



PulseBlasterESR-cPCI™

PulseBlasterESR-PRO-cPCI™

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.**

© 2000-2007 SpinCore Technologies, Inc. All rights reserved.

SpinCore Technologies, Inc. reserves the right to make changes to the product(s) or information herein without notice. PulseBlasterESR-cPCI™, 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

| | |
|--|-----------|
| I. Introduction..... | 5 |
| Product Overview | 5 |
| Board Architecture | 5 |
| <i>Block Diagram</i> | <i>5</i> |
| <i>Output Signals.....</i> | <i>6</i> |
| <i>Timing Characteristics.....</i> | <i>6</i> |
| <i>Instruction Set.....</i> | <i>6</i> |
| <i>External Triggering</i> | <i>6</i> |
| <i>Summary.....</i> | <i>6</i> |
| Specifications..... | 7 |
| <i>TTL Specifications.....</i> | <i>7</i> |
| <i>Pulse Parameters.....</i> | <i>7</i> |
| <i>Pulse Program Control Flow</i> | <i>7</i> |
| II. Installation..... | 8 |
| Installing the PulseBlasterESR-cPCI Driver | 8 |
| III. Programming the PulseBlasterESR-cPCI..... | 8 |
| The PulseBlaster Interpreter..... | 8 |
| IV. Connecting to the PulseBlasterESR-cPCI Board..... | 9 |
| Connector Information..... | 9 |
| <i>BNC Header Connectors.....</i> | <i>9</i> |
| <i>IDC Headers.....</i> | <i>9</i> |
| <i>Header HWTrig/Reset.....</i> | <i>10</i> |
| Appendix I: Controlling the PBESR-cPCI with SpinAPI..... | 11 |
| Instruction Set Architecture..... | 11 |
| <i>Machine-Word Definition.....</i> | <i>11</i> |
| <i>Breakdown of 80-bit Instruction Word.....</i> | <i>11</i> |
| About SpinAPI..... | 13 |
| Using SpinAPI Functions to Program the PulseBlasterESR..... | 14 |

PulseBlasterESR-cPCI

Example Use of SpinAPI Functions in C/C++..... 16

Contact Information..... **17**

I. Introduction

Product Overview

The PulseBlasterESR-cPCI™ device is a high-speed, multi-channel, intelligent pulse/pattern generation unit in the Compact-PCI (c-PCI) form factor. The digital logic output signals are 3.3 V TTL compatible.

The intelligence of the PulseBlasterESR-cPCI comes from an imbedded microprogrammed controller core (uPC). The controller is able to execute instructions that allow it to control program flow. This means that the PulseBlasterESR-cPCI understands Operational Control Codes, OpCodes, and will execute them much the same way as a general-purpose microprocessor does. The PulseBlasterESR-cPCI'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 PulseBlasterESR-cPCI processor is that the execution time of instructions is user programmable. This feature makes the PulseBlasterESR-cPCI processor capable of executing complex patterns at greatly varying update rates, ranging from nanoseconds to years, with a constant setting accuracy of just one clock period (e.g., a 4.0 ns setting accuracy when using a 250 MHz clock frequency).

Board Architecture

Block Diagram

Figure 1 presents the general architecture of the PulseBlasterESR-cPCI 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 the SRAM memory and 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 integrated bus controller (IBC) over the c-PCI bus.

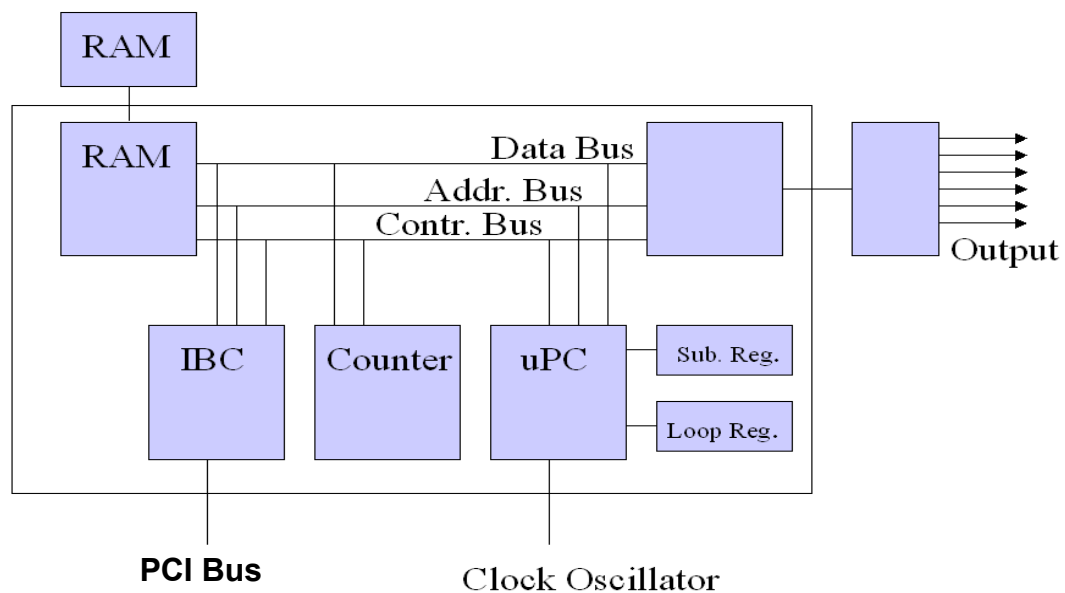


Figure 1: PulseBlasterESR-cPCI board architecture

Output Signals

PulseBlasterESR-cPCI models come with up to 24 digital output signal lines. All signal lines are routed to two sets of 26-pin IDC on-board connectors. The first four output bits are also routed to four bracket mounted BNC connectors. The output signals are impedance matched to 50 ohm.

These individually controlled digital output bits comply with the TTL-levels' standard, and are capable of delivering ± 25 mA per bit/channel. If more output current is necessary, the individual bits/channels can be driven in parallel.

Timing Characteristics

The PulseBlasterESR-cPCI's timing controller accepts an internal (on-board) crystal oscillator up to 125 MHz. This input frequency is internally doubled to 250 MHz. The innovative architecture of the timing controller allows the processing of either simple timing instructions (delays of up to $2^8 = 256$ clock cycles), or double-length timing instructions (up to 2^{28} clock cycles long – 1.07 seconds at 250 MHz). Regardless of the type of timing instruction, the timing resolution remains constant for any delay – just one clock period (e.g., 4.0 ns at 250 MHz). The core-timing controller has a very short minimum delay cycle – five clock periods on ESR models, and just one clock period on ESR-PRO models. This translates to a 20 ns smallest pulse/delay/update at 250 MHz for the ESR model and 4 ns for the ESR-PRO model.

Instruction Set

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

External Triggering

The PulseBlasterESR-cPCI 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).

Summary

The PulseBlasterESR-cPCI is a versatile, high-performance pulse/pattern TTL signal generator operating at speeds of up to 250 MHz and capable of generating pulses ranging from 4.0 ns to 1.07 s per instruction and delays/intervals ranging from 20 ns to 1.07 s per instruction. It can accommodate pulse programs with highly flexible control commands of up to 4096 program words (varies by model). Its high-current output logic bits are independently controlled with an output voltage of 3.3 V (unterminated).

Specifications

TTL Specifications

- 24 individually controlled digital output lines, 3.3V TTL (ESR-PRO model has 21 output lines)
- 4 bracket mounted BNC connectors, impedance matched to 50 ohm.
- Variable pulses/delays for every TTL line
- 25 mA output current per TTL line

Pulse Parameters

- 4.0 ns shortest pulse (at 250 MHz) (ESR-PRO models only)
- 20 ns shortest pulse (at 250 MHz) (all other models)
- 1.07 s longest pulse/interval (at 250 MHz)
- 4.0 ns pulse/interval resolution (at 250 MHz)
- Up to 4096 instructions
- 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 – user adjustable latency as low as 8 clock cycles (32.0 ns at 250 MHz)

II. Installation

Installing the PulseBlasterESR-cPCI Driver

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

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

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 PulseBlasterESR-cPCI.

III. Programming the PulseBlasterESR-cPCI

The PulseBlaster Interpreter

The PulseBlasterESR-cPCI 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. A sample screenshot is shown below.

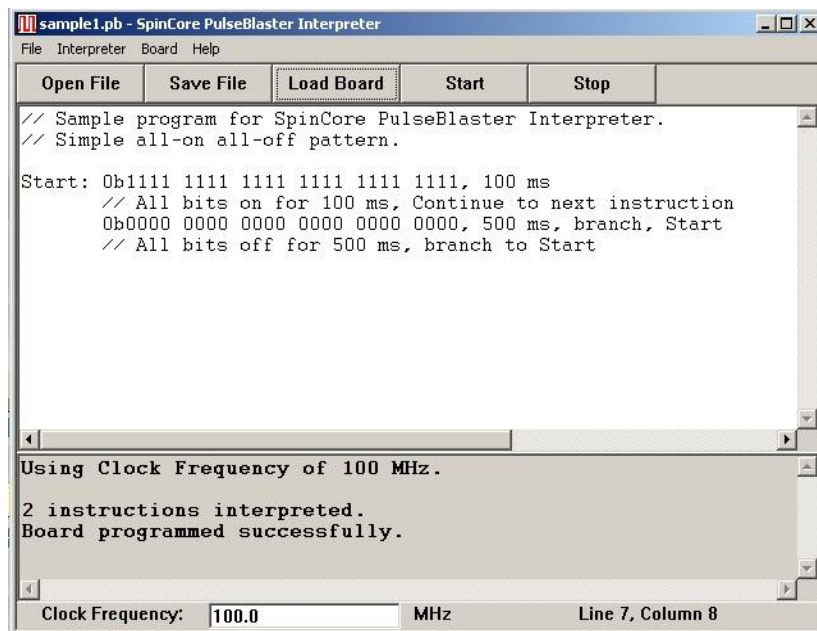


Figure 2: The PulseBlaster Interpreter

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 PulseBlasterESR-cPCI with C functions. For more information on this, please see Appendix I.

IV. Connecting to the PulseBlasterESR-cPCI Board

Connector Information

BNC Header Connectors

The four bracket mounted BNC connectors output the first four bits, starting from the list-significant bit, as depicted below. When using coaxial transmission lines, please terminate the lines with a 50 ohm impedance.

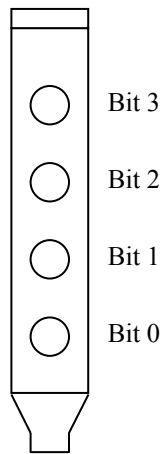


Figure 3: Connector Locations

IDC Headers

There are three IDC headers on the PulseBlasterESR which provide access to all of the digital outputs. These are labeled Flag0..11_Out, Flag12..23_Out and Flag24..35_Out. On each IDC header, the top row of pins (14-26) are grounds, and the signals are carried on pins 1-13. The IDC connector pinout is shown below in Figure 4.

| | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |

Figure 4: IDC header pinout

PulseBlasterESR-cPCI

Each pin on an IDC header corresponds to a bit in the flag field of an instruction. The association between bits and pins is shown in the table below.

| Pin Assignments | | | |
|-----------------|-----------|------------|------------|
| Pin# | Flag0..11 | Flag12..23 | Flag24..35 |
| 1 | Bit 0 | Bit 12 | Stopped |
| 2 | Bit 1 | Bit 13 | Reset |
| 3 | Bit 2 | Bit 14 | Running |
| 4 | Bit 3 | Bit 15 | Waiting |
| 5 | Bit 4 | Bit 16 | Unused |
| 6 | Bit 5 | Bit 17 | Unused |
| 7 | Bit 6 | Bit 18 | Unused |
| 8 | Bit 7 | Bit 19 | Unused |
| 9 | Bit 8 | Bit 20 | Unused |
| 10 | Bit 9 | Bit 21 | Unused |
| 11 | Bit 10 | Bit 22 | Unused |
| 12 | Bit 11 | Bit 23 | Unused |
| 13 | Unused | Unused | Unused |
| 14-26 | Ground | Ground | Ground |

Table 1: IDC connector pin outs

Some models of the PulseBlasterESR-cPCI and PulseBlasterESR-PRO-cPCI also feature four extra output signals, or "Status" signals, which will tell the user what state the device is in. Pin assignments for these signals are shown above in Table 1. The status bits are organized as follows:

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

Reset – Driven low 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 – Driven high when the PulseBlaster device has encountered a WAIT Opcode and is waiting for the next trigger (either hardware or software) to resume operation.

Header HWTrig/Reset

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

HW_Trigger is pulled high by default, and pin 2 is active (pin 4 = GND). When a low value is detected (e.g., when shorting pins 2-4), 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 1 is active (pin 3 = GND). It can be used to halt the execution of a program by pulling it low (e.g., by shorting pins 1-3). 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.

Appendix I: Controlling the PBESR-cPCI 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-board memory is 4096. The execution time of instructions can be varied and is controlled by one of the fields of the instruction word – the shortest being five clock cycles (one cycle for PRO models) and the longest being 2^{28} clock cycles. All instructions have the same format and bit length, and all bit fields have to be filled. Figure 5 shows the fields and bit definitions of the 80-bit instruction word.

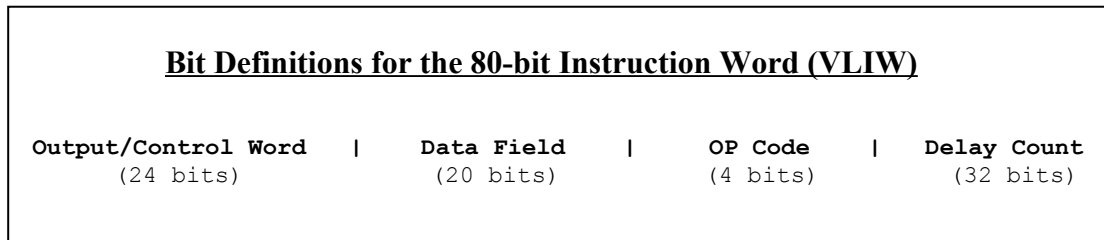


Figure 5: Bit definitions of the 80-bit instruction/memory word

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 # | | Bit # | Function |
|-------|--|-------|---|
| 23 | Output Connector labeled Flag12..23 Out pin 12 | 11 | Output Connector labeled Flag0..11 Out pin 12 |
| 22 | Output Connector labeled Flag12..23 Out pin 11 | 10 | Output Connector labeled Flag0..11 Out pin 11 |
| 21 | Output Connector labeled Flag12..23 Out pin 10 | 9 | Output Connector labeled Flag0..11 Out pin 10 |
| 20 | Output Connector labeled Flag12..23 Out pin 9 | 8 | Output Connector labeled Flag0..11 Out pin 9 |
| 19 | Output Connector labeled Flag12..23 Out pin 8 | 7 | Output Connector labeled Flag0..11 Out pin 8 |
| 18 | Output Connector labeled Flag12..23 Out pin 7 | 6 | Output Connector labeled Flag0..11 Out pin 7 |
| 17 | Output Connector labeled Flag12..23 Out pin 6 | 5 | Output Connector labeled Flag0..11 Out pin 6 |
| 16 | Output Connector labeled Flag12..23 Out pin 5 | 4 | Output Connector labeled Flag0..11 Out pin 5 |
| 15 | Output Connector labeled Flag12..23 Out pin 4 | 3 | Output Connector labeled Flag0..11 Out pin 4 |
| 14 | Output Connector labeled Flag12..23 Out pin 3 | 2 | Output Connector labeled Flag0..11 Out pin 3 |
| 13 | Output Connector labeled Flag12..23 Out pin 2 | 1 | Output Connector labeled Flag0..11 Out pin 2 |
| 12 | Output Connector labeled Flag12..23 Out pin 1 | 0 | Output Connector labeled Flag0..11 Out pin 1 |

Table 2: Output Pattern and Control Word Bits

When the bit corresponding to an IDC output connector pin is one, that output line will remain high for the duration of the instruction. When the bit is zero, the output line will remain low for the duration of the instruction.

Short Pulses

On ESR-PRO model boards, the minimum pulse length is one clock cycle. The minimum *instruction* length is still five clock cycles, and it is through bits 23-21 that the user can control the time that the pulse is “high”, or “ON”. Thus, bits 23, 22, and 21 are no longer considered digital output channels from a pulse program standpoint, even though they are still present on the IDC header.

When bits 21-23 are zero, the digital outputs remain low for the entire duration of the instruction, regardless of their individual bit settings. When bits 23-21 are 001, all output bits set to 1 in the instruction word will go high at the beginning of the instruction. These outputs will stay high for one clock cycle and then go low again. If bits 23-21 are 010, the outputs will remain high for *two* clock cycles, and so on. If bits 23-21 are 110 or 111, all output bits set to 1 in the instruction word will be high for the entire duration of the instruction. This behavior is summarized below in Table 2.

| Bits 23-21 | Clock periods that output is high | Total instruction length (clock periods) |
|------------|-----------------------------------|--|
| 000 | 0 | Delay Count value |
| 001 | 1 | Delay Count value |
| 010 | 2 | Delay Count value |
| 011 | 3 | Delay Count value |
| 100 | 4 | Delay Count value |
| 101 | 5 | Delay Count value |
| 110 | Delay Count value | Delay Count value |
| 111 | Delay Count value | Delay Count value |

Table 2: Short Pulses

Please note that no matter what the values of bits 23-21 are, the duration of the pulse program instruction will be the delay that was entered (“Delay Count”), regardless of how long the individual output bits are set high via the Short Pulse feature.

Data Field and Op Code

Please refer to Table 3 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 3: Op Code and Data Field Description

Delay Count

The value of the Delay Count field (a 32-bit value) determines how long the current instruction should be executed. The allowed minimum value of this field is 0x00000002 and the allowed maximum is 0x000000FF. 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 PulseBlasterESR board. 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 PulseBlasterESR board. 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/spinapi/>. A complete reference document for the API is also available at that location.

Using SpinAPI Functions to Program the PulseBlasterESR

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 PulseBlasterESR board, making it ready to begin accepting commands. Initialization must be performed before any other functions can interface with the PulseBlasterESR, 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 PulseBlaster board. 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 board is. For example, when using the 250MHz PBESR-250, the number 250.0 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. (Note: the actual operating frequency of your PulseBlasterESR is *twice* the frequency printed on the on-board oscillator).

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

Makes the PulseBlasterESR 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.

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

Used to send one instruction of the pulse program to the PulseBlasterESR. 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).

PulseBlasterESR-cPCI

```
int pb_stop_programming();
```

Used to notify the PulseBlasterESR 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 board. 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 PulseBlasterESR so that the pulse program can be run again by calling *pb_start()* or by applying a hardware trigger to the board. The board's TTL outputs will remain in the state they were in when the stop command was received. 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 40ns, b) bit #0 will be OFF and the
//remaining 23 output bits will be ON for 80 ns, and c) all output bits will be
//OFF for 1us.

#include <stdio.h>
#define PBESR
#include "spinapi.h"
#define CLOCK 250.0 //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 board uses
    pb_set_clock(CLOCK);

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

    // Instruction #0 - All outputs ON, continue to Instruction #1 after 40ns
    // 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 = 40*ns (other valid units are *us, *ms)
    start = pb_inst(0xFFFFFFFF, CONTINUE, 0, 40*ns);

    // Instruction #1 - Output bit 0 OFF, all others ON, continue to instruction 2
    // after 80ns
    // 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 = 80*ns (other valid units are *us, *ms)
    pb_inst(0xFFFFFE, CONTINUE, 0, 80*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 PulseBlasterESR board
    return 0;
}
```

Contact Information

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

Product URL: <http://www.spincore.com/products/PulseBlasterESR/>