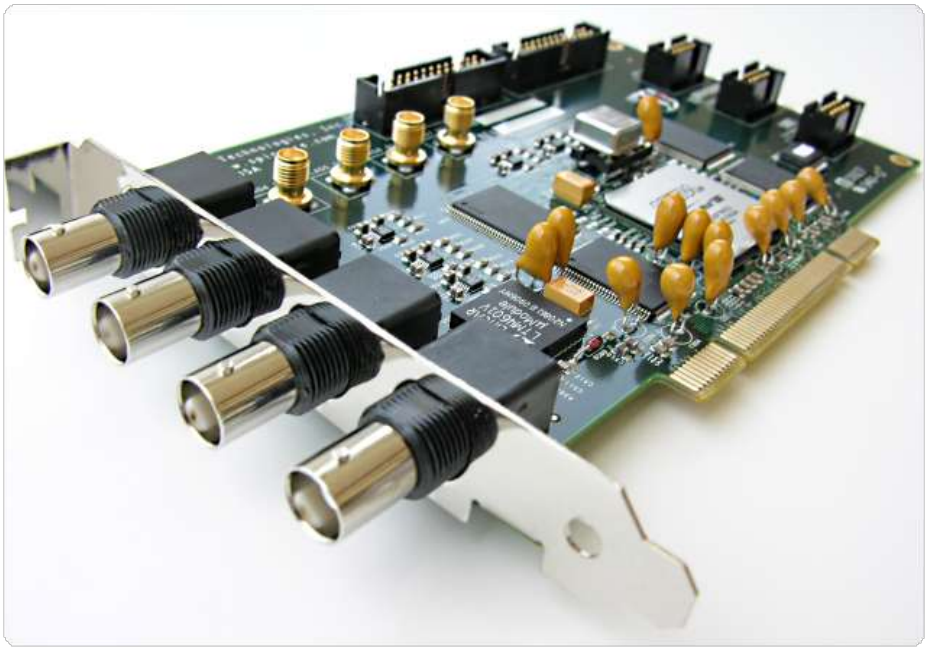
SpinCore Technologies, Inc.

PulseBlasterESR QuadCore 8M™

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 SpinCore PulseBlasterESR QuadCore 8M is a quad-core PulseBlaster design implemented on a new series of PulseBlasterESR PCI boards with up to 2097152 instruction per core . The quad-core design uses four of SpinCore's proprietary PulseBlaster processor cores on a single chip. This new design allows the user to program and run completely independent programs on each core, in parallel, while maintaining precise timing synchronization between each core.

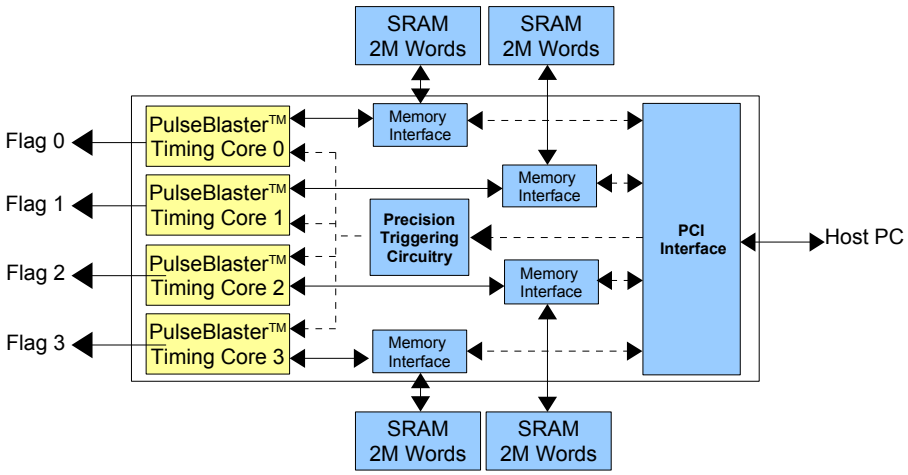


Diagram 1 : SpinCore PulseBlasterESR Quad Core Design Architecture

All cores are driven by the same single clock source at 500 MHz. They are synchronized to start at the same time and run unique pulse programs/sequences concurrently. At 500 MHz, the available resolution of each pulse/delay/interval is 2.00 ns (one clock cycle), the minimum pulse/delay/interval length is 18 clock cycles, or 36 ns, and the maximum pulse/delay/interval length is 2^{29} clock cycles (~1.07 seconds). Each core has 2M (2097152) memory words available for writing pulse programs, i.e., there can be up to 2097152 lines in your pulse program per core.

The basic architecture of the individual PulseBlaster processor core is described in multiple documents, including the manuals for PulseBlaster and PulseBlasterESR boards, available on-line at the SpinCore's website www.spincore.com. (Note that the PulseBlasterESR QuadCore uses simplified PulseBlaster Cores that allow for 'continue' and 'stop' operations only.)

Programming Paradigm

Each core can be individually programmed with an arbitrary sequence of intervals. Each interval can be of unique length, and up to 2M intervals can be accommodated per sequence. Since each interval can be a pulse or a delay, the programming of each core involves the loading of two basic parameters per interval: the output state (logical 0 or 1), and the duration of the state (in nanoseconds, microseconds, milliseconds).

Each core can be independently programmed and triggered. The low-level interaction is accomplished through a dedicated Application Programming Interface (API) package called the MultiCore SpinAPI, available for download on SpinCore's website: www.spincore.com. Virtually any higher-level application package (Matlab, LabVIEW etc.) can interact with the board through the provided SpinAPI functions.

Specifications

Parameter		***Min***	***Typ***	***Max***	***Units***
Digital Outputs	Number of Digital Outputs		4		
	Logical 1 Output Voltage		2.00(1)	3.3	V
	Logical 0 Output Voltage		0		V
	Output Drive Current			40	mA
	Rise/Fall Time			<1	ns
Digital Inputs (HW_Trig, HW_Reset)	Logical 1 Input Voltage	1.7		3.3	V
	Logical 0 Input Voltage	-0.5		0.7	V
Pulse Program	Number of Cores	1	4	> 4	
	Number of Instruction (per Core)			2097152	instructions
	Pulse Resolution		2.00		ns
	Pulse Length	36 ns		~1.07 s	
	Supported Operations	CONTINUE, STOP			

Table 1: Specifications of the PulseBlasterESR QuadCore Design

(1)This value is with a 50Ω terminating resistance.

II. Installing and Using Your PulseBlasterESR QuadCore

Installation

To install the board you must complete the following three steps:

1) Install the latest MultiCore SpinAPI package, available at http://spincore.com/support/spinapi/
   - The QuadCore PulseBlasterESR board requires a specialized version of SpinAPI called the MultiCore SpinAPI.
2) Shut down computer, insert the PulseBlasterESR MultiCore 8M card, and fasten the PC bracket.
   - Your system should detect the board as a "PulseBlaster Multicore" device.
3) Power up and follow the installation prompts.

Now you are ready to run the test programs provided in the SpinAPI package.

Note: To compile and run your own C programs, you may want to download the *SpinAPI Tools* package that contains a pre-configured compiler; the *SpinAPI Tools* package is also available for download at the URL above.

General API Programming Information

Seven test programs (executables and their C source files) are available for testing. Assuming the default installation, the test programs will be available on the computer at the following location: Start → All Programs → SpinAPI → Examples → QuadCore Examples (the default installation location is: "C:\SpinCore\SpinAPI\Examples\QuadCore Examples"). The .c files can be modified and recompiled to create custom test programs.

The SpinAPI programming paradigm is simple:

1. Include the "spinapi.h" in your C-file and link your executable to the SpinAPI library.
2. Initialize the API by calling the function *pb_init()*. This function must be called and return successfully in order for the API to function properly.
3. If there is more than one board installed in your system, select the correct board by calling *pb_select_board(board_number).*
4. Tell the API the boards internal operating clock frequency. This can be done by calling *pb_set_clock(clock_freq)* with the appropriate internal operating frequency (500 MHz for the PulseBlasterESR QuadCore 8M.)
5. Start programming a PulseProgram memory device. This can be accomplished by calling *pb_start_programming(device).* The available devices on the PulseBlasterESR QuadCore are the *CORE0_MEM* (core 0's PulseProgram memory), *CORE1_MEM* (core 1's PulseProgram memory) *CORE2_MEM* (core 2's PulseProgram memory) and *CORE3_MEM* (core 3's PulseProgram memory)
6. Begin programming a PulseProgram sequence. A CONTINUE instruction can be programmed by calling *pb_inst(flag, time_ns)*. A STOP instruction can be programmed by calling *pb_inst_stop()*.
   Note: The last instruction in the PulseProgram must be a STOP instruction, or the program will loop infinitely.
7. Stop programming the selected device. This can be accomplished by calling *pb_stop_programming()*.
8. Select which core's are enabled by calling *pb_core_select(core_mask)*. Each bit of the core mask corresponds to that core being enabled (i.e. bit 0 corresponds to core 0.)
9. Trigger the selected cores by using *pb_start()*, or reset the board with *pb_stop()*.
10. Close the API by calling pb_close().

For more information on using the QuadCore SpinAPI, see the "SpinAPI Reference Manual.pdf" found in the SpinAPI directory.

Triggering PulseProgram Execution

The PulseBlasterESR QuadCore can be triggered in two ways, either by software trigger or hardware trigger. The software trigger is initiated by sending a command from the host PC via the *pb_start()* function. Since the PulseBlasterESR QuadCore boards are typically used with non real-time operating systems, the exact time between issuing a software trigger and the board acting on that trigger cannot be precisely specified. For precision control, the pulse program can also be triggered by setting the HW_Trigger pin to a logical 0. This will cause the pulse program to be triggered with a latency of seven clock cycles. For more information on the hardware trigger, see Appendix B: Hardware Reset/Triggering.

Stopping PulseProgram Execution

The PulseBlasterESR QuadCore can be stopped by using either the software or hardware reset. The software reset is initiated by sending a command from the host PC via the *pb_stop()* function. Since the PulseBlasterESR QuadCore boards are typically used with non real-time operating systems, the exact time between issuing a software reset and the board acting on that reset cannot be precisely specified. For precision control, the pulse program can also be reset by setting the HW_Reset pin to a logical 0. This will cause the pulse program stop and the program counter to reset within one clock cycle. For more information on the hardware reset, see Appendix B: Hardware Reset/Triggering.

III. Test Programs

Example Programs

Seven test programs have been packaged with the SpinAPI driver suite to illustrate the basic features and functionality of the PulseBlasterESR QuadCore design. All programs can be found at: Start → All Programs → SpinAPI → Examples → QuadCore Examples (the default installation location is: "C:\SpinCore\SpinAPI\Examples\QuadCore Examples").

Example 1

The first test program, pb_quadcore_example1.c demonstrates that all cores (channels) can generate identical pulses that are precisely synchronized.

An excerpt from the code to program the first two cores is as follows:

```
//********* Program Core0 ******************-/
pb_start_programming (CORE0_MEM);
  pb_inst(1, 50.0 * ns);
  pb_inst(0, 50.0 * ns);
  pb_inst(1, 50.0 * ns);
  pb_inst(0, 50.0 * ns);
  pb_inst(1, 50.0 * ns);
  pb_inst(0, 50.0 * ns);
  pb_inst(1, 50.0 * ns);
  pb_inst_stop();
  instruction_count += pb_stop_programming ();

//******** Program Core1 ************************-/
  pb_start_programming (CORE1_MEM);
  pb_inst(1, 50.0 * ns);
  pb_inst(0, 50.0 * ns);
  pb_inst(1, 50.0 * ns);
  pb_inst(0, 50.0 * ns);
  pb_inst(1, 50.0 * ns);
  pb_inst(0, 50.0 * ns);
  pb_inst(1, 50.0 * ns);
  pb_inst_stop();
  instruction_count += pb_stop_programming ();
```

PulseProgram 1: Excerpt from pb_Quadcore_example1.c

In Example 1, all cores are programmed with identical content. Later in the program, all cores are triggered at the same time using *pb_start()*. The resulting output should be four identical 50.0 ns pulses on BNC0 through BNC3.

NOTE: When attaching an oscilloscope to the board to observe the pulses, care should be taken to use cables of the same type and length for each channel, as skew can be induced due to propagation delays. Conversely, any inherent variations in on-chip propagation delays can be compensated by appropriate variations in cable length.

Example 2

The second example is composed of two separate files: pb_quadcore_example2a.c and pb_quadcore_example2b.c. Each of these examples are used to program one core with a pulse sequence (50 ns on/50 ns off) that will occupy the entire 2M instruction memory, and one core with a single pulse that is on for the equivalent pulse program time. This allows for verification that the core's full memory is working properly and that there are no timing inaccuracies in any of the 2M instructions.

•pb_quadcore_example2a.c : Tests core0's entire Pulse Program memory, and uses core1,2 and 3 to generate the equivalent time pulse.

•pb_quadcore_example2b.c : Tests core1's entire Pulse Program memory, and uses core 0,2, and 3 to generate the equivalent time pulse.

•pb_quadcore_example2c.c : Tests core2's entire Pulse Program memory, and uses core 0,1, and 3 to generate the equivalent time pulse.

•pb_quadcore_example2d.c : Tests core3's entire Pulse Program memory, and uses core 0,1, and 2 to generate the equivalent time pulse.

An excerpt from the code to program the first two cores is as follows:

```
//******** Program Core0 ******************-/
    pb_start_programming (CORE0_MEM);
    i=0;
    while(i<FULL_MEMORY_SIZE-2) {
                    pb_inst(1, 50.0*ns);
                    pb_inst(0, 50.0*ns);
                    i+=2;
    }
    pb_inst(1, 50.0*ns);
    pb_inst_stop();
    instruction_count_count += pb_stop_programming ();

//******** Program Core1 ******************-/
    pb_start_programming (CORE1_MEM);
    pb_inst(1, (FULL_MEMORY_SIZE*50.0)*ns);
    pb_inst_stop();
    instruction_count_count += pb_stop_programming ();
```

PulseProgram 2: Excerpt from pb_quadcore_example2a.c

Later in the program, all all cores are triggered at the same time using *pb_start()*.

NOTE: When attaching an oscilloscope to the board to observe the pulses, care should be taken to use cables of the same type and length for each channel, as skew can be induced due to propagation delays. Conversely, any inherent variations in on-chip propagation delays can be compensated by appropriate variations in cable length.

Example 3

The third test program, pb_quadcore_example3.c, demonstrates creating Pulse Programs with an adjustable offset between the first pulse as low as 2.00 ns. When the program is run, the user will be prompted for the offset between flag 0 and flags 1 through 3. This must be a multiple of 2.00 ns.

An excerpt from the code to program the (first two) cores is as follows:

```
//********* Program Core0 ******************-/
   pb_start_programming (CORE0_MEM);
   pb_inst(0, 36.0*ns + offset);
                    pb_inst(1, 50.0*ns);
                    pb_inst(0, 50.0*ns);
                    pb_inst(1, 50.0*ns);
                    pb_inst_stop();
   instruction_count += pb_stop_programming ();

//********* Program Core1 ******************-/
   pb_start_programming (CORE1_MEM);
   pb_inst(0, 36.0*ns);
                    pb_inst(1, 50.0*ns);
                    pb_inst(0, 50.0*ns);
                    pb_inst(1, 50.0*ns);
                    pb_inst_stop();
   instruction_count += pb_stop_programming ();
```

PulseProgram 3: Excerpt from pb_quadcore_example3.c

Later in the program, all cores are triggered at the same time using *pb_start().* When the board is triggered, there should be two 50.0 ns pulses on BNC0 through BNC3, with the pulses on BNC0 starting the specified offset after the pulses on BNC1 through BNC3.

Note that the offset of down to 2.00 ns is created by having an initial instruction with at least the minimum pulse length.

NOTE: When attaching an oscilloscope to the board to observe the pulses, care should be taken to use cables of the same type and length for each channel, as skew can be induced due to propagation delays. Conversely, any inherent variations in on-chip propagation delays can be compensated by appropriate variations in cable length.

Example 4

The fourth test program, pb_quadcore_example4.c, demonstrates the stability of the counters in each PulseBlaster cores by generating an increasingly long pulse, starting at the minimum pulse length and increasing by 2.00 ns every 100 ms. This is accomplished by using a continuous loop within the C-program. To exit the program, enter CTRL-C or close the prompt window.

An excerpt from the code to program the cores is as follows:

```
while(1) {
//******** Program Core0 ************************-/
                    pb_start_programming (CORE0_MEM);
                    pb_inst(1, (36.00 + i*2.00)*ns);
                    pb_inst_stop();
                    pb_stop_programming ();

//********* Program Core1 ******************-/
                    pb_start_programming (CORE1_MEM);
                    pb_inst(1, (36.00 + i*2.00)*ns);
                    pb_inst_stop();
                    pb_stop_programming ();

                    i++;

                    pb_start();
                    Sleep(100);
}
```

PulseProgram 4: Excerpt from pb_quadcore_example4.c

As shown above, the increasingly long pulse is generated by reprogramming the board memory with a new instruction with an increasing pulse length every loop.

NOTE: When attaching an oscilloscope to the board to observe the pulses, care should be taken to use cables of the same type and length for each channel, as skew can be induced due to propagation delays. Conversely, any inherent variations in on-chip propagation delays can be compensated by appropriate variations in cable length.

Example 5

The fifth test program, pb_quadcore_example5.c, explores a range of offsets between cores. The program starts with no offset, then increases the offset between cores at different rates. This demonstrates the the timing versatility of the PulseBlasterESR QuadCore 8M.

An excerpt from the code to program the cores is as follows:

```
while(1) {
//******** Program Core0 ******************-/
                    pb_start_programming (CORE0_MEM);
                    pb_inst(0, (36.00 + i*2.00)*ns);
                    pb_inst(1, 50*ns);
                    pb_inst_stop();
                    pb_stop_programming ();

//******** Program Core1 ******************-/
                    pb_start_programming (CORE1_MEM);
                    pb_inst(0, (36.00+i*3.00)*ns);
                    pb_inst(1, 50*ns);
                    pb_inst_stop();
                    pb_stop_programming ();

///******** Program Core2 ******************-/
                    pb_start_programming (CORE2_MEM);
                    pb_inst(0, (36.00+i*1.00)*ns);
                    pb_inst(1, 50*ns);
                    pb_inst_stop();
                    pb_stop_programming ();

//******** Program Core3 ******************-/
                    pb_start_programming (CORE3_MEM);
                    pb_inst(0, 36.00*ns);
                    pb_inst(1, 50*ns);
                    pb_inst_stop();
                    pb_stop_programming ();

                     i+= dir;

                     if(i==100){
                         dir = -1;
                       }
                    else if(i==0){
                        dir = 1;
                    }

                    pb_start();
                    Sleep(25);
                    pb_stop();
}
```

PulseProgram 5: Excerpt from pb_quadcore_example5.c

NOTE: When attaching an oscilloscope to the board to observe the pulses, care should be taken to use cables of the same type and length for each channel, as skew can be induced due to propagation delays. Conversely, any inherent variations in on-chip propagation delays can be compensated by appropriate variations in cable length.

Example 6

The sixth example, pb_quadcore_mem_test.c, is a memory verification tool. This program is used primarily to test the memory read/write speeds and verify that the memory is working properly. When the program is run, it will write a random patterns to each memory address, and then read back the result, verifying that the memory is working properly and displaying the read and write speeds that were obtained.

If a large number of errors are occurring, it could mean that the memory is malfunctioning. Please note that the PulseBlasterESR board must be stopped before performing a memory test.

IV. Appendices

Appendix A: Connectors

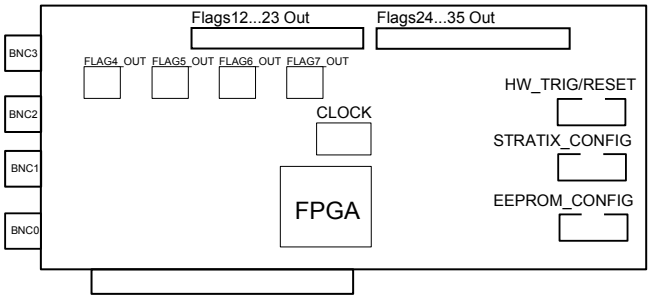


Diagram 2: PulseBlaster QuadCore Connector Layout

Note: For the PulseBlasterESR QuadCore design, only flags 0 through 3 are used (BNC0 through BNC3 respectively.)

The shrouded IDC connectors labeled Flag 12..23and Flag 24.. 35 can also be accessed using an SP32 board (Figure 1) which allows the use of MMCX cables. This enables the individual bits of the PulseBlaster to be more easily accessed. Pin 1 on the MMCX adapter board can identified with a square pin.

HW_TRIG/RESET Header

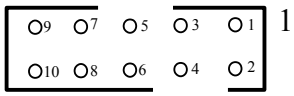


Diagram 3: HW_TRIG/RESET Header

Pin Number	Function
1	GND
2	INT0
3	GND
4	INT1
5	GND
6	INT2
7	GND
8	HW_Reset
9	GND
10	HW_Trigger

Table 2: HW_TRIG/RESET Header Pin-out

CLOCK Header

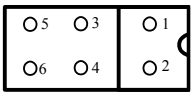


Diagram 4: CLOCK Header

Pin Number	Function
1	No Connect
2	VCC(3.3V)
3	No Connect
4	VCC(3.3V)
5	GND
6	CLOCK_INPUT

Table 3: CLOCK Header Pin-out

Appendix B: Hardware Triggering/Reset

In order to provide precise and predictable triggering and reset latencies, the PulseBlasterESR QuadCore provides an external hardware reset and hardware trigger inputs. These inputs can be found on the HW_TRIG/RESET header.

The hardware trigger (HW_Trigger) and reset pins (HW_Reset) are pulled internally to a logical high level via a 10kΩ resistor. In order for a hardware trigger or reset to be detected, the appropriate pin must be driven low via an external source for at least one clock period (2.00 ns.) Once a hardware trigger has been detected, the enabled PulseBlaster Cores will start executing after seven clock cycles. A hardware reset has a latency of one clock cycle.

In cases where it is necessary to have precise control of the triggering and reset timings, an external triggering source such as a PulseBlaster24 should be used.

Appendix C: Synchronization of Multiple Boards

In cases where it is necessary to achieve synchronization between multiple boards, it is possible to combine the use of the hardware trigger and reset mechanisms along with a single clock source. Diagram 5 below shows an example setup for achieving multiple board synchronization.

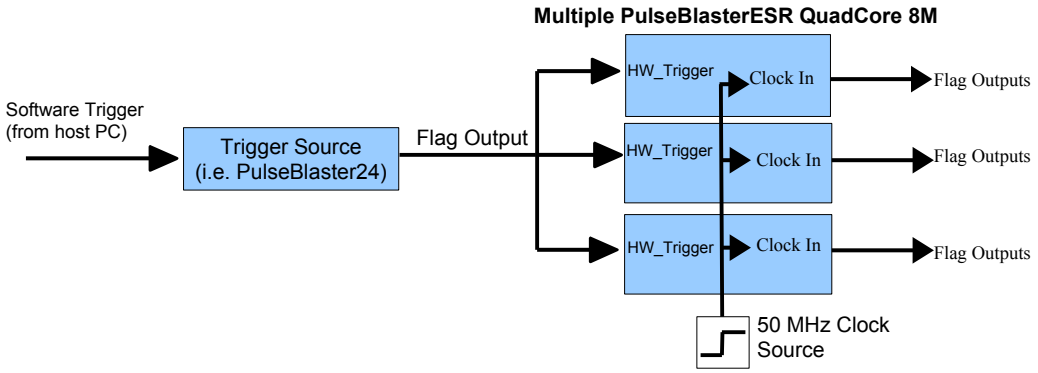


Diagram 5: Example setup for achieving multiple board synchronization

In order to ensure accurate synchronization of cores between boards, it is necessary that the boards be driven from the same clock source. The CLOCK header diagram in Appendix A shows the pin-out for the clock header for providing the clock signal to the boards.

Once the boards are being driven from a single clock source, an external trigger source (such as a PulseBlaster24) should be used to trigger the boards with precise timing. A single flag from the PulseBlaster24 can be used to drive all of the Hardware Triggers low in order to trigger the boards. In order to avoid multiple triggers being detected, the trigger pulse cannot be longer than the Pulse Program.

Each board can be can then be programmed from the host PC, and then triggered by sending a software trigger to the Trigger Source.

V. Related Products and Accessories

1. SpinCore MMCX Adapter Board Figure 1 – This adapter board allows easy access to the individual bits of the PulseBlasterESR DualCore. MMCX cables are available upon request. For ordering information contact SpinCore at http://www.spincore.com/contact.shtml.

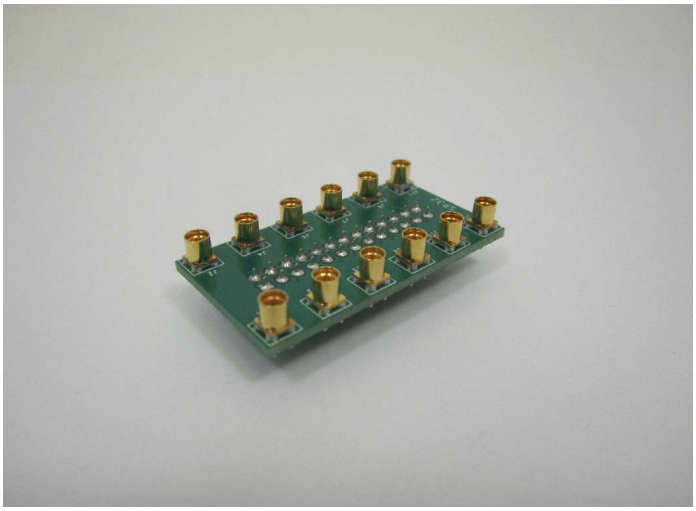


Figure 1: *MMCX Adapter Board allows easy access to individual bits*

VI. Contact Information

SpinCore Technologies, Inc.

4631 NW 53rd Avenue, Suite 103
Gainesville, Florida 32653, USA

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

Website: http://www.spincore.com

VII. Document Information

Document Title:	PulseBlasterESR QuadCore 8M
Document Number:	DA-83
File Name:	PBESR_QuadCore_Manual
Revision History:	Revision History Available at SpinCore