# PulseBlasterESR-PRO-500-USB-SMA™

**(SP62)**

## Owner's Manual

**SpinCore Technologies, Inc.**
**https://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-500-USB-SMA™ is a high-speed, intelligent pulse/pattern/delay generator designed for outputting precisely timed TTL patterns. The intelligence of the PulseBlasterESR-PRO-500-USB-SMA 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 PulseBlasterESR-PRO-500-USB-SMA'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-500-USB-SMA processor is that the execution time for instructions is user programmable. This feature makes the PulseBlasterESR-PRO-500-USB-SMA 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.

The PulseBlasterESR-PRO-500-USB-SMA runs at 500 MHz. The SP62 measures 5.800 x 2.782 in (14.73 x 7.066 cm), offering unparalleled size-to-performance ratio. All 12 individually controlled digital output lines are routed to 50 ohm impedance matched SubMiniature version A (SMA) connectors. The shortest pulse can be as short as one clock period (2 ns) and the duration per instruction can range from 12 ns to 104 days. Up to 4096 instructions can be programmed onto the board. The board offers hardware pins to externally trigger and reset the board.

# Board Architecture

### *Block Diagram*

Figure 1 presents the general architecture of the PulseBlasterESR-PRO-500-USB-SMA system. The major building blocks are the static random access memory (SRAM), 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 IBC over the universal serial bus (USB).

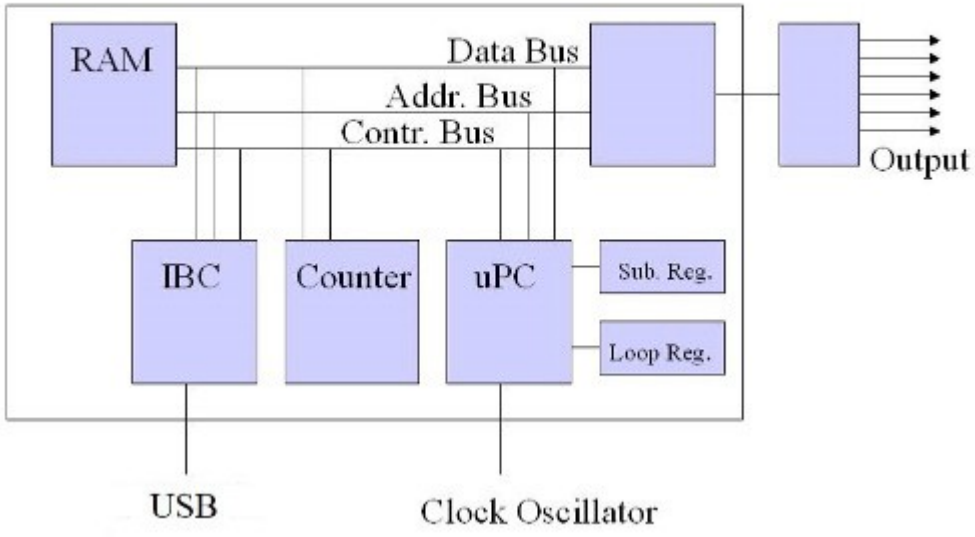


**Figure 1:** PulseBlasterESR-PRO-500-USB-SMA Board Architecture. The clock oscillator signal is derived from an on-chip PLL circuit typically using a 50 MHz on-board reference clock.

# Key Features

### *Output Signals*

The PulseBlasterESR-PRO-500-USB-SMA allows for 12 digital output signal lines, which are all routed to SMA connectors. The output signals are impedance matched to 50 ohm.

The 12 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 if more current is necessary beyond this, 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-500-USB-SMA is available with 500 MHz internal clock frequency. 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 six clock periods. This translates to a 12 ns machine instruction time at 500 MHz. The PulseBlasterESR-PRO-500-USB-SMA 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

The PulseBlasterESR-PRO-500-USB-SMA can be triggered and/or reset externally via dedicated hardware lines.

### Status Readback

The status of the pulse program can be read in hardware or software. The hardware status output signals consist of four insulation-displacement contact (IDC) connectors. The same output can be read through software using C. See Section IV (Connector Information for PulseBlasterESR-PRO-500-USB-SMA, page 20) for more details about the hardware lines and Section III (Using C Functions to Program the PulseBlasterESR-PRO-500-USB-SMA, page 16) for more details about the C function pb_read_status().

### *Summary*

The PulseBlasterESR-PRO-500-USB-SMA is a versatile, high-performance pulse/pattern TTL signal generator operating at speeds of 500 MHz and capable of generating pulses ranging from 2 ns to 104 days per instruction at delays/intervals ranging from 12 ns to 8.59 s per instruction (using a 500 MHz clock signal). It is connected via USB port and can accommodate pulse programs with highly flexible control commands of up to 4096 instruction words. Its high-current output logic bits are individually controlled with an unterminated output voltage of 3.3 V.

## Specifications

### *TTL Specifications*

- 12 individually controlled digital output lines (LVTTL levels, 3.3 V logical “one” unterminated)
- 12 SMA connectors
- 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
- 12 ns shortest interval
- 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
- 15 MHz max. re-triggering frequency

# II. Installation

## Installing the PulseBlasterESR-PRO-500-USB-SMA

Whenever installing or uninstalling the PulseBlasterESR-PRO-500-USB-SMA, always have it disconnected from the computer initially. Uninstall any previous version of SpinAPI.

1. Install the latest version of SpinAPI found at: https://www.spincore.com/support/spinapi/.
   - SpinAPI is a custom Application Programming Interface developed by SpinCore Technologies, Inc. for use with the PulseBlasterESR-PRO-500-USB-SMA 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.
3. Connect the PulseBlasterESR-PRO-500-USB-SMA to the computer.
   a) Plug a USB-B cable into the PulseBlasterESR-PRO-500-USB-SMA and the other end of the USB cable into the host computer. Be sure to support the board on the opposite side of the USB-B connector to prevent bending of the board when the connection is being made.
4. Power the PulseBlasterESR-PRO-500-USB-SMA board either by:
   a) Connecting the USB-C connector (PWR) with a USB-C cable and a power adapter that can supply 5V/3A, **OR**
   b) Connecting the KK Plus 396 connector (PWR2) with a cable that has a KK Plus 396 2 pin mating part where pin 1 is +5V and pin 2 is ground.

   Note: If powering through the USB-C connector (PWR), the board will not be powered on if the power supply is not capable of providing 5V/3A.
   **CAUTION:** Do not connect both the USB-C power (PWR) and KK Plus 396 2 pin cable (PWR2).
5. Plug the PC power cord back in, turn on the computer and follow the installation prompts if prompted.
6. 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.
   a) 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 the .exe programs to test your PulseBlaster.

# III. Programming the PulseBlasterESR-PRO-500-USB-SMA

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.

## The PulseBlaster Interpreter

The PulseBlasterESR-PRO-500-USB-SMA 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.

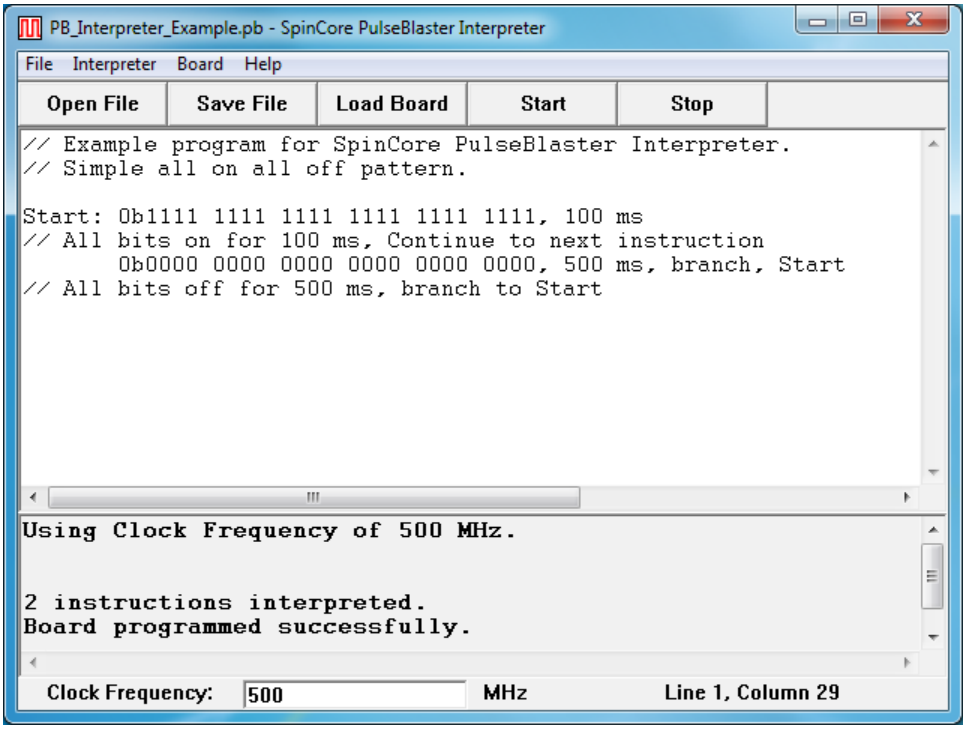


**Figure 2:** Graphical Interface of PulseBlaster Interpreter. The example shown creates a pulse that toggles all TTL bits on for 100 ms, then off for 500 ms, and repeats.

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 https://www.spincore.com/support/SPBI/Doc/.

## PulseBlaster.NET

PulseBlaster.NET is a graphical interface for creating pulse programs and loading them to the PulseBlasterESR-PRO-500-USB-SMA board. PulseBlaster.NET currently provides the simplest interface possible to pulse control. Figure 3 shows an example instance of the program.

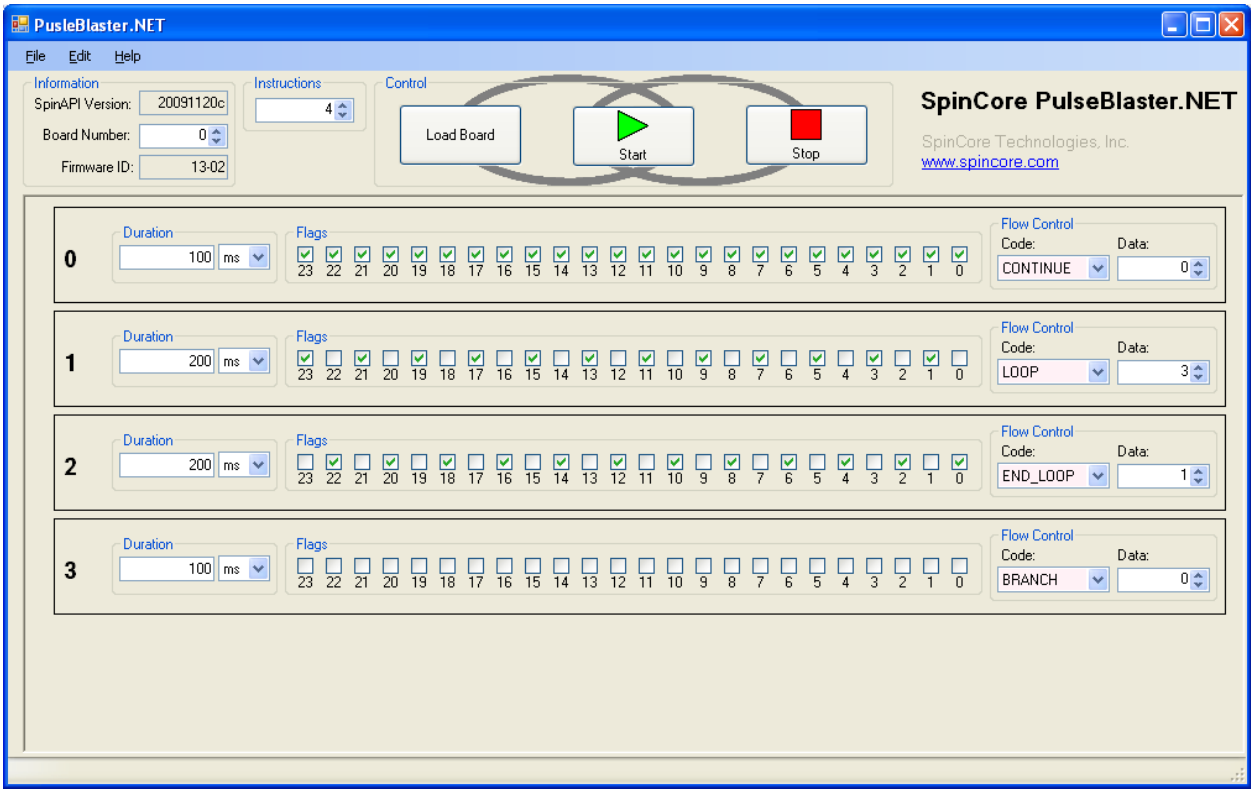


**Figure 3:** An example pulse program in PulseBlaster.NET. This example creates a pulse that has all TTL bits on for 100 ms, alternating bits on for 400 ms (looping three times), and then all bits off for 100 ms.

PulseBlaster.NET is available on the web from https://www.spincore.com/support/net/.

# LabVIEW Extensions

The SpinCore PulseBlaster LabVIEW Extensions (PBLV) provide the ability to program and control the functionality of PulseBlasterESR-PRO-500-USB-SMA board 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.

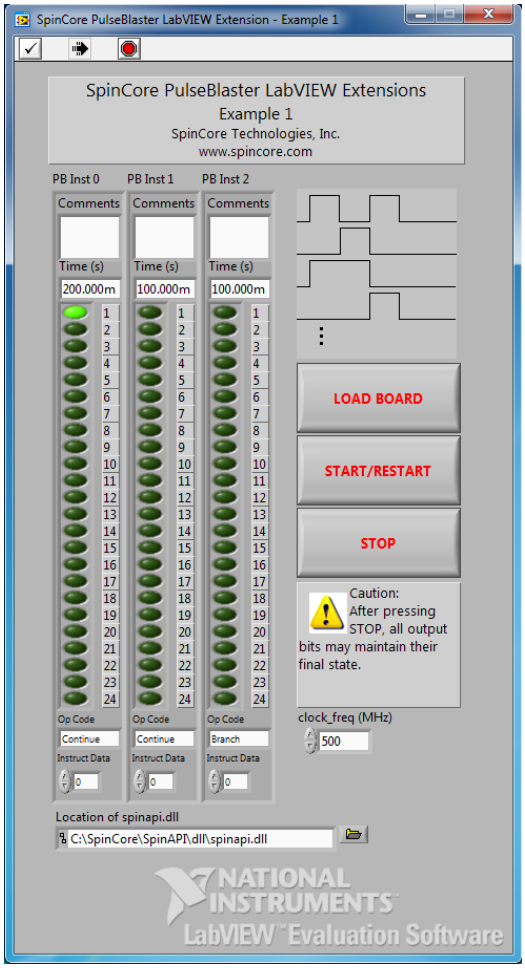


**Figure 4:** Example of PulseBlaster LabVIEW Extensions User Interface.

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 4 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-500-USB-SMA board. For more information and downloads please visit: https://www.spincore.com/support/PBLV/

# PulseBlaster MATLAB GUI

PulseBlaster MATLAB GUI is a graphical interface for creating pulse programs and loading them to the PulseBlasterESR-PRO-500-USB-SMA board. PulseBlaster MATLAB GUI currently provides the simplest interface possible to pulse control. Figure 5 shows an example instance of the program.

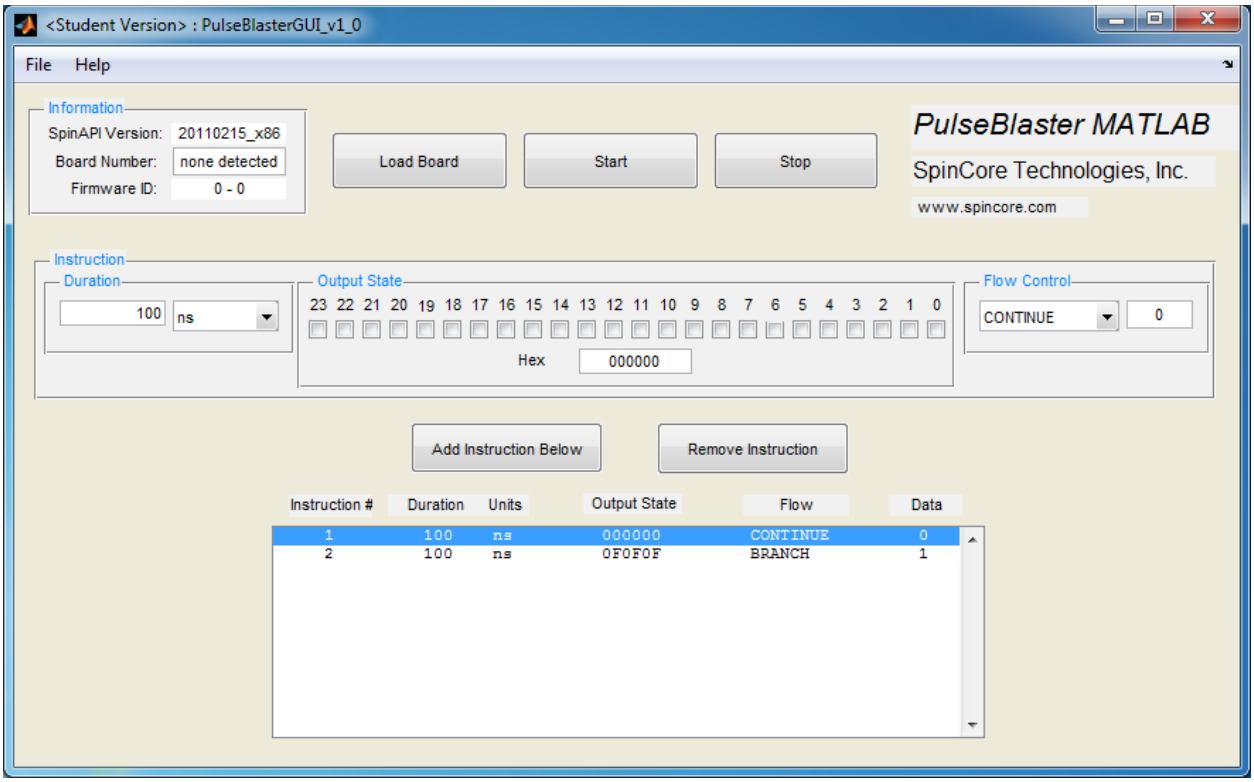


**Figure 5:** An example pulse program in PulseBlaster MATLAB GUI.

PulseBlaster MATLAB GUI is available at:

https://spincore.com/support/PulseBlasterMATLABGUI/pbmgui_main.shtml

## C/C++ Programming

The most dynamic and flexible way to program the PulseBlasterESR-PRO-500-USB-SMA 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. The instructions to compile on Windows can be found at https://www.spincore.com/support/spinapi/Windows_Help.shtml. After configuring the compiler, changing one of our example programs and recompiling the executable file for use with your PulseBlasterESR-PRO-500-USB-SMA board is as easy as clicking “Rebuild All” (see Figure 6 below).

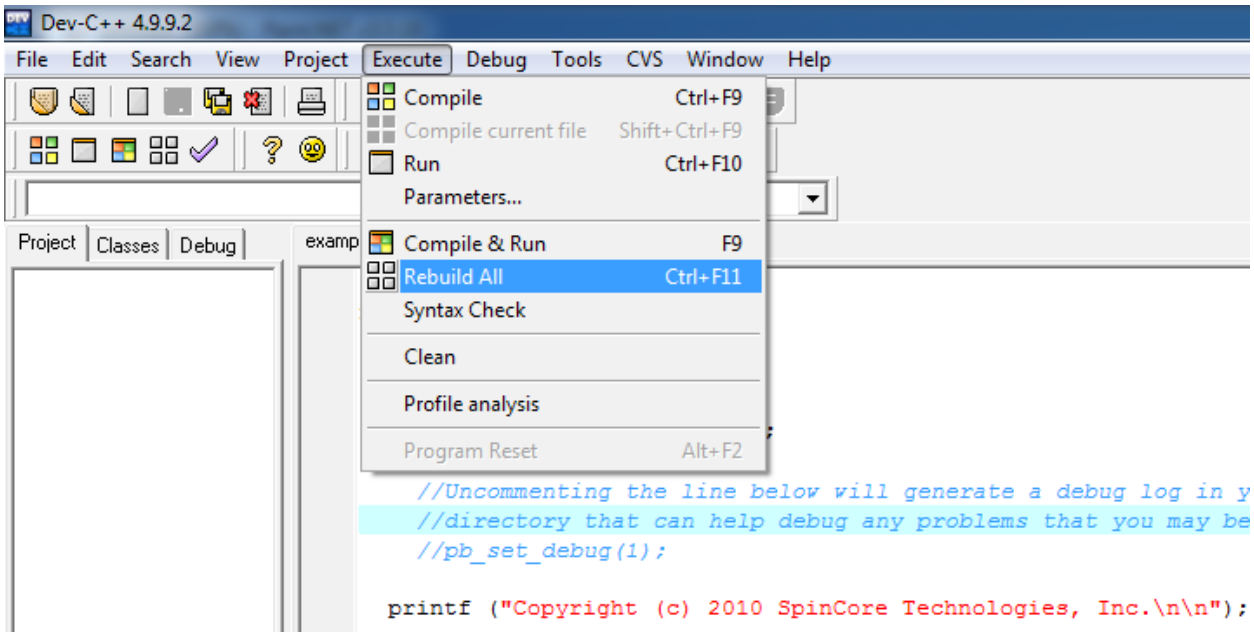


**Figure 6:** Compiling a C program to run the PulseBlasterESR-PRO-500-USB-SMA board is easy!

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 500 MHz for the
4:                          PulseBlasterESR-PRO-500 */
5:
6:   /*Start programming the Pulse Program*/
7:  pb_start_programming (PULSE_PROGRAM);
8:  start=pb_inst(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.
  - *Line 7*: pb_start_programming (PULSE_PROGRAM) must be called before using the pb_inst(..) function.
  - *Line 8*, 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.
  - *Line 9*, instruction 2: All bits off for 200.0 ms, then branch to “start.”
  - *Line 10:* 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-500-USB-SMA

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 10 ns. 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), and the header file (spinapi.h), must be in the working directory of your C compiler[1].

```
int pb_init();
```

Initializes the PulseBlasterESR-PRO-500-USB-SMA board. Needs to be called before calling any functions using the device. It returns a 0 on success or a negative number on an error.

```
int pb_close();
```

Releases the PulseBlasterESR-PRO-500-USB-SMA board. Needs to be called as last command in pulse program. It returns a 0 on success or a negative number on an error.

```
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.

```
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.

[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 pb_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 0xFFFFFF. 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 5 for details.

**int inst_data** – data to be used with the previous inst field. Please see Table 5 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 5).  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.

```
int pb_read_status();
```

Read status from the board. Each bit of the returned integer indicates whether the board is in that state. Bit 0 is the least significant bit.

- Bit 0 – Stopped
- Bit 1 – Reset
- Bit 2 – Running
- Bit 3 – Waiting

The SpinAPI package contains C examples in the PulseBlasterESR-PRO folder which is within the examples folder. In addition, many examples in the PulseBlaster24 folder will work for this model.

## Example Use of C Functions

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

#define PBESRPRO
#include "spinapi.h"

#define CLOCK 500.0    // 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-500-USB-SMA board.  The
pulse program
    // will continue to run even after this is called.
    pb_close();

    return 0;
}
```

# IV. Connecting to the PulseBlasterESR-PRO-500-USB-SMA

The connectors for the PulseBlasterESR-PRO-500-USB-SMA board are explained below.

## Connector Information for PulseBlasterESR-PRO-500-USB-SMA

The PBESR-PRO-USB-SMA board has SMA, Trigger/Reset/Status IDC, USB type B, USB type C, and 2-pin KK Plus 396 Molex connectors. The locations of these connectors are shown in Figure 7.

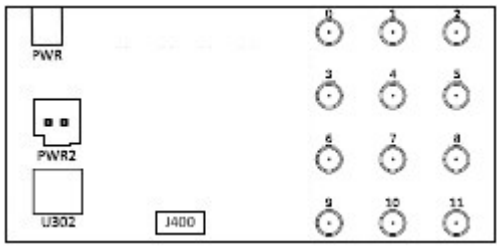


**Figure 7:** Top-down view of the SP62 showing connector locations. The reference numbers are printed next to the connector.

Please refer to the subsequent sections for detailed connector information.

## General Connectors

### *SMA Headers*

There are 12 SMA headers on the board which provide access to all of the digital outputs. These are labeled with Flag followed with a number which corresponds with the bit number.

If using a high input impedance oscilloscope to monitor the PulseBlasterESR-PRO-500-USB-SMA's output, place a resistor that matches the characteristic impedance of the transmission line in parallel with the coaxial transmission line at the oscilloscope input (e.g., a 50 Ω resistor with a 50 Ω transmission line, see Figures 8 and 9, on the next page). 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 8:** Left: BNC T-Adapter and Right: BNC 50 Ohm resistor.


**Figure 9:** 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.

### *Power Connector (PWR)*

The PulseBlasterESR-PRO-500-USB-SMA has a USB-C connector for supplying power and is labeled PWR. Use a power supply and cable capable of 5V/3A.

### *Power Connector (PWR2)*

The PulseBlasterESR-PRO-500-USB-SMA has a 2-pin KK Plus 396 Molex connector (part number: 2074790002) as an alternate method for supplying power and is labeled PWR2. To use this header, connect pin 1, which can be identified by a small 1 next to the header, to a +5 V DC source and pin 2 to ground.

**CAUTION: Connecting both the PWR2 and PWR connectors at the same time will damage the *PulseBlasterESR-PRO-500-USB-SMA*.**

### *U302 Connector*

The PulseBlasterESR-PRO-500-USB-SMA communicates with the PC via USB. This board has a USB 2.0 type B connector that is labeled as U302 and is used for data transfer. When connecting the USB B cable, please be sure to support the board on the opposite side of the header to ensure that the board does not bend when the connection is being made.

### *J400 Header*

The Trig/Res/Stat IDC connector information is shown in Figure 10 and Table 1, below. The Hardware Trigger and Hardware Reset are both low-true, so each of these pins would need to be shorted to ground to cause a trigger or reset, respectively. Please refer to the Status and Hardware Pins section for additional information about each pins functionality.

**CAUTION:** ***Applying voltages to the input pins that are greater than 3.3 V or less than 0 V will damage the PulseBlasterESR-PRO-500-USB-SMA***.

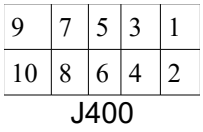


**Figure 10:** HW_Trig/Reset Header Pin-Out. Top-down view.

| Pin Assignments | | | |
|---|---|---|---|
| Pin# | | Pin# | |
| 1 | Ground | 2 | Hardware Trigger |
| 3 | Ground | 4 | Hardware Reset |
| 5 | Ground | 6 | WAITING |
| 7 | Ground | 8 | RUNNING |
| 9 | STOPPED | 10 | RESET |

**Table 1:** J400 Pin Assignments.

# Status and Hardware Pins

### Status Pins Description

**Stopped** – Driven high when the PulseBlasterESR-PRO-500-USB-SMA has encountered a STOP OpCode during program execution and has entered a stopped state.

**Reset** – Driven high when the PulseBlasterESR-PRO-500-USB-SMA is in a RESET state.

**Running** – Driven high when the PulseBlasterESR-PRO-500-USB-SMA is executing a program. The pin is driven low when the PulseBlasterESR-PRO-500-USB-SMA enters either a reset or idle state.

**Waiting** – Driven high when the PulseBlasterESR-PRO-500-USB-SMA has encountered a WAIT OpCode, and is waiting for the next trigger (either hardware or software) to resume operation. Note that the Running bit will also be high during a WAIT state.

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

### *Hardware Reset*

The PulseBlasterESR-PRO-500-USB-SMA has the HW_Reset hardware reset pin. HW_Reset is pulled to high voltage (3.3V) on the board and can be activated by a low voltage pulse (or shorting to ground). When the signal is activated during the execution of a program, the controller resets itself back to the beginning of the program. Program execution can be started from the beginning by either a software start command (pb_start()) or by a hardware trigger.

***NOTE:*** *The PulseBlaster requires a 3.3V input signal for HW_Reset.* ***Applying voltages to the input pins that are greater than 3.3V or less than 0V will damage the PulseBlasterESR-PRO-500-USB-SMA****.*

### *Hardware Trigger*

The PulseBlasterESR-PRO-500-USB-SMA has the HW_Trigger hardware trigger pin. HW_Trigger is pulled to high voltage (3.3V) on the board and can be triggered by a low pulse (or shorting to ground). When the falling edge is detected, and the program is idle, code execution is triggered. If the program is idle due to a WAIT instruction, the HW_Trigger will cause the program to continue to the next instruction. If the program is idle due to hardware or software reset, the HW_Trigger will start execution from the beginning of the program. If the board is in the STOPPED state, a HW_Reset or software reset (pb_reset() or pb_stop()) needs to be applied prior to the HW_Trigger, otherwise nothing occurs.

**NOTE:** *The PulseBlaster requires a 3.3V input signal for HW_Trigger.* ***Applying voltages to the input pins that are greater than 3.3V or less than 0V will damage the PulseBlasterESR-PRO-500-USB-SMA****.*

Figure 11, below, shows an example of the HW_Trigger signal with a latency of 80 ns. 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. If using a long TTL cable, make sure it is terminated and a buffer is used. If necessary, use an inverter or program the triggering device to match the high-low-high HW_Trigger signal. The input impedance of the HW_Trigger pin is 10 kOhms.

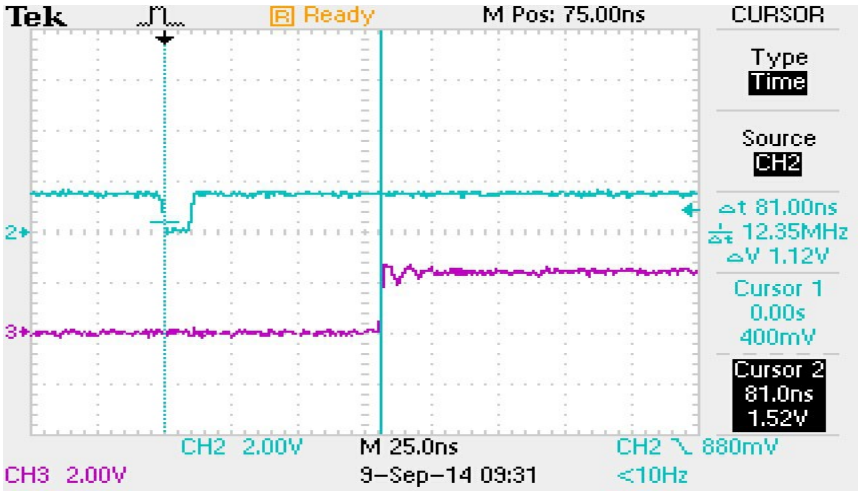


**Figure 11:** Demonstration of HW_Trigger high-low-high signal. The blue shows the HW_Trigger signal, the pink shows one of the output flags.
**Caution: applying voltages to the input pins that are greater than 3.3V or less than 0V will damage the PulseBlasterESR-PRO-500-USB-SMA.**

## Clock Oscillator Header

The PulseBlasterESR-PRO-500-USB-SMA comes with a crystal oscillator mounted on the oscillator socket 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-500-USB-SMA 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 PulseBlasterESR-PRO-500-USB-SMA 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.

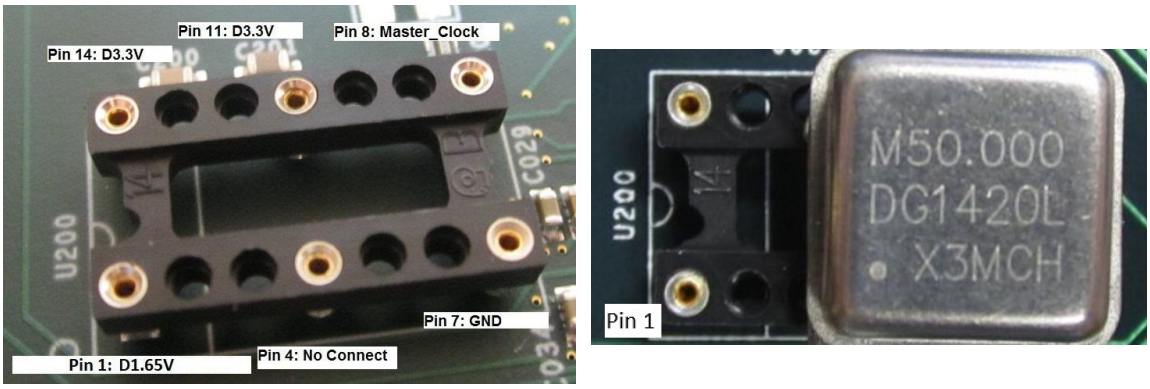


**Figure 12:** Both the bare header socket and the installed clock module are shown above. Please note the proper orientation of the 50 MHz clock.

Please take caution to provide a controlled signal at the correct frequency. The PulseBlasterESR-PRO-500-USB-SMA 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.

**NOTE:** *The PulseBlasterESR-PRO-500-USB-SMA requires a 3.3V TTL input signal.* ***A signal that is more than 3.3V or less than 0V will damage the device****.*

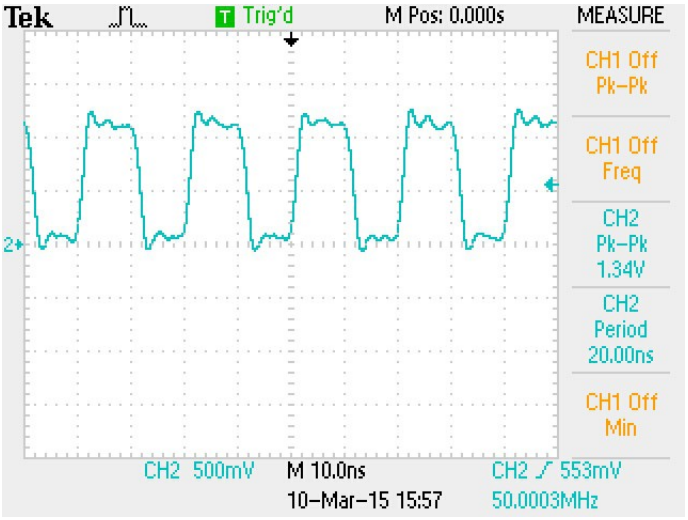


**Figure 13:** Example clock signal. Note that a small degree of voltage ripple is acceptable, so long as the voltage always remains above threshold for logical-high signals and below for logical-low signals.

# Appendix I: Controlling the PulseBlasterESR-PRO-500-USB-SMA with SpinAPI

## Instruction Set Architecture

### *Machine-Word Definition*

The PulseBlasterESR-PRO-500-USB-SMA 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-500-USB-SMA 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 six 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 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 | OpCode | Delay Count |
|---|---|---|---|
| 24 bits | 20 bits | 4 bits | 32 bits |

**Table 2:** 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 3 shows the output pattern and control bit assignments of the 24-bit output/control word.

| Bit # | Function | Bit # | Function |
|---|---|---|---|
| 23 | Controls Pulse Length of instruction | 11 | Output Connector labeled Flag11 |
| 22 | Controls Pulse Length of instruction | 10 | Output Connector labeled Flag10 |
| 21 | Controls Pulse Length of instruction | 9 | Output Connector labeled Flag9 |
| 20 | Unused | 8 | Output Connector labeled Flag8 |
| 19 | Unused | 7 | Output Connector labeled Flag7 |
| 18 | Unused | 6 | Output Connector labeled Flag6 |
| 17 | Unused | 5 | Output Connector labeled Flag5 |
| 16 | Unused | 4 | Output Connector labeled Flag4 |
| 15 | Unused | 3 | Output Connector labeled Flag3 |
| 14 | Unused | 2 | Output Connector labeled Flag2 |
| 13 | Unused | 1 | Output Connector labeled Flag1 |
| 12 | Unused | 0 | Output Connector labeled Flag0 |

**Table 3:** 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 | - | No Short Pulse |
| 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 4:** 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.

Figure 14 gives an example of the Short Pulse feature. The example uses a 3 period duration. This example only shows 4 bits, but all bits will be affected.

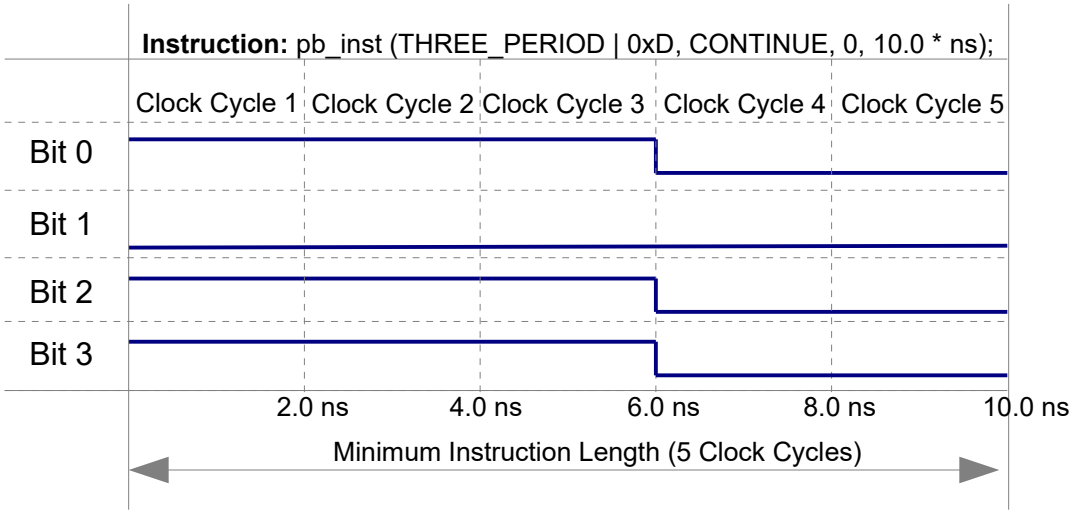


**Figure 14:** Example of the Short Pulse Feature. This example uses a 3 period duration. This example displays the output of 4 bits, but all bits are affected by the Short Pulse feature. Timing is done using a 500.0 MHz Clock.

**NOTE:** The Short Pulse functionality is firmware-dependent. Please inquire with SpinCore Technologies for details.

### *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) 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 may remain from current instruction). Recommended that prior to the STOP OpCode a short interval (minimum seven 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. |
| 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 | 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. |
| 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.*

**Table 5:** 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-500-USB-SMA 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 on installing and finding them can be found in the “Installing the PulseBlasterESR-PRO-500-USB-SMA” section. Reference documents for the API are available online at:

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

https://www.spincore.com/support/spinapi/

# Related Products and Accessories

1. Oven Controlled Clock Oscillator (sub-ppm stability) shown in Figure 15. For ordering information, please visit https://spincore.com/products/OCXO/ or contact SpinCore at https://www.spincore.com/contact.shtml.


**Figure 15:** An Oven Controlled Clock Oscillator (or OCXO) with sub-ppm frequency stability is available for the PulseBlasterESR-PRO-500-USB-SMA upon request.

2. SpinCore TTL Line Driver Figure 16 - 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 https://www.spincore.com/products/SpinCoreTTLLineDriver/SpinCoreTTLLineDriver.shtml.


**Figure 16**: TTL Line Driver assures TTL levels over 50 Ohm loads.

3. Other PulseBlasterESR-PRO models can be found at https://spincore.com/products/PulseBlasterESR-PRO/.
4. 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 https://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**
**Website:** https://www.spincore.com
**Web Contact Form:** https://www.spincore.com/contact/

# Document Information

Revision history available at SpinCore.