



# **Hands-On: CompactRIO Part II**

## **Programming with LabVIEW FPGA**

# NI LabVIEW



- System design software
- Graphical programming
- Built-in analysis and control functions
- One software tool for HMI, PAC, and FPGA



2011 NI TECHNICAL SYMPOSIUM



LabVIEW is a powerful graphical programming language that has been around for over 20 years. Its intuitive development environment and built-in control, analysis, and logging functions allow you to more easily develop complex industrial measurement and control applications. This single software tool can be used to program the entire system, including the controller, HMI, and even the FPGA.

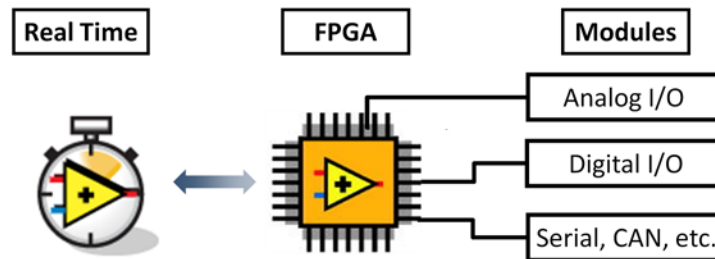
## CompactRIO Platform Components



NI CompactRIO is one of the fastest growing PAC platforms on the market, which is highly differentiated by its built-in FPGA.

3 Parts, RT Proc, IO Modules, FPGA make up each cRIO

# CompactRIO and FPGA



- **Reconfigurable FPGA** for high-speed and custom I/O timing, triggering, and control
- **I/O modules** with built-in signal conditioning for connection to sensors/actuators
- **Real-time processor** for reliable measurement, analysis, connectivity, and control

2011 NI TECHNICAL SYMPOSIUM



At the heart of the CompactRIO architecture is the FPGA, giving you the highest reliability, performance, and flexibility

An FPGA is a special kind of chip that rewires itself to implement your application's code in hardware.

It is like having a blank slate of custom circuitry that you can rewire to implement your application.

I like to think of it as a primordial ooze of transistors ☺

Typically FPGAs require an engineer who is an expert in VHDL or another hardware descriptor language.

Fortunately, LabVIEW makes it is easy for you to program FPGAs with the same graphical development environment used to create applications on your PC.

Has anyone ever tried to implement an FFT algorithm on a FPGA using an HDL? About how long did that take to program? (usually 10+ hours if done from scratch)

In LabVIEW this can be done using a simple loop and a couple of pre-made functions in less than an hour (Includes compile time!!!)

The FPGA connects to I/O modules that provide your interface to the outside world. As you can see from the diagram, the I/O modules have built in signal conditioning for direct connectivity to industrial sensors and actuators, motors, drives, proximity switches, hydraulic and pneumatic systems.

NI offers I/O modules with sample rates up to 800kS/sec and up to 24bit ADCs

Augmenting this architecture is the processor, which runs a real-time operating system, providing the high performance measurement, analysis, and control capabilities that you would expect from a PC.

# CompactRIO Applications

## Machine Control

- **Packaging/Processing**
  - High-speed motion control, batch control, discrete control
- **Heavy Machinery Control**
  - Real-time signal processing and control of power electronics, hydraulic systems
- **Semiconductor/Biomed**
  - Custom motion and vision inspection, material handling

## Machine Monitoring

- **Machine Condition Monitoring**
  - Bearing order analysis, lubrication monitoring, cooling, combustion, and so on
- **Mobile/Portable DSA, NVH**
  - Noise, vibration, harshness, dynamic signal analysis, acoustics
- **Distributed Acquisition**
  - Central controller with distributed I/O nodes over Ethernet/wireless

## In-Vehicle Data Acquisition

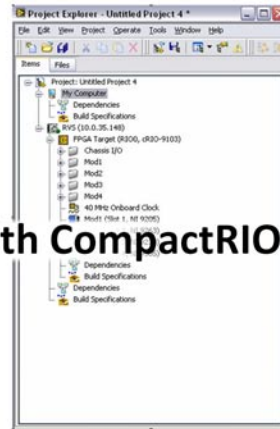
- **In-Vehicle Data Acquisition**
  - Automobiles, motorcycles, recreational vehicles, research aircraft, trains
- **Engine and ECU test cells**
  - HIL testing of engines and engine controllers, sensor simulation using FPGA
- **Rapid Control Prototyping**
  - Automotive/aerospace control prototyping



CompactRIO has been a very successful platform for NI. Because we are so well known for test and measurement, we found a quick initial success with CompactRIO in measurement applications such as in vehicle datalogging, as well as machine condition monitoring. As companies are becoming more familiar with the RIO architecture however, CompactRIO is being used in more mission critical control applications such as machine control and biomedical applications.

# Exercise 1

- **Setup LabVIEW Project With CompactRIO**



2011 NI TECHNICAL SYMPOSIUM



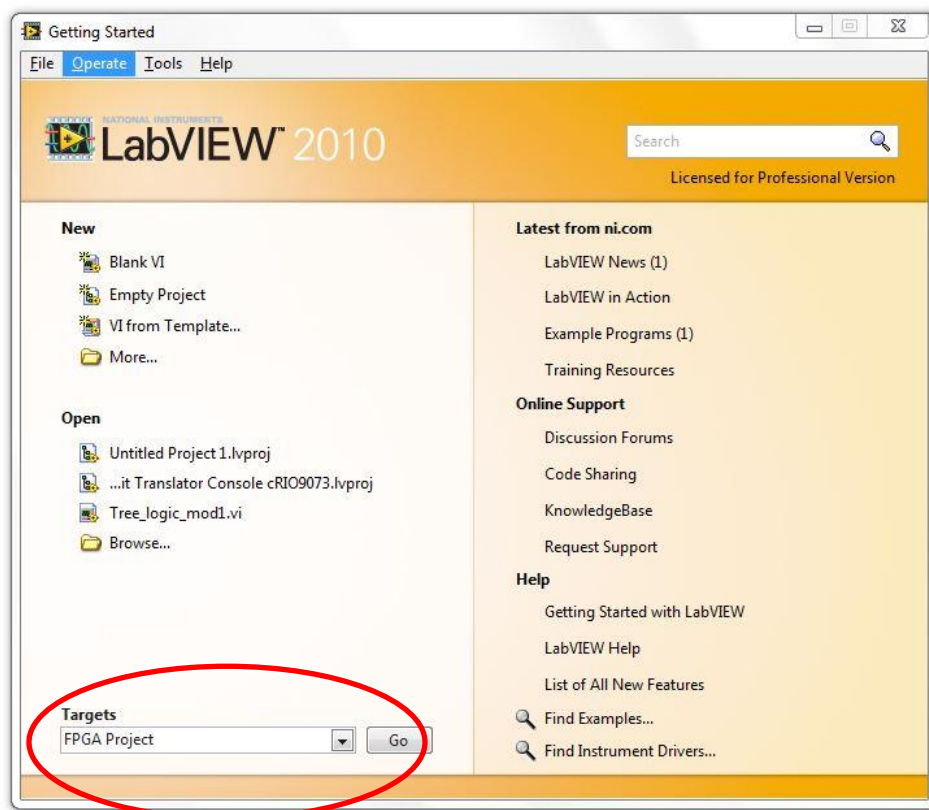
## Exercise 1 - Creating the FPGA Project

In this section you will learn the following:

- How to configure a system using the FPGA Project Wizard.
- Detection of the CompactRIO modules.

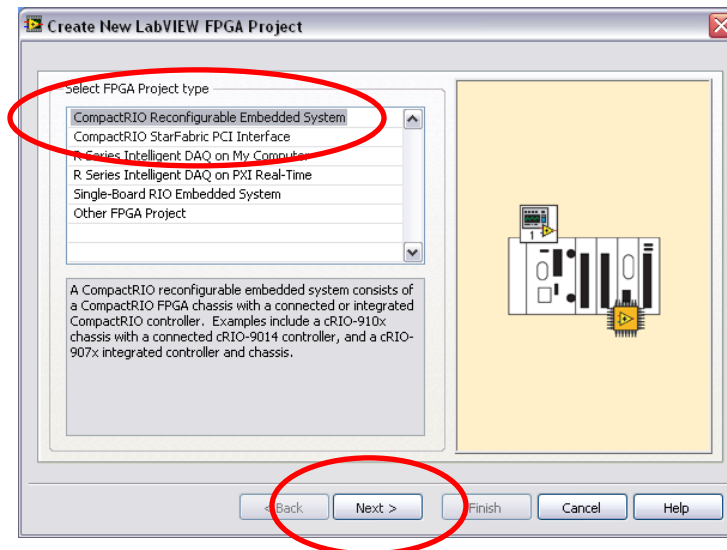
1. Launch **LabVIEW 2010** and select **FPGA Project** from the **Targets** section and click **Go**.

*LabVIEW 2010 has built-in functionality like the FPGA Project Wizard that makes creating and configuring FPGA applications much easier. With the FPGA Project Wizard you use configuration based dialogs to set-up the FPGA hardware for your application.*

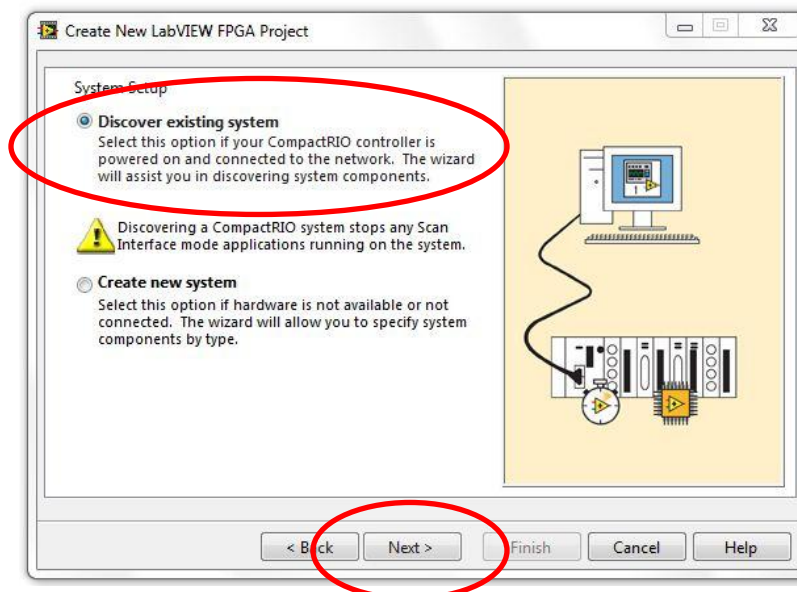




2. Select **CompactRIO Reconfigurable Embedded System** from the project type list and click **Next**.

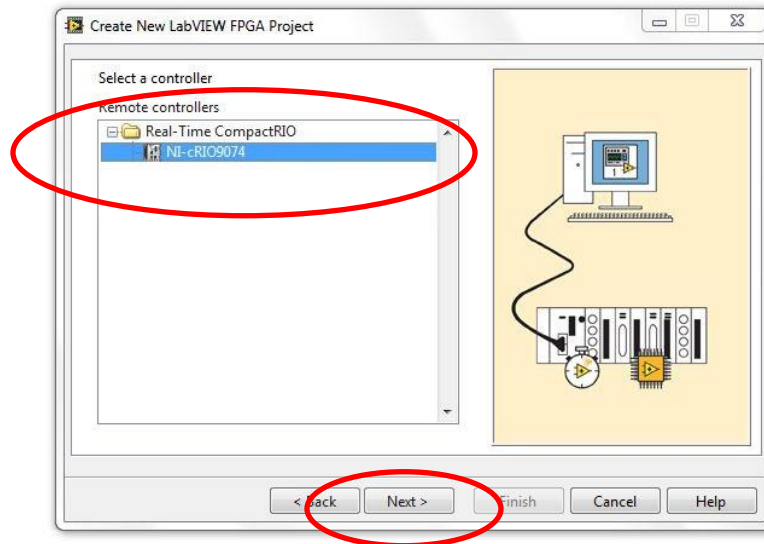


3. Select **Discover existing system** under System Setup and click **Next**. During this step the wizard discovers all CompactRIO systems on your subnet. Note that any applications running on the CompactRIO will be stopped.

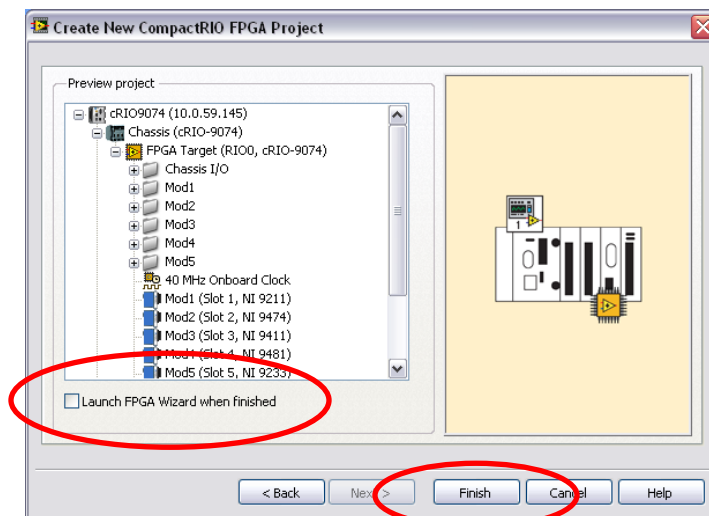




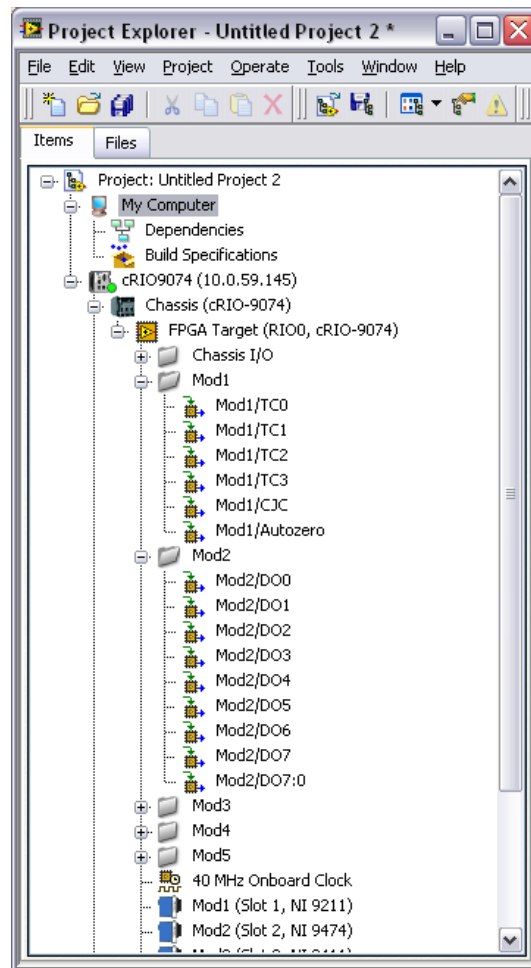
4. **Select your CompactRIO system** from the list and click **Next** and the wizard will discover all C Series I/O modules currently installed in your CompactRIO chassis and add them to a LabVIEW Project. This step can take a few minutes to complete.



5. Ensure the **Launch FPGA Wizard when finished** check box is **unchecked** and click **Finish** to create the LabVIEW project with your FPGA I/O. Notice that LabVIEW now creates and opens a project with all of the I/O available in your CompactRIO system.



6. Expand the FPGA Target folder to view the I/O modules. Expand the “Mod1” and “Mod2 sub folders to see the I/O Node items associated with each module. The LabVIEW Project should look similar to the image below.

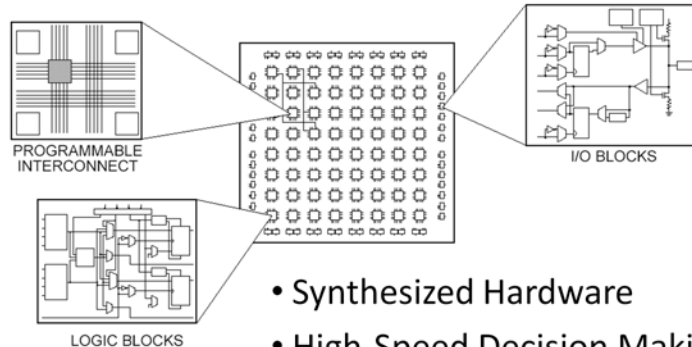


7. Rename the I/O variables below to the following by right clicking the I/O variable and selecting **Rename . . .**

- |             |   |               |
|-------------|---|---------------|
| 1. Mod1/TC0 | → | Temperature   |
| 2. Mod2/DO0 | → | Light         |
| 3. Mod2/DO1 | → | Fan           |
| 4. Mod3/DI0 | → | User Button 1 |

8. Keep the project open for Exercise 2.

# The Advantages of FPGA



- Synthesized Hardware
- High-Speed Decision Making (ns)
- Precise Determinism (ps)
- Truly Parallel Execution

2011 NI TECHNICAL SYMPOSIUM



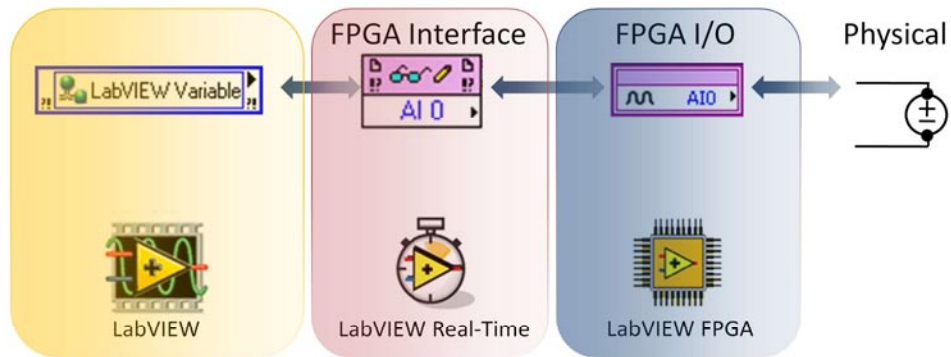
Synthesized Hardware means that you are actually building a circuit to implement your application.

Because the application exists in hardware, the On-Board clock determines how fast we can make decisions in our code. At the standard 40MHz clock, this equates to a 25ns period between outputs.

Another benefit of the application being implemented in hardware is that there is no jitter introduced by an operating system or other software. Since the flow of electrons is more or less constant from one execution to the next, we have PicoSecond Determinism.

Since any independent logic is implemented on slices that have no connection to each other, all logic is going to execute in a truly parallel fashion.

# Programming With LabVIEW FPGA



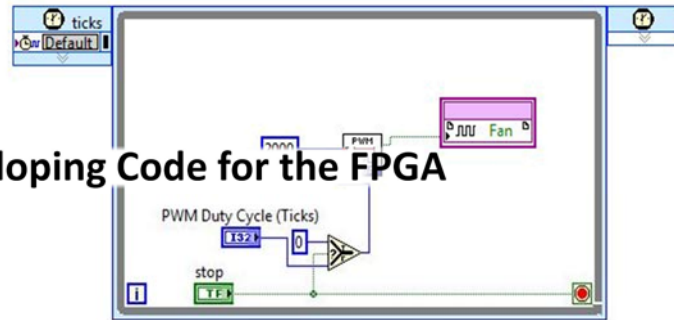
2011 NI TECHNICAL SYMPOSIUM



Programming the FPGA requires creating at least two Vis (or programs). The process normally starts with creating the FPGA VI, which interacts with the I/O, provides high speed control, and advanced control/analysis functionality.

## Exercise 2

### Developing Code for the FPGA



2011 NI TECHNICAL SYMPOSIUM




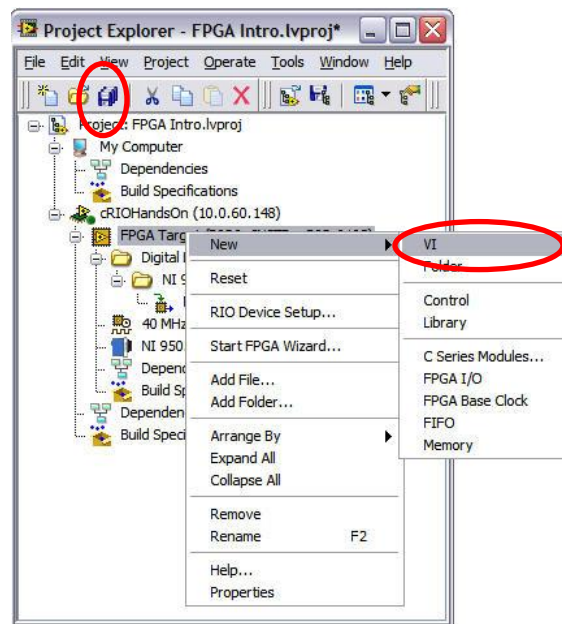
## Exercise 2 - Developing Code for the FPGA

In this section you will learn the following:

- How to create a new FPGA VI.
- Using FPGA input and output.
- Develop, target, download, and run an FPGA VI in simulation mode.

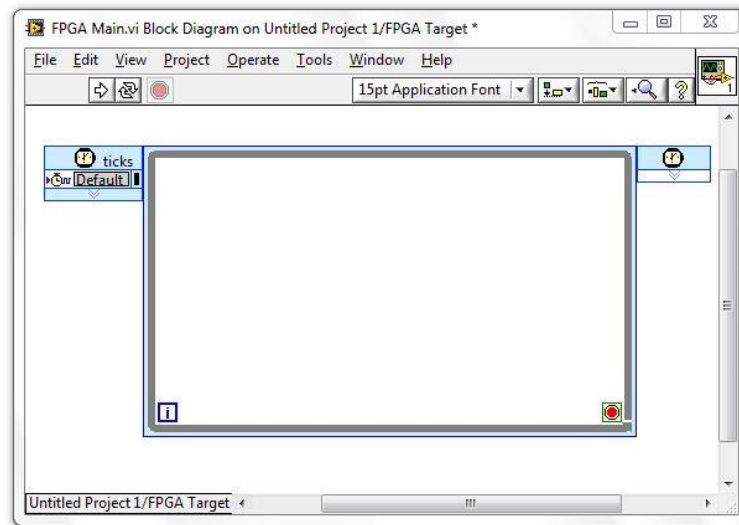
Our application requires us to start monitoring temperature when started. Pressing the **User Button 1** turns on the heat source (light) and will start control of the temperature of the chamber. Each iteration of the loop compares the current measured temperature to the threshold and will activate or deactivate the cooling system (fan) based on the value.

1. With the C Series I/O added to your project by the project wizard, you are ready to begin creating the FPGA VI that reads the inputs and generates the outputs of the program. Right-click on the **FPGA Target** and select **New » VI** to create a new LabVIEW FPGA application. After the new VI pops up, temporarily toggle back to the project and press Save All (  ) on the toolbar. Save the project as **FPGA Hands On Project.lvproj** and the new VI as **FPGA Main.vi** in C:\NITS\cRIO 2 FPGA\Hands On Work



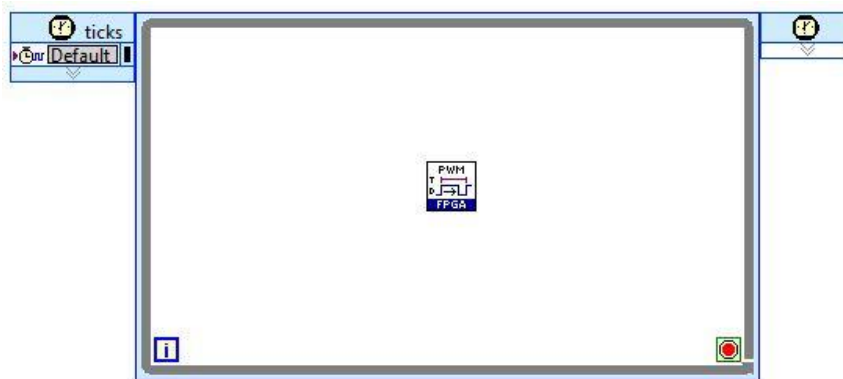
2. Navigate to the block diagram window of your new application. Right-click in the white area of the block diagram to display the **Functions** palette. Click on the thumbtack icon in the top left corner of the **Functions** palette to tack it down.

3. Add one timed loop to generate a PWM signal that precisely controls the amount of power delivered to the fan. Place a **Timed Loop** from the **Programming » Structures » Timed Structures** palette on the block diagram. This loop uses the FPGA base clock by default and will execute all code contained in it at 40 MHz, it is also referred to as a Single Cycle Timed Loop (SCTL). Ensuring that all code is capable of executing at this speed is finalized when the VI is compiled (later).



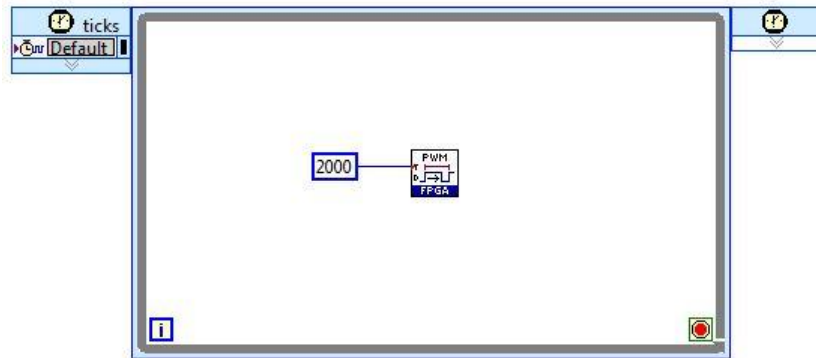
4. On the **Functions** palette, navigate to **Select a VI**, browse to this exercise folder at `C:\NITS\cRIO 2 FPGA\LabVIEW FPGA and CompactRIO Hands On\ Support VIs`, and select **Pulse Width Modulation (FPGA, Use in SCTL).vi**. Place this function block inside the Timed Loop. This VI is prewritten to save time.

*This is an example of bringing in modular IP that you have already created, which is great for code reuse. Once you have placed the PWM VI in the loop, double-click it to examine to code behind it. Once you are done exploring the VI, close and return to your current application.*

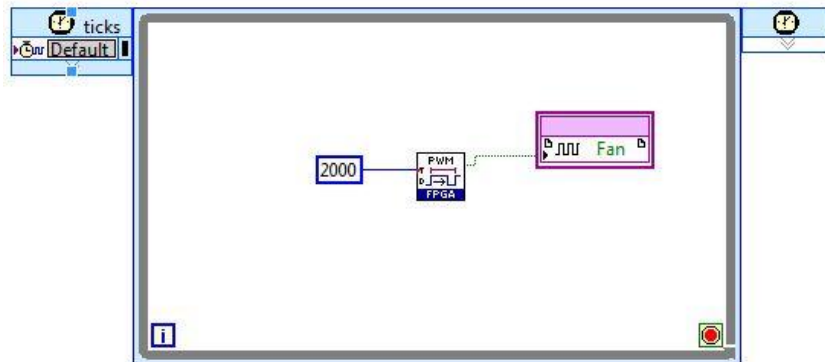




5. There are many inputs and outputs to this PWM VI, but you use only a few. Right-click on the **PWM Period (Ticks)** input and select **Create » Constant**.



6. From the FPGA Hands On Project window, drag and drop the **Fan I/O variable** inside the timed loop and to the right of the PWM VI. The Fan I/O variable is located under the Mod2 item in the project tree.



- 

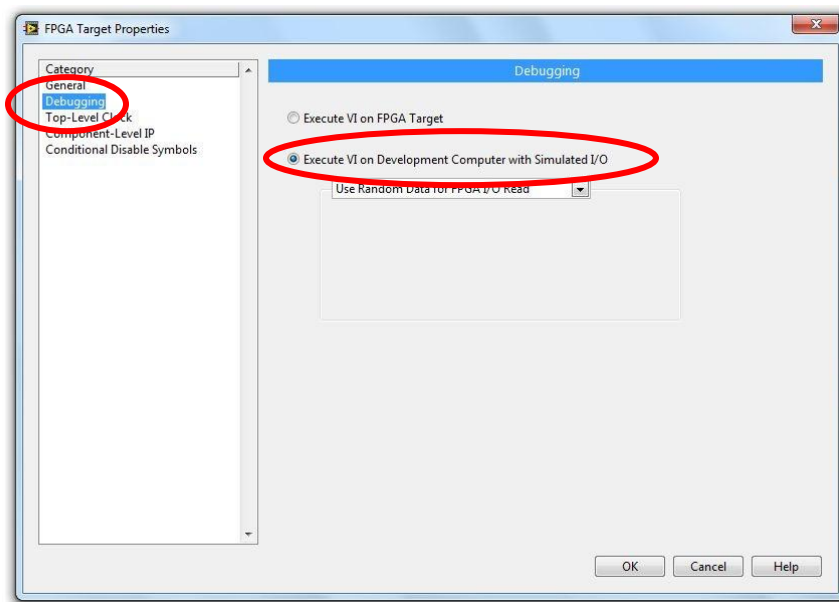
- 



**NATIONAL  
INSTRUMENTS™**

- Now we will do some preliminary debugging of the VI to make sure our logic is correct before beginning the compile. Right click the FPGA Target item in the LabVIEW Project and select **Properties** to open the FPGA Target Properties window. Click on **Debugging** and select the **Execute VI on Development Computer with Simulated I/O** option and click **OK**.

*The FPGA VI we created so far will now run in simulation mode and will not need to be compiled before running. Before deploying the actual application to the FPGA Target to run with real I/O we will need to compile, but simulation allows us to double check our logic and step through the code without this step. Note: the **Use Custom VI for FPGA I/O** option allows you to configure specific simulated data for troubleshooting. We are using random data instead, which is suitable for our purposes here.*

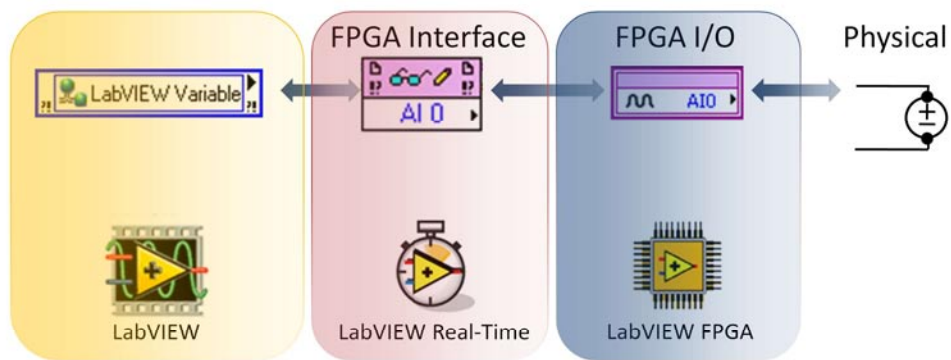


- Re-open and navigate to the block diagram of your FPGA Main VI and click the Highlight Execution (💡) icon and then run your VI. Note that you can watch the data flow in the FPGA VI as it is running and verify that the program is functioning as you expected.

*Simulation mode provides a powerful way to debug and troubleshoot your application with advanced debugging techniques such as highlight execution, break points, and probes.*

- Stop execution and close your VI. Open the FPGA Target Properties window again from the project and select the **Execute VI on FPGA Target** option inside of the **Debugging** category. **Do not compile.** Keep the project open.

# Programming With LabVIEW FPGA



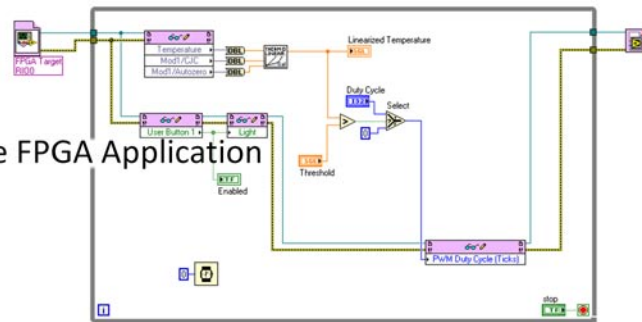
2011 NI TECHNICAL SYMPOSIUM



Now that we've programmed the main FPGA algorithm, The second step in creating your program is to create an RT FPGA interface VI that will handle the communication with the FPGA.

### Exercise 3

## Completing the FPGA Application



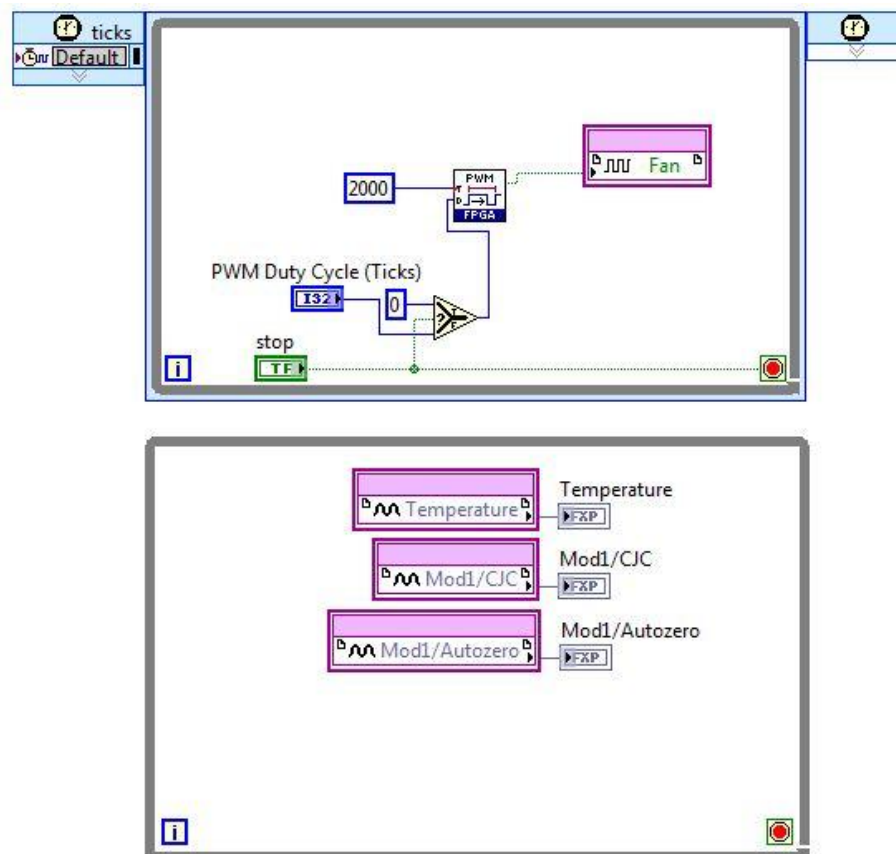
2011 NI TECHNICAL SYMPOSIUM



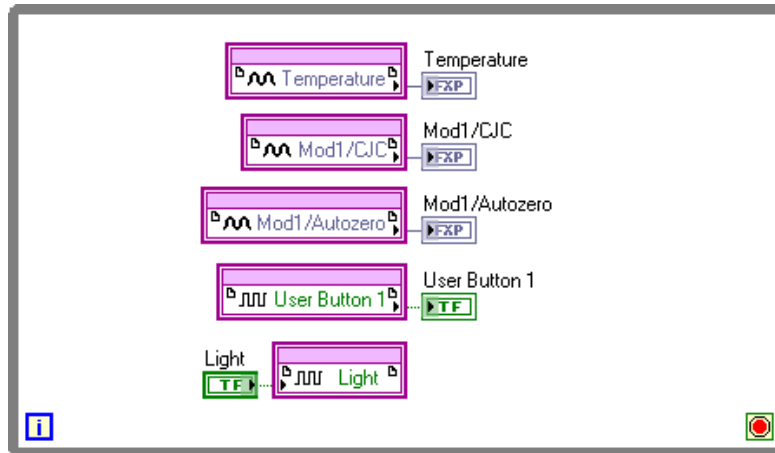
### Exercise 3 - Completing the FPGA Application by Implementing a Real Time Host


In this section we will continue development of the FPGA Main VI and will add additional I/O channels to our program. We will also compile our FPGA Main VI and create the final real-time host interface for this application.

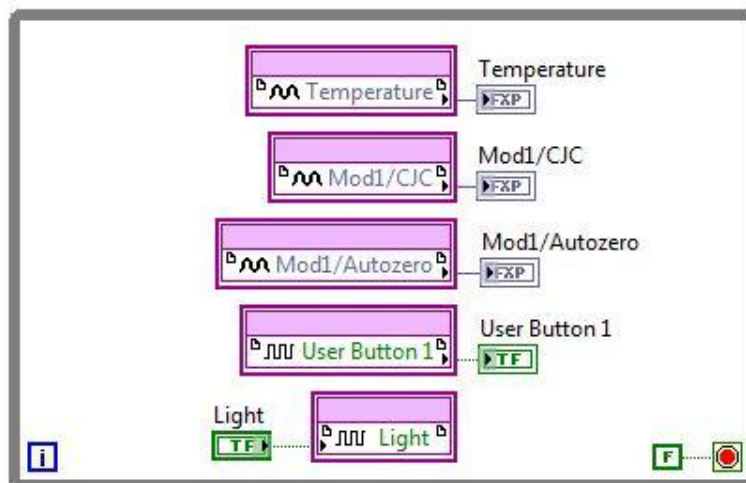
1. Open the **FPGA Main VI**. Right click the block diagram and draw a new **while loop** below the Single Cycle Timed Loop you created in the previous exercise. Expand the Mod1 item in the LabVIEW Project Tree and drag and drop the **Temperature**, **Mod1/CJC**, and **Mod1/Autozero I/O** nodes inside the while loop. Then **right click** the variable name output and **create** indicators for each I/O node as shown below.




2. Drag and drop the **User Button 1** from Mod3 inside the loop and create an indicator on its output. Then add the **Light** item from Mod2 and create a control on the input by right clicking the input and selecting **Create » Control**.



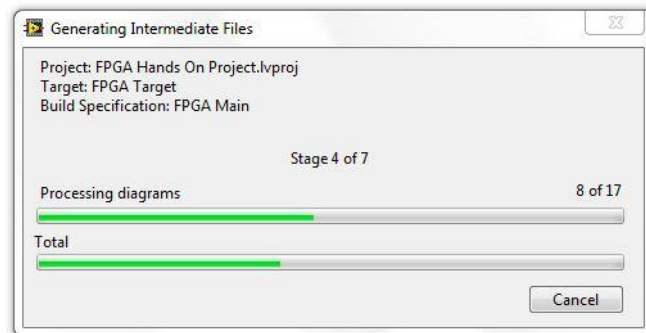
3. Create a false constant on the loop condition icon (  ) by right clicking on the input and clicking **Create » Constant**. *This loop will run as fast as the slowest I/O Node and does not need any extra timing to control it because in our case we would like it to run as fast as possible. However if we wanted to run this at a specified rate, we could add a timing VI such as a Loop Timer from the Timing palette.*



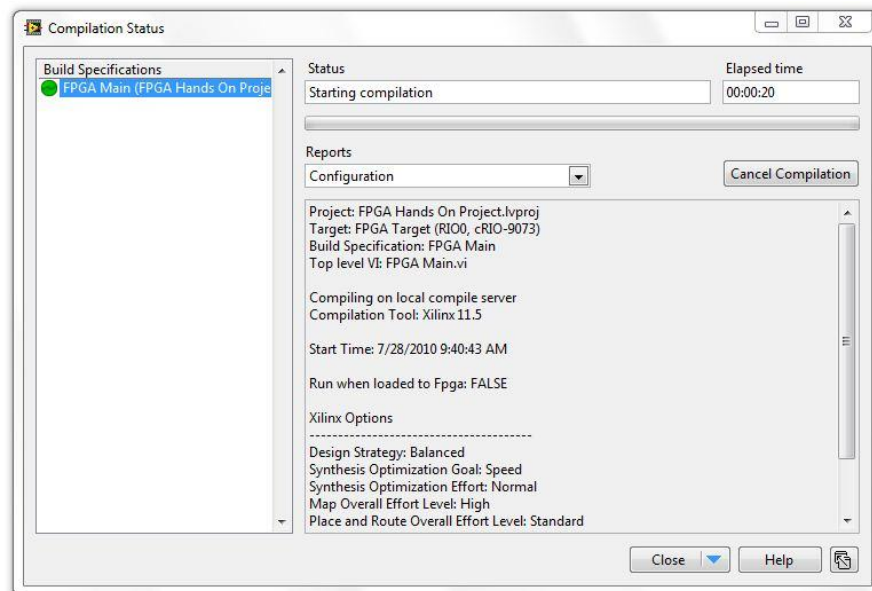
4. Our FPGA VI is now complete. **Save** and then click the **Run** (  ) button on either the front panel or the block diagram. This will start the compile process for the VI.



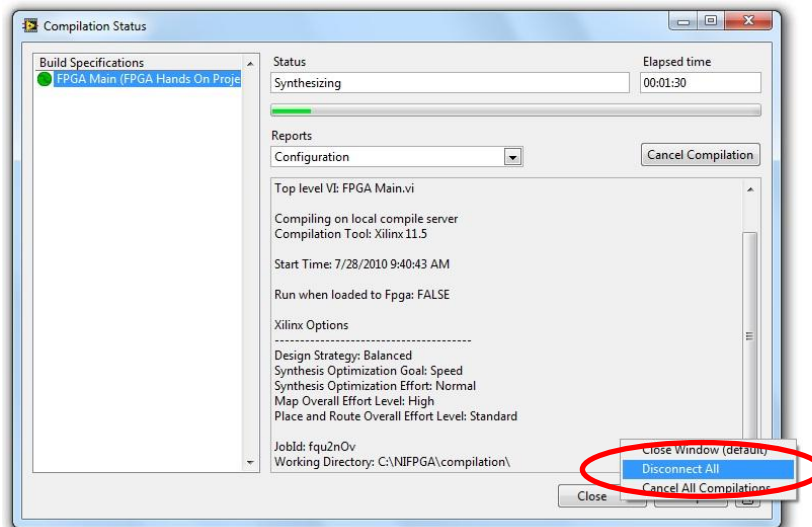
5. The next dialog shows that LabVIEW is “Generating intermediate files.” This is the step where your LabVIEW code is converted to VHDL ( a hardware description language for FPGAs).



6. The next dialog that appears is the **Compile Status** window. The **Compile Server** is based on Xilinx tools for compiling the VHDL converted from the previous step. This process can take a significant amount of time depending on the amount of logic present in your LabVIEW FPGA program. However, this wait time is expected whether you are programming in LabVIEW FPGA or any other hardware description language for targeting FPGAs.

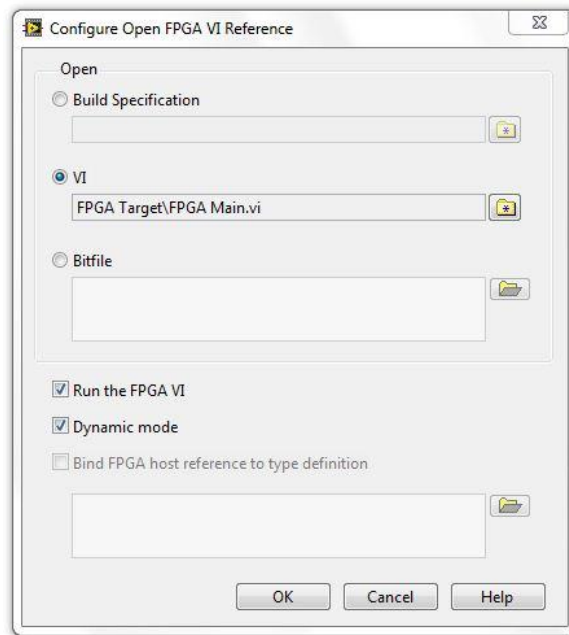


7. We will not wait for the compile to complete however and will disconnect from the compile to continue programming our RT host interface while it completes. To do so expand **Close** by clicking the arrow and then click the **Disconnect All** button on the compile window.

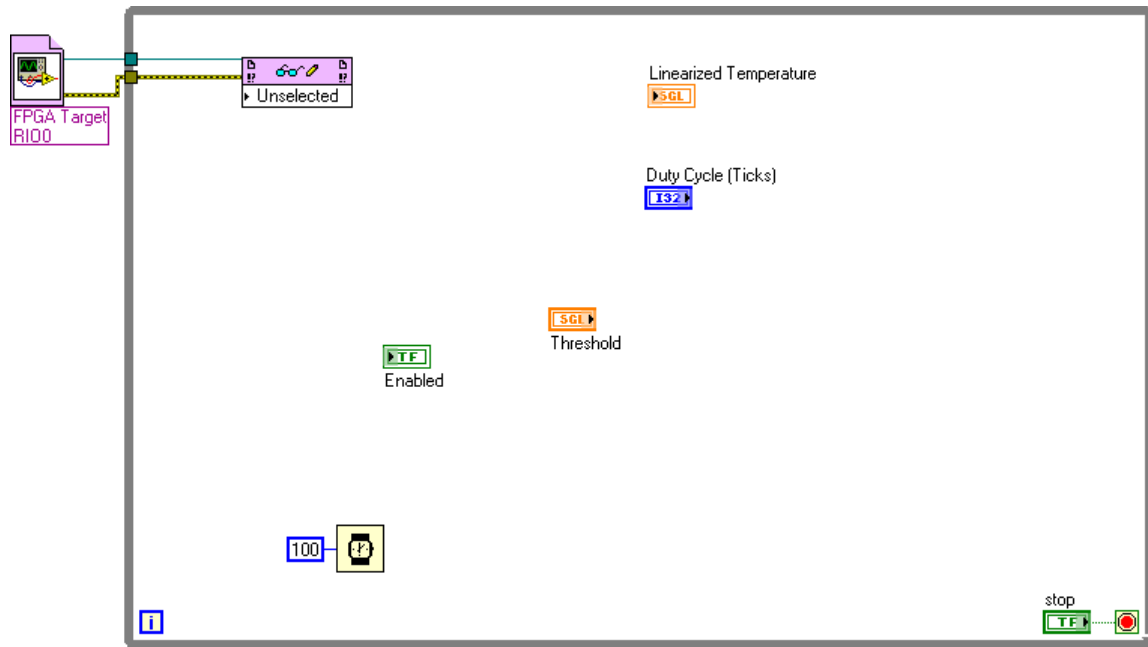


8. Now we will begin creating the Real-Time host VI. A simple VI with a prebuilt user interface has already been created to help save time in this session. To open this VI right click the **cRIOHandsOn** controller item in the LabVIEW Project tree and select **Add » File**. Browse to the *C:\NITS\cRIO 2 FPGA\LabVIEW FPGA and CompactRIO Hands On\ Support VIs* folder and select the **RT Host VI**. Open the VI. Notice that the front panel has a pre configured set of controls and indicators, and that the block diagram has only the basic structure for the program we will implement.

9. Right click the block diagram and add an Open FPGA VI Reference ( **Functions » FPGA Interface » Open FPGA VI Reference**). Configure the reference to use the **FPGA Main VI**. Right click the reference and select the **Configure Open FPGA VI Reference . . .** option. Select the **VI** option and browse to your FPGA Main VI and click OK. The FPGA VI Reference is now associated with your FPGA Main VI. Move the Open FPGA VI Reference outside of the while loop.

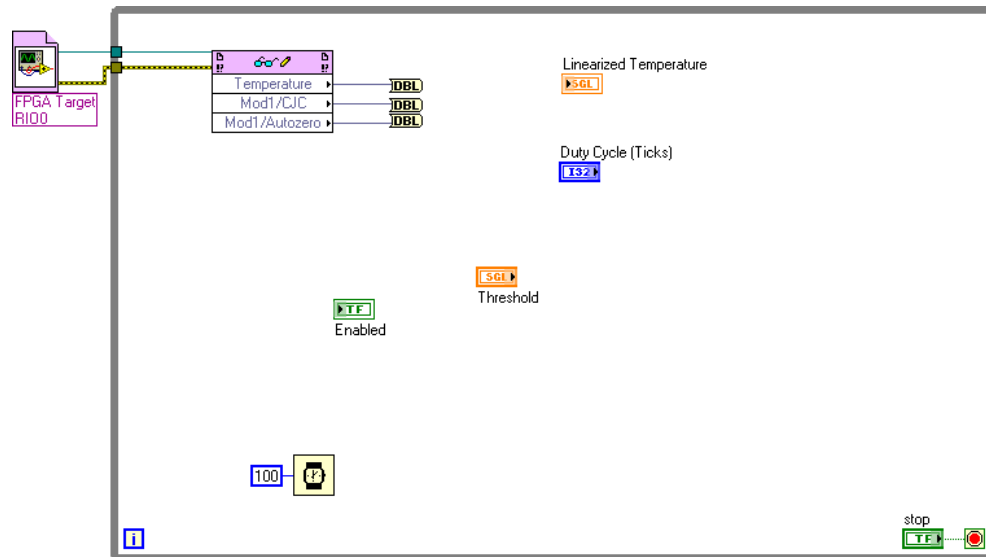


10. Next, right click the block diagram and add a **Read/Write Control** node within the while loop. It can be found in the **FPGA Interface** Pallet. Connect the FPGA VI Reference output and the Error output to the input of the Read/Write Control as shown below.

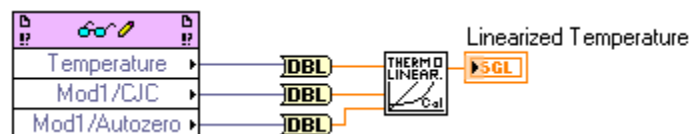


11. Bring up the drop down menu for the **Read/Write Control** by left clicking on **Unselected** and then select the **Temperature** item. Expand the node to have three inputs (Temperature, Mod1/CJC, Mod1/Autozero) and select each item individually.

12. Next, add three **To Double Precision Float** functions by right clicking the block diagram and navigating to the **Programming » Numeric » Conversion** Palette. Connect the outputs of the Read/Write Control to the To Double Precision Float functions as shown in the image below.



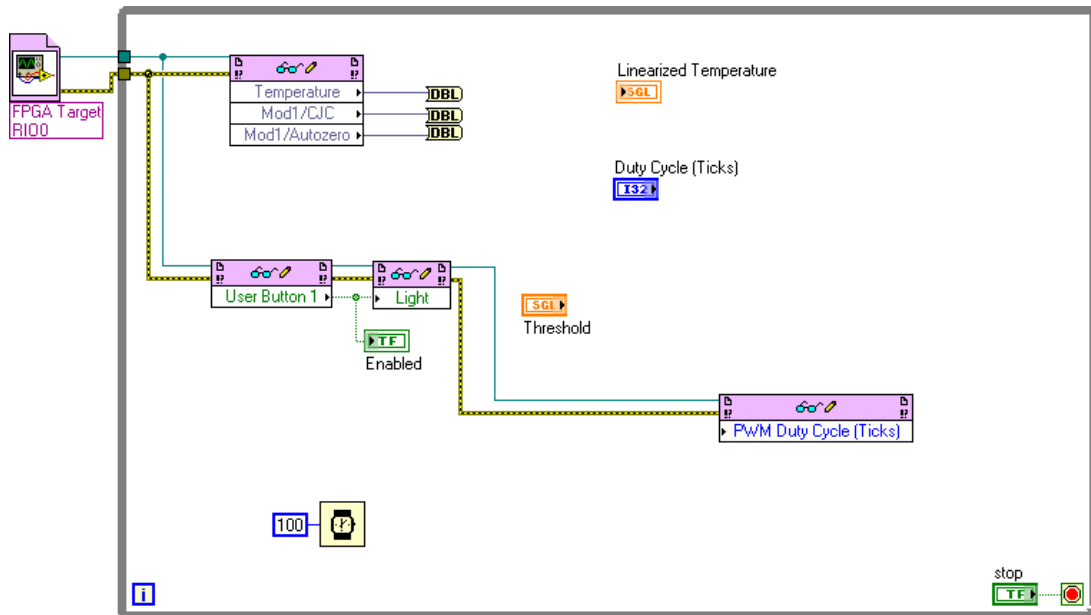
13. Next we need to convert the Temperature information from the nominal value provided by the module through the FPGA, to a cold junction compensated value. A VI for performing this action can be found in *C:\NITS\cRIO 2 FPGA\LabVIEW FPGA and CompactRIO Hands On\ Support VIs*. Right click the block diagram to open the functions palette and click on **Select a VI . . .** Browse to *Thermocouple VIs* and select and place the **NI 9211 Convert to Temperature (Calibrated).vi** to the right of the To Double Precision Float functions. Wire the VI as show below.



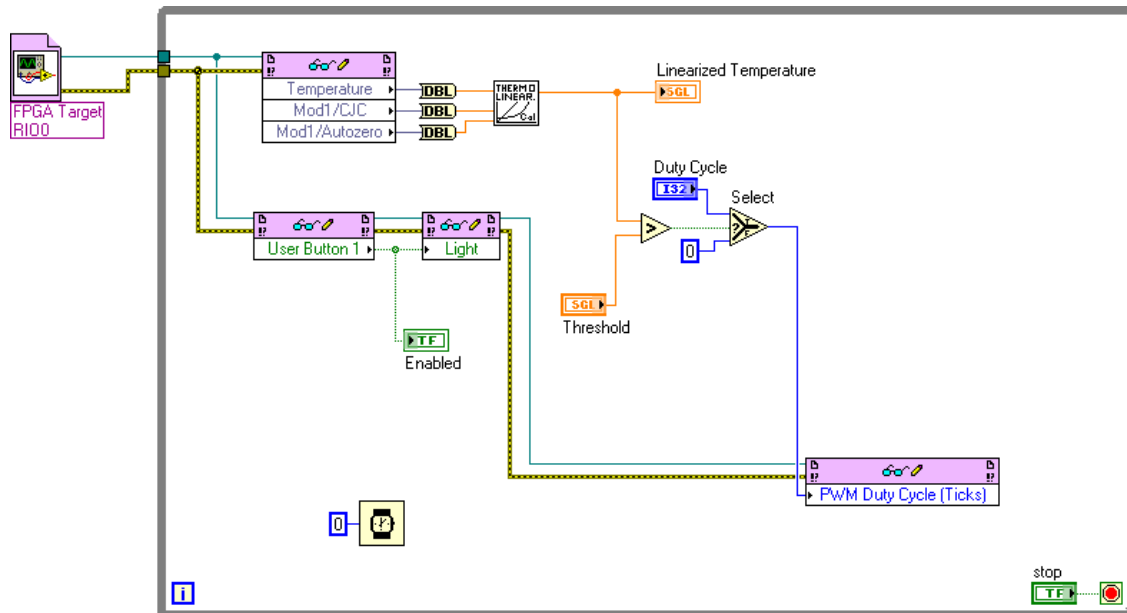
14. Now add three more **Read/Write Control** nodes to the block diagram from the **FPGA Interface Palette**. Associate these with the items below by connecting the FPGA VI Reference wire to the FPGA VI Reference In input for each node then selecting the desired I/O from the drop down list below the node. Items to select are:

- a. User Button 1
- b. Light
- c. PWM Duty Cycle (Ticks)

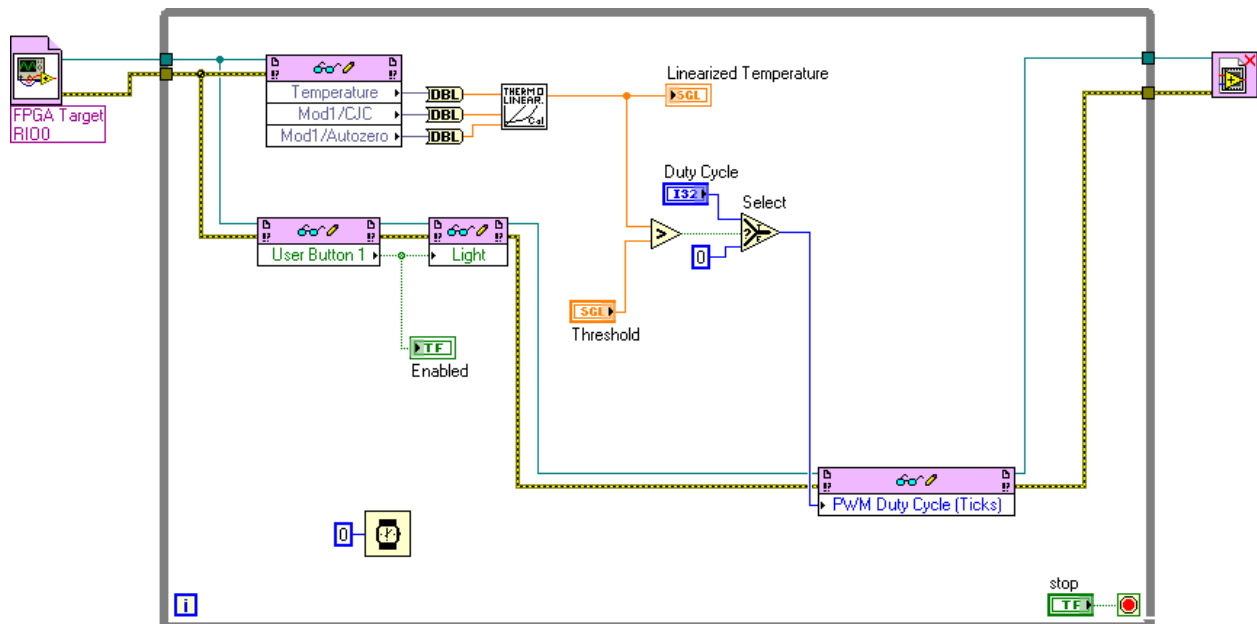
Your VI should look similar to the image below.



15. Next we will add the logic to control the temperature by turning the fan off and on. Navigate to the **Programming » Comparison Palette** and add a **Greater?** function and a **Select** function and wire the diagram as shown in the image below.

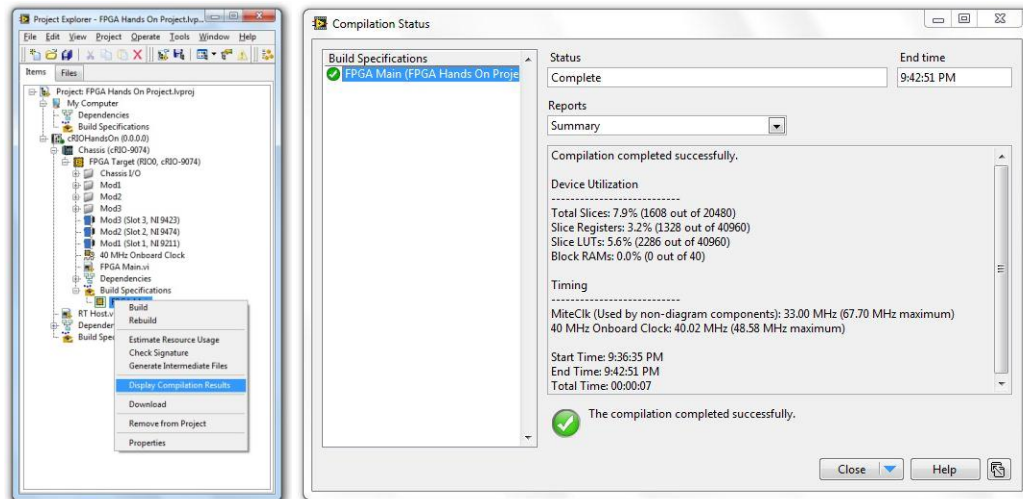


16. Finally we need to close the FPGA reference. Navigate to the **FPGA Interface palette** and place a **Close FPGA VI Reference** to the right of the while loop and connect it as shown in the image below. **Save the VI.**





17. Now we will reconnect to the compilation of our FPGA code and run our program. To reconnect, expand the Build Specifications in LabVIEW Project Tree under the RIO Target. Right click on FPGA Main and click **Display Compilation Results**. This will reconnect to the compile server. Right click on the FPGA Main Build Specification in the left hand pane and choose **Reconnect to Compilation**. You should now see the compile report window showing a successfully completed compilation. Click **Close** on the compile report to finish writing the generated bit file to disk.



18. Run the RT Host.vi and test out the embedded control system you've just created.