equal to 2.	For long interval instructions. Data field specifies a multiplier of the delay field. Execution continues to next instruction
8	*WAIT	Ignored	Program execution stops and waits for software or hardware trigger. Execution continues to next instruction after receipt of trigger. The latency is equal to the delay value entered in the WAIT instruction line plus a fixed delay of 6 clock cycles.

Table 2: Op Code and Data Field Description

*Note: The WAIT opcode may not be used as the first instruction in a pulse program and must be preceded by an instruction of at least 110ns in length if used as the second instruction in a pulse program.

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 PB24-512 and 0x00000006 for PB24-32K 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:

http://www.spincore.com/CD/spinapi/spinapi_reference/

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.

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

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

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

```
void set_clock(double clock_freq);
```

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

```
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 alid value for **device** is PULSE_PROGRAM, It returns a 0 on success or a negative number on an error.

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

```
int pb_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 0x3FF. For example, 0x010 would correspond to bit 7 being on, and all other bits being 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 with the previous inst field. Please see Table 2 for details.

int length – duration of this pulse program instruction, specified in 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.

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_set_clock(100.0);

      pb_start_programming(PULSE_PROGRAM);

      // Instruction 0 - Continue to instruction 1 in 100ms
      // 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

A more complex program using C Functions is provided below.

```
//*
 * 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_set_clock(100.0);

      pb_start_programming(PULSE_PROGRAM);

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

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

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

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

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

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

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

      // Subroutine
      // Instruction 5 - Continue to next instruction in 1 * s
```

```
        pb_inst(0x0,CONTINUE,0,500.0*ms);

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

        // End of pulse program
        pb_stop_programming();

        // Trigger the pulse program
        pb_start();

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

        pb_close();

        return 0;
}
```

Available Options and Accessories

1. Ribbon Cable with 2x13 IDC plug and DB-25 connector on PC bracket – can be used to route the upper 8 output bits to the back panel of the computer.

2. Oven Controlled Clock Oscillator, sub-ppm stability – please contact SpinCore Technologies, Inc. for details.

3. Higher Clock Frequency (currently up to 400 MHz) – PulseBlasterESR, PulseBlasterESR-PRO, and PulseBlasterESR-PRO-II series of boards. For more information, please visit the individual Product URLs of the aforementioned products at the general URL http://www.spincore.com

4. USB control – the PulseBlasterPlus product line. For more information, please visit the URL http://www.spincore.com/products/PulseBlasterPlus/

5. Additional programmable DC output and RF pulse generation – the PulseBlasterDDS line of products. For more information, please visit the URL http://www.spincore.com/products/PulseBlasterDDS-300/

6. Additional high-speed signal acquisition, currently up to 200 MHz – the PulseBlasterAD series of products. For more information, please contact SpinCore Technologies, Inc.

Contact Information

Telephone (USA): **352-271-7383**

Web Form: http://www.spincore.com/