PulseBlasterDDS-IV-1000

Owner's Manual

SpinCore Technologies, Inc.

http://www.spincore.com

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

We appreciate your business!

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

Table of Contents

I. Introduction

Product Overview

The PulseBlasterDDS-IV-1000 is a programmable pattern- and waveform- generation system from SpinCore Technologies, Inc. that couples SpinCore's unique intelligent-pattern-generation processor core, the PulseBlaster, with two direct digital synthesis (DDS) units for use in system control and pulse generation. By interfacing the system with a high-performance Analog Devices' AD9148 Quad DAC (Digital to Analog Converter) operating at a sampling frequency of 1000 MS/s, a range of RF (Radio Frequency) signals can be produced from 200 kHz to 400 MHz. Additionally, 16 independently-programmable digital TTL outputs are provided to generate user-defined pulse sequences.

The DDS-IV is a complete high frequency excitation system for high field NMR, MRI, NQR, spintronics, quantum computing, and related resonance and testing technologies up to 400 MHz.

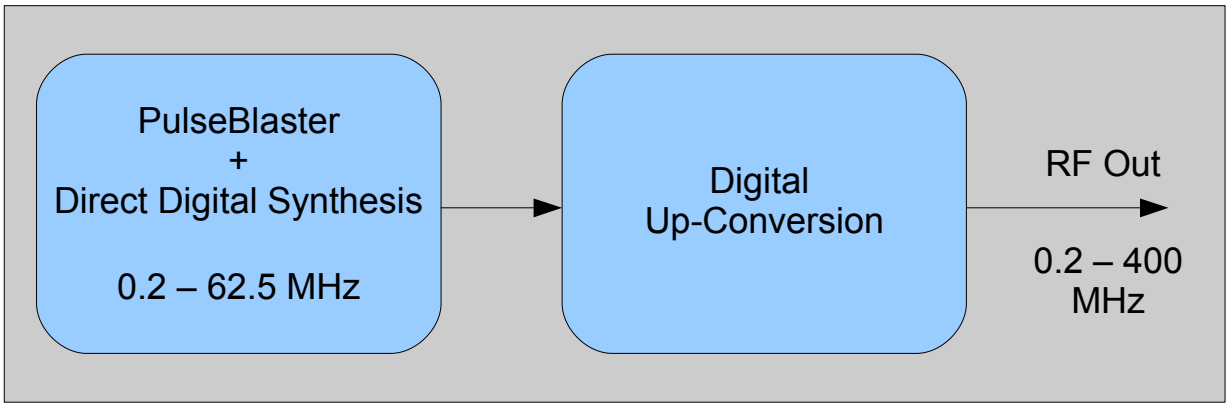


Figure 1: The PulseBlasterDDS-IV-1000 DDS cores produce patterns in the base-band range of 0.2 to 62.5 MHz. These signals are then digitally up-converted and placed in the desired output band up to 400 MHz.

Key Features

- Two independently-programmable DDS cores capable of outputting RF signals up to 400 MHz, each with 256/128/128 programmable frequency/phase/amplitude registers
- Each DDS core is equipped with quadrature outputs for a total of four RF outputs.
- 16 independently-programmable digital TTL outputs with pulse resolutions of 8 ns.
- Advanced pulse-program flow including loops, subroutines, and hardware/software interrupts.
- 256 programmable interrupts (interrupts are immediate and always active).

Product Architecture

Figure 2 presents the general architecture of the PulseBlasterDDS-IV-1000 system. The three major building blocks of the DDS-IV are the PulseBlaster timing core (1), the two independent DDS Cores (2), and the AD9148 Quad DAC (3).

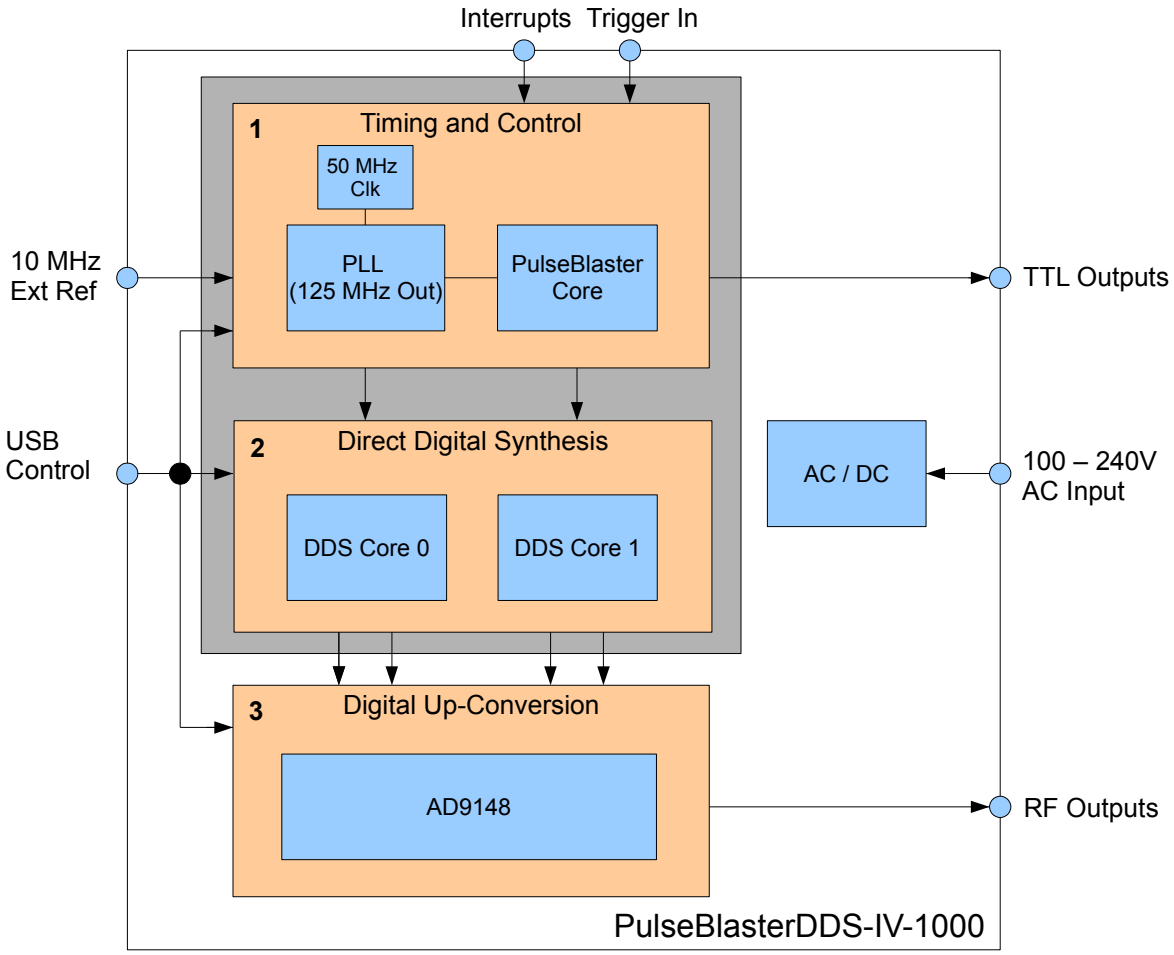


Figure 2: Block diagram of the PulseBlasterDDS-IV-1000 architecture. The PulseBlaster core provides precision timing and control which directs the DDS units that provide samples to the AD9148 Quad DAC. The PulseBlaster core also outputs 16 independent digital TTL pulses, each with a timing resolution of 8 nanoseconds.

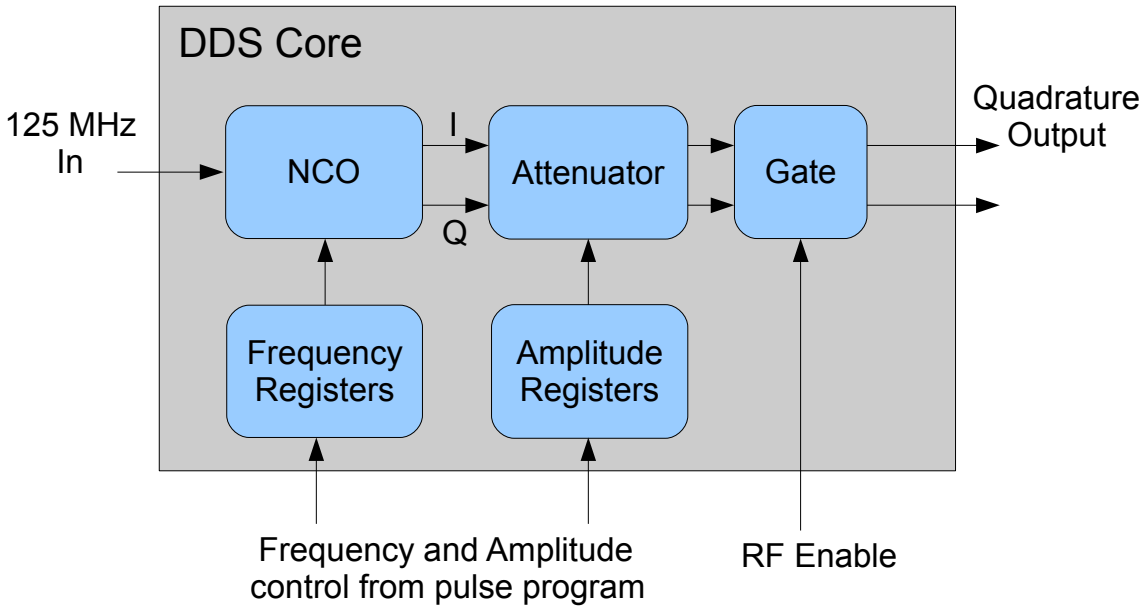


Figure 3: Block diagram of the Direct Digital Synthesis (DDS) core. The PulseBlaster core provides the control signals which configure the DDS cores to output signals at the desired frequency and amplitude.

Figure 3, shown above, presents the architecture of the Direct Digital Synthesis (DDS) core. There are two DDS cores in the DDS-IV system which provide independent RF output channels. The NCO (Numerically Controlled Oscillator) provides the quadrature data for each RF output channel.

The NCO operates with a 125 MHz clock frequency and is configured by selecting one of the 256 user-programmable frequency registers. There are also 128 programmable amplitude and phase registers to adjust the output RF amplitude and initial RF phase.

The PulseBlaster Core controls the timing and output of the RF pulses and provides the necessary control signals to select the desired DDS frequency, phase, and amplitude registers. Additionally, the PulseBlaster core generates 16 independently-programmable digital TTL outputs.

Figure 4 diagrams the interface between the PulseBlasterDDS cores and the AD9148 DAC. The AD9148 contains three configurable digital half-band interpolation filters which can be configured to shift the base-band frequencies produced by the DDS cores (0-62.5 MHz) up to higher frequencies (up to 400 MHz). When all of the half-band interpolation filters are used together, the maximum interpolation ratio of 8x is achieved, and the AD9148 will interpolate eight samples of data for each sample it receives from the DDS cores.

In order to ensure that the half-band interpolation filters can be adjusted to produce all frequencies up to 400 MHz, the DDS generates two quadrature outputs, each consisting of two parts: an In-Phase channel (no phase offset) and Quadrature channel (90-degrees offset) which represent a single signal.

Each DDS core outputs a quadrature stream into one of the two data paths within the AD9148. The two complex data paths within the AD9148 feed the four DAC outputs. Consequently, only two of the DAC outputs (Output 1 and Output 3) generate independent RF waveforms. Outputs 2 and 4 produce the the same signal as Outputs 1 and 3 with a phase offset of 90 degrees.

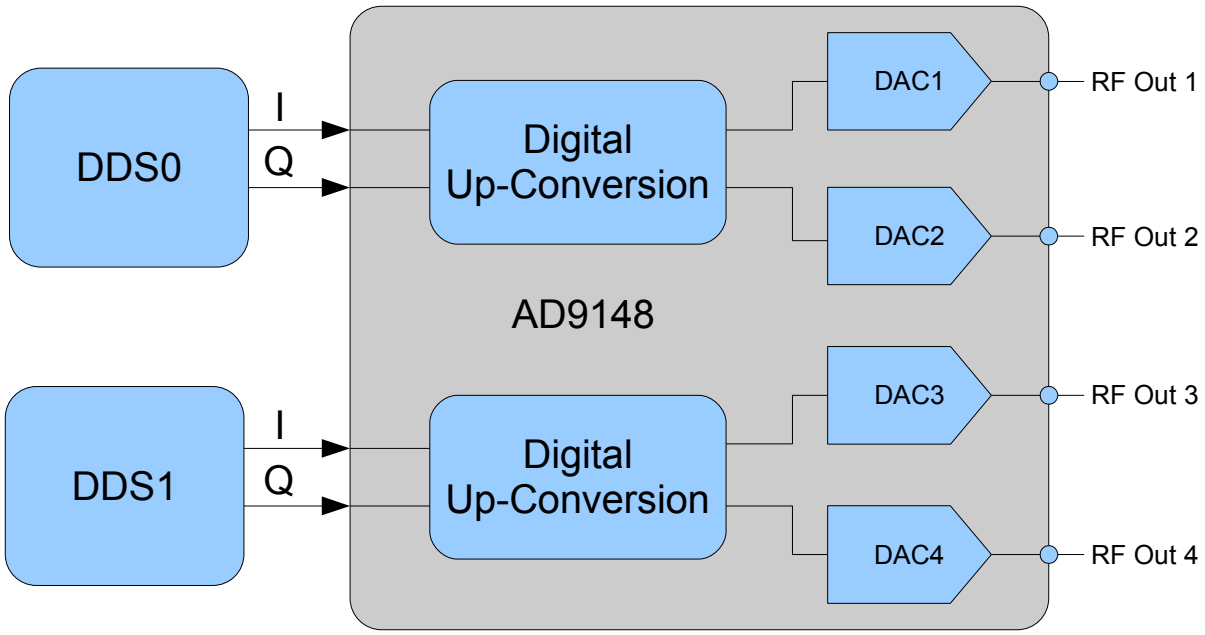


Figure 4: Overview the DDS/AD9148 DAC interface. The DDS cores output quadrature data (I and Q channels) to the AD9148. The digital up-conversion feature of the AD9148 consists of half-band interpolation filters that are used to shift base-band frequencies from the DDS cores to higher frequencies (up to the Nyquist frequency). Any desired output higher than 62.5 MHz requires the use of these digital up-conversion features.

Product Specifications

	Parameter	Min	Typical	Max	Units
Analog Output	D/A sampling rate		1000		MHz
	D/A sampling precision			14	bits
	Output voltage range (peak-peak, terminated with a 50 Ohm load)			1.05	V
	Phase resolution			0.9	deg.
	Frequency resolution		0.12		Hz
	TTL to RF Latency		508		ns
	Frequency Registers per DDS		256		registers
	Phase Registers per DDS		128		registers
	Amplitude Registers per DDS		128		registers
Digital Output	Number of digital TTL outputs		16		bits
	Logical 1 output voltage		3.3[1]		V
	Logical 0 output voltage		0		V
	Output drive current			66	mA
	Rise/Fall time			< 1	ns
Digital Input	Number of Interrupts		256		
	HW Interrupt Activation Level	2.5		3.3	V
	HW Trigger/Reset Activation Level			0.8	V
	10 MHz Clock Input Reference Voltage, Square Wave			3.3	V
Pulse Program	Number of instruction words		8K		words
	Pulse timing resolution		8		ns
	Instruction time length	40 ns		693 days	
Power Input	AC Voltage	110		240	V
	AC Current			5	A
	AC Frequency	50		60	Hz

Table 1: Technical specifications for the PulseBlasterDDS-IV-1000.

[1] This is the un-terminated voltage. The required minimum logical high TTL level of 2.5 V is attained when the load impedance is 150 Ω.

II. System Installation

To install the PulseBlasterDDS-IV-1000 system, please complete the following steps:

1. Download and install the latest PulseBlasterDDS-IV API version for your architecture available at: http://www.spincore.com/products/PulseBlasterDDS-IV-1000 under “Manuals, etc.” This C/C++-based API provides a custom programming interface developed by SpinCore Technologies, Inc. for use with the PulseBlasterDDS-IV.

2. Connect the provided USB 2.0 and power cables to the PulseBlasterDDS-IV.

3. Turn on the PulseBlasterDDS-IV using the toggle switch on the front. The internal fan should be audible.

4. Verify the PulseBlasterDDS-IV system is found by Windows by verifying it is listed in the Windows device manager.

We recommend running the pre-compiled example programs in the examples directory to verify that your device is functioning properly. If your device is not found by the operating system, please verify that it is powered on and that you have installed the proper API version which matches the operating system of your computer.

Testing the PulseBlasterDDS-IV-1000

Once the PulseBlasterDDS-IV system has been installed properly, you are ready to test the functionality of your device with the example programs provided by the DDS-IV API installer package . You can begin by running the pre-compiled examples programs and verifying that the output matches the description displayed after execution. After the device functionality has been verified, you can modify the C source files as needed to start making your own programs. For information on compiling examples using Microsoft Visual Studio, please see: http://www.spincore.com/support/PBDDSIV/MSVC/msvc_tutorial.html. All example programs are easily verified using an oscilloscope triggered by any of the TTL outputs.

The program *pbddsiv_read_firmware_example* reads the DDS-IV firmware register and displays the value. This is the simplest provided example program and should be executed to verify device communication.

The program *pbddsiv_pulsed_rf_example* provides a simple example outputting RF pulses on each RF output. RF outputs one and two output a 10 MHz sine wave. RF outputs three and four output a 20 MHz sine wave. The pulse is on for 10.0 us, off for 5.0 us, then repeats indefinitely.

The program *pbddsiv_4phase_pulsed_rf_example* provides an example outputting four RF pulses on each channel, each time with a different phase. Each RF channel outputs four pulses on for 10.0 us, off for 5.0 us, with phase offsets of 0°, 90°, 180°, and 270° in sequence. The sequence then repeats indefinitely.

The program *pbddsiv_pulsed_sinc_example* provides an example of using the AWG to modulate the output RF signal with a sinc-wave envelope.

The program *pbddsiv_frequency_modulation_example* demonstrates the zero-latency continuous-frequency-switching capability of the DDS, and demonstrates how to select the RF baseband using the DDS-VI API.

III. Using the PulseBlasterDDS-IV-1000

The DDS-IV API available on our website at: http://www.spincore.com/products/PulseBlasterDDS-IV-1000/ provides a C/C++ interface for controlling and configuring the PulseBlasterDDS-IV system. The DDS-IV API provides easy-to-use functions for controlling the PulseBlaster core, on-chip DDS units, and the external AD9148 quad DAC. Users can easily specify pulse programs and configure each RF output.

For specifics on using each function provided by the DDS-IV API, please see the PDF available in the "doc" directory installed by the DDS-IV API installer. This PDF provides a detailed description of each function and any arguments required.

Controlling the PulseBlasterDDS-IV-1000

This section describes the function and use of each feature of the PulseBlasterDDS-IV API.

The PulseBlasterDDS-IV-1000 is a versatile excitation system with several programming options. The following steps outline the basic approach to programming and running a program on the DDS-IV:

1. Load frequency, phase, and amplitude registers with the desired values.
2. Configure the external DAC settings using the function *pbddsiv_configure_interpolation_mode(...)*
3. Specify a pulse program which will control the timing of the experiment.
4. Trigger the pulse program. The experiment will then proceed autonomously.

The PulseBlasterDDS-IV API is a C-based software library used for controlling your PulseBlasterDDS-IV system. Using this library you can program and configure your PulseBlasterDDS-IV.

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 dynamically-linked libraries such as the PulseBlasterDDS-IV API library: *pbddsiv.dll.*

We recommend the user look at the provided example programs to demonstrate how to use the PulseBlasterDDS-IV. The example programs provide an overview of using all of the provided API functions.

The PulseBlasterDDS-IV is a highly versatile excitation system, and as such there are many possible approaches to controlling the device. However, most applications can be programmed using the following steps:

1. Select the device (if using more than one device.)
2. Initialize the device and program the default device parameters.
3. Program the required DDS frequency, amplitude, and phase registers, as well as the DDS shape memory (if using the AWG feature.)
4. Specify the interpolation settings (if frequencies about 0 to 62.5 MHz are required.)
5. Specify a pulse program which will control the TTL output and RF output configuration and timings.
6. Program any interrupt vectors using the instruction address returned from the previous step.
7. Trigger the pulse program.

Selecting and Initializing the System

By default, the API uses the first PulseBlasterDDS-IV device found in the system. In order to select between multiple DDS-IV systems connected to a host, it is necessary to first call the *pbddsiv_select_device(..)* routine with the appropriate device number. To determine the number of your device when multiple DDS-IV systems are connected, please check the Device Manager device numberings.

Before using any API functions to control your system, it is first initialize the device using the *pbddsiv_init()* routine. The *pbddsiv_init()* routine opens a handle to the selected device allowing the API to communicate with it.

Once the device has been initialized, it is necessary to set default PulseBlaster, DDS, and DAC settings using *pbddsiv_set_defaults().* Next, the clock frequency must be specified so that the API knows the operating clock frequency. This can be accomplished using *pbddsiv_set_core_clock(..).* This function takes the frequency (in MHz) at which the PulseBlaster core is operating. Please see Example 1 below.

```
pbddsiv_select_device(1); //(Optional) Select a specific device (in this case, device 1) 0 is default

//Initialize the PulseBlasterDDS-IV
if(pbddsiv_init() != 0) {
    fprintf(stderr, "Failed to initialize DDS-IV device. Please see debug log for more information.\n");
    system("pause");
    return -1;
}

//Set the default DAC/PulseBlaster settings.
pbddsiv_set_defaults();

//Set the core clock frequency - typically 125.0 MHz
pbddsiv_set_core_clock(125.0);

pbddsiv_reset(); //Reset the device state before use

/* Other device programming */

pbddsiv_close(); //Always close the device handle before exiting the program
```

Example 1: Selecting and initializing the PBDDS-IV.

Configuring the DDS Units

The PulseBlasterDDS-IV contains two independently-programmable DDS units. Each DDS provides quadrature data to one of the two channels of the AD9148 DAC. The RF output frequency, amplitude, and initial phase of each DDS is controlled by selecting from a bank of on-board registers. Each DDS contains its own set of frequency, amplitude, and phase registers. The registers should be programmed with the appropriate values after device initialization using the PBDDS-IV API.

Each DDS register bank can be selected for programming using the *pbddsiv_start_programming(..)* function. This function allows the user to specify the target device type, device number to program, and target device resource. For example, *pbddsiv_start_programming(DDS, 0, FREQUENCY_REGISTERS)* selects the frequency registers of DDS 0 for programming. Each instruction in the pulse program is capable of selecting the desired register during execution. The available registers are listed in Table 2.

Device Type	*Device Number*	*Resource Type*	*Number of Registers*
DDS	0	FREQUENCY_REGISTERS	256
DDS	0	PHASE_REGISTERS	128
DDS	0	AMPLITUDE_REGISTERS	128
DDS	0	SHAPE_PERIOD_REGISTERS	7
DDS	1	FREQUENCY_REGISTERS	256
DDS	1	PHASE_REGISTERS	128
DDS	1	AMPLITUDE_REGISTERS	128
DDS	0	SHAPE_PERIOD_REGISTERS	7

Table 2: Available DDS Registers

```
//Configure DDS-0
pbddsiv_start_programming(DDS, 0, FREQUENCY_REGISTERS);
   pbddsiv_program_frequency(5.0);
   pbddsiv_program_frequency(10.0);
pbddsiv_stop_programming();

pbddsiv_start_programming(DDS, 0, PHASE_REGISTERS);
   pbddsiv_program_phase(0.0);
pbddsiv_stop_programming();

pbddsiv_start_programming(DDS, 0, AMPLITUDE_REGISTERS);
   pbddsiv_program_amplitude(1.0);
pbddsiv_stop_programming();

//Configure DDS-1
pbddsiv_start_programming(DDS, 1, FREQUENCY_REGISTERS);
   pbddsiv_program_frequency(20.0);
   pbddsiv_program_frequency(15.0);
pbddsiv_stop_programming();

pbddsiv_start_programming(DDS, 1, PHASE_REGISTERS);
   pbddsiv_program_phase(0.0);
pbddsiv_stop_programming();

pbddsiv_start_programming(DDS, 1, AMPLITUDE_REGISTERS);
   pbddsiv_program_amplitude(1.0);
pbddsiv_stop_programming();
```

Example 2: Configuring the DDS registers.

In addition to the DDS registers, each DDS has a programmable shape memory which can be used to specify an envelope for the RF output pulse. In order to use the AWG shape feature, the user first program the DDS shape memory, and then program the DDS shape-period registers.

The function *pbddsiv_load_dds_shape(..)* takes a target DDS number and a floating-point array of 1024 values. Each value in the array specifies a scale factor which is applied to the output RF waveform over one shape period. Each value in the array must be between -1.0 and 1.0 inclusive, or the function will return an error.

Once the shape memory has been programmed, the user must also program the shape-period registers. Each register stores a time value which refers to how long it takes to iterate over the shape memory once. For example, if the pulse duration is 20.0 us, and the shape-period is 10.0 us, the shape envelope will be output twice per pulse duration. To use the shape feature, the pulse program must use *pbddsiv_program_inst_shape(...).*

```
//Generate sinc shape data with two lobes
shape_make_sinc (dds_shape_data, 2);

//Load DDS shape information
pbddsiv_load_dds_shape(0, dds_shape_data);
pbddsiv_load_dds_shape(1, dds_shape_data);

pbddsiv_start_programming(DDS,0, SHAPE_PERIOD_REGISTERS);
   pbddsiv_program_shape_period(5.0 * us);
pbddsiv_stop_programming();

pbddsiv_start_programming(DDS, 1, SHAPE_PERIOD_REGISTERS);
   pbddsiv_program_shape_period(5.0 * us);
pbddsiv_stop_programming();
```

Example 3: Programming the DDS-IV shape feature.

Configuring the AD9148 Interpolation Settings

The DDS-IV API provides a simple method of configuring the interpolation settings (and therefore baseband frequency) of the AD9148 DAC. The function *pbddsiv_configure_interpolation_mode(..)* takes a *interpolation_settings_t* data structure which contains the interpolation settings to be programmed. This structure contains two fields, *factor* and *mode*, as defined in the header file “ddsivapi.h”. Factor refers to the interpolation factor (i.e., number of interpolation filters used). Mode refers to the interpolation filter settings. Valid interpolation settings can be found in Table 3 below.

Mode	Compatible Factor Settings	Baseband Frequency
Mode_DC	2, 4, or 8	0 MHz
Mode_Fs8	2, 4, or 8	125 MHz
Mode_Fs4	2, 4 or 8	250 MHz
Mode_3Fs8	4 or 8	375 MHz
Mode_Shifted_DC	8	62.5 MHz
Mode_Shifted_Fs8	8	187.5 MHz
Mode_Shifted_Fs4	8	312.5 MHz

Table 3: Valid interpolation settings values and resulting base-band frequency. These modes are further illustrated in Figure 5.

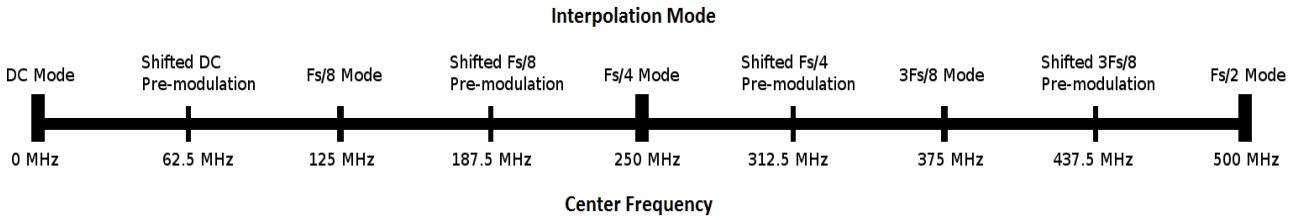


Figure 5: Frequency bands corresponding to their respective coarse modulation modes.

It is important to note that the function *pbddsiv_set_defaults()* should be called prior to configuring the interpolation settings to set the default values for all of the registers found on the AD9148.

```
pbddsiv_set_defaults();  //Ensure all registers have been programmed

//Configure AD9148 DAC
struct interpolation_settings_t settings;

settings.factor = 8;
settings.mode = Mode_Fs8; //Baseband setting of 125 MHz

pbddsiv_configure_interpolation_mode(&settings); //Write the settings
```

Example 4: Programming the interpolation settings.

Real Time vs. Slow Software Control

As stated previously, the PulseBlasterDDS-IV-1000 API allows you to input base-band frequencies of up to 62.5 MHz in real time using the DDS cores and PulseBlaster timing core. Using the AD9148 DAC software, you can shift the DDS frequencies up to cover the range from DC to 400 MHz.

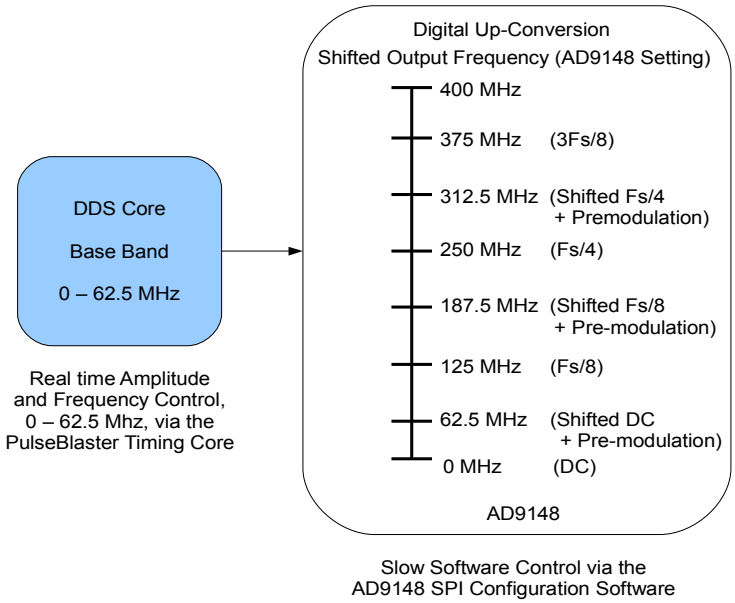


Figure 6: The slow software AD9148 interface is used to output frequencies across the entire output range, DC to 400 MHz. The PulseBlaster timing core provides real time amplitude and frequency control in the base band (0 to 62.5 MHz)

Writing Pulse Programs

After the device has been selected, initialized, and the DDS units have been configured, it is time to program the pulse program which controls the TTL and RF outputs. Similar to programming the DDS register banks, the function *pbddsiv_start_programming(..)* is used to start programming the PulseBlaster core instruction memory. To program core 0 we use: *pbddsiv_start_programming(PULSEBLASTER, 0, INSTRUCTION_MEMORY)*.

pbddsiv_program_inst			**pbddsiv_program_inst_shape**		
Argument	**Type**	**Description**	**Argument**	**Type**	**Description**
flags	uint32_t	TTL flag values (16 bits).	flags	uint32_t	TTL flag values (16 bits).
oe	uint32_t[2]	RF output enable.	oe	uint32_t[2]	RF output enable.
phase_rst	uint32_t[2]	Reset DDS phase.	phase_rst	uint32_t[2]	Reset DDS phase.
freq	uint32_t[2]	Frequency register number.	freq	uint32_t[2]	Frequency register number.
phase	uint32_t[2]	Phase register number.	phase	uint32_t[2]	Phase register number.
amplitude	uint32_t[2]	Amplitude register number.	amplitude	uint32_t[2]	Amplitude register number.
Inst	Instruction	See instruction table.	shape_period	uint32_t[2]	Shape period register number.
inst_data	uint32_t	See instruction table.	Inst	Instruction	See instruction table.
time_sec	double	Instruction duration in seconds.	inst_data	uint32_t	See instruction table.
			time_sec	double	Instruction duration in seconds.

Table 4: Breakdown of the pbddsiv_program_inst(..) and pbddsiv_program_inst_shape(..) functions.

For ease of use, the API defines a mnemonic constant which refers to each instruction opcode. Please see Table 5 for a list of each op code, the associated mnemonic constant, and information about the instruction data. Example 5 below illustrates how to write a simple pulse program to the PBDDS-IV.

```
//Simple pulse program example of a single RF pulse on each RF output.
pbddsiv_start_programming(PULSEBLASTER, 0, INSTRUCTION_MEMORY);
                        phase_reset[0] = phase_reset[1] = 1;
                        pbddsiv_program_inst(0x0000, oe, phase_reset, frequency, phase, amplitude, CONTINUE, 0, 5.0 * us);

                        phase_reset[0] = phase_reset[1] = 0;
                        oe[0] = oe[1] = 1;
                        address = pbddsiv_program_inst(0xFFFF, oe, phase_reset, frequency, phase, amplitude, CONTINUE, 0,
10.0 * us);

                        phase_reset[0] = phase_reset[1] = 1;
                        oe[0] = oe[1] = 0;
                        pbddsiv_program_inst(0x0000, oe, phase_reset, frequency, phase, amplitude, BRANCH, address, 5.0 * us);
pbddsiv_stop_programming();
```

Example 5: Programming a pulse program using the *pbddsiv_start_programming(..)* interface.

Op Code #	Instruction	inst_data field	Function
0	CONTINUE	Unused	Program execution continues to the next instruction.
1	STOP	Unused	Stop execution of program. Aborts the operation of the micro-controller with no control of output states (all TTL values remain from previous instruction). Recommended that prior to the STOP op-code a short interval (minimum six clock cycles) be added to set the output states as desired.
2	LOOP	Number of desired loops (must be at least 1).	Specify beginning of a loop. Execution continues to next instruction. Data used to specify number of loops.
3	END_LOOP	Address of the beginning of the loop.	Specify end of a loop. Execution returns to begging of loop and decrements loop counter.
4	JSR	Address of the first subroutine instruction.	Program execution jumps to beginning of a subroutine.
5	RTS	Unused	Program execution returns to the instruction after JSR was called.
6	BRANCH	Address of the instruction to branch to.	Program execution continues at specified instruction. This behaves like the GOTO statement found in many programming languages.
7	LONG_DELAY	Number of desired loops. Must be at least 2.	Used to long intervals. Data field specifies a multiplier of the delay field. Execution continues at the next instruction.
8	WAIT	Unused	Program execution pauses and waits for a software of hardware trigger to resume it.

Table 5: PulseBlaster Instruction Set

Sample Output

In this section you will find oscilloscope screen captures of the PulseBlasterDDS-IV-1000 system running through its example and other test programs. All screen shots were captured with a Tektronix TDS 2024B. This scope greatly attenuates high frequency signals over 200 MHz.

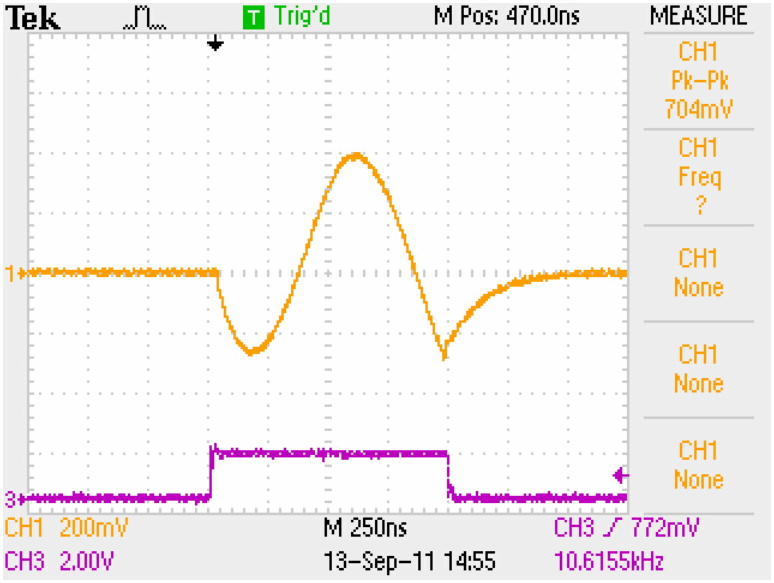


Figure 7: A 1000 ns 1 MHz RF pulse is shown in the base band with no digital up-conversion features enabled. The 500 ns stabilization time after the RF gate is disabled is associated with the AD9148 DAC at lower frequencies.

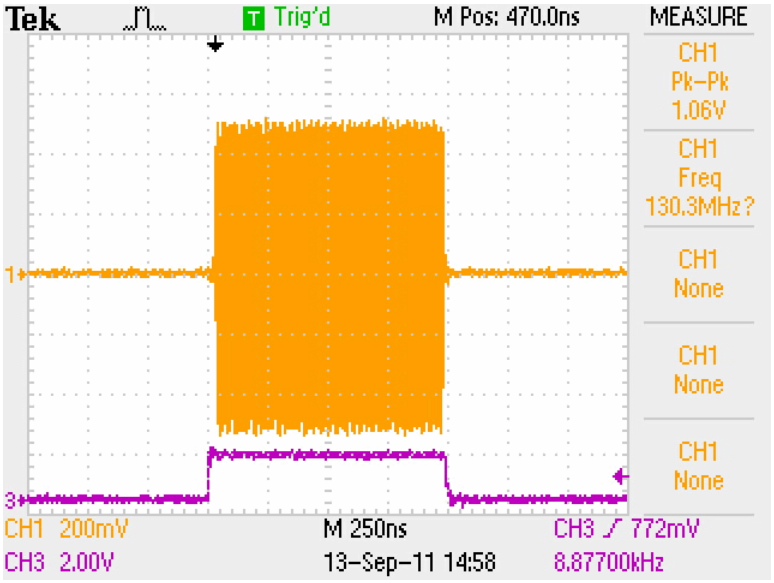


Figure 8: The digital up-conversion option "Fs/8" has been applied to the signal from Figure 7. The signal is shifted up by one-eight of the DAC sampling rate, or 125 MHz.

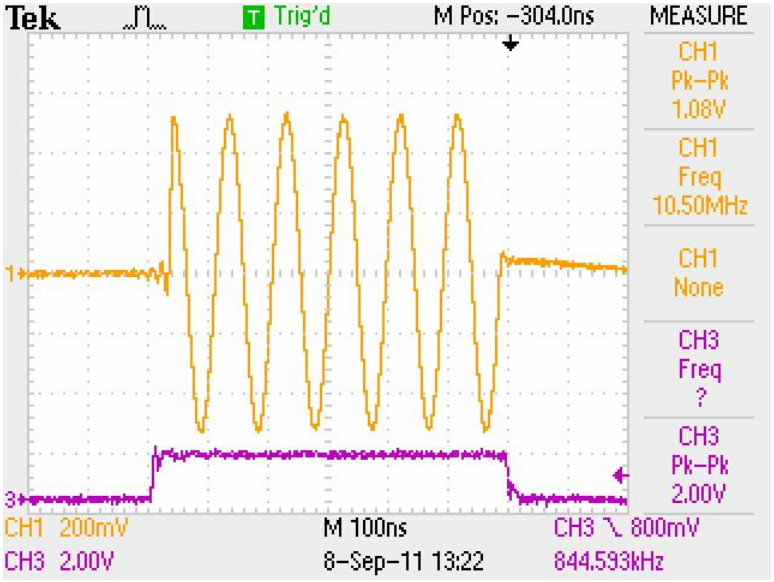


Figure 9: A 600 ns RF pulse at 10 MHz is shown in the base band with no digital up-conversion features turned on.

Figure 10: The digital up-conversion option "Shifted DC with Pre-modulation" has been applied to the signal from Figure 9. The signal is shifted up 62.5 MHz.

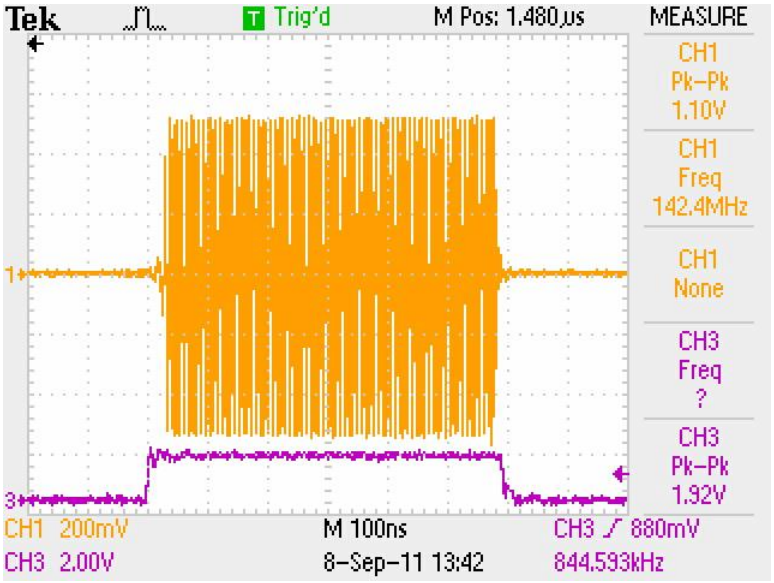


Figure 11: The digital up-conversion option "Fs/8" has been applied to the signal from Figure 9. The signal is shifted up 125 MHz.

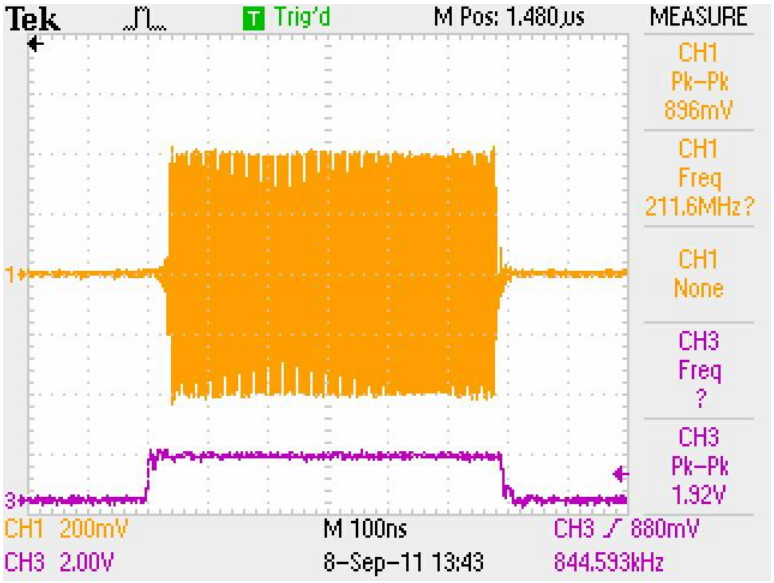


Figure 12: The digital up-conversion option "Shifted Fs/8 with Pre-modulation" has been applied to the signal from Figure 9. The signal is shifted up 187.5 MHz.

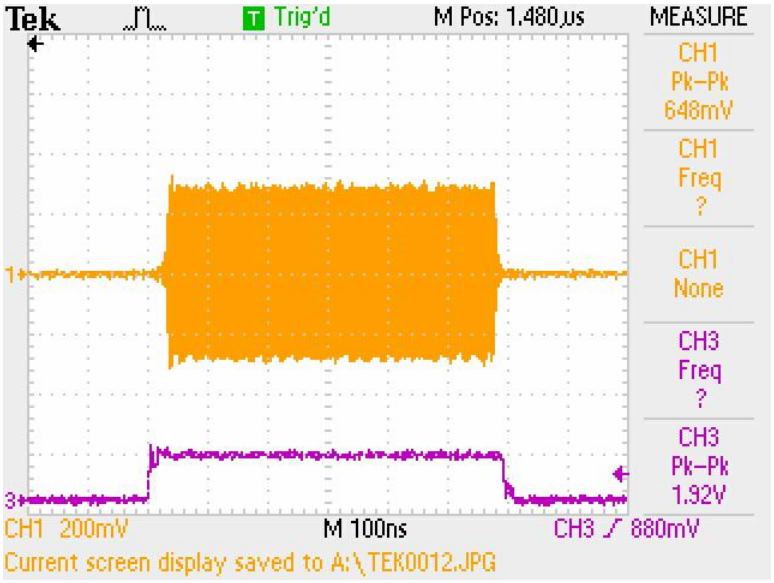


Figure 13: The digital up-conversion feature "Fs/4" has been applied to the signal from Figure 9. The signal has been shifted up 250 MHz.

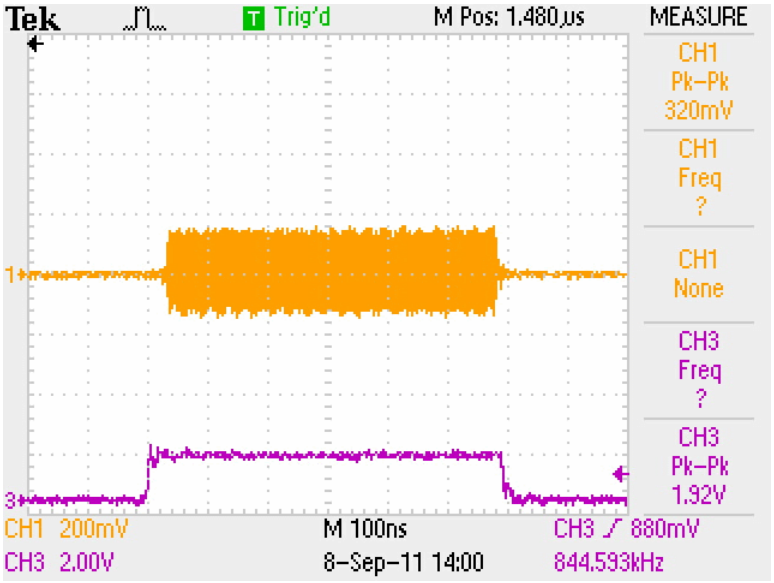


Figure 14: The digital up-conversion feature "Shifted Fs/4 with Pre-modulation" has been applied to the signal from Figure 9. The signal has been shifted up 312.5 MHz.

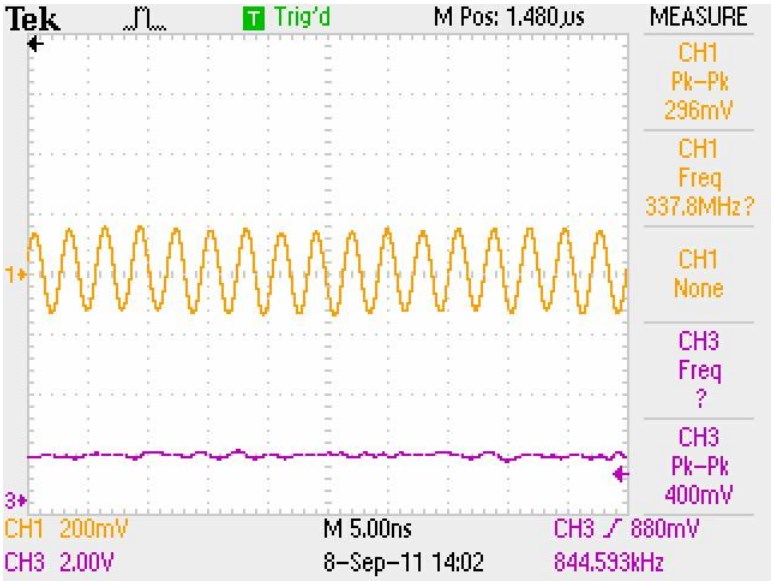


Figure 15: We have zoomed in on the signal in Figure 14 in order to show the integrity of the sinusoid. The signal is attenuated due to the oscilloscope.

Figure 16 below shows the agile frequency switching capability of the PulseBlasterDDS-IV.

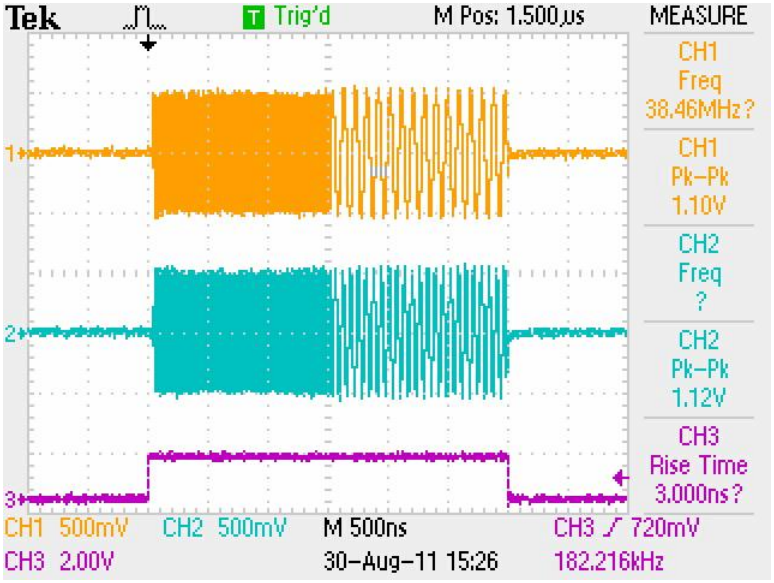


Figure 16: A demonstration of the agile frequency switching capabilities of the DDS-IV system. Channel 1 shows a transition from 40 MHz to 10 MHz and channel 2 shows a transition from 60 MHz to 15 MHz. The TTL pulse has been compensated to synchronize the TTL pulse with the RF pulse.

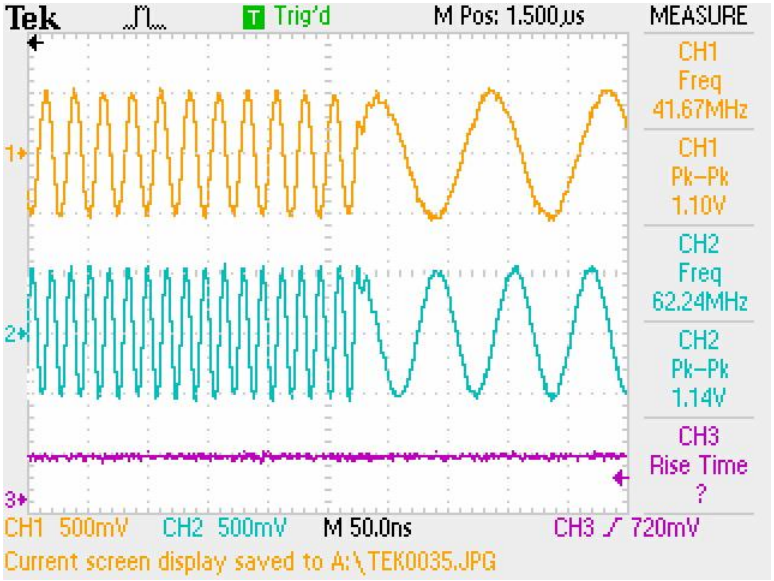


Figure 17: This screen capture shows the same signals from Figure 16 but we have zoomed in on the frequency transitions. You can see that the frequency switching is instantaneous.

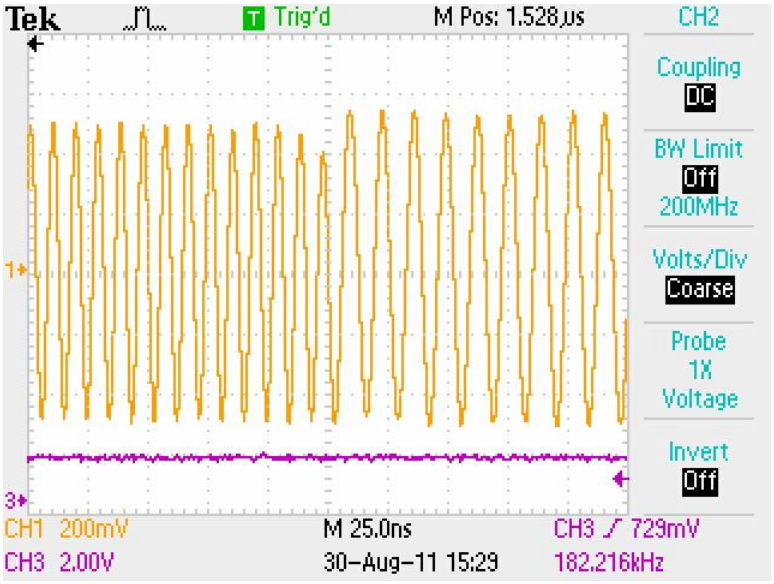


Figure 18: In this example we take a closer look at the signal on channel 1 from Figure 16. The AD9148 digital up-conversion features have been enabled to increase the frequency of the RF output. With AD9148 shifted DC coarse modulation and pre-modulation enabled the resulting signal transitions from 102.5 MHz to 72.5 MHz.

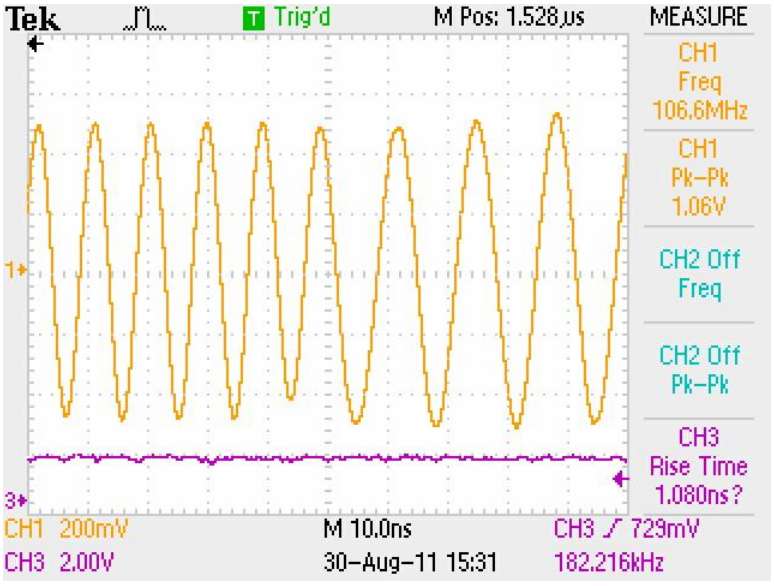


Figure 19: This image is shows the same signal from Figure 18 but we have zoomed in on the frequency transition. The transition is still instantaneous with AD9148 digital up-conversion enabled.

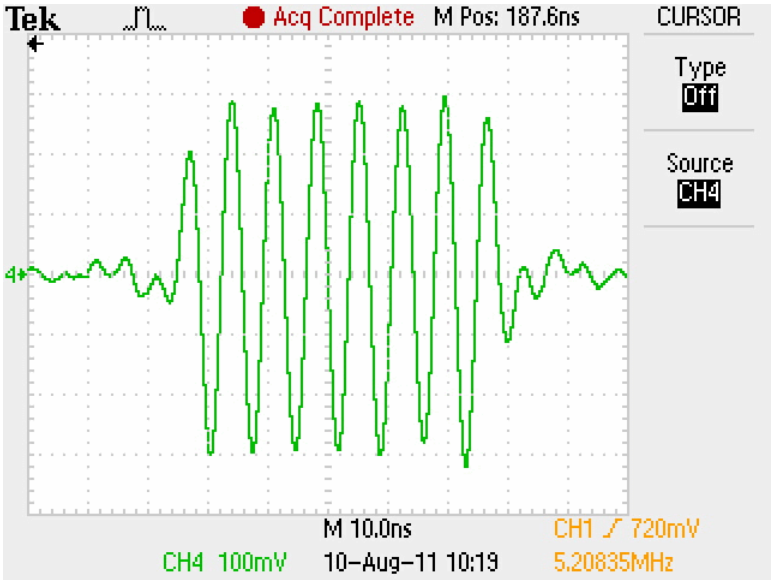


Figure 20: The above images shows a 60 ns RF pulse at 135 MHz.

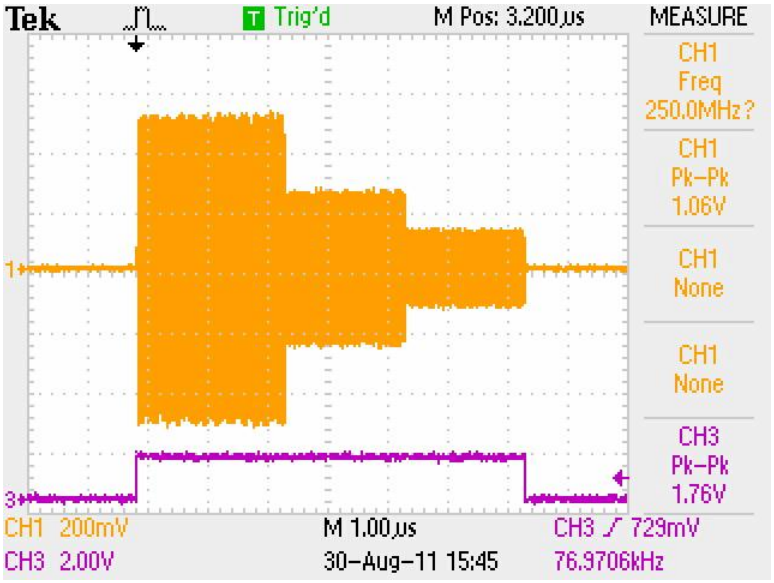


Figure 21: This image demonstrates the agile amplitude switching of the DDS-IV. A 75 MHz wave is shown switching from full scale to half scale and finally to quarter scale voltage output.

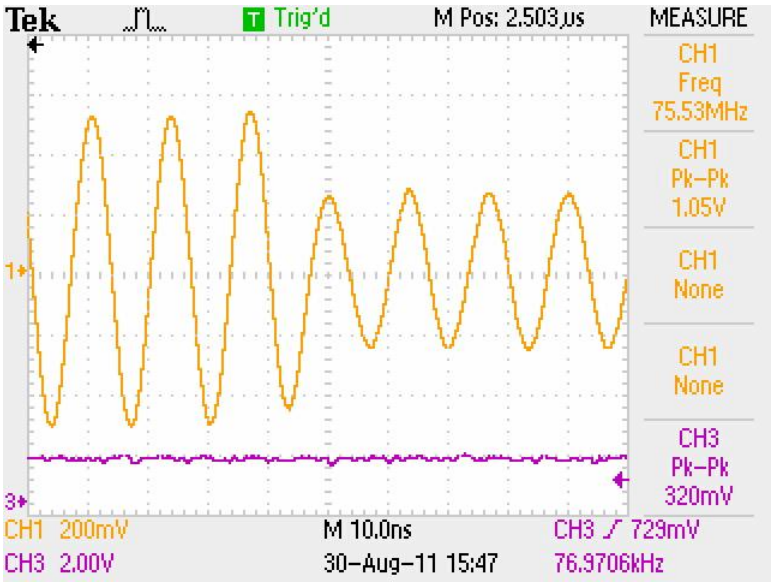


Figure 22: This image shows the same signal as in Figure 21 but we have zoomed in on the transition from full scale to half scale voltage.

The next image demonstrates the effects of the 0.5 us latency, as mentioned in Table 1. The frequency of the wave is in the base band at 10 MHz so that it can be seen easily. CH1 shows the TTL pulse while CH4 shows the RF pulse.

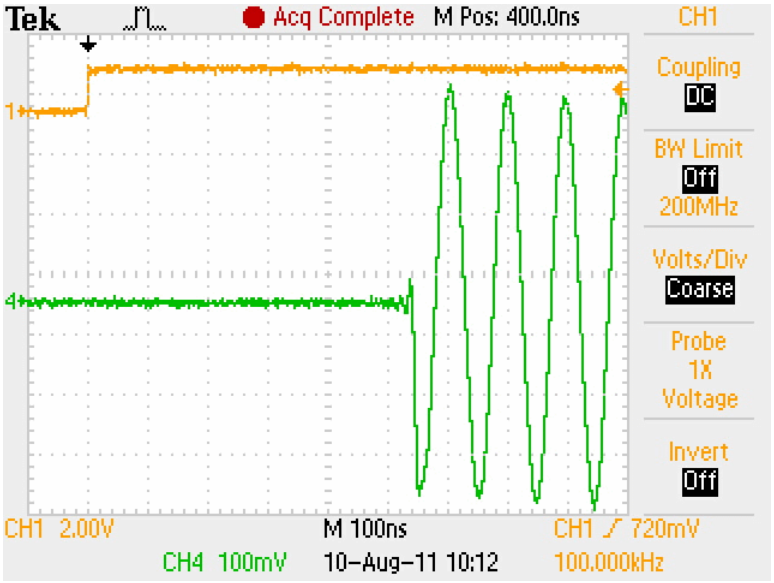


Figure 23: Demonstration of the 0.5 us latency. There is a 508 ns delay associated with the AD9148. At 1 GHz this equates to roughly 0.5 us.

The next figure shows how latency compensation can be used to synchronize the TTL pulse with the RF pulse. Once again, the RF wave is in the base band so that it can be seen easily.

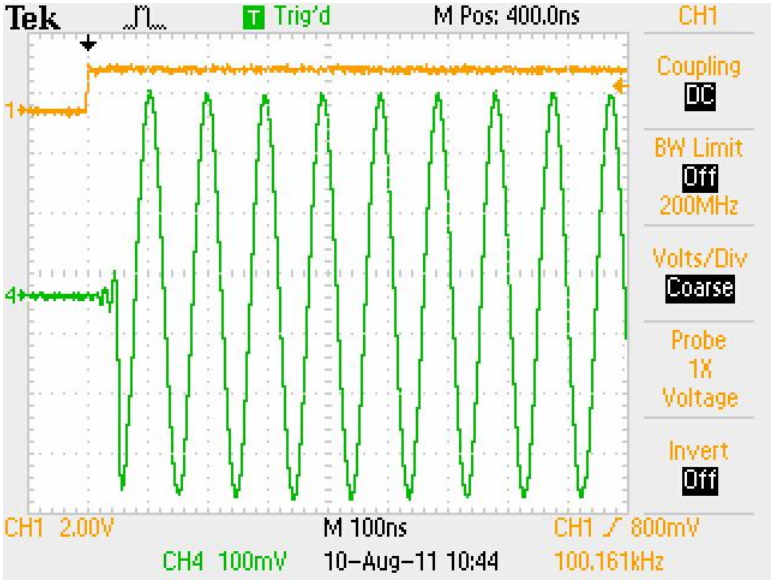


Figure 24: Latency compensation demonstration.

In the figure below, you can see the effects of the latency compensation using a 135 MHz output frequency. This was obtained using the same base band frequency as in Figure 24 above. In this case, the AD9148 digital up-conversion has shifted the frequency by 125 MHz (Fs/8 coarse modulation).

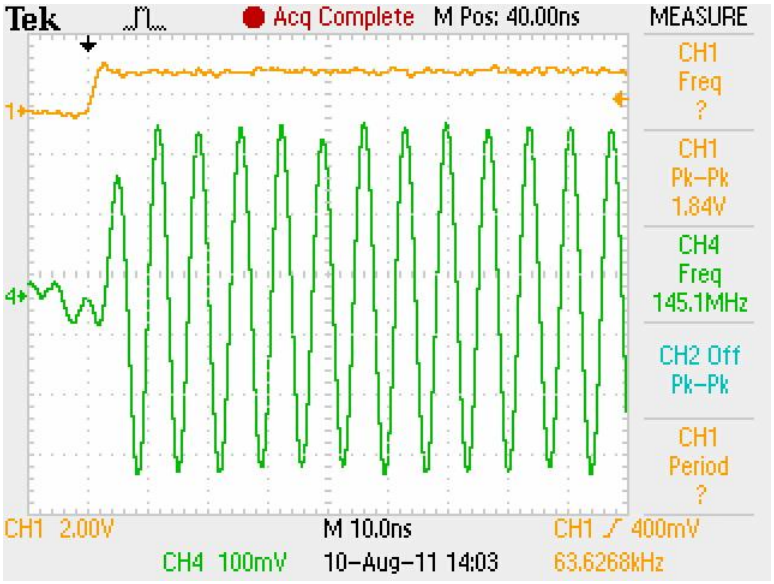
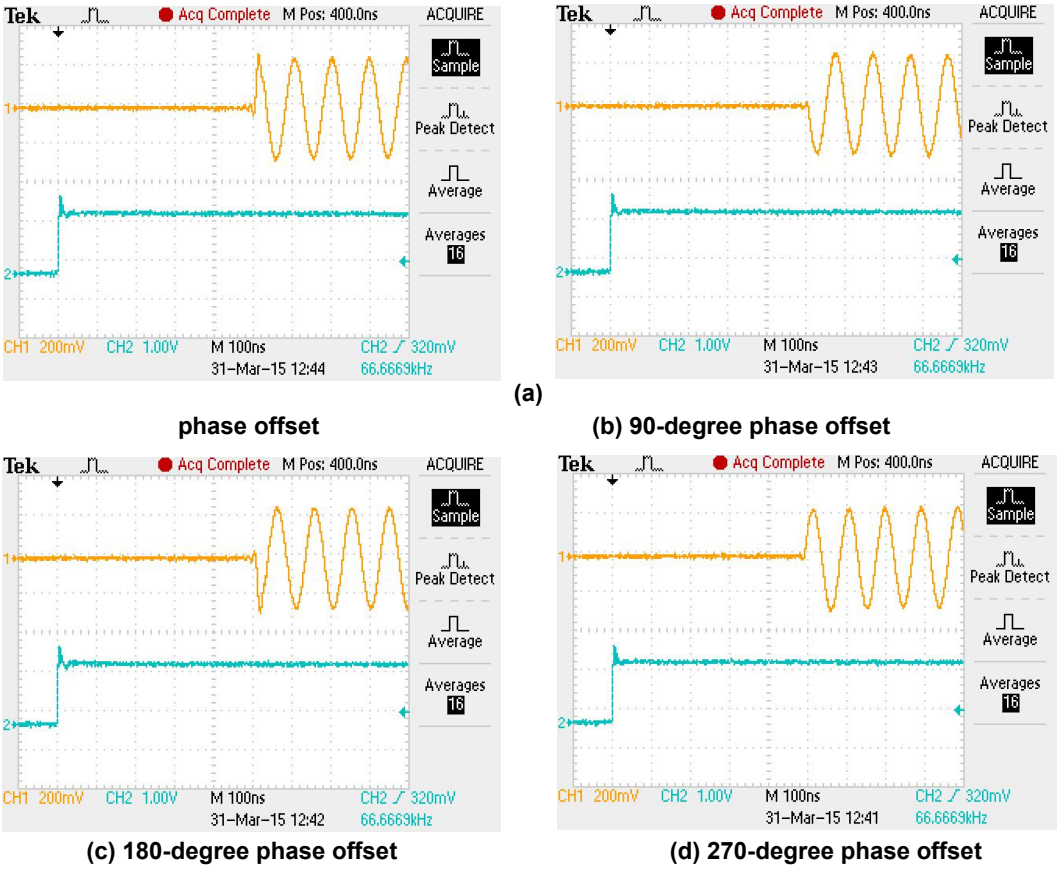


Figure 25: Latency compensation at 135 MHz.

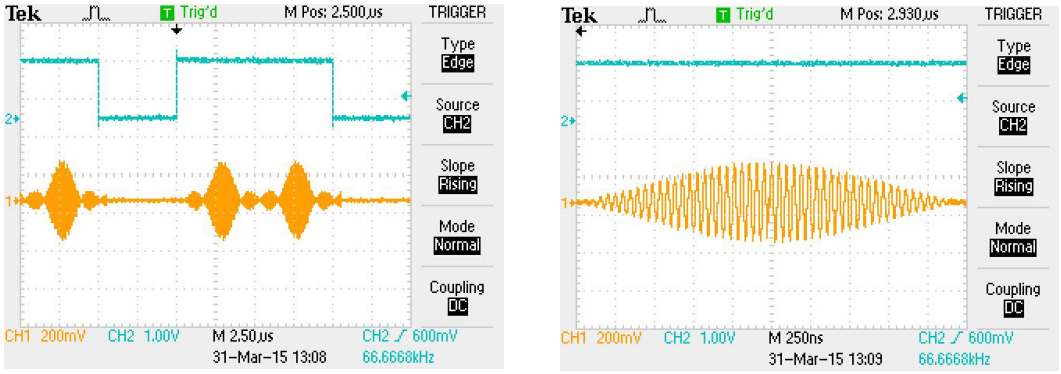


(a) No phase offset **(b) 90-degree phase offset**

(c) 180-degree phase offset **(d) 270-degree phase offset**

Figure 26: The PulseBlasterDDS-IV allows for selection of the initial RF-output phase using any of its 128 programmable-phase registers. The images above demonstrate selection between four

The figure below demonstrates the arbitrary-waveform feature of the DDS-IV. Each DDS's shape memory can be programmed independently with 1024 floating-point values describing the shape envelope.



(a) Sinc-modulated RF pulses (b) Single sinc lobe

Figure 27: Example of the arbitrary waveform-generation (AWG) feature of the DDS-IV. The RF pulse is modulated by a sinc envelope programmed by the user during DDS-IV configuration.

API Reference

The following tables provides an overview of the API functions provided by the PulseBlasterDDS-IV API. Unless otherwise specified, each function returns 0 on success.

Function Prototype	Function Description
int pbddsiv_init()	Initialize the currently selected PBDDS-IV device. MUST be called before any other functions.
int pbddsiv_close()	Close all open device handles.
int pbddsiv_count(**unsigned int** *count)	Count the number of devices in the system. • **count:** Variable to receive the number of DDS-IV systems detected.
int pbddsiv_set_defaults()	Set the default settings for the PulseBlasterDDS-IV including DAC registers.
int pbddsiv_get_firmware_id(**unsigned int** *fw)	Get the firmware ID of the currently selected device. • **fw:** Variable to receive the firmware ID of the selected DDS-IV system.
int pbddsiv_get_status(**unsigned int** *status)	Read the status register of the currently selected device. • **status:** variable to receive the status register value. ◦ Bit 0 – Stopped ◦ Bit 1 – Reset ◦ Bit 2 – Running ◦ Bit 3 – Waiting
int pbddsiv_trigger()	Trigger the PulseBlaster DDS-IV pulse program.
int pbddsiv_reset()	Stop DDS-IV execution and reset the program counter to the start of the program.
int pbddsiv_set_core_clock(**double** frequency)	Set core clock frequency. • **frequency**: frequency (in MHz).
int pbddsiv_set_interrupt_address(**unsigned int** interrupt_number, **unsigned int** address)	Set the interrupt address for a specified interrupt number. • **interrupt_number:** The interrupt vector number to set the interrupt vector for. • **address:** The address of the PulseBlaster instruction.
int pbddsiv_set_interrupt_source(**unsigned int** src)	Set the interrupt source for interrupt vectors. • **src:** Value representing the source ◦ 0 – hardware source (TTL inputs) ◦ 1 – software source (API calls)
int pbddsiv_trigger_sw_interrupt(**unsigned int** int)	Send a software interrupt trigger. • **int**: Trigger a software interrupt. Interrupt source must be set to software.

Table 6: PulseBlasterDDS-IV API Definition.

Function Prototype	Function Description
int pbddsiv_get_last_interrupt(unsigned int *int)	Get the interrupt vector of the last interrupt that was triggered.
int pbddsiv_set_flag_defaults(unsigned int flags[4])	Set the default flag values of all 128 TTL flags of the PulseBlasterDDS-IV (including internal TTL lines). • **flags**: Array of four 32-bit values representing all 128 TTL flags. ◦ Flag[0] – TTL bits 0 to 31 ◦ Flag[1] – TTL bits 32 to 63 ◦ Flag[2] – TTL bits 64 to 95 ◦ Flag[3] – TTL bits 96 to 127
int pbddsiv_write(unsigned int addr, unsigned int data)	Write a specific value to a specified register. • **addr:** Register address. • **data:** Data value to be written to the specified register.
int pbddsiv_read(unsigned int addr, unsigned int *data)	Read register at specified address. • **addr:** Register address. • **data:** Variable to receive read data into.
int pbddsiv_start_programming(unsigned int device, unsigned int device_id, unsigned int target)	Start programming a target device resource. Must be called before any of the pbddsiv_program_* functions. • Device: Device type (DDS or PULSEBLASTER) • device_id: Number of the device • Target: Target resource in the selected device to progam ◦ FREQUENCY_REGISTERS ◦ AMPLITUDE_REGISTERS ◦ PHASE_REGISTERS ◦ SHAPE_PERIOD_REGISTERS ◦ INSTRUCTION_MEMORY
int pbddsiv_stop_programming()	Stop programming the selected device.
int pbddsiv_program_frequency(double frequency)	Program the next frequency register with the specified frequency. Must be programming a DDS FREQUENCY_REGISTER. • **frequency:** Frequency value (in MHz)
int pbddsiv_program_phase(double phase)	Program the next phase register with the specified phase. Must be programming a DDS PHASE_REGISTER. • **phase:** Phase value (in degrees)
int pbddsiv_program_shape_period(double period)	Program the next frequency register with the specified frequency. Must be programming a DDS SHAPE_PERIOD_REGISTER • **period:** Shape period value (in seconds)
int pbddsiv_program_amplitude(double scale)	Program the next frequency register with the specified frequency. Must be programming a DDS AMPLITUDE_REGISTER. • **scale:** Scale value (0 to 1.0)

Table 7: PulseBlasterDDS-IV API Definition cont.

Function Prototype	Function Description
int pbddsiv_program_inst(unsigned int flags, const unsigned int oe[2], const unsigned int phase_reset[2], const unsigned int frequency[2], const unsigned int phase[2], const unsigned int amplitude[2], unsigned int inst, unsigned int inst_data, double time_sec)	Program an instruction without shape feature to program memory. • **Flags:** TTL flag values. • **Oe:** Array specifying the output-enable for each DDS. • **phase_reset:** Array specifying the phase-reset bit for each DDS. • **Frequency:** Frequency register select for each DDS. • **Phase:** Phase register select for each DDS. • **Ampltiude:** Amplitude register select for each DDS. • **Instruction:** Instruction opcode. • **inst_data:** Data argument for the instruction. • **time_sec:** Instruction duration (in seconds) **Returns** the instruction address.
int pbddsiv_program_inst_shape(unsigned int flags, const unsigned int oe[2], const unsigned int phase_reset[2], const unsigned int frequency[2], const unsigned int phase[2], const unsigned int amplitude[2], const unsigned int shape_period[2], unsigned int inst, unsigned int inst_data, double time_sec)	Program an instruction with shape feature to program memory. • **Flags:** TTL flag values. • **Oe:** Array specifying the output-enable for each DDS. • **phase_reset:** Array specifying the phase-reset bit for each DDS. • **Frequency:** Frequency register select for each DDS. • **Phase:** Phase register select for each DDS. • **Ampltiude:** Amplitude register select for each DDS. • **Shape_period:** Shape period register select. Value of 7 disables shape. • **Instruction:** Instruction op code. • **inst_data:** Data argument for the instruction. • **time_sec:** Instruction duration (in seconds) **Returns** the instruction address.
int pbddsiv_configure_interpolation_mode(const struct interpolation_settings_t *settings);	Configure the interpolation mode settings. • **Settings:** Interpolation mode settings structure with the following fields: ◦ Mode – Interpolation mode setting. ◦ Factor – Interpolation filters enabled.
int pbddsiv_log_set(const char *filename)	Set output log name. **filename**: target file.
int pbddsiv_log_enable()	Enable logging to log file.
int pbddsiv_log_disable()	Disable logging.

Table 8: PulseBlasterDDS-IV API Definition cont.

IV. Connecting to the PulseBlasterDDS-IV-1000

Front Panel Connector Locations

The PulseBlasterDDS-IV-1000 system is housed in a 3U rack mount enclosure. The front panel contains all of the ports needed to interact with the DDS-IV-1000. There are four main elements on the front panel of the enclosure: the BNC connectors, a female DE9 (sometimes called a DB9) connector, a USB Type-B port and a DC power supply on/off switch. The back panel contains the AC power cord connection and an AC on/off switch.

The BNC connectors provide the means of interfacing with the input and output signals of the PulseBlasterDDS-IV-1000. The RF and digital outputs, 10 MHz reference input clock, and the trigger input are all accessed via BNC connectors. The DE9 connector is used to interface with the interrupt input. The USB port is used to communicate with the host PC. The front panel switch is used to turn the DDS-IV-1000 on and off when the back panel AC switch is also on.

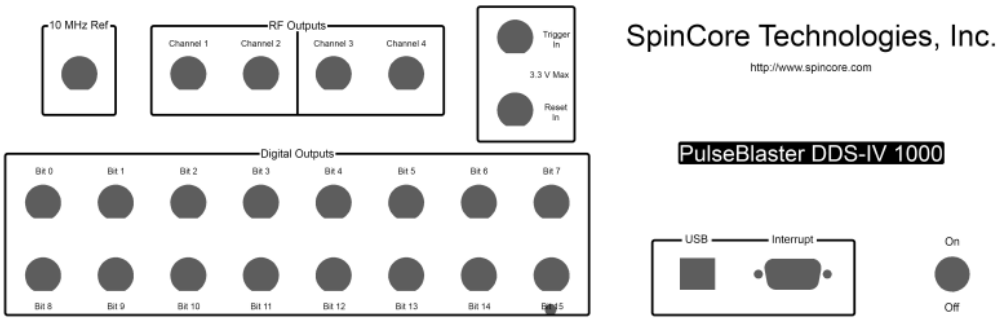


Figure 28: Front panel connector locations.

RF Outputs

The PulseBlasterDDS-IV-1000 has four RF outputs, all of which are accessed through female BNC connectors. Figure 29 presents the numbering scheme for these outputs. Please note that independent waveforms are generated by outputs 1 and 3; output 2 is the same as output 1 with a 90 degree phase offset. Similarly, output 4 is a 90 degree phase shifted version of output 3. The source impedance is 50 Ohm.

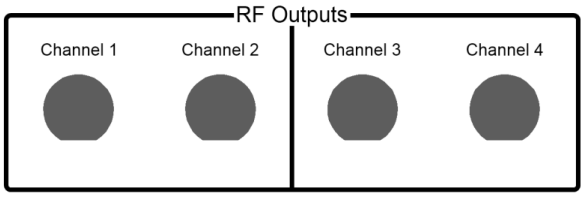


Figure 29: RF BNC output jacks.

Digital Outputs

The 16 digital outputs of the PulseBlasterDDS-IV-1000 are accessible through female BNC connectors on the front panel. The numbering of the bits is presented in Figure 30. The required minimum logical high TTL level of 2.5 V is attained when the load impedance is 150 Ω.

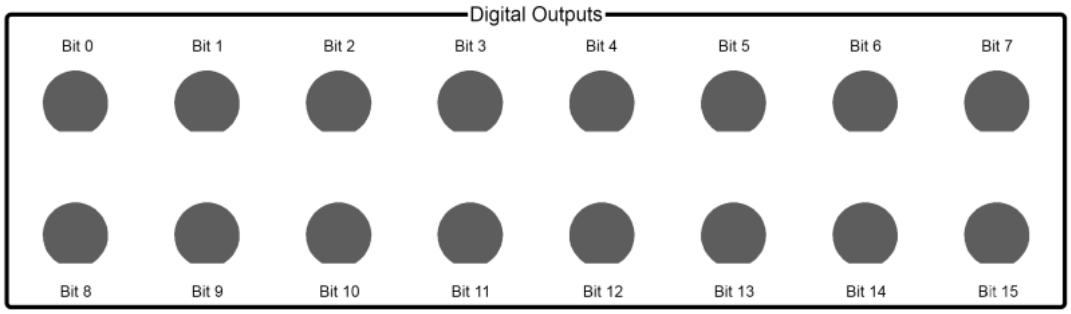


Figure 30: Numerical order of the digital outputs.

10 MHz Reference Clock Input

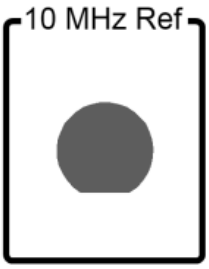


Figure 31: 10 MHz reference clock input label

The PulseBlasterDDS-IV-1000 features a 10 MHz reference clock input that can be used to synchronize the system to an external 10 MHz, square, 50% duty cycle, 3.3V-level clock signal. This signal is sent through a PLL (phase locked loop), which creates the higher clock frequencies required by the PulseBlaster core, DDS cores, and the AD9148 DAC.

The external 10 MHz reference can be connected before the board is powered on or during operation. The PulseBlasterDDS-IV-1000 will automatically begin using the external reference once it detects that it is present. Please note, however, that a power reset is required to switch back to the internal oscillator after removing the external clock source.

Trigger and Reset Input

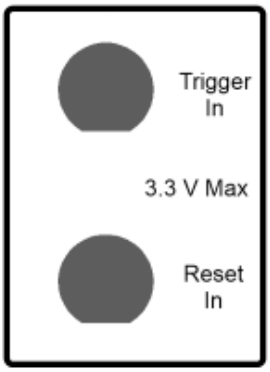


Figure 32: HW trigger and reset input labels

The trigger input is used to start a pulse program that has been loaded into the instruction memory of the PulseBlasterDDS-IV-1000. This input is used in tandem with the DE9 interrupt port, which can be used to configure the instruction memory address that will be read from first when the trigger input is activated.

Doing so produces the same results as the pbddsiv_trigger() API function. The input is normally pulled high and the maximum voltage that can be input to the connection is 3.3V.

The trigger input is active-low, meaning that inputting a low voltage level will cause the PBDDS-IV to be triggered. The board is not triggered on a rising or falling edge. Please note that as long as the input to the trigger is a logical-low voltage level, the board will be triggering.

The reset input is used to reset a pulse program to the beginning (the program counter is reset to 0). The reset input is active-low, meaning that inputting a low voltage level will reset the pulse program running on the PBDDS-IV. This is the same behavior as the pbddsiv_reset() API function. The input is normally pulled high and the maximum voltage that can be input to the connection is 3.3V.

Interrupt Input

The DE9 interrupt connector, shown below in Figure 33, is used to control the 256 available interrupts on the PulseBlasterDDS-IV-1000. Please note that in order to use this hardware interrupt control, you must enable hardware interrupts as described in the API reference section.

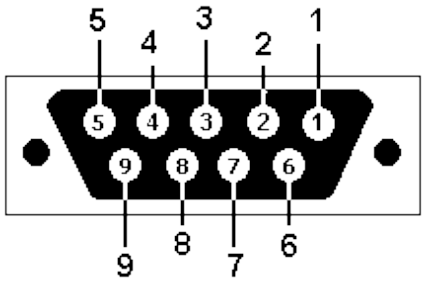


Figure 33: Pin out of the DE9 Interrupt connector

The pin out for this connector is shown below in Table 9. Be sure to match this pin out exactly if you are creating your own DE9 interrupt cable. Please note that every signal pin is active-high. The maximum voltage level that can be asserted on a signal pin is 3.3 V, and the minimum voltage level is 0 V. As an example, asserting a high logic level on interrupt bits 0, 3, and 4 while hardware interrupts are enabled in the API has the same effect as the pbddsiv_trigger_sw_interrupt(0x19) function call does when software interrupts are enabled. Please note that all the signal pins are weakly pulled high, so if no input is connected then the highest interrupt (0x255) is always asserted in hardware.

Pin Number	Pin Function
1	Interrupt bit 0
2	Interrupt bit 1
3	Interrupt bit 2
4	Interrupt bit 3
5	Interrupt bit 4
6	Interrupt bit 5
7	Interrupt bit 6
8	Interrupt bit 7
9	GND

Table 9: Interrupt Input Pin-out

If using a high input impedance oscilloscope to monitor the PulseBlasterDDS-IV-1000, place a resistor that matches the characteristic impedance of the transmission line in parallel with the coaxial transmission line at the oscilloscope input. (e.g., a 50 Ω resistor with a 50 Ω transmission line, see Figures 34 and 35 below). When using an oscilloscope with an adjustable bandwidth, set the bandwidth to as large as possible. Failure to do so may yield inaccurate readouts on the oscilloscope.

Figure 34: Left - BNC T-Adapter and Right - BNC 50 Ohm resistor

Figure 35: BNC T-Adapter on the oscilloscope with coaxial transmission line connected on the left and BNC 50 Ohm resistor connected on the right, to terminate the line.

V. Contact Information

SpinCore Technologies, Inc.
4631 NW 53rd Avenue, SUITE 103
Gainesville, Florida, 32653
USA

Telephone: +1-352-271-7383
Fax: +1-352-371-8679

Website: http://www.spincore.com
Web Contact Form: http://www.spincore.com/contact.shtml

VI. Document Information

A detailed revision history is available by contacting SpinCore Technologies, Inc at the address above.

If you have any feedback or questions, do not hesitate to contact us. We look forward to hearing from each and every one of our customers!

From everyone at SpinCore, we appreciate you reading this manual to learn about your product.