Version 15.11

Hands-On Workshop

CompactRIO – Programming with LabVIEW FPGA

Please do not remove this manual

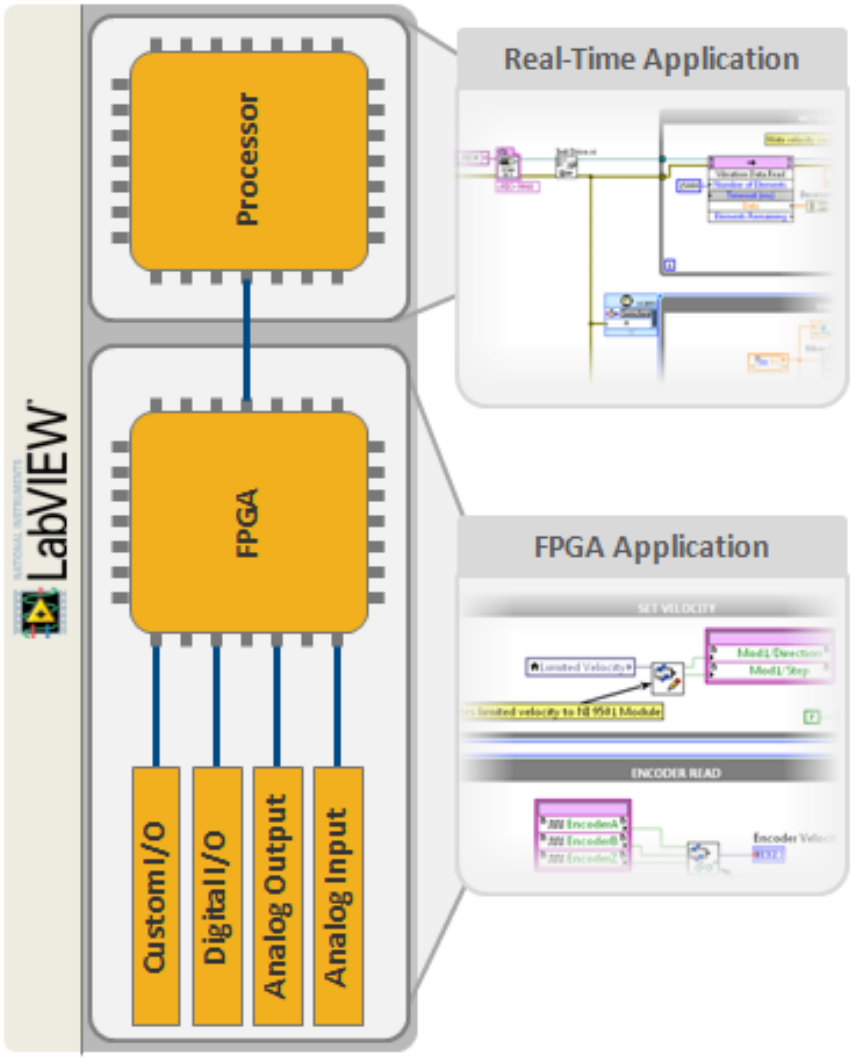
You will be sent an email which enables you to download the presentations and manual

uk.ni.com
ireland.ni.com

NATIONAL INSTRUMENTS

Hands-On: CompactRIO

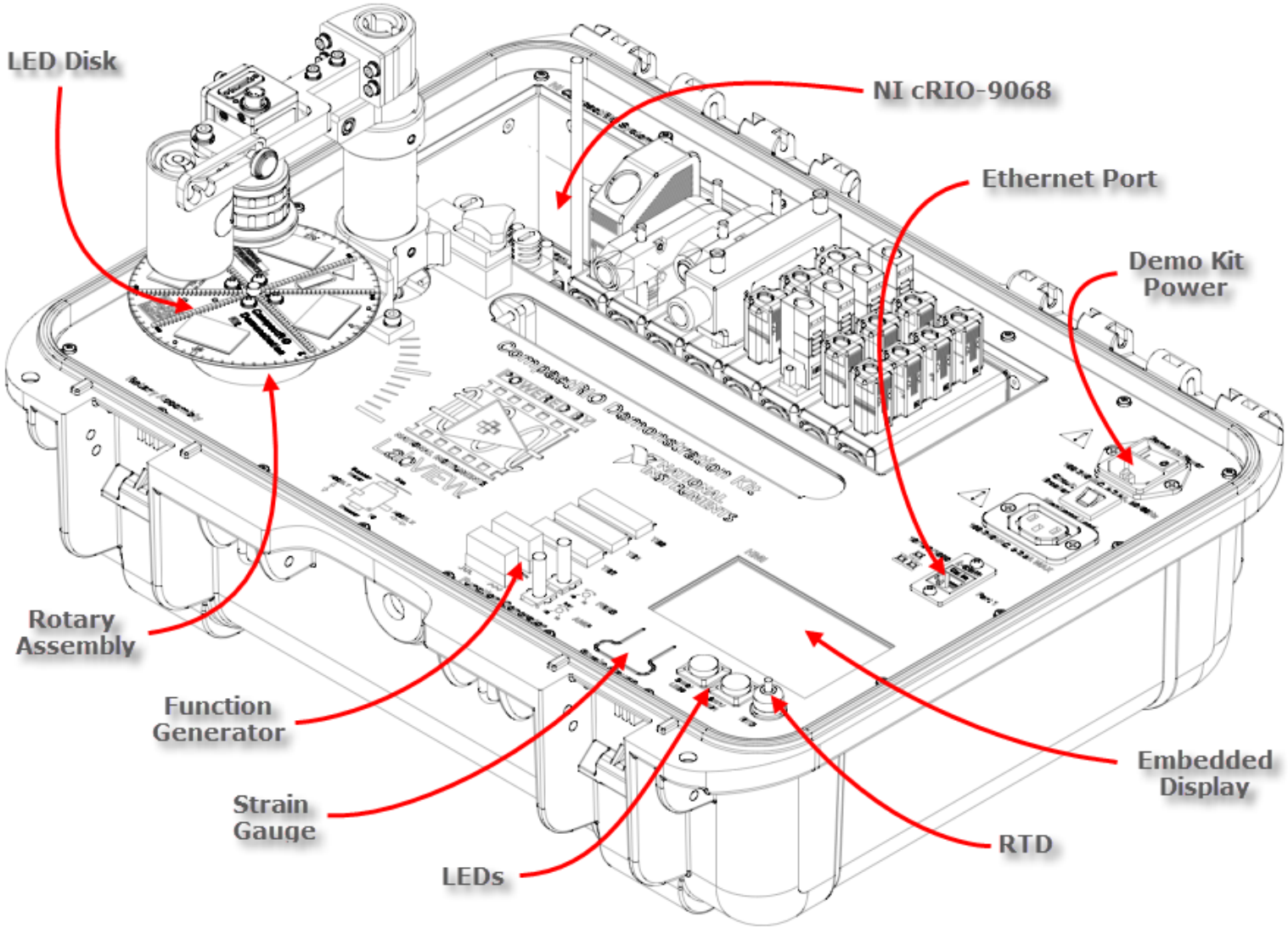
Contents



- Real-time OS
- Application software
- Networking and peripheral I/O drives
- DMA, interrupt, and bus control drivers

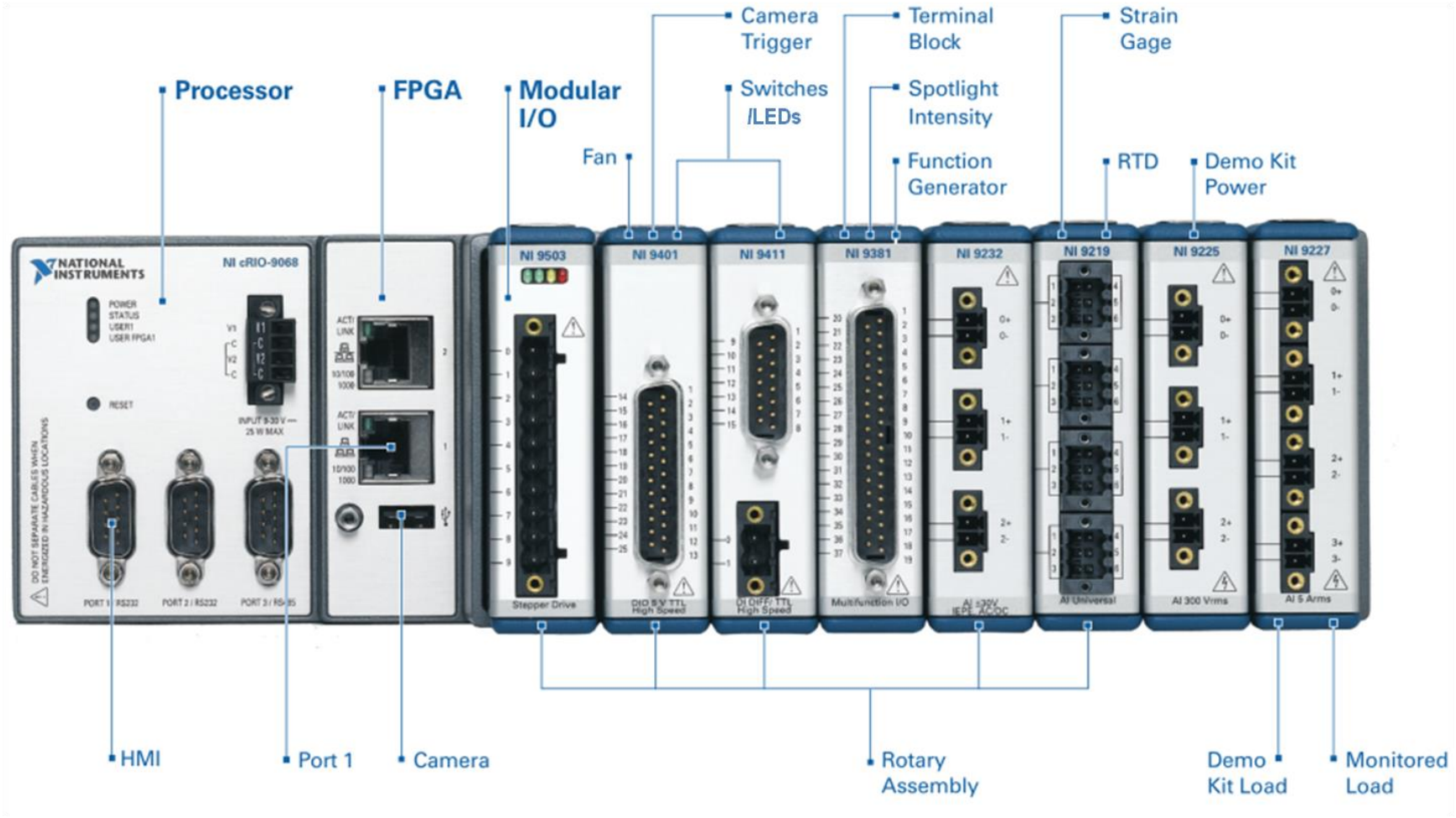
- Application IP
- Control IP
- Digital Signal Processing IP
- Specialized I/O drivers and interface
- DMA controller

NI CompactRIO DEMONSTRATION KIT



NI CompactRIO DEMONSTRATION KIT CONNECTION GUIDE

Built on the NI LabVIEW RIO Architecture

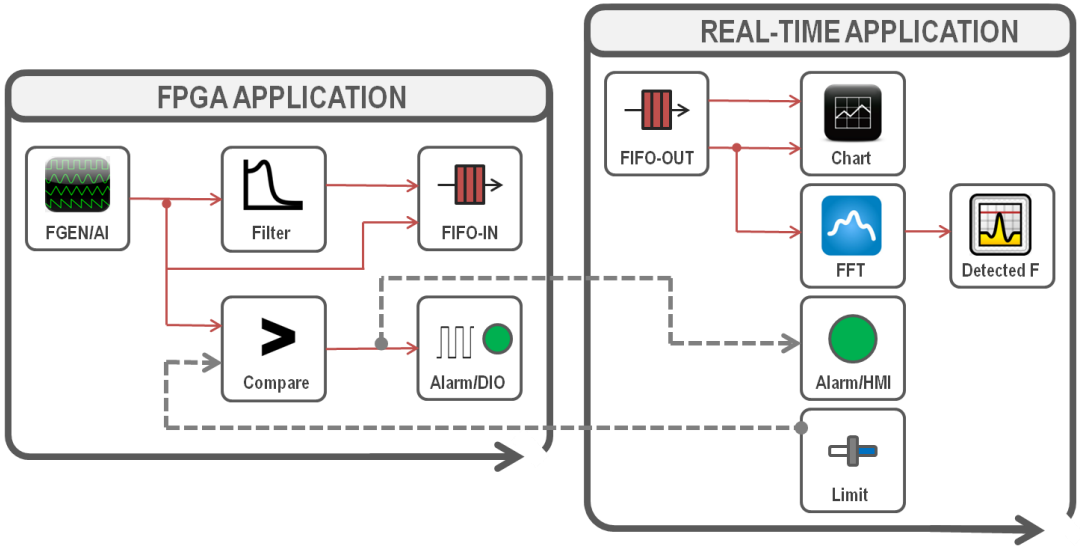


EXERCISE: FPGA-BASED BUTTERWORTH FILTER

Goals

- Use the ***FPGA*** portion of the ***CompactRIO system*** to implement a ***Butterworth filter*** for the signal coming from the ***function generator*** of the CompactRIO Demonstration Kit
- Use the ***simulation*** capabilities of ***LabVIEW FPGA*** to validate the design before compiling
- Use the ***real-time processor*** of the ***CompactRIO system*** to interface with the ***FPGA*** and generate a ***user interface*** to visualize the function generator and filtered signals as well as the calculated frequency

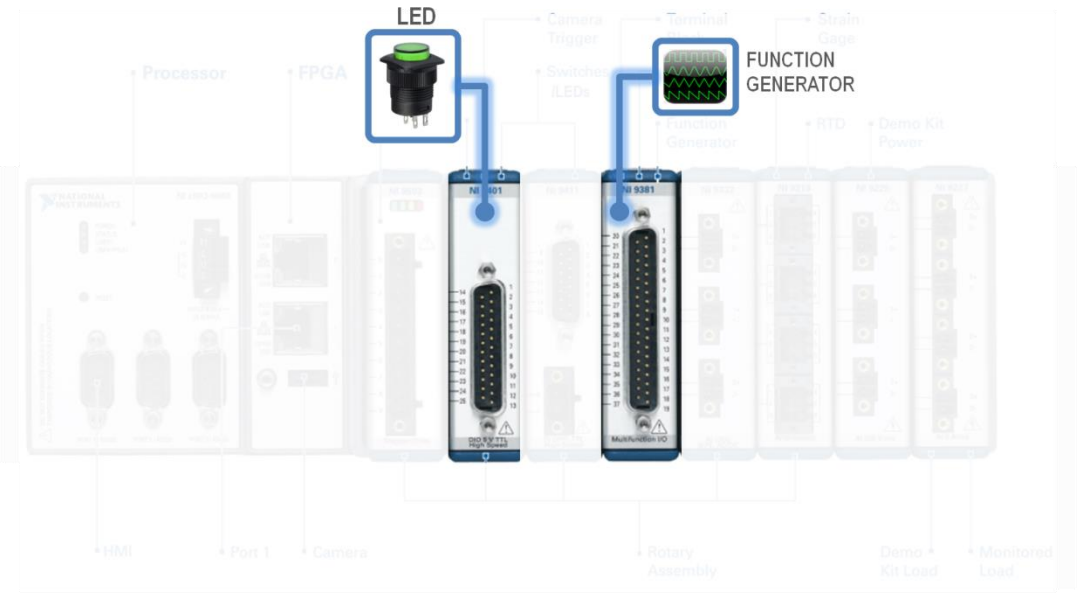
Part A—APPLICATION DESCRIPTION



In this exercise, use ***LabVIEW*** to configure the ***FPGA*** chip in the ***CompactRIO system*** to implement a ***Butterworth filter***.

As shown in the diagram on the left, the FPGA portion of the architecture is used to acquire a signal ranging from ***0 V to 5V*** at ***0 kHz to 1 kHz*** coming from the ***function generator*** in the ***CompactRIO Demonstration Kit.*** Filter the signal in this section of the application and send it via a ***DMA FIFO*** to the ***real-time processor*** for additional processing and visualization. Additionally, implement an ***alarming system*** based on a limit value coming from the user interface implemented in the real-time processor.

CompactRIO System



Field-Programmable Gate Array (FPGA)

An FPGA is a reprogrammable silicon chip. In contrast to programming the processors in your PC, programming an FPGA "physically" rewires the chip itself to implement your functionality rather than run a software application.

Real-Time Operating System (RTOS)

An RTOS is specially design to manage hardware resources and run applications with very precise timing and a high degree of reliability.

In this exercise, use measurement modules in ***slots 2 and 4*** of the **CompactRIO *system***.

Inputs

Slot 4: NI 9381—*0 V to 5 V AI/AO Module With 4 LVTTL DIO Lines*
Mod4/AI7→Function Generator
Sampling Period→20 kS/s
Resolution→12-bit

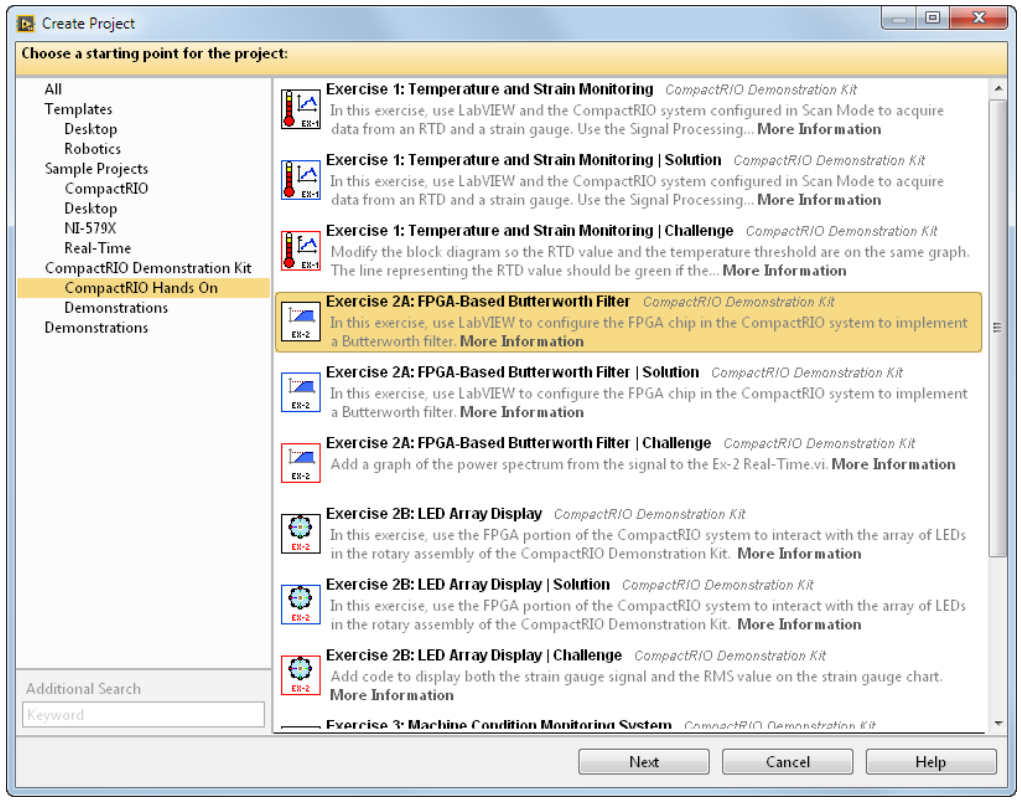
Outputs

Slot 2: NI 9401—*8 Ch, 5 V/TTL High-Speed Bidirectional Digital I/O Module*
Digital I/O 4 (DIO4)→LED0

Part B—CODE IMPLEMENTATION

1. Create an Instance of the Exercise 2A: FPGA-Based Butterworth Filter Sample Project

Launch LabVIEW and create a new LabVIEW project from the Exercise 2A: FPGA-Based Butterworth Filter sample project.



DETAILED INSTRUCTIONS

In this exercise, use an existing ***LabVIEW sample project*** as the starting point of the application.

- ***Launch LabVIEW***
  Launch LabVIEW by navigating to ***Start » All Programs » National Instruments » LabVIEW <Year>» LabVIEW.***

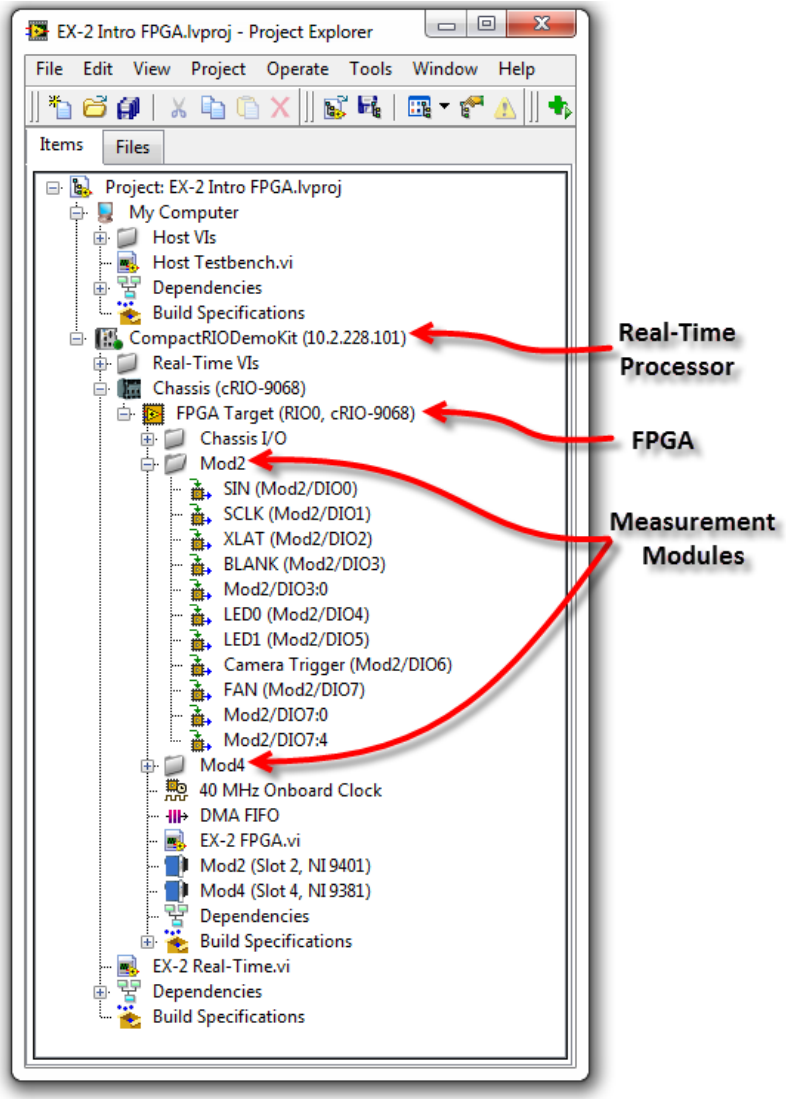
- ***Create an Instance of the Existing LabVIEW Project***
  Click on the ***Create Project*** button on the right side of the main screen of ***LabVIEW*** and create an instance of the **Exercise 2A: FPGA-Based Butterworth Filter** sample project located at:

CompactRIO Demonstration Kit>> CompactRIO Hands On

Click **Next** to customize the **Project Name** and **Project Root** if desired. Finally, click **Finish** to create the project.

2. Explore the project and reveal its components.

Expand each hierarchy in the LabVIEW Project Explorer to reveal the real-time processor and the FPGA of the CompactRIO system.



DETAILED INSTRUCTIONS

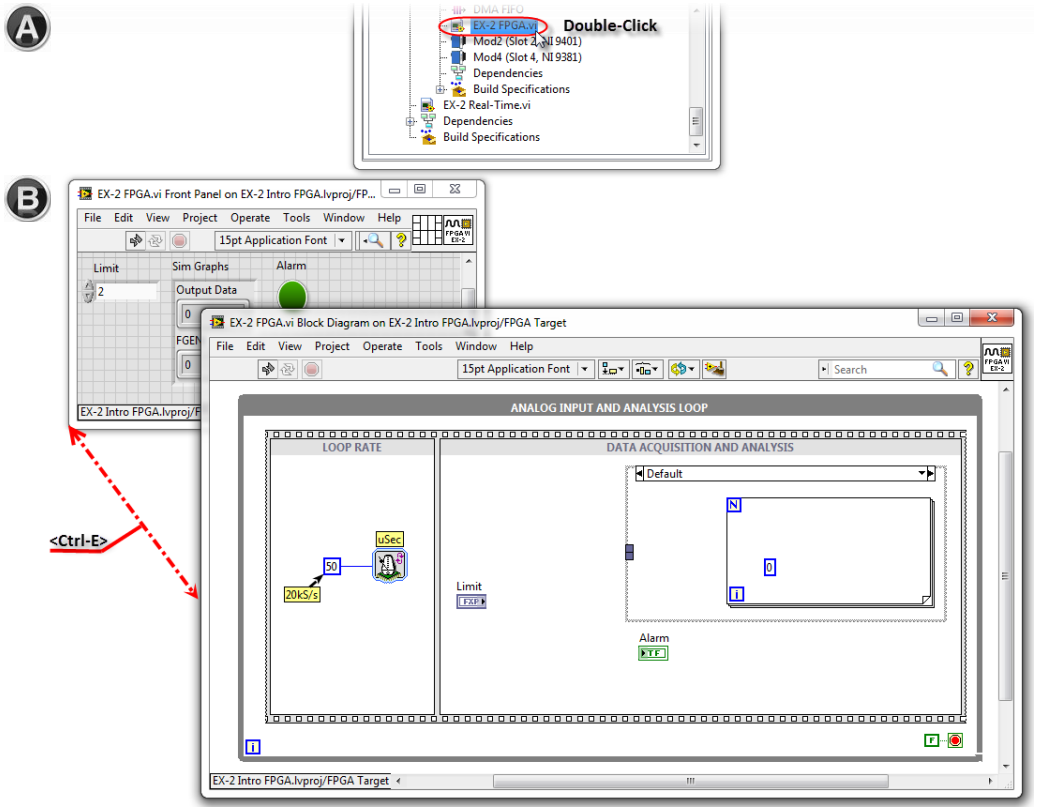
- ***Expand the Hierarchies* Exercise 2A: FPGA-Based Butterworth Filter sample project**
  This project already contains the **CompactRIO** system exposing the **FPGA** and the measurement modules that you are going to use.

  Notice that the measurement modules are located under the FPGA target in the hierarchy.

  To reveal the components of the project, expand the hierarchies in the project tree by clicking on the "**+**" boxes.

3. Open the Ex-2 FPGA.vi and show its block diagram.

The Ex-2 FPGA.vi is located under the FPGA target in the hierarchy. This VI already contains timing code to use for this exercise.



DETAILED INSTRUCTIONS

- ***Open the Ex-2 FPGA.vi***
  Double-click on the file named **Ex-2 FPGA.vi** located under the FPGA target in the hierarchy as shown in ***Figure A***.

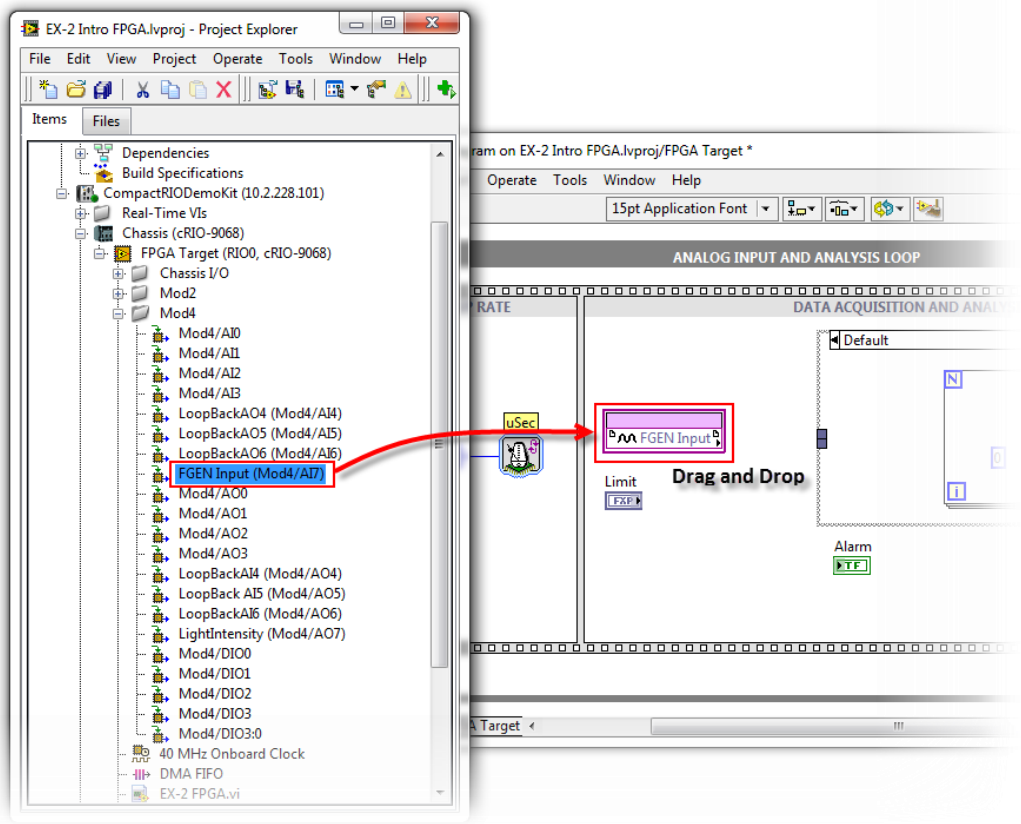
- ***Show the Block Diagram***
  Press ***<Ctrl-E>*** or click on ***Window » Show Block Diagram*** to display or toggle between the block diagram and the front panel as shown in ***Figure B***.

  The block diagram contains a ***While Loop*** with a ***sequence structure*** embedded to enforce loop timing.

  Program this VI to acquire input from a ***function generator*** and write the analog waveform data to memory that can be accessed by the ***real-time operating system*** on the ***CompactRIO***.

4. Add the Function Generator Channel to the block diagram.

Drag and drop the Function Generator Input node of the NI 9381 C Series measurement module (Mod4) into the sequence structure.

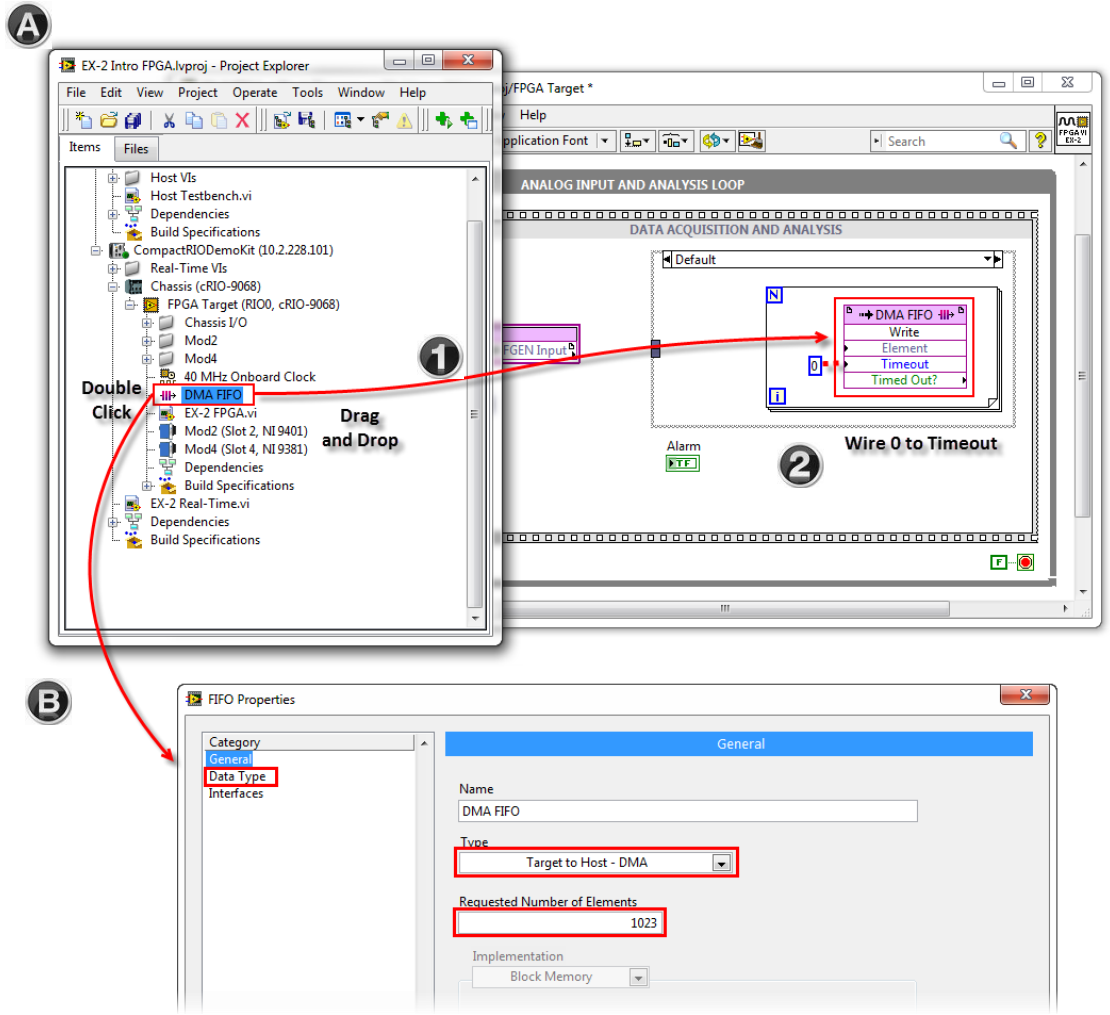


DETAILED INSTRUCTIONS

- ***Drag and Drop the FGEN Input (Mod4/AI7) Into the Sequence Structure***
  Locate the ***FGEN Input (Mod4/AI7)*** terminal located in the LabVIEW project under the **Mod4** measurement module of the FPGA target in the LabVIEW Project Explorer. Drag and drop the terminal onto the block diagram and place it inside the ***Data Acquisition and Analysis*** section of the sequence structure.

5. Add the communication channel between the FPGA and the real-time processor.

With the DMA FIFO, you can transfer data from the FPGA to the real-time processor. Drag and drop the DMA FIFO from the LabVIEW Project Explorer into the For Loop.



DETAILED INSTRUCTIONS

- ***Drag and Drop the DMA FIFO Onto the Block Diagram***
  Drag and drop the ***DMA FIFO*** node onto the block diagram and place it inside the ***For Loop*** as shown in ***Figure A1***.

- ***Wire the Constant With Value 0 to the Timeout Terminal of the DMA FIFO as Shown in Figure A2***
  A constant with value 0 indicates that the method cannot wait for space to be available in the FIFO before it attempts to write data to the FIFO.

  The ***DMA FIFO*** has been preconfigured as follows:

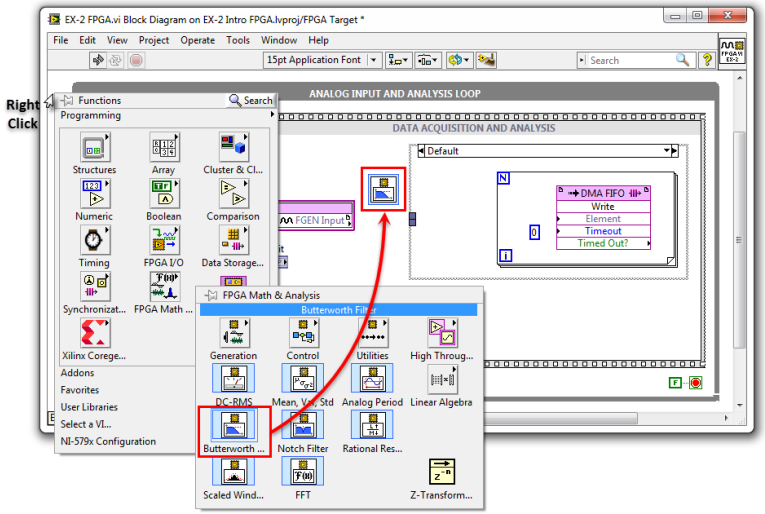
  - Target to Host-DMA
  - 1023 Elements
  - Fixed-Point Data Type

  Double-click on the ***DMA FIFO*** to open its configuration window as shown in ***Figure B***.

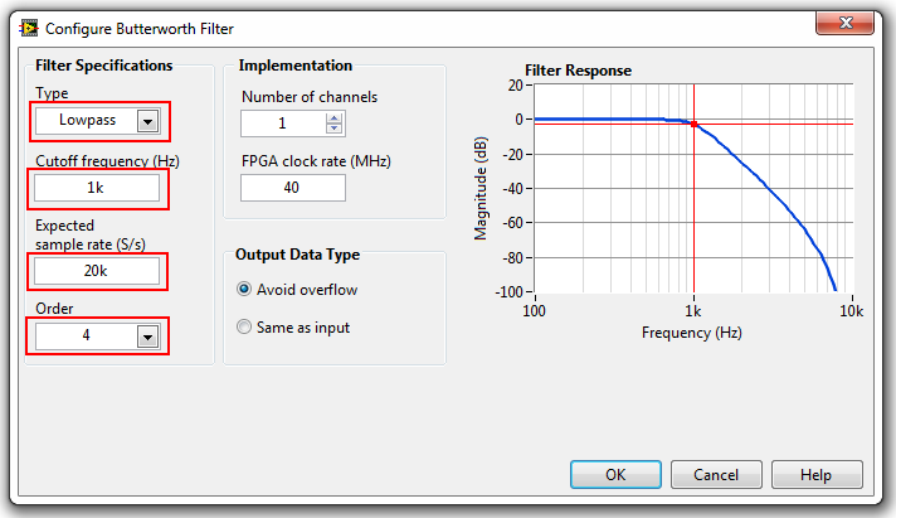
6. Filter the data from the Function Generator Input.

Filter the data from the Function Generator Input by sending it through a Butterworth filter configured to be a lowpass filter.

A



B



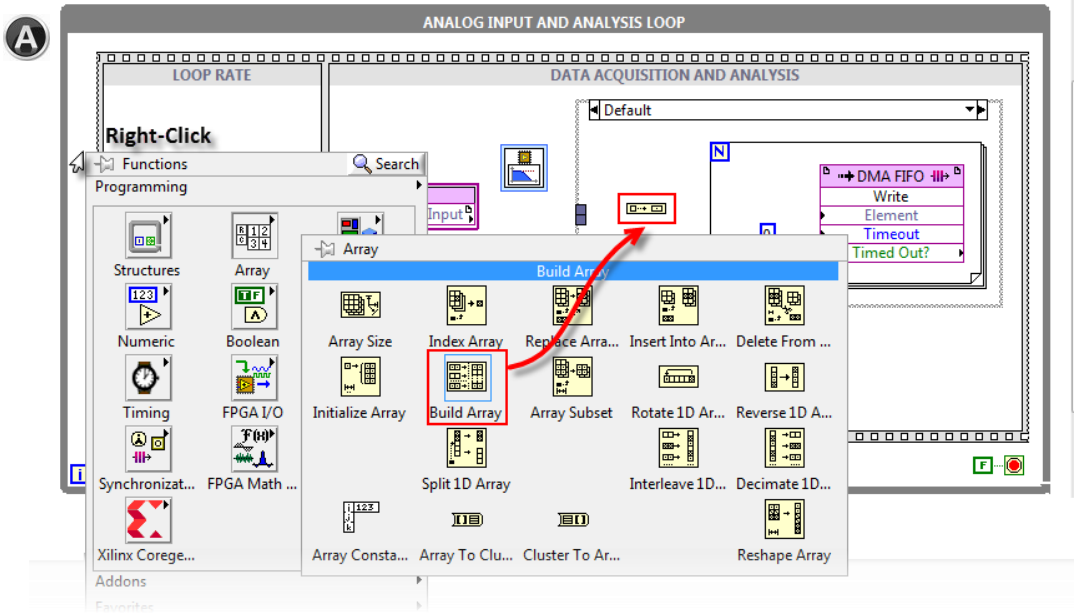
DETAILED INSTRUCTIONS

- ***Add a Butterworth Filter Function to the Block Diagram***
  Right-click on the block diagram to open the ***Functions Palette***. Navigate to ***Programming » FPGA Math & Analysis***. Drag and drop the ***Butterworth Filter*** function onto the ***block diagram*** as shown in ***Figure A***. The configuration window automatically displays. Configure the function as shown in ***Figure B*** using the following parameters:

  Type: Lowpass
  Cutoff frequency (Hz): 1 k
  Expected Sample Rate (S/s): 20k
  Order: 4

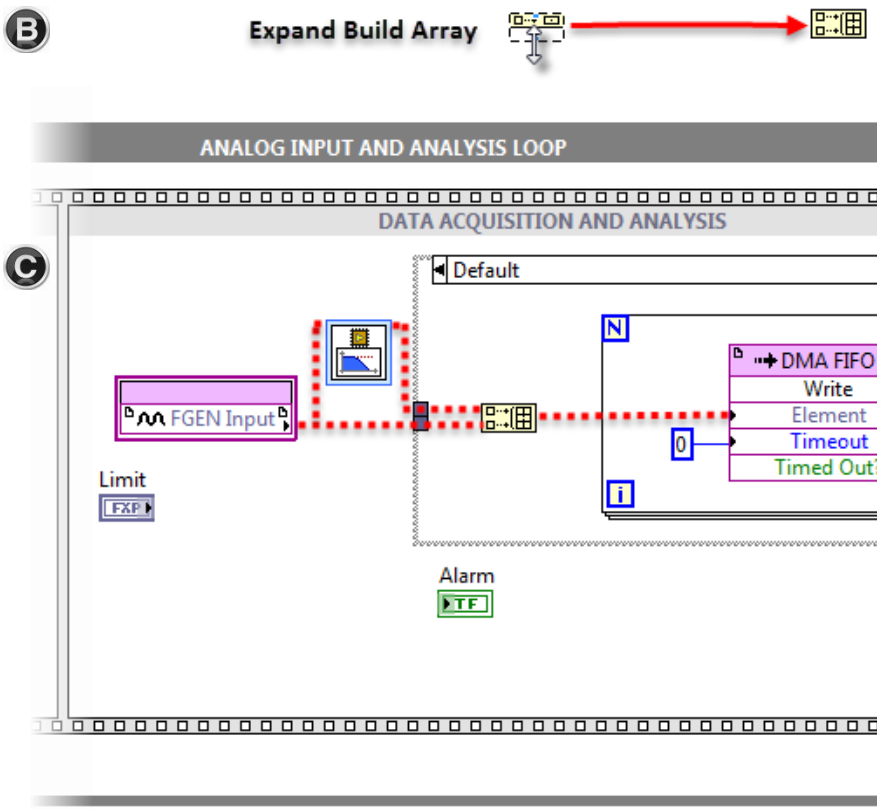
7. Send the data and filtered data to the DMA FIFO.

Send signal data and filtered data from the FPGA FGEN Input to the DMA FIFO.



DETAILED INSTRUCTIONS

- ***Add a Build Array Function to the Block Diagram***
  Right-click on the block diagram to open the ***Functions Palette***. Navigate to ***Programming » Array***. Drag and drop the ***Build Array function*** onto the block diagram as shown in ***Figure A***.

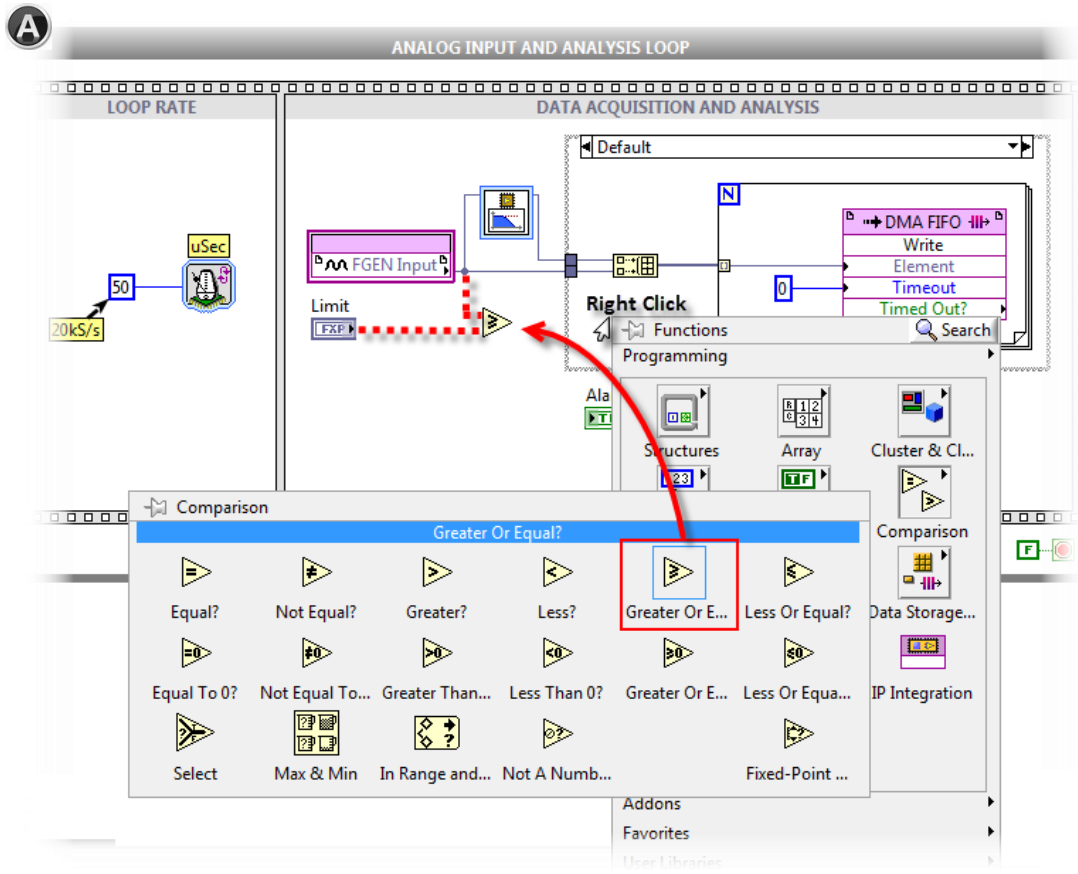


- ***Expand the Build Array Function to Show Two Input Terminals***
  Move the cursor over the ***Build Array function*** to show the blue points from which you can expand the node. Expand the node to include an additional terminal as shown in ***Figure B***.

- ***Wire the Filtered and Unfiltered Signals Into the Build Array Function and Wire the Output Into the DMA FIFO Element Node as Shown in Figure C***
  Connect the ***FGEN Input*** node to the ***Build Array function*** and ***Butterworth Filter*** input. Connect the output of the ***Butterworth Filter*** to the ***Build Array function***. Wire the array into the ***DMA FIFO Element node***.

8. Create an alarm if the data exceeds a preset limit.

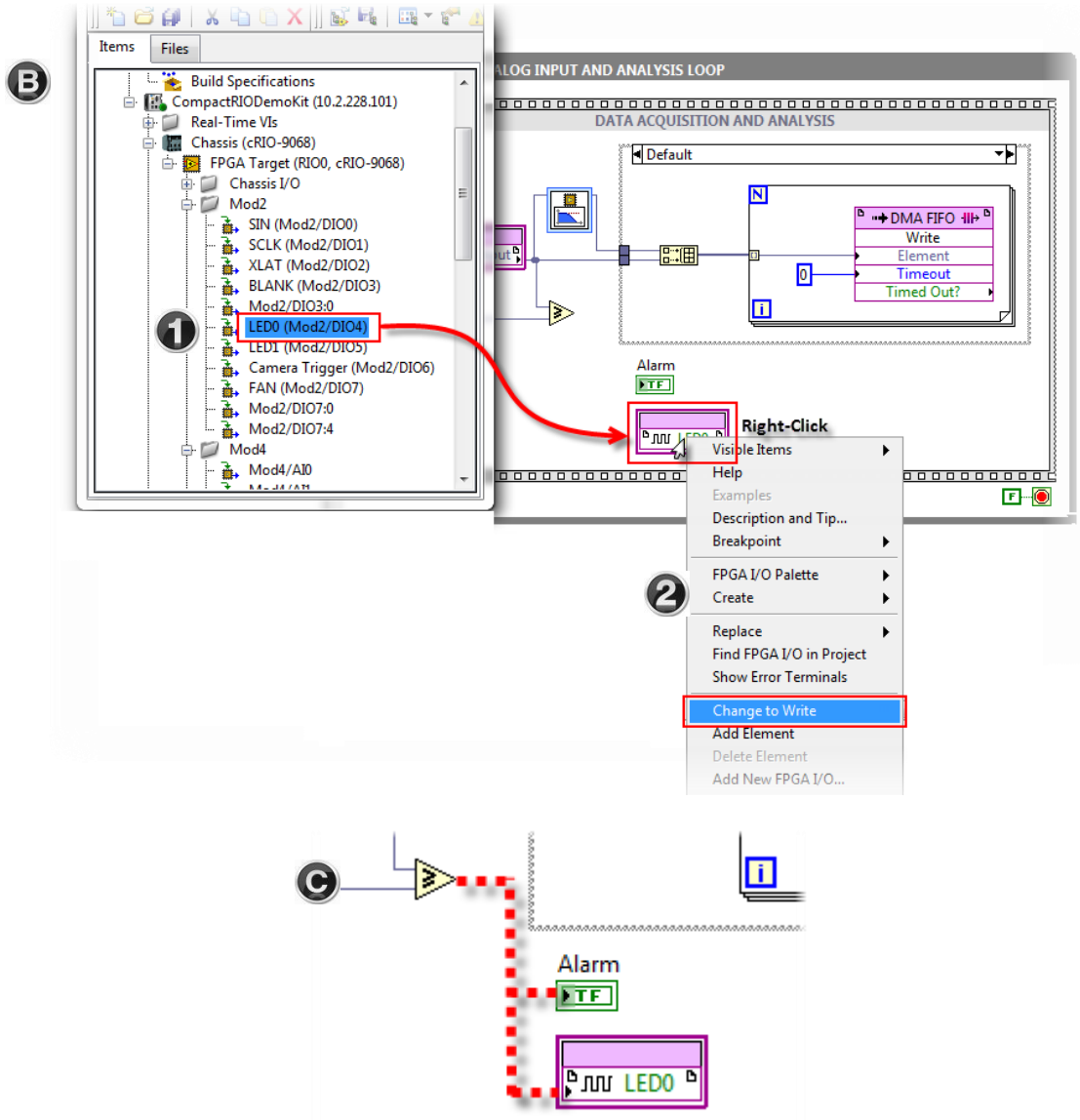
Configure an LED alarm on the user interface and CompactRIO to light up if the signal exceeds a user-set limit.



DETAILED INSTRUCTIONS

- ***Add a Greater Or Equal? Function to the Block Diagram***
  Right-click on the block diagram to open the ***Functions Palette***. Navigate to ***Programming » Comparison***. Drag and drop the ***Greater Or Equal? function*** onto the ***block diagram*** as shown in ***Figure A***.

- ***Wire the FGEN Input Signal and the Limit Control Into the Greater Or Equal? Function as Shown in Figure A***
  Create a new branch from the wire produced in the previous step to connect the ***FGEN Input*** terminal to the ***Greater Or Equal? function***. Connect the output of the ***Limit Control*** to the ***Greater Or Equal? function***.



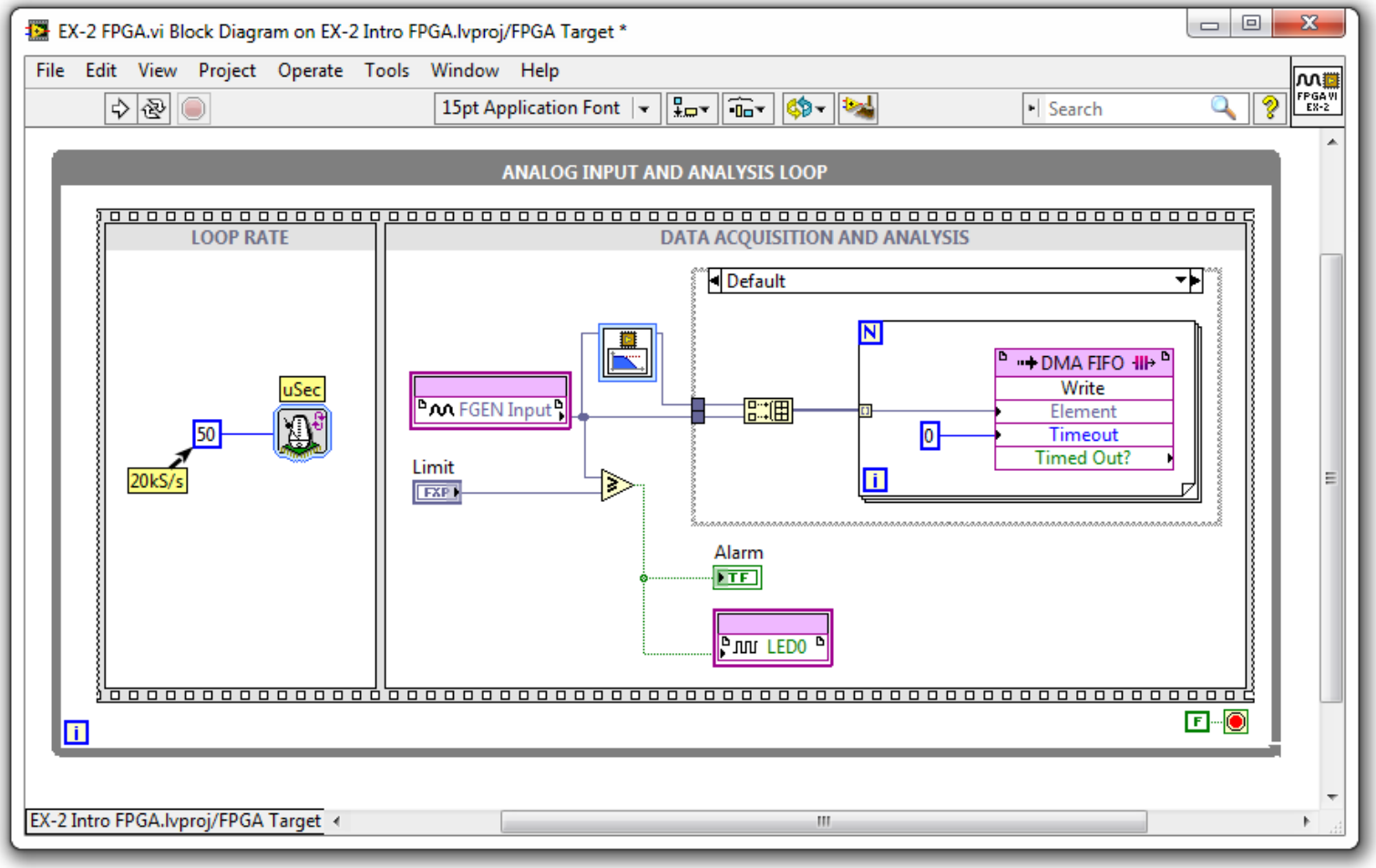
- ***Drag and Drop an LED0 Node Into the Sequence Structure as Shown in Figure B1***
  Drag and drop the ***LED0*** terminal located in the LabVIEW Project Explorer under the Mod2 measurement module of the FPGA target. Place it beneath the ***Alarm*** indicator.

- ***Change the LED0 Input to Write as Shown in Figure B2***
  Right-click on the LED0 text within the node and select **Change to Write**.

- ***Wire the Greater Or Equal? Function Output to the Alarm Indicator and the LED0 Node as Shown in Figure C***
  Wire the output from the ***Greater Or Equal? function*** to the ***Alarm*** indicator. Create a new branch from the wire to connect to the ***LED0*** input.

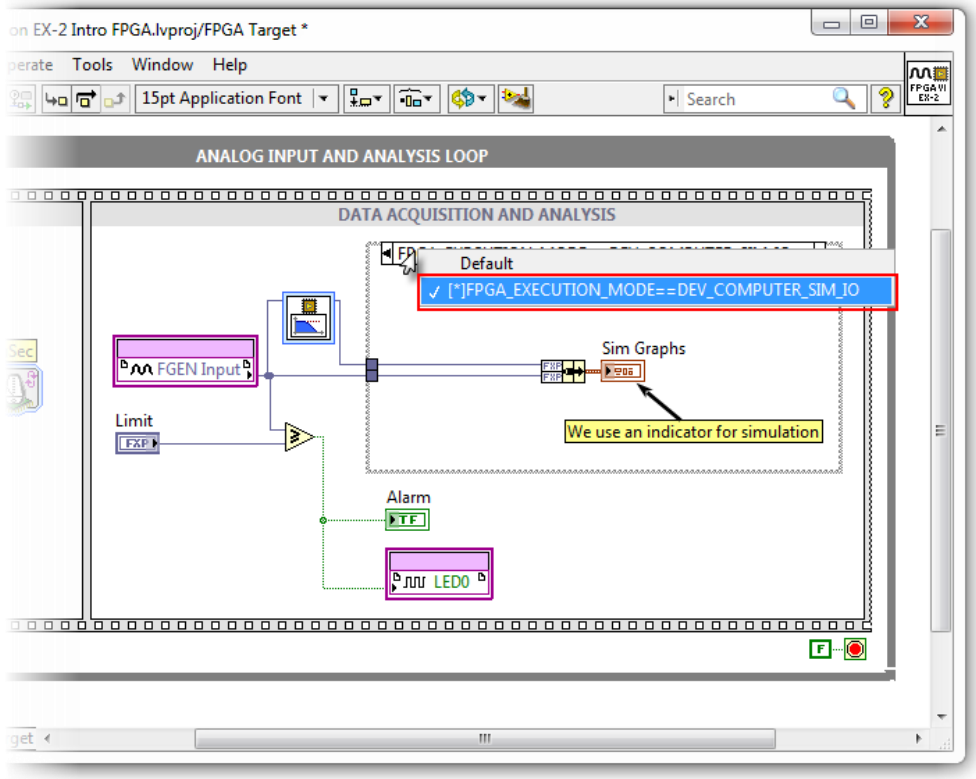
9. The completed block diagram for Ex-2 FPGA.vi should look like the image below.

Be sure to save your work by pressing <Ctrl-S> or clicking File » Save.



10. Observe the simulated case in the Conditional Disable structure.

Create a Host Testbench Application in Part C to test your FPGA code in a simulated mode without having to compile it. Observe the code that enables you to do this within the Ex-2 FPGA.vi.



DETAILED INSTRUCTIONS

- ***View the Simulated Case Within the Conditional Disable Structure***
  Click on the ***Default*** text on the menu bar of the ***Conditional Disable structure*****.** From the pull down menu, select:

FPGA_EXECUTION_MODE==DEV_COMPUTER_SIM_IO

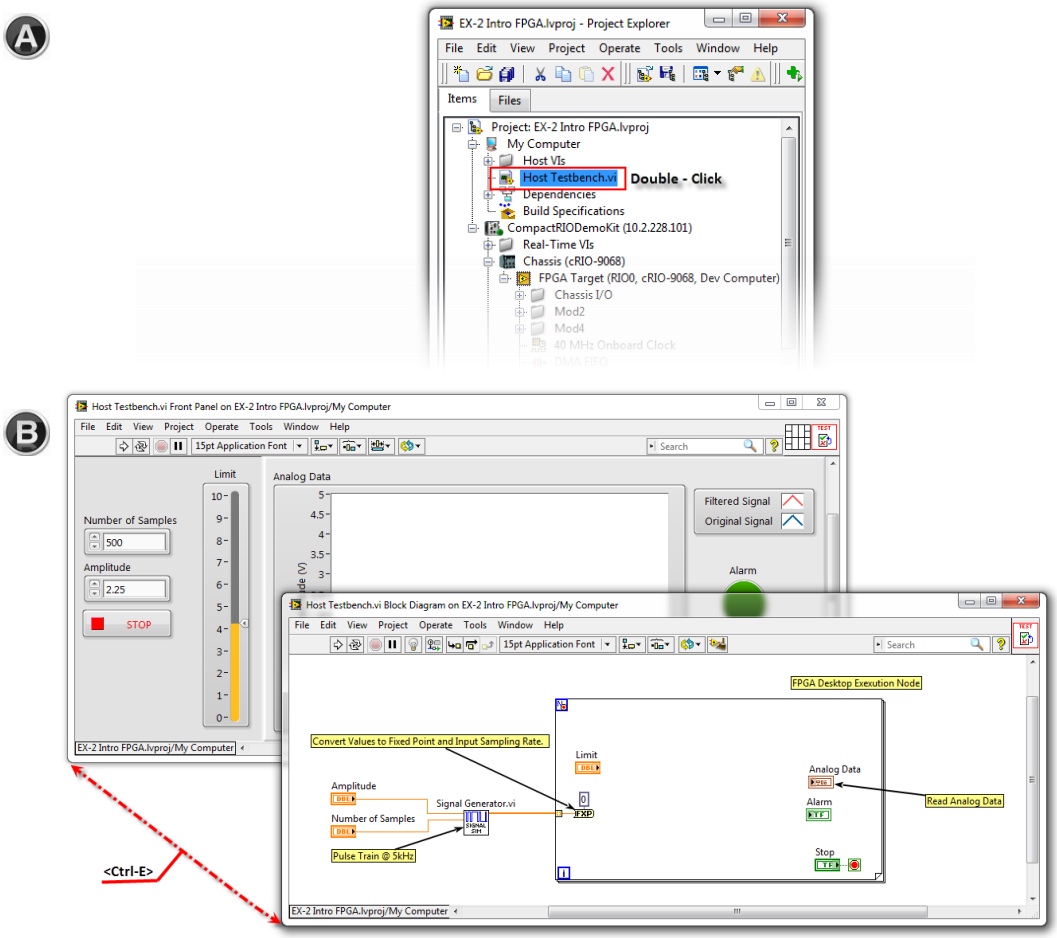
- ***Observe the Graph Indicator Used for Simulation***
  Rather than passing the data onto a memory buffer (***DMA FIFO***), view the simulated data on a graph. Bundle the filtered and unfiltered data together and display it on a graph called ***Sim Graph***.

  This code runs when the ***FPGA Target*** is set to run in ***simulation mode*** on the development computer rather than on the actual FPGA hardware. You will configure the simulation mode when you run the ***Host Testbench*** application in ***Part D***. This section is grayed out while the FPGA target is not set to run in simulation mode.

Part C—HOST TESTBENCH APPLICATION (FPGA SIMULATION)

11. Open the Host Testbench.vi and show its block diagram.

This VI is located under My Computer in the LabVIEW Project Explorer, and it already contains some code that you will use for this application.



DETAILED INSTRUCTIONS

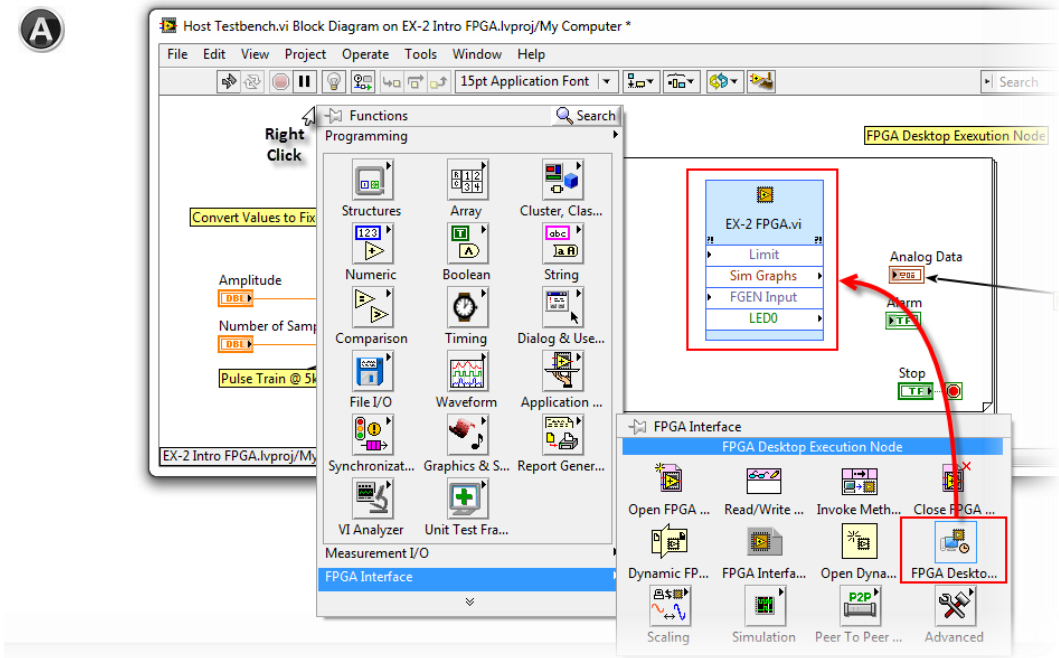
- ***Open the Host Testbench.vi***
  Double-click on the file named ***Host Testbench.vi*** located under My Computer in the LabVIEW Project Explorer as shown in ***Figure A***.

- ***Show the Block Diagram***
  Press ***<Ctrl-E>*** or click on ***Window » Show Block Diagram*** to open or toggle between the block diagram and the front panel as shown in ***Figure B***.

  The VI contains a conditional For Loop, signal generator, and some controls and indicators for the user interface.

12. Modify Host Testbench.vi to simulate the Ex-2 FPGA.vi on the development computer.

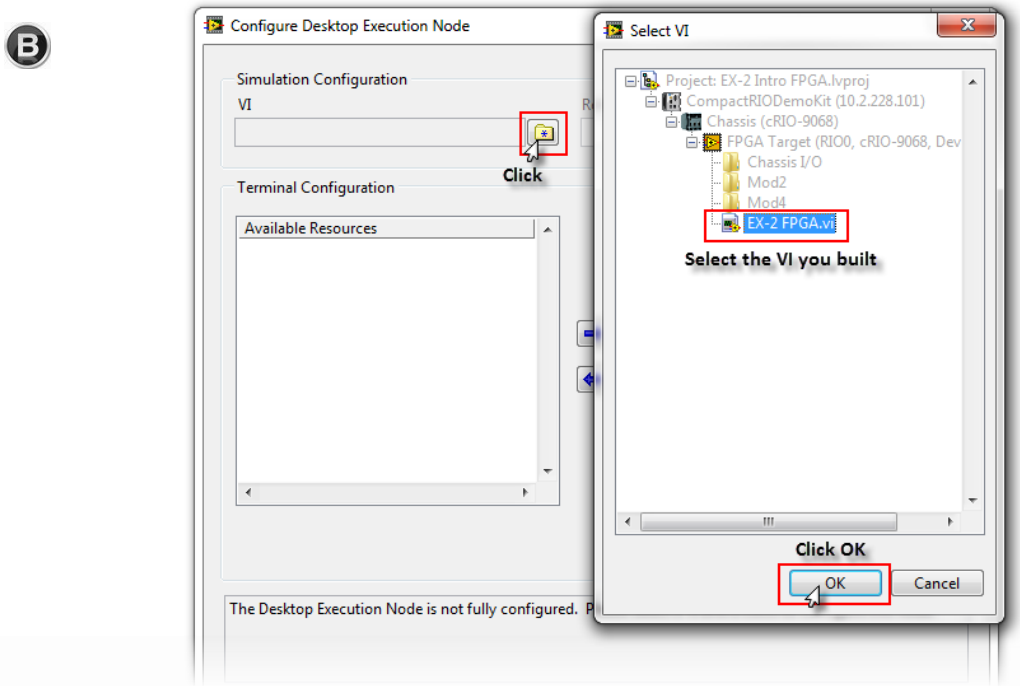
Use an FPGA Desktop Execution Node to run the FPGA VI from a host computer.

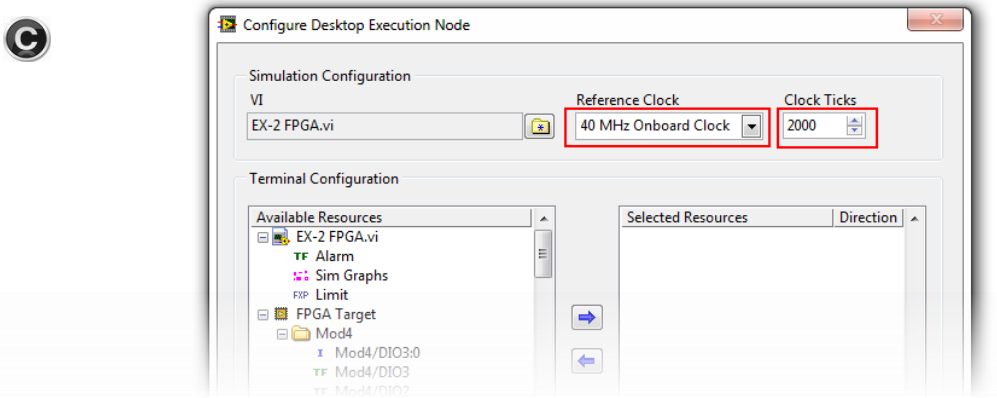


DETAILED INSTRUCTIONS

- ***Add an FPGA Desktop Execution Node to the Block Diagram as Shown in Figure A***
  Right-click on the block diagram to open the ***Functions Palette***. Navigate to **FPGA Interface** and drag and drop an ***FPGA Desktop Execution Node*** onto the block diagram.

  Once you place this function on the block diagram, the configuration window displays automatically. Configure the function as shown in ***Figure B.***





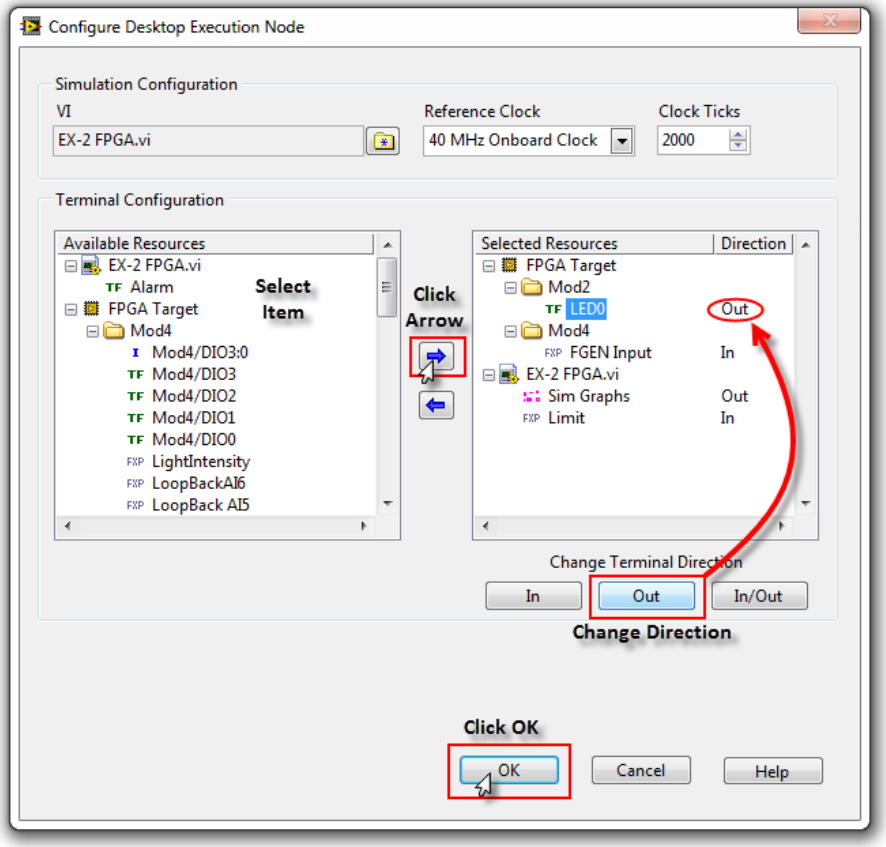
DETAILED INSTRUCTIONS

- ***Configure FPGA Desktop Execution Node to Run Ex-2 FPGA.vi as Shown in Figure B***
  In the configuration window, click the file icon next to the VI category. Select ***Ex-2 FPGA.vi*** from the menu and click ***OK***.

- ***Configure the Timing Settings for the FPGA Desktop Execution Node as Shown in Figure C***
  Configure the timing settings as shown in ***Figure C*** using the following parameters:

  Reference Clock: 40 MHz Onboard Clock
  Clock Ticks: 2000

  To achieve the expected sample rate of ***20 kS/s***, there must be ***2000 clock ticks*** of the 40 MHz onboard clock.



DETAILED INSTRUCTIONS

- ***Configure the Terminals for the FPGA Desktop Execution Node as Shown in Figure D***

  Select the following four items in the **Available Resources** table and click the **Arrow** to send them to **Selected Resources** one at a time:

  Ex-2 FPGA.vi:
  1. Limit
  2. Sim Graphs

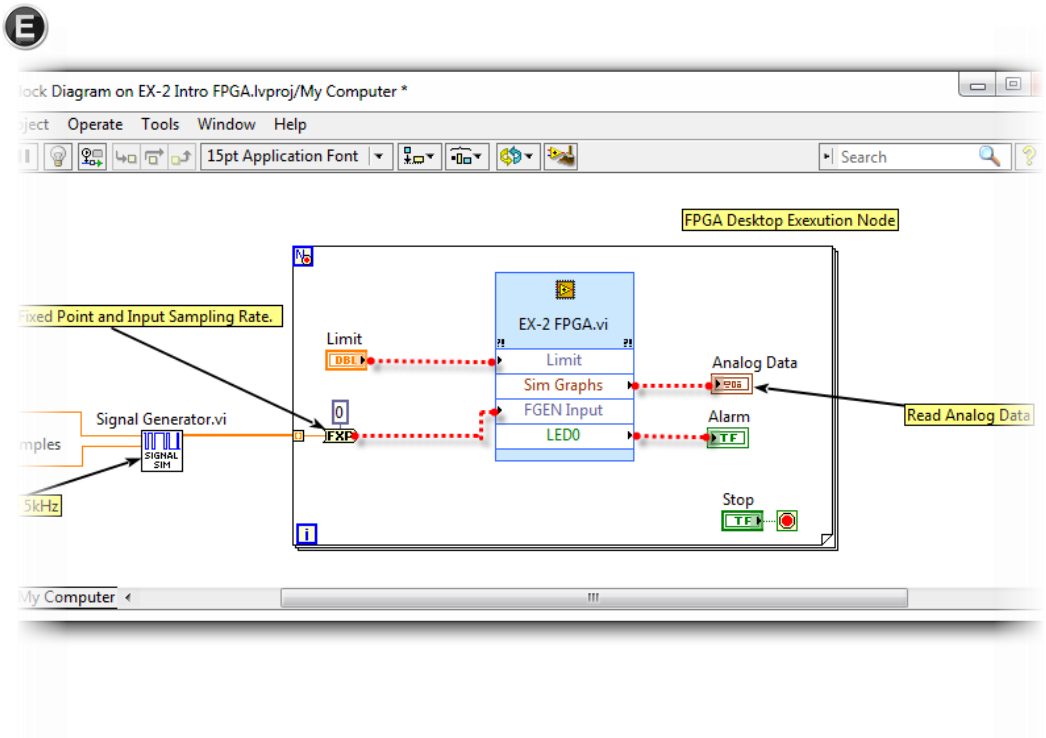
  FPGA Target\Mod4:
  3. FGEN Input

  FPGA Target\Mod2:
  4. LED0

- ***Change the Direction of the LED0 Terminal FPGA Desktop Execution Node as Shown in Figure D***

  Select ***LED0*** in the **Selected Resources** table. Click **Out** to change the terminal direction. Use the default directions for the other three terminals.

- ***Save Your Changes by Clicking OK***



DETAILED INSTRUCTIONS

- ***Wire the FPGA Desktop Execution Node According to Figure E***

  Connect the ***Limit*** control to the **Limit** terminal.

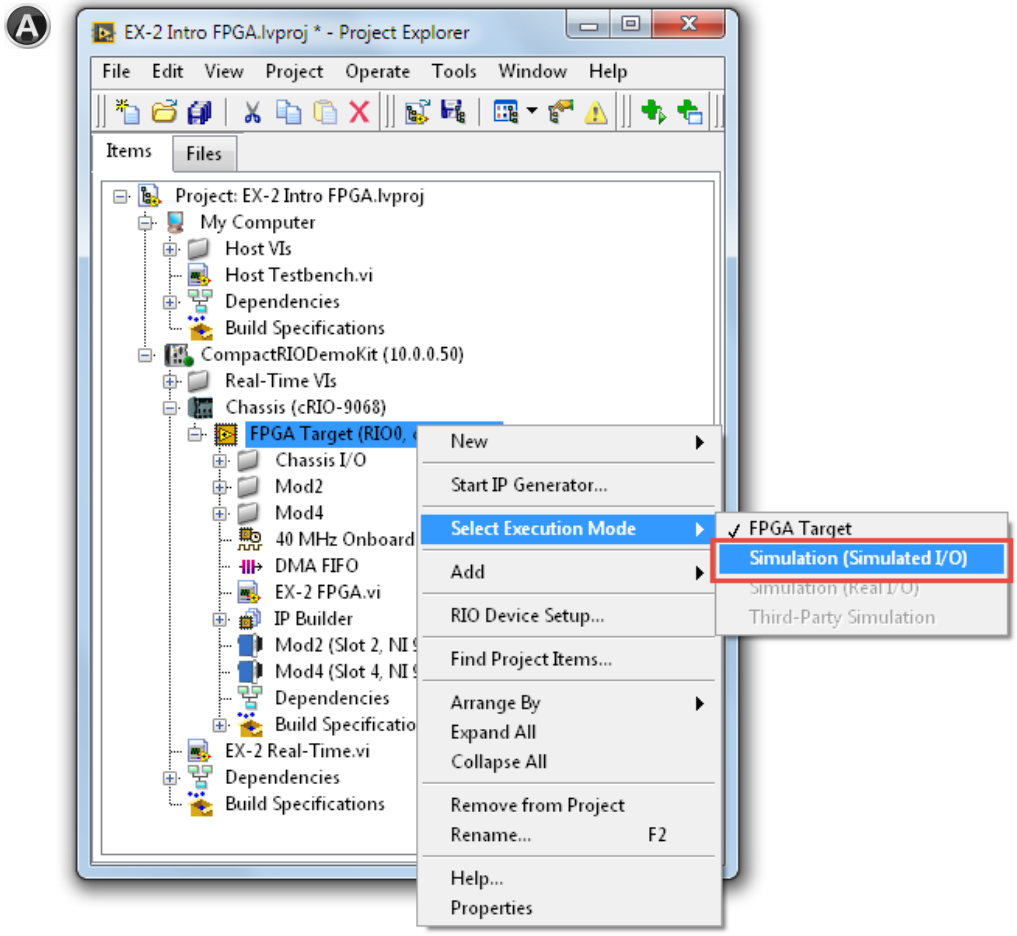
  Connect the **Signal Generator.vi** output that has been converted to the fixed point data type to the **FGEN Input** terminal.

  Connect the **Sim Graphs** terminal to the ***Analog Data*** indicator.

  Connect the **LED0** terminal to the ***Alarm*** indicator.

13. Set the FPGA target to run in simulated mode.

Set the FPGA target to run in simulated mode on the development computer rather than on the FPGA target.

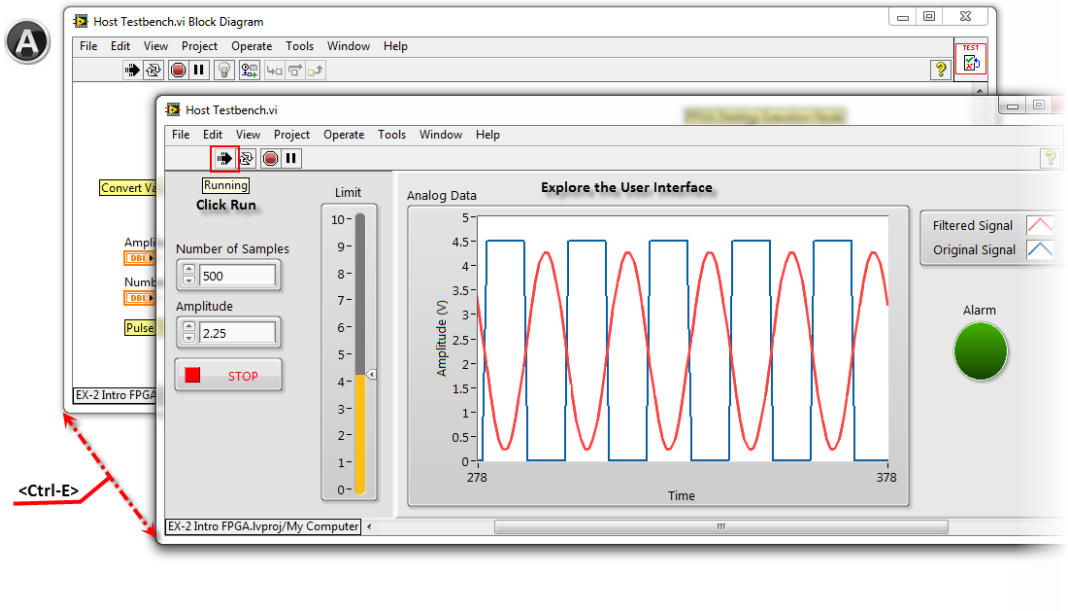


DETAILED INSTRUCTIONS

- ***Configure the FPGA Target to Run in Simulated Mode as Shown in Figure A***
  Right-click on the **FPGA Target** in the LabVIEW Project Explorer. In the **Execute VI on** category, choose **Simulation**.

14. Verify the FPGA code by running the host testbench application.

Run the **Host Testbench.vi** to verify that the FPGA code behaves correctly before compiling the FPGA code.



DETAILED INSTRUCTIONS

- ***Run Host Testbench.vi as Shown in Figure A***

  Switch back to the front panel of **Host Testbench.vi** by pressing ***<Ctrl-E>*** and run the program by clicking on the arrow button at the top of the front panel.

  Adjust the **Number of Samples** to control how long the **signal generator** runs.

  Adjust the **Amplitude** to change the amplitude of the **signal generator**.

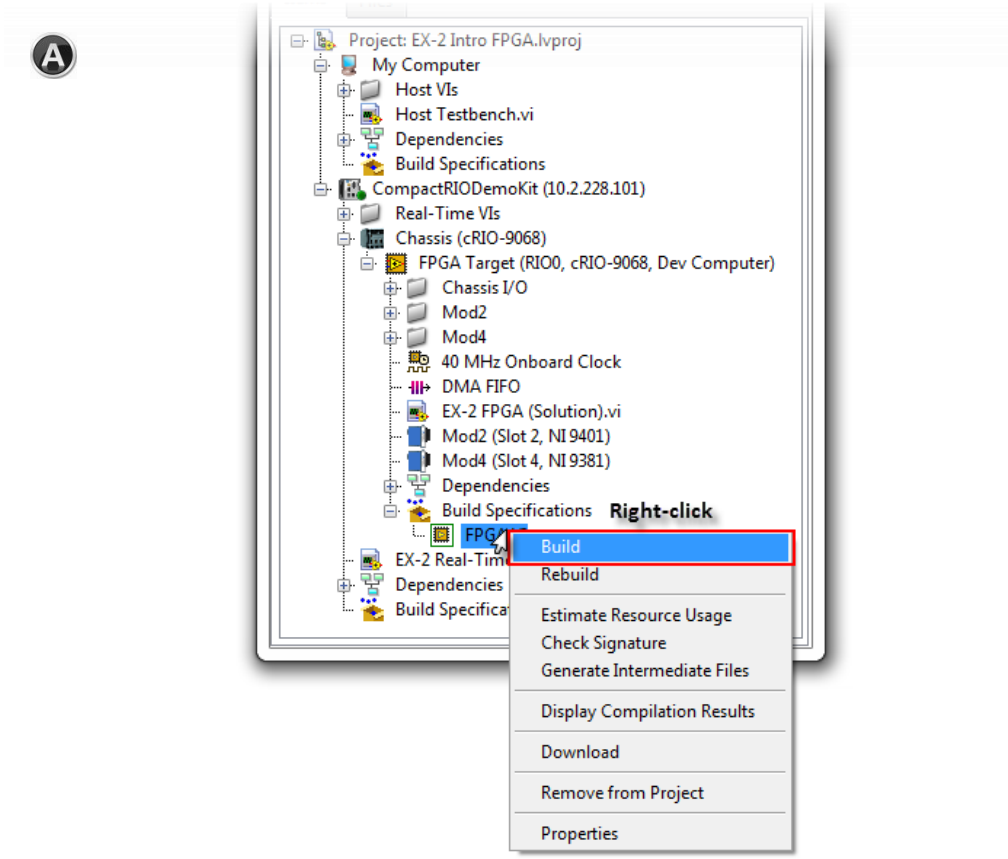
  Adjust the limit that triggers the **Alarm** LED by moving the slide control up or down.

  Using the **FPGA Desktop Execution Node**, you could verify that the FPGA programming executes as expected prior to compiling the FPGA code. This can help save development time because compiling FPGA code is a slow process.

Part D—FPGA COMPILATION PROCESS

15. Save and compile the EX-2 FPGA.vi.

The compilation process of the EX-2 FPGA.vi takes about 30 minutes. While the code is compiling, continue working on the real-time portion of the application in **Part E**.

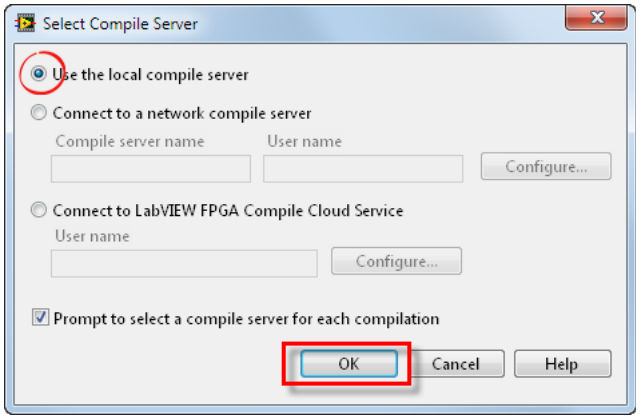


DETAILED INSTRUCTIONS

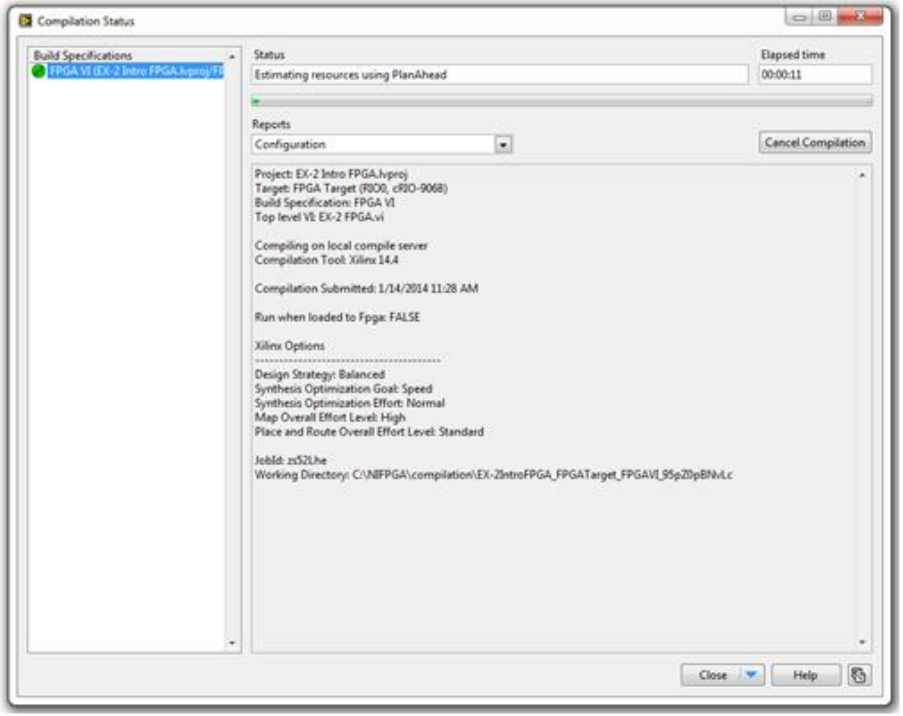
- ***Save the EX-2 FPGA.vi***
  Press ***<Ctrl-S>*** or go to ***File » Save.***

- ***Compile the EX-2 FPGA.vi***
  Navigate to the ***Build Specifications*** category under the FPGA hierarchy. Expand this category and right-click on the existing ***EX-2 FPGA*** and select ***Build*** as shown in ***Figure A***.

B



C



DETAILED INSTRUCTIONS

LabVIEW prompts you to select the ***compile server***. Select the ***local compile server*** and click ***OK*** as shown in ***Figure B***. This will start the compilation process.

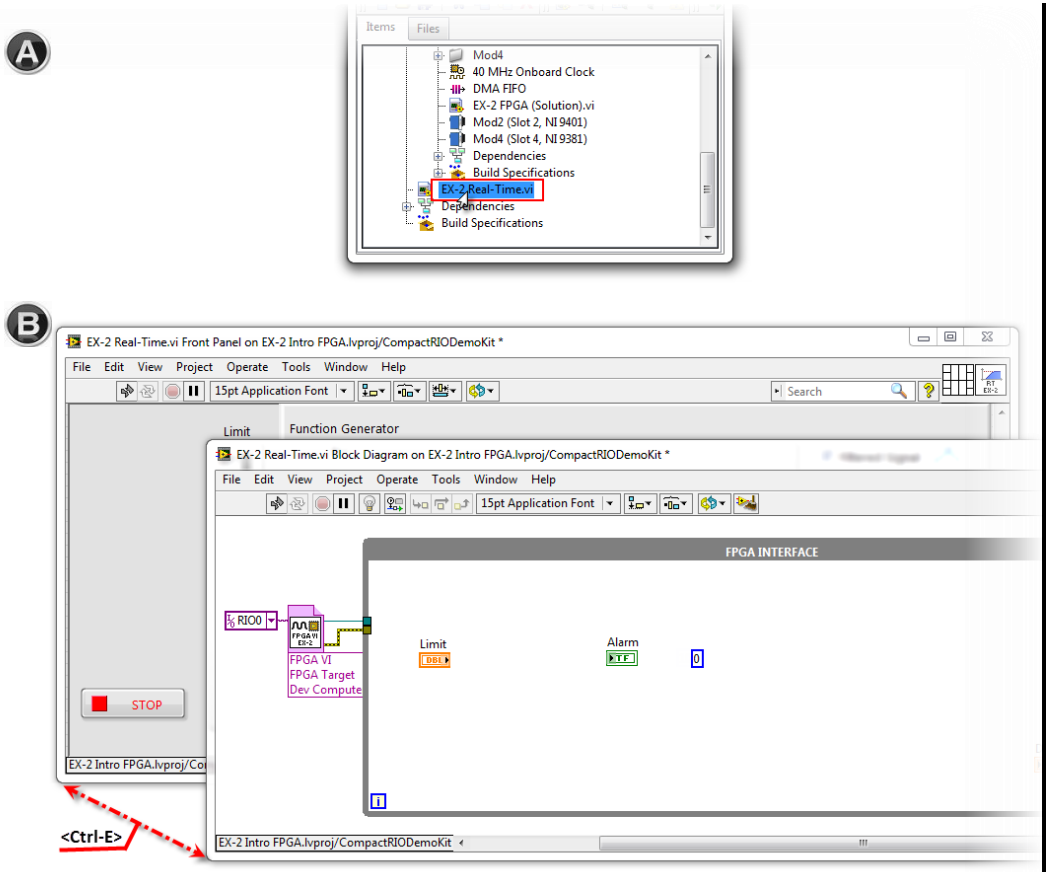
During the compilation process, ***LabVIEW*** generates intermediate HDL files that are later processed by the ***Xilinx Compiler,*** which outputs a bitfile containing the placing and routing information of the FPGA design.

The compilation process for this VI takes **about 15 minutes**. Once you have reached the Compilation Status window, as shown in ***Figure C***, ***PROCEED TO PART E*** to start developing the real-time portion of the application.

Part E—REAL-TIME CODE IMPLEMENTATION

16. Open the Ex-2 Real-Time.vi and show its block diagram.

This VI is located under the CompactRIO hierarchy, and it already contains some code that you will use for this application.



DETAILED INSTRUCTIONS

- ***Open the Ex-2 Real-Time.vi***
  Double-click on the file named **Ex-2 Real-Time.vi** located under CompactRIO in the LabVIEW Project Explorer as shown in ***Figure A***.

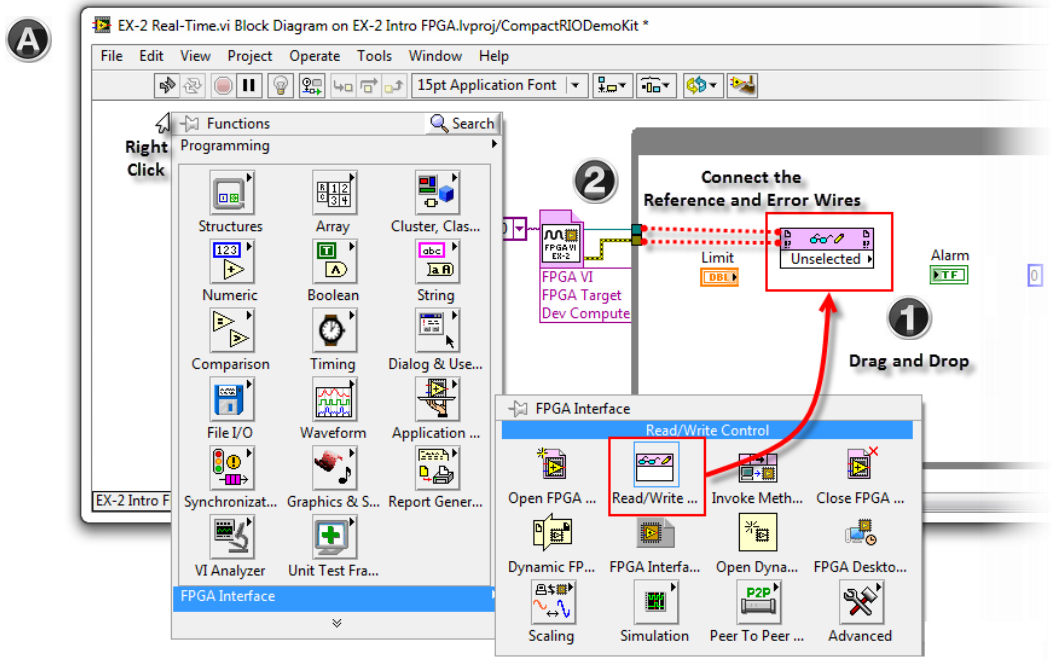
- ***Show the Block Diagram***
  Press ***<Ctrl-E>*** or click on ***Window » Show Block Diagram*** to open or toggle between the block diagram and the front panel as shown in ***Figure B***.

  The block diagram contains a While Loop, FPGA target references, and some controls and indicators to interact with the user interface.

  This VI runs on the real-time OS. It reads from the DMA FIFO and displays the data.

17. Set and view limit information between the real-time OS and the FPGA.

Send and receive data between the real-time OS and the FPGA to configure the limit that triggers the LED alarm to light up.

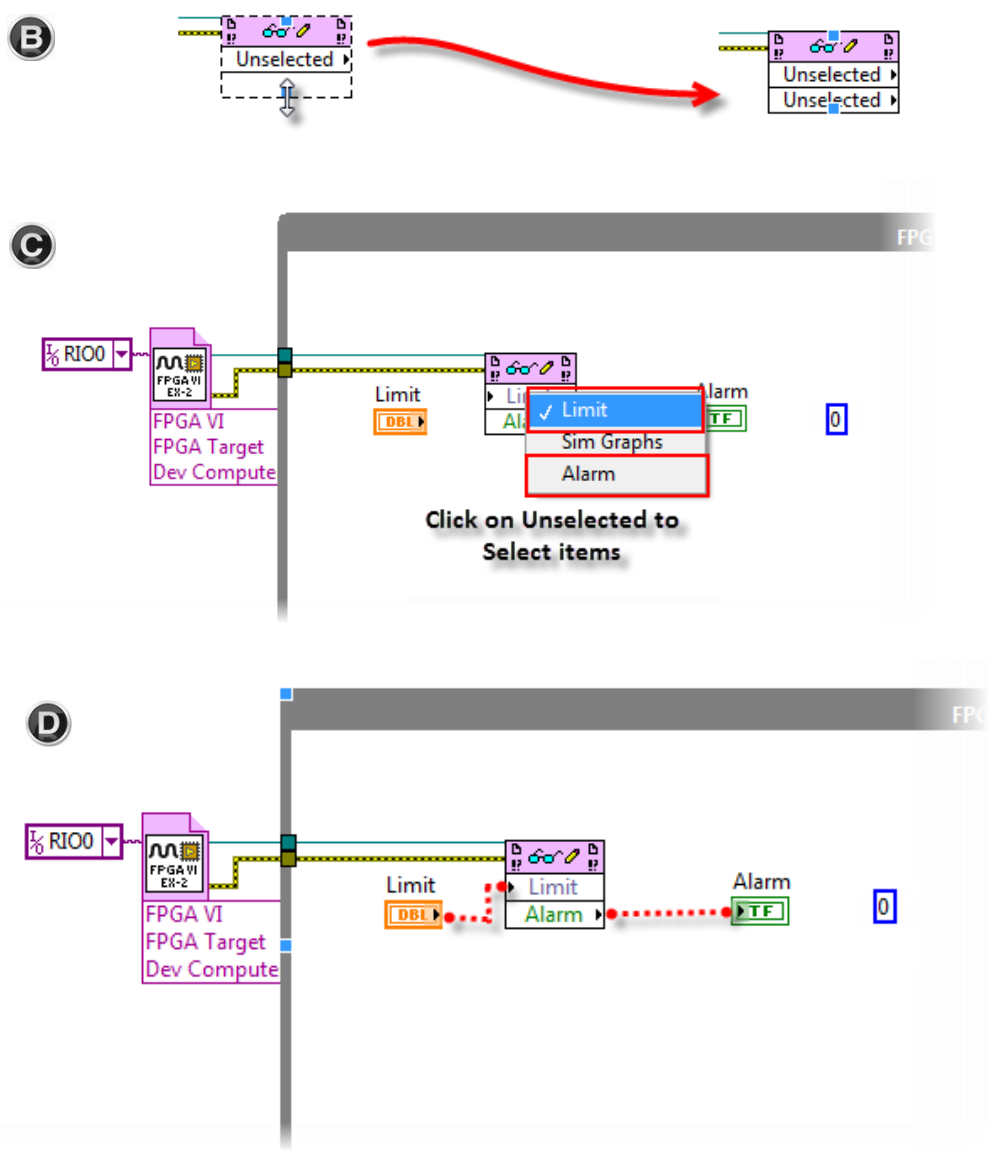


DETAILED INSTRUCTIONS

- ***Add a Read/Write Control Node as Shown in Figure A1***
  Right-click on the block diagram to open the ***Functions Palette***. Navigate to **FPGA Interface** and drag and drop a ***Read/Write Control Node*** onto the block diagram.

- ***Configure the Read/Write Control Node as Shown in Figure A2***
  Continue the ***Reference*** (aqua) and ***Error*** (yellow) wires coming from the ***Open FPGA VI Reference*** function and connect them to the ***Read/Write Control***.

  Connecting the FPGA reference to the ***Read/Write Control*** populates the item menu with items specific to your FPGA VI.



DETAILED INSTRUCTIONS

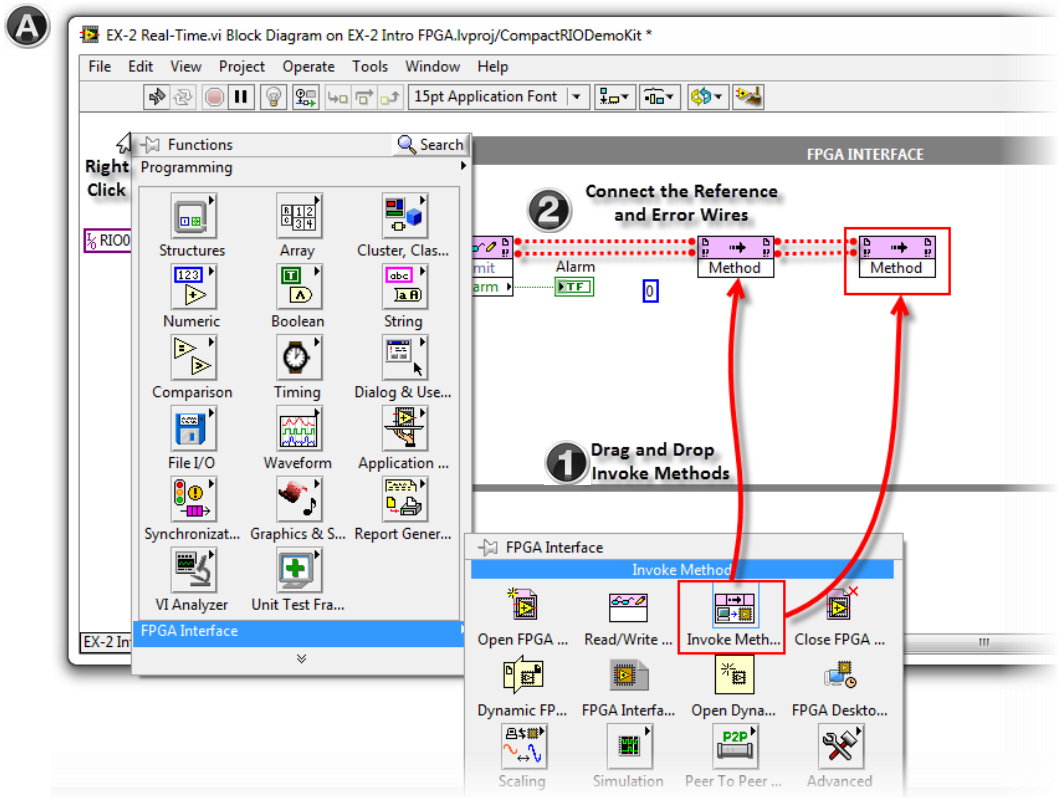
- ***Expand the Read/Write Control to Display Two Items as Shown in Figure B***
  Move the cursor over the ***Read/Write Control*** to show the blue points from which you can expand the node. Expand the node to show one additional terminal.

- ***Select the Read/Write Control Items as Shown in Figure C***
  Click on the **Unselected** text to show menu items. Select **Limit** for the top item and **Alarm** for the bottom item.

- ***Connect the Limit Control and the Alarm Indicator to the Read/Write Control Items as Shown in Figure D***

18. Read the elements from the DMA FIFO.

Place two Invoke Method functions from the FPGA Interface Palette and configure them to read data from the DMA FIFO.



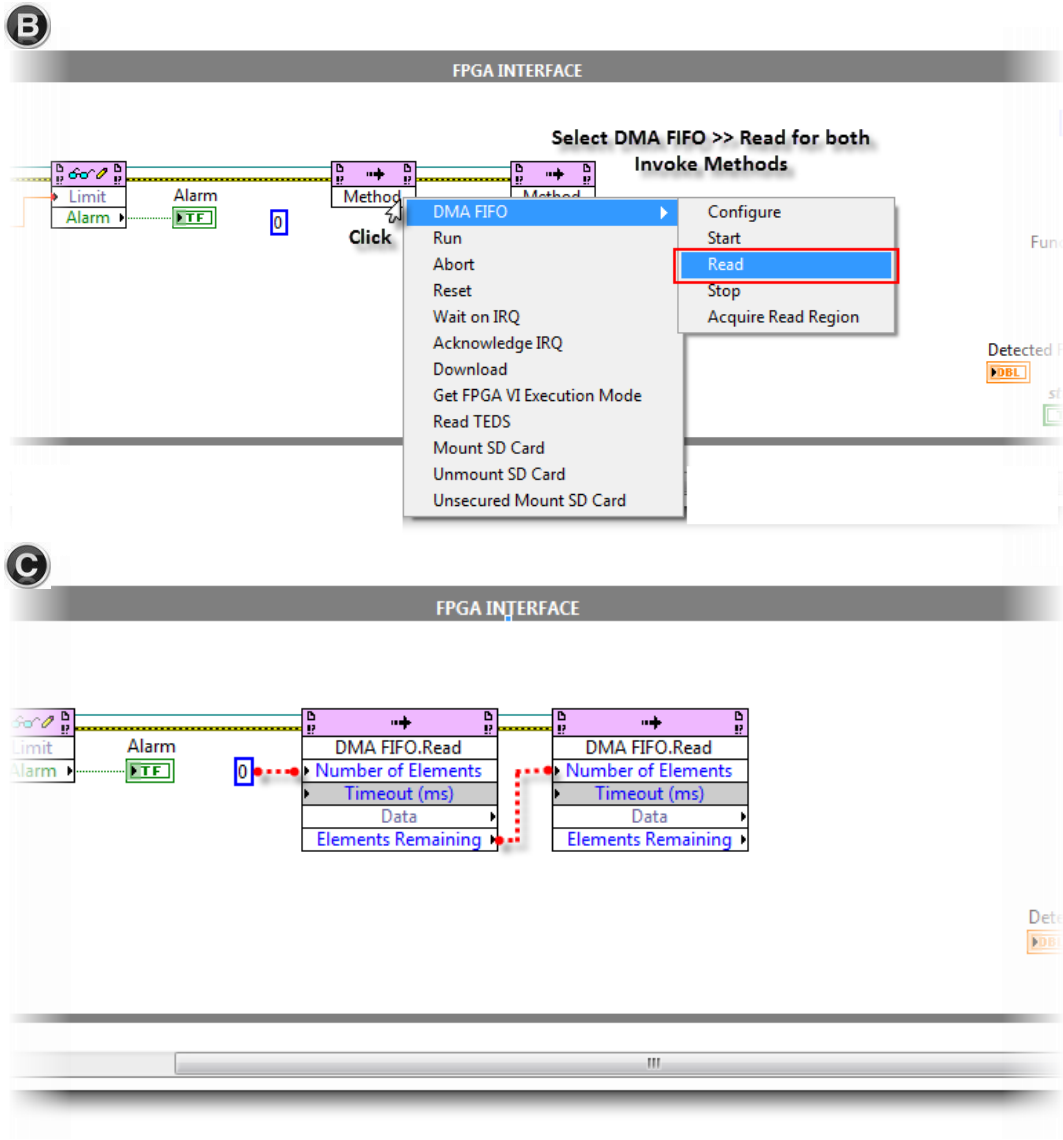
DETAILED INSTRUCTIONS

- ***Add Two Invoke Method Functions as Shown in Figure A1***
  Right-click on the block diagram to open the ***Functions Palette***. Navigate to **FPGA Interface** and drag and drop two ***Invoke Method functions*** onto the block diagram.

  This node invokes a method or action from a host VI on the FPGA VI. In this exercise, use it to access the Function Generator Input data.

- ***Connect the Invoke Method as Shown in Figure A2***
  Continue the ***Reference*** (aqua) and ***Error*** (yellow) wires coming from the ***Read/Write Control*** and connect them to the ***Invoke Method functions***.

  Connecting the FPGA reference to the ***Invoke Method*** populates the item menu with items specific to your FPGA VI.



DETAILED INSTRUCTIONS

- ***Select the Invoke Method Items as Shown in Figure B***
  Click on the **Method** text to show menu items. Under **DMA FIFO**, select **Read.**

- ***Configure the DMA FIFO Read Methods as Shown in Figure C***
  Connect the numeric constant of **0** to the **Number of Elements** input on the first ***Invoke Node.*** Connect the **Elements Remaining** output from the first ***Invoke Node*** to the **Number of Elements** input on the second ***Invoke Node***.

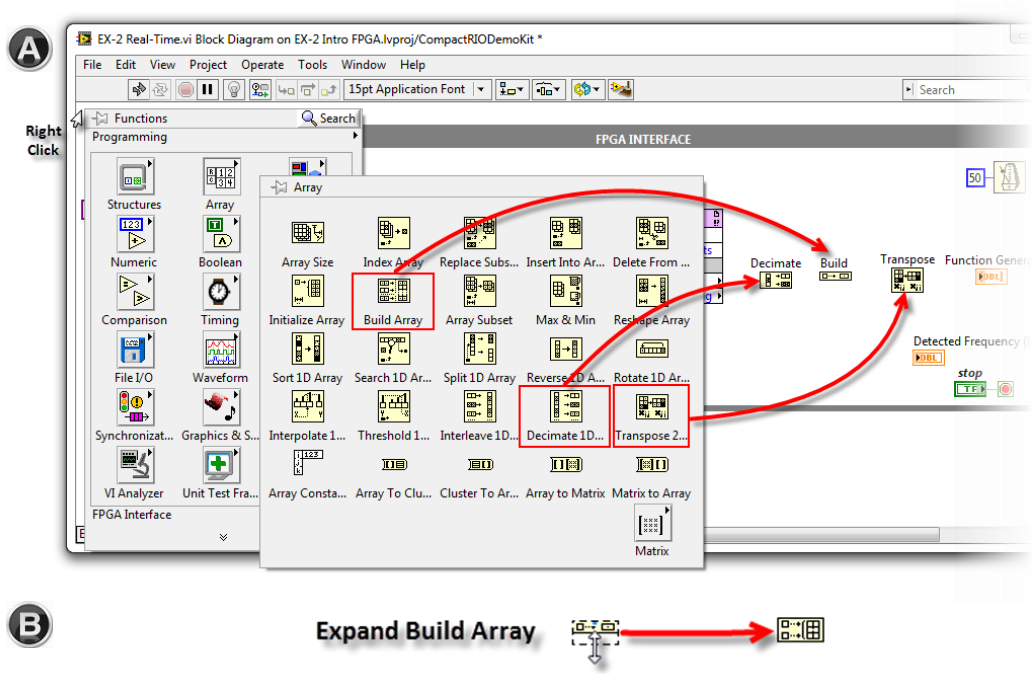
  The dual-FIFO Read is designed to prevent FIFO Overflow.

  The first Read assesses the number of remaining elements. The second read actually reads the elements.

  Using this setup, you can read all remaining elements on the FIFO and not worry about memory overflow.

19. Show the function generator data from the FPGA to the user.

Sort filtered and unfiltered data into two separate arrays. Build them into a 2D array and plot them for the user.

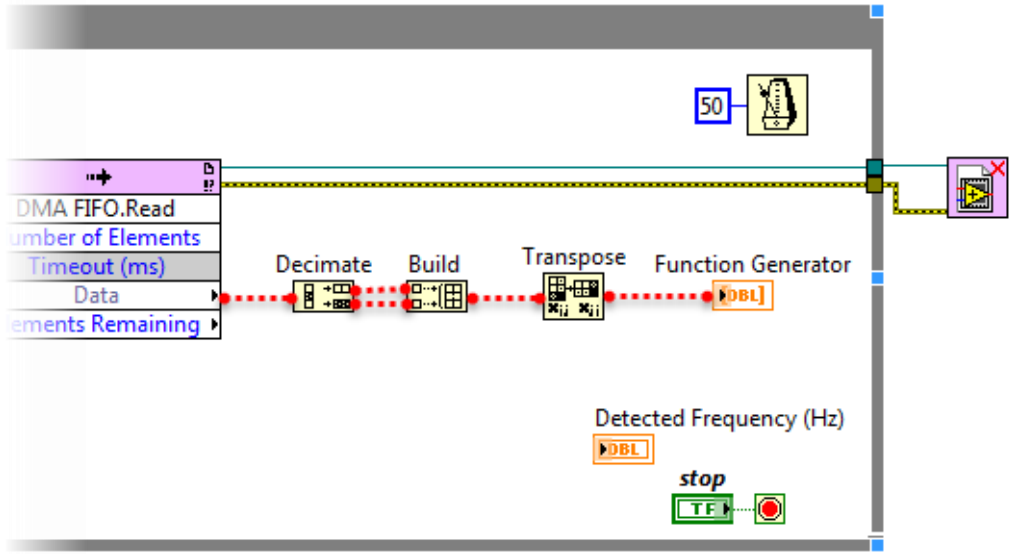


DETAILED INSTRUCTIONS

- ***Create a 2D array of data to send into the Function Generator Chart***
  Right-click on the block diagram to bring up the functions palette. Navigate to the Programming>>Arrays subpalette. Drag and drop the following three functions onto the block diagram as shown in ***Figure A***:

  1. **Decimate 1D Array** – Sorts the elements of the array into two separate arrays
  2. **Build Array** – Builds a 2D array from two 1D arrays
  3. **Transpose 2D Array**

- ***Expand the Build Array Function to show two Input Terminals***
  Move the cursor over the **Build Array** function to show the blue points from which you can expand the node. Expand the node to show the additional terminal as shown in ***Figure B***.

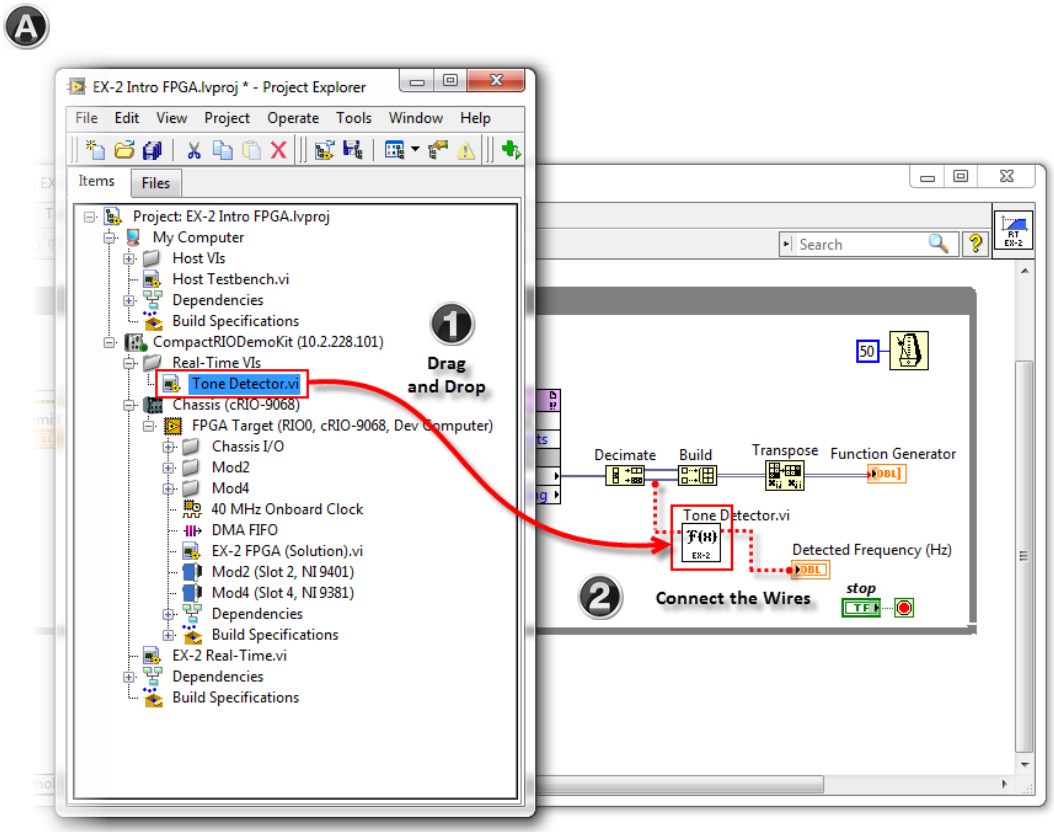


DETAILED INSTRUCTIONS

- ***Connect the Functions From the Data Output to the Function Generator Chart as Shown in Figure C***
  Connect the ***Data*** output to the ***Decimate Array*** function. Connect the two outputs from the ***Decimate Array*** function to the ***Build Array*** function. Connect the ***Build Array*** output to the ***Transpose Array*** input. Connect the ***Transpose Array*** output to the ***Function Generator Chart*** input.

20. Analyze the frequency of the signal.

Use the prebuilt Tone Detector.vi to filter out a frequency from the function generator signal.



DETAILED INSTRUCTIONS

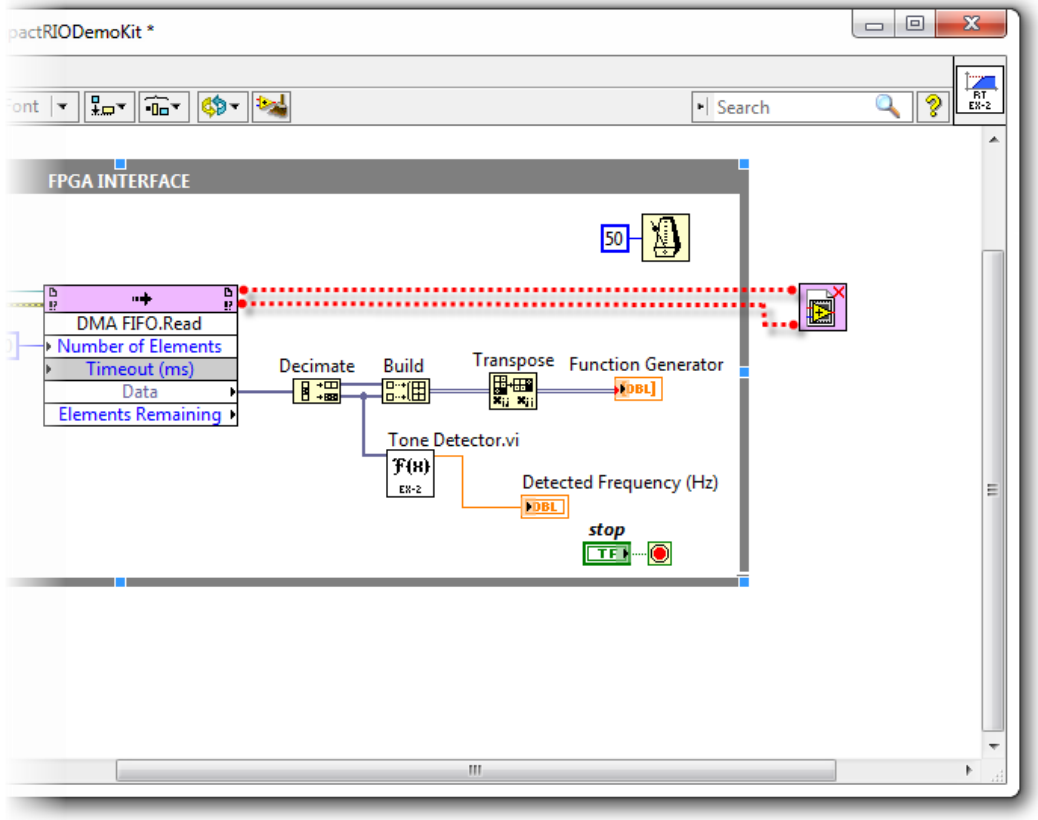
- ***Drag and Drop the Tone Detector.vi Onto the Block Diagram as Shown in Figure A***
  Drag and drop the ***Tone Detector.vi*** in the LabVIEW project under the Real-Time VIs folder in the CompactRIO target.

  This VI was prebuilt to accept an input signal and output the frequency.

- ***Connect the Unfiltered Signal to the Tone Detector.vi and Display the Frequency to the User as Shown in Figure B***
  Connect the bottom array (the unfiltered signal) to the **Tone Detector.vi**. Connect the output frequency to the **Detected Frequency (Hz)** indicator to display the frequency to the user.

21. Close the FPGA target reference.

Close the FPGA target reference at the end of the program outside the While Loop.

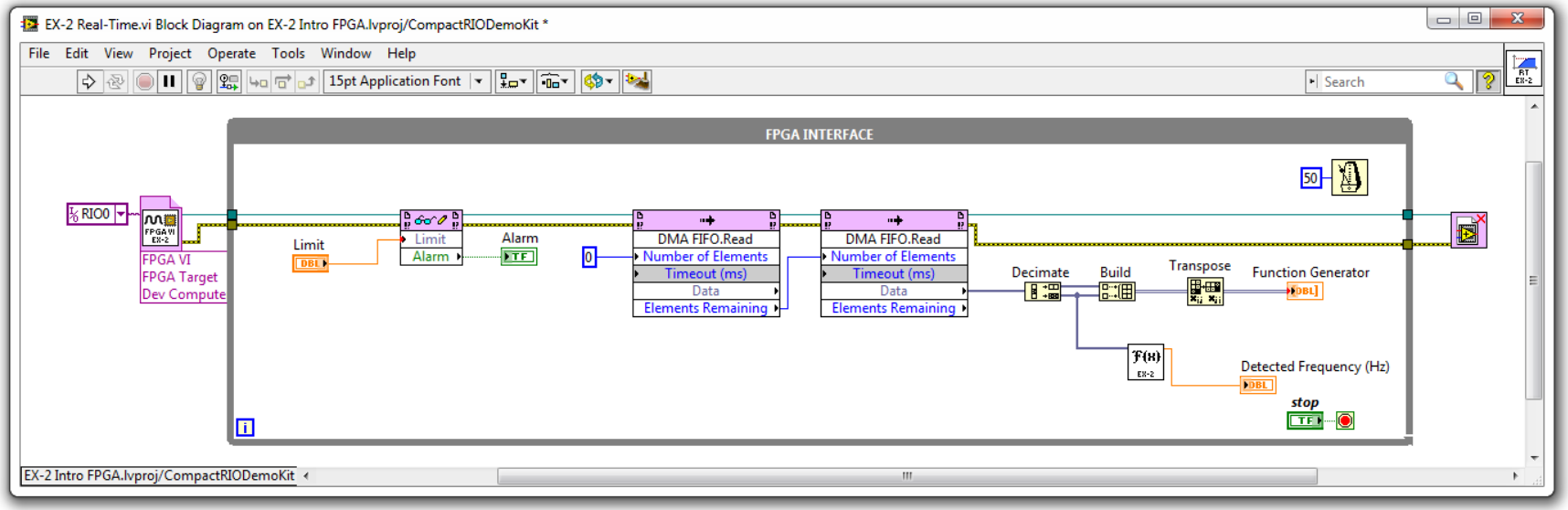


DETAILED INSTRUCTIONS

- ***Connect the FPGA VI Reference From the Invoke Method to the Close FPGA VI Reference***
  Continue the ***Reference*** (aqua) and ***Error*** (yellow) wires coming from the ***Invoke Method*** and connect them to the ***Close FPGA VI Reference***.

22. The completed block diagram for Ex-2 Real-Time.vi should look like the image below.

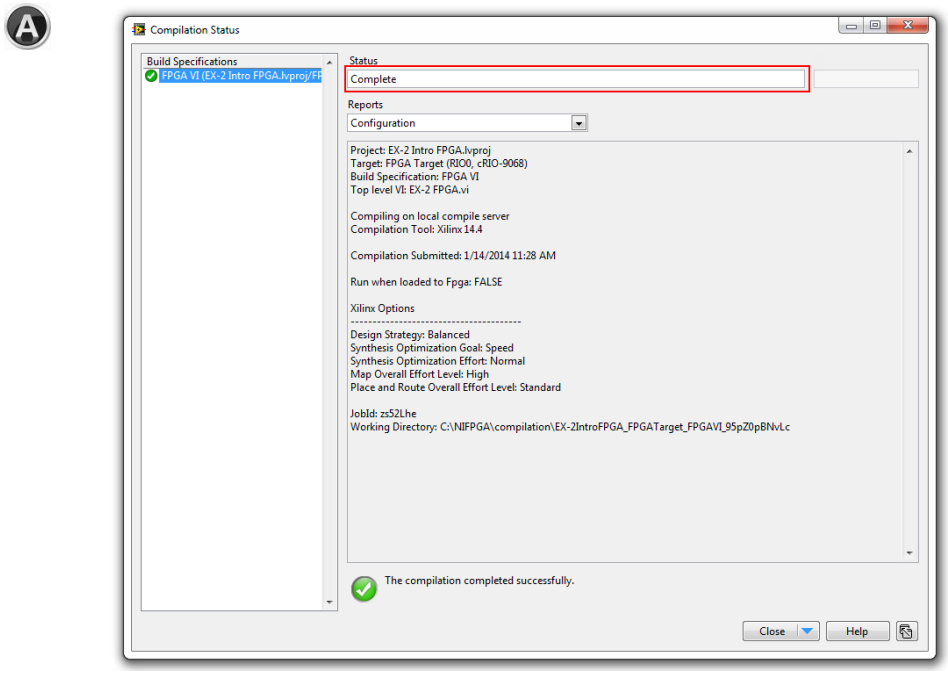
Be sure to save your work by pressing <Ctrl-S> or clicking File » Save.



Part F—RUN THE APPLICATION

23. Verify that the compilation process is complete.

At this point, the compilation of Ex-2 FPGA.vi should be complete. Run Ex-2 Real-Time.vi and view function generator signals on the plot. Interact with the limits and watch the alarm trigger to verify the correct behavior of the application.



DETAILED INSTRUCTIONS

- ***Compilation Process Complete***
  Verify that the compilation process has completed successfully. The ***Compilation Status window*** shows a summary of the performance and resource use of the FPGA as shown in
  Figure A. If the code has not finished compiling, wait for the compilation to complete.

  NOTE: If you closed the ***Compilation Status window***, you can reopen it by right-clicking on the ***FPGA Main specification*** and selecting ***Display Compilation Result***.

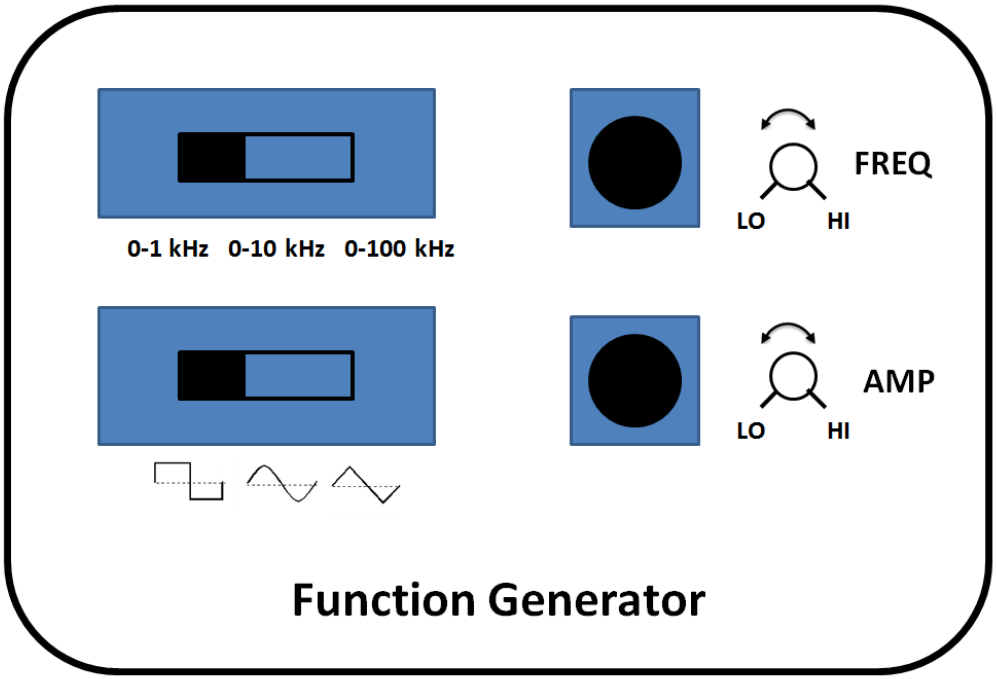
  Device Utilization indicates the percentage of FPGA elements that the FPGA application uses.

  Timing is a summary of the FPGA clocks as estimated during the mapping of the FPGA VI.

  Compilation time depends on the size of the VI, processor speed, and amount of memory in the computer on which you are compiling.

24. Set up the function generator on the CompactRIO demo box.

Set the amplitude and frequency of the function generator with settings similar to your simulation settings for comparison purposes.



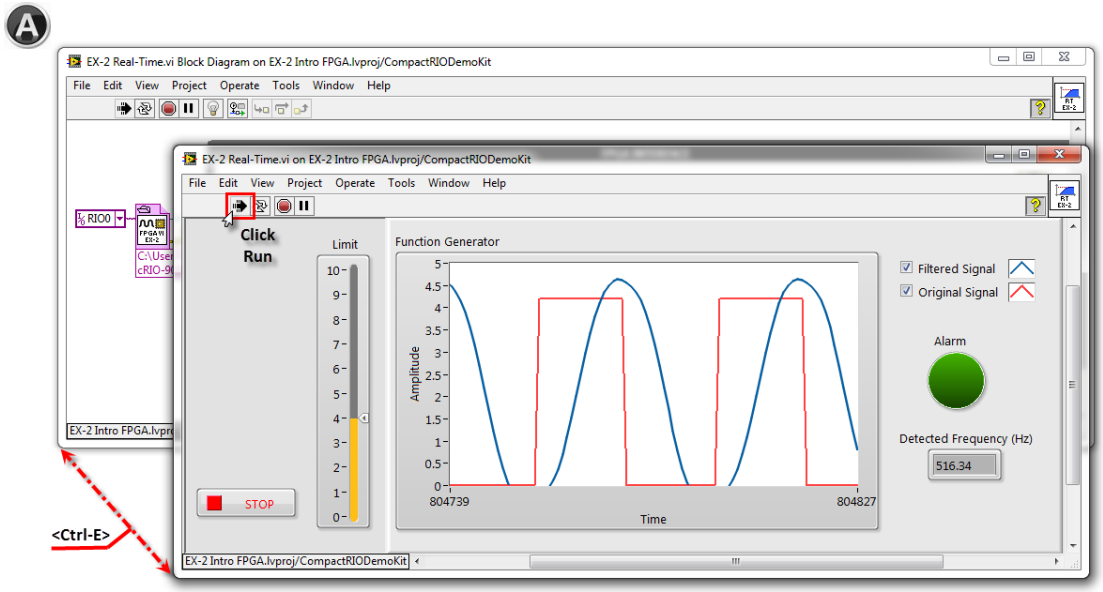
DETAILED INSTRUCTIONS

- ***Configure the Function Generator on the CompactRIO Demo Box***
  Set the ***Frequency*** to 0 kHz–1 kHz and the ***WaveType*** to Square.

  This configuration produces a waveform like the one you simulated in the ***Host Testbench.vi***.

25. Run the Ex-2 Real-Time.vi on the CompactRIO target.

Run Ex-2 Real-Time.vi and view the function generator reading on the plot.



DETAILED INSTRUCTIONS

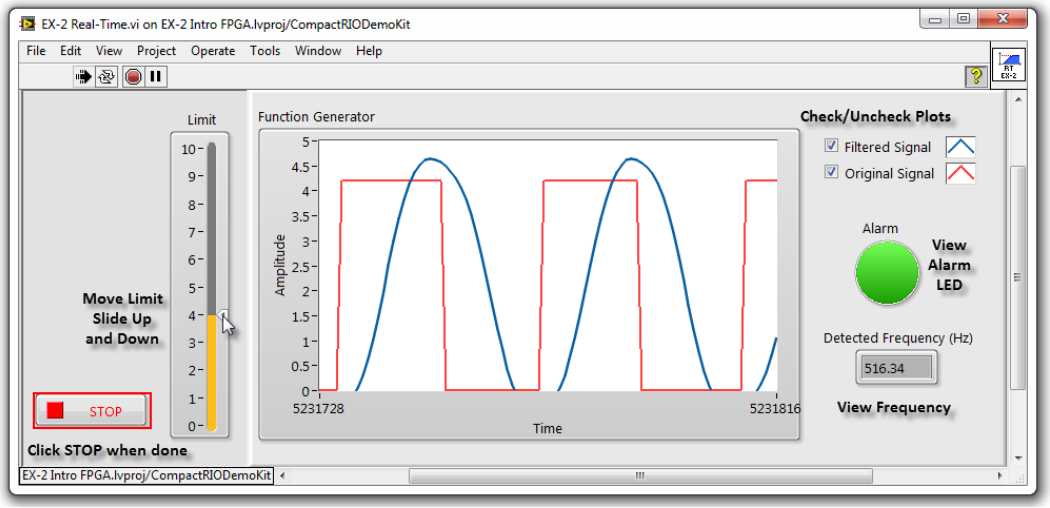
- ***Run Ex-2 Real-Time.vi***
  Switch back to the Front Panel by pressing **<Ctrl-E>** and run the program by pressing the arrow button at the top of the front panel as shown in ***Figure A.*** Save the VI when prompted to do so.

  NOTE: A warning window may appear indicating that the chassis is configured in **Scan Engine Mode** (Exercise 1). Close the warning window, go to the project tree, right-click on the chassis, and select **Deploy**.

  Click the **Apply** button on any subsequent warnings during this process. This changes the configuration of the chassis to **FPGA mode**.

26. Interact with the front panel of Ex-2 Real-Time.vi.

Adjust the waveform types, frequency, and amplitude and watch the response on the front panel display. Interact with the limits and watch the alarm trigger to verify the correct behavior of the application.



DETAILED INSTRUCTIONS

- ***Adjust the Waveform Type, Frequency, and Amplitude on CompactRIO Demo Box***
  Use knobs and switches on CompactRIO Demo Box to adjust waveform, frequency, and amplitude. Watch the signal on the user interface.

  NOTE: Due to your sampling rates, you cannot accurately read signals above 1 kHz. Do not adjust the frequency switch beyond 0 kHz to 1 kHz.

- ***Interact With the Front Panel***
  Adjust the limit control, select which plots to display, and view alarm and frequency data. Press the Stop button to stop the program.

Part G—CHALLENGE

27. Plot the power spectrum.

Add a graph of the power spectrum from the signal to the **Ex-2 Real-Time.vi**.

<END OF EXERCISE 2A: FPGA-BASED BUTTERWORTH FILTER>

ADDITIONAL RESOURCES

BUILD YOUR OWN EMBEDDED SYSTEM WORKSHOP

In the NI Build Your Own Embedded System hands-on workshop, focus on extending your NI LabVIEW skills into FPGA-based embedded design using NI reconfigurable I/O (RIO) hardware.

Purchase the LabVIEW RIO Evaluation kit through the registration process and attend the workshop to receive the following:

- 90-day evaluation of LabVIEW and the LabVIEW FPGA and LabVIEW Real-Time modules
- Board-level NI RIO evaluation hardware device and daughterboard for easy I/O interfacing
- Introduction to the LabVIEW RIO architecture with a qualified NI instructor
- Hands-on experience building your first FPGA-based RIO embedded system with the evaluation kit

To find an event in your area, visit ***ni.com/byoes***.

LabVIEW RIO EVALUATION KIT

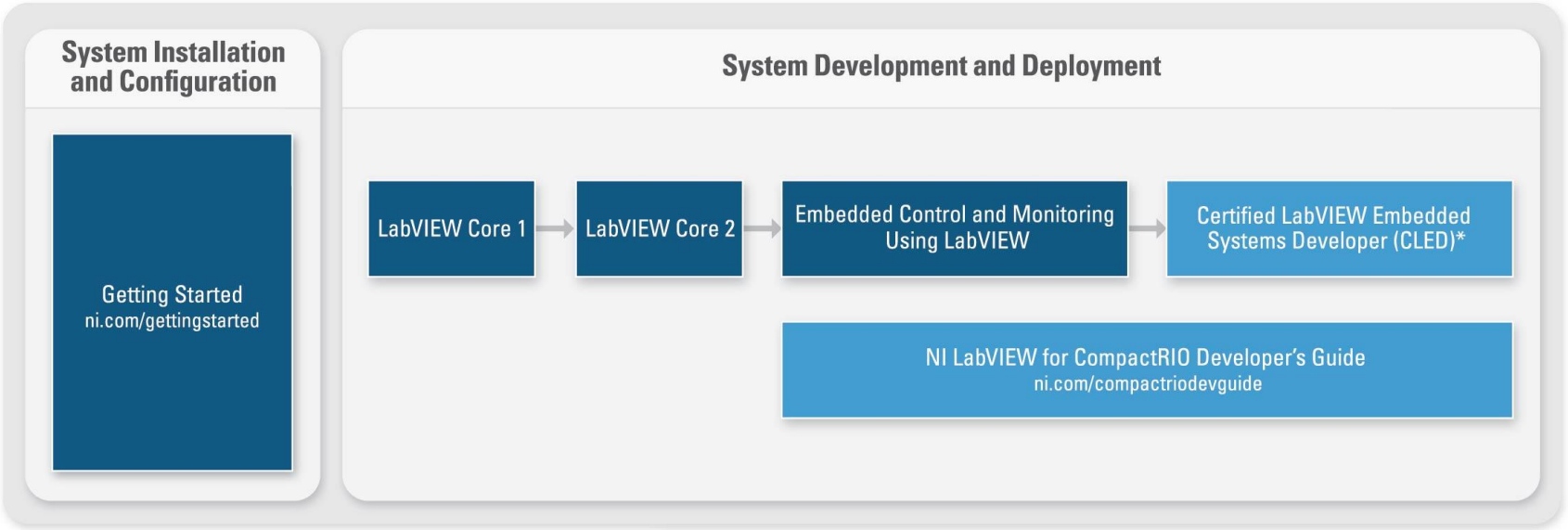
NATIONAL INSTRUMENTS LabVIEW™

Using the evaluation kit, develop an embedded system with the LabVIEW RIO architecture. LabVIEW system design software helps you program NI RIO hardware, which includes a real-time processor, FPGA, and I/O. NI CompactRIO hardware uses this same architecture for prototyping through deployment with a flexible array of configuration, expansion, and NI C Series module I/O options.

The kit includes an extended evaluation of the LabVIEW FPGA and LabVIEW Real-Time modules; an NI RIO evaluation device; a daughterboard for easy I/O interfacing; a step-by-step tutorial; and numerous fully documented, ready-to-run examples of common embedded tasks implemented in LabVIEW.

To learn more, visit ***ni.com/rioeval***.

NI LabVIEW Real-Time and NI LabVIEW FPGA RECOMMENDED RESOURCES AND TRAINING OPTIONS



* A CLD or higher is required before attempting the CLED exam