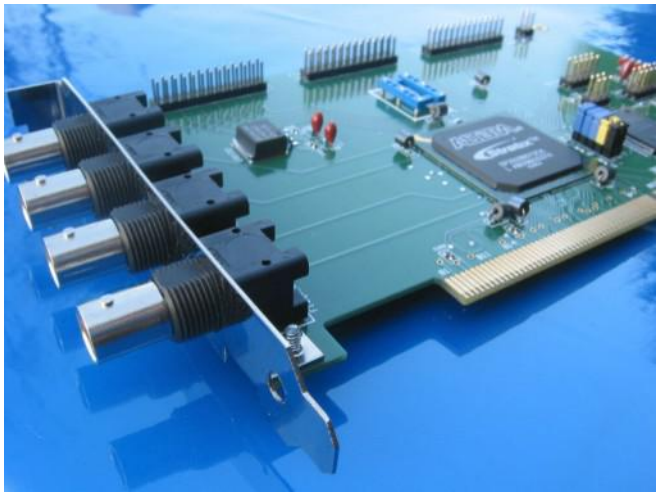
PulseBlasterESR™

Models PBESR-100 and PBESR-200

(PCI Board SP4B)

Owner's Manual



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

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

We appreciate your business!

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

Table of Contents

I. Introduction

Product Overview

The PulseBlasterESR™ device is a high-speed, intelligent, programmable pulse/pattern/word generator with 24 TTL (digital logic) output signals/channels/bits. Clock frequencies are 200 and 100 MHz for Models PBESR-200 and PBESR-100, respectively.

The intelligence of the PulseBlasterESR comes from an embedded microprogrammed controller core (uPC). The controller is able to execute instructions that allow it to control program flow. This means that the PulseBlasterESR understands Operational Control Codes(OpCodes), and will execute them much the same way as a general-purpose microprocessor does. Unlike the general-purpose processors, however, the PulseBlasterESR features a highly-optimized set of OpCodes that have been specifically designed for timing applications. Moreover, a unique and distinguishing feature of the PulseBlasterESR processor is that the execution time of instructions is user programmable.

The application-specific OpCodes and the user-programmable instruction time make the PulseBlasterESR processor capable of generating output pulse patterns at greatly varying update rates, ranging from nanoseconds to years per instruction, with a constant setting accuracy of just one clock period (e.g., a 10.0 ns setting accuracy when using a 100 MHz clock frequency), and suitable for the precise, i.e., clock-count-accurate, timing control of complex systems. Once programmed, the PulseBlasterESR operates autonomously.

Physically, PulseBlasterESR is a PCI board[1] (Peripheral Component Interconnect), and it is a high-speed version of the original PulseBlaster design. PulseBlasterESR boards are equipped with bracket-mounted BNC connectors.

PulseBlasterESR boards can be used with motherboards equipped with Intel, AMD, and other x86-type processors and at least one standard 32-bit PCI slot operating at 33 MHz. User interaction with the board is through the dedicated Application Programming Interface (API) package called SpinAPI, available free of charge. The SpinAPI package can be used with virtually all programming environments under Windows (including LabView and Matlab), and also with other operating systems, including Linux. A graphical programming environment for Windows is provided free of charge.

[1] CompactPCI versions of the PulseBlasterESR boards are also available - for more information, please contact SpinCore Technologies, Inc.

Board Architecture

Block Diagram

Figure 1 presents the general architecture of the PulseBlasterESR system. The major building blocks are the SRAM memory, the microcontroller (uPC), the integrated bus controller (IBC), the counter, and the output buffers (unlabeled). The entire logic design, including output buffers and SRAM memories, 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 controlled over the PCI bus.

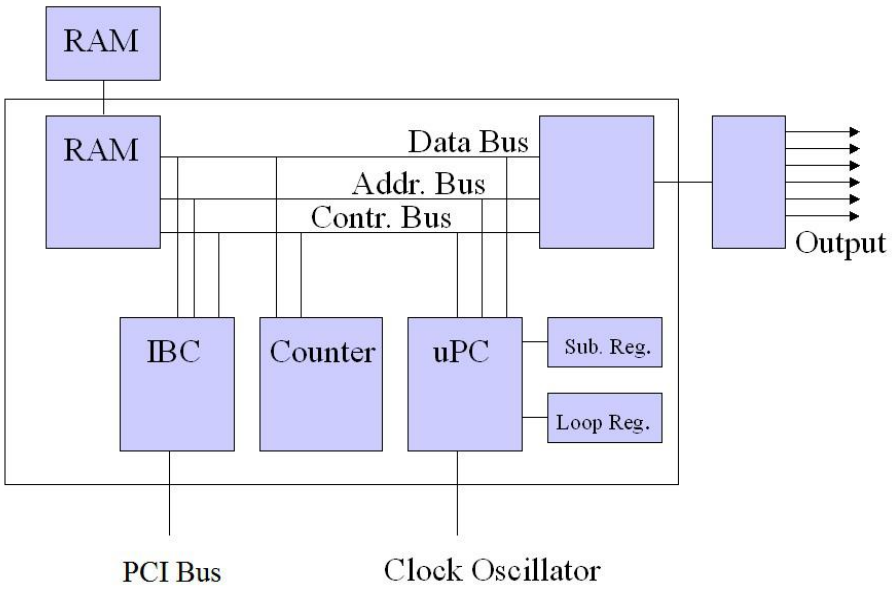


Figure 1: Block diagram of the PulseBlasterESR architecture. The clock oscillator signal is derived from an on-chip PLL circuit typically using a 50 MHz on-board reference clock.

In the section below, the *general characteristics* of the PulseBlasterESR are described. For information on the *internal operation* of the PulseBlasterESR processor, please see Appendix I.

Output signals

The PulseBlasterESR allows for 24 digital output signal lines. All 24 signal lines are routed to two sets of 26-pin IDC (dual row) on-board header connectors for flat ribbon cables. The first four output bits are also routed to four bracket mounted BNC connectors for use with coaxial cables. The output signals are impedance matched to 50 Ω (ohms).

The 24 individually controlled digital output bits are 3.3 V TTL standard, and are capable of delivering in excess of 25 mA per bit/channel.

Timing characteristics

PulseBlaster's timing controller accepts an internal (on-board) crystal oscillator (or external 3.3 V TTL clock signal) up to 125 MHz. The input frequency is internally increased by PLL up to 200 MHz. The innovative architecture of the timing controller allows the processing of either simple timing instructions (delays of up to 2^{32} –1 clock cycles, i.e., approximately 22 seconds at 200 MHz), or double-length timing instructions (up to 2^{52} – 1 clock cycles, nearly 255 days at 200 MHz). Regardless of the type of timing instruction, the timing resolution remains constant for any delay – just one clock period (e.g., 5 ns at 200 MHz).

The core timing controller has a very short minimum delay cycle – only five clock periods. This translates to a 25 ns shortest pulse/delay/update at 200 MHz.

Instruction set

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

External triggering

PulseBlasterESR 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 hardware lines are pulled high via 10 kΩ resistors, and the required control signals are "active low" (or short to ground).

Summary

PulseBlasterESR is a versatile, high-performance, programmable pulse/pattern/word TTL signal generator PCI board operating at speeds of up to 200 MHz and capable of generating pulses/delays/intervals ranging from 40 ns to 2.2×10^7 s per instruction. It has 24 bits/channels, and it can accommodate pulse programs with highly flexible flow control up to 4k (4096) instruction words. Its high-current output logic bits are independently controlled with an output voltage of 3.3 V (no-load voltage level). User interaction and control is through the SpinAPI dedicated software package.

Specifications

TTL Specifications

- 24 individually controlled digital output lines (TTL levels, 3.3 V logical “one”)
- 4 bracket mounted BNC connectors, impedance matched to 50 Ω
- Variable pulses/delays for every TTL line
- 25 mA output current per TTL line

Pulse Parameters

- 25 ns shortest pulse/interval (at 200 MHz)
- 2.2×10^7 s longest pulse/interval (at 200 MHz)
- 5 ns pulse/interval setting resolution (at 200 MHz)
- Up to 4k 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 – 8 clock cycle latency (40 ns at 200 MHz), up to 2.2×10^7 seconds in duration
- 10 MHz max. re-triggering frequency
- Status polling via hardware and software

II. Installation

Installing the PulseBlasterESR

Whenever installing or uninstalling the PulseBlasterESR, 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 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 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” folder and run all .exe programs to test your PulseBlaster.

III. Programming the PulseBlaster

There are several ways of programming the PulseBlaster board. In this section the PulseBlaster Interpreter, LabVIEW extensions, and C/C++ methods of programming will be introduced. In addition to these programming methods, the PulseBlaster board is programmable using Matlab or any interface that lets you utilize a C API package such as SpinAPI. Matlab and SpinAPI are described on our website http://www.spincore.com/. Finally, programming can be accomplished using C/C++ on Linux computers. All four methods are described in more details below.

The PulseBlaster Interpreter

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

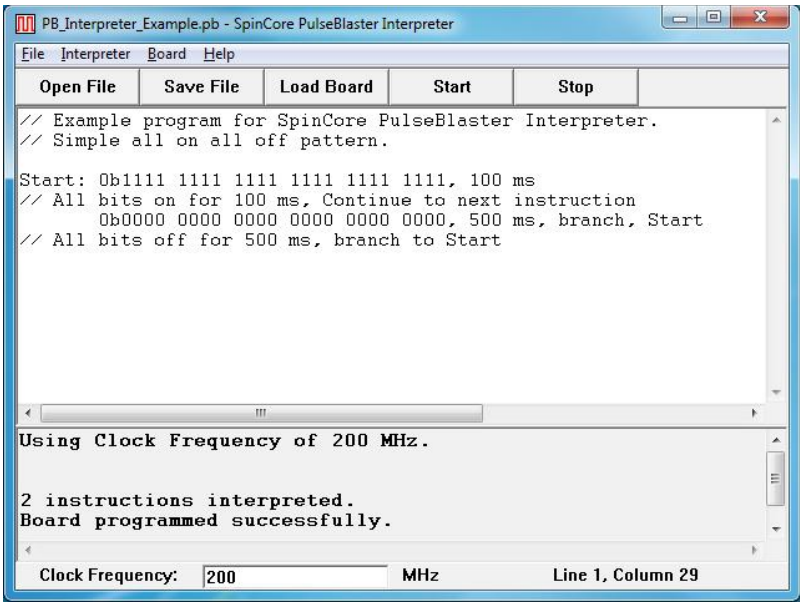


Figure 2: Graphical Interface of the PulseBlaster Interpreter. The example shown creates a pulse program that toggles all output TTL bits on for 100 ms, and all off for 500 ms.

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. 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 SpinCore PulseBlaster Interpreter 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.

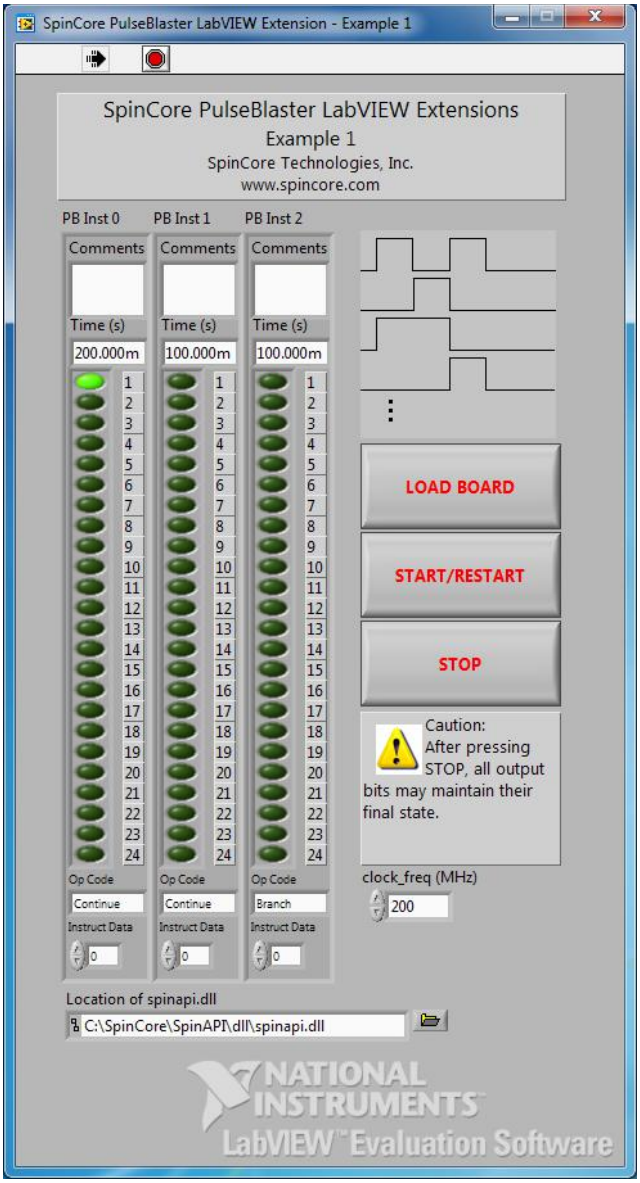


Figure 3: Example of PulseBlaster LabVIEW Extensions User Interface. The example shown has three instructions that toggle TTL bit 1 on for 200 ms and off for 200 ms.

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 PulseBlaster board. 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 PulseBlaster 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 interrupt aspects 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 our pre-configured compiler on our website, changing one of our example programs and recompiling the executable file for use with your PulseBlaster board is as easy as clicking “Rebuild All” (see Figure 4 below). You can get this compiler on our website at:

http://www.spincore.com/CD/Setup/SpinCore_SpinAPI_Tools_2007_07_11.exe

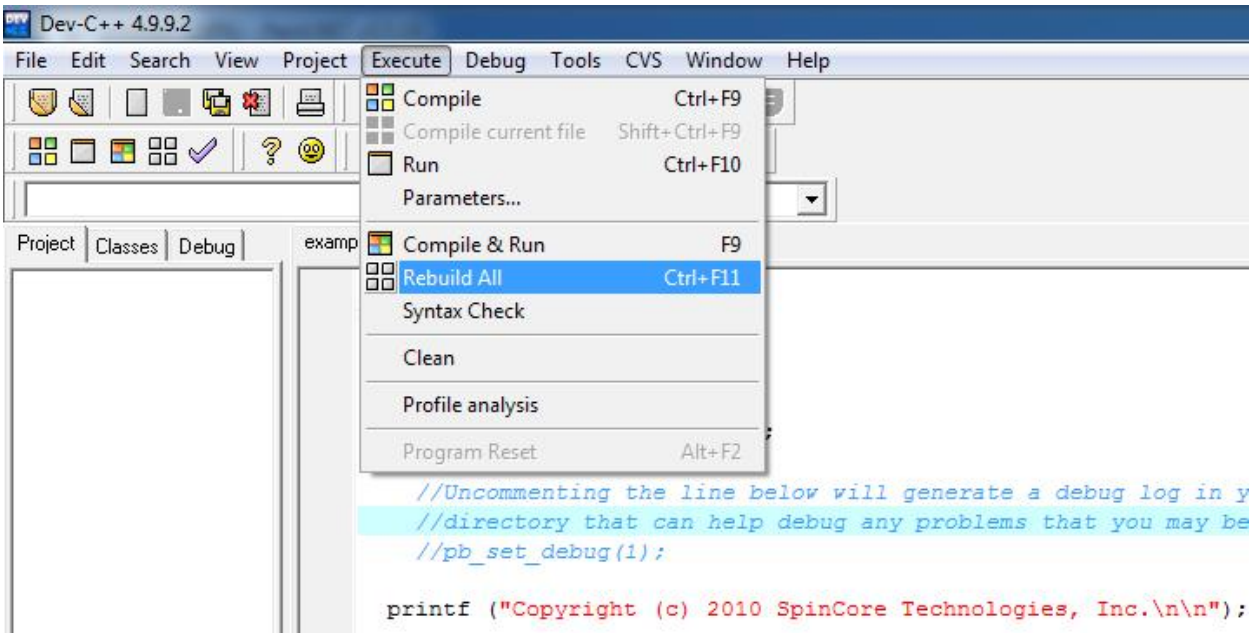


Figure 4: Compiling a C program to run the PulseBlaster board is easy.

Making changes to an example program requires understanding of only a few lines of code. The most important is the following line from pb24_ex1.c found in your SpinCore directory:

```
pb_inst(0xFFFFFF, CONTINUE, 0, 200.0*ms);
```

This line of code produces a high output on all the TTL bits lasting for 200 ms and then continues on to the next instruction. This is accomplished using the four parameters in the function call (parameters are located between parentheses and are separated by commas).

- The first is the hexadecimal 0xFFFFFF which corresponds to setting the 24 output bits to a logical high since it translates to a binary string of 24 1's.
- The second parameter is CONTINUE which means to proceed on to the next instruction after this one completes. Other examples for what this parameter could be are BRANCH or LOOP.
- The third parameter for a CONTINUE instruction is left as 0 because it is unnecessary for that particular instruction, but in the event of a BRANCH instruction for example, this parameter would be the target of the BRANCH instruction.
- The fourth parameter is 200.0*ms which means that this instruction will last for 200 ms.

A simple program to generate a square pulse will have two intervals (as in the GUI Interpreter described earlier), as shown below:

```
start= pb_inst(0xFFFFFF, CONTINUE, 0, 200.0*ms);
pb_inst(0x000000, BRANCH, start, 200.0*ms);
```

The first line of the code above corresponds to the logical "one" on all output bits. The second line corresponds to the logical "zero," after which the program branches (jumps) back to the beginning, thus resulting in a continuous generation of a square wave on all outputs.

A complete C program will have, in addition to the two lines above, the initialization section, the closing section, and, optionally, the (software) trigger to start the execution immediately upon launch of the program. For more detailed information on programming the board using C/C++ see the appendices starting on page 17.

Development under Linux

SpinAPI can be used with Linux on x86 processors. To begin running programs on Linux:

1. In the spinapi_source/ directory, type "make linux-direct". This will create a file called libspinapi.a which you can link your programs against to use SpinCore hardware. Note that this is not the same thing as the file in the dll/ directory. The dll/ directory contains files for use only with Windows and is not usable on linux.

2. To compile your program, you must link with both the spinapi and math libraries. For example, when compiling the "pb24_ex1.c" example program, you would use a command like (assuming both libspinapi.a and pb24_ex1.c are in the current directory):
   gcc -opb24_ex1 pb24_ex1.c -L. -lspinapi -lm

   This will create an executable called "pb24_ex1".

3. When running programs, you MUST have superuser privileges (i.e., be running as root). This is because the spinapi library needs to be able to gain access to the low-level hardware resources.

IV. Connecting to the PulseBlaster Board

Connector Information

BNC Bracket Connectors

The four bracket mounted BNC connectors output the first four bits of the output word, starting from the least-significant bit, as depicted below.

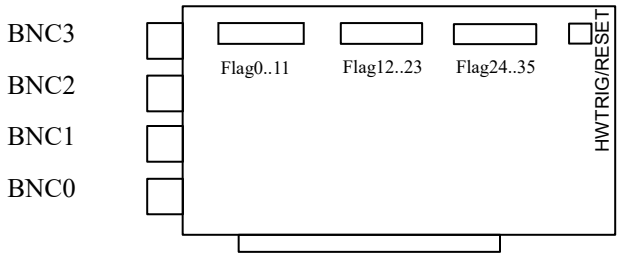


Figure 5: Connector Locations for the PulseBlasterESR board.

If using a high input impedance oscilloscope to monitor the PulseBlasterESR'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 Ω resistor.

Figure 7: BNC T-Adapter on oscilloscope with coaxial transmission line connected on the left and BNC 50 Ω resistor connected on the right, to terminate the line.

IDC Headers

There are three IDC headers on the PulseBlasterESR, which provide access to all digital 24 outputs as well as the status bits. 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 1-12 (Pin 13 is not connected), see Figure 8, below.

```
 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.

Each pin on an IDC header corresponds to a bit in the output word of an instruction. The association between bits and pins is shown in the Table 1. The Status pins present on one of the IDC connectors are defined 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.

Pin Assignments			
Header 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	*see*
6	Bit 5	Bit 17	*Appendix II*
7	Bit 6	Bit 18	*for additional*
8	Bit 7	Bit 19	*features*
9	Bit 8	Bit 20	-
10	Bit 9	Bit 21	-
11	Bit 10	Bit 22	-
12	Bit 11	Bit 23	-
13	-	-	-
14-26	Ground	Ground	Ground

Table 1: IDC connector pin assignments.

HWTrig/Reset Header

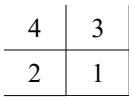


Figure 9: HWTrig/Reset Header Pin-out.

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

HW_Trigger (active-low) is pulled high by default using a 10 kΩ resistor, and pin 2 is active (pin 4 = GND). 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.

HW_Reset (active-low) is pulled high by default using a 10 kΩ resistor, 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 (pb_start()) or by a hardware trigger.

Clock Oscillator Header

The PulseBlasterESR 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 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.

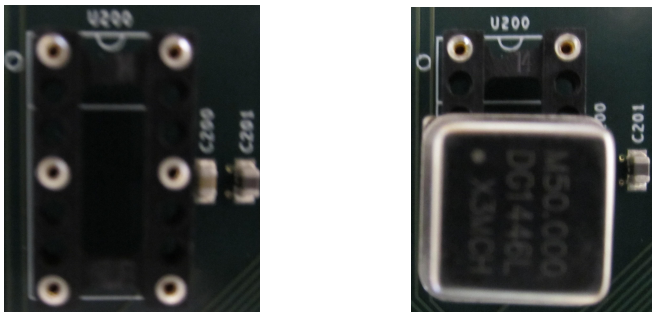


Figure 10: PulseBlaster ESR clock header. Both the bare header socket and the installed clock module are shown above. Please note the proper orientation of the 50 MHz clock.

Please take caution to provide a controlled signal at the correct frequency. The PulseBlasterESR 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.

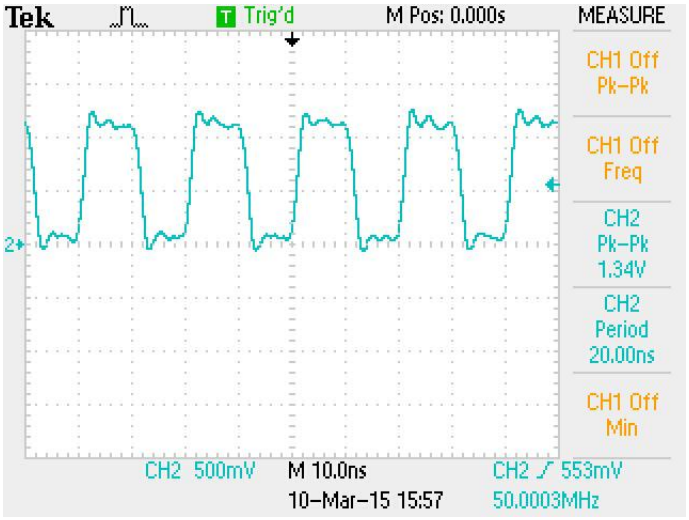


Figure 11: Example clock signal for SP19. Note that a small degree of voltage ripple is acceptable, so long as the voltage always remains above threshold for logical-high signals and below for logical-low signals.

NOTE: *The PulseBlasterESR 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 PulseBlaster with SpinAPI

About SpinAPI

SpinAPI is a control library which allows programs to be written to communicate with the PulseBlaster board. The most straightforward way to interface with this library is with a C/C++ program, and the API (Application Programming Interface) definitions are described in this context. In addition, virtually all programming languages and software environments (including software such as LabView and Matlab) provide mechanisms for accessing the functionality of standard libraries such as SpinAPI.

This Appendix contains a brief description of the architecture of the PulseBlaster processor, followed by the list of the most commonly used SpinAPI functions for the PulseBlasterESR board. Also included is an example C program. For more example programs, please download the latest SpinAPI package from the URL http://www.spincore.com/support/spinapi. A complete reference document for the API is available at the same URL as well.

Instruction Set Architecture of the PulseBlaster Processor

Machine-Word Definition

The PulseBlaster pulse timing and control processor implements an 80-bit wide Very Long Instruction Word (VLIW) architecture. The VLIW memory words have specific bits/fields dedicated to specific purposes, and every word should be viewed as a single instruction of the micro-controller. The maximum number of instructions that can be loaded to on-board memory varies by model.

The execution time of instructions can be varied and is under (self) control by one of the fields of the instruction word – the shortest being five clock cycles of the reference clock oscillator and the longest being 2^{32} - 1 clock cycles (for standard instructions). 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.

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

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

Figure 12: 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

The first 24 bits of the 80-bit machine word of the PulseBlaster processor function as the output bits. These 24 signals are routed to the on-board IDC headers as shown in Table 1 in the Manual. Bits 0-3 are also routed to BNC connectors on the PC bracket

The 24 output bits retain their value throughout the entire duration of the instruction represented by the machine word.

Data Field and Op Code

Table 2 (below) describes the available operational codes (hereafter referred to as Op-Codes) and their associated data fields. Note that the data field's function is dependent on the Op-Code.

Op Code #	Inst	Inst_data	Function
0	CONTINUE	Ignored	Program execution continues to next instruction
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 op-code 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 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	Desired multiplication factor for the “Delay Count” field. This value must be greater than or equal to 2.	For long interval instructions. Executes length of pulse given in the time field multiplied by the value in the data field.
8	WAIT	Ignored	Program execution stops and waits for software or hardware trigger. Execution continues to next instruction after receipt of trigger. The exit latency is equal to the delay value entered in the WAIT instruction line plus a fixed delay of 6 clock cycles. The WAIT OpCode may not be used by the first instruction in memory.

Table 2: Op Code and Data Field Description.

The PulseBlaster processor cycles through the instructions following the general steps:

- The machine word is read from memory and the 24-bit output pattern it contains is fed to the outputs of the board. These bits will then remain constant throughout the duration of the given instruction.
- The processor now remains in the given instruction for the time specified in the “Delay Count” field of the instruction.
- The processor takes the action specified by the instruction's OpCode and its associated data field, e.g., the program flow CONTINUES to next instruction, BRANCHES to a specific instruction, etc.

NOTE: The following exceptions apply:

- The processor will ignore the delay field of instructions containing the STOP command. Upon encountering an instruction containing the STOP command, the processor will halt execution immediately rather than after the delay specified in the Delay Count field.
- The Delay Field associated with the WAIT OpCode refers to the exit latency from the WAIT state upon detecting the trigger. The WAIT OpCode may not be used by the first instruction in memory. If you wish to have the processor wait for an external trigger at the beginning of your pulse routine, we suggest that you use two instructions: one with the CONTINUE OpCode (with a short delay) followed by an instruction which uses the WAIT OpCode.

Delay Count

The value of the Delay Count field (a 32-bit value) determines how long the current instruction should be executed. The SpinAPI package allows for the use of standard time units into the Delay Field. The allowed units are ns (nanoseconds), us (microseconds), and ms (milliseconds).

Using SpinAPI Functions to Program the PulseBlasterESR

In order to use the functions provided in the SpinAPI package, 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. 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 PulseBlaster board. Needs to be called before calling any functions using the PulseBlaster. Returns a negative number on an error or 0 on success.

`int pb_close();`

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

`void pb_core_clock(double clock_frequency);`

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. The function returns a 0 on success or a negative number on an error.

```
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 2 for details.

int inst_data – data to be used with the previous inst field. Please see Table 2 for details.

int length – duration of this pulse program instruction, specified in nanoseconds (multipliers of us and ms, defined as constants, can be used for executing longer intervals).

```
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 the board has been programmed, this instruction will start execution of pulse program. It returns a 0 on success or a negative number on an error.

```
int pb_stop();
```

Stops output of board. TTL outputs will remain in the state they were in when stop command was received. It returns a 0 on success or a negative number on an error.

Example Use of 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 theremaining 23 output bits will be ON for 80 ns, and c) all output
// bits will be OFF for 1us.

#include <stdio.h>
#define PBESRPRO
#include "spinapi.h"
#define CLOCK 200.0  //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_core_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 = 0xFFFFFF (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(0xFFFFFF, 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;
}
```

Appendix II: Fixed Clock Output Design

The PulseBlaster-ESR design 9-14 adds three custom clock output channels and an alternative trigger to the existing functionality of the PulseBlaster-ESR. The Clock Output Channels 1-3 will output the pulse pattern seen in Figure 13 upon being powered up.

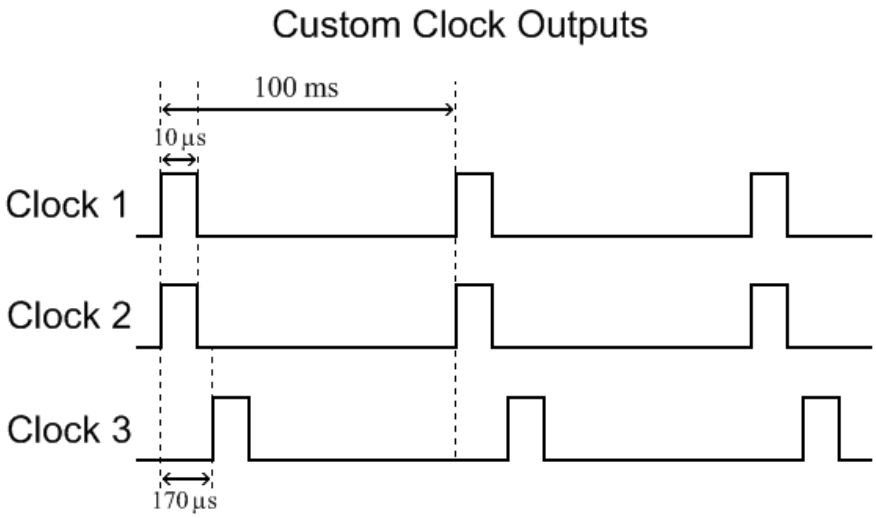


Figure 13: Timing Diagram for Custom Clock Output Channels (not drawn to scale).

The 24 adjustable TTL outputs can be triggered with an active low trigger pulse from the HWTrig/Reset header or with an active high trigger pulse from the designated pin on the Flag24..35 IDC header.

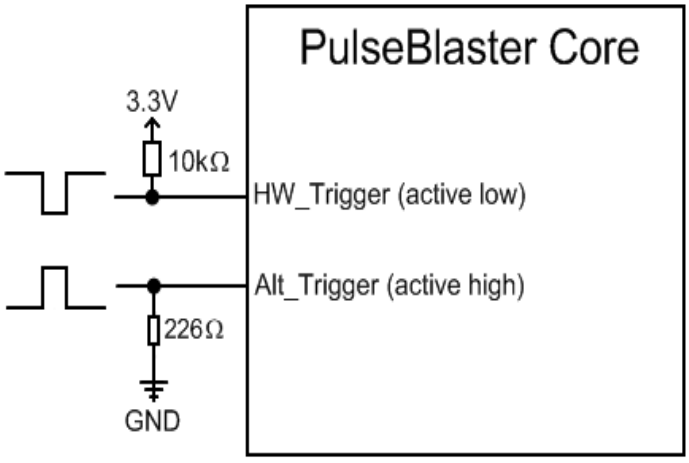


Figure 14: Triggering Diagram.

The BNC connectors, Flag0..11, and Flag12..23 headers still output the 24 channels from the PulseBlaster-ESR core (can be seen in Figure 5 on Page 15). The Flag24..35 IDC header contains all of the added functionality, shown in Figure 15 below.

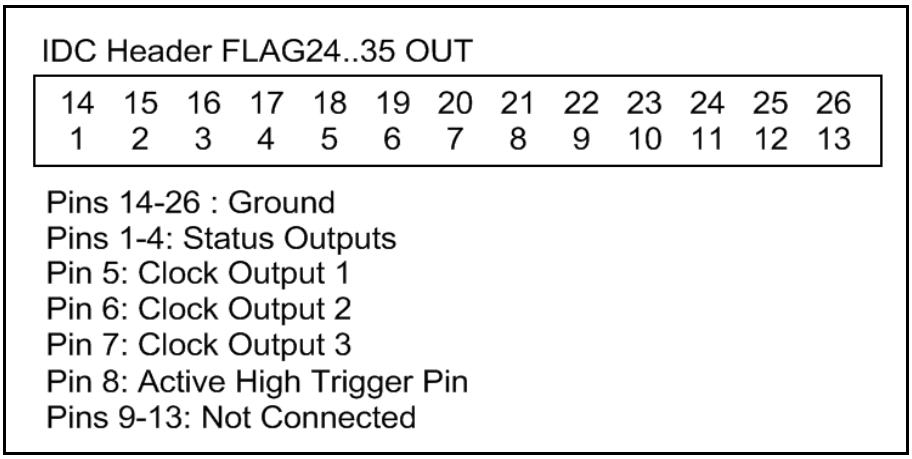


Figure 15: IDC header FLAG24..35 pin-out for Fixed Clock Output Design.

Appendix III: Custom BNC Triggering

The PulseBlaster-ESR design 9-15 builds upon design 9-14 (see Appendix II) to add a custom active low trigger on one of the BNC connectors. BNC3 will now act as a triggering connection instead of an output. The orientation for this is shown in Figure 17. The Fixed Clock Output Channels 1-3 will still be included, as shown in Figure 13 and Figure 15, above, in Appendix II. There are now three ways to trigger the board, as shown in Figure 16.

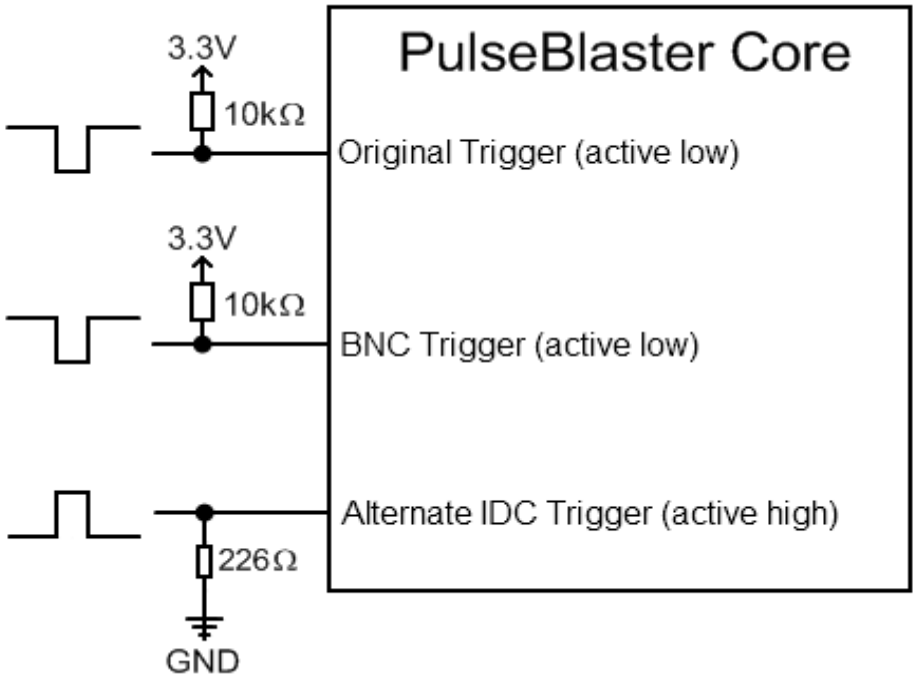


Figure 16: Triggering Diagram for Firmware #9-15 (Custom BNC Triggering).

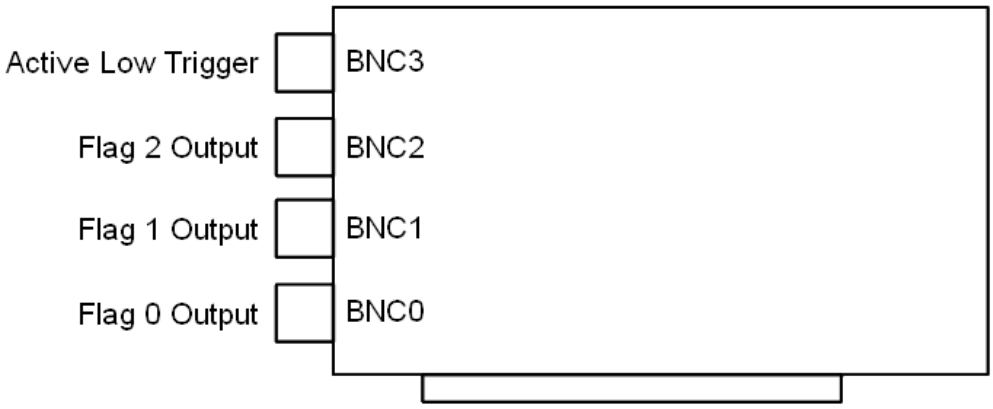


Figure 17: BNC Trigger shown on the PB-ESR Board.

Appendix IV: User Programmable Clock Output Design

The PulseBlaster-ESR design 9-11 builds upon the fixed clock design 9-14, described in Appendix II, and adds user programmability to customize the clock outputs. The alternative triggering functionality is not included in this design but can be added as an option. The Clock Output Channels 0-3 will output the pulse pattern seen in Figure 18 upon being powered up, with default period, clock high and offset values. The period will be 2 us, the clock high time will be 400 ns, and the offset will be 100 ns on all four channels. This feature has runs on a 50 MHz clock and has 32-bit resolution, meaning the shortest pulse possible is 20 ns and the longest interval is 85 seconds.

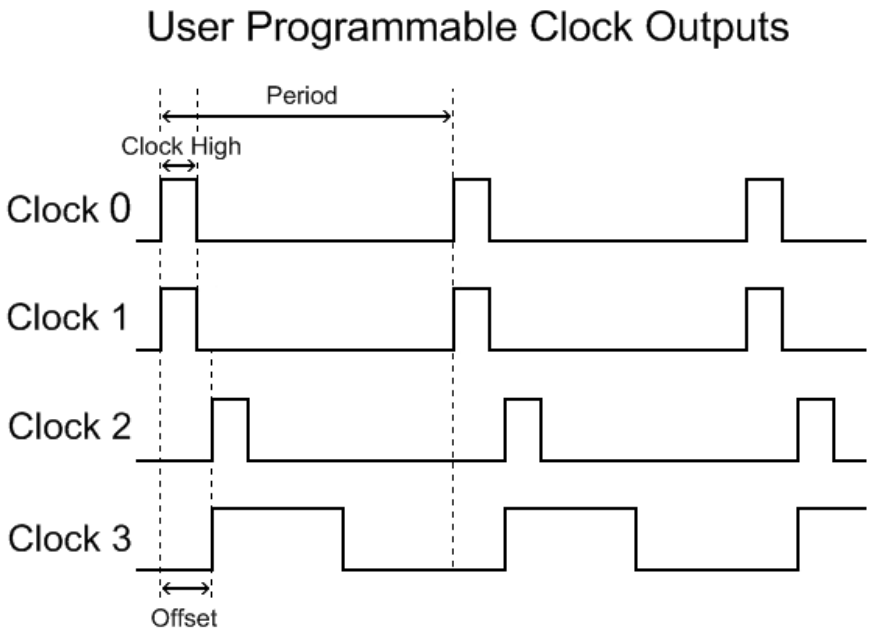


Figure 18: Timing Diagram for User Programmable Clock Outputs.

IDC Header FLAG24..35 OUT

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

Pins 14-26 : Ground
Pins 1-4: Status Outputs
Pin 5: Clock Output 0
Pin 6: Clock Output 1
Pin 7: Clock Output 2
Pin 8: Clock Output 3
Pins 9-13: Not Connected

Figure 19: IDC header FLAG24..35 pinout for User Programmable Clock Output Design.

Example Use of Programmable Clock Feature in C/C++

To program the three parameters into your board, you must use a function in SpinAPI called pb_set_pulse_regs. The function takes four parameters:

pb_set_pulse_regs(unsigned int channel, double period, double clock_high, double offset)

Channel must be an integer between 0 and 3 for the four channels. Period can be as long as 85 seconds or as short as 40 ns. Clock High and Offset must be between 20 ns and 85 seconds. The following example will produce outputs on all channels with equal period and clock high times but with varying delays.

```
pb_set_pulse_regs(0, 200.0 * us, 20.0 * us, 20.0 * us);
pb_set_pulse_regs(1, 200.0 * us, 20.0 * us, 40.0 * us);
pb_set_pulse_regs(2, 200.0 * us, 20.0 * us, 60.0 * us);
pb_set_pulse_regs(3, 200.0 * us, 20.0 * us, 80.0 * us);
```

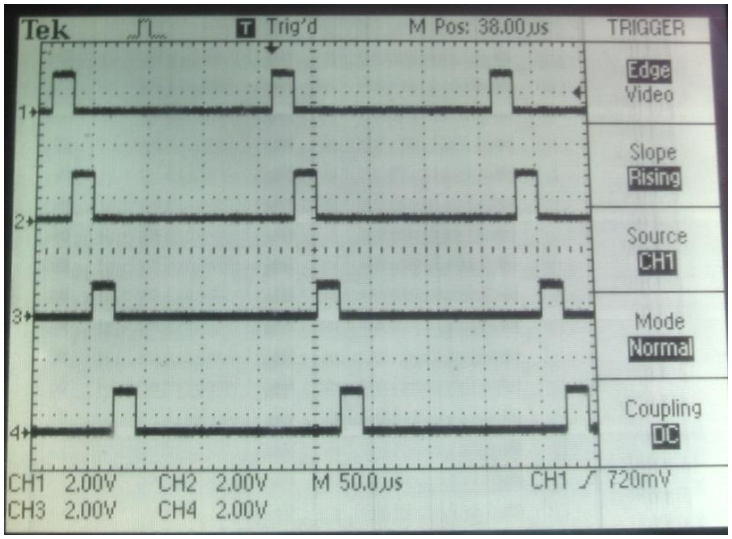


Figure 20: Scope output of example C program.

Related Products and Accessories

1. PulseBlasterESR-PRO – A High Speed Programmable Pulse and Digital Word Generator with a 500 MHz Clock Frequency. For more information, please visit http://spincore.com/products/PulseBlasterESR-PRO/

2. PulseBlaster-cPCI – A High Speed Programmable Multichannel Pulse Generator with a CompactPCI form factor. For more information, please visit http://www.spincore.com/products/PulseBlasterESR-CompactPCI/

3. PulseBlasterDDS – Built upon the PulseBlaster, the PulseBlasterDDS features programmable TTL outputs and RF Pulse Generation. For more information, please visit http://www.spincore.com/products/PulseBlasterDDS-300/

4. If you require alternate triggering, higher frequencies, a custom design, or custom interface cables, please inquire with SpinCore Technologies through our contact form, which is available at http://www.spincore.com/contact.shtml

5. SpinCore TTL Line Driver Figure 21 - 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 21: TTL Line Driver assures TTL levels over 50 Ohm loads.

6. IDC to BNC Adapter Set-Up Figure 22 - 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).

   For more information on the IDC-MMCX adapter board, see: http://www.spincore.com/products/IDC-MMCX_Adapter/IDC-MMCX_Adapter.shtml

   For more information on the SMA-BNC adapter board, see: http://spincore.com/products/SMA-BNC_Adapter/SMA-BNC_Adapter.shtml

   MMCX-SMA cables are available upon request.

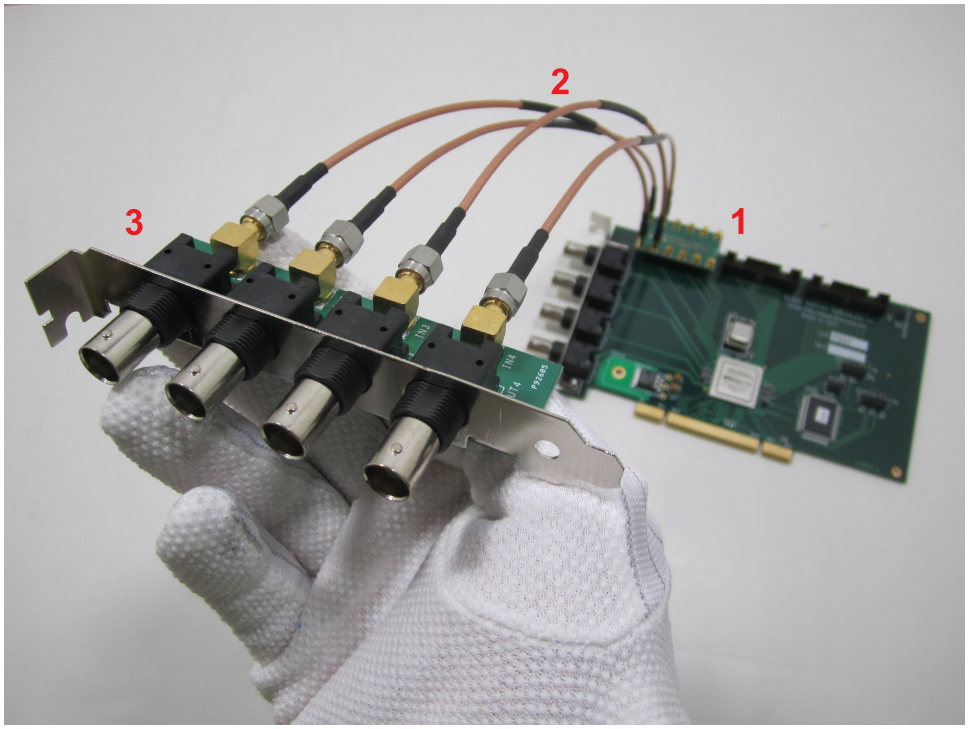


Illustration 22:
Figure 22: IDC to BNC Adapter Set-Up using (1) SP32, (2) MMCX-SMA cables, and (3) SP29P

7. If you require an Oven Controlled Clock Oscillator (sub-ppm stability) or other custom features, please inquire with SpinCore Technologies through our contact form, which is available at http://www.spincore.com/contact.shtml

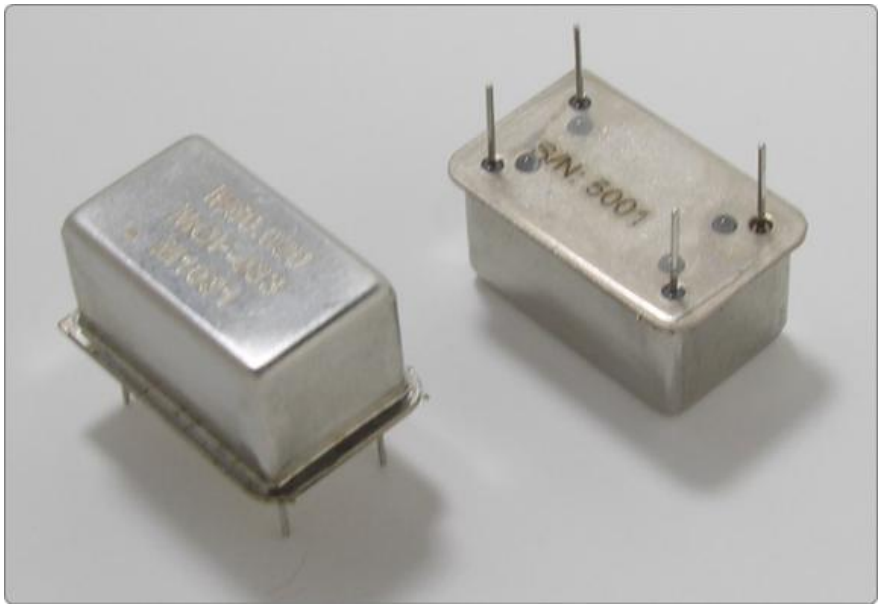


Figure 23: An Oven Controlled Clock Oscillator (or OCXO) with sub-ppm frequency stability is available for the PulseBlaster upon request.

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.