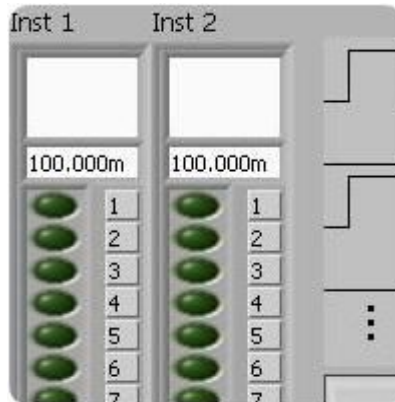




PulseBlasterESR MultiCore 8M

LabVIEW Extensions

User's Manual



SpinCore Technologies, Inc.
<http://www.spincore.com>

SpinCore PulseBlaster LabVIEW Extension

**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.**

© 2008 SpinCore Technologies, Inc. All rights reserved.

SpinCore Technologies, Inc. reserves the right to make changes to the product(s) or information herein without notice.
RadioProcessor™, PulseBlaster™, SpinCore, and the SpinCore Technologies, Inc. logos are trademarks of SpinCore Technologies, Inc.
All other trademarks are the property of their respective owners.

SpinCore Technologies, Inc. makes every effort to verify the correct operation of the equipment. This equipment version is not intended
for use in a system in which the failure of a SpinCore device will threaten the safety of equipment or person(s).

SpinCore PulseBlaster LabVIEW Extension

Table of Contents

I. Overview.....	4
II. Installation.....	7
Method 1: Stand-Alone Executables.....	7
Method 2: Customizable VIs.....	7
III. General Information.....	8
Instruction Description.....	8
Other Fields.....	8
Path Terminals.....	9
Error Terminals.....	10
LabVIEW Program Flow.....	10
IV. VI descriptions	
.....	11
Example VIs.....	11
PBLV_MultiCore8M.vi.....	11
Basic SubVIs.....	13
pb_error_handler.vi.....	13
pb_init.vi.....	13
pb_set_clock.vi.....	13
pb_close.vi.....	13
pb_start_programming.vi.....	13
pb_inst.vi.....	14
pb_stop_programming.vi.....	14
pb_start.vi.....	14
pb_stop.vi.....	14
V. Contact Information	
.....	15

SpinCore PulseBlaster LabVIEW Extension

I. Overview

The SpinCore PulseBlaster LabVIEW Extensions (PBLV) provide the functionality of programming and controlling PulseBlaster functionality in PulseBlaster, PulseBlasterESR, PulseBlasterESR MultiCore, PulseBlasterDDS, and RadioProcessor boards using the simple NI LabVIEW graphical programming interface. The package contains basic subVIs that can be used to include PulseBlaster interaction from your own LabVIEW programs, as well as some complete example VIs. Additionally, all of the examples are available as stand-alone applications to control.

The PBLV PulseBlasterESR MultiCore is an intuitive graphical equivalent of the SpinAPI C functions. The GUI (known as the front panel) has all the inputs needed to access the PulseBlasterESR including instruction registers, clock frequency, and buttons for starting and stopping the board. The input is then used in the back-end code (known as the block diagram) to access the C functions that control and program the PulseBlasterESR MultiCore. The LabVIEW block diagram is a one-to-one equivalent of the corresponding C code, without having to write code. An example of the front panel and block diagram are shown in Figure 1 and Figure 2.

All example VIs and subVIs are described in detail below.

SpinCore PulseBlaster LabVIEW Extension

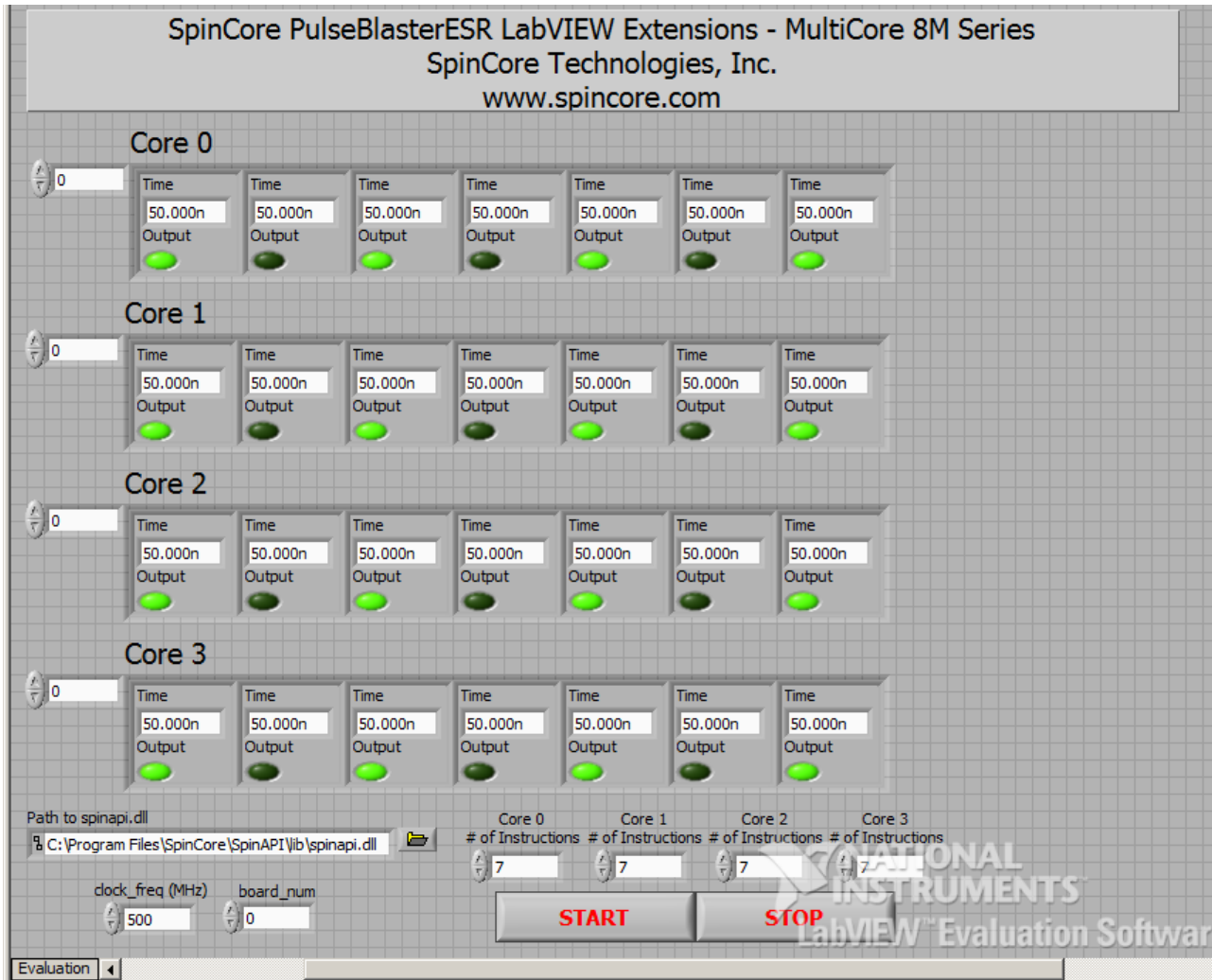


Figure 1: Example of PulseBlasterESR MultiCore LabVIEW Extensions User Interface

SpinCore PulseBlaster LabVIEW Extension

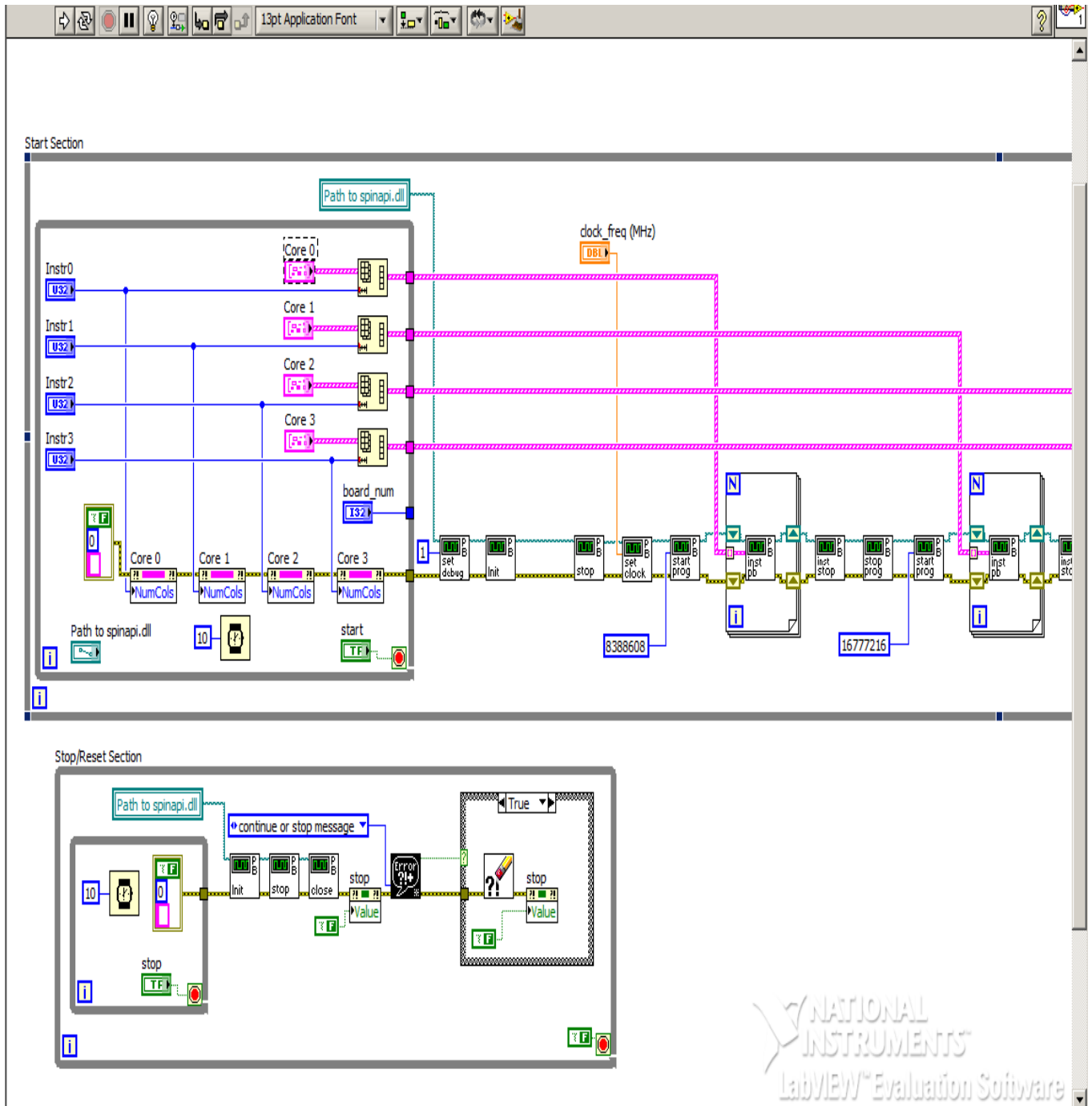


Figure 2: Example of PulseBlasterESR MultiCore LabVIEW Extensions Block Diagram

SpinCore PulseBlaster LabVIEW Extension

II. Installation

There are two methods of using the LabVIEW PulseBlaster Extensions. The first method is a set of stand-alone executables which will control the PulseBlaster boards with a simple, intuitive interface with no other necessary knowledge of LabVIEW programming. The second method is a set of LabVIEW VIs which can be used with the LabVIEW Development platform to create custom programs using the PBLV interface.

Method 1: Stand-Alone Executables

In order for PBLV stand-alone executables to work, the following must be installed:

- SpinCore Driver Suite - Please see the [SpinCore Driver Suite Installation Guide](http://www.spincore.com/CD/Setup/instructions.htm) (<http://www.spincore.com/CD/Setup/instructions.htm>) for more information.
- National Instruments [LabVIEW Run-Time Engine 8.6](http://www.spincore.com/CD/PBLV/LVRTE86min.exe) (<http://www.spincore.com/CD/PBLV/LVRTE86min.exe>) - Note if you have LabVIEW 8.6 or later installed, this is not needed.
- LabVIEW PulseBlasterESR MultiCore Extensions Stand-Alone executable
 - [Workbench](http://www.spincore.com/support/PBLV/PBLV_MultiCore8M.exe) (http://www.spincore.com/support/PBLV/PBLV_MultiCore8M.exe)

Method 2: Customizable VIs

In order for PBLV customizable VIs to work, the following must be installed:

- SpinCore Driver Suite - Please see the [SpinCore Driver Suite Installation Guide](http://www.spincore.com/CD/Setup/instructions.htm) (<http://www.spincore.com/CD/Setup/instructions.htm>) for more information.
- National Instruments LabVIEW 8.6 or later - If you do not have an installation of LabVIEW 8.6 or later installed, you may download a [30-day evaluation](https://lumen.ni.com/nicif/us/lveval/content.xhtml) (<https://lumen.ni.com/nicif/us/lveval/content.xhtml>) of the Development software.
- LabVIEW PulseBlaster Extensions
 - [Full PBLV project - LabVIEW 8.6](http://www.spincore.com/support/PBLV/PulseBlasterMultiCore_LabVIEW_extensions_86.zip) (http://www.spincore.com/support/PBLV/PulseBlasterMultiCore_LabVIEW_extensions_86.zip)
 - [Full PBLV project - LabVIEW 8.5](http://www.spincore.com/support/PBLV/PulseBlasterMultiCore_LabVIEW_extensions_85.zip) (http://www.spincore.com/support/PBLV/PulseBlasterMultiCore_LabVIEW_extensions_85.zip)
 - [Full PBLV project - LabVIEW 8.2](http://www.spincore.com/support/PBLV/PulseBlasterMultiCore_LabVIEW_extensions_82.zip) (http://www.spincore.com/support/PBLV/PulseBlasterMultiCore_LabVIEW_extensions_82.zip)

III. General Information

Instruction Description

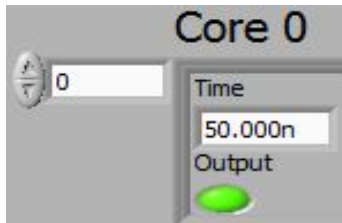


Figure 3: PulseBlasterESR
MultiCore Instruction

Throughout the LabVIEW PulseBlaster Extension, the instruction word, corresponding to a given interval in the pulse sequence, is given as shown to the left. Each core has its own set of instructions represented as an array under the corresponding header. The instruction on the front panel (Figure 3) is broken up into 3 fields displayed from top to bottom:

- Instruction Number is the box to the left of the array. It tells what instruction number is located at the left most instruction of the array. For example, if you wanted to jump to instruction 50 you would type 50 into the box and press enter. Then the array to the right will move so that instruction 50 is the first instruction, 51 is the next instruction and so on.
- Time is the duration that the current instruction is to be held in seconds. Smaller timescales can be used by using m for milliseconds, u for microseconds and n for nanoseconds. The range for the time varies from 36 nanoseconds to about 1.07 seconds for a 500 MHz clock.
- Output determines the state of the output bit. If an LED is on then it's corresponding output bit is high, and if the LED is off then it's corresponding output bit is low.

Other Fields

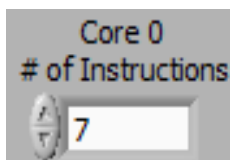


Figure 4: PulseBlasterESR
MultiCore Field

- Number of Instructions This field, shown in Figure 4, determines how many instructions will be implemented. There is one field for each core, so each core can have a different number of instructions. Once the number of instructions is entered, that many instructions will appear in the array for that particular core.
- Path to spinapi.dll is used to set the path to the spinapi.dll so the LabVIEW functions can connect to the spinapi library. (Figure 5)
- clock_frequency determines the state of the output bit. If an LED is on then it's corresponding output bit is high, and if the LED is off then it's corresponding output bit is low. (Figure 5)
- board_num is the board number of the PulseBlaster board wanting to be programmed. If more than one boards are installed, the specific board can be selected, if there is only one board, the default is board 0.

SpinCore PulseBlaster LabVIEW Extension

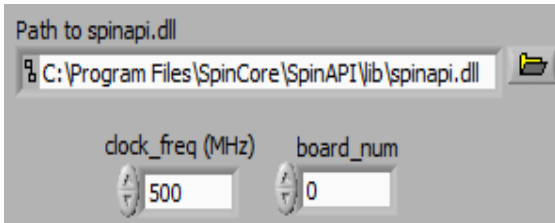


Figure 5: PulseBlasterESR MultiCore Field

Path Terminals

All of the SubVIs have path input and output terminals. This is a reference to the path where spinapi.dll is installed on the PC. The default is C:\Program Files\SpinCore\SpinAPI\lib\spinapi.dll however this may be changed depending on the installation. All subVIs should have the “path in” connected. For ease of programming, the path terminals can be daisy chained since all subVIs point to the same dll. See Figure 6 and Figure 7 for an example of chaining these terminals.

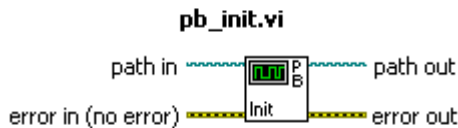


Figure 6: Path and error terminals



Figure 7: Example of chaining path and error terminals

Error Terminals

All of the SubVIs have error input and output terminals. When a VI is built, the error out terminal of a subVI should be connected to the error in terminal of the following subVI. These terminals are used to help facilitate sequential execution of the functions, as well as provide debugging information to the user if an error occurs in the VI. See Figure 7 and Figure 6 for an example of these terminals. When chaining, the order of the subVIs corresponds to the order in which the functions will be called.

LabVIEW Program Flow

The LabVIEW Block diagram is set up to independently control the 2 major functions (start and stop/reset) using two while loops running in parallel. Within each loop is another loop that continuously waits for the specified button to be pressed. Once the Start or Stop button is pressed, the inner loop will exit and program flow will be passed to the chain of subVIs. After the chain of functions completes, program flow will return to the inner loop to wait for the button again. An example of this is shown in Figure 8.

IV. VI descriptions

Example VIs

PBLV_MultiCore8M.vi

This VI is a full system to be used to control the PulseBlasterESR MultiCore boards to their full capabilities. The VI can use the full range of instructions capable for the board (see specific board manual for the maximum number of instructions allowed), program multiple boards, as well as have the same functionality of the previous example VIs.

To use this VI, first you must set the location of spinapi.dll and the clock frequency of the board in use. Then you can set how many instructions are to be used as well as the contents of these instructions using the instruction registers as described above. Then click "START" to load and start the program and "STOP" to stop the currently running program. Finally, if you have multiple boards in your system, you can change the board by entering the correct board number into "board_num" and then clicking "CHANGE BOARD" (NOTE: You must wait until the current button becomes released before you can access the other controls). The number of instructions can be changed on the fly and then reprogrammed to the board.

See Figure 9 for the front panel of this VI.

SpinCore PulseBlaster LabVIEW Extension

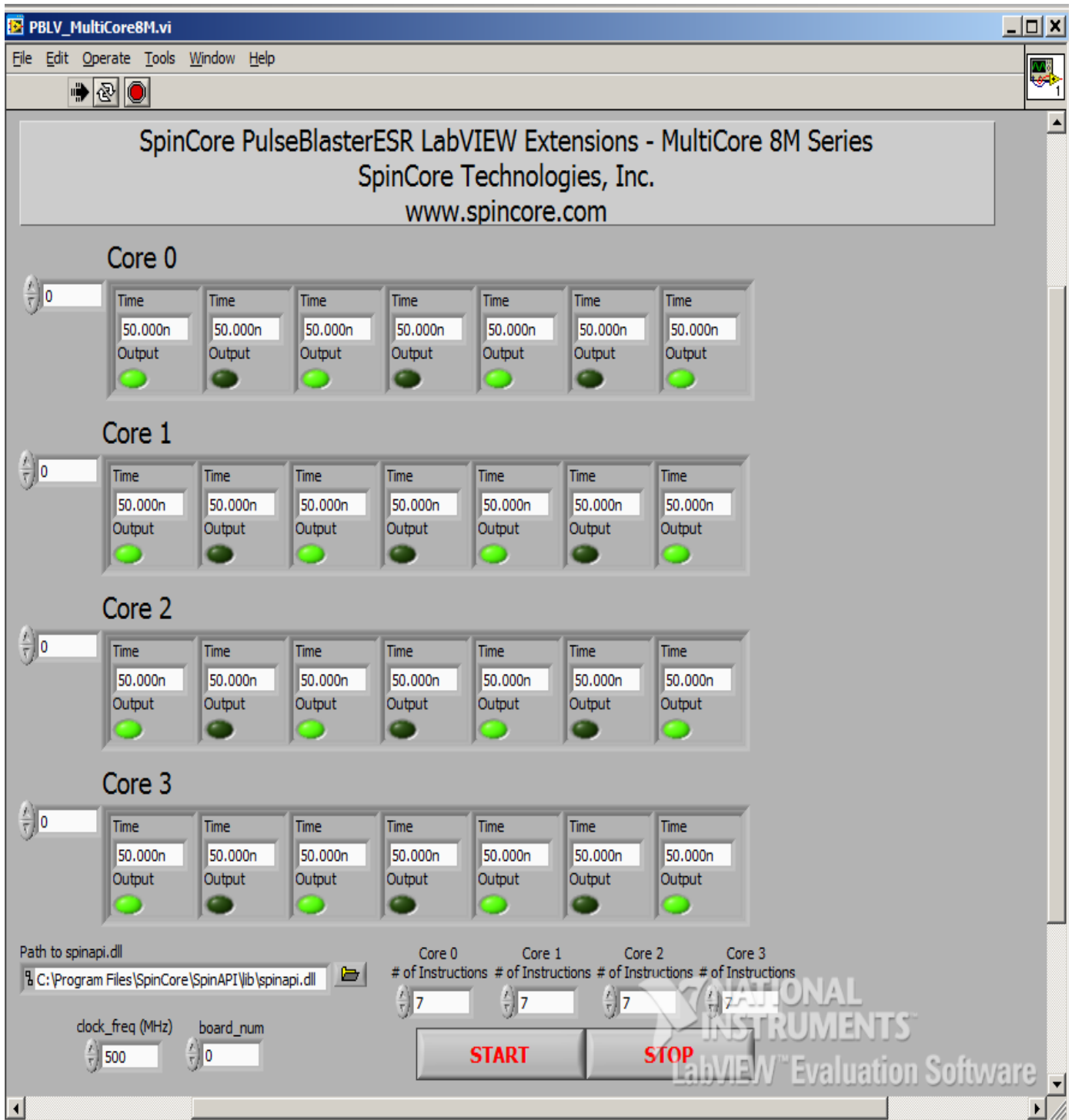


Figure 9: Front panel of PulseBlasterESR MultiCore Workbench. When the number of instructions for a specific core is increased, they will show up to the right of the current visible instruction of the corresponding core and when the number is decreased they will disappear from the screen. Only the visible instructions will be loaded onto the PulseBlaster's memory.

SpinCore PulseBlaster LabVIEW Extension

Basic SubVIs

pb_error_handler.vi

This VI will catch an error from SpinAPI and pass it into the LabVIEW code for debugging purposes. It takes the return code from any function call and determines if there is an error (return code is a negative number). If an error occurred, the VI calls pb_get_error from SpinAPI which returns the error text. This error text added to the LabVIEW error cluster and the error status bit is set true. All functionality will be bypassed if a previous error is passed into the VI.

pb_init.vi

Corresponds to the SpinAPI function int pb_init()

This VI initializes the PulseBlaster board. It calls the pb_init function from SpinAPI using the Call Library Function Node. It then checks for an error using the pb_error_handler. If there was an error at this stage, the VI determines if the board was not previously closed, if so the VI clears the error, closes the board and retries pb_init. All functionality will be bypassed if an error is passed into the VI.

pb_set_clock.vi

Corresponds to the SpinAPI function void set_clock(double clock_freq)

This VI sets the clock frequency of the board in MHz. It calls the pb_set_clock function from SpinAPI using the Call Library Function Node. All functionality will be bypassed if an error is passed into the VI.

pb_close.vi

Corresponds to the SpinAPI function int pb_close()

This VI releases PulseBlaster board. It calls the pb_close function from SpinAPI using the Call Library Function Node. It then checks for an error using the pb_error_handler. All functionality will be bypassed if a previous error is passed into the VI.

pb_start_programming.vi

Corresponds to the SpinAPI function int pb_start_programming(int device)

This VI initializes the system to receive programming information. It calls the pb_start_programming function from SpinAPI using the Call Library Function Node. The Device Enum Control sets the device input of the pb_start_programming function. It then

SpinCore PulseBlaster LabVIEW Extension

checks for an error using the pb_error_handler. All functionality will be bypassed if an error is passed into the VI.

pb_inst.vi

Corresponds to the SpinAPI function DWORD pb_inst(DWORD flag, double time_ns);

This VI sends one instruction of the pulse program to the board. It calls the pb_inst function from SpinAPI using the Call Library Function Node. It contains the output flag, length of the. It then checks for an error using the pb_error_handler. All functionality will be bypassed if an error is passed into the VI.

pb_stop_programming.vi

Corresponds to the SpinAPI function int pb_stop_programming()

This VI tells the board to stop receiving programming information. It calls the pb_stop_programming function from SpinAPI using the Call Library Function Node. It then checks for an error using the pb_error_handler. All functionality will be bypassed if an error is passed into the VI.

pb_start.vi

Corresponds to the SpinAPI function int pb_start()

Once board has been programmed, this instruction will start execution of pulse program. It calls the pb_start function from SpinAPI using the Call Library Function Node. It then checks for an error using the pb_error_handler. All functionality will be bypassed if an error is passed into the VI.

pb_stop.vi

Corresponds to the SpinAPI function int pb_stop()

This VI stops output of the board. It calls the pb_stop function from SpinAPI using the Call Library Function Node. It then checks for an error using the pb_error_handler. All functionality will be bypassed if an error is passed into the VI.

SpinCore PulseBlaster LabVIEW Extension

V. Contact Information

**SpinCore Technologies, Inc.
4623 NW 53rd Avenue, SUITE 5
Gainesville, FL 32653
USA**

Telephone: 352-271-7383

Web: <http://www.spincore.com/>