PulseBlasterDDS™

Model DDS-III-100

(PCI Board SP3)

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 PulseBlasterDDS™ series of Intelligent Pattern and Waveform Generation boards 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 pulse generation.

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

PulseBlasterDDS provides users the ability to control their systems through the generation of fully synchronized (digital and analog) excitation pulses from a small form factor PC board, providing users a compelling price/performance proposition unmatched by any other device on the market today. Figure 1 presents sample capabilities of the board.

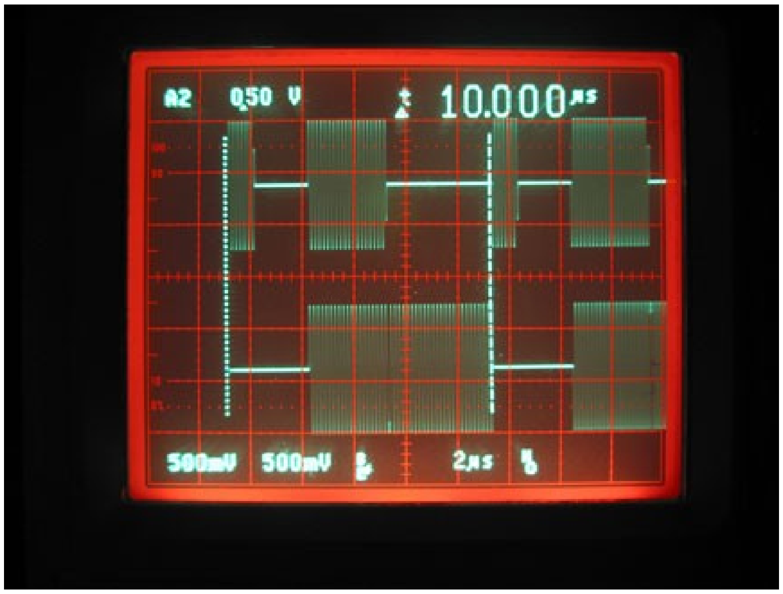

Figure 1: Sample PulseBlasterDDS output capabilities

Board Architecture

Block Diagram

Figure 2 presents the general architecture of the PulseBlasterDDS system. The two major building blocks are the DDS Core and the Pulse Programming and Timing Processor Core (PP Core). The DDS Core contains a numerically controlled oscillator and has 16 programmable frequency registers that are under the pulse program control. Prior to gating, the DDS signal can be phase offset by one of two sets of 16 programmable phase registers. The PP Core controls the timing of the gating pulses and provides the necessary control signals for frequency and phase registers. The DDS and PP cores have been integrated onto a single silicon chip. High performance DAC chips and high current output amplifiers complement the design. User control to the system is provided through the host-programming interface over the PCI bus.

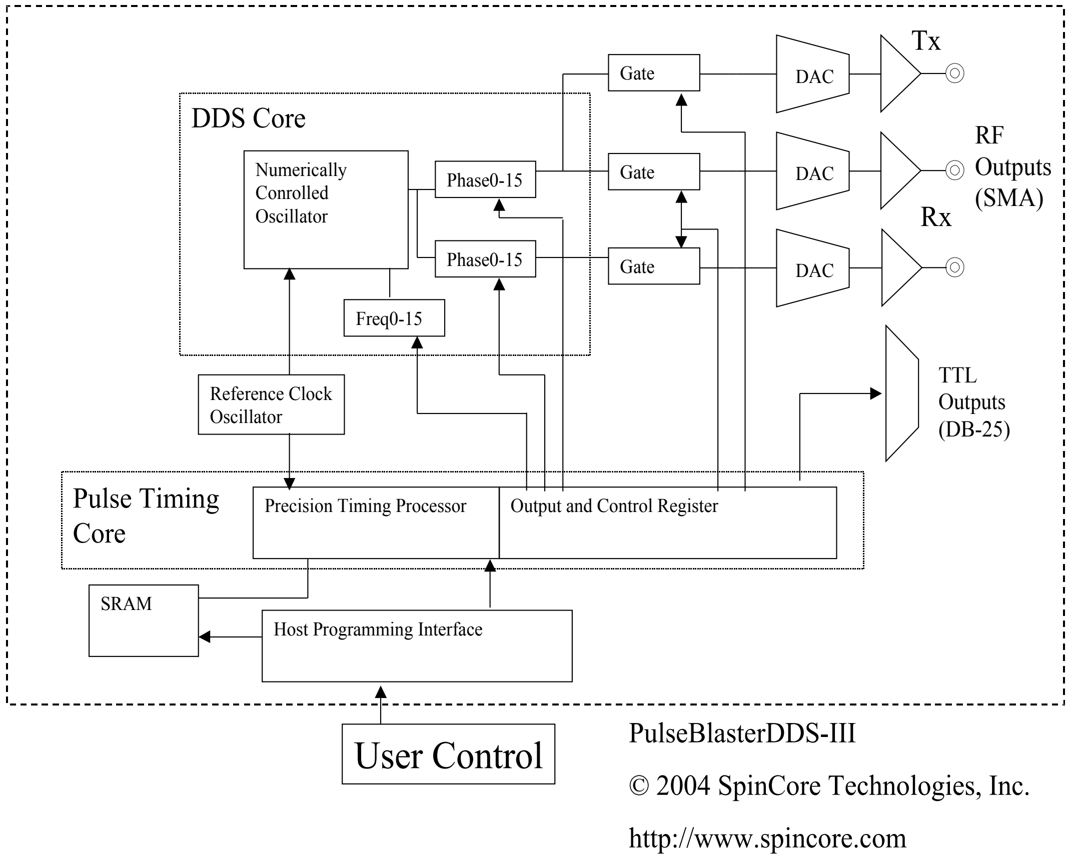
DDS Core
Numerically Controlled Oscillator
Phase0-15
Phase0-15
Freq0-15
Gate
Gate
Gate
DAC
DAC
DAC
Tx
RF Outputs (SMA)
Rx
TTL Outputs (DB-25)
Reference Clock Oscillator
Pulse Timing Core
Precision Timing Processor
Output and Control Register
SRAM
Host Programming Interface
User Control
PulseBlasterDDS-III
© 2004 SpinCore Technologies, Inc.
http://www.spincore.com

Figure 2: PulseBlasterDDS board architecture

Output signals

The PulseBlasterDDS comes with three analog output channels configured to output radio-frequency (RF/IF) pulses, and 10 digital output signal lines (one of the output lines has a dual use and functions as a phase reset for the DDS generator). The frequency and phase of the RF pulses generated by the DDS are under the control of the user and are specified through software programming. The phase of the numerically controlled oscillator can be reset on demand within the pulse program. PulseBlasterDDS provides the ability to gate the output of the DDS channels allowing for independent pulsed RF operation. With digital sampling rate of 100 MHz (max. reference clock frequency), the maximum theoretical output frequency is 50 MHz (the Nyquist Theorem). [1] The

[1] Note that the usefulness of a waveform with two samples per period is limited, and, depending on applications, practical considerations would often call for more than two samples per period.

analog output signal is available on an on-board SMA connector. The output impedance of the analog signal is 50-ohms. There are no interpolating filters on board.

The 10 individually controlled digital (TTL/CMOS) output bits are capable of delivering ±25 mA per bit and have an output voltage of 3.3V. These signals are available on the PC bracket-mounted DB-25 connector. Setting output bit 10 high via the output control word also resets the phase of the RF waveforms for phase coherent switching, and can be used to generate a constant voltage on the DACs.

Timing characteristics

PulseBlasterDDS's timing controller can accept either an internal (on-board) crystal oscillator or an external frequency source of up to 100 MHz. The innovative architecture of the timing controller allows the processing of either simple timing instructions (delays of up to 2^{32} = 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).

The timing controller has a very short minimum delay cycle – only nine clock periods. This translates to a 90 ns minimum pulse/delay/update with a 100 MHz clock.

Phase Coherent Switching

The board allows for phase continuous and/or phase coherent switching. In addition, the DDS can be reset to zero whenever a new RF pulse is started. Consult the explanation of the flags parameter to the pb_inst instruction on page 12 for implementing the phase reset.

Instruction set

PulseBlasterDDS' 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

PulseBlasterDDS 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).

Status Readback

The status of the program can be read in hardware or software. The hardware status output signals consist of five IDC connector pins labeled "Status". The same output can be read through software using C. See section IV (Connecting to the PulseBlaster Board, page 16) for more detail about the hardware lines and section III (Programming the PulseBlaster, page 11) for more detail about the C function pb_read_status(). *Note: Status readback via software is available in firmware versions 7-3 and newer; older versions will return meaningless data when pb_read_status( ) is called.*

Summary

PulseBlasterDDS is a versatile, high-performance pulse/pattern TTL and RF/IF generator operating at speeds of up to 100 MHz and capable of generating pulses/delays/intervals ranging from 90 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 with a voltage of 3.3 V. The output impedance of the analog channel is 50-ohms.

Specifications

DDS Specifications

- 100 MHz reference clock oscillator (other frequencies available upon request)
- 0.37 Hz frequency resolution (28 bits)
- 16 loadable frequency registers for agile frequency modulation/switching/selection (32 bits each)
- Two sets of 16 loadable phase-offset registers for agile phase modulation/switching/selection (12 bits each)
- 0.09° phase resolution (12 bits)
- 40 ns phase switching latency
- 40 ns frequency switching latency (phase continuous)
- phase coherent switching
- 10 dBm RF output power
- 50 ohm output impedance
- SMA connectors
- 30 MHz 3dB bandwidth
- RF Output capable of outputting DC at programmed output level (using phase offset)

TTL Specifications

- 10 individually controlled digital output lines (TTL levels; one of the output lines has a dual use and functions as a phase reset for the DDS generator)
- 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 Specifications)

- 90 ns shortest pulse/interval per instruction
- 2 years longest pulse/interval per instruction
- 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, adjustable to 2 years in duration
- Approximately 2 MHz max. re-triggering frequency (based on the latency of the WAIT opcode)

II. Installation

Installing the PulseBlasterDDS Driver

The PulseBlasterDDS uses the spinapi driver and control library. The latest version of this driver can be downloaded from http://www.spincore.com/CD/spinapi. This URL contains both the driver as well as a detailed installation guide. Please refer to this document for instructions on how to properly install the drivers.

Testing PulseBlasterDDS

The simplest way to test whether the PulseBlasterDDS has been installed properly and can be controlled as intended is to run a simple test program. Example test programs can be found at: http://www.spincore.com/CD/PulseBlasterDDS/PCI/SP3/ under the title “PBDDSexample_programs.zip”. The pbdds3_ex1 program that comes in the package will produce an 1MHz sine wave on both TX and RX channels. The TX Channel cycles between being on and off with a period of 8us. The TX and RX channels are set to have a 90 degree phase offset. A second test program that can be ran is pbdds3_ex2 program. This program makes use of all available instructions (except WAIT). On the TX channel there will be a single period of a 1MHz sine wave, followed by a 1us gap. This pattern will be repeated 2 more times followed by a long (5ms) gap, after which the pattern will repeat again. On the RX channel there will be a 1 MHz sine wave, which will be turned off for short periods of time and have its phase changed several times. Please see the respective source code for details.

The PulseBlasterDDS board is now ready for use!

III. Programming the PulseBlasterDDS

Instruction Set Architecture

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 is 32k. 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 512 memory-word models) and the longest being 2^52 clock cycles. All instructions have the same format and bit length, and all bit fields have to be filled. Figure 3 shows the fields and bit definitions of the 80-bit instruction word.

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

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

Figure 3: 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, next page, for output pattern and control bit assignments of the 24-bit output/control word.

Bit #	Function	Bit #	Function
23	Selects Frequency Register (bit 3)	11	Output Enable for SMA connector labeled DAC_OUT_2 (0 = on, 1 = off)
22	Selects Frequency Register (bit 2)	10	Output Enable for SMA connectors labeled DAC_OUT_1 and DAC_OUT_0 (0 = on, 1 = off)
21	Selects Frequency Register (bit 1)	9	RF phase reset for phase coherent switching, also routed to Output Connector DB25 pin 19
20	Selects Frequency Register (bit 0)	8	Output Connector DB25 pin 7
19	Selects Phase Register for SMA connector labeled DAC_OUT_1 (bit 3)	7	Output Connector DB25 pin 8
18	Selects Phase Register for SMA connector labeled DAC_OUT_1 (bit 2)	6	Output Connector DB25 pin 21
17	Selects Phase Register for SMA connector labeled DAC_OUT_1 (bit 1)	5	Output Connector DB25 pin 22
16	Selects Phase Register for SMA connector labeled DAC_OUT_1 (bit 0)	4	Output Connector DB25 pin 10
15	Selects Phase Register for SMA connectors labeled DAC_OUT_2 and DAC_OUT_0 (bit 3)	3	Output Connector DB25 pin 11
14	Selects Phase Register for SMA connectors labeled DAC_OUT_2 and DAC_OUT_0 (bit 2)	2	Output Connector DB25 pin 24
13	Selects Phase Register for SMA connectors labeled DAC_OUT_2 and DAC_OUT_0 (bit 1)	1	Output Connector DB25 pin 25
12	Selects Phase Register for SMA connectors labeled DAC_OUT_2 and DAC_OUT_0 (bit 0)	0	Output Connector DB25 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	Instruction	Data Field	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 0x6 for the 32k memory models. The timing controller has a fixed delay of three clock cycles and the value that one enters into the Delay Count field should account for this inherent delay. (NOTE: the pb_inst() family of functions in spinapi and the PulseBlaster Interpreter automatically account for this delay.)

Controlling the PulseBlasterDDS with spinapi

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

Please see the example programs for an an explanation of how to use spinapi. A reference document for the API is available online at:

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

Example Use of C Functions

```
/*
 * This program outputs a 1MHz sine wave on both TX and RX channels.
 * The TX Channel cycles between being on and off with a period
 * of 8us.
 * The TX and RX channels are set to have a 90 degree phase offset.
 */
#include <stdio.h>
#define PBDDS
#include "spinapi.h"

#define CLOCK 100.0

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

    printf ("Using spinapi library version %s\n", pb_get_version());
    if(pb_init() != 0) {
        printf ("Error initializing board: %s\n", pb_get_error());
        return -1;
    }

       // Tell the driver what clock frequency the board has
       pb_set_clock(CLOCK);

       // Program the frequency registers
       pb_start_programming(FREQ_REGS);
       pb_set_freq(1.0*MHz); // Register 0
       pb_set_freq(2.0*MHz); // Register 1
       pb_stop_programming();

       // Program RX phase registers (DAC_OUT_1) [Units in degrees]
       pb_start_programming(PHASE_REGS_1);
       pb_set_phase(0.0);  // Register 0
       pb_set_phase(90.0); // Register 1
       pb_stop_programming();

       // Program the TX phase registers (DAC_OUT_0 and DAC_OUT_2)
       pb_start_programming(PHASE_REGS_0);
       pb_set_phase(0.0);   // Register 0
       pb_set_phase(180.0); // Register 1
       pb_stop_programming();
```

```
        // Send the pulse program to the board
        pb_start_programming(PULSE_PROGRAM);

       //For PulseBlasterDDS boards, the pb_inst function is translated to the
       //pb_inst_tworf() function, which is defined as follows:
       //pb_inst_tworf(freq, tx_phase, tx_output_enable, rx_phase, rx_output_enable,
       //flags, inst, inst_data, length);

        //Instruction 0 - Continue to instruction 1 in 2us
        //Output frequency in freq reg 0 to both TX and RX channels
        //Set all TTL output lines to logical 1
       start = pb_inst(0, 1, TX_ANALOG_ON, 0, RX_ANALOG_ON,
                    0x1FF, CONTINUE, 0, 2.0*us);

        //Instruction 1 - Continue to instruction 2 in 4us
        //Output frequency in freq reg 0 RX channels, TX channel is disabled
        //Set all TTL outputs to logical 0
        pb_inst(0, 1, TX_ANALOG_OFF, 0, RX_ANALOG_ON,
             0x000, CONTINUE, 0, 4.0*us);

        //Instruction 2 - Branch to "start" (Instruction 0) in 2us
        //Output frequency in freq reg 0 to both TX and RX channels
        //Set all TTL output lines to logical 0
        pb_inst(0, 1, TX_ANALOG_ON, 0, RX_ANALOG_ON,
             0x000, BRANCH, start, 2.0*us);

        pb_stop_programming();

        // Trigger the pulse program
        pb_start();

    // Retreive the status of the current board. This will be 0x04 since
    // the pulse program is a loop and will continue running indefinitely.
    status = pb_read_status();
    printf("status: 0x%.2x\n", status);

        // Release Control of the PulseBlasterDDS Board
        pb_close();

        return 0;
}
```

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

IV. Connecting to the PulseBlasterDDS Board

Connector Information

SMA Connectors labeled DAC_OUT_0, DAC_OUT_1, and DAC_OUT_2

Outputs DDS signals generated by the user's Program. The output impedance is 50 ohms. Output power is approximately 10 dBm. Figure 4 is a portion of figure 2 that illustrates which RF output corresponds to which SMA connector.

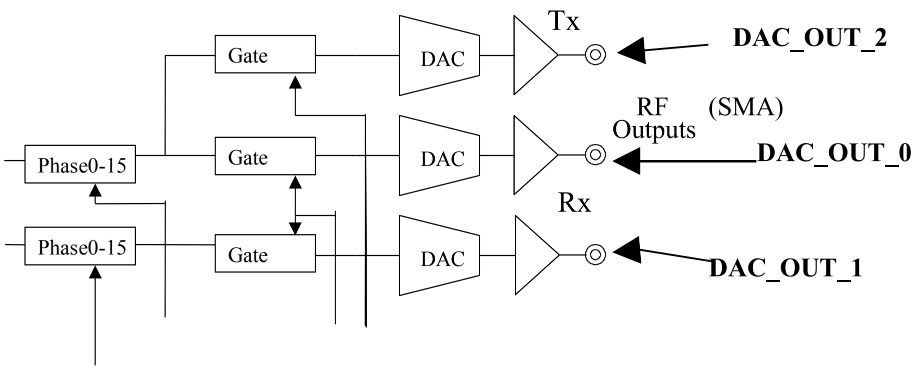

Figure 4: SMA Connectors

DB-25 - TTL Output Signal Bits

Outputs TTL signals generated by the user's Program. Please consult the table below for bit assignments.

Pin Assignments			
Pin#	Bit#	Pin#	Bit#
1	GND	14	GND
2	Reserved	15	Reserved
3	GND	16	Reserved
4	Reserved	17	GND
5	Reserved	18	Reserved
6	GND	19	9
7	8	20	GND
8	7	21	6
9	GND	22	5
10	4	23	GND
11	3	24	2
12	GND	25	1
13	0		

Table 3: Output bits and signals of the PulseBlasterDDS board

IDC Connector Status - Pin Assignments

The IDC connector labeled Status outputs TTL signals based on status of the user's program. Please consult the table below for pin assignments.

Pin Assignments			
Pin#		Pin#	
1	Stopped	5	Running
2	GND	6	GND
3	Reset	7	Waiting
4	GND	8	GND

Table 4: Status signals of the PBDDS-III

The status pins correspond to the current state of the pulse program and are defined as follows:

Stopped – Driven high when the PulseBlaster device has encountered a STOP Op Code during program execution and has entered a stopped state.

Reset – Driven high 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 – the PulseBlaster device has encountered a WAIT Op Code and is waiting for the next trigger (either hardware or software) to resume operation.

Header JP100

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

HW_Trigger is pulled high by default, and pin 1 is active (pin 2 = GND). When a low state is detected (e.g., when shorting pins 1-2), it initiates code execution. This trigger will also restart execution of a program from the beginning of the code if it is detected after the design has reached an idle state. The idle state could have been created either by reaching the STOP Op Code of a program, or by the detection of the HW_Reset signal. When the WAIT Op Code is used in the pulse program, the HW_Trigger will cause the program to continue to the next instruction.

HW_Reset is pulled high by default, and pin 3 is active (pin 4 = GND). It can be used to halt the execution of a program by pulling it low (e.g., by shorting pins 3-4). 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 or by a hardware trigger.

SMA Connector labeled "SMA0"

This SMA connector outputs the reference clock as a 3.3 V TTL signal, i.e., it generates positive-only voltage. The output resembles a square wave if properly terminated. This signal can be measured with an oscilloscope using either a high impedance probe at the SMA connector or a 50 ohm coaxial line that is terminated.

SMA Connector labeled "SMA400"

This SMA connector is used as an external clock input. Before attaching the external clock, the internal clock must be removed from its socket. The internal clock's orientation should be noted (if the internal clock is reconnected, it must be inserted in the same orientation or board damage may occur). Also, the external clock must be a 3.3 V TTL signal. Another requirement is that a 50 ohm resistor be soldered directly on the board on R401 pads or use a T connector with a 50 ohm terminator connected directly to SMA400. Failure to follow the above requirements could result in damaging the board.

Appendix I.

```
/*
 * This program makes use of all available instructions (except WAIT)
 *
 * On the TX channel:
 * There will be a single period of a 1MHz sine wave, followed by a 1us gap.
 * This pattern will be repeated 2 more times followed by a long (5ms) gap,
 * after which the pattern will repeat again.
 *
 * On the RX channel:
 * There will be a 1 MHz sine wave, which will be turned off for short
 * periods of time and have its phase changed several times. See the
 * pulse program below for details.
 *
 */
#include <stdio.h>
#define PBDDS
#include "spinapi.h"
#define CLOCK 100.0

int main(void)
{
    int start, loop, sub;

    printf ("Using spinapi library version %s\n", pb_get_version());
    if(pb_init() != 0) {
        printf ("Error initializing board: %s\n", pb_get_error());
        return -1;
    }

    // Tell driver what clock frequency the board has
    pb_set_clock(CLOCK);

    //Program the frequency registers
    pb_start_programming(FREQ_REGS);
    pb_set_freq(1.0*MHz);  // Set register 0
    pb_set_freq(2.0*MHz);  // Set register 1
    pb_set_freq(3.0*MHz);  // Set register 2
    pb_set_freq(4.0*MHz);  // Set register 3
    pb_set_freq(5.0*MHz);  // Set register 4
    pb_set_freq(6.0*MHz);  // Set register 5
    pb_set_freq(7.0*MHz);  // Set register 6
    pb_set_freq(8.0*MHz);  // Set register 7
    pb_set_freq(9.0*MHz);  // Set register 8
    pb_set_freq(10.0*MHz); // Set register 9
    pb_set_freq(11.0*MHz); // Set register 10
    pb_set_freq(12.0*MHz); // Set register 11
    pb_set_freq(13.0*MHz); // Set register 12
    pb_set_freq(14.0*MHz); // Set register 13
    pb_set_freq(15.0*MHz); // Set register 14
    pb_set_freq(16.0*MHz); // Set register 15
    pb_stop_programming();

    //Program the RX phase registers (DAC_OUT_1) [Units in degrees]
    pb_start_programming(PHASE_REGS_1);
    pb_set_phase(0.0);   // Set register 0
    pb_set_phase(22.5);  // Set register 1
    pb_set_phase(45.0);  // Set register 2
    pb_set_phase(67.5);  // Set register 3
    pb_set_phase(90.0);  // Set register 4
    pb_set_phase(112.5); // Set register 5
    pb_set_phase(135.0); // Set register 6
    pb_set_phase(157.5); // Set register 7
    pb_set_phase(180.0); // Set register 8
    pb_set_phase(202.5); // Set register 9
    pb_set_phase(225.0); // Set register 10
    pb_set_phase(247.5); // Set register 11
    pb_set_phase(270.0); // Set register 12
    pb_set_phase(292.5); // Set register 13
    pb_set_phase(315.0); // Set register 14
    pb_set_phase(337.5); // Set register 15
    pb_stop_programming();

    //Program the TX phase registers (DAC_OUT_0 and DAC_OUT_2)
pb_start_programming(PHASE_REGS_0);
pb_set_phase(0.0);   // Set register 0
pb_set_phase(22.5);  // Set register 1
pb_set_phase(45.0);  // Set register 2
pb_set_phase(67.5);  // Set register 3
pb_set_phase(90.0);  // Set register 4
pb_set_phase(112.5); // Set register 5
pb_set_phase(135.0); // Set register 6
pb_set_phase(157.5); // Set register 7
pb_set_phase(180.0); // Set register 8
pb_set_phase(202.5); // Set register 9
pb_set_phase(225.0); // Set register 10
pb_set_phase(247.5); // Set register 11
pb_set_phase(270.0); // Set register 12
pb_set_phase(292.5); // Set register 13
pb_set_phase(315.0); // Set register 14
pb_set_phase(337.5); // Set register 15
pb_stop_programming();


//Begin pulse program
pb_start_programming(PULSE_PROGRAM);

//For PulseBlasterDDS boards, the pb_inst function is translated to the
//pb_inst_tworf() function, which is defined as follows:
//pb_inst_tworf(freq, tx_phase, tx_output_enable, rx_phase, rx_output_enable,
//flags, inst, inst_data, length);

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 5 in 1us
start =
pb_inst(0,0,TX_ANALOG_OFF,0,RX_ANALOG_ON,0x1FF,JSR,sub,1*us);

// Loop. The next two instructions make up a loop which will be repeated
// 3 times.
// Instruction 1 - Beginning of Loop (Loop 3 times).  Continue to next instruction in 1us
loop =
pb_inst(0,0,TX_ANALOG_OFF,0,RX_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,TX_ANALOG_ON,0,RX_ANALOG_OFF,0x0,END_LOOP,loop,1*us);

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

// Instruction 4 - Branch to "start" (Instruction 0) after 1us
// This has the effect of looping the entire program indefinitely.
pb_inst(0,0,TX_ANALOG_OFF,0,RX_ANALOG_ON,0x0,BRANCH,start,1*us);

// Subroutine. This subroutine is called by using the JSR instruction
// and specifying the address of instruction 5. (As is done in instruction
// 1)
// Instruction 5 - Reset phase and continue to next instruction in 2us
pb_inst(0,4,TX_ANALOG_OFF,8,RX_ANALOG_ON,0x200,CONTINUE,0,2*us);

// Instruction 6 - Return from Subroutine after 2us
pb_inst(0,8,TX_ANALOG_OFF,8,RX_ANALOG_ON,0x0,RTS,0,2*us);

pb_stop_programming();

// Trigger the pulse program
pb_start();

// Release control of the PulseBlasterDDS board
pb_close();

return 0;
}
```

Contact Information

Phone	(352) 271-7383
FAX	(352) 371-8679
Email	Web contact form
Web	http://www.spincore.com/

Product URL:

http://www.spincore.com/CD/PulseBlasterDDS/PCI/SP3