# PulseBlasterPlus!™

**Models:**
**PBP-24-100-32k, PBP-24-50-32k**
**PBP-16-100-32k, PBP-16-50-32k**

# 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 try 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 PulseBlasterPlus!™ is a 24-channel (Models 24-x) stand-alone general-purpose digital (TTL) pulse and pattern generator. Its state-of-the-art timing processor core, implemented in programmable logic, features a dedicated set of optimized instructions for efficient coding and generation of advanced flow of events, including loops, nested loops (16 levels deep), 20-bit loop counters, and nested subroutines, providing all the necessary timing control signals required for overall control and synchronization of complex systems. PulseBlasterPlus!™ provides users the ability to control their systems through the generation of high-precision TTL signals from a stand-alone box controlled by a simple Universal Serial Bus (USB) or RS-232 interface. With a 100 MHz master clock oscillator, pulses/delay ranging from 50 ns to over 2 years per single interval can be generated and independently controlled on all 24 channels. With a memory depth of 32k words, variable output rate, nested hardware loops and subroutines, PulseBlasterPlus! can generate agile and complex pulse sequences for demanding applications.

# System Architecture

## *Block Diagram*

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

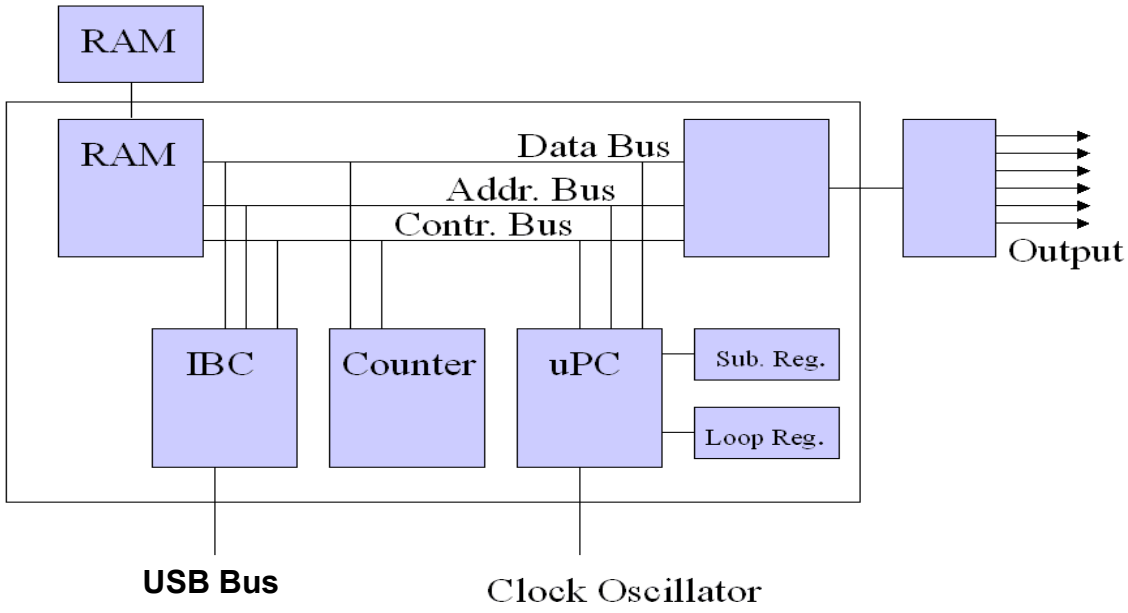


**Figure 1:** PulseBlasterPlus system architecture

## *Output Signals*

The PulseBlasterPlus!™ can output up to 24 digital (TTL) output lines. The individually controlled digital (TTL/CMOS) output bits are capable of delivering +-25 mA per bit/channel. The output lines can be set to either the 5 V or 3.3V I/O TTL logic standard (for more information on this feature, please contact SpinCore Technologies, Inc.). All 24 digital TTL output lines are available on the PC bracket-mounted Centronix-50 connector (models 16-x may feature a DB-25 connector).

## *Timing characteristics*

PulseBlasterPlus!™ timing controller runs off of an internal crystal oscillator of up to 100 MHz. The innovative architecture of the timing controller allows the processing of either simple timing instructions (delays of up to 4,294,967,296 clock cycles), or double-length timing instructions (up to 2^52 clock cycles long - nearly 2 years with a 100 MHz clock!). Regardless of the type of timing instruction, the timing resolution remains constant for any delay – just one clock period (e.g., 10 ns for a 100 MHz clock, or 100 ns for a 10 MHz clock).

The core timing controller has a very short minimum delay cycle – only five clock periods for internal memory (512 words) models. This translates to a 50 ns pulse/delay/update with a 100 MHz clock. The external memory models (up to 32k words) have a nine-clock period minimum instruction cycle.

### *Instruction set*

PulseBlasterPlus!™ 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 16 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 PulseBlasterPlus!™ can be triggered and/or reset externally via dedicated hardware lines. The two separate lines combine the convenience of triggering (e.g., in cardiac gating) with the safety of the "stop/reset" line. The required control signals are “active low” (or short to ground) and are connected to internal pull-up resistors.

### *Summary*

The PulseBlasterPlus™ is a versatile, high-performance pulse/pattern TTL generator operating at speeds of up to 100 MHz and capable of generating pulses/delays/intervals ranging from 50 ns to over 2 years per instruction. It can accommodate pulse programs with highly flexible control commands of up to 32k program words. Its high-current output logic bits are independently controlled and some bits are 5/3.3 V user-selectable.

## Specifications

### *Pulse Parameters*

- 24 individually controlled digital output lines (TTL levels, 3.3 V logical “one”)
- variable pulses/delays for every TTL line
- 25 mA output current per TTL line
- 50 ns shortest pulse/interval (at 100 MHz, Internal Memory Model, PBP24-512)
- 2 years longest pulse interval (at 100 MHz, with the use of the “Long Delay” instruction)
- 10 ns pulse/interval resolution (at 100 MHz)
- up to 32k pulse program memory words/instructions (Model PBP24-32k)
- external triggering and reset – TTL levels

### *Pulse Program Control Flow*

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

# II. Installation

## Installing the PBPlus Driver for Windows Operating Systems

1. Go to http://www.spincore.com/CD/PulseBlasterPlus/v1/Drivers/ and download the zip file.
2. Unzip the files to their own directory.
3. Power on the PulseBlasterPlus! and connect it to the host computer with a USB cable (provided).

**Windows 98:**

4. an "Add New Hardware Wizard" dialog box will appear. Click the **Next** Button.
5. Select **Search for the best driver for your device (Recommended)** and click the **Next** button.
6. Select **Specify a location** and click the **Browse…** button.
7. Browse to the folder you created when downloading the zip file. Choose the "win-98-me" folder and click the **OK** button.
8. You will return to the previous screen. Now, click the **Next** button.
9. Windows is now ready to install the PBPlus driver. When you see the following screen, click the **Next** button.
10. Windows will now copy the necessary files to your PC. When the process completes, click **Finish**.

**Windows XP:**

6. After booting, an "Add New Hardware Wizard" dialog box will appear. Choose "Install from a specific location" and click the **Next** Button.
7. Select **Search for the best driver for your in these locations.** Click the browse button and navigate to the folder you created when downloading the zip file and click on the "win-xp" directory and click the **OK** button.
8. Click the **Next** button.
9. Click the **Continue Anyway** button.
10. Windows will now copy the necessary files to your PC. When the process completes, click **Finish**.

## Testing Control of the PulseBlasterPlus

To test if the drivers are working properly, go to http://www.spincore.com/CD/PulseBlasterPlus/v1/TTL_Examples/ and download pbplus_ttl_ex1.exe and pbplus.dll.

Run pbplus_ttl_ex1.exe. All output pins on the PulseBlasterPlus! should toggle on and off with a period of 1s using 50MHz clock.

***The PBPlus system is now ready for use!***

# III. Programming the PBPlus

## Instruction Set Architecture

### *Machine-Word Definition*

The PulseBlasterPlus pulse timing and control processor implements an 80-bit wide Very Long Instruction Word (VLIW) architecture. The VLIW memory words have specific bits/fields dedicated to specific purposes, and every word should be viewed as a single instruction of the micro-controller. The maximum number of instructions that can be loaded to on-chip memory is 512 (Model PBP24-512, also called the "Internal Memory Model"); the maximum number of instructions that can be loaded to the on-system memory is 32k (Model PBP24-32k, also called the "External Memory Model). The execution time of instructions can be varied and is under (self) control by one of the fields of the instruction word – the shortest being five clock cycles for the "Internal Memory Model" and nine clock cycles for the "External Memory Model." All instructions have the same format and bit length, and all bit fields have to be filled. Figure 2 shows the fields and bit definitions of the 80-bit instruction word.

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

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

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

### *Breakdown of 80-bit Instruction Word*

The 80-bit VLIW is broken up into 4 sections

1. Output Pattern and Control Word - 24 bits
2. Data Field - 20 bits
3. OP Code - 4 bits
4. Delay Count - 32 bits.

### *Output Pattern and Control Word*

Please refer to Table 1 for output pattern and control bit assignments of the 24-bit output/control word.

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

**Table 1:** Output Pattern and Control Word Bits

### *Data Field and Op Code*

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

| Op Code # | Inst | Inst_data | Function |
|---|---|---|---|
| 0 | CONTINUE | Ignored | Program execution continues to next instruction |
| 1 | STOP | Ignored | Stop execution of program (*Note all TTL values remain from previous instruction, and analog outputs turn off) |
| 2 | LOOP | Number of desired loops. This value must be greater than or equal to 1. | Specify beginning of a loop. Execution continues to next instruction. Data used to specify number of loops |
| 3 | END_LOOP | Address of beginning of loop | Specify end of a loop. Execution returns to begging of loop and decrements loop counter. |
| 4 | JSR | Address of first subroutine instruction | Program execution jumps to beginning of a subroutine |
| 5 | RTS | Ignored | Program execution returns to instruction after JSR was called |
| 6 | BRANCH | Address of next instruction | Program execution continues at specified instruction |
| 7 | LONG_DELAY | Number of desired loops. This value must be greater than or equal to 2. | For long interval instructions. Data field specifies a multiplier of the delay field. Execution continues to next instruction |
| 8 | WAIT | Ignored | Program execution stops and waits for software or hardware trigger. Execution continues to next instruction after receipt of trigger. The latency is equal to the delay value entered in the WAIT instruction line plus a fixed delay of 6 clock cycles. |

**Table 2:** Op Code and Data Field Description

*Delay Count*

The value of the Delay Count field (a 32-bit value) determines how long the current instruction should be executed. The allowed minimum value of this field is 0x00000002 for PBP24-512 and 0x00000006 for PBP24-32K and the allowed maximum is 0xFFFFFFFF. The timing controller has a fixed delay of three clock cycles and the value that one enters into the Delay Count field should account for this inherent delay.

# Using C Functions to Program the PulseBlasterPlus

A series of functions have been written to control the system and facilitate the construction of pulse program instructions.

In order to use these functions, the DLL (PBPlus.dll), the library file (PBPlus.lib), the header files (PBPlus.h and pbdfuncs.h), and source file (pbdfuncs.cpp) must be in the working directory of your C compiler[1].

**`int pb_init();`**

Initializes PulseBlasterPlus system. Needs to be called before calling any functions using the PulseBlasterPlus. Returns a negative number on an error or 0 on success.

**`int pb_close();`**

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

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

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

**`int start_programming(int device);`**

Used to initialize the system to receive programming information. It accepts a parameter referencing the target for the instructions. The only valid value for **device** is PULSE_PROGRAM, It 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 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 5 being on, and all other bits being off.

**int inst** – determines which h 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.

[1] These functions and library files have been generated and tested with the MS Visual Studio 6 environment. Support and updated functions/DLLs for other environments may be provided upon request if available.

**int length** – duration of this pulse program instruction, specified in nanoseconds. The duration of the instruction can be specified in seconds, milliseconds, or microseconds by multiplying the value by the predefined constants s, ms, or us, respectively.

**`int stop_programming();`**

Used to tell that programming the system is complete. System execution cannot start until this command is received. It returns a 0 on success or a negative number on an error.

**`int start_pb();`**

Once system 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 stop_pb();`**

Stops output of system. Analog output will return to ground, and TTL outputs will remain in the state they were in when stop command was received. It returns a 0 on success or a negative number on an error.

### *Example Use of C Functions*

```
/*
Example 1 for PBPLUS
Toggles flags with 1s period using 50MHz clock.
Last updated 3/22/05
*/

#include "pbdfuncs.h"
#include <stdio.h>

int main()
{

//set clock
set_clock(50*MHz);

//init device
pb_init();

//start programming device
start_programming(PULSE_PROGRAM);

//continue to next instruction after one second with all flags off
pb_inst(0, CONTINUE, 0, .5*s);

//branch to first instruction after one second with all flags on
pb_inst(MAX_FLAG_BITS, BRANCH, 0, .5*s);

//stop programming
stop_programming();

//start program
start_pb();
```

```
//close device
pb_close();

return 0;

}//end main()
```

***A more complex program using C Functions is provided in Appendix I.***

# IV. Connecting to the PulseBlasterPlus

All input-output connectors have been mounted at the back of the PulseBlasterPlus! enclosure. Figure 3 presents a sketch of the available connectors. PulseBlasterPlus! Models 16-x can be equipped with a DB-25 output connector.

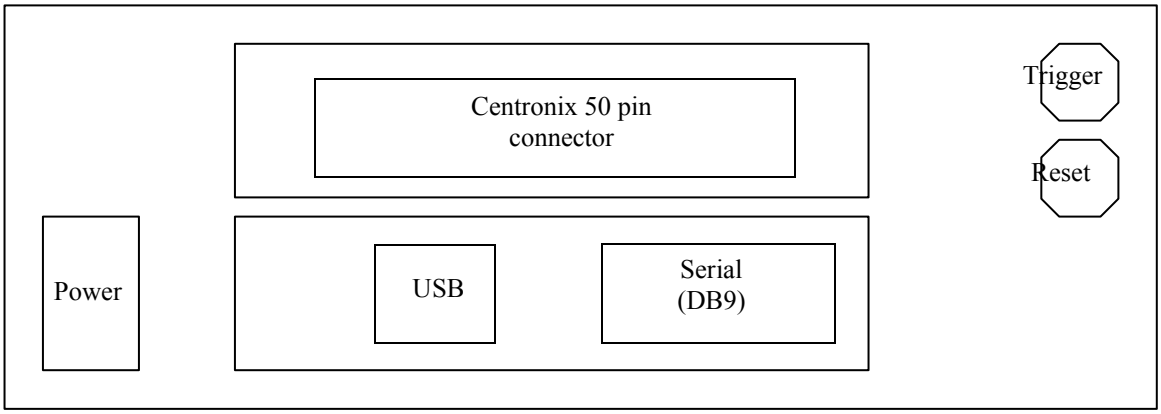


**Figure 3 – Back view of PulseBlasterPlus!™**

- Centronix 50 pin connector – TTL Output signals (a DB-25 connector may be used for PBP 16x models)
- Serial (DB9) – unused when a USB model is specified
- USB – USB Connection to host computer (unused when an RS-232 model is specified)
- Reset – TTL input signal to stop output
- Trigger – TTL input signal to start output

# Output Bit Assignments for Centronix 50 pin connector

| By Bit | |
|---|---|
| Bit | Pin |
| | |
| 0 | 38 |
| 1 | 14 |
| 2 | 15 |
| 3 | 40 |
| 4 | 41 |
| 5 | 17 |
| 6 | 18 |
| 7 | 43 |
| 8 | 44 |
| 9 | 20 |
| 10 | 21 |
| 11 | 46 |
| 12 | 47 |
| 13 | 23 |
| 14 | 24 |
| 15 | 49 |
| 16 | 35 |
| 17 | 34 |
| 18 | 33 |
| 19 | 32 |
| 20 | 31 |
| 21 | 30 |
| 22 | 29 |
| 23 | 28 |
| Clock | 50 |

| By Pin | | | |
|---|---|---|---|
| Pin | Bit | Pin | Bit |
| | | | |
| 1 | NC | 26 | GND |
| 2 | NC | 27 | GND |
| 3 | NC | 28 | 23 |
| 4 | NC | 29 | 22 |
| 5 | NC | 30 | 21 |
| 6 | GND | 31 | 20 |
| 7 | GND | 32 | 19 |
| 8 | GND | 33 | 18 |
| 9 | GND | 34 | 17 |
| 10 | GND | 35 | 16 |
| 11 | GND | 36 | RESERVED |
| 12 | GND | 37 | RESERVED |
| 13 | GND | 38 | 0 |
| 14 | 1 | 39 | GND |
| 15 | 2 | 40 | 3 |
| 16 | GND | 41 | 4 |
| 17 | 5 | 42 | GND |
| 18 | 6 | 43 | 7 |
| 19 | GND | 44 | 8 |
| 20 | 9 | 45 | GND |
| 21 | 10 | 46 | 11 |
| 22 | GND | 47 | 12 |
| 23 | 13 | 48 | GND |
| 24 | 14 | 49 | 15 |
| 25 | GND | 50 | Clock |

**Table 3:** Bit Definitions

# Additional Output Signals

The internal **Output Clock** signal is available to the outside world, as it is tied to pin 50 of the output connector**.** This clock is **not** a 50% duty cycle clock. The width of the high part of the signal is one system clock period. For a 100 MHz model, the signal is at a logical one for 10 ns at every transition to a new pulse program line.

**The following are internal signals only (they can be routed outside of box upon request).**

**System Reset –** is used to indicate (when low) to the external world that the uPC controller is in a reset state. It can be used in larger systems to monitor the state of the PulseBlasterPlus!™ design.

A signal that is similar to System Reset is the **Running** signal**.** It is driven high when the uPC is executing code. It is taken low when the uPC has entered either a reset of idle state.

The **Stopped** signal is the last signal used to indicate the state of the uPC. Stopped is asserted when the uPC has encountered the stop command while normally executing code. This signal informs the external world that the uPC has successfully executed its program and has halted operation.

# Appendix I: Sample C program

## Example Program

```
/*

Example 2 for PBPLUS

Shows how to use loops and subroutines

Last updated 3/22/05

*/

#include "pbdfuncs.h"
#include <stdio.h>

int main()
{

//set clock
set_clock(50*MHz);

//init device
pb_init();

//start programming device
start_programming(PULSE_PROGRAM);
int start, loop, sub; // define instruction labels

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

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

// Instruction 1 - Beginning of Loop (Loop 3 times).  Continue to next
instruction in 1s
loop =      pb_inst(0x0,LOOP,3,1 * s);

// Instruction 2 - End of Loop.  Return to beginning of loop or continue to
next instruction in .5 s
pb_inst(0x0,END_LOOP,loop,50 * ms);

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

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

```
// Instruction 5 - Continue to next instruction in 1 * s
pb_inst(0x0,CONTINUE,0,1*s);

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

//stop programming
stop_programming();

//start program
start_pb();

//close device
pb_close();

return 0;

}//end main()
```

# Appendix II: Programming the PBPlus Using Direct Outputs

If you do not wish to use the provided C functions, you can take advantage of the PBPLUS_outp() function in your own applications. An explanation of using this function to program the system is included below.

## Using DLL Functions to Send Instructions

The provided driver DLL, PBPlus.dll, provides three functions necessary for programming the PulseBlasterPlus:

**`int PBPlus_Init();`**

This function locates and initializes the PulseBlasterPlus system. It returns a 0 upon successful completion, or a negative number for an error.

**`int PBPlus_outp(unsigned short address, int data);`**

This function sends 8 bits of data to the PulseBlasterPlus, at a specified address offset. (Only the lowest 8 bits of the integer value are sent)

**`int PBPlus_Close();`**

This function releases control of the PulseBlasterPlus. It returns a 0 upon successful completion, or a negative number for an error.

### *Building Instructions Using the DLL Functions*

- To send instructions to the PulseBlasterPlus, the programmer must first call the **`PBPLUS_Init()`** function to locate and initialize the PulseBlasterPlus for sending

instructions

- Instructions are then build using the **`PBPLUS_outp(unsigned short address, int data)`** function, 8 bits at a time. For example, 10 **`PBPLUS_outp`** function calls are made to construct one 80-bit instruction in accordance with the Instruction Set Architecture

- Finally, the **`PBPLUS_Close()`** instruction is called to signal an end to programming, and to release control of the PulseBlasterPlus.

# Programming Information

Initialization of the PulseBlasterPlus for operation involves a minimum of four steps. The steps are as follows:

1) Send **LOAD NUMBER OF BYTES PER WORD** instruction.
2) Send **SELECT PERIPHERAL DEVICE** instruction.
3) Send **CLEAR ADDRESS COUNTER** instruction.
4) Loading data to memory.*
5) Send **PROGRAMMING FINISHED** instruction.

* Once the system has been programmed, this step is optional. If you are just restarting the pulse program after a STOP command or an HARDWARE_RESET has stopped the program, you can skip this step during re-initialization.

If these five commands are not sent from a PC, the PulseBlasterPlus will not run as desired. All five instructions are required as an attempt to ensure that the device has been programmed before it can be armed. The first time the system is used, the loading of the memory with data has to be performed. Upon reset, four instructions must be executed to restart the device again.

**WE for Peripherals:** This register is used to select the peripheral that is to be programmed. The value of this register that is used to select program memory is always zero and this is the default value for the register. A complete listing of the values and the associated hardware that can be programmed when appropriately set is listed below in Table 4. By selecting 0xFF for this value, no device is selected and the initial value of the output flags may be set. (*Note that the clock signal to the FF must still be transitioned, which may be accomplished by writing to the base output port + 5 twice in succession.)

| WE Register | Value (hex) |
|---|---|
| Program Memory | 0 |
| Change Flags Only | FF |

**Table 4:** Peripheral List

**CLEAR ADDRESS COUNTER:** The Address Counter is used to manufacture the memory address. The Address Counter is not loadable; it can only be cleared and started at zero. It is not possible to load a particular section of memory. All loads must start from either the beginning of memory, or wherever the Address Counter left off.

**Flag Initialization Strobe:** The output flags of the PulseBlasterPlus can be programmed while the device is in a reset state. This is useful to initialize flags after powering-up and to reset flags to a known state if a program must be aborted. Writing to the Flag Initialization Strobe register will toggle the line used to clock data into the output latches.

**LOAD_MEMORY:** This instruction is used to specify data that should be used to program the memory used by the device. Since the incoming data is taken only one byte at a time, the IBC must reconstruct the data word to be programmed. The data word is reconstructed in the IBC most significant byte first.

**PROGRAMMING FINISHED:** This instruction enables the pattern generator of the PulseBlasterPlus. This instruction prevents the pattern generator from accepting a hardware trigger or software start command before the device has been programmed. Once the design has been programmed, the **PROGRAMMING FINISIHED** command must be sent to arm the device for operation. After the pattern generator has been armed, any hardware trigger or software start command will cause the system to start operation. The PulseBlasterPlus can be reset by issuing the **DEVICE_RESET** command. This will internally clear the **PROGRAMMING FINISHED** instruction and prevent the pattern generator from operating again until the IBC has been re-initialized.

In order to select each of the commands mentioned about, you write to the port base + offset. A table of each offset's meaning is included below.

| Offset | Command | Function |
|---|---|---|
| 0 | DEVICE_RESET | Stops Pulse Program |
| 1 | DEVICE_START | Starts Pulse Program (only when in initialized state) |
| 2 | SELECT_BPW | Selects number of BPW (10 for instructions) |
| 3 | SELECT_DEVICE | Selects internal device to be programmed |
| 4 | CLEAR_ADDRESS_COUNTER | Resets internal memory address counter |
| 5 | FLAG_STROBE | Strobes output clock signal to preset digital outputs |
| 6 | DATA_TRANSFER | Data to be written to internal device |
| 7 | PROGRAMMING_FINISHED | Sets device in initialized state |

**Table 5:** Port Offset Command Functions

# Example Program

The following is an example of the output sequence to program the PulseBlasterPlus. You must use the PBPLUS_outp(addr, data) function from the provided DLL in order to use this method.

```
//      Initialization:

PBPLUS_outp(0,0);                    //(Issue device reset)
PBPLUS_outp(2,4);                    //(Select number of bytes per word)
PBPLUS_outp(3,0xFF);                 //(Select device to program (Flag initial values))
PBPLUS_outp(4,0);                    //(Reset address counter)

//      Set initial flag values
//      Values for this example are "0x000000f0"

PBPLUS_outp(6,0);                    //(Data transfer)
PBPLUS_outp(6,0);                    //(Data transfer)
PBPLUS_outp(6,0);                    //(Data transfer)
PBPLUS_outp(6,0xF0);                 //(Data transfer)

PBPLUS_outp(5,0);                    //(Clock data into external buffer)
PBPLUS_outp(5,0);                    //(Return clock signal to low)

//      Pulse Program Setup

PBPLUS_outp(0,0);                    //(Issue Device Reset)
PBPLUS_outp(2,10);                   //(Select number of bytes per word)
PBPLUS_outp(3,0);                    //(Select device to program (RAM))
```

```
PBPLUS_outp(4,0);                    //(Reset address counter)

//Send pulse program
PBPLUS_outp(6,0x18);          //(Byte 9 of first instruction)
PBPLUS_outp(6,0xFF);          //(Byte 8 of first instruction)
PBPLUS_outp(6,0xFF);          //(Byte 7 of first instruction)
PBPLUS_outp(6,0x00);          //(Byte 6 of first instruction)
PBPLUS_outp(6,0x00);          //(Byte 5 of first instruction)
PBPLUS_outp(6,0x00);          //(Byte 4 of first instruction)
PBPLUS_outp(6,0x00);          //(Byte 3 of first instruction)
PBPLUS_outp(6,0x00);          //(Byte 2 of first instruction)
PBPLUS_outp(6,0x00);          //(Byte 1 of first instruction)
PBPLUS_outp(6,0x07);          //(Byte 0 of first instruction)

PBPLUS_outp(6,0xXX);          //(Byte 9 of second instruction)

/*
    Continue this process for all instructions.  The
    explanation of how to create the 80 bit instruction
    words is explained in the "Programming the PulseBlasterPlus
    secion of this manual.  When finished with all
    instructions, continue with the sequence below.
*/

PBPLUS_outp(7,0);                    //(Programming Finished)


/*
    Only execute the following command when you are ready
    for the program to start running.
*/

    PBPLUS_outp(1,0);                //(Start pulse program)
```

# Contact Information

**Address:** SpinCore Technologies, Inc.
Gainesville, FL 32653, USA

**Telephone:** +1-352-271-7383

**Email:** Use Product Inquiry Form at www.spincore.com

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