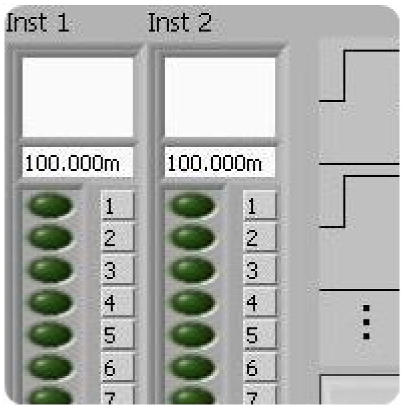
SpinCore PulseBlaster

LabVIEW Extensions

User'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. Overview

The SpinCore PulseBlaster LabVIEW Extensions (PBLV) provide the functionality of programming and controlling PulseBlaster functionality in PulseBlaster, PulseBlasterESR, 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 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 PulseBlaster including instruction registers, clock frequency, and buttons for loading, 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 PulseBlaster. 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.

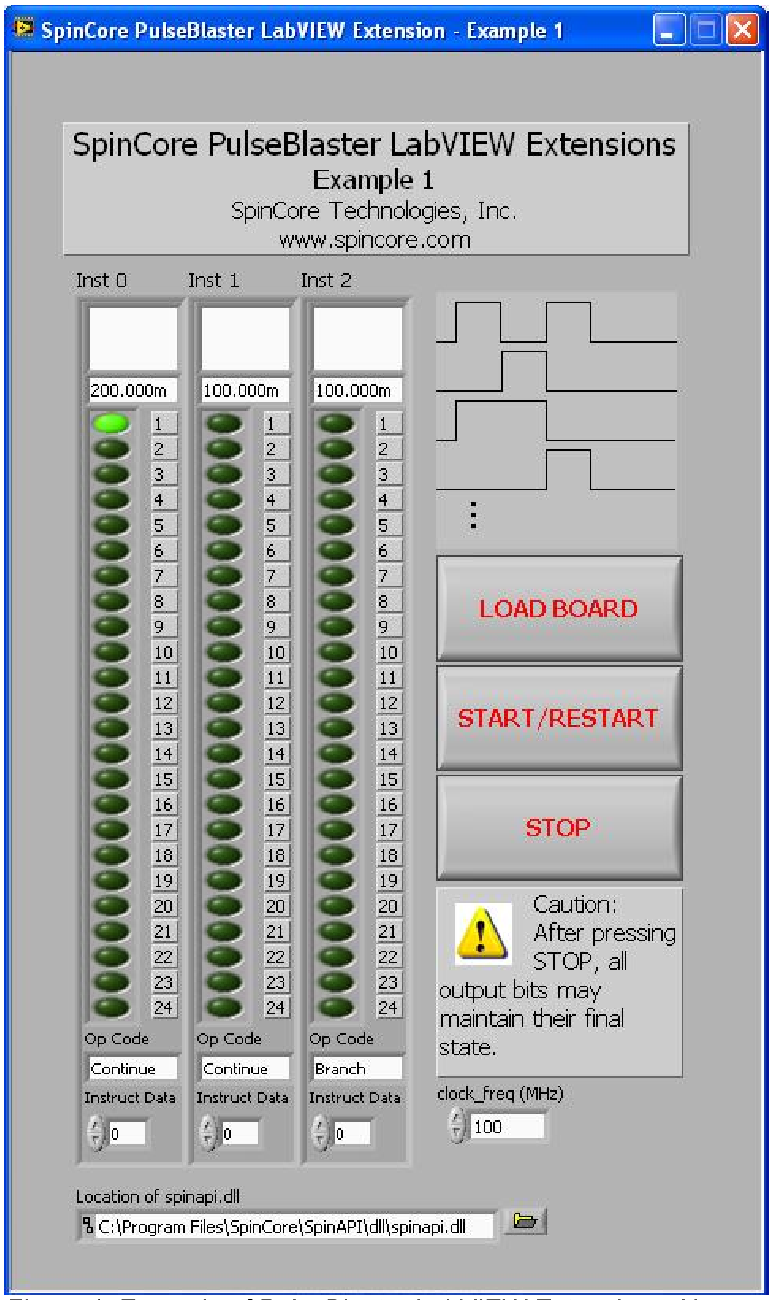

SpinCore PulseBlaster LabVIEW Extension - Example 1
SpinCore PulseBlaster LabVIEW Extensions
Example 1
SpinCore Technologies, Inc.
www.spincore.com
Inst 0
Inst 1
Inst 2
200.000m
100.000m
100.000m
Op Code
Continue
Continue
Branch
Instruct Data
0
LOAD BOARD
START/RESTART
STOP
Caution: After pressing STOP, all output bits may maintain their final state.
clock_freq (MHz)
100
Location of spinapi.dll
C:\Program Files\SpinCore\SpinAPI\dll\spinapi.dll

Figure 1: Example of PulseBlaster LabVIEW Extensions User Interface

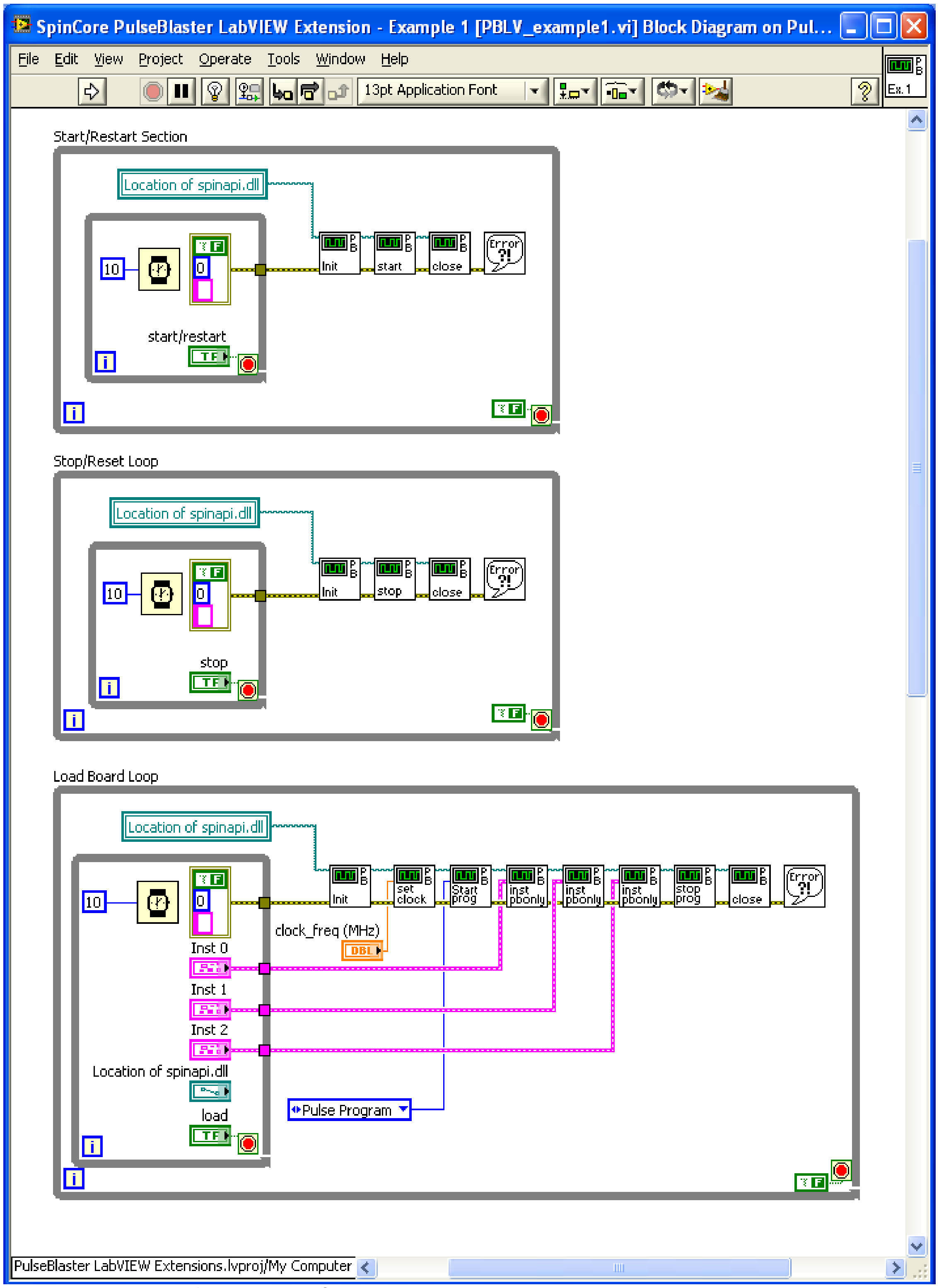


Figure 2: Example of PulseBlaster LabVIEW Extensions Block Diagram

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-DDS-II 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/support/spinapi/instructions/) for more information.
- National Instruments LabVIEW Run-Time Engine 2010 (http://www.spincore.com/support/PBLV/LVRTE2010std.exe) - Note if you have LabVIEW 9.0 or later installed, this is not needed.
- LabVIEW PulseBlaster Extensions Stand-Alone executables located here.

Method 2: Customizable VIs

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

- SpinCore Driver Suite - Please see the SpinCore Driver Suite Installation Guide (http://www.spincore.com/support/spinapi/instructions/) for more information.
- National Instruments LabVIEW 8.6 or later - If you do not have LabVIEW 8.6 or later installed, you may download a 30-day evaluation (https://lumen.ni.com/nicif/us/lveval/content.xhtml) of the latest LabVIEW development software.
  (NOTE: Customizable VIs are availabe for versions of LabVIEW as old as LabVIEW 8.0. For customizable VIs older than LabVIEW 8.6, please contact SpinCore Technologies via the web forum.)
- LabVIEW PulseBlaster Extensions located here.
  (NOTE: Customizable VIs use the C-calling convention. For customizable VIs that use the WINAPI calling convention please contact SpinCore Technologies via the web forum.)

III. General Information

Instruction Description

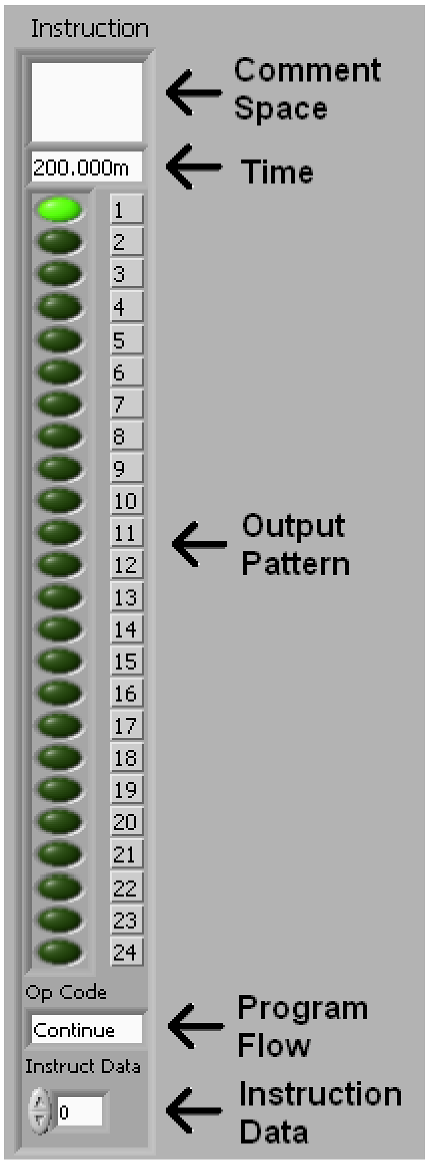


Figure 3: PulseBlaster Instruction

Throughout the LabVIEW PulseBlaster Extension, the instruction word, corresponding to a given interval in the pulse sequence, is given as shown to the left. The instruction on the front panel (Figure 3) is broken up into 5 fields displayed from top to bottom:

- Comment space is an empty text box. This area is used to annotate your instruction and is not programmed onto the PulseBlaster board.
- 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. Range and resolution varies depending on what board is being used. This corresponds to Length in SpinAPI.
- Output Pattern determines the state of each TTL 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. This corresponds to Flags in SpinAPI.
  NOTE: Some firmware versions for the PulseBlasterESR-PRO boards require bits 22-24 to be high in order to have an instruction length longer than 6 clock cycles. For more information see the “BNC Short Pulse Feature” in Appendix I of the PulseBlasterESR-PRO Manual.
- Program Flow determines the flow of program instructions *after* the bit pattern is displayed. Available instructions are:
  - CONTINUE - Program execution continues to next instruction.
  - STOP - Stop execution of program.
  - LOOP - Specify beginning of a loop. Execution continues to next instruction. Instruction data used to specify number of loops.
  - END_LOOP Execution returns to beginning of loop and decrements loop counter. Instruction data used to specify beginning of loop.

  - JSR - Program execution jumps to beginning of a subroutine. Instruction data used to specify address of first subroutine instruction.
  - RTS - Program execution returns to instruction after JSR was called.
  - BRANCH - Program execution continues at specified instruction. Instruction data specifies address of next instruction.
  - LONG_DELAY - For long interval instructions. Data field specifies a multiplier of the length field. Execution continues to next instruction.
  - WAIT - Program execution stops and waits for software or hardware trigger. Execution continues to next instruction after receipt of trigger. (Please see SpinAPI documentation for more information on limits of certain Op Codes)

- Instruction Data is the data to be used for certain instructions determined by the Op Code. Note for certain instructions this field is not used.

  Figure 4 Below shows detail of the corresponding instruction word as depicted in LabVIEW's block diagram.

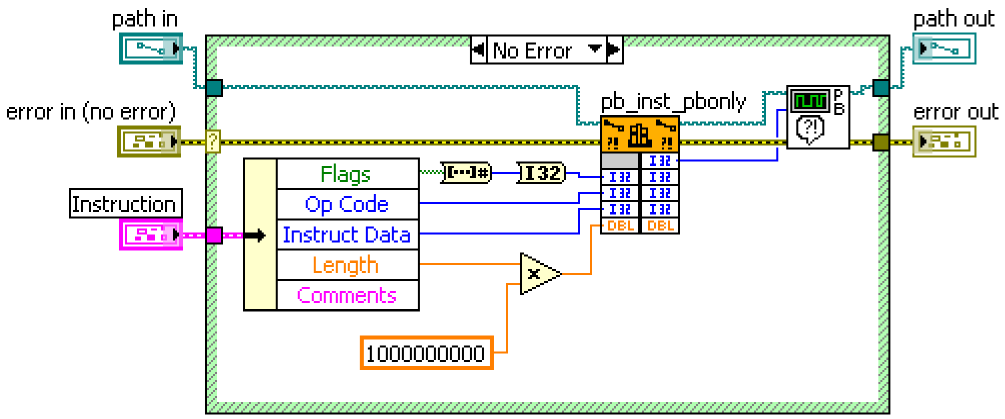


Figure 4: Example of how the instruction is created and sent to the board in the PBLV Block Diagram

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\dll\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 5 and Figure 6 for an example of chaining these terminals.

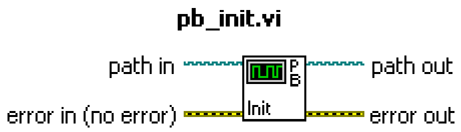


Figure 5: Path and error terminals

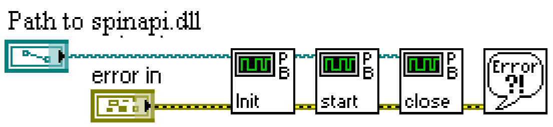


Figure 6: Example of chaining path and error terminals

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 6 and Figure 5 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 three major functions (start/restart, stop/reset, load) using three while loops running in parallel. Within each loop is another loop that continuously waits for the specified button to be pressed. Once the 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 7.

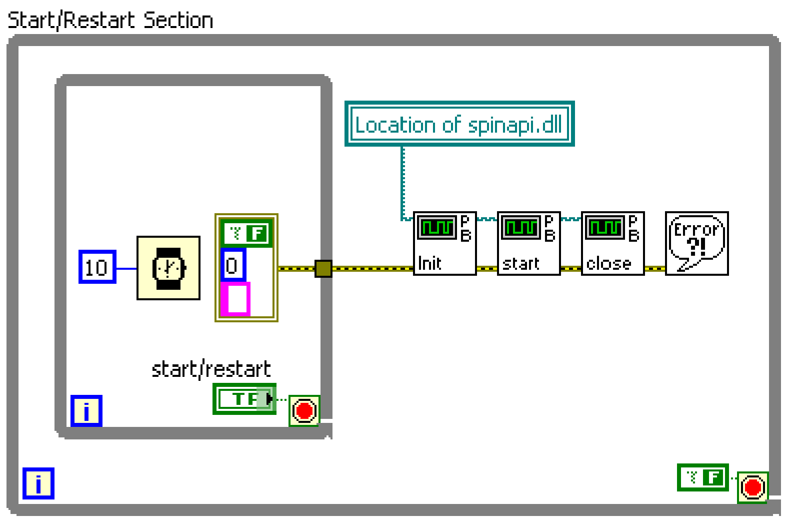


Figure 7: LabVIEW Program Flow Example

IV. VI descriptions

Example VIs

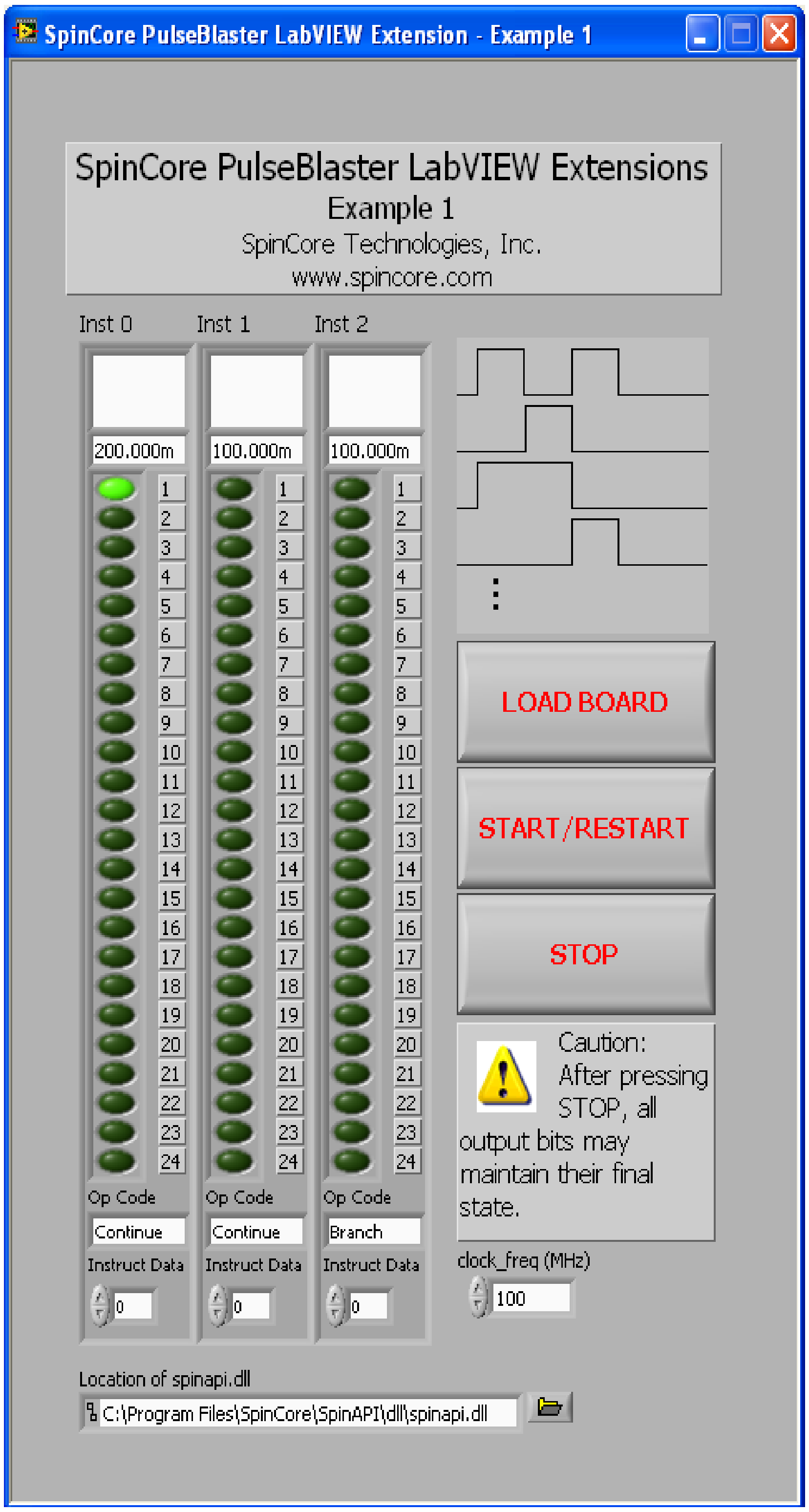
PBLV_Example1.vi

This corresponds to the example c file pb24_ex1.c

This VI programs three 80-bit VLIW instructions into the PulseBlaster memory and starts and stops the program on command. To start the VI press the Run button at the top of the LabVIEW window. First you must insert the clock frequency value of the board into the clock_freq (MHz) field as well as the path to spinapi.dll as described above. Initially the instructions are already set up to turn on bit 1 for 200 ms and continue, turn off bit 1 for 100 ms and continue, and leave the bits off for 100 ms and branch to the start. You may change these instructions as desired.

When the instructions are set up successfully, you must first load the board by clicking the “LOAD BOARD” button. To start the program click on the “START” button, and to stop the program click on the “STOP” button. All three functions are independent and may be run at any time, however an instruction sequence must be loaded into the board before it can be started. **Caution**: After pressing STOP, all output bits may maintain their final state.

See Figure 8 for a screen shot of PBLV_Example1.



	Inst 0	Inst 1	Inst 2
Duration	200.000m	100.000m	100.000m
Op Code	Continue	Continue	Branch
Instruct Data	0	0	0

clock_freq (MHz): 100

Location of spinapi.dll: C:\Program Files\SpinCore\SpinAPI\dll\spinapi.dll

Caution: After pressing STOP, all output bits may maintain their final state.

Figure 8: LabVIEW PulseBlaster Extension - Example 1

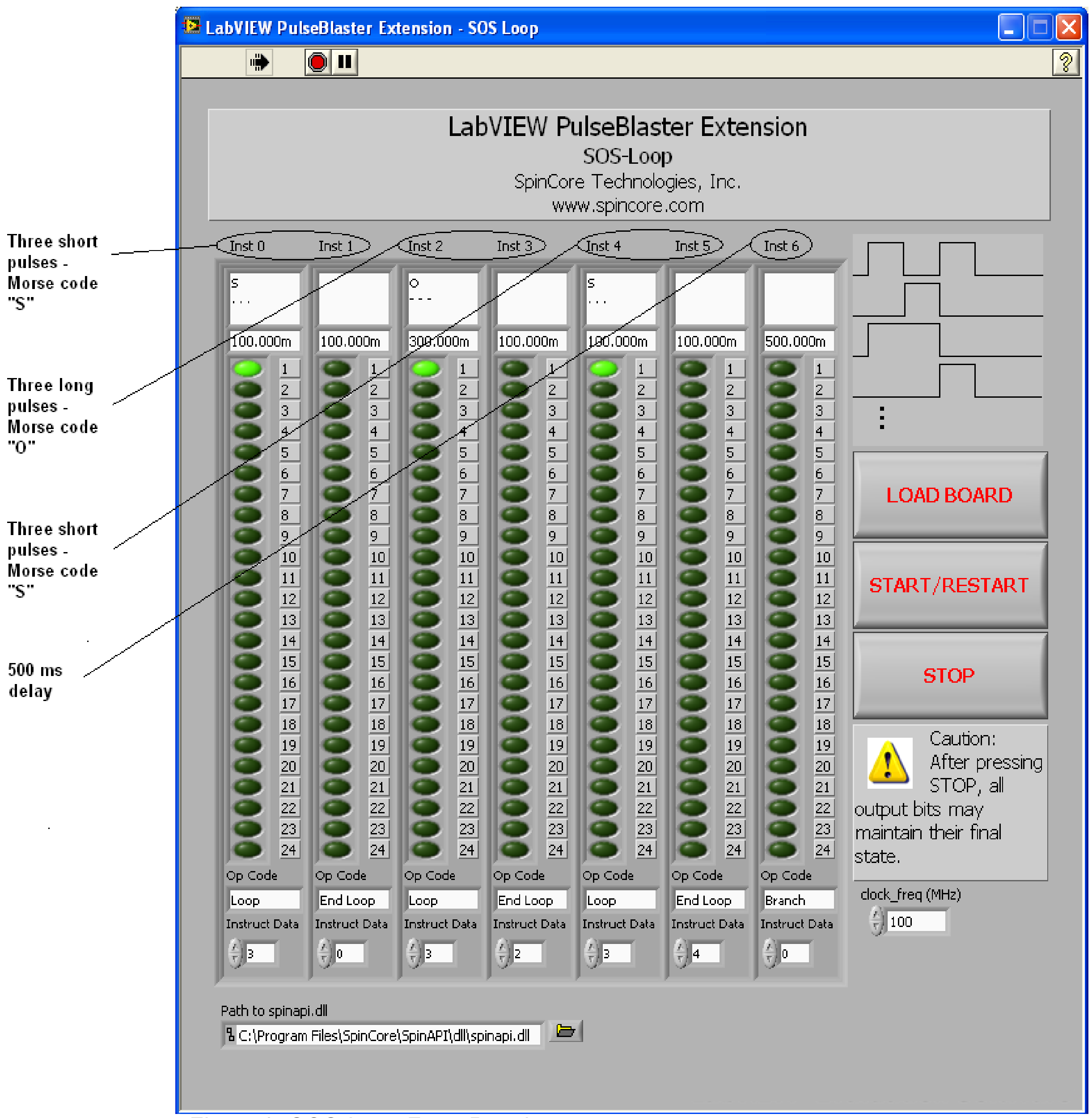
PBLV_SOS-Loop.vi

This VI corresponds to the PulseBlaster Interpreter example SOS-Loop.pb

This VI demonstrates the LOOP and ENDLOOP instructions by outputting the Morse code SOS signal (three short pulses, three long pulses and three short pulses). The VI is run the same as in LVPB_Example1.vi.

This example VI sends the following 7 instructions to the PulseBlaster board:

Inst #	**Time**	**Output Pattern**	**Program Flow**	**Inst Data**	**Comments**
0	100 ms	On	Loop	3	On for 100 ms, mark beginning of loop, loop 3 times
1	100 ms	Off	End Loop	0	Off for 100 ms, return to beginning of loop at inst 0
2	300 ms	On	Loop	3	On for 300 ms, mark beginning of loop, loop 3 times
3	100 ms	Off	End Loop	2	Off for 100 ms, return to beginning of loop at inst 2
4	100 ms	On	Loop	3	On for 100 ms, mark beginning of loop, loop 3 times
5	100 ms	Off	End Loop	4	Off for 100 ms, return to beginning of loop at inst 4
6	500 ms	Off	Branch	0	Off for 500 ms, then branch to instruction 0


Three short pulses - Morse code "S"
Three long pulses - Morse code "O"
Three short pulses - Morse code "S"
500 ms delay
LOAD BOARD
START/RESTART
STOP

LabVIEW PulseBlaster Extension - SOS Loop

LabVIEW PulseBlaster Extension
SOS-Loop
SpinCore Technologies, Inc.
www.spincore.com

	Inst 0	Inst 1	Inst 2	Inst 3	Inst 4	Inst 5	Inst 6
	S ...		O - - -		S ...		
	100.000m	100.000m	300.000m	100.000m	100.000m	100.000m	500.000m
Op Code	Loop	End Loop	Loop	End Loop	Loop	End Loop	Branch
Instruct Data	3	0	3	2	3	4	0

Caution: After pressing STOP, all output bits may maintain their final state.

clock_freq (MHz) 100

Path to spinapi.dll
C:\Program Files\SpinCore\SpinAPI\dll\spinapi.dll

Figure 9: SOS-Loop Front Panel

The LabVIEW block diagram is slightly different in this example than in Example 1. In Example 1, 3 different instructions are read and the 3 different pb_inst_pbonly subVIs are called. In SOS-Loop, an array of 7 instructions is read and later passed into a for loop where the individual instructions are indexed in order and programmed to the board using the same pb_inst_pbonly subVI. This is done for a cleaner block diagram. All other aspects of this example are identical to PBLV_Example1.vi. See Figure 10 for the SOS-Loop Block Diagram.

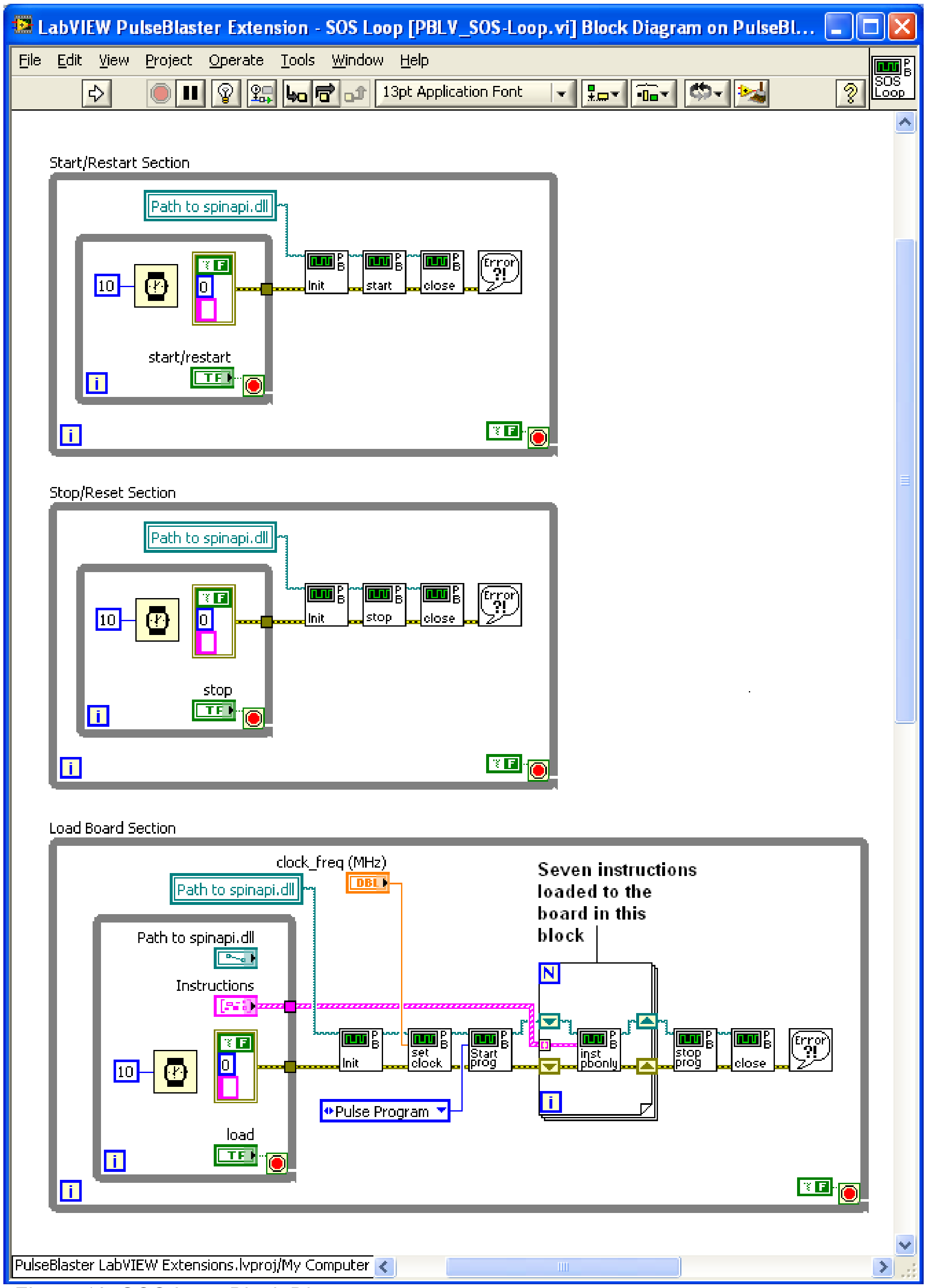

LabVIEW PulseBlaster Extension - SOS Loop [PBLV_SOS-Loop.vi] Block Diagram on PulseBl...
File Edit View Project Operate Tools Window Help
13pt Application Font
Start/Restart Section
Path to spinapi.dll
10
Init
start
close
start/restart
Stop/Reset Section
stop
Load Board Section
clock_freq (MHz)
DBL
Instructions
load
set clock
Start prog
inst pbonly
stop prog
Pulse Program
Seven instructions loaded to the board in this block
PulseBlaster LabVIEW Extensions.lvproj/My Computer

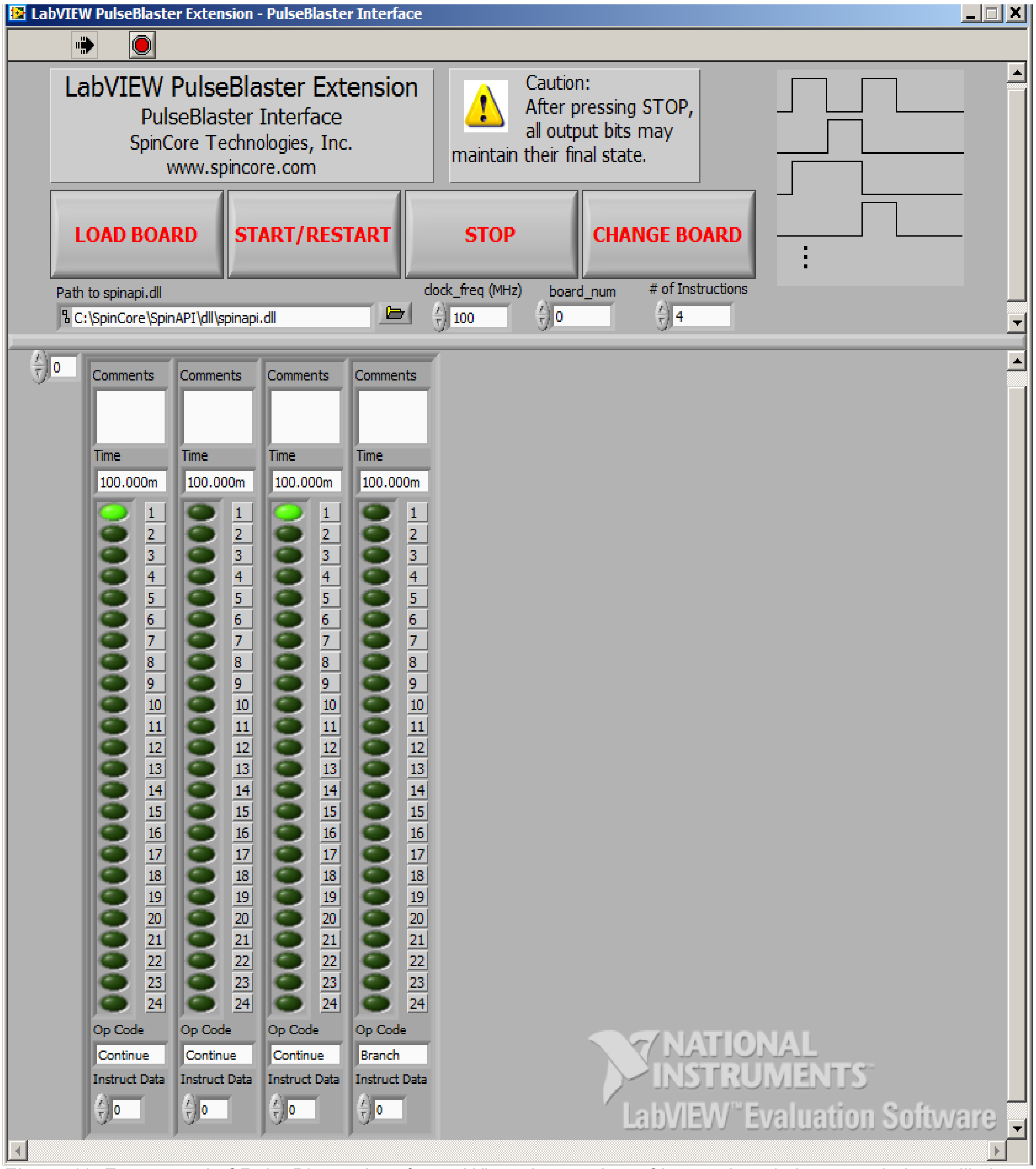
Figure 10: SOS Loop Block Diagram

PBLV_Interface.vi

This VI is a full system to be used to control the PulseBlaster 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 and then program the board by clicking “LOAD BOARD”. Then click “START” to 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 11 for the front panel of this VI and Figure 12 for the block diagram.



LabVIEW PulseBlaster Extension - PulseBlaster Interface

LabVIEW PulseBlaster Extension
PulseBlaster Interface
SpinCore Technologies, Inc.
www.spincore.com

Caution:
After pressing STOP, all output bits may maintain their final state.

LOAD BOARD | START/RESTART | STOP | CHANGE BOARD

Path to spinapi.dll: C:\SpinCore\SpinAPI\dll\spinapi.dll
clock_freq (MHz): 100
board_num: 0
# of Instructions: 4

Instruction index: 0

	Instruction 1	Instruction 2	Instruction 3	Instruction 4
Comments				
Time	100.000m	100.000m	100.000m	100.000m
Bits on	1		1	
Op Code	Continue	Continue	Continue	Branch
Instruct Data	0	0	0	0

Figure 11: Front panel of PulseBlaster Interface. When the number of instructions is increased, they will show up to the right of the current visible instruction and when the number is decreased they will disappear from the screen. Only the visible instructions will be loaded onto the PulseBlaster's memory.

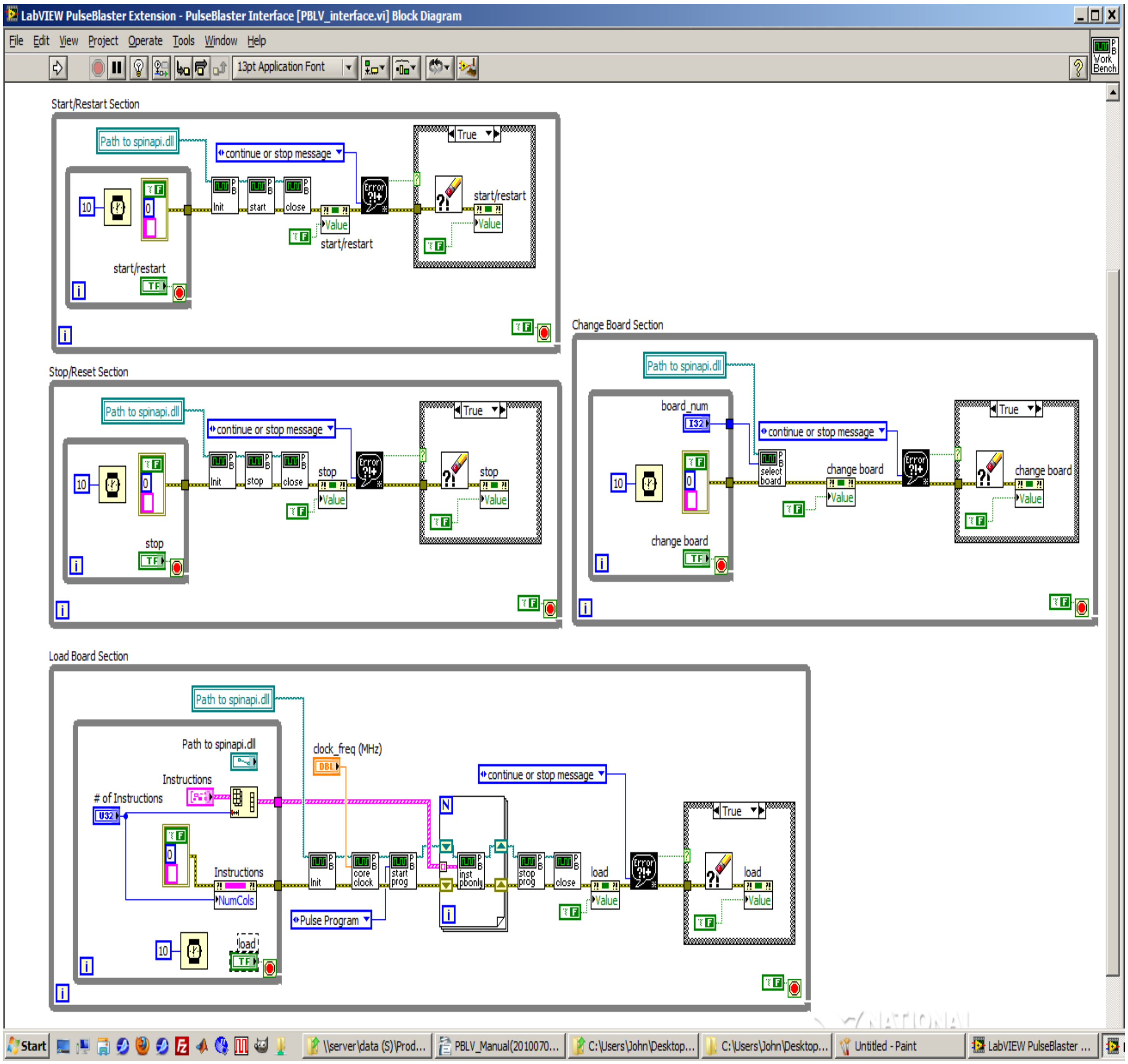


Figure 12: Block diagram of PulseBlaster Interface.

Basic SubVIs

These SubVIs use Call Library Function Nodes to access SpinAPI's C-functions. By default these nodes use the C-calling convention. For customizable VIs that use the WINAPI calling convention please contact SpinCore Technologies via the web forum.

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

pb_inst_pbonly.vi

Corresponds to the SpinAPI function int pb_inst_pbonly(int flags, int inst, int inst_data, double length);

This VI sends one instruction of the pulse program to the board. It calls the pb_inst_pbonly function from SpinAPI using the Call Library Function Node. The instruction control is the 80-bit VLIW to be programmed to the board. It contains the 24-bit TTL flags, Op Code (which instruction to be executed), instruction data and length that the instruction should be carried out 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.

V. Contact Information

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

Telephone (USA):	**352-271-7383**
Fax (USA):	**352-371-8679**
Website:	http://www.spincore.com
Web Form:	http://www.spincore.com/contact.shtml