SpinCore Technologies, Inc.

Owner's Manual for the

PulseBlasterPlus!™

Complete Digital Excitation System:
Direct Digital Synthesis (DDS) and TTL

Models:

PBP-DDS-100-USB	**PBP-DDS-100-RS232**
PBP-DDS-50-USB	**PBP-DDS-50-RS232**

SpinCore Technologies, Inc.
3525 NW 67th Avenue
Gainesville, Florida 32653, USA
Phone: (352)-271-7383

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, so if you ever need assistance please contact us and we will strive to provide the necessary help.

Contents

Section I: Introduction

1. Quick Product Overview

The PulseBlasterPlus!™ series of Intelligent Pattern and Waveform Generation systems from SpinCore Technologies, Inc., couples SpinCore's unique Intelligent Pattern Generation processor core, dubbed PulseBlaster™, with Direct Digital Synthesis (DDS) for use in system control and waveform generation.

The PulseBlaster's™ state-of-the-art timing processor core, implemented in programmable logic, provides all the necessary timing control signals required for overall system control and waveform synchronization. By adding DDS and arbitrary waveform generation features, PulseBlasterPlus™ can provide not only digital (TTL) but also analog output signals, meeting high-performance and high-precision complex excitation/stimuli needs of demanding users.

PulseBlasterPlus!™ provides users the ability to control their systems through the generation of fully synchronized (digital and analog) excitation waveforms from a stand alone system, providing users a compelling price/performance proposition unmatched by any other device on the market today. Figure 1 presents sample capabilities of the system.

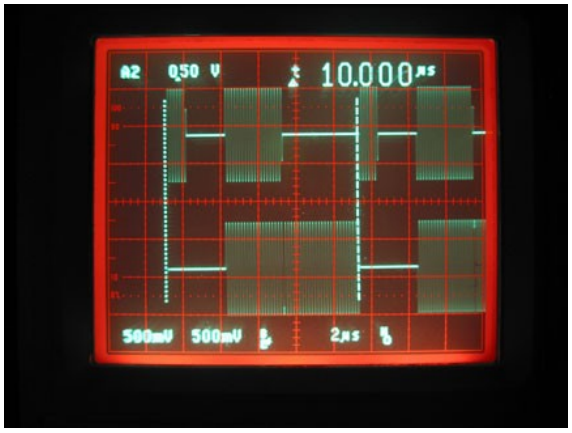

Figure 1. Sample PulseBlasterPlus™ output capabilities

2. System Architecture

Block diagram

Figure 2 presents the general architecture of the PulseBlasterPlus™ system. The DDS, Pulse Timing and Gating cores have been integrated into a single stand alone box. The internal DDS core has 16 programmable 32 bit frequency registers that are under the pulse program control. Prior to gating, the internal DDS signal can be phase offset by one of 16 programmable 12 bit phase registers on each output. The RF outputs are available on either BNC or SMA connectors. The host interface can be either RS-232 or USB.

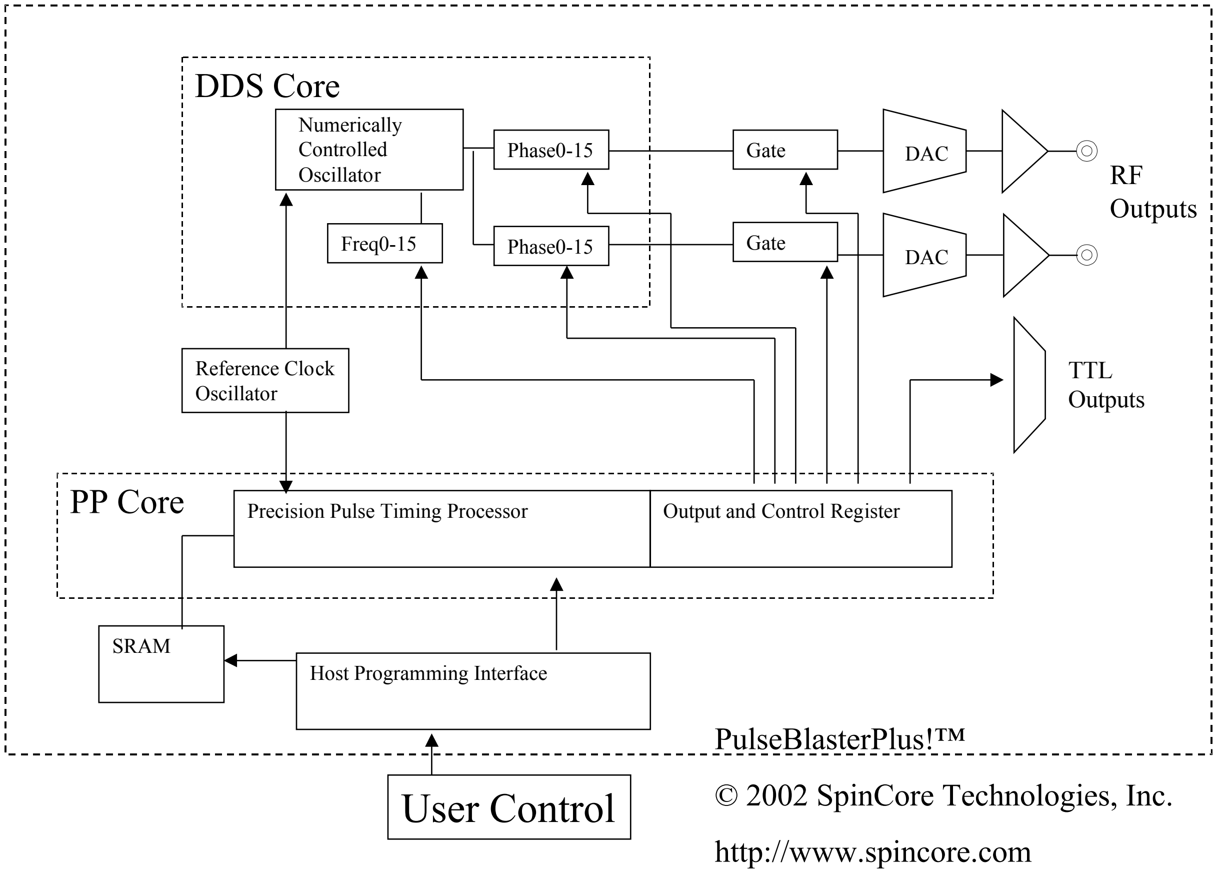
DDS Core
Numerically Controlled Oscillator
Phase0-15
Freq0-15
Phase0-15
Gate
Gate
DAC
DAC
RF Outputs
TTL Outputs
Reference Clock Oscillator
PP Core
Precision Pulse Timing Processor
Output and Control Register
SRAM
Host Programming Interface
PulseBlasterPlus!™
User Control
© 2002 SpinCore Technologies, Inc.
http://www.spincore.com

Fig. 2. PulseBlasterPlus™ system architecture.

Output signals

This product comes with up to two analog output channels that can be configured to output radio-frequency (RF/IF) waveforms, arbitrary waveforms, or a combination of both, and up to 10 digital output signal lines. The frequency and phase of the RF waveforms generated by the DDS output channels are under the complete control of the user and are specified through software programming.

PulseBlasterPlus!™ also provides the ability to gate the output of the DDS channels allowing for independent pulsed RF operation of all output channels. With digital sampling rate of 100 MHz (max. clock frequency), analog signals up to approx. 50 MHz can be generated. Both analog output signals are available on external BNC connectors. The output impedance of the analog signals is 50 ohms.

The individually controlled digital (TTL/CMOS) output bits are capable of delivering +-25 mA per bit. For non-rf models, twenty-four of the output lines can be set to either the 5 V or 3.3V I/O TTL logic standard. For DDS models, up to 10 digital TTL output lines are available on the 15-pin connector.

Timing characteristics

PulseBlasterPlus’s™ timing controller uses an internal (on-board) 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!’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

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

PulseBlasterPlus!™ is a versatile, high-performance pulse/pattern TTL and RF/IF/AWG 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. Two analog channels (50 ohm impedance) are available from a single system.

Summary Specifications

DDS Section (Model DDS-100):

- 100 MHz reference clock oscillator (other frequencies upon request)
- 0.047 Hz frequency resolution (32 bits)
- 16 loadable frequency registers for agile frequency modulation/selection (32 bits each)
- 16 loadable phase-offset registers for agile phase modulation/selection (12 bits each)
- 0.09 degree phase resolution (12 bits)
- 40 ns phase/frequency switching latency
- 5 MHz max. modulating frequency
- 12 dBm rf output power
- 50 ohm output impedance
- BNC connectors

TTL Section:

- 10 individually controlled digital output lines (TTL levels)
- variable pulses/delays for every TTL line
- 25 mA output current per TTL line
- output lines can be combined to increase the max. output current

Common Parameters (DDS and TTL Sections):

- 90 ns shortest pulse/interval (50 ns for non-rf models at 100 MHz clock frequency)
- 2 years longest pulse/interval
- 10 ns pulse/interval resolution
- rf and TTL pulses are synchronized
- 32k max. memory space
- external triggering and reset – TTL levels

Pulse Program Control Flow (Common):

- loops, nested 8 levels deep
- 20 bit loop counters (max. 1,048,576 repetitions)
- subroutines, nested 8 levels deep
- wait for trigger - 80 ns latency
- 5 MHz max. retriggering frequency

Section II: Installation Guide

1. Go to http://www.spincore.com/CD/PulseBlasterPlus/v1/Drivers/ and download PBPlus_Drivers.zip.

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).

4. When the USB device is first connected to the host, a dialog box will appear that says "New Hardware Found." Click "Next"

5. Place a check by the box that says "Specify Location" and uncheck all others. Click "Next"

6. A window will pop up asking for the driver directory. Click "Browse"

7. Browse to the newly created folder containing the drivers. There will be 3 subdirectories – win-98-me, win-2k, and win-xp. Select the appropriate directory.

8. The next dialog box should say that an appropriate driver was found – "SpinCore Technologies, Inc. using ARS Technologies USB to ISA device."

9. Click "Next" and "Finish" until installation is complete.

10. To test if the drivers are working properly, go to http://www.spincore.com/CD/PulseBlasterPlus/v1/DDS_Examples/ and download the following files:

    Example1_DDS.exe, Example2_DDS.exe, pb01us02.dll
    *pb01us02.dll must be in the same working directory as the executable files.

11. Run Example1_DDS.exe. All digital output pins on the PulseBlasterPlus! should toggle on and off at about 1 second intervals. Both analog output signals should be generating a 0.1 MHz sine wave, with the two signals being 90 degrees out of phase.

12. Run Example2_DDS.exe. All output pins should pulse on for 2us, off for 2us, on for 2us, then off for 14us. The digital outputs should alternate being on and off, while outputting a 1 MHz sine wave.

Section III: Physical Description

1. Connector Information

All input-output connectors have been mounted at the back of the PulseBlasterPlus! enclosure. Figure 2 presents a sketch of the available connectors.

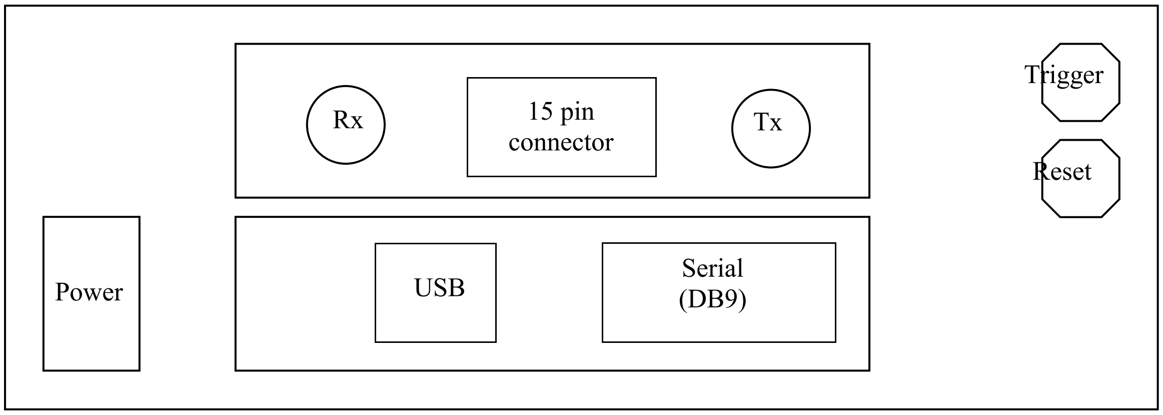

Figure 3 - Back view of PulseBlasterPlus!™

- Rx (BNC) – Analog output with 50 ohm impedance
- 15 pin connector – A high density 15 pin connector for TTL Output signals
- Tx (BNC) – Analog output with 50 ohm impedance
- Serial (DB9) – RS232 connection to host computer (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

2. Output Bit Assignments for high density 15 pin connector

Pin Assignments	
Pin#	Bit#
1	0
2	2
3	4
4	6
5	8
6	GND
7	GND
8	GND
9	GND
10	GND
11	1
12	3
13	5
14	7
15	9

Table 1: Connector Pin Assignments

3. Additional Output Signals

The following signals are available internally, and can be routed externally upon request:

Output Clock - It is the clock signal used to latch patterns in the output buffers. This clock has been configured to have a relatively slow slew rate so as to avoid noise problems on a transmission line. 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 system, the signal is at a logical one for 10 ns at every transition to a new pulse program line.

System Reset – 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.

Running - 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.

Stopped - 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.

Section IV: Programming Information

1. Provided DLL Functions

The provided driver DLL, pb01us02.dll, provides a function necessary for programming the PulseBlasterDDS.

`int pbp_outp(unsigned short address, int data);`

This function sends 8 bits of data over the USB, at a specified address offset. Only the lowest 8 bits of are valid.

2. Using C Functions

A series of functions have been written to make writing pulse programs easier. These functions use the provided DLL functions. In order to use these functions, the DLL (pb01us02.dll), the library file (pb01us02.lib), the header files (pb01us02.h and pbdfuncs.h), and source file (pbdfuncs.c) must be in the working directory of your C compiler. You can obtain these files from http://www.spincore.com/CD/PulseBlasterPlus/v1/DDS_Examples/Example1_DDS_Source.zip

`void set_clock(double clock_freq);`

Used to set the clock frequency of the system. The variable clock_frequency is specified in MHz. The default value is 50MHz. You only need to call this function if you are not using a -50 system.

`void start_programming(int device);`

Used to initialize the system to receive programming information. It accepts a parameter to know which registers are being programmed. Valid values for device are PULSE_PROGRAM, FREQ_REGS, and PHASE_REGS.

`void set_freq(double freq);`

Used to set the values in the frequency registers. Can only be called after start_programming(FREQ_REGS) has been called. Registers are programmed one at a time, starting at 0 and incrementing each time this function is called. It accepts the value for the frequency register in MHz

`void set_phase(double phase);`

Used to set the values in the phase registers. Can only be called after start_programming(PHASE_REGS) has been called. Registers are programmed one at a time, starting at 0 and incrementing each time this function is called. It accepts the value for the phase register in degrees.

**int pb_inst(int freq, int phase, int RF_Output_Enable, int flags,
int inst, int inst_data, int length);**

Used to send one instruction of the pulse program. Can only be called after start_programming(PULSE_PROGRAM) has been called. It returns –1 on an error, or the instruction number upon success. Instructions are numbered starting at 0.

int freq – Number of the frequency register to be used. Valid range is from 0 to 16.

int phase – Number of the phase register to be used. Valid range is from 0 to 16.

int RF_Output_Enable – Determines whether analog output is generating a sinusoid or is at ground. Valid values are ANALOG_ON and ANALOG_OFF
int flags – determines state of each TTL output bit. Valid values are 0x0 to 0x3FF. For example, 0x010 would correspond to bit 7 being on, and all other bits being off.

int inst – determines which type of instruction is to be executed. Valid instructions are included in table 2.

int inst_data – data to be used with the previous inst field. The meaning of this is described in table 2.

int length – duration of this pulse program instruction, specified in ns.

void stop_programming();

Used to tell that programming the system is complete. Program execution cannot start until this command is received.

void start_pb();

Once system has been programmed, this instruction will start execution of pulse program.

void 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.

Example use of C functions

The following example is a complete pulse program that can be executed on the PulseBlasterPlus! systems. The program, upon executing, will load and start generating a sequence of 10 us RF pulses on Tx channel every 1 ms. The Rx channel will be turned ON during the 1 ms gap between the Tx pulses and OFF during the Tx pulse. The phase of the Rx signal will be phase shifted by 180 degrees with respect to the Rx phase. The generated rf frequencies will be 1.012 MHz. While executing, the TTL outputs will toggle between the logical “one” during the 10 us intervals and logical “zero” during the 1 ms intervals.

```
//Sample Program for PulseBlasterPlus!

#include "pbdfuncs.h"
#include “pbdfuncs.cpp”

int main()
{
   int start;
   set_clock(100); // 100 MHz reference clock frequency

   start_programming(FREQ_REGS);
   set_freq(1.012); // 1.012 MHz in 0th frequency register

   start_programming(TX_PHASE_REGS);
   set_phase(0); // Tx phase value set to 0 degrees (in 0th phase register)

   start_programming(RX_PHASE_REGS);
   set_phase(180); // Rx set to 180 degrees out of phase

   start_programming(PULSE_PROGRAM);

   // Instruction 0 - Continue to next instruction in 10.0 us
   start = pb_inst(0,0,TX_ANALOG_ON,1,RX_ANALOG_ON,0x3FF,CONTINUE,0,10.0*us);

   // Instruction 1 - Branch to "start" (Instruction 0) in 1 ms
   pb_inst(0,0,TX_ANALOG_OFF,1,RX_ANALOG_ON,0x0,BRANCH,start,1*ms);

   stop_programming();

   start_pb(); //the pulse program  will start

   return 0;
}
```

A more complex pulse program example is provided in Appendix I.

inst	inst_data	Meaning
CONTINUE	Ignored	Program execution continues to next instruction
STOP	Ignored	Stop execution of program (*Note all TTL values remain from previous instruction, and analog outputs turn off)
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
END_LOOP	Address of beginning of loop	Specify end of a loop. Execution returns to begging of loop and decrements loop counter.
JSR	Address of first subroutine instruction	Program execution jumps to beginning of a subroutine
RTS	Ignored	Program execution returns to instruction after JSR was called
BRANCH	Address of next instruction	Program execution continues at specified instruction
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
WAIT	Ignored	Program execution stops and waits for software or hardware trigger. Execution continues to next instruction after receipt of trigger.

Table 2: Instruction Types and meaning of associated data.

3. Using Direct Outputs

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

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. 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
Program Frequency Registers	1
Program Phase Registers	2
Change Flags Only	FF

Table 3: 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. Appendix I provides more information on how to use the Flag Initialization Strobe to program the output flags while the PulseBlasterPlus! is in a reset state.

LOAD_MEMORY: This instruction is used to specify data that should be used to program the memory used by the device. Since the ISA 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	Meaning
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	Using Table 2, 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 4: Port Offset Meaning

Calculating DDS Frequency

$$\text{DDS frequency word (32 bits)} = \frac{\text{Desired Freq (in MHz)} * 2^{32}}{\text{Clock Freq (in MHz)}}$$

The calculated value above can be used when programming the DDS units. The frequency resolution of the default design is:

$50e6 / 2^{32} = 0.012$ Hz

Calculating DDS Phase

$$\text{DDS phase word (32 bits)} = \frac{\text{Desired Phase (in degrees)} * 2^{32}}{\text{360 degrees}}$$

The calculated value above can be used when programming the DDS units. Only the top 12 bits are used. This makes the phase resolution of the default design is:

$360 / 2^{12} \approx 0.09$ degrees

Example Output:

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

The following is an example of the output sequence to program the PulseBlasterPlus!™. Explanations are included in brackets in the middle of the code.

```
// Initialization:

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

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

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

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

// Set up DDS frequency registers

pbp_outp(0,0);       //(Issue device reset)
pbp_outp(2,4);       //(Select number of bytes per word)
pbp_outp(3,1);       //(Select device to program (DDS Frequency Registers))
pbp_outp(4,0);       //(Reset address counter)

/*
   DDS Register Values
   Reg0 = 051EB852 (1 MHz)
   Reg1 = 0A3D70A4 (2 MHz)
   .
   .
   Reg15 = 147AE148 (4 MHz)

   Formula for finding these values:

   REG0 = DESIRED_FREQUENCY * 2^32 / PBDDS_CLOCK
      = 1 MHz * 2^32 / 50 MHz = 858993459.2 = 0x051EB852
*/

pbp_outp(6,0x05);    //(Data Transfer - Byte 3 of Reg0)
pbp_outp(6,0x1E);    //(Data Transfer - Byte 2 of Reg0)
pbp_outp(6,0xB8);    //(Data Transfer - Byte 1 of Reg0)
pbp_outp(6,0x52);    //(Data Transfer - Byte 0 of Reg0)

pbp_outp(6,0x05);    //(Data Transfer - Byte 3 of Reg1)
pbp_outp(6,0x3D);    //(Data Transfer – Byte 2 of Reg1)
pbp_outp(6,0x70);    //(Data Transfer - Byte 1 of Reg1)
pbp_outp(6,0xA4);    //(Data Transfer - Byte 0 of Reg1)

.
.

pbp_outp(6,0x14);    //(Data Transfer - Byte 3 of Reg15)
pbp_outp(6,0x7A);    //(Data Transfer - Byte 2 of Reg15)
pbp_outp(6,0xE1);    //(Data Transfer - Byte 1 of Reg15)
pbp_outp(6,0x48);    //(Data Transfer - Byte 0 of Reg15)

// Set up phase registers

pbp_outp(0x0);       //(Issue device reset)
pbp_outp(2,4);       //(Select number of bytes per word)
pbp_outp(3,2);       //(Select device to program (TX Phase Registers))
pbp_outp(4,0);       //(Reset address counter)

/*
   Phase Register Values
   Reg0 = 093E93E9 (13 degrees)
   Reg1 = 40000000 (90 degrees)
   .
   .
   Reg15 = E0000000(315 degrees)

   Formula for finding these values:

   REG0 = DESIRED_PHASE * 2^32 / 360
      = 13 * 2^32 / 360 = 155096041.24 ≈ 0x093E93E9

   * Note that only the top 3 hex digits (0x093) are actually used in
programming the phase register.  The other digits must still be sent so that
the dds word remains 32 bits long.

*/

pbp_outp(6,0x09);    //(Data Transfer - Byte 3 of Reg0)
pbp_outp(6,0x3E);    //(Data Transfer - Byte 2 of Reg0)
pbp_outp(6,0x93);    //(Data Transfer - Byte 1 of Reg0)
pbp_outp(6,0xE9);    //(Data Transfer - Byte 0 of Reg0)

pbp_outp(6,0x40);    //(Data Transfer - Byte 3 of Reg1)
pbp_outp(6,0x00);    //(Data Transfer - Byte 2 of Reg1)
pbp_outp(6,0x00);    //(Data Transfer - Byte 1 of Reg1)
pbp_outp(6,0x00);    //(Data Transfer - Byte 0 of Reg1)

.
.

pbp_outp(6,0xE0);    //(Data Transfer - Byte 3 of Reg15)
pbp_outp(6,0x00);    //(Data Transfer - Byte 2 of Reg15)
pbp_outp(6,0x00);    //(Data Transfer - Byte 1 of Reg15)
pbp_outp(6,0x00);    //(Data Transfer - Byte 0 of Reg15)

// Pulse Program Setup

pbp_outp(0,0);       //(Issue Device Reset)
pbp_outp(2,10);      //(Select number of bytes per word)
pbp_outp(3,0);       //(Select device to program (RAM))
pbp_outp(4,0);       //(Reset address counter)


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

pbp_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 included below.  When finished with all
   instructions, continue with the sequence below.
*/

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


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

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

Breakdown of 80 bit instruction word

Instruction Bits 79...0 are broken up into 4 sections

2. Output Pattern - 24 bits (Instruction Bits 79..56)
3. Data Field - 20 bits (Instruction Bits 55..36)
4. OP Code - 4 bits -(Instruction Bits 35..32)
5. Delay Count - 32 bits - (Instruction Bits 31..0)

Output Pattern (Bits 79 – 56):

Instruction Bit #	Function
79	Selects one of 16 Frequency Registers (bit 3)
78	Selects one of 16 Frequency Registers (bit 2)
77	Selects one of 16 Frequency Registers (bit 1)
76	Selects one of 16 Frequency Registers (bit 0)
75	Selects one of 16 TX Phase Registers (bit 3)
74	Selects one of 16 TX Phase Registers (bit 2)
73	Selects one of 16 TX Phase Registers (bit 1)
72	Selects one of 16 TX Phase Registers (bit 0)
71	Selects one of 16 RX Phase Registers (bit 3)
70	Selects one of 16 RX Phase Registers (bit 2)
69	Selects one of 16 RX Phase Registers (bit 1)
68	Selects one of 16 RX Phase Registers (bit 0)
67	Enables output of TX signal (0 = on, 1 = off)
66	Enables output of RX signal (0 = on, 1 = off)
65	Output Connector pin 15
64	Output Connector pin 5
63	Output Connector pin 14
62	Output Connector pin 4
61	Output Connector pin 13
60	Output Connector pin 3
59	Output Connector pin 12
58	Output Connector pin 2
57	Output Connector pin 11
56	Output Connector pin 1

Table 5: Output Pattern Bit Definitions

Bits 79-76 select the frequency register used by the output signals. Bits 75..72 select the phase offset register used by the TX output. Bits 71..68 select the phase offset register used by the RX output.

Data Field (Bits 55 – 36) and Op Code (Bits 35 – 32):
The data field's function is dependent on the OpCode.

OpCode #	OpCode Meaning	Data Field used for
0	Continue	Ignored
1	Stop	Ignored
2	Loop	Number of desired Loops - 1
3	End Loop	Address of Instruction originating loop
4	Jump to Subroutine	Address of first subroutine instruction
5	Return From Subroutine	Ignored
6	Branch	Address of next instruction
7	Long Delay	Number of desired loops - 2
8	Wait	Ignored

Table 6: Opcode definitions

Delay Count (Bits 31 – 0):
How long the current instruction should be executed. The smallest possible delay is 6 clock cycles (120ns). The formula for determining its value is below (Assumed with a 50MHz clock)

DELAY_VALUE = (Desired Delay (ns) – 60 ns) / 20 ns

Ex. for delay of 1000 ns

```
         = (1000ns – 60ns) / 20ns = 940ns / 20ns
         = 47
            = 0x2F
```

Appendix I: Sample C program

```
// SpinCore Technologies, Inc.
// August 2002
// http://www.spincore.com

#include "pbdfuncs.h"
#include “pbdfuncs.cpp”

int main()
{
      // Check for proper initialization of PulseBlasterDDS
      if (InitPMster()!=0)
      {
         printf("Error initializing\n");
         return -1;
      }

      // Set clock frequency
      set_clock(50);

      // Start programming the frequency registers
      start_programming(FREQ_REGS);

      // Program the registers in order from 0 to 15
      // Valid units are MHz, kHz, and Hz - default is MHz

      set_freq(1*MHz); // Set register 0
      set_freq(2*MHz); // Set register 1
      set_freq(3*MHz); // Set register 2
      set_freq(4*MHz); // Set register 3
      set_freq(5*MHz); // Set register 4
      set_freq(6*MHz); // Set register 5
      set_freq(7*MHz); // Set register 6
      set_freq(8*MHz); // Set register 7
      set_freq(9*MHz); // Set register 8
      set_freq(10*MHz); // Set register 9
      set_freq(11*MHz); // Set register 10
      set_freq(12*MHz); // Set register 11
      set_freq(13*MHz); // Set register 12
      set_freq(14*MHz); // Set register 13
      set_freq(15*MHz); // Set register 14
      set_freq(16*MHz); // Set register 15

      // Start programming the phase registers
      start_programming(PHASE_REGS);

      // Program the registers in order from 0 to 15
      // Units are in degrees

      set_phase(0); // Set register 0
      set_phase(22.5); // Set register 1
      set_phase(45); // Set register 2
      set_phase(67.5); // Set register 3
      set_phase(90); // Set register 4
      set_phase(112.5); // Set register 5
      set_phase(135); // Set register 6
      set_phase(157.5); // Set register 7
      set_phase(180); // Set register 8
      set_phase(202.5); // Set register 9
      set_phase(225); // Set register 10
      set_phase(247.5); // Set register 11
      set_phase(270); // Set register 12
      set_phase(292.5); // Set register 13
      set_phase(315); // Set register 14
      set_phase(337.5); // Set register 15

      //Begin pulse program
      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 format
      //int pb_inst(int freq, int phase, int RF_Output_Enable, int flags, int
inst, int inst_data, int length)

      // Instruction 0 - Jump to Subroutine at Instruction 4 in 1us
      start =  pb_inst(0,0,ANALOG_OFF,0x3FF,JSR,sub,1*us);

      // Instruction 1 - Beginning of Loop (Loop 3 times).  Continue to next
instruction in 1us
      loop =   pb_inst(0,0,ANALOG_OFF,0x0,LOOP,3,1*us);
      // Instruction 2 - End of Loop.  Return to beginning of loop or continue
to next instruction in 1us
            pb_inst(0,0,ANALOG_ON,0x0,END_LOOP,loop,1*us);

      // Instruction 3 - Stay here for (5*1us) then continue to Instruction 4
            pb_inst(0,0,ANALOG_OFF,0x0,LONG_DELAY,5,1*us);

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

      // Instruction 5 - Continue to next instruction in 2us
            pb_inst(0,0,ANALOG_OFF,0x0,CONTINUE,0,2*us);
      // Instruction 6 - Return from Subroutine to Instruction 1 in 2us
            pb_inst(0,0,ANALOG_ON,0x0,RTS,0,2*us);

      // End of programming registers and pulse program
      stop_programming();

      // Start execution of the pulse program
      start_pb();

      // Release PusleBlasterDDS resources
      ClosePMster();
   return 0;
}
```

Appendix II: Available Custom Options

PulseBlasterPlus!™ can be customized to fit some specific needs. Below is a sample list of available options.

1. Custom master-clock frequency.
2. Temperature-controlled crystal oscillator.
3. Custom output connectors.
4. Reset/Running/Stopped status signals wired to an output connector.
5. Custom software.

To discuss your specific needs, please contact SpinCore Technologies, Inc., at sales@spincore.com.