Getting Started with the LabVIEW FPGA Module and the Spartan-3E Starter Board

Contents

Software Requirements and Installation Notes

- LabVIEW 8.5 and LabVIEW FPGA Module 8.5.
- Software is not supported on 64-bit Windows Vista.
- If you already have Xilinx tools installed and your Spartan-3E is plugged in, Windows will not allow you to install the Xilinx USB cable driver. You can either turn off your Spartan-3E during install or uninstall the USB cable in the Windows Device Manager.

Software Installation

Complete the following steps to install LabVIEW 8.5, LabVIEW FPGA Module 8.5, and LabVIEW FPGA support for Spartan-3E for Windows.

1. Install LabVIEW 8.5.
   a. Disable any automatic virus detection programs before you install. Some virus detection programs interfere with the installation program.
   b. Log on as an administrator or as a user with administrator privileges.
   c. Insert the LabVIEW 8.5 installation CD and follow the instructions that appear on the screen. You first are prompted to install LabVIEW. The directions on the screen will help you to activate your LabVIEW license.

2. Insert the LabVIEW FPGA 8.5 installation CD and follow the instructions that appear on the screen.

3. Install LabVIEW FPGA support for Spartan-3E.
   a. Double-click *Setup.exe* to run the LabVIEW FPGA Support for Spartan-3E installer.
   b. Follow the instructions that appear on the screen to finalize the installation.

Xilinx ISE WebPACK

The LabVIEW FPGA Module does not require the ISE WebPACK. However, you must install the Xilinx ISE WebPACK if you plan to use VHDL code with the LabVIEW FPGA Module. This document includes a demonstration of how to link existing VHDL code with LabVIEW FPGA code. An IP core integration example is shown in Example 4 below.

Complete the following steps to install ISE WebPACK for Spartan-3E.

1. Go to the Xilinx website at http://www.xilinx.com/ise/logic_design_prod/webpack.htm
2. Click the *Download WebPACK* link**.**
3. Logon to Xilinx's website or register a new account.
4. You have two options for installing WebPACK.
   Option A: Download WebPACK from a web installer (1st link provided)
   Option B: Download WebPACK from a single file download (2nd link provided)
5. Follow the instructions that appear on the screen to finalize the installation.
6. You may be prompted to install WebPACK service packs. Make sure to install the service packs provided.

After installations are complete, enable any virus detection programs you disabled and restart your computer.

Connecting Spartan-3E Starter Board to Computer

Complete the following steps to connect the Spartan-3E Starter Board to the computer.

1. Connect the USB B Connector from the USB Cable provided to the Spartan-3E Starter Board as shown in Figure 1:

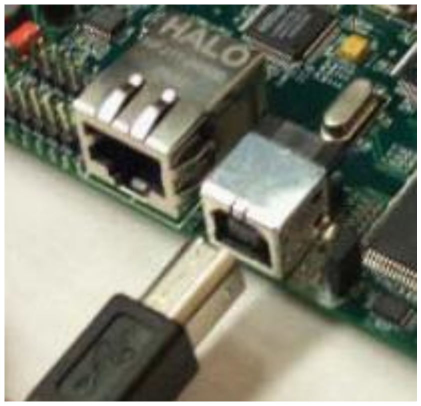


Figure 1: USB B Connector to Spartan-3E Starter Board

2. Connect the USB A Connector from the USB cable to the host computer.

3. Connect the power cable provided to the Spartan-3E Starter Board power port.

4. Switch the **On/Off Dip Switch** to On to power on the Spartan-3E Starter Board as shown in Figure 2. When the power is turned on to the device, the Power LED located at the top-left corner of the device will light up.

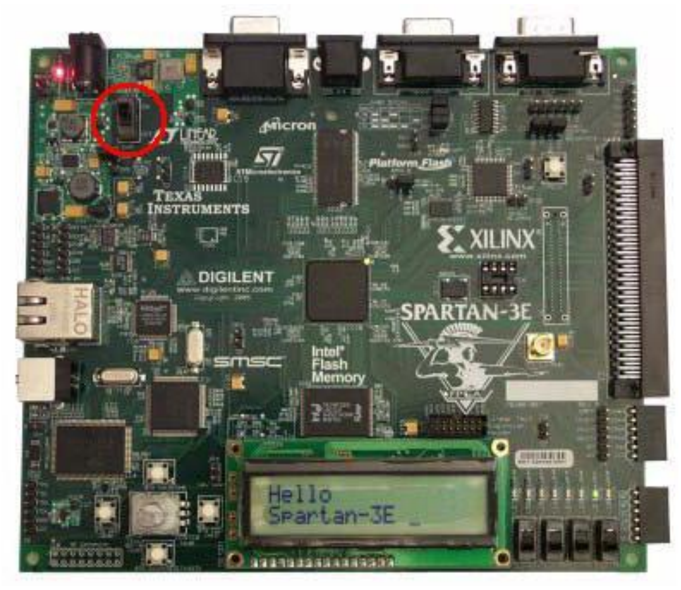


Figure 2: On/Off Switch Circled

Driver Installation
Once the board is powered on and connected to the computer, the Windows operating system automatically recognizes the new hardware and prompts you to install the associated drivers.

1. Make sure to install all appropriate drivers.
2. You might be prompted to install two USB cables- install all features of the plug and play drivers.

Getting Started Examples

Note: This document assumes you are familiar with the LabVIEW graphical programming environment. Refer to the *Getting Started with LabVIEW* manual and *LabVIEW Fundamentals* manual to learn more about the basic LabVIEW features you use to build VIs and projects. You can find these manuals in the <LabVIEW Dir>\manuals\ directory.

To verify your setup and become familiar with the LabVIEW FPGA environment, run a few shipping examples. The examples are located in the <LabVIEW Dir>\examples\Spartan-3E\... directory. Load the LabVIEW project (.lvproj) file for any example and view the readme located in the project for more information. Note that opening the VI before the project might lead to unexpected failures. Also note that all examples are pre-compiled and do not need to be recompiled to be targeted to the Spartan-3E Starter Board.

Example 1: LED

This example demonstrates how to shift two LEDs on the Spartan-3E Starter Board. The LEDs shift in opposite directions and converge back to the middle in a continuous manner. Figure 3 shows the LEDs found on the Spartan-3E Starter Board.

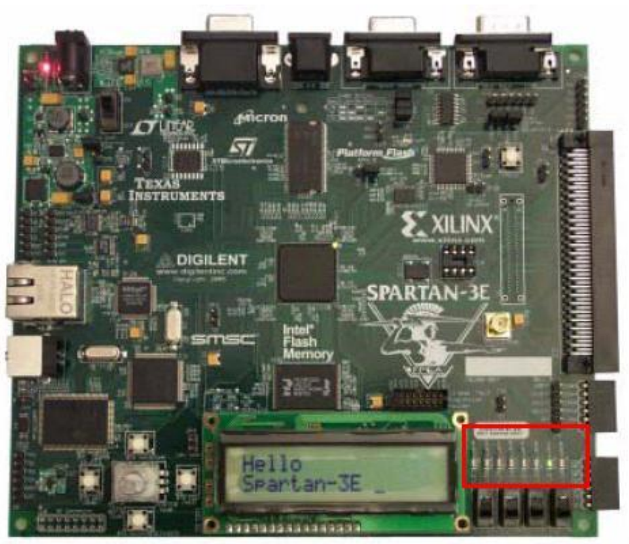
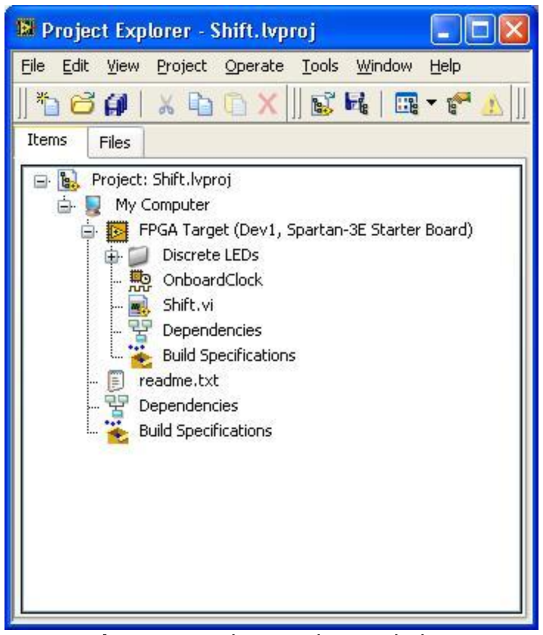


Figure 3: LED Location on the Spartan-3E

1. Open the LabVIEW project (.lvproj) located at <LabVIEW Dir>\examples\Spartan-3E\LEDs\Shift.lvproj
2. In the **Project Explorer** window, expand the plus sign located near FPGA target name. The **Project Explorer** window should be as shown in Figure 4.


Project Explorer - Shift.lvproj
File Edit View Project Operate Tools Window Help
Items
Files

```
Project: Shift.lvproj
  My Computer
    FPGA Target (Dev1, Spartan-3E Starter Board)
      Discrete LEDs
      OnboardClock
      Shift.vi
      Dependencies
      Build Specifications
    readme.txt
    Dependencies
    Build Specifications
```

Figure 4: Project Explorer Window

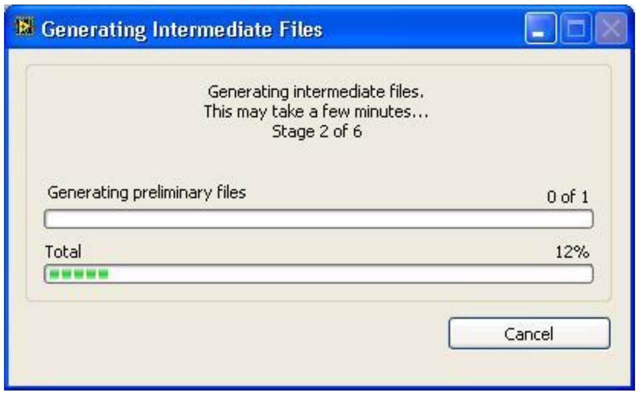
3. Double-click the Shift VI under the FPGA target to open the front panel for the example. Press <Ctrl>-E to open the block diagram for further observation.
4. Press the **Run** arrow to download the VI to the FPGA via USB. After a few moments, you should be able to see the LEDs shift on the Spartan-3E. The front panel should also indicate the status of the eight LEDs. Press the **Stop** button to abort the program. Note that this example was precompiled. If you change to the VI, LabVIEW displays the **Save** dialog box. If you save the VI, you must recompile the VI.

The LED example demonstrates front panel communication via USB directly to the FPGA code running on the Spartan-3E. This is extremely useful for debugging purposes, and you do not need a host VI to communicate to the FPGA code. There are some cases in which the host VI serves an extremely important purpose. For example, if you were to perform a rigorous mathematical computation and save the results into a file, you might want to execute this code on the host VI. The host VI will receive data from the FPGA, perform the mathematical computation, and save the results in a file. More information about host VI communication is shown in Example 3.

Example 2: LCD

This example demonstrates how to compile an FPGA VI. This example also demonstrates how to use the LCD on the Spartan-3E Starter Board. The FPGA waits for characters being sent via the front panel of the VI on the host computer. The characters are then displayed on the LCD on the Spartan-3E.

1. Open the LabVIEW project (.lvproj) located at <LabVIEW Dir>\examples\Spartan-3E\LCD\fpgaLCD.lvproj
2. In the **Project Explorer** window, expand the plus sign located near FPGA target name.
3. Double-click the PadWrite VI under the FPGA target to open the front panel for the example. Press <Ctrl>-E to open the block diagram for further observation.
4. Hold down the <Ctrl> key on your keyboard while pressing the **Run** arrow in LabVIEW. This will trigger the VI to be compiled. An intermediate file conversion will take place as shown in Figure 5. Note that you may be prompted to save changes. Save the changes and continue with the compile process.


Generating Intermediate Files

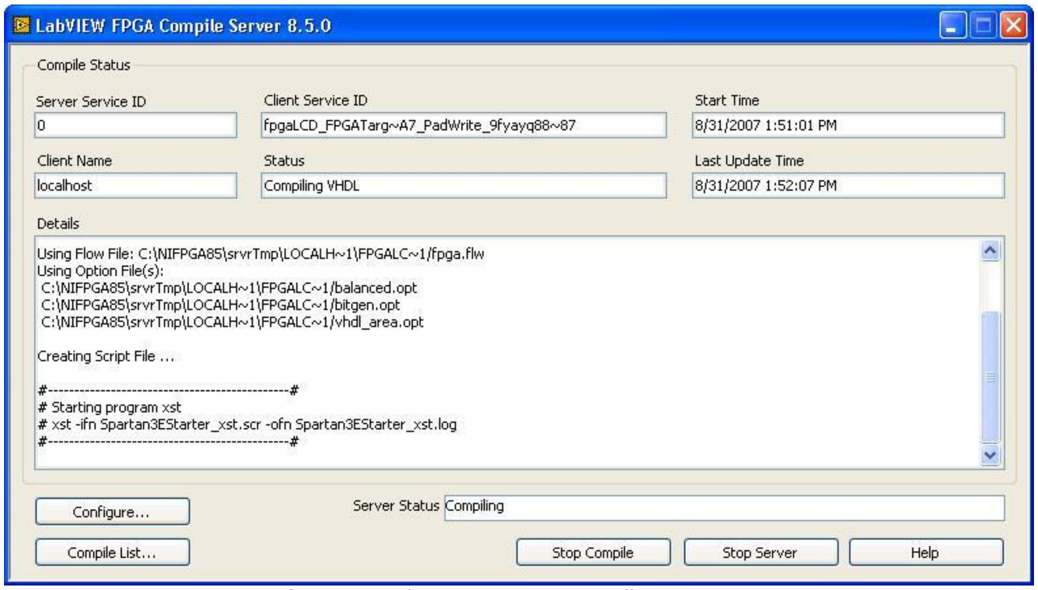
Generating intermediate files.
This may take a few minutes...
Stage 2 of 6

Generating preliminary files 0 of 1

Total 12%

Figure 5: Generating Intermediate Files

5. The LabVIEW FPGA Compile Server 8.5 will show the progress of the compilation. Note that it may take a few moments for the information to populate the fields. The LabVIEW code is now being compiled to run on the Spartan-3E. The status and output of the conversion is shown in Figure 6.


LabVIEW FPGA Compile Server 8.5.0

Compile Status

Server Service ID: 0
Client Service ID: fpgaLCD_FPGATarg~A7_PadWrite_9fyayq88~87
Start Time: 8/31/2007 1:51:01 PM
Client Name: localhost
Status: Compiling VHDL
Last Update Time: 8/31/2007 1:52:07 PM

Details

```
Using Flow File: C:\NIFPGA85\srvrTmp\LOCALH~1\FPGALC~1/fpga.flw
Using Option File(s):
 C:\NIFPGA85\srvrTmp\LOCALH~1\FPGALC~1/balanced.opt
 C:\NIFPGA85\srvrTmp\LOCALH~1\FPGALC~1/bitgen.opt
 C:\NIFPGA85\srvrTmp\LOCALH~1\FPGALC~1/vhdl_area.opt

Creating Script File ...

#----------------------------------------------#
# Starting program xst
# xst -ifn Spartan3EStarter_xst.scr -ofn Spartan3EStarter_xst.log
#----------------------------------------------#
```

Server Status: Compiling

Figure 6: LabVIEW FPGA Compile Server 8.5

6. Once the compilation is complete, the **Server Status** will indicate that it is in "Idle" mode. Open the PadWriteHost VI.
7. Click the **Run** arrow to download the code to the FPGA. Wait for the **Init Done** LED on the front panel to be ON to confirm that the initialization is finished as shown in Figure 7.
8. Click the letters "H", "E", "L", "L", "O" to display HELLO on the Spartan-3E LCD.
9. Click the **Quit** button to stop execution.
10. Close the project.

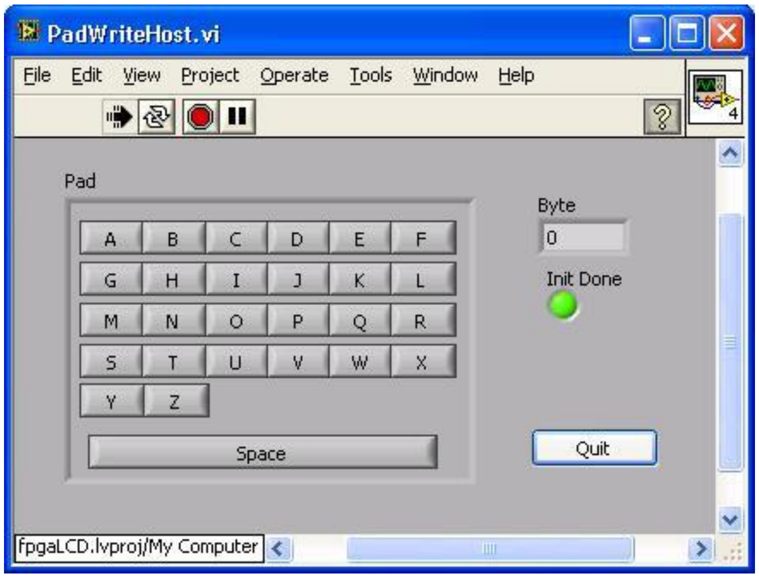


Figure 7: PadWriteHost VI

Example 3: Creating a Spartan-3E Project Example

In this project, we will learn how to target the Spartan-3E Starter Board, create LabVIEW FPGA I/O, compile our code, and talk to our FPGA target from our host VI.

Creating a Spartan-3E Target

1. Open LabVIEW 8.5. To create an empty project, click **Empty Project** on the **Getting Started** window or go to **File »New Project**.
2. Right-click **My Computer** under the project name and go to **New»Targets and Devices** as shown in Figure 8 below.

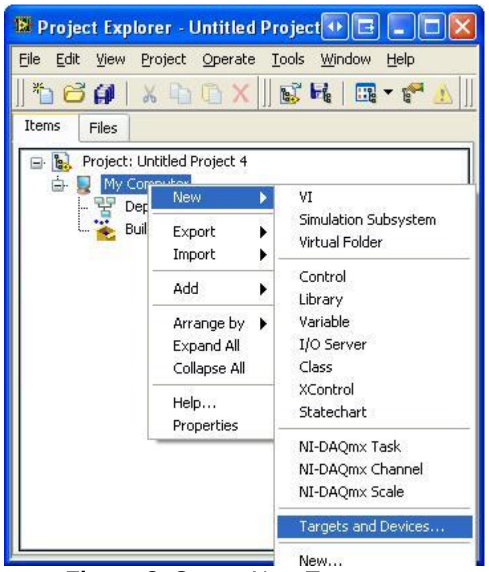
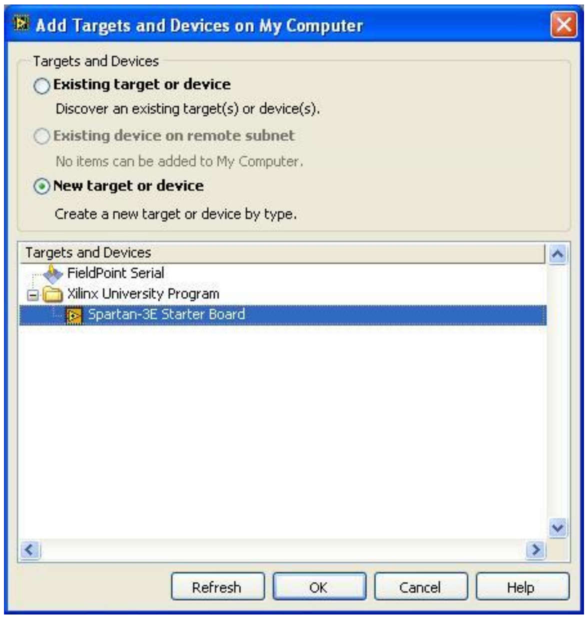


Figure 8: Create New Target

3. With the **Add Targets and Devices** dialog box open, select the **New target or device** radio button. Under **Targets and Devices**, expand **Xilinx University Program** and select the **Spartan-3E Starter Board** as the target. Your configuration should look similar to Figure 9. Press **OK** to confirm.


Add Targets and Devices on My Computer

Targets and Devices

[ ] **Existing target or device**
Discover an existing target(s) or device(s).

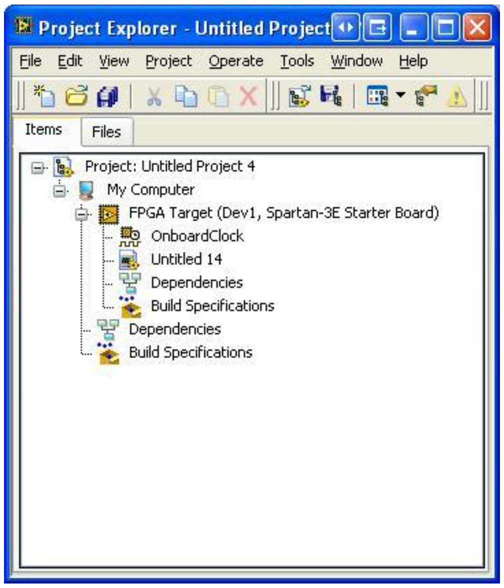
[ ] **Existing device on remote subnet**
No items can be added to My Computer.

[✓] **New target or device**
Create a new target or device by type.

Targets and Devices
- FieldPoint Serial
- Xilinx University Program
  - Spartan-3E Starter Board

Figure 9: Add Spartan-3E Target

4. Your new project should now contain the Spartan-3E Starter Board as a target. To create an FPGA VI, right-click the name **FPGA target (Dev1, Spartan-3E Starter Board)** and select **New»VI**. Your project should look similar to Figure 10. Note that your new VI is under the FPGA tree. This indicates that the VI belongs to the FPGA target and not the host computer. This VI will be compiled to VHDL code and then downloaded via USB to the Spartan-3E target.


Project Explorer - Untitled Project

- Project: Untitled Project 4
  - My Computer
    - FPGA Target (Dev1, Spartan-3E Starter Board)
      - OnboardClock
      - Untitled 14
      - Dependencies
      - Build Specifications
    - Dependencies
    - Build Specifications

Figure 10: Create New FPGA VI

Creating FPGA I/O

1. To create FPGA I/O, right-click the name **FPGA target (Dev1, Spartan-3E Starter Board)** and select **New»FPGA I/O** as shown in Figure 11.

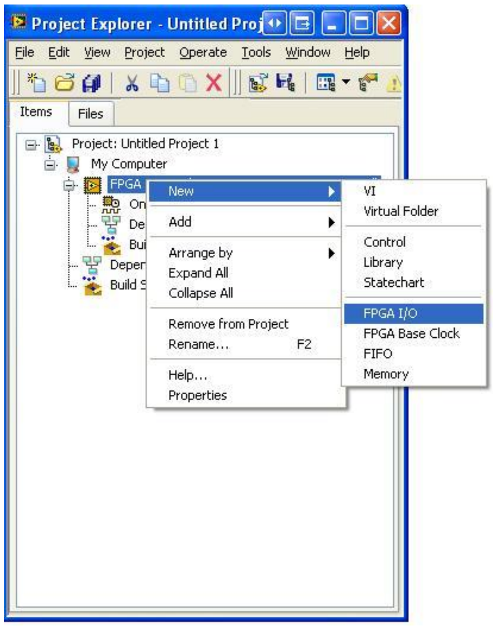
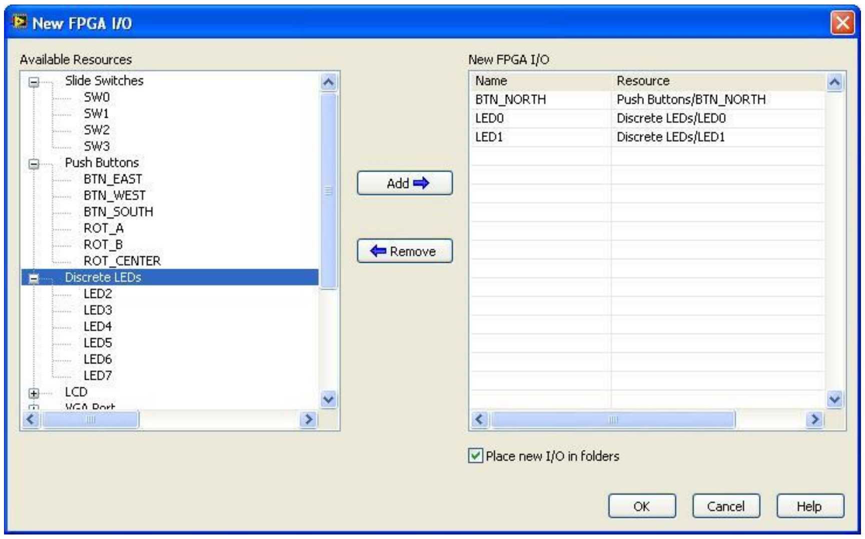


Figure 11: Create New FPGA I/O

2. Expand the **Push Buttons** and **Discrete LEDs** resources.
3. Select **BTN_NORTH** under the **Push Buttons** resources and click the **Add** button to add the button to the FPGA I/O.
4. Select **LED0** and **LED1** under the **Discrete LEDs** resource and click the **Add** button to add the LEDs to the FPGA I/O. Your New FPGA I/O should look similar to Figure 12.
5. Click the **OK** button. Your new project should now contain two folders that access the I/O on the card.



Name	Resource
BTN_NORTH	Push Buttons/BTN_NORTH
LED0	Discrete LEDs/LED0
LED1	Discrete LEDs/LED1

Figure 12: New FPGA I/O

Adding FPGA I/O to the Block Diagram

1. Right-click the name **FPGA target (Dev1, Spartan-3E Starter Board)** and select **New»VI.**
2. While the block diagram of your VI is open, drag and drop BTN_NORTH from the I/O folder in the **Project Explorer** window to the block diagram. Drag and drop the remaining two LEDs (LED0 and LED1) onto the block diagram. Your FPGA VI should look similar to Figure 13.

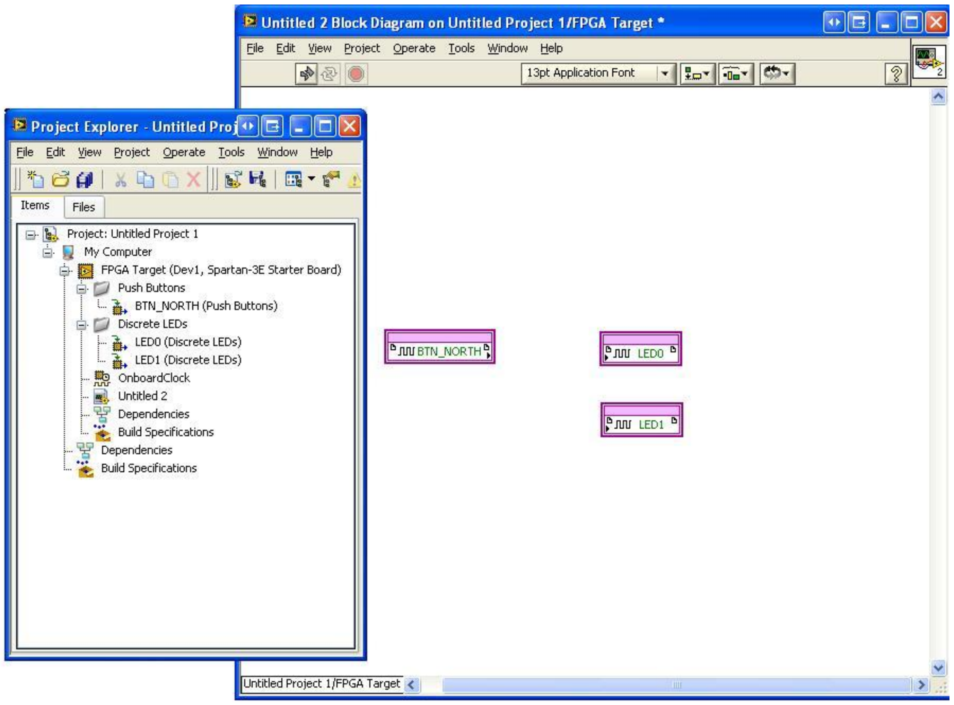

Untitled 2 Block Diagram on Untitled Project 1/FPGA Target *
File Edit View Project Operate Tools Window Help
13pt Application Font
Project Explorer - Untitled Proj
Items
Files
Project: Untitled Project 1
My Computer
FPGA Target (Dev1, Spartan-3E Starter Board)
Push Buttons
BTN_NORTH (Push Buttons)
Discrete LEDs
LED0 (Discrete LEDs)
LED1 (Discrete LEDs)
OnboardClock
Untitled 2
Dependencies
Build Specifications
Dependencies
Build Specifications
BTN_NORTH
LED0
LED1
Untitled Project 1/FPGA Target

Figure 13: Adding I/O to Block Diagram

3. Wire the **BTN_NORTH** output to **LED0**. The North button output will now be displayed on LED0 on the Spartan-3E Starter Board. Right-click the output terminal of BTN_NORTH and create an indicator.
4. Right-click the input terminal of LED1 and create a control.
5. Drop a While Loop around all of the code on the block diagram by clicking and dragging the loop to expand the loop. The While Loop is located on the **Functions** palette under **Programming»Structures**.
6. Right-click the Conditional Terminal of the While Loop and create a FALSE constant.
7. Drop a Wait function in the While Loop. The Wait function is located on the **Functions** palette under **Programming»Timing**. In the dialog, change the **Counter Units** to msec. Click the **OK** button. Wire a constant of “75” to the Wait function. Your FPGA VI should look similar to Figure 14. Save your FPGA VI as “FPGAVI”.

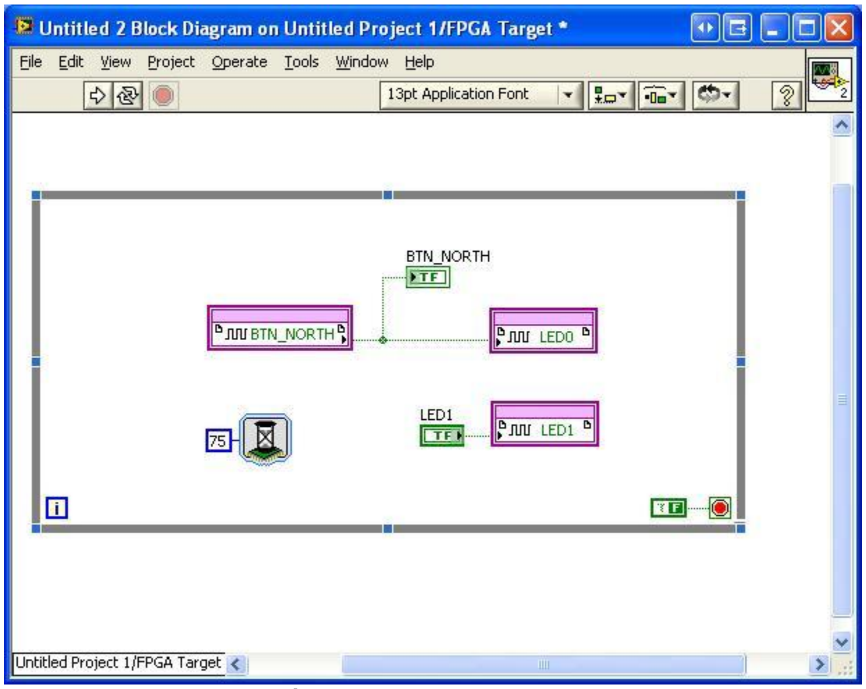
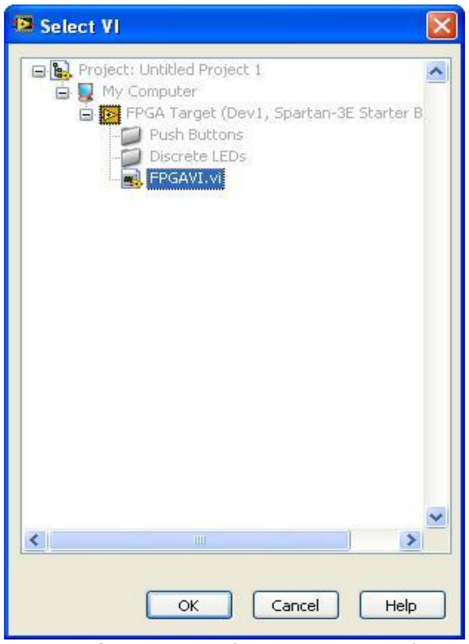

Untitled 2 Block Diagram on Untitled Project 1/FPGA Target *
File Edit View Project Operate Tools Window Help
13pt Application Font
BTN_NORTH
BTN_NORTH
LED0
LED1
LED1
75
Untitled Project 1/FPGA Target

Figure 14: FPGA VI

Creating a Host VI

1. Right-click **My Computer** in the **Project Explorer** window and select **New»VI**. Notice how the new VI is under the **My Computer** tree, indicating that this VI belongs to the host computer and not the Spartan-3E.
2. On the block diagram, place an Open FPGA VI Reference function. The Open FPGA VI Reference function is located on the **Functions** palette under **FPGA Interface**. The Open FPGA VI Reference is used to reference the FPGA VI located on the target FPGA.
3. Right-click the Open FPGA VI Reference function and click **Select VI** to access the FPGA VI. In the **Select VI** dialog box, select FPGAVI, as shown in Figure 15. Click the **OK** button.


Select VI
Project: Untitled Project 1
My Computer
FPGA Target (Dev1, Spartan-3E Starter B
Push Buttons
Discrete LEDs
FPGAVI.vi
OK
Cancel
Help
Figure 15: Select FPGA VI Reference

4. Place a Read/Write Control function to the right of the Open FPGA VI Reference function. Wire the **FPGAVI Reference Out** terminal from the Open FPGA VI Reference function to **FPGA VI Reference In** terminal on the Read/Write Control function. Wire the **error out** terminal on the Open FPGA VI Reference function to the **error in** terminal on the Read/Write Control function.
5. Place a Close FPGA VI Reference function to the right of the Read/Write Control function. Wire the **FPGA VI Reference Out** terminal from the Read/Write Control function to the **FPGA VI Reference In** terminal on the Close FPGA VI Reference function. Wire the **error out** terminal on the Read/Write Control function to the **error in** terminal on the Close FPGA VI Reference function.
6. Click the **Unselected** box on the Read/Write Control function and select LED1. Right-click the Read/Write Control function and select **Add Element**. Click the second element and select BTN_NORTH. Your two elements should be LED1 and BTN_NORTH.
7. Right-click the input terminal of LED1 and create a control.
8. Right-click the output terminal of BTN_NORTH and create an indicator.
9. Place a While Loop around the Read/Write Control function. Right-click the **Loop Condition** terminal and create a control to stop the While Loop.
10. Place a Wait (ms) function in the While Loop. Right-click the **milliseconds to wait** input terminal and create a constant of 75.
11. Your host VI should look similar to Figure 16. Save your host VI as "FPGAHost".

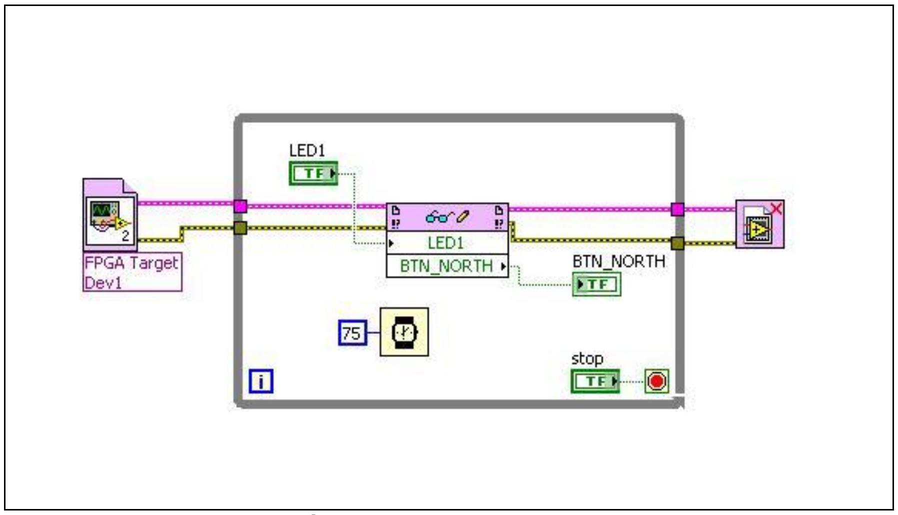
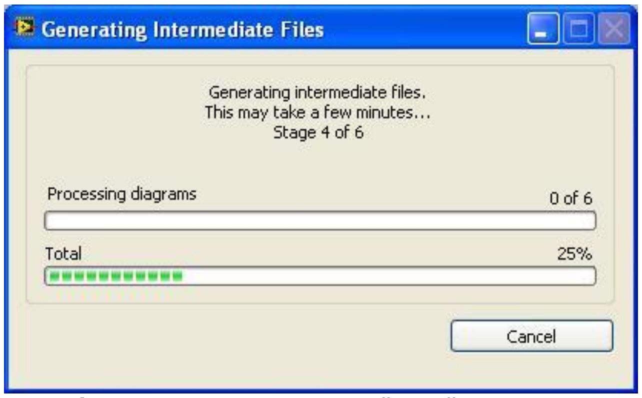


Figure 16: LV FPGA Host VI

Compiling Code

1. Open the FPGAVI.vi if not already open. While pressing the <Ctrl> key, press the **Run** button to compile the FPGA code. An alternate method of compiling is to right-click the VI in the **Project Explorer** window and select **Compile**. Note that LabVIEW might ask you to save your project. Save your project as Push LED Test. The LabVIEW code is now being translated to intermediate files as shown in Figure 17.



Generating Intermediate Files

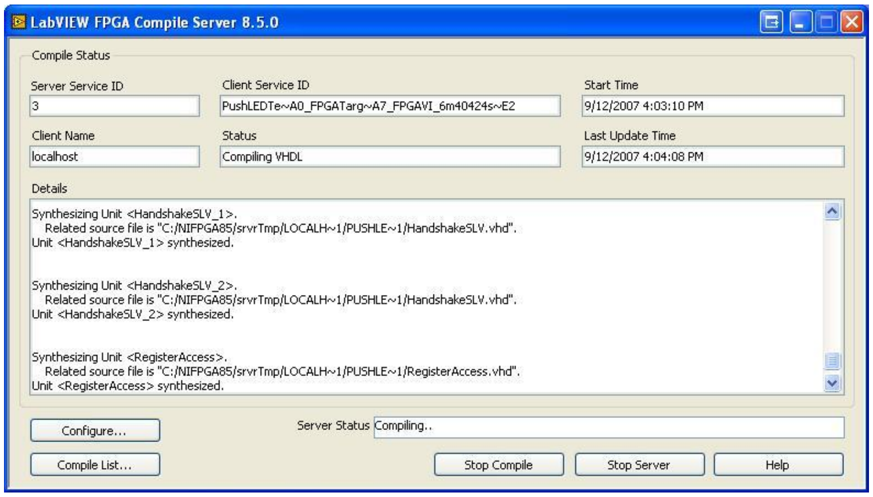
Generating intermediate files.
This may take a few minutes...
Stage 4 of 6

Processing diagrams 0 of 6

Total 25%

Figure 17: Generate Intermediate Files

2. After generating the intermediate files, the LabVIEW FPGA Compile Server 8.5.0 will open as shown in Figure 18.



Server Service ID: 3
Client Service ID: PushLEDTe~A0_FPGATarg~A7_FPGAVI_6m40424s~E2
Start Time: 9/12/2007 4:03:10 PM
Client Name: localhost
Status: Compiling VHDL
Last Update Time: 9/12/2007 4:04:08 PM

Details

```
Synthesizing Unit <HandshakeSLV_1>.
   Related source file is "C:/NIFPGA85/srvrTmp/LOCALH~1/PUSHLE~1/HandshakeSLV.vhd".
Unit <HandshakeSLV_1> synthesized.


Synthesizing Unit <HandshakeSLV_2>.
   Related source file is "C:/NIFPGA85/srvrTmp/LOCALH~1/PUSHLE~1/HandshakeSLV.vhd".
Unit <HandshakeSLV_2> synthesized.


Synthesizing Unit <RegisterAccess>.
   Related source file is "C:/NIFPGA85/srvrTmp/LOCALH~1/PUSHLE~1/RegisterAccess.vhd".
Unit <RegisterAccess> synthesized.
```

Server Status: Compiling..

Figure 18: Compile FPGA Code

3. Once the **Server Status** is Idle, close the Compile Server and run the host VI. This VI will open a FPGA reference to the FPGA on the Spartan-3E. Press the BTN_NORTH button to the left of the LCD and notice that the host VI will update the status of the button. Also notice that LED0 to the right of the LCD lights up once the button is pressed.

Example 4: WebPACK IP Core Integration with LabVIEW FPGA (Advanced)

On the FPGA platform, reusable code modules are commonly called IP (intellectual property) cores, the name having evolved over time from traditional chip design. A core is a subcomponent of a HDL design. It provides functionality such as filtering or implementing communication protocols. You can obtain cores from a variety of sources and many are available free of charge. Developing code modules based on proven designs enables better reuse of existing IP in future applications and sharing and exchange of code between different developers and internal organizations.

In this example, we will integrate a Xilinx WebPACK IP Core with the LabVIEW FPGA Module. The hybrid programming approach will use both VHDL and LabVIEW FPGA code to filter an analog input signal. The analog input code will be written using the LabVIEW FPGA Module, and the IP Core will be an arithmetic distributed filter generated by the Xilinx WebPACK.

WebPACK IP Core Generation and Integration

To generate the IP core, make sure to install the Xilinx ISE WebPACK tool and all of the appropriate service packs, as described in the Software Installation section above. This document describes how to generate and integrate an IP core into a LabVIEW application using the HDL Interface Node in the LabVIEW FPGA Module.

This application instantiates a Xilinx distributed-arithmetic, 10 kHz lowpass, FIR filter core in the LabVIEW FPGA Module using the HDL Interface Node. Documentation for this core can be found by clicking the following Xilinx Distributed Arithmetic Filter Core Documentation link:

http://www.xilinx.com/ipcenter/catalog/logicore/docs/da_fir.pdf

Designing Filter with Coefficients

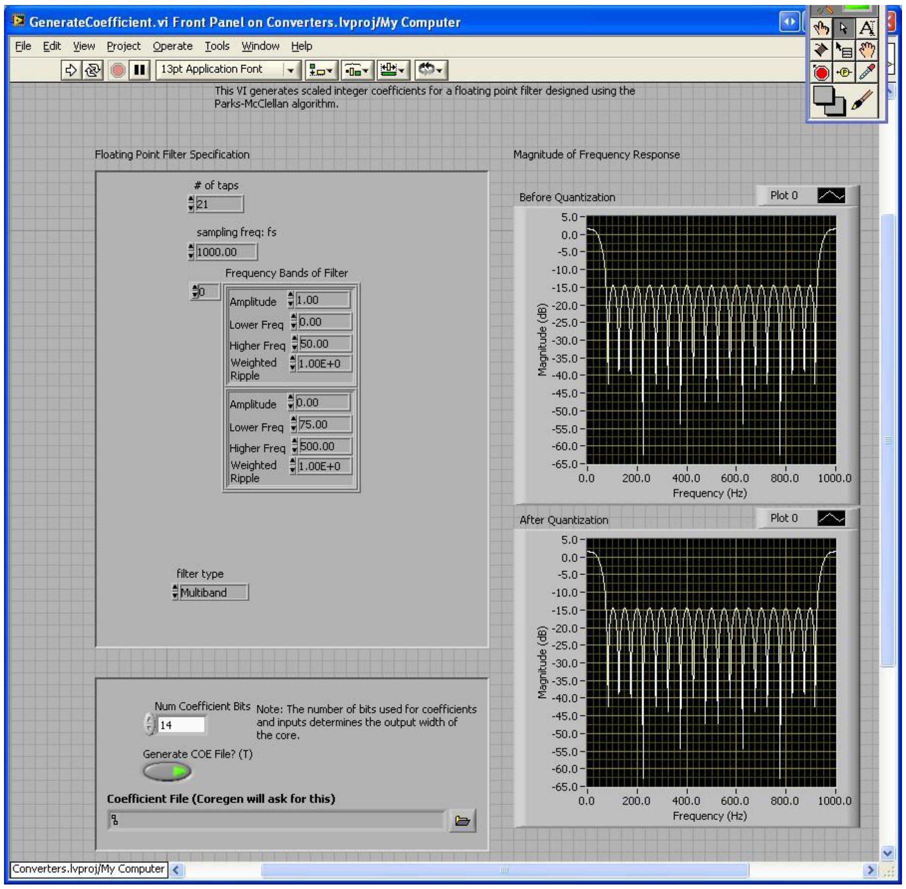
The first step to use the filter core is to generate the filter coefficients and store them in a .coe file format. You will be able to load this file into the configurator of the filter core. You can generate the coefficients manually or can use various tools to help with this process. You may use the Generate Coefficient example VI in the following DeveloperZone article:

Integrating IP Cores into LabVIEW FPGA with the HDL Interface Node-
http://zone.ni.com/devzone/cda/tut/p/id/3516

The file is located in the Filter_Design.zip attached to the DeveloperZone article. Open GenerateCoefficient.vi. Notice that the front panel control defaults are set to generate a 10 kHz bandstop filter. The VI also quantizes the generated coefficients and scales them to integers. Change the number of taps to be 21. Change the sampling frequency to 1000 Hz. Finally, change the frequency bands of the filter to be two elements with values as follows:

Element 0- Amplitude:1, Lower Freq: 0, Higher Freq: 50, Weighted Ripple: 1
Element 1- Amplitude:0, Lower Freq: 75, Higher Freq: 1000, Weighted Ripple: 1

Your Generate Filter Coefficients front panel should look similar to Figure 19.


GenerateCoefficient.vi Front Panel on Converters.lvproj/My Computer

This VI generates scaled integer coefficients for a floating point filter designed using the Parks-McClellan algorithm.

Floating Point Filter Specification

# of taps: 21

sampling freq: fs: 1000.00

Frequency Bands of Filter: 0

Amplitude	1.00
Lower Freq	0.00
Higher Freq	50.00
Weighted Ripple	1.00E+0
Amplitude	0.00
Lower Freq	75.00
Higher Freq	500.00
Weighted Ripple	1.00E+0

filter type: Multiband

Num Coefficient Bits: 14

Note: The number of bits used for coefficients and inputs determines the output width of the core.

Generate COE File? (T)

Coefficient File (Coregen will ask for this)

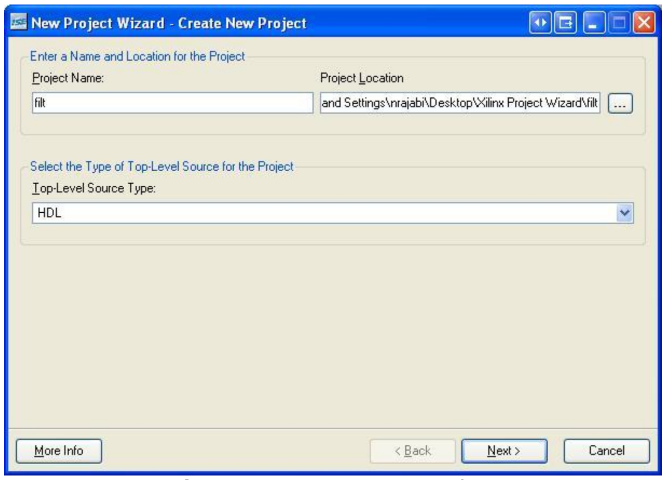
Figure 19: Filter Parameters

Run the VI and save the coefficients into file named cutoff.coe. The resulting response is shown on the front panel. It is always a good idea to check the filter response after quantization.

Generating the IP Core Netlist

Now that you have generated the filter coefficients, you are ready to configure the core. Launch the Xilinx Project Navigator from the Xilinx ISE 9.2i program group. Follow the steps below to configure the core and generate the netlist.

1. Select **File»New Project** to open the New Project Wizard. Name your project "filt" and select HDL for **Top-Level Source Type**. Your dialog box configuration should look similar to Figure 20. Click the **Next** button to continue.

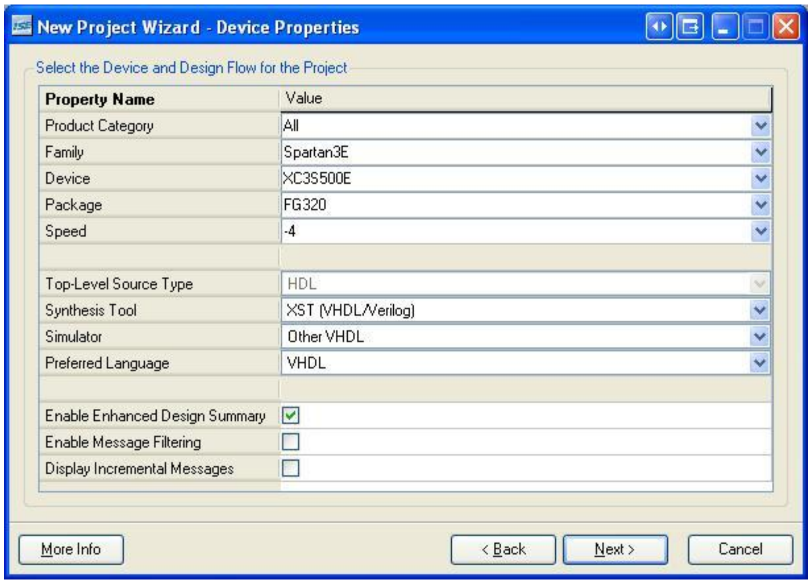


New Project Wizard - Create New Project
Enter a Name and Location for the Project
Select the Type of Top-Level Source for the Project
More Info
< Back
Next >
Cancel

Project Name:	filt
Project Location	and Settings\nrajabi\Desktop\Xilinx Project Wizard\filt
Top-Level Source Type:	HDL

Figure 20: Create New Project

2. Set the values to match those shown below in Figure 21 and click the **Next** button.



New Project Wizard - Device Properties
Select the Device and Design Flow for the Project
More Info
< Back
Next >
Cancel

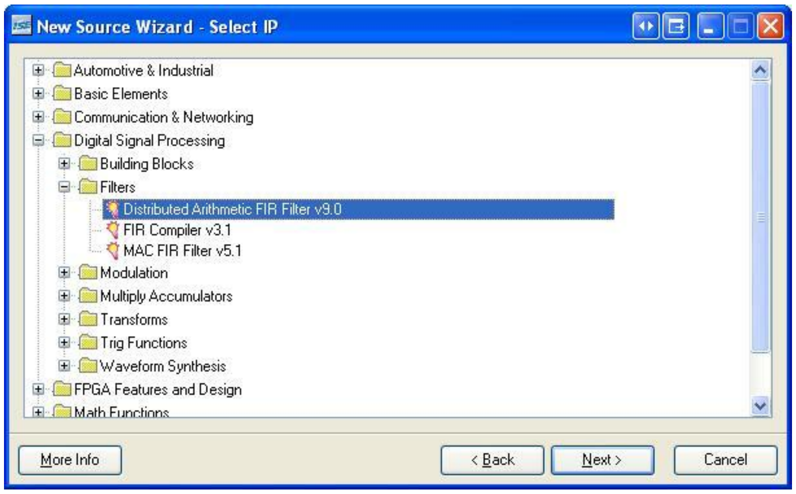
Property Name	Value
Product Category	All
Family	Spartan3E
Device	XC3S500E
Package	FG320
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Other VHDL
Preferred Language	VHDL
Enable Enhanced Design Summary	✓
Enable Message Filtering	☐
Display Incremental Messages	☐

Figure 21: Device Configuration

3. When prompted to "Create a New Source", press the **New Source** button.

4. Select IP (CoreGen & Architecture Wizard) as the type and name the new source "filt". Make sure that there is a checkmark next to **Add to project** to add the IP Core to the project. Click the **Next** button.

5. Expand the **Digital Signal Processing** category. Expand the **Filters** subcategory and locate **Distributed Arithmetic FIR Filter v9.0**. Your new Source Wizard – Select IP should look similar to Figure 22 below. Click the **Next** button.



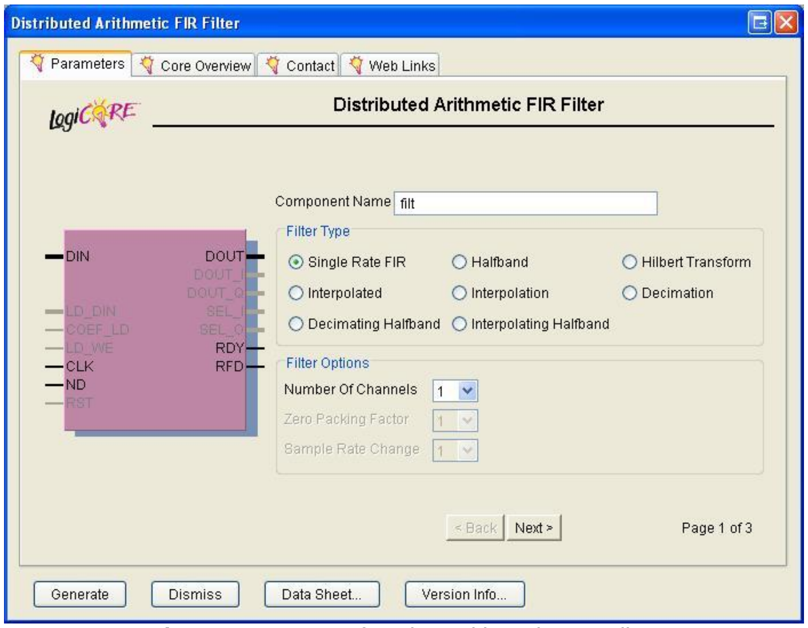
New Source Wizard - Select IP

- Automotive & Industrial
- Basic Elements
- Communication & Networking
- Digital Signal Processing
  - Building Blocks
  - Filters
    - Distributed Arithmetic FIR Filter v9.0
    - FIR Compiler v3.1
    - MAC FIR Filter v5.1
  - Modulation
  - Multiply Accumulators
  - Transforms
  - Trig Functions
  - Waveform Synthesis
- FPGA Features and Design
- Math Functions

Figure 22: Select IP

6. Click **Finish** to complete the new source wizard. You may be prompted to create a new directory if one does not already exist. Click **Yes** to create the directory. Click **Next** to continue the process. Click **Next** again to continue the process. Click **Finish** to complete the process.

7. A new dialog box should display the characteristics of the Distributed Arithmetic FIR Filter. In the **Component Name** field, type in "filt" if not already populated. For the **Filter Type**, select **Single Rate FIR**. The number of channels should be "1". Your new Distributed Arithmetic FIR Filter page 1 of 3 should look similar to Figure 23 below. Click **Next** when confirmed.



Distributed Arithmetic FIR Filter

Component Name: filt

Filter Type: [✓ Single Rate FIR] Halfband Hilbert Transform Interpolated Interpolation Decimation Decimating Halfband Interpolating Halfband

Filter Options

Number Of Channels: 1

Zero Packing Factor: 1

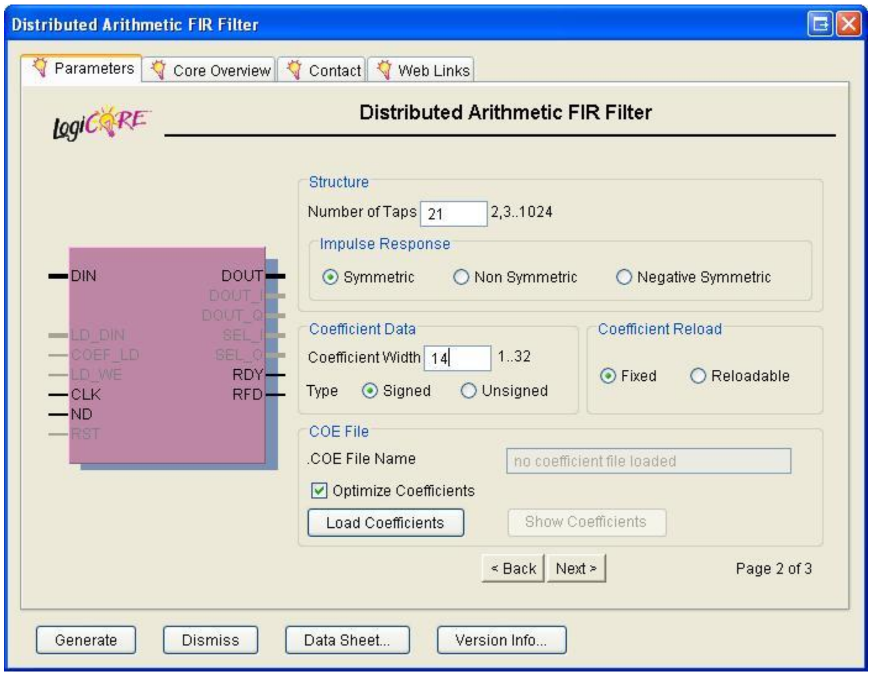
Sample Rate Change: 1

Page 1 of 3

Figure 23: Step 1 of 3 Dist. Arithmetic FIR Filter

8. For **Number of Taps**, type in “21”. The **Impulse Response** should be **Symmetric**. For **Coefficient Width** type in “14”.

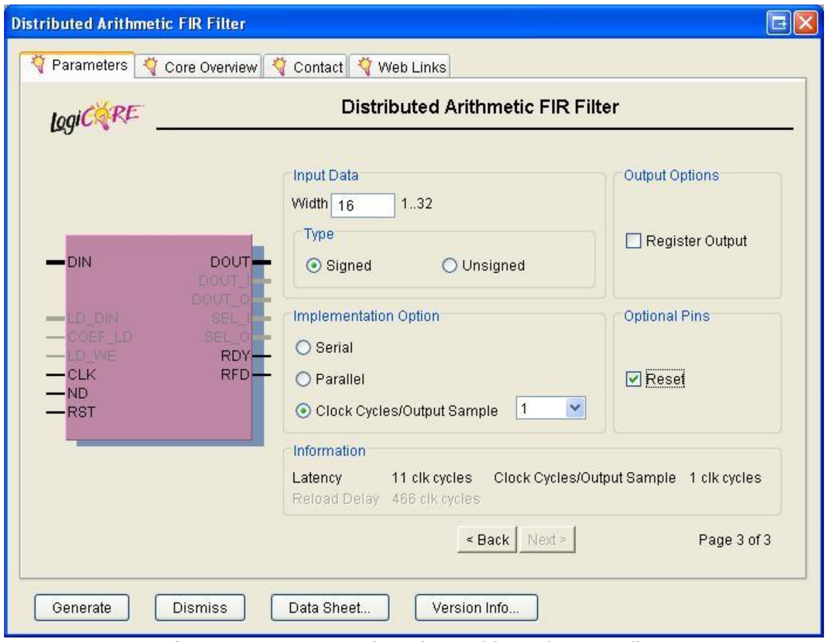
9. To load your coefficient file, click the **Load Coefficient** button. Select your .coe file that you have generated. Your new Distributed Arithmetic FIR Filter page 2 of 3 should look similar to Figure 24 below. Click the **Next** button once finished.



Structure
Number of Taps 21 2,3..1024
Impulse Response: [✓ Symmetric] Non Symmetric Negative Symmetric
Coefficient Data
Coefficient Width 14 1..32
Type [✓ Signed] Unsigned
Coefficient Reload: [✓ Fixed] Reloadable
COE File
.COE File Name: no coefficient file loaded
☑ Optimize Coefficients
Page 2 of 3

Figure 24: Step 2 of 3 Dist. Arithmetic FIR Filter

10. For the **Input Data Width**, type in “16”. On the **Optional Pins**, place a checkmark next to **Reset**. Your new Distributed Arithmetic FIR Filter page 3 of 3 should look similar to Figure 25 below. Click the **Generate button** to generate the FIR Filter.



Input Data
Width 16 1..32
Type
[✓ Signed] Unsigned

Output Options
Register Output

Implementation Option
Serial
Parallel
[✓ Clock Cycles/Output Sample] 1

Optional Pins
[✓ Reset]

Information
Latency 11 clk cycles Clock Cycles/Output Sample 1 clk cycles
Reload Delay 466 clk cycles

Figure 25: Step 3 of 3 Dist. Arithmetic FIR Filter

11. Your Distributed Arithmetic FIR Filter IP Core has now been generated. A whole list of files have been generated, one of which is called a Netlist. The Netlist is sometimes saved in binary format under the .ngc file extension. The Netlist is the heart of your IP Core. The VHDL code wrapper that calls the IP Core has also been generated.

Integrating IP Core with LabVIEW FPGA

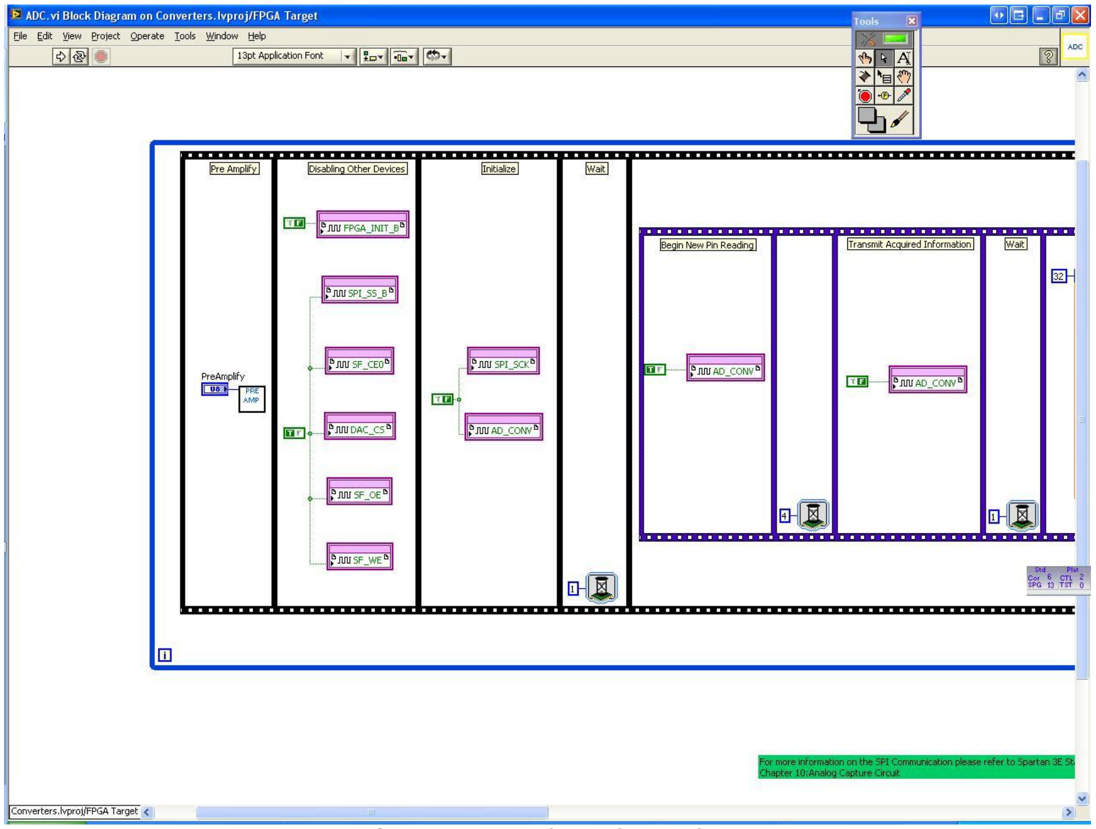
We are now going to learn how to integrate your IP Core into the LabVIEW FPGA Module. The following steps will discuss how to use the files you just created to configure the HDL Interface Node. Your ADC/Filter integration will be built upon an example for the Spartan-3E Starter Board.

1. Launch LabVIEW 8.5. Open the Converter project located at <LabVIEW Dir>\examples\Spartan-3E\Converters\Converters.lvproj.

2. Select **File»Save As**. Select **Duplicate .lvproj file and contents** to create a new project based on the example. Save the new project as “IPCoreFilt”.

3. Expand the **FPGA Target name (Dev1, Spartan-3E Starter Board)** and double click the **ADC.vi**.

4. Go to the block diagram of the ADC.vi. Notice that there is an outer While Loop with a Case structure determining when to start the acquisition. Since we want the program to acquire data right away from the ADC, we are now going to remove the outer While Loop and Case structure. Right-click the outer While Loop and select **Remove While Loop**. Next, right-click the outer Case structure and select **Remove Case Structure**. Remove any broken wires from the case as well as any unused controls. Your left part of the block diagram should look similar to Figure 26.



For more information on the SPI Communication please refer to Spartan 3E St
Chapter 10:Analog Capture Circuit

Figure 26: Outer loop of ADC.vi

5. Scroll all the way right on the block diagram. Notice that there is an indicator for both ADC channels. Delete the **Output** indicator.

6. Expand the outer While Loop and the outer Flat Sequence structure. Create an indicator on the output of the inner Flat Sequence structure that is connected to the Boolean output. Name the indicator "**Non Filtered Signal**".

7. Place a Split 1D Array function inside the open area of the expanded Flat Sequence structure. The Split 1D Array function is located under **Programming»Array**. Wire the Boolean array output into the **array** input of the Split 1D Array function. Wire a constant of 16 into the index of the Split 1D Array function. Since the ADC combines both channels A and B into a 32-bit number, we have to cut the array by half to retrieve only the first channel.

8. Place a Reverse 1D Array function to the right of the Split 1D Array function. The Reverse 1D Array is located under **Programming»Array**. Wire the first subarrary output of the Split 1D Array function into the Reverse 1D Array function.

9. Place a Boolean Array to Number function to the right of the Reverse 1D Array function. The Boolean Array to Number function is located under **Programming» Numeric»Conversion**. Wire the revered array output into the Boolean array input.

10. Place a To Word Integer function to the right of the Boolean Array to Number function. The To Word Integer function is located under **Programming» Numeric»Conversion**. Wire the output of the Boolean Array to Number function to the input of the To Word Integer function.

11. Your block diagram should look similar to Figure 27. We are now going to integrate the IP Core into your block diagram.

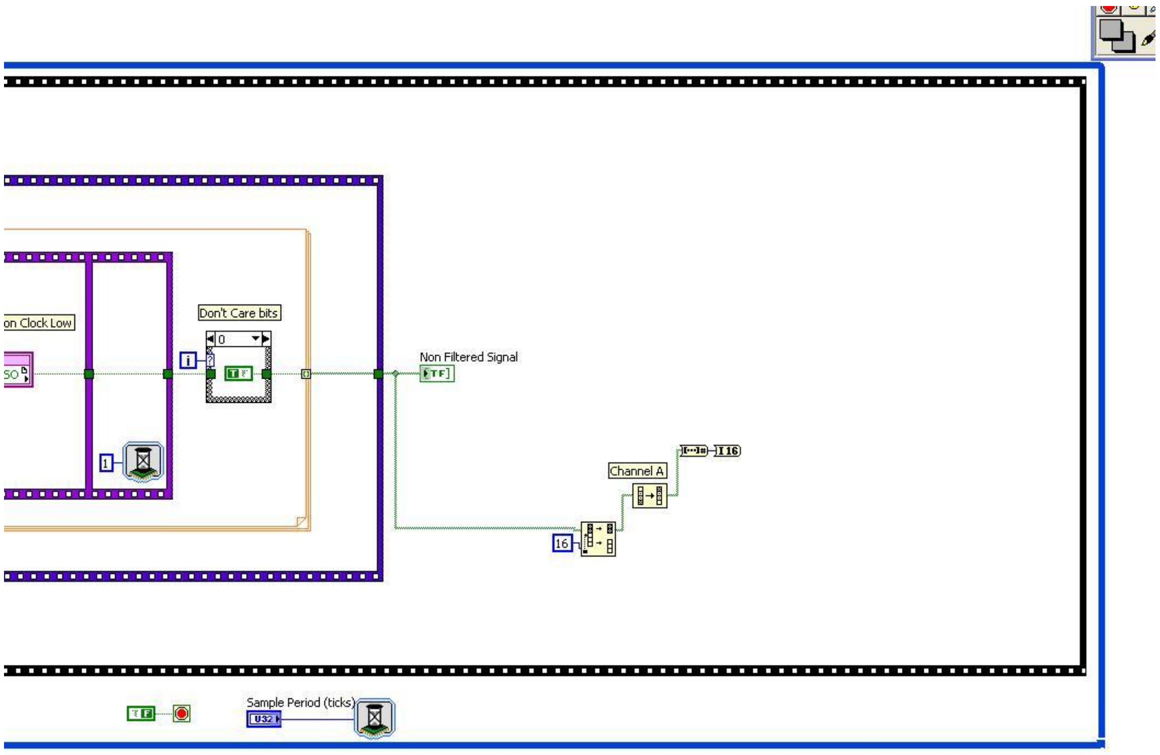
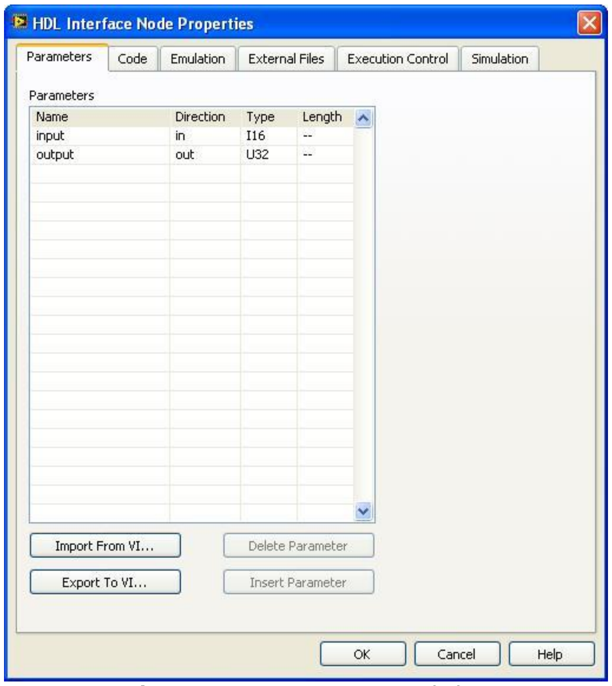


Figure 27: Show Connector ADC

Integrating IP Core with LabVIEW FPGA: HDL

1. Place an HDL Interface Node to the right of the To Word Integer function. The HDL Interface Node is located under **Programming»Advanced»HDL Interface Node.**

2. Double-click the HDL Interface Node to display the **HDL Interface Node Properties** dialog box. On the **Parameters** tab, double-click the first element under name and call it “input”. Select **in** as a direction and **I16** as its type. Double-click the second element under name and call it “output”. Select **out** as a direction and **U32** as its type. The **Parameters** tab should look similar to Figure 28. Note that the name is case sensitive.



Parameters

Name	Direction	Type	Length
input	in	I16	--
output	out	U32	--

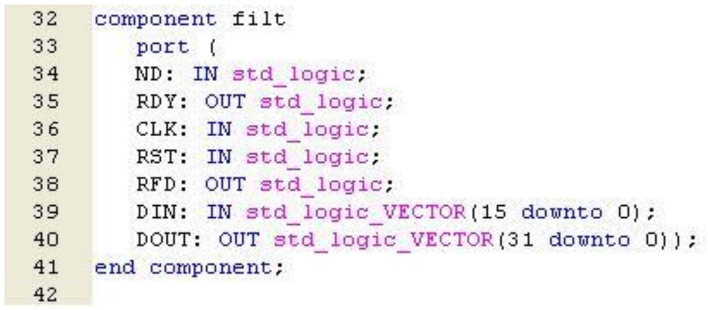
Figure 28: HDL Parameter Listing

3. Go to the **Code** tab to integrate the IP Core into LabVIEW. Note that we also must take care of the LabVIEW dataflow paradigm in the VHDL code. With LabVIEW, a node will only execute once all inputs are ready. We must then ensure that our VHDL code also satisfies the paradigm.

   To satisfy the LabVIEW dataflow paradigm, we must use intermediate signals called enable signals. Refer to the *LabVIEW Help* for more information about the HDL Interface Node and configuring the enable signals. In LabVIEW, select **Help»Search the LabVIEW Help** to display the *LabVIEW Help*. Using the **Contents** tab, navigate to the **FPGA Module»FPGA Module Concepts»Importing HDL Code into FPGA VIs** book for more information.

4. Go back to the Xilinx Project Window and open the project you created called "filt". Select **File»Open** and open the **filt.vho** code generated from the IP Core. Notice that the IP Core generation created a component called filt. It also created a port mapping for the filter created.

   Copy the section of code in filt.vho for the component:



```
component filt
   port (
   ND: IN std_logic;
   RDY: OUT std_logic;
   CLK: IN std_logic;
   RST: IN std_logic;
   RFD: OUT std_logic;
   DIN: IN std_logic_VECTOR(15 downto 0);
   DOUT: OUT std_logic_VECTOR(31 downto 0));
end component;
```

Go back to LabVIEW and the HDL Interface Node and paste the lines into the **architecture implementation of hdlnode is** section.

5. Paste the two lines of code for the enable signal and the output under the component you have just copied into the HDL Interface Node.

```
signal nd, rdy, rfd, prev_enable_in : std_logic;
signal dout : std_logic_vector( 31 downto 0 );
```

Once complete, the **architecture implementation of the hdlnode** should be as follows:

```
component filt

        port (
        ND: IN std_logic;
        RDY: OUT std_logic;
        RST: IN std_logic;
        CLK: IN std_logic;
        RFD: OUT std_logic;
        DIN: IN std_logic_VECTOR(15 downto 0);
        DOUT: OUT std_logic_VECTOR(31 downto 0));

end component;

signal nd, rdy, rfd, prev_enable_in : std_logic;
signal dout : std_logic_vector( 31 downto 0 );
```

6. We are now going to write the code for our implementation. The filter core interfaces nicely into the HDL Interface Node because there is a **nd** (new data) flag that tells the core when the current sample should be processed and there is a **rdy** (ready) flag that tells the user when data is available. This roughly corresponds to the LabVIEW FPGA enable_in and enable_out. The main difference is the enable signals are constant high when asserted while nd and rdy are pulsed. The code surrounding the core's instantiation creates a pulse for nd on the first clock cycle of an assertion of enable_in and then holds enable_out high when rdy asserts. Enable_out must be driven low when enable_clr asserts. This provides for looping capability. We are now going to create **delay_enable** process. Copy the code provided into the implementation section of the HDL Interface Node (last box in the HDL provided).

```
delay_enable_in:

process( clk )
begin
  if rising_edge(clk) then
    prev_enable_in <= enable_in;
  end if;
end process delay_enable_in;

nd <= enable_in and not(prev_enable_in);
```

7. Now we are going to create the filter instantiation and map our signals to the appropriate IP Core signals. Copy the following code for the filter into the implementation section of the HDL Interface Node:

```
your_inst: filt

your_inst: filt
  port map (
  ND => nd,
  RDY => rdy,
  CLK => clk,
  RST => reset,
  RFD => rfd,
  DIN => input,
  DOUT => dout );
```

8. We are now going to create an **enable_controller** process. This process is necessary because of the LabVIEW dataflow restrictions. Output data must be valid when enable_out asserts and must remain valid until enable_in deasserts or enable_clr asserts. Enable_out must only assert in response to enable_in asserting. Copy the following code in the instantiation for the filter into the implementation section of the HDL Interface Node:

```
enable_controller:

process( clk, reset )

begin

    if reset = '1' then
       enable_out <= '0';
    elsif rising_edge(clk) then
        if( enable_clr = '1' ) then
            enable_out <= '0';
        elsif( enable_in = '1' and rdy = '1' ) then
            enable_out <= '1';
        end if;
    end if;

end process enable_controller;

-- Output must be valid when enable_out asserts

output_register:
process( clk, reset )

begin
    if reset = '1' then
        output <= (others=>'0');
    elsif rising_edge(clk) then
        if( rdy = '1' ) then
            output <=  dout;
        end if;
    end if;
end process output_register;
```

9. Once complete, the implementation section of the HDL Interface Node should look as follows:

```
delay_enable_in:
process( clk )
begin
    if rising_edge(clk) then
        prev_enable_in <= enable_in;
    end if;
end process delay_enable_in;
nd <= enable_in and not(prev_enable_in);
```

```
your_inst: filt
   port map (
   ND => nd,
   RDY => rdy,
   CLK => clk,
   RST => reset,
   RFD => rfd,
   DIN => input,
   DOUT => dout );

enable_controller:
process( clk, reset )
begin
   if reset = '1' then
      enable_out <= '0';
   elsif rising_edge(clk) then
      if( enable_clr = '1' ) then
         enable_out <= '0';
      elsif( enable_in = '1' and rdy = '1' ) then
         enable_out <= '1';
      end if;
   end if;
end process enable_controller;

output_register:
process( clk, reset )
begin
   if reset = '1' then
      output <= (others=>'0');
   elsif rising_edge(clk) then
      if( rdy = '1' ) then
         output <=  dout;
      end if;
   end if;
end process output_register;
```

10. Click the **Check Syntax** button to check the syntax to confirm your VHDL code. Once successful, go to the **External Files** tab on the HDL Interface Node.

11. Click the **Add File** button to add the Netlist file of your IP Core. Locate the “filt.ngc” file created earlier and click the **OK** button to add the file to the HDL Interface Node. Click the **OK** button to close the HDL Interface Node.

12. Wire the output of the To Word Integer function to the **input** terminal on the HDL Interface Node.

13. Wire the **output** terminal on the HDL Interface Node to an indicator. Label the indicator "Filtered Signal". Part of the final ADC block diagram should look similar to Figure 29.

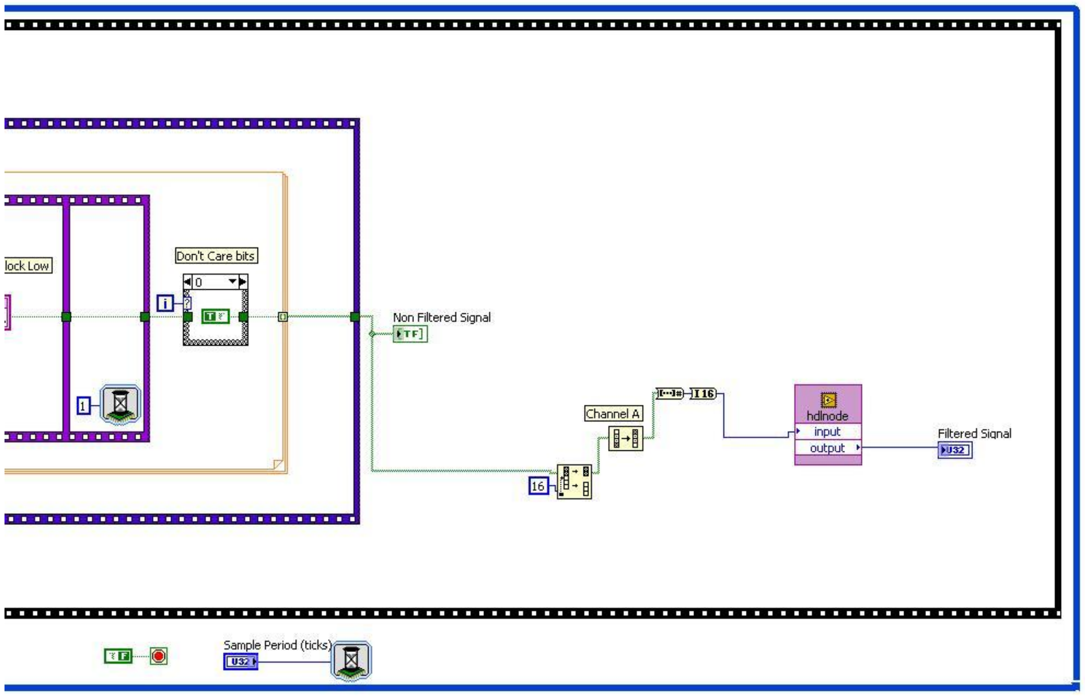


Figure 29: Final ADC VI

Completing the Project and Creating the Host VI

1. Go to the **Project Explorer** window and open the ADCHost VI**.** Your final ADCHost VI should look similar to Figure 30.

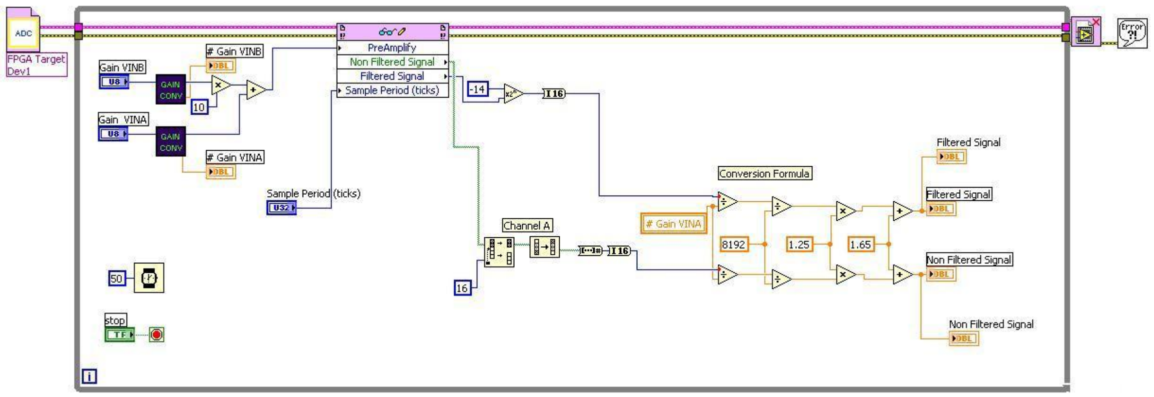


Figure 30: Final ADC VI

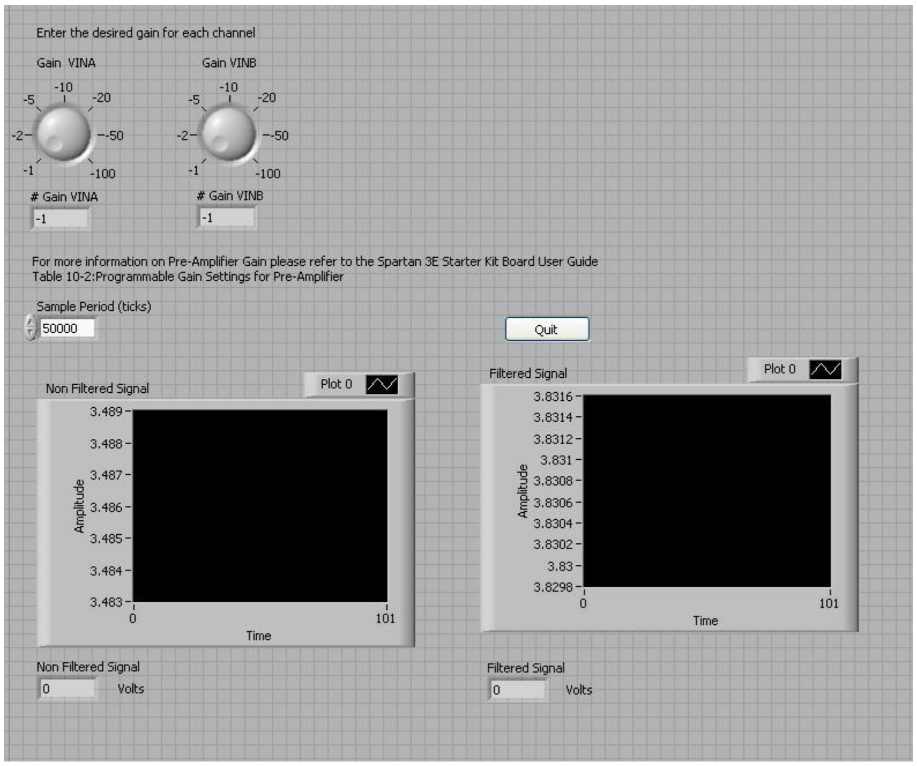
2. Change the Read/Write Control function to be four elements, starting from the top: **PreAmplify, Non Filtered Signal, Filtered Signal,** and **Sampling Period (ticks).**

3. Wire the Non Filtered Signal Output terminal to the Split 1D Array, Reverse 1D Array, Boolean to Number, and To Word Integer functions. Pass the output of the To Word Integer function to the x element of the bottom division terminal.

4. Delete the local variable called “# Gain VINB”. Wire “Gain VINA” to the y terminal of the bottom division terminal.

5. Wire the Filtered Signal Output terminal to the **x** input of a Scale By Factor of 2 function. On the **n** input, wire a constant of “-14”. Wire the output of the Scale By Factor of 2 function to a To Word Integer function**.** Wire the output of the To Word Integer function to the x terminal of the top division.

6. Go to the front panel and drop two waveform charts. Call one chart “Filtered Signal” and the other “Non Filtered Signal”. Change the **Sample Period** to 50000 ticks. On the block diagram, wire the Filtered Signal chart on the output of the top arithmetic conversion. On the block diagram, wire the Non Filtered Signal chart on the output of the bottom arithmetic conversion. Your front panel should look similar to Figure 31.



Enter the desired gain for each channel

Gain VINA: -1

Gain VINB: -1

For more information on Pre-Amplifier Gain please refer to the Spartan 3E Starter Kit Board User Guide
Table 10-2:Programmable Gain Settings for Pre-Amplifier

Sample Period (ticks): 50000

Non Filtered Signal: 0 Volts

Filtered Signal: 0 Volts

Figure 31: Final ADCHost VI Front Panel

Running Your Code

1. Open your TopLevel VI and hold the <Ctrl> key while pressing the **Run** arrow. This will trigger the VI to be compiled.

2. Once the compile process is complete, connect a square wave analog signal of 5Hz. Notice that the square wave should be completely passing through since it is within our filter limits.

3. Connect a square wave analog signal of 100 Hz. Since 100 Hz is in our filtered range, we should see the signal filtered significantly.

Where to Go From Here

Refer to the **FPGA Module** book in the *LabVIEW Help* for more information about the LabVIEW FPGA Module, including conceptual topics about the features available with the FPGA Module, procedures to complete common tasks with the FPGA Module, and reference information about VIs and functions available with the FPGA Module. In LabVIEW, select **Help»Search the LabVIEW Help** to display the *LabVIEW Help*.

Refer to the **FPGA Interface** book in the *LabVIEW Help* for more information about programming host VIs to communicate with FPGAVIs.