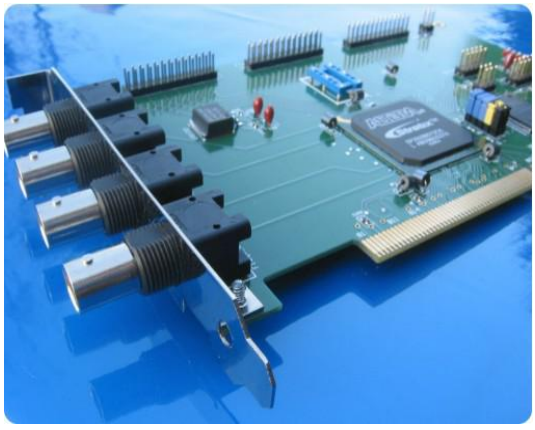
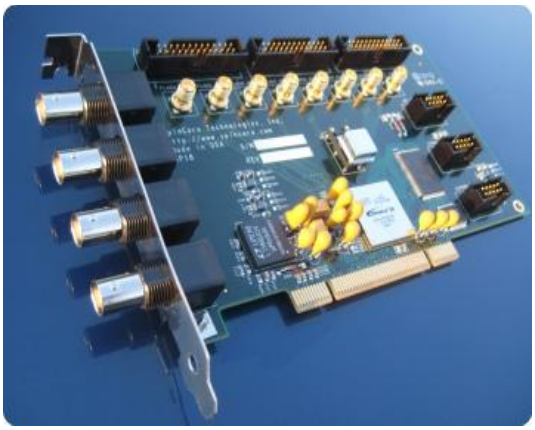
PulseBlasterESR-PRO™

(PCI Boards SP4B and SP18A)

300, 400, and 500 MHz

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.

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

Product Overview

The PulseBlasterESR-PRO™ is a high-speed, intelligent pulse/pattern/delay generator designed for outputting precisely timed TTL patterns. The intelligence of the PulseBlasterESR-PRO comes from an embedded microprogrammed controller core nicknamed the PulseBlaster™. The controller is able to execute instructions that allow it to control program flow much like a general purpose microcontroller. The speed of the PulseBlasterESR-PRO comes from a maximum available clock of 500 MHz.

The PulseBlasterESR-PRO's microprogrammed controller core is different from the general-purpose microcontroller in that it contains a set of highly optimized instructions developed specifically for timing and control applications. A unique and distinguishing feature of the PulseBlasterESR-PRO processor is that the execution time for instructions is user programmable. This feature makes the PulseBlasterESR-PRO processor capable of executing complex timing patterns at greatly varying update rates, ranging from nanoseconds to months, with a constant setting accuracy of just one clock period.

Board Architecture

Block Diagram

Figure 1 presents the general architecture of the PulseBlasterESR-PRO system. The major building blocks are the SRAM memory, the PulseBlaster™ core, 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 PCI bus.

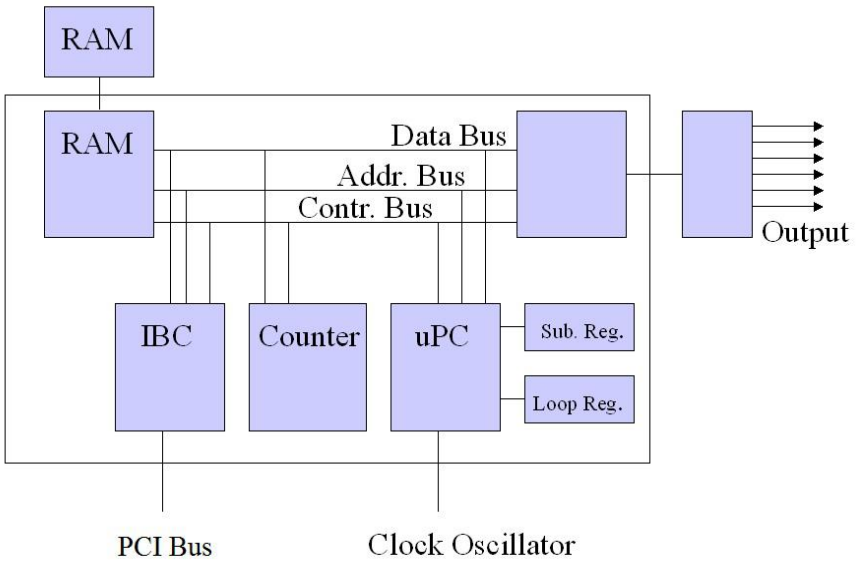


Figure 1: PulseBlasterESR-PRO Board Architecture. The clock oscillator signal is derived from an on-chip PLL circuit typically using a 50 MHz on-board reference clock.

Output Signals

The PulseBlasterESR-PRO allows for 21 digital output signal lines. All 21 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.

The 21 individually controlled digital output bits comply with the 3.3V TTL-levels' standard, and are capable of delivering ±25 mA per bit/channel. Keep in mind that this is sufficient to provide a signal to a 132 ohm load, but more current is necessary beyond this. If more output current is necessary, the individual bits/channels can be driven in parallel.

Timing Characteristics

The PulseBlaster™ core's timing controller accepts an external (on-board) crystal oscillator of 50 MHz. The input frequency is internally multiplied. The PulseBlasterESR-PRO is available with 3 different internal clock frequencies: 300 MHz, 400 MHz, and 500 MHz. The innovative architecture of the timing controller allows the processing of either simple timing instructions (with delays of up to 2^{32} clock cycles or 8.59 s at 500 MHz), or double-length timing instructions (up to 2^{52} clock cycles long – over 100 days at 500 MHz!). Regardless of the type of timing instruction, the timing resolution remains constant for any delay – just one clock period (e.g., 2 ns at 500 MHz).

The PulseBlaster core-timing controller has a very short minimum instruction time – only five clock periods[1]. This translates to a 10 ns machine instruction time at 500 MHz. The PulseBlasterESR-PRO is also capable of generating pulses on all outputs of lengths down to one clock cycle. For more information on this feature, please see the Short Pulse Feature section in Appendix I.

Instruction Set

The PulseBlaster™ core features a set of instructions for creating highly flexible pulse 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

PulseBlasterESR-PRO 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 (firmware-dependent). The input pins for the trigger and reset are pulled to logical-high on the board and can be activated by a low-voltage pulse (short to ground).

Summary

PulseBlasterESR-PRO is a versatile, high-performance pulse/pattern TTL signal generator operating at speeds of up to 500 MHz and capable of generating pulses ranging from 2 ns to 104 days per instruction at intervals ranging from 10 ns to 8.59 s per instruction (using a 500 MHz clock signal). It can accommodate pulse programs with highly flexible control commands of up to 4096 instruction words. Its high-current output logic bits are independently controlled with an unterminated output voltage of 3.3 V.

[1]NOTE: For PulseBlasterESR-PRO-500 design 17-11, instructions with CONTINE, JSR, RTS and LONG_DELAY OpCodes, require a minimum instruction time of at least 6 clock-cycles (12.0 ns). For PulseBlasterESR-PRO-400 design 9-18, the minimum instruction time is 7 clock-cycles (17.5 ns).

Specifications

TTL Specifications

- 21 individually controlled digital output lines (LVTTL levels, 3.3 V logical “one” unterminated)
- 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 (using 500 MHz clock frequency)

- 2 ns shortest pulse
- 10 ns shortest interval[1]
- 104 days longest pulse/interval (using the long delay instruction)
- 2 ns pulse/interval resolution
- 4096 instructions
- External triggering and reset – 3.3V LVTTL levels

Pulse Program Control Flow (Common)

- 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 (16ns at 500 MHz), adjustable to 0.89 seconds in duration
- 15 MHz max. re-triggering frequency

[1]NOTE: For PulseBlasterESR-PRO-500 design 17-11, instructions with CONTINE, JSR, RTS and LONG_DELAY OpCodes, require a minimum instruction time of at least 6 clock-cycles (12.0 ns). For PulseBlasterESR-PRO-400 design 9-18, the minimum instruction time is 7 clock-cycles (17.5 ns).

II. Installation

Installing the PulseBlasterESR-PRO

Whenever installing or uninstalling the PulseBlasterESR-PRO, always have it disconnected from the computer. Uninstall any previous version of SpinAPI.

1. Install the latest version of SpinAPI found at: http://www.spincore.com/support/spinapi/.
   - SpinAPI is a custom Application Programming Interface developed by SpinCore Technologies, Inc. for use with the PulseBlasterESR-PRO and most of SpinCore's other products. It can be utilized using C/C++ or graphically using the options in the next section below. The API will also install the necessary drivers.

2. Shut down the computer, unplug the power cord, insert the PulseBlasterESR-PRO card into an available PCI slot and fasten the PC bracket securely with a screw.

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

4. The simplest way to test whether the device has been installed properly and can be controlled as intended is to run a simple test program. These example files can be found in the SpinAPI package.

5. To open the SpinAPI package on a Windows 10 PC, simply click the Window Start icon, and scroll down to find and open the "spincore" folder. Example .exe files and their C source code can be found in the folder /SpinAPI/examples. From there, you may select the “PulseBlasterESR-PRO” folder and run all .exe programs to test your PulseBlaster.

III. Programming the PulseBlasterESR-PRO

Spincore Technologies Inc. is dedicated to providing an easy and efficient method of programming your board. Various control methods available are detailed below, making PulseBlaster products flexible for any number of applications.

Special consideration of the ESR-PRO short pulse feature must be taken when programming or operating this board. For signals of instruction time greater than 10 ns, the short pulse feature must be disabled by setting output bits 21-23 to HIGH at all times. For more information, see Appendix I.

The PulseBlaster Interpreter

The PulseBlasterESR-PRO can be programmed using *PulseBlaster Interpreter*, which is a free programming utility provided by SpinCore for writing pulse programs. This easy-to-use editor allows you to create, edit, save, and run your pulse sequence. Figure 2 below shows the PulseBlaster Interpreter being used with an example program.

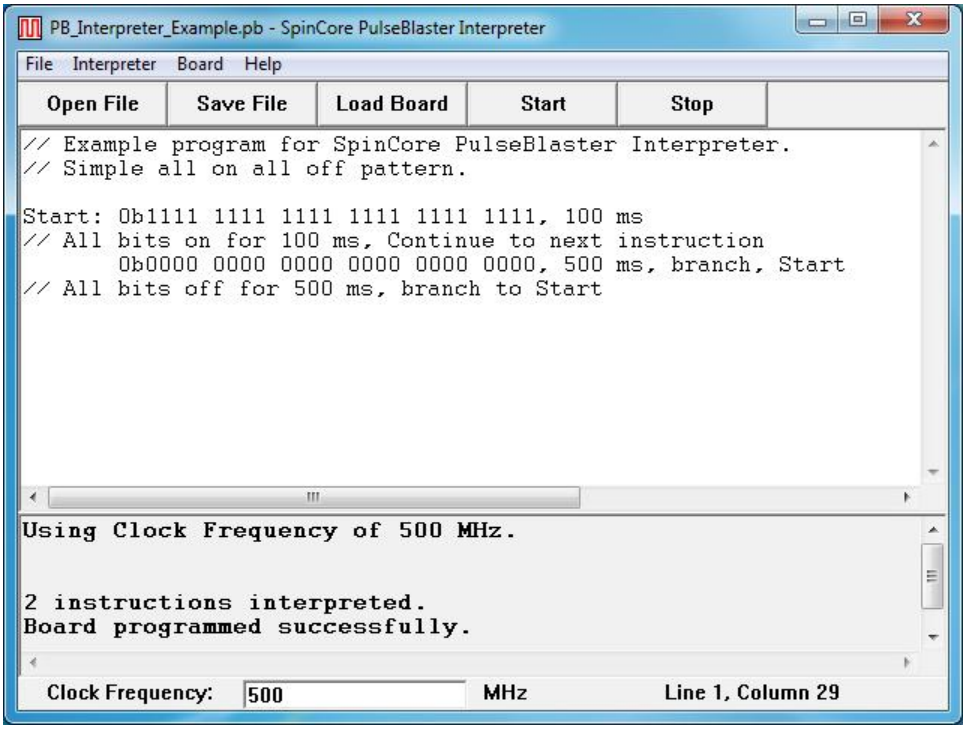


Figure 2: Graphical Interface of PulseBlaster Interpreter. The example shown creates a pulse that toggles all TTL bits on for 100 ms, then off for 500 ms, and repeats.

The PulseBlaster Interpreter is available as part of the SpinCore driver suite, and will be automatically installed during the setup process (setup process is described in Section II. Installation). For convenience, a shortcut to the PulseBlaster Interpreter will be added to your desktop. For more information on programming using the PulseBlaster Interpreter, see the manual located at http://www.spincore.com/support/SPBI/Doc/

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.

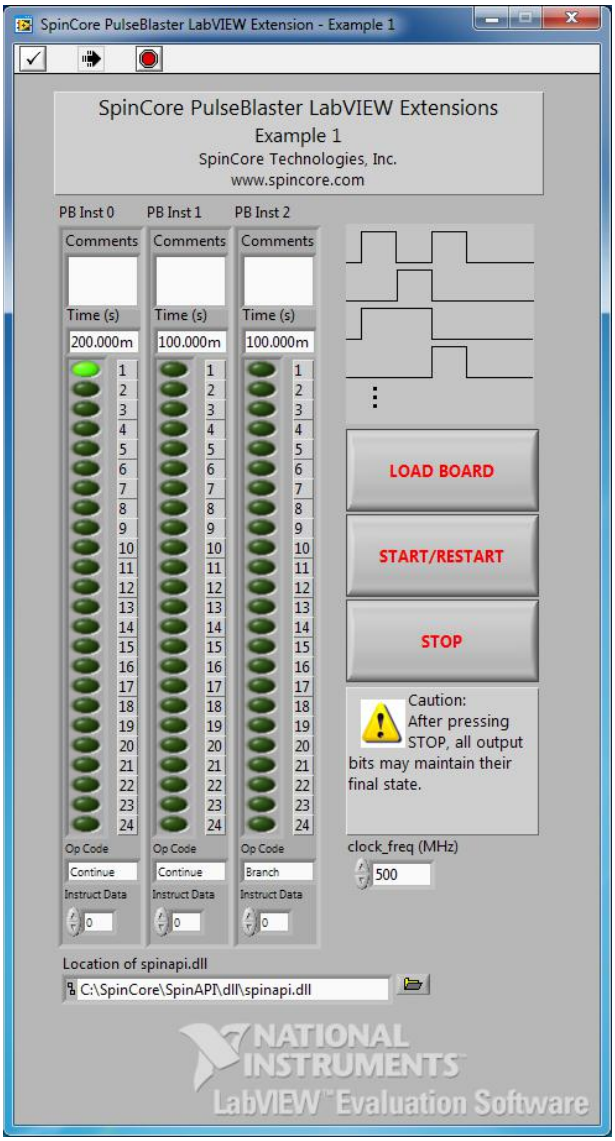


Figure 3: Example of PulseBlaster LabVIEW Extensions User Interface.

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 3 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 PulseBlasterESR-PRO. For more information and downloads please visit:
http://www.spincore.com/support/PBLV/

C/C++ Programming

The most dynamic and flexible way to program the PulseBlasterESR-PRO board is with C/C++ using the SpinAPI package. The GUI based approaches to programming the board are designed for simplicity so they can be used by someone with no programming experience.

While GUI's are easier to use, coding in C/C++ allows you to better utilize the various features of the board and in some cases it may be easier to copy and paste lines of code than to make 100 instructions on a GUI. With the pre-configured compiler on our website, changing one of our example programs and recompiling the executable file for use with your PulseBlasterESR-PRO board is as easy as clicking “Rebuild All” (see Figure 4 below). You can get this pre-configured compiler on our website at:
http://www.spincore.com/support/spinapi/.

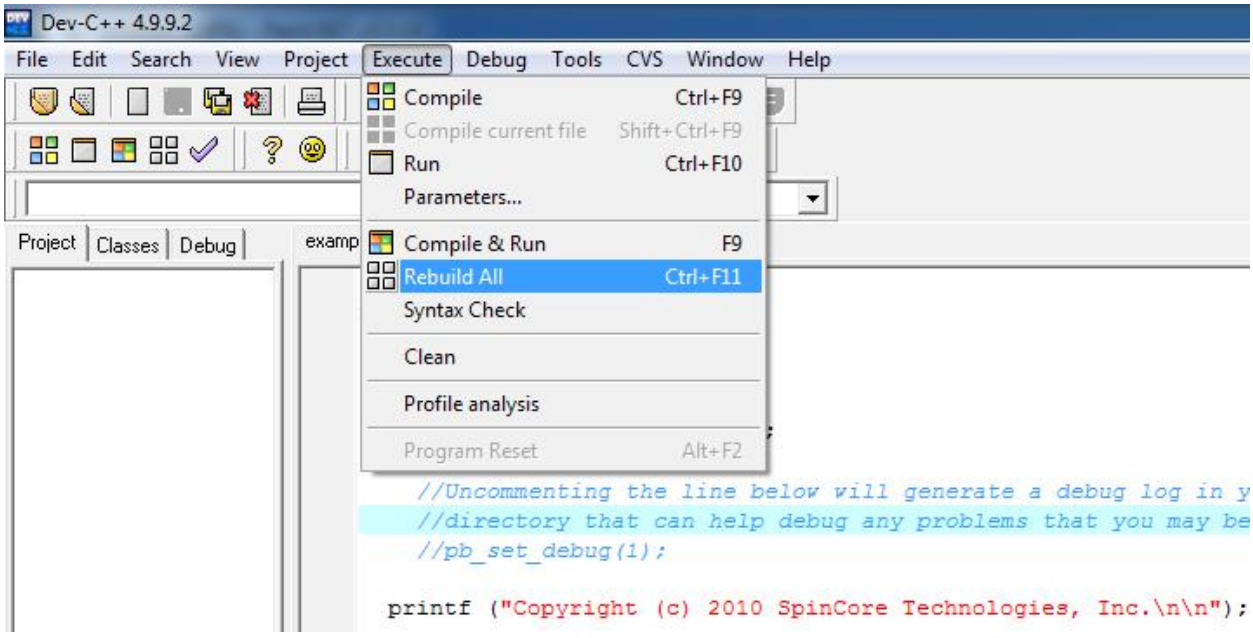


Figure 4: Compiling a C program to run the PulseBlasterESR-PRO board is easy!

Making changes to an example program requires understanding of only a few lines of code. The following C code example generates a 50% duty cycle square wave with a 400.0 ms period.

```
1:  pb_init(); /*Initialize communication with the board*/
2:  pb_core_clock (CLOCK); /*Set the internal clock frequency value - this
3:                          will be either 300 MHz, 400 MHz, or 500 MHz
4:                          depending on your product */
5:
6:   /*Start programming the Pulse Program*/
7:  pb_start_programming (PULSE_PROGRAM);
8:  start=pb_inst(ON|0x01, CONTINUE, 0, 200.0*ms); /*Bit 0 on, 200ms*/
9:        pb_inst(0x00, BRANCH, start, 200.0*ms);/*All bits off, 200ms*/
10:  pb_stop_programming();
11:
12: pb_start(); /*Start the board executing*/
13: pb_close(); /*Close the communication with the board*/
```

A breakdown of the previous C code segment is as follows:

- *Line 1:* Initialize communication with the selected board. This must be called before any other functions that communicate with the board.
- *Line 2:* Set the internal block clock frequency (in MHz). This must be called to insure proper timings in the pulse program.
- *Lines 7-10:* Programs the board's pulse program memory.
- pb_start_programming(PULSE_PROGRAM) must be called before using the pb_inst(..) function.
- Instruction 1: Turn on bit 0 for 200.0 ms then continue to the next instruction. The address of this instruction is stored in the “start” variable.
- *Note: If the output is high for more than 5 clock cycles, it is necessary to turn off the short pulse feature by settings bits 21-23 of the flag bits to “111.” This can be easily accomplished by using the C-macro “ON” defined in “spinapi.h.”*
- Instruction 2: All bits off for 200.0 ms, then branch to “start.”
- pb_stop_programming() must be called before calling any other SpinAPI functions.
- *Line 12:* Start the board executing the Pulse Program.
- *Line 13:* Close communication with the board (Pulse Program execution will continue.)

Using C Functions to Program the PulseBlasterESR-PRO

A series of functions have been written to control the board and facilitate the construction of pulse program instructions. It should be noted that the pb_inst C function accepts any delay value greater than 17 ns. Since the clock period is 3.3 ns for a 300 MHz clock, values which are not integer multiples of the clock will be rounded to the closest integer multiple.

In order to use these functions, the DLL (spinapi.dll), the library file (libspinapi.a for mingw, spinapilibgcc for borland, and spinapi.lib for msvc), the header file (spinapi.h), must be in the working directory of your C compiler[2].

```
int pb_init();
```

Initializes the PulseBlasterESR-PRO board. Needs to be called before calling any functions using the device. Returns a negative number on an error or 0 on success.

```
int pb_close();
```

Releases the PulseBlasterESR-PRO board. Needs to be called as last command in pulse program. Returns a negative number on an error or 0 on success.

```
void pb_core_clock(double clock_freq);
```

Used to set the clock frequency of the board. The variable **clock_frequency** is specified in MHz when no units are entered. Valid units are MHz, kHz, and Hz. The default clock value is 400 MHz. You only need to call this function if you are not using a 400 board.

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

Used to initialize the system to receive programming information. It accepts a parameter referencing the target for the instructions. The only valid value for **device** is PULSE_PROGRAM, It returns a 0 on success or a negative number on an error.

[2] 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 0x000000 to 0x1FFFFF. For example, 0x000010 would correspond to bit 4 being on, and all other bits being off.

int inst – determines which type of instruction is to be executed. Please see Table 6 for details.

int inst_data – data to be used with the previous inst field. Please see Table 6 for details.

double length – duration of this pulse program instruction, specified in nanoseconds (ns), microseconds (us) or milliseconds (ms).

```
The largest value for the delay field of the pb_inst is 8589 ms (using a 500 MHz clock).
For longer delays, use the LONG_DELAY instruction (see Table 6).  The maximum value for
the data field of the LONG_DELAY is 1048576.  Even longer delays can be achieved using
the LONG_DELAY instruction inside of a loop.
```

```
int pb_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();
```

Stop the Pulse Program execution. TTL outputs will either remain in their last state or return to zero, depending on the firmware version of the board. It returns a 0 on success or a negative number on an error.

There are currently six example C programs available with the SpinAPI package in the PBESR-PRO directory.

Example Use of C Functions

```
#include <stdio.h>
#include <stdlib.h>

#define PBESRPRO
#include "spinapi.h"

#define CLOCK 400.0    // PulseBlaster core clock rate

int main (int argc, char **argv)
{
    int start;

    printf ("Copyright (c) 2010 SpinCore Technologies, Inc.\n\n");
    printf("Using SpinAPI library version %s\n", pb_get_version());

    if (pb_init () != 0) {
        printf ("Error initializing board: %s\n", pb_get_error());
        system("pause");
    }

    // Tell driver what clock frequency the board uses
    pb_core_clock(CLOCK);

    // Prepare the board to receive pulse program instructions
    pb_start_programming(PULSE_PROGRAM);

    // Instruction 0 - Continue to instruction 1 in 20ns. The lower 4 bits
    // (all BNC connectors) will be driving high. For PBESR-PRO boards,
    // or-ing THREE_PERIOD with the flags causes a 3 period short
    // pulse to be used.
    start = pb_inst(THREE_PERIOD | 0xF, CONTINUE, 0, 20.0 * ns);

    // Instruction 1 - Continue to instruction 2 in 40ns
    // The BNC1-3 will be driving high the entire 40ns.
    pb_inst(ON | 0xE, CONTINUE, 0, 40.0 * ns);

    // Instruction 2 - Branch to "start" (Instruction 0) in 40ns
    // Outputs are off
    pb_inst(0, BRANCH, start, 40.0 * ns);

    pb_stop_programming();    // Finished sending instructions

    pb_reset();
    pb_start();    // Trigger the pulse program

    // End communication with PulseBlasterESR-PRO board. The pulse program
    // will continue to run even after this is called.
    pb_close();

    return 0;
}
```

IV. Connecting to the PulseBlasterESR-PRO

Connector Information

The PulseBlasterESR-PRO functionality is available on SP4B, Sp18A and SP19 boards. On all versions of the boards, there are three main connector banks: the BNC headers, the IDC headers, and the Trigger/Reset header. Additionally, the SP18A and Sp19 boards offer SMA headers. Please refer to the sections below for detailed connector information.

Connector Locations for the SP4B board

Figure 5 shows the connector locations for SP4B boards.

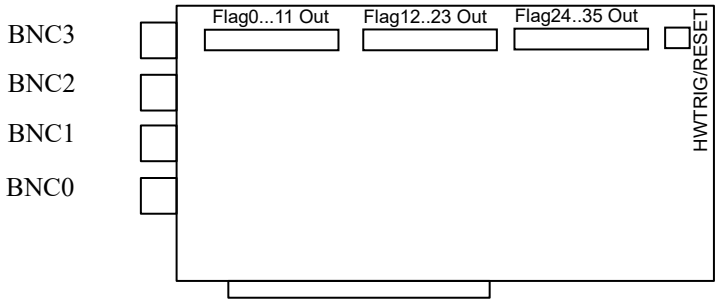


Figure 5: Connector Locations for SP4B. The SP4B board is shown on the first page of this manual, left picture.

BNC Headers

The four BNC headers provide access to the least significant four bits of the flag word. Bit 3 is connected to the output farthest from the PCI connector, and Bit 0 is connected to the connector closest to the PCI connector. Please note that the BNC connectors have the same configuration for all board models.

If using a high input impedance oscilloscope to monitor the PulseBlasterESR-PRO's output via the BNC connectors, place a resistor that matches the characteristic impedance of the transmission line in parallel with the coaxial transmission line at the oscilloscope input (e.g., a 50 Ω resistor with a 50 Ω transmission line, see Figures 6 and 7 below). When using an oscilloscope with an adjustable bandwidth, set the bandwidth to as large as possible. Failure to do so may yield inaccurate readouts on the oscilloscope.

Figure 6: Left: BNC T-Adapter and Right: BNC 50 Ohm resistor.

Figure 7: 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.

IDC Headers for SP4B and SP18A

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 8: IDC header Pin-Out.

There are three IDC headers on SP4B and SP18 boards, which provide access to all digital 21 outputs as well as the status bits. These are labeled on the board as **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 recommended mating connector for IDC header can be obtained through Digi-Key Part Number: CKR26G-ND.

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

Alternatively, the IDC headers can be connected to SP32 boards (Figure 16) which allow the use of MMCX cables. This enables the individual bits of the PulseBlasterESR-PRO to be more easily accessed. Pin 1 on the MMCX adapters can be identified with a square pin.

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.

The Status pins are set or cleared based on the state of the PulseBlasterESR-PRO. The pins are located on IDC header Flag24...35, from pin 1 to pin 4. The status pins are defined as follows:

- **Stopped** – Driven high when the PulseBlasterESR-PRO has encountered a STOP OpCode during program execution and has entered a stopped state.
- **Reset** – Driven low when the PulseBlasterESR-PRO is in a RESET state. The device must be reprogrammed before code execution can begin again.
- **Running** – Driven high when the PulseBlasterESR-PRO is executing a program. The pin is driven low when the PulseBlasterESR-PRO enters either a reset or idle state.
- **Waiting** – Driven high when the PulseBlasterESR-PRO has encountered a WAIT OpCode, and is waiting for the next trigger (either hardware or software) to resume operation.

Note that it is also possible to read the status bits via software by using the *pb_read_status()* function. Please see http://www.spincore.com/CD/spinapi/spinapi_reference/ for details.

HWTrig/Reset Header for SP4B

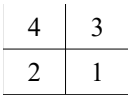


Figure 9: HWTrig/Reset Header Pin-Out(SP4B).

This is an input connector, for hardware triggering (HW_Trigger) and hardware resetting (HW_Reset). Pins 1 and 2 are the reset and trigger inputs, respectively, and pins 3 and 4 are grounds.

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

HW_Trigger is pulled to logical-high voltage (3.3V) on the board. Pin 2 is active and pin 4 is the corresponding ground. When a falling edge is detected (e.g., when shorting pins 2-4) and the program is idle, code execution is triggered. If the program is idle due to a WAIT OpCode, then the HW_Trigger will cause the program to continue to the next instruction. If the program is idle due to a STOP OpCode or a HW_Reset signal, then the HW_Trigger starts execution from the beginning of the program. When using the STOP OpCode, a HW_Reset or software reset (pb_reset()) needs to be applied prior to the HW_Trigger.

Figure 12 in HW_Trig/Reset Header for SP18A and SP19 shows an example of the HW_Trigger signal.

HW_Reset is pulled to logical-high voltage (3.3V) on the board. Pin 1 is active and pin 3 is the corresponding ground. 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 (pb_start()) or by a hardware trigger.

Connector Locations for the SP18A board

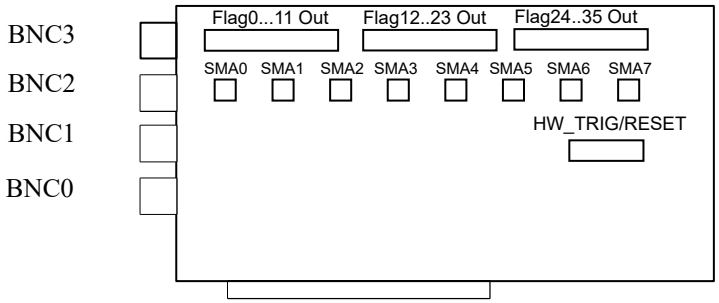


Figure 10: Connector Locations (SP18A).

The SP18A board has identical connectors to the SP4B board, but includes SMA headers for easy access to more flag outputs and a modified HW_Trig/Reset header.

SMA Headers for SP18A

The eight SMA headers provide access to the flag bits 4 through 11. SMA0, the header closest to the BNC headers, corresponds to flag bit 4. SMA7, the header furthest from the BNC header, corresponds to flag bit 11.

HW_Trig/Reset Header for SP18A and SP19

9	7	5	3	1
10	8	6	4	2

Figure 11: HW_Trig/Reset Header Pin-Out(SP18A).

This is an input connector for hardware triggering (HW_Trigger) and resetting (HW_Reset). Pins 8 and 10 are the HW_Reset and HW_Trigger pins, respectively. Pins 2, 4, and 6 are reserved. Pins 1,3,5,7, and 9 are all grounds.

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

HW_Trigger is pulled to logical-high voltage (3.3V) on the board. Pin 10 is active and pin 9 is the corresponding ground. When a falling edge is detected (e.g., when shorting pins 10-9) and the program is idle, code execution is triggered. If the program is idle due to a WAIT OpCode, then the HW_Trigger will cause the program to continue to the next instruction. If the program is idle due to a STOP OpCode or a HW_Reset signal, then the HW_Trigger starts execution from the beginning of the program. When using the STOP OpCode, a HW_Reset or software reset (pb_reset()) needs to be applied prior to the HW_Trigger.

Figure 12 shows an example of the HW_Trigger signal.

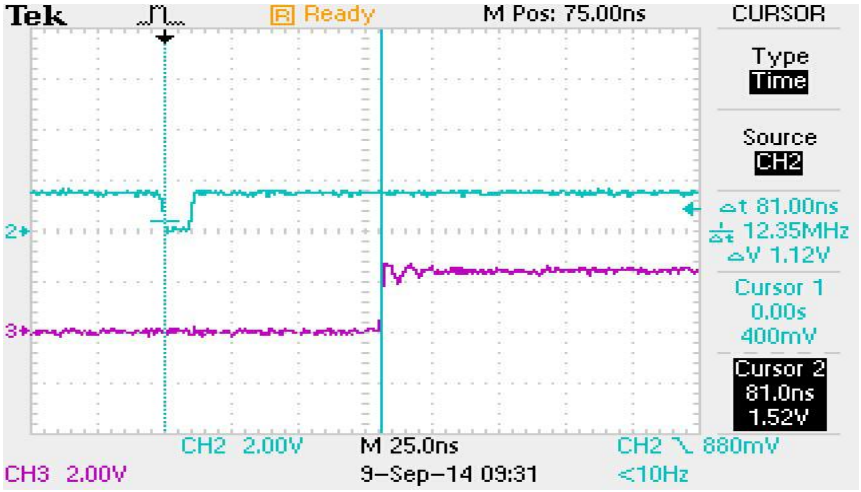


Figure 12: Demonstration of HW_Trigger signal. The blue shows the HW_Trigger signal, the pink shows one of the output flags. A latency of 80 ns is shown in this figure. Please refer to Instruction Set Architecture in Appendix I for more details on programming the duration of the WAIT latency. To trigger once, the trigger signal must begin at logical-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 logical-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 PulseBlasterESR-PRO.**

HW_Reset is pulled to logical-high voltage (3.3V) on the board. Pin 8 is active and pin 7 is the corresponding ground. It can be used to halt the execution of a program by pulling it low (e.g., by shorting pins 8-7). 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 (pb_start()) or by a hardware trigger.

Connector Locations for the SP19 board

Figure 13 shows the connector locations for the SP19 board. More details on the connectors can be found in the following sections. Please note the HW_TRIG/RESET header for the SP19 board has the same connections as the SP18A board.

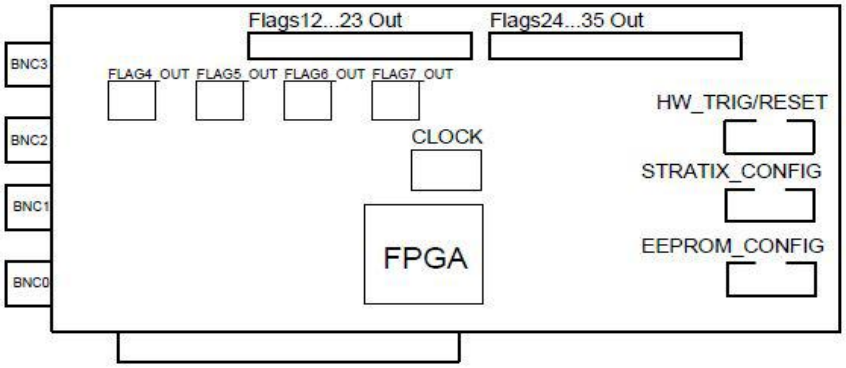


Figure 13: Connector locations for SP19.

SMA Headers for SP19

The four SMA headers provide access to output flag bits 4 through 7 and are labeled accordingly.

IDC Headers for SP19

Output flag bits 8 through 20 and the status bits are available on IDC headers. Table 2 shows the pin connectivity for these headers.

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

Table 2: SP19 IDC connector pin outs. Please refer to Figure 8 for pin numbering information.

Clock Oscillator Header

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

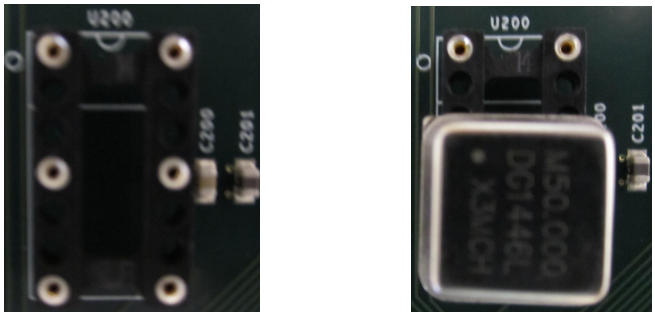


Figure 14: PulseBlaster ESR-PRO clock header. Both the bare header socket and the installed clock module are shown above. Please note the proper orientation of the 50 MHz clock.

Please take caution to provide a controlled signal at the correct frequency. The PulseBlasterESR-Pro requires a 50 MHz signal. A reliable option for this purpose is the Oven Controlled Clock Oscillator available for purchase. This component will provide a precision low ripple signal for all PulseBlaster boards, and ensure that appropriate signal voltages are applied to the board. Information on this product can be found in the “Related Products and Accessories” section.

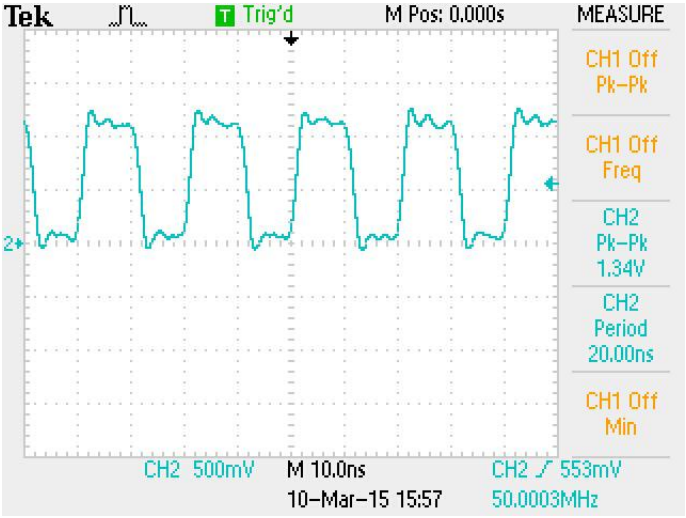


Figure 15: Example clock signal for SP19. Note that a small degree of voltage ripple is acceptable, so long as the voltage always remains above threshold for logical-high signals and below for logical-low signals.

NOTE: *The PulseBlasterESR-Pro requires a 3.3V TTL input signal.* ***A signal that is more than 3.3V or less than 0V will damage the device****.*

Appendix I: Controlling the PulseBlasterESR-PRO with SpinAPI

Instruction Set Architecture

Machine-Word Definition

The PulseBlasterESR-PRO pulse timing and control processor implements an 80-bit wide Very Long Instruction Word (VLIW) architecture. The VLIW is partitioned into fields dedicated to specific purposes, and every VLIW is viewed as a single instruction by the microcontroller. The maximum number of instructions that can be loaded onto the PulseBlasterESR-PRO is 4096. 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 and the longest being 2^{32} clock cycles.

Breakdown of 80-bit Instruction Word

All instructions have the same format and bit length, and all bit fields need to be filled. Table 3 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	OpCode	Delay Count
24 bits	20 bits	4 bits	32 bits

Table 3: Partitioning of the 80-bit Instruction Word (VLIW).

The 80-bit VLIW is broken up into 4 sections:

1. Output Pattern and Control Word: 24 bits.
2. Data Field: 20 bits.
3. OpCode: 4 bits.
4. Delay Count: 32 bits.

Output Pattern and Control Word

Table 4 shows the output pattern and control bit assignments of the 24-bit output/control word.

Bit #		Bit #	Function
23	Controls Pulse Length for BNC connectors	11	Output Connector labeled Flag0..11, Pin 12
22	Controls Pulse Length for BNC connectors	10	Output Connector labeled Flag0..11, Pin 11
21	Controls Pulse Length for BNC connectors	9	Output Connector labeled Flag0..11, Pin 10
20	Output Connector labeled Flag12..23, Pin 9	8	Output Connector labeled Flag0..11, Pin 9
19	Output Connector labeled Flag12..23, Pin 8	7	Output Connector labeled Flag0..11, Pin 8
18	Output Connector labeled Flag12..23, Pin 7	6	Output Connector labeled Flag0..11, Pin 7
17	Output Connector labeled Flag12..23, Pin 6	5	Output Connector labeled Flag0..11, Pin 6
16	Output Connector labeled Flag12..23, Pin 5	4	Output Connector labeled Flag0..11, Pin 5
15	Output Connector labeled Flag12..23, Pin 4	3	Output Connector labeled Flag0..11, Pin 4
14	Output Connector labeled Flag12..23, Pin 3	2	Output Connector labeled Flag0..11, Pin 3
13	Output Connector labeled Flag12..23, Pin 2	1	Output Connector labeled Flag0..11, Pin 2
12	Output Connector labeled Flag12..23, Pin 1	0	Output Connector labeled Flag0..11, Pin 1

Table 4: Output Pattern and Control Word Bits.

When the bit corresponding to an IDC output connector is one, the voltage will be high for the duration of the instruction. If the bit is zero, the voltage will be low for the duration of the instruction.

Short Pulse Feature

The Short Pulse feature utilizes the upper three bits of the instruction flag output bits (bits 21 to 23) to control the number of clock cycles output flags are enabled. This allows for short pulses down to a single clock cycle during the instruction period. Note that **all flags** are synchronized to this instruction period. It is possible to create pulses longer than this length by setting a channel on for multiple instructions, so the short pulse feature does not limit the maximum length of pulses. The following table provides information on using the short pulse feature.

SpinAPI Define	Bits 21-23	Clock periods	Pulse Length at 500 MHz (ns)
-	000	-	Always Low
ONE_PERIOD	001	1	2
TWO_PERIOD	010	2	4
THREE_PERIOD	011	3	6
FOUR_PERIOD	100	4	8
FIVE_PERIOD	101	5	10
ON	111	-	No Short Pulse

Table 5: Short Pulse Feature Characteristics.

When bits 21-23 are zero, the output flags remain low for the duration of the instruction. When bits 23-21 are from “000” to “101,” the programmed flag values will be outputted for the specified number of clock cycles. To disable the short pulse feature, bits 21 to 23 must be set to “111.”

Figure 16 gives an example of the Short Pulse feature. The example uses a 3 period duration. This example only shows 4 flags, but all flags will be affected.

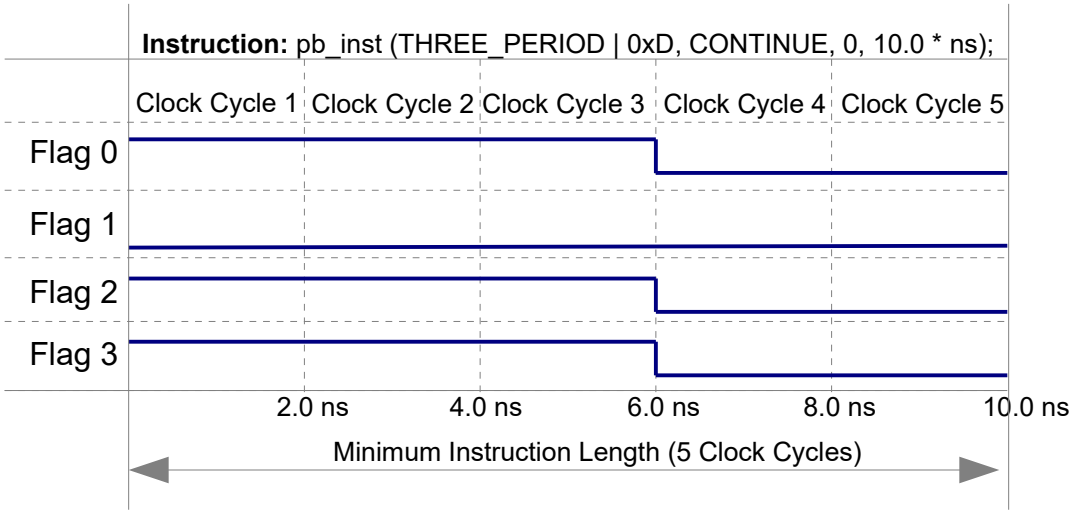


Figure 16: Example of the Short Pulse Feature. This example uses a 3 period duration. This example displays the output of 4 flags, but all flags are affected by the Short Pulse feature. Timing is done using a 500.0 MHz Clock.

NOTE: The short pulse functionality is firmware-dependent. Please inquire with SpinCore Technologies for upgrades or details. For firmware 17-11, a four clock cycle short pulse requires a minimum six clock cycle instruction length.

Data Field and OpCode

Please refer to the following table for information on the available instructions and their associated data field argument.

OpCode #	Inst	Inst_data	Function
0	CONTINUE	Ignored	Program execution continues to next instruction. *See note (1) and (2) following this table.*
1	STOP	Ignored	Stop execution of program. Aborts the operation of the micro-controller with no control of output states (all TTL values remain from previous instruction). Recommended that prior to the STOP OpCode a short interval (minimum six clock cycles) be added to set the output states as desired.
2	LOOP	Number of desired loops. This value must be greater than or equal to 1.	Specify beginning of a loop. Execution continues to next instruction. Data used to specify number of loops
3	END_LOOP	Address of beginning of loop	Specify end of a loop. Execution returns to beginning of loop and decrements loop counter.
4	JSR	Address of first subroutine instruction	Program execution jumps to beginning of a subroutine. *See Note (2).*
5	RTS	Ignored	Program execution returns to instruction after JSR was called. *See Note (2).*
6	BRANCH	Address of next instruction	Program execution continues at specified instruction
7	LONG_DELAY	Desired multiplier of the delay. 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. *See Note (2).*
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. The WAIT OpCode may not be used by the first instruction in memory

(1) For instructions longer than 8589 ms please use a LONG_DELAY instruction.
(2) For PulseBlasterESR-PRO-500 design 17-11, instructions with CONTINUE, JSR, RTS and LONG_DELAY OpCodes, require a minimum instruction length of at least 6 clock-cycles (12.0 ns).

Table 6: OpCode 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 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 that can communicate with the PulseBlasterESR-PRO 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 SpinAPI.

Please see the example programs for examples of how to use SpinAPI. If the programs have not been installed, then information to installing and finding them can be found in the “Installing the PulseBlasterESR-PRO” section. Reference documents for the API are available online at:

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

http://www.spincore.com/support/spinapi/SpinAPI_Main.shtml

Related Products and Accessories

1. Oven Controlled Clock Oscillator (sub-ppm stability) Figure 17. For ordering information, contact SpinCore at http://www.spincore.com/contact.shtml.

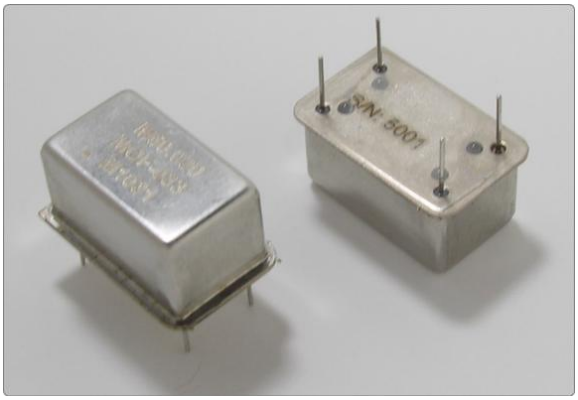


Figure 17: An Oven Controlled Clock Oscillator (or OCXO) with sub-ppm frequency stability is available for the PulseBlaster upon request.

2. Ribbon Cable with 2x13 IDC plug and DB-25 (Parallel port style*) connector on PC bracket. Please note this uses a standard computer ribbon cable. For ordering information, contact SpinCore at http://www.spincore.com/contact.shtml.

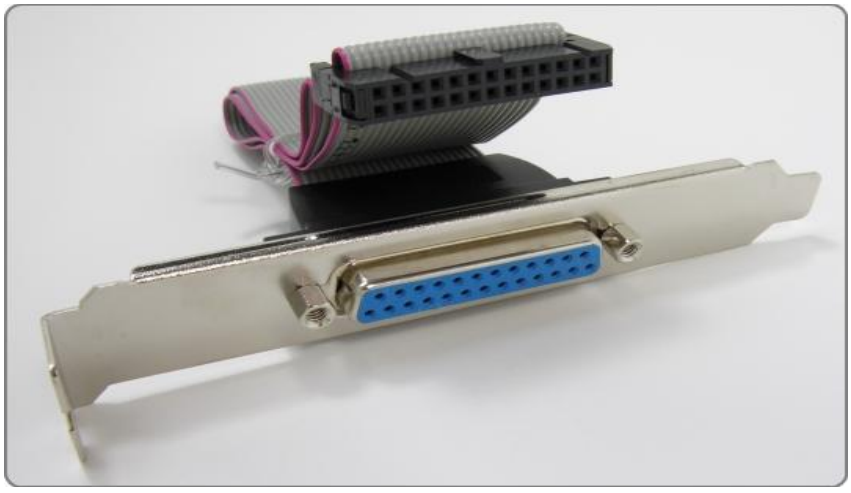


Figure 18: PulseBlaster Parallel Port Interface Cable.

3. SpinCore MMCX Adapter Board Figure 19 – This adapter board allows easy access to the individual bits of the PulseBlasterESR-PRO boards. This adapter board can be part of a package that includes 12 MMCX to BNC cables and three SMA to BNC adapters. However, this package can be changed to include any number of cables and any number of adapter boards.

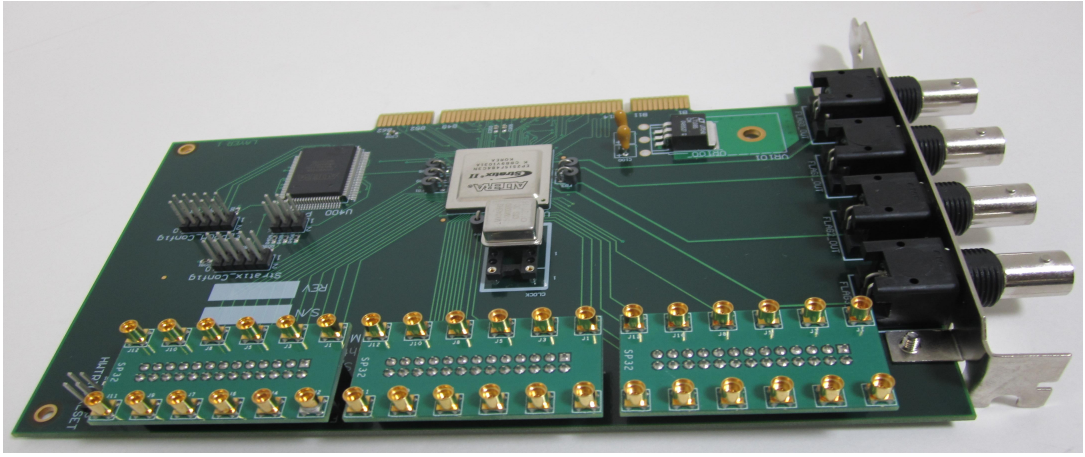


Figure19: MMCX Adapter Board allows easy access to individual bits

The SMA-BNC Adapter Board, shown in figure 20, provides easy access to four additional output signals from the back panel of your computer. SMA-SMA cables are available from SpinCore upon request. For ordering information, contact SpinCore at http://www.spincore.com/contact.shtml.

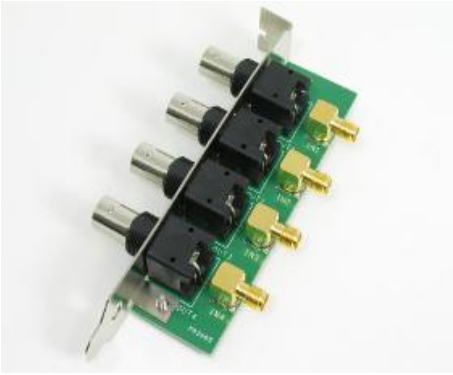


Figure 20: SMA-BNC Adapter Board is available for the SP18A and SP4B boards to access additional flag bits.

IDC to BNC Adapter Set-Up on an SP4B Board Figure 21 – Additional BNC output signals can be accessed using a set-up consisting of an IDC-MMCX adapter board (SP32), MMCX-SMA cables, and an SMA-BNC adapter board (SP29P).

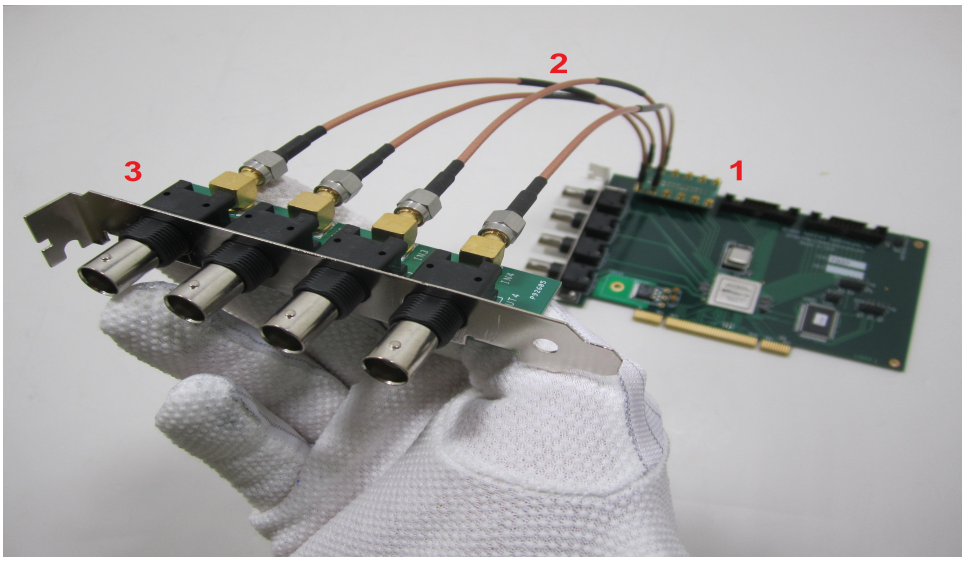


Figure 21: IDC to BNC Adapter Set-Up on an SP4B Board

4. SpinCore TTL Line Driver Figure 22 - A USB-powered device with four input channels and 8 output lines. Each output line is equipped with current driving capabilities to insure TTL voltage level over 50 Ohm loads. The SpinCore TTL Line Driver is the perfect tool to accompany any TTL device. Additional specifications, ordering information, and the manual for the TTL Line Driver are available at http://www.spincore.com/products/SpinCoreTTLLineDriver/SpinCoreTTLLineDriver.shtml.

Figure 22: TTL Line Driver assures TTL levels over 50 Ohm loads.

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

Contact Information

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

Telephone (USA):	**352-271-7383**
Fax (USA):	**352-371-8679**
Website:	http://www.spincore.com
Web Contact Form:	http://spincore.com/contact.shtml

Document Information

Revision history available at SpinCore.