

Summary

Tutorial
TU0116 (v2.0) May 17, 2008

This tutorial introduces you to the basics of FPGA design using Altium's Innovation Station. It covers FPGA project creation within Altium Designer, targeting of a design to a physical FPGA device on a daughter board plugged into the Desktop NanoBoard, and processing of the design – ultimately programming the FPGA. Use of design hierarchy and virtual instrumentation is also briefly explored.

The Altium Innovation Station – the powerful combination of Altium Designer software and Desktop NanoBoard reconfigurable hardware platform – provides all of the tools and technology needed to capture, implement, test and debug your FPGA designs, in real-time.

With Altium's Innovation Station the low level detail is managed for you, leaving you free to focus on device intelligence and functionality – the source of true and sustainable product differentiation. Before embarking into the world of processors and embedded software intelligence however, it is beneficial to receive a solid grasp of the fundamentals of designing within this innovative environment – how to implement the most basic of designs and get it running on a physical FPGA device plugged into the Desktop NanoBoard.

This tutorial implements a simple counter-based design (non-processor) which, when programmed into the target daughter board FPGA, will cause the User LEDs on the Desktop NanoBoard to light sequentially from left to right, or from right to left. During the course of this tutorial, you will gain knowledge of the basics of FPGA design, introducing you to:

- FPGA project creation within Altium Designer and how to implement a schematic-based design, including sourcing and placing parts and wiring them on a schematic sheet.
- Targeting of a design to a daughter board FPGA using the auto-configuration feature.
- Processing of a design – compiling, synthesizing and building the design to obtain the programming file which is used to program the target device.
- Use of design hierarchy within an FPGA project, including simple custom logic (HDL).
- Virtual instrumentation.

The example design featured in this tutorial is a simple twisted-ring counter (Figure 1). This is a synchronous counter where the inverted output of the last stage is connected to the input of the first stage. Rather than individual flip-flops, we will use a shift register component, readily supplied with Altium Designer. The base design schematic and additional files can be found in the \Examples\Tutorials\Getting Started with FPGA Design folder of your Altium Designer installation. Refer to this example at any time to get further insight or to skip some of the steps.

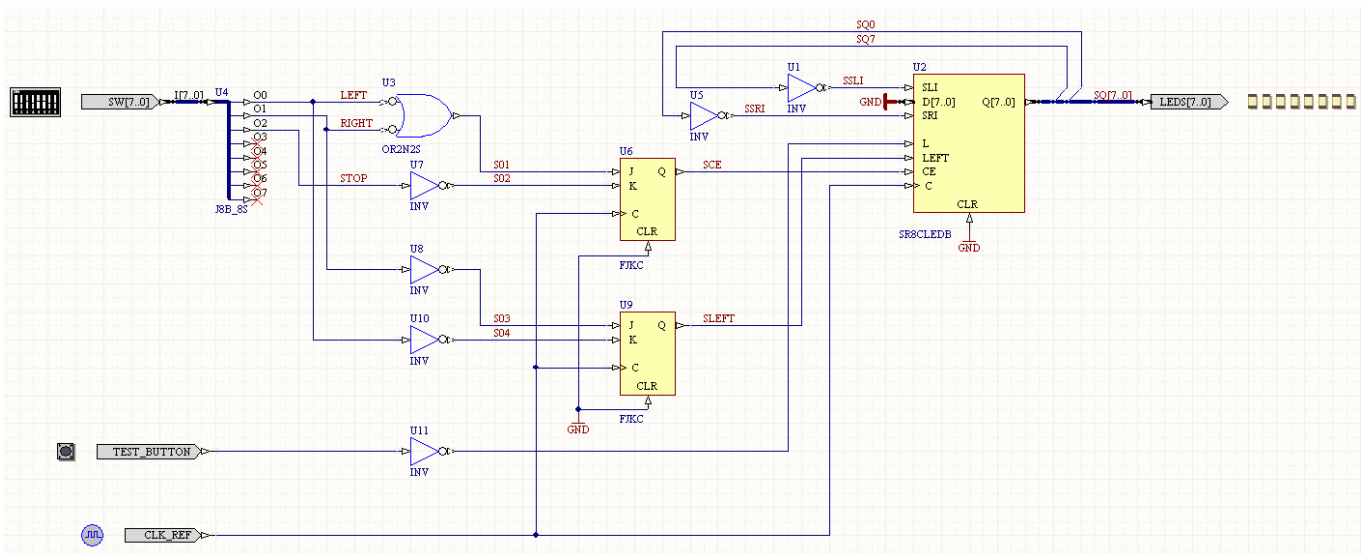



Figure 1. Simple, non-processor-based design – a twisted-ring counter.

Getting Started with FPGA Design

The synchronizing clock signal for the logic components in our circuit is provided courtesy of the reference clock on the Desktop NanoBoard. The counter output is displayed on the NanoBoard's User LEDs.

Additional logic in the design, coupled with use of resources found on the NanoBoard, allow for the following controls:

- **Direction control** – the count will proceed from left to right or from right to left, dependent on the setting of an associated switch on the NanoBoard (part of the DIP-switch).
- **Stop control** – the count can be stopped or resumed, dependent on the setting of an associated switch on the NanoBoard (part of the DIP-switch).
- **Clear control** – the counter output can be cleared (all LEDs turned OFF), by pressing the 'DAUGHTER BD TEST/RESET' button on the NanoBoard.

 For detailed information on the Desktop NanoBoard, refer to the document [TR0143 Technical Reference Manual for Altium's Desktop NanoBoard NB2DSK01](#).

 For information on the range of supported daughter boards available for the Desktop NanoBoard, and additional documentation specific to each, go to www.altium.com/nanoboard/resources.


Important Note on FPGA Vendor Tools

Before you can download a design to the Desktop NanoBoard – or rather the physical device resident on a daughter board plug-in – you must have the appropriate vendor tools installed on your computer. These tools are used to place and route the FPGA design for the target device. The FPGA vendor tools ARE NOT supplied with the system and must be sourced independently.

A range of daughter boards are available for use with the Desktop NanoBoard. The FPGA devices on these daughter boards are supported by the respective vendor's downloadable tools available on the web, as well as by the commercial versions of these tools. In order to use your chosen daughter board, you will need to install the relevant tools.

More information on the vendor tools can be found on the respective FPGA vendor's web site:

- Actel[®] Designer or Libero[®] IDE from www.actel.com. The software can be downloaded but does require a license. Check the website for licensing options.
- Altera[®] Quartus[®] II from www.altera.com. The Altera Quartus II Web Edition Software can be freely downloaded and does not require a license.
- Lattice[®] ispLever[®] from www.latticesemi.com. The ispLever Starter software can be downloaded but does require a license. Check the website for licensing options.
- Xilinx[®] ISE[™] from www.xilinx.com. The Xilinx ISE WebPACK can be freely downloaded and does not require a license.

 Links to each vendor's downloadable tools can be found in the Vendor Resources area of the Altium website (www.altium.com/Community/VendorResources). This page can be accessed directly from within Altium Designer. With the **Devices** view active (**View » Devices View**), simply choose the **Vendor Tool Support** entry on the main **Tools** menu.

Note: Altium does not provide technical support for FPGA vendor tools. For information on installing these tools, please refer to the information provided by the FPGA vendors.

Capturing the Design

The first thing we need to do is capture our design within the Altium Designer environment. For our simple design circuit, this will involve adding required components to a schematic sheet and wiring them accordingly. Before we can address the schematic and its contents however, we must create a project. The following sections take you through the steps required to capture our twisted-ring counter design.

Creating the FPGA Project

The basis of every design created in the Altium Designer environment is a project file. For an FPGA design, we need to create a new FPGA project (*.PrjFpg). The project document itself is an ASCII file that stores project information, such as the documents that belong to the project, output settings, compilation settings, error checking settings, and so on.

Let's go ahead and create the FPGA project.

1. Create a new FPGA project using the **File » New » Project » FPGA Project** command.
2. Right-click on the name for this new project (FPGA_Project1.PrjFpg) in the **Projects** panel and choose the **Save Project** command. Save the project in the location of your choice, with the name Simple_Counter.PrjFpg, and in a new folder called Basic FPGA Design Tutorial.

Note: Spaces and/or dashes (-) must not be used in project or document filenames. Doing so will result in synthesis errors during design processing. Underscores (_) should be used instead, to provide readability.

Adding a Schematic Source Document

An FPGA project is hierarchical in nature. Any number of schematic, HDL (VHDL or Verilog) or OpenBus documents can be included, with all descendent sub-files referenced using sheet symbols. One common denominator for all projects however is that they must have a single top-level schematic. This sheet not only contains the ports for the design – which interface directly to pins of the physical FPGA device to which the design is targeted – but also facilitates FPGA-PCB integration within Altium Designer.

We shall explore design hierarchy later in this tutorial. For now, we will simply add a single schematic sheet (our top sheet) to our new FPGA project:

1. Add a new schematic document by right-clicking on the FPGA project entry in the **Projects** panel and choosing the **Add New to Project » Schematic** command. A blank schematic sheet will open as the active document in the main design window.
2. Save this document (**File » Save**) with the name Simple_Counter.SchDoc, in the same folder as the parent project.
3. The project itself will appear as modified in the **Projects** panel. Save the project also (right-click on its name and choose **Save Project**)

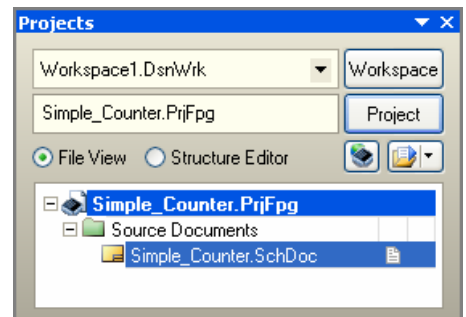


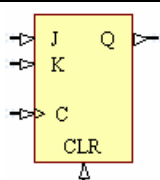
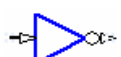
Figure 2. New FPGA project with added source schematic.

Placing Components

Now we have our 'blank schematic canvas' it's time to add-in the required components – those that depict the functionality of the design circuitry, and those that provide the interface with resources found on the Desktop NanoBoard NB2DSK01.

Table 1 identifies the components required in defining the twisted-ring counter circuit. All of these components can be found in the FPGA Generic integrated library (FPGA_Generic.IntLib), located in the \Library\Fpga folder of the installation.

Table 1. Design components required in the twisted-ring counter schematic.

Symbol	Component Name	Description	Quantity Required
	FJKC	J-K Flip-Flop with Asynchronous Clear	2
	INV	Inverter	6

Getting Started with FPGA Design

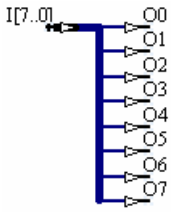

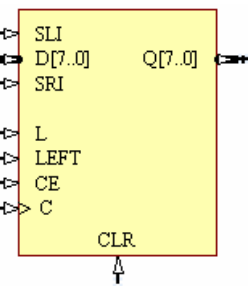




	J8B_8S	Bus Joiner – 8-pin input bus to 8 single pin outputs	1
	OR2N2S	2-input OR gate with active Low A and B inputs, (single pin version)	1
	SR8CLED	8-bit loadable serial/parallel-in parallel-out bidirectional shift register with clock enable and asynchronous clear (bus version)	1

Table 2 identifies the design interface components required for our design. These components, commonly referred to as port components (or port-plugs), automatically establish connectivity between the relevant resources on the Desktop NanoBoard and the physical IO pins of the daughter board FPGA device to which our design is targeted. From the design perspective, they depict the extents of the design – the physical pins of the target device to which the signals from the design connect. These port components can be found in the FPGA NB2DSK01 Port-Plugin integrated library (FPGA_NB2DSK01_Port-Plugin.IntLib), also located in the \Library\Fpga folder of the installation.

Table 2. Port components required in the twisted-ring counter schematic.

Symbol	Component Name	Description
	CLOCK_REFERENCE	This component interfaces to the fixed 20MHz system clock signal on the Desktop NanoBoard. We will use this signal to provide the synchronous clock signal to our flip-flop and shift register logic.
	DIPSWITCH	This component interfaces to the DIP-switch on the Desktop NanoBoard. We will use three of these switches to control direction of the counter (Left or Right) and also to stop the counter.
	LED	This component interfaces to the user LEDs on the Desktop NanoBoard. We will use the LEDs to visually display the output of our counter.
	TEST_BUTTON	This component interfaces to the 'DAUGHTER BD TEST/RESET' button (push button switch SW7) on the Desktop NanoBoard. We will use this signal (inverted) as a control input to the Load Enable pin of the shift register. As we will tie the D inputs of this register to GND, pressing this button will load '0' into the register's data outputs.

Go ahead and place all of these components on the schematic sheet, as shown in Figure 3. Both of the integrated libraries are installed and available from the **Libraries** panel by default, so there is no need to add any libraries for this tutorial. Simply make the relevant library active in the panel, select the component entry in the list and either click the **Place** button at the top-right of the panel, or click and drag the component directly onto the sheet. Schematic placement controls, such as flipping and rotating, allow for fine tuning as needed.

Once placed, finalize the designation of each component using the **Tools » Annotate Schematics Quietly** command.

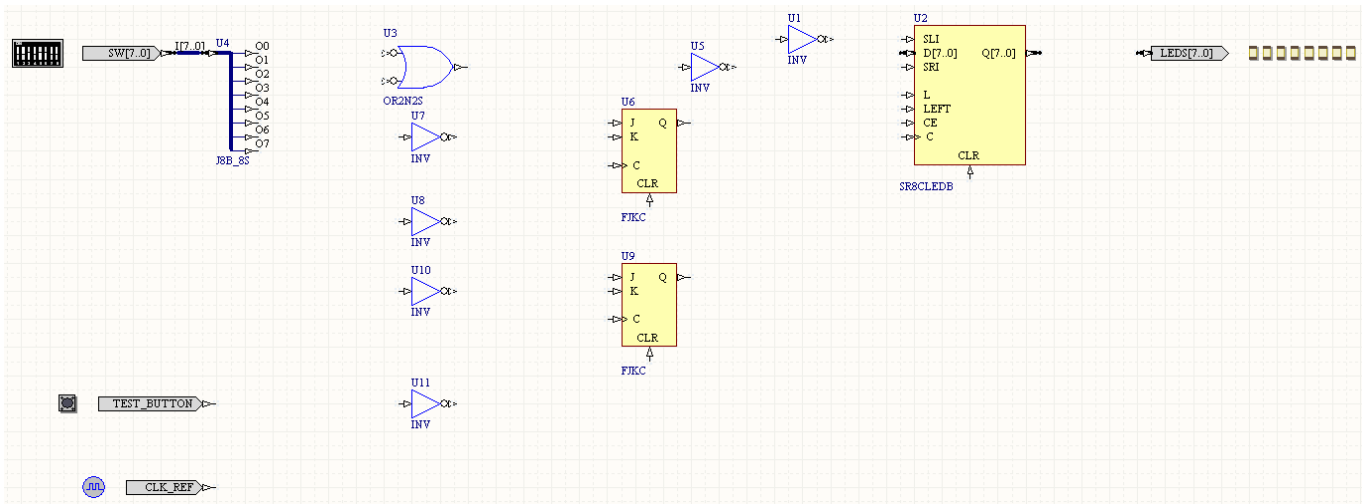


Figure 3. Initial placement of components on the schematic sheet, Simple_Counter.SchDoc.

Wiring the Design

Now all components have been placed it is time to wire them all together – adding the connectivity to the design.

1. Go ahead and wire the design initially, as shown in Figure 4, by using the **Place » Wire** and **Place » Bus** commands (these commands are also available from the **Wiring** toolbar, using the and buttons respectively).

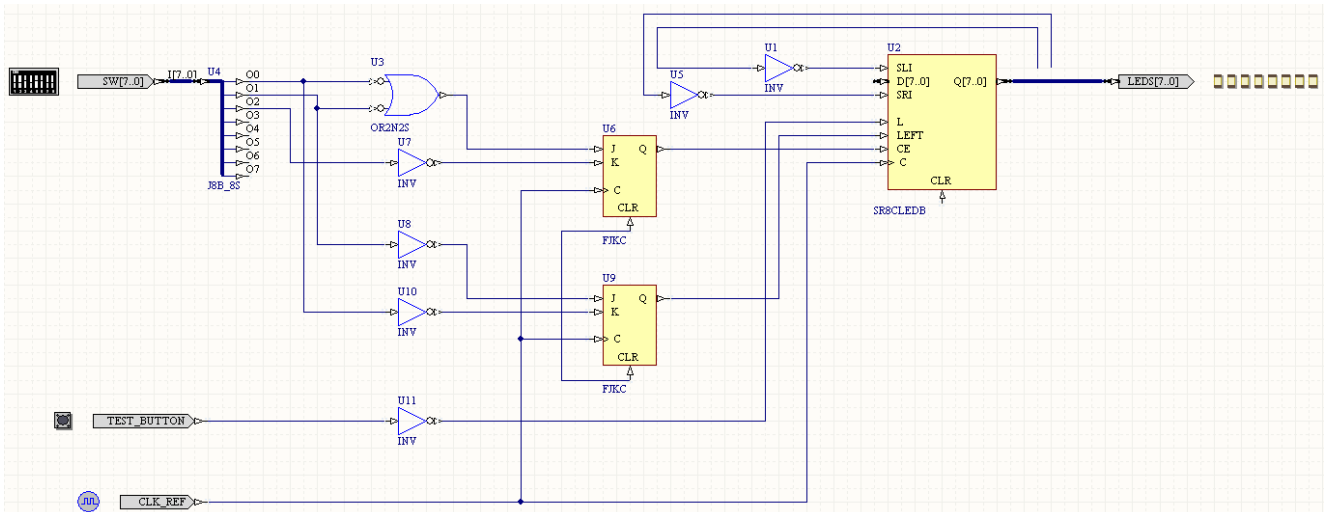


Figure 4. Initial wiring – placement of wires and buses.

2. From the **Wiring** toolbar, click the button to place a GND Power Port. While the port is still floating on the cursor, press the **Tab** key. From the **Power Port** dialog that appears, change the **Style** property to **Bar**. Now place the port so that it connects to the bottom-left of the wire connecting the CLR pins of the two flip-flops.

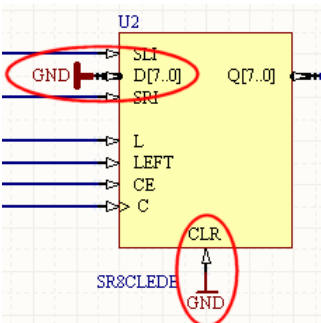


Figure 6. Tying shift register signals to GND.

3. Now place another GND Power Port – again using the **Bar** style – so that it connects to the CLR pin of the shift register.
4. From the **Wiring** toolbar, click the button to place a GND Bus Power Port. Again, change the port's **Style** property to **Bar**. Now place the port so that it connects to the D[7..0] pin of the shift register. Press **Spacebar** to rotate the port as required.

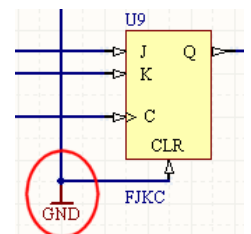


Figure 5. Tying flip-flop CLR signals to GND.

Getting Started with FPGA Design

- We now need to add bus entries for 'tapping off' individual signals from the output bus of the shift register. These entries need to connect to the individual wires that feed, via respective inverters, into the SLI and SRI inputs of the shift register. From the **Wiring**

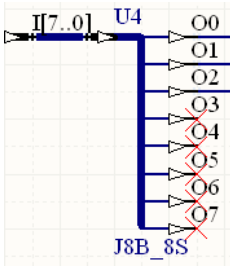


Figure 8. Mopping up loose ends with No ERC markers.

- To tidy up the design and also minimize compilation warnings, we can also place No ERC markers on each of the unused outputs of the Bus Joiner component (U4). From the **Wiring** toolbar, click the button and place the two bus entries as shown in Figure 7.

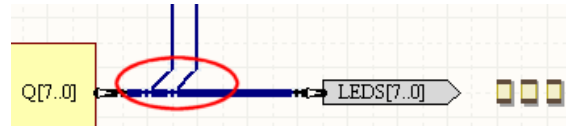
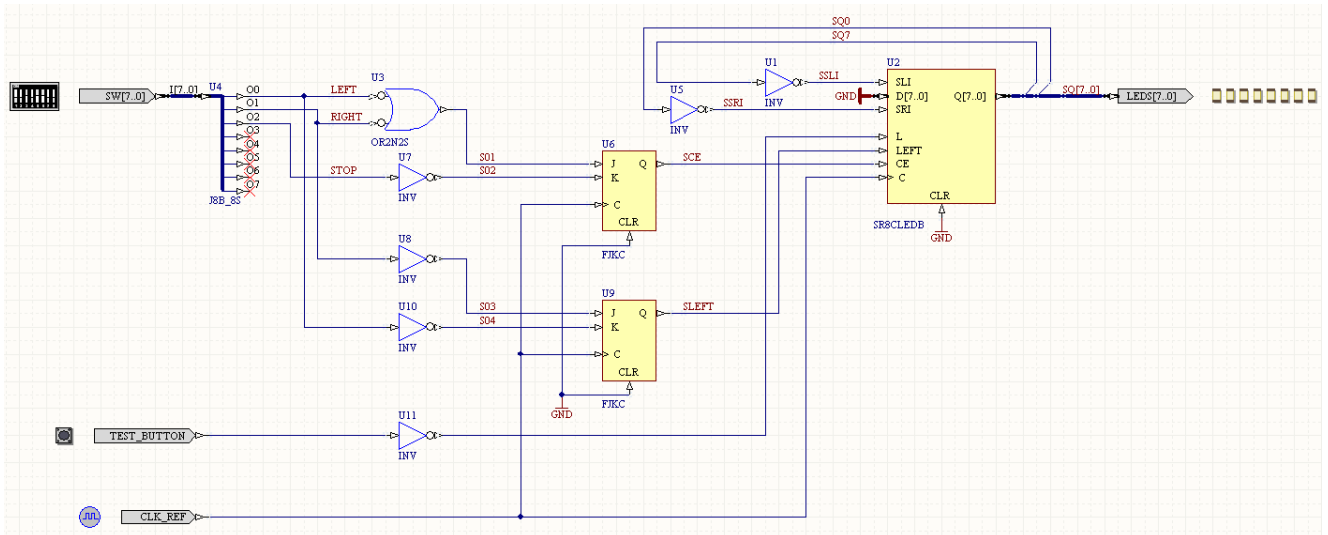


Figure 7. Adding bus entries.

- To tidy up the design and also minimize compilation warnings, we can also place No ERC markers on each of the unused outputs of the Bus Joiner component (U4). From the **Wiring** toolbar, click the button to enter No ERC marker placement mode. Then position and place a marker on each of the unused pins, O3 to O7, as illustrated in Figure 8.

To complete the wiring of our design, we will add labels to key nets in the circuit. This will make the design both easier to understand and also enable any problems to be tracked down with greater efficiency when checking the design at the compilation stage.

- Go ahead and add net labels to the design, as illustrated in Figure 9, using the **Place » Net Label** command (or clicking the button on the **Wiring** toolbar). The net labels shown in Figure 9 are an example. You could use different names for the net labels, as long as they are unique.



Verifying the Design

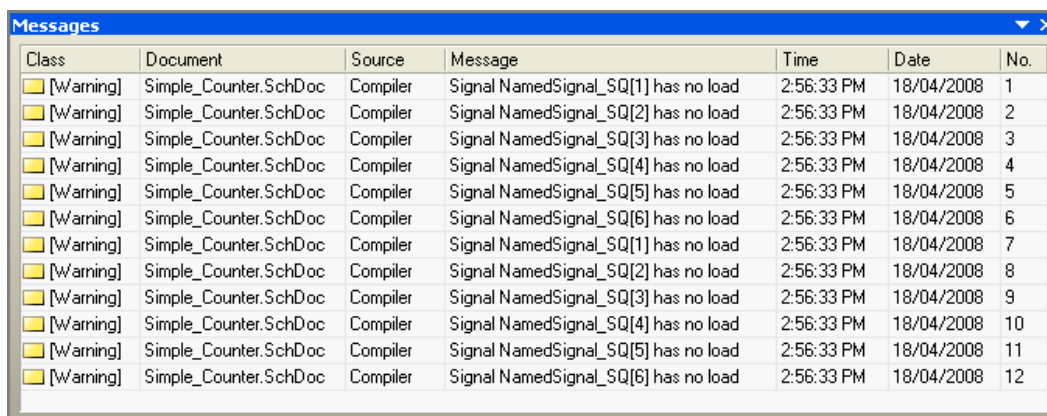
Before we look at targeting and downloading our design to the intended physical FPGA device, it is a good time to verify the integrity of the design. To do this, we must invoke Altium Designer's powerful Design Compiler. The process of compiling is integral to producing a valid netlist for a project. The Compiler checks a range of electrical and drafting errors, in accordance with options defined on the **Error Reporting** and **Connection Matrix** tabs of the *Options for FPGA Project* dialog (**Project » Project Options**).

Note: There are no modifications to be made on either of these tabs for the purposes of this tutorial. The default settings are fine.

1. From the main schematic menus, choose **Project » Compile FPGA Project Simple_Counter.PrjFpg**. Compilation of the project will proceed.
2. Any Warnings, Errors and Fatal Errors will be listed as entries in the **Messages** panel. If there are any Errors or Fatal Errors found during compilation, this panel will automatically appear. If there are Warnings only, you will need to display the panel manually – simply click on the **System** button below the main design window and choose **Messages** from the menu that appears.

Double-clicking on a message entry will display further information about the error in the **Compile Errors** panel. The offending entity will be zoomed into and highlighted on the schematic sheet.

3. With the schematic (*Simple_Counter.SchDoc*) wired correctly, you should only see a number of Warning messages relating to signals having no load (Figure 11). These occur because we have tapped off SQ0 and SQ7 from the bus SQ[7..0] but not SQ1 to SQ6. We can safely ignore these warnings for our design.



Class	Document	Source	Message	Time	Date	No.
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[1] has no load	2:56:33 PM	18/04/2008	1
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[2] has no load	2:56:33 PM	18/04/2008	2
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[3] has no load	2:56:33 PM	18/04/2008	3
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[4] has no load	2:56:33 PM	18/04/2008	4
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[5] has no load	2:56:33 PM	18/04/2008	5
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[6] has no load	2:56:33 PM	18/04/2008	6
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[1] has no load	2:56:33 PM	18/04/2008	7
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[2] has no load	2:56:33 PM	18/04/2008	8
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[3] has no load	2:56:33 PM	18/04/2008	9
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[4] has no load	2:56:33 PM	18/04/2008	10
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[5] has no load	2:56:33 PM	18/04/2008	11
[Warning]	Simple_Counter.SchDoc	Compiler	Signal NamedSignal_SQ[6] has no load	2:56:33 PM	18/04/2008	12

Figure 11. With the design wired correctly, these are the only messages that should appear after compilation.

If you have any different error messages, you will need to resolve them and recompile the design project.

For detailed information on the various compiler errors, how they can occur and possible methods by which to resolve them, refer to the document [TR0142 Project Compiler Error Reference](#).

4. Save the schematic and its parent project.

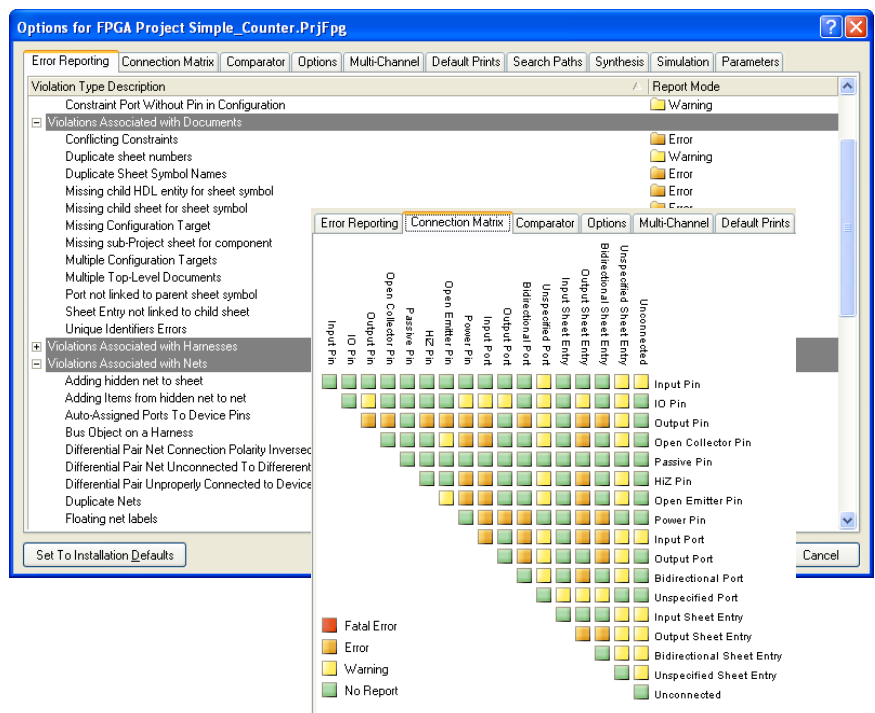


Figure 10. Compiler settings are defined on the Error Reporting and Connection Matrix tabs of the Options for FPGA Project dialog.

Targeting the Physical FPGA Device

Now we are finished with the capture phase of our design, we need to specify which physical FPGA device we want to use – the target for our design and the end medium into which the design will ultimately be programmed and run. For this tutorial, we will target an FPGA device located on a 3-connector daughter board that is plugged into the Desktop NanoBoard NB2DSK01.

The process of mapping or constraining a design to its physical implementation is done by creating constraint files – files that specify implementation detail such as the target device, the port-to-pin mapping, pin IO standards, and so on. The minimum information required to synthesize a design is the device specification.

Sets of constraint files are targeted to a design by creating a configuration, which is simply a named list of constraint files.

The constraint system in place for the Desktop NanoBoard NB2DSK01 utilizes various constraint files covering:


- Resources and pin-mapping local to the NB2DSK01 motherboard and satellite peripheral and daughter boards
- Connection of a satellite board (peripheral boards and daughter boards) to the NB2DSK01 motherboard.


Although an FPGA design project targeting the Desktop NanoBoard can be configured manually – by adding a configuration, assigning the required board constraint files and creating a mapping constraint file by hand – the process is greatly simplified through use of an auto-configuration feature.

Using this feature, a target configuration for the FPGA design project is automatically created. The required board-level constraint files are then automatically determined and added to this configuration, based on the hardware (motherboard, daughter board and peripheral boards) detected in your system. An additional mapping constraint file is also generated and added to the configuration, which handles connection of the satellite boards detected in the system (daughter board and any peripheral boards), to the main motherboard.

The key to being able to configure an FPGA design project automatically, is the ability of Altium Designer to identify the specific hardware you are currently using in your system. Identification is made possible through the use of 1-Wire-based memory devices, located on the NB2DSK01 motherboard, all 3-connector daughter boards and peripheral boards.

Note: Although older, 2-connector Altium daughter boards can be used with the Desktop NanoBoard NB2DSK01, they do not possess the memory device required for auto-configuration.

 For more information on the concept of configurations and constraints, and their role in design portability, refer to the article [AR0124 Design Portability, Configurations and Constraints](#).

 For more detailed information on the Desktop NanoBoard NB2DSK01 constraint system, including auto-configuration, refer to the application note [AP0154 Understanding the Desktop NanoBoard NB2DSK01 Constraint System](#).

Let's go ahead and configure our FPGA project.

1. Prior to using the auto-configuration feature, ensure the following:
 - The 3-connector daughter board carrying the FPGA device to which the design will be targeted is plugged into the NB2DSK01 motherboard.
 - Our simple counter design does not use any resources resident on plug-in peripheral boards. These boards can be left attached to the motherboard, or removed, as required.
 - The NB2DSK01 is connected to the PC via USB (or parallel) connection and is powered-on.
2. Open the **Devices** view (**View » Devices View**). Enable the **Live** option and ensure that the **Connected** indicator is Green.
3. The auto-configuration feature can be accessed and run in two ways. For the purposes of this introductory tutorial, it is worth taking a look at both of these methods. For this step, pick only one of the following methods to run the auto-configuration process:

Method 1: Right-click on the icon for the Desktop NanoBoard – in the NanoBoard chain of the view – and from the menu that appears choose **Configure Fpga Project » Simple_Counter.PrjFpg** (Figure 12).

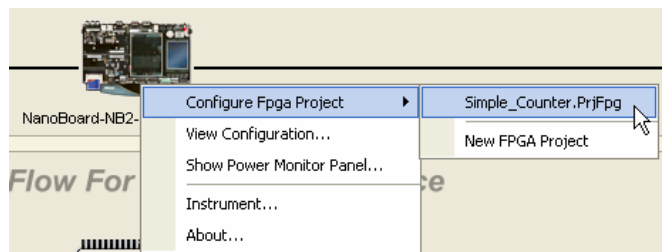


Figure 12. Auto-configure direct from Devices view.

Method 2: Double-click on the icon for the Desktop NanoBoard to access its corresponding instrumentation in the Instrument Rack – NanoBoard Controllers panel. Then click on the Board View button to access the *NanoBoard Configuration* dialog. Use the Auto Configure FPGA Project drop-down at the bottom-left of the dialog and choose *Simple_Counter.PrjFpg* (Figure 13). By using the dialog, you are presented with a visual (and dynamic) summary of your current Desktop NanoBoard NB2DSK01 system. The image in the dialog displays the specific peripheral board(s) and daughter board that are physically plugged in to the NB2DSK01 motherboard.

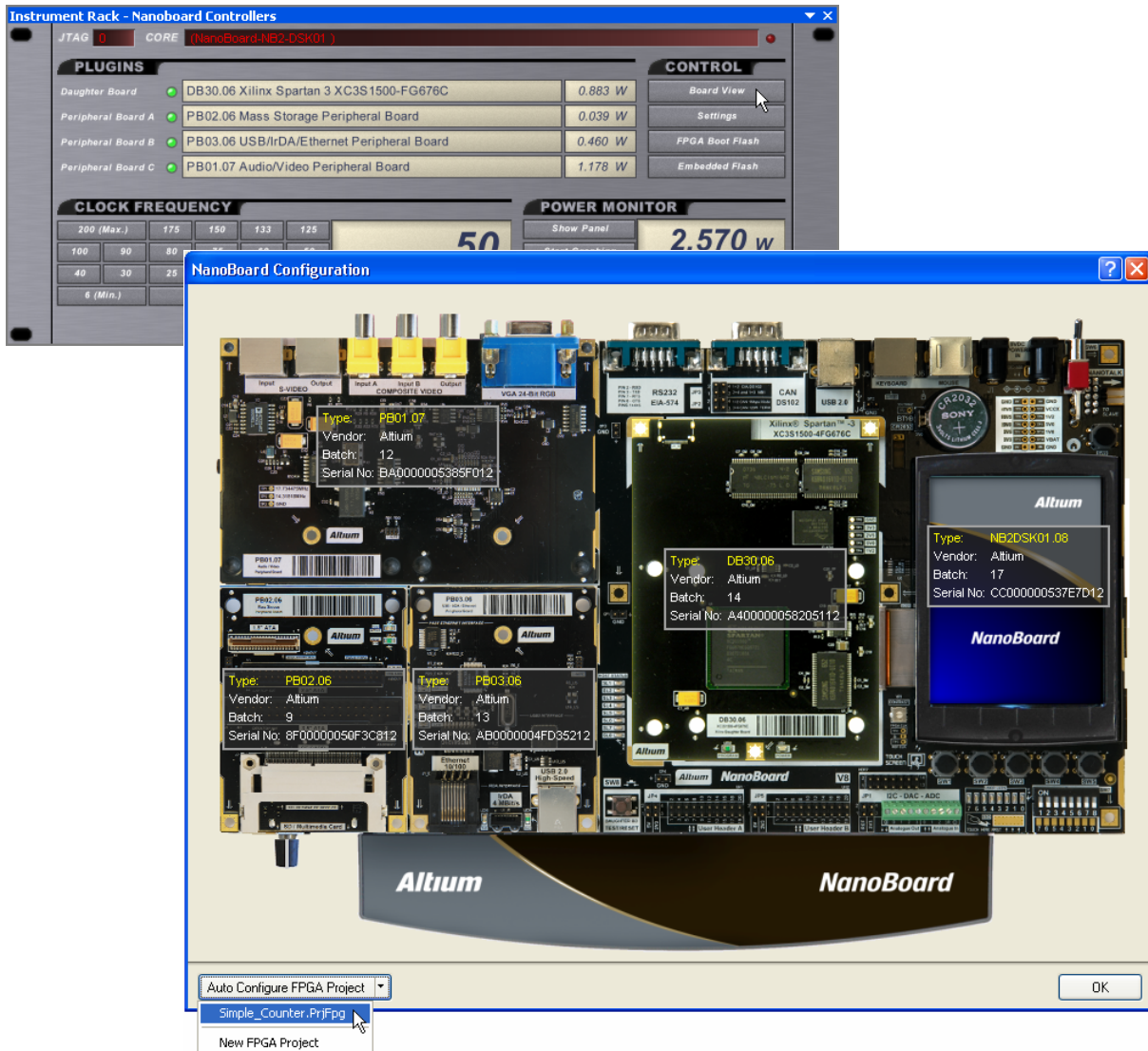


Figure 13. Auto-configure from the visually-based NanoBoard Configuration dialog.

The auto-configuration process will proceed, which involves the following:

- A configuration is created and added to the Simple_Counter project. The name for the configuration will depend on the version of your Desktop NanoBoard and the particular daughter board being used. The generic format for the name is: `motherboard code_revision_daughter board code_revision`. For example, with a Desktop NanoBoard NB2DSK01 (revision 8), and a Xilinx Spartan-3 daughter board DB30 (revision 6), the configuration will be named `NB2DSK01_08_DB30_06`.
 - Constraint files will then be added to the configuration for each of the detected boards in the system (motherboard, daughter board and peripheral board(s)). These are sourced from the `\Library\Fpga\NB2 Constraint Files` folder of the installation. In each case, the file used will be determined by the type and version of board. For example, if you are using a Xilinx Spartan-3 daughter board DB30 (revision 6), then the constraint file retrieved and added to the configuration will be `DB30.06.Constraint`.
 - The constraint file that defines the mapping of daughter board and peripheral board(s) to the motherboard is also created, on-the-fly, and added to the configuration. The name of this file will simply be that of the configuration itself, with the additional suffix `'_BoardMapping'` (e.g. `NB2DSK01_08_DB30_06_BoardMapping.Constraint`). The file will be saved to the same location as the project file (`Simple_Counter.PrjFpg`) itself.
4. The configuration and assigned constraint files will appear listed in the *Configuration Manager* dialog, as shown in Figure 14.

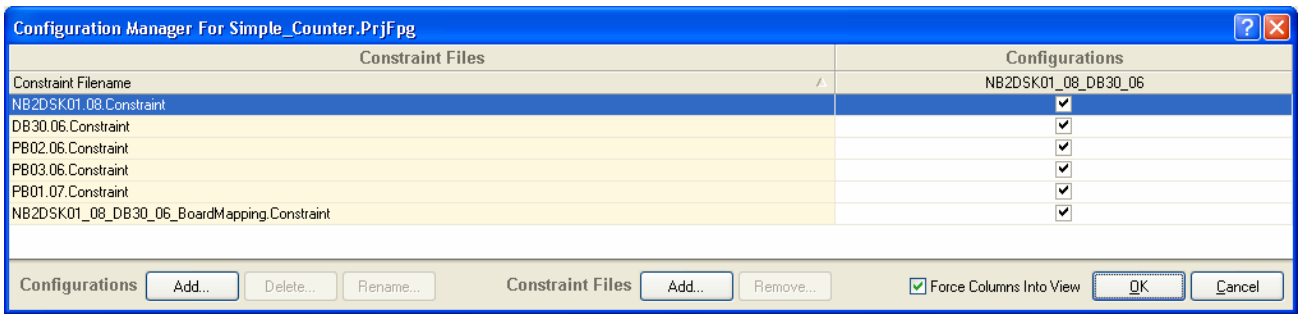


Figure 14. Resulting configuration and constituent constraint files for the Simple_Counter project.

Click **OK**. A sub-folder named `Settings` will be added to the project and appear listed in the **Projects** panel. The constraint files themselves will be listed in the sub-folder `Constraint Files`.

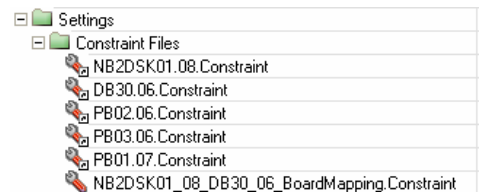


Figure 15. Constraint files added to configuration.

5. Save the project file.

We have now finished configuring the design. It is targeted to the daughter board FPGA device and is now ready to be processed and ultimately programmed into that device.

Processing the Design

Once the task of capturing an FPGA-based design is complete, the next logical step is to process the source files. This involves compilation and synthesis of the design, to obtain a source netlist file for input to relevant vendor place and route tools.

The process continues, running the place and route tools to ensure that the design will fit within a chosen physical device and to generate an FPGA programming file. This programming file can then be taken to the ultimate step in the processing chain – programming the physical FPGA device with the design.

This entire process flow – from captured source files to programmed physical device – is carried out from the **Devices** view.

1. Ensure that the **Devices** view is the active view within Altium Designer (**View » Devices View**).
2. Ensure that the **Live** option is still enabled and that the **Connected** indicator is still Green.

Within this view, it is the controls associated with the detected physical FPGA device, in the Hard Devices chain, that we are interested in for this part of the tutorial – collectively referred to as the 'Process Flow' for the physical device (Figure 15).

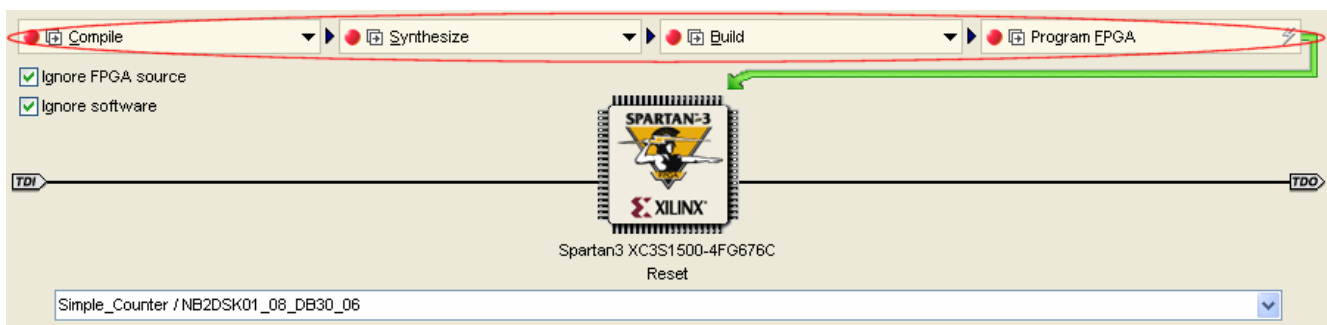



Figure 16. Associated Process Flow for the physical FPGA device.

This Process Flow will only be presented providing the following additional conditions are met:

- The relevant vendor tools, for the FPGA device fitted to the daughter board you are using, are installed.
- The drop-down field below the icon for the device in the chain shows a valid `Project / Configuration` pairing. A valid pairing exists when a project has an associated configuration that contains a constraint file targeting the design to the detected physical device.

Having run the auto-configuration feature, a valid configuration already exists for our example project, containing a constraint file that targets the daughter board device. The `Project / Configuration` entry therefore appears as `Simple_Counter / ConfigurationName` (e.g. `Simple_Counter / NB2DSK01_08_DB30_06` in Figure 15).

The Process Flow itself consists of four distinct stages, with the output of each stage required as input to the next. Although the entire Process Flow can be run by clicking directly on the **Program FPGA** button – the last stage in the flow – it is worth considering and using each stage in turn.

You can run all stages of the flow up to and including the current stage by clicking on the arrow icon located on the left side of the stage button .

- Click on the **Compile** button. This stage is used to perform a compile of the source documents in the associated FPGA project. If our project included processor cores, the associated embedded software projects would also be compiled during this stage. For our simple, non-processor design, this stage will simply use the Design Compiler to verify that the design is free from electrical and drafting errors. As we have previously compiled the project, we will be fine for this stage!

Feedback on the Compile process can be viewed in the **Messages** panel. Once any stage has completed successfully, its associated indicator will turn Green.

- Click on the **Synthesize** button. During synthesis, source documents are translated into intermediate VHDL files which are then synthesized into a top-level EDIF netlist, suitable for vendor Place & Route tools. For the particular FPGA device being targeted, synthesized model files associated with components in the design will be searched for and copied into the relevant synthesis output folder.

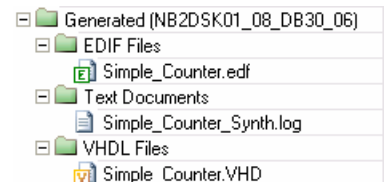


Figure 17. Generated files from the Synthesis stage of the flow.

If the synthesis is completed successfully, a folder called Generated [ConfigurationName] is created which holds the generated EDIF (Simple_Counter.edf), VHDL (Simple_Counter.vhd) and synthesis log (Simple_Counter_Synth.log) files.

Feedback on the Synthesis process can be viewed in the **Messages** panel.

- Click on the **Build** button. In this stage, Altium Designer calls and runs the vendor Place & Route tools, to process the design for the target physical FPGA device.

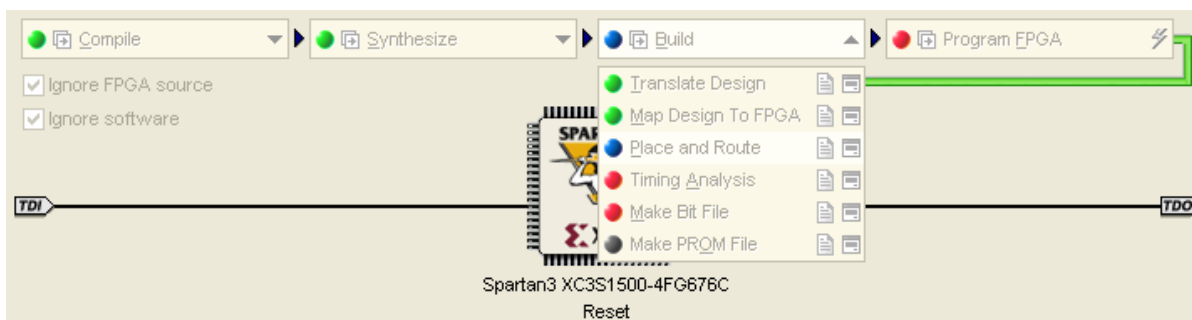


Figure 18. Reaching the Place and Route stage of the overall Build process.

Each of the following five sub-stages will run in turn:

- **Translate Design:** uses the top-level EDIF netlist and related synthesized model files, obtained from the Synthesis stage of the Process Flow, to create a file in Native Generic Database (NGD) format.
- **Map Design To FPGA:** maps the design to FPGA primitives.
- **Place and Route:** takes the low-level description of the design (from the mapping stage) and works out how to place the required logic inside the FPGA. Once arranged, the required interconnections are routed.
- **Timing Analysis:** performs a timing analysis of the design, in accordance with any timing constraints that have been defined. If there are no specified constraints – we don't have any for our simple design – default enumeration will be used.
- **Make Bit File:** generates the programming file that is required for downloading the design to the physical device.

Feedback on the build progress can be obtained by viewing the **Messages** panel. More detailed vendor-related feedback can be obtained from the **Output** panel (accessed from the menu associated to the **System** button at the bottom of the main design window).

Once the stage has completed successfully, the *Results Summary* dialog will appear. This dialog provides summary information with respect to resource usage within the target device, as well as any timing information. Close this dialog.

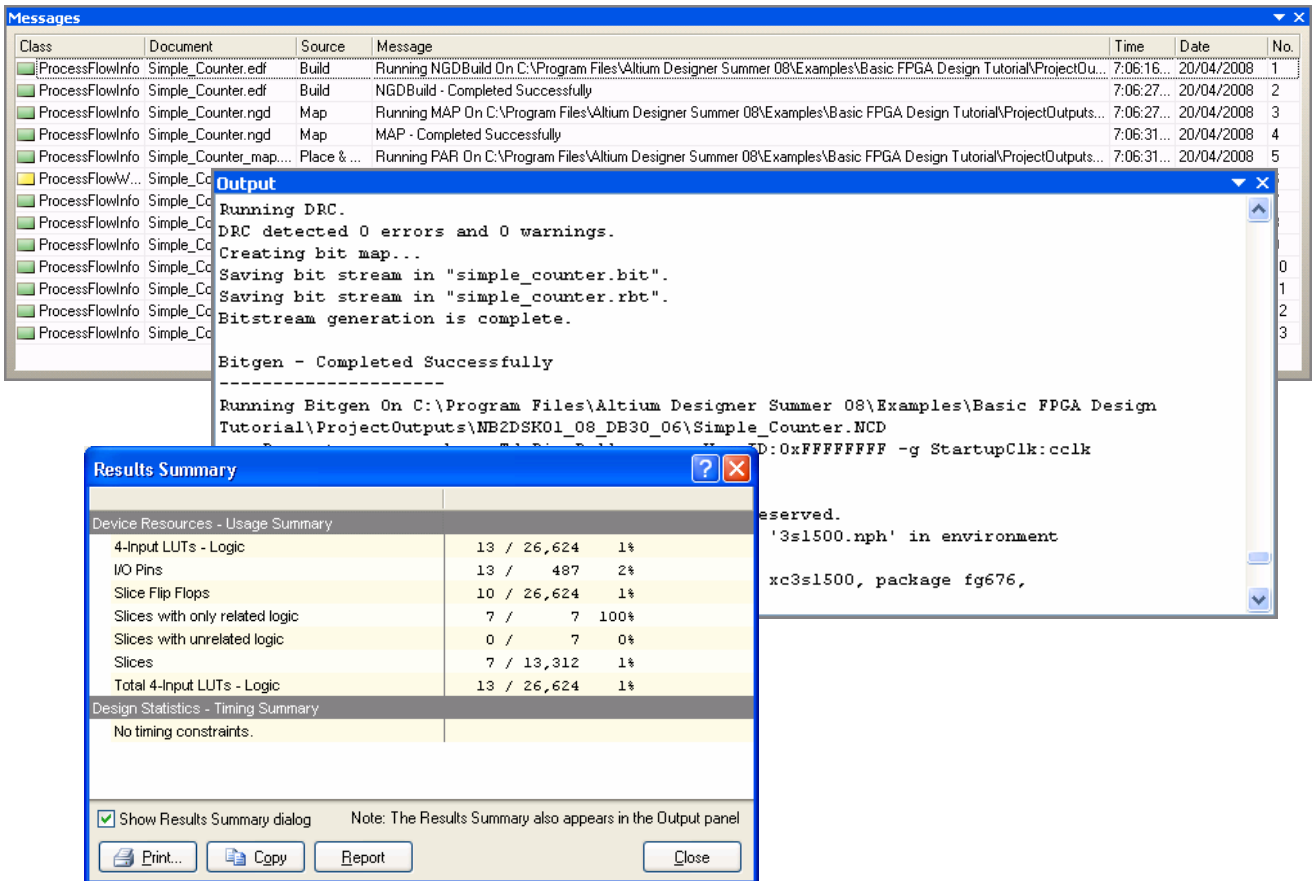


Figure 19. Accessing build process information and obtaining a final report of the build results.

- Click on the **Program FPGA** button. The programming file obtained from the Build stage is downloaded to the physical FPGA device on the daughter board. Download occurs over the PC's JTAG link to the Desktop NanoBoard, with progress displayed in Altium Designer's Status Bar.

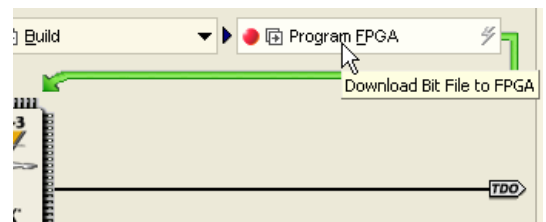


Figure 20. Starting the programming process.

Once the design has been downloaded, the text underneath the physical device's icon in the Devices view will change from *Reset* to *Programmed*. On the hardware side, the 'Program' LED on the daughter board will be lit (Green), confirming that the design has been loaded into the physical device.

For more information on the Process Flow and the **Devices** view, refer to the document [AP0103 Processing the Captured FPGA Design](#).

- Use switches on the NanoBoard's DIP-switch as follows:
 - Switch 8*: switch ON to start the counter if previously stopped and/or display the LEDs shifting in from the left
 - Switch 7*: switch ON to start the counter if previously stopped and/or display the LEDs shifting in from the right
 - Switch 6*: switch ON to stop the counter. Count will be stopped provided Switches 7 and 8 are OFF.
- Press the 'DAUGHTER BD TEST/RESET' button on the NanoBoard to clear the LEDs.
- Save the project.

You will notice that the User LEDs on the Desktop NanoBoard look to be all on at the same time, which really defeats the purpose of having a twisted-ring counter! This is because the reference clock on the NanoBoard is 20MHz. We need to slow down the clock by a factor of one million to see the LEDs displaying sequentially. In the next section of this tutorial, we will look at adding some clock division to achieve this and, in doing so, also explore the use of hierarchy in an FPGA design.

Exploring Design Hierarchy

While the FPGA project file (*PrjFpg) links the various source documents into a single project, the document-to-document and net connective relationships are defined by information in the documents themselves.

In a hierarchical design, the design is partitioned into logical blocks, with each block represented on the top schematic sheet by a sheet symbol. The **Filename** attribute of each sheet symbol references the underlying design file that it represents. This underlying file can be:

- A schematic sheet
- An OpenBus System document
- A VHDL file
- A Verilog file.

A schematic sub-sheet can also include sheet symbols referencing lower design files. Using this approach a design hierarchy of any depth or complexity can be created.

Hierarchical net and bus connectivity between documents obeys the standard hierarchical project connection behavior, where ports on the sub-document connect to sheet entries of the same name in the sheet symbol that represents that document, as shown for schematic and VHDL sub-documents in Figure 21.

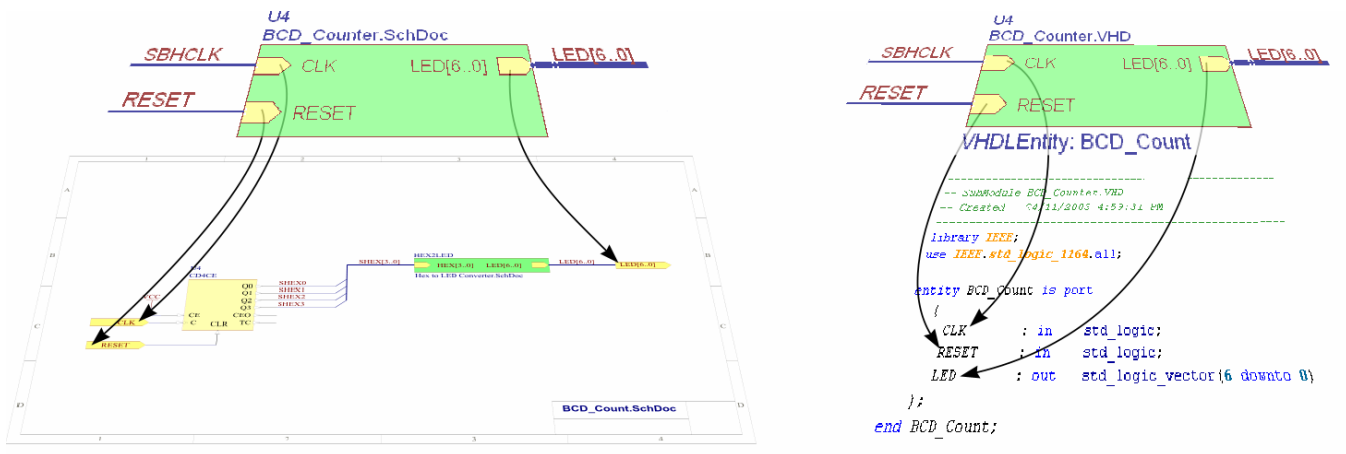


Figure 21. Hierarchical net connectivity is from the sheet entries to matching ports on the document below.

In the base design for our simple counter example, we saw that the synchronous clock signal sourced from the Desktop NanoBoard was too fast. To slow down the counter we will add clock division circuitry. Rather than adding this circuitry to the Simple_Counter schematic, we will capture it in a sub-file (first in a schematic and then in a VHDL file), to demonstrate how hierarchical designs can be used when programming an FPGA.

Clock Division using a Schematic Sub-Sheet

Let's go ahead and capture the clock division circuitry in a schematic sub-sheet.

1. Open the schematic document Simple_Counter.SchDoc.
2. Place a sheet symbol (**Place » Sheet Symbol**) initially into free space on the sheet. This sheet symbol will represent the sub-sheet on which our clock division circuitry will be defined.
3. Double-click on the sheet symbol and in the *Sheet Symbol* dialog that appears set the following:
 - **Designator:** U_Clock_Divider
 - **Filename:** Clock_Divider.SchDoc.
4. Add a single sheet entry (**Place » Add Sheet Entry**) to the left and right of the sheet symbol. Set the following properties:
 - Left-hand sheet entry: **Name:** CLK_REF, **I/O Type:** Input
 - Right-hand sheet entry: **Name:** CLK_OUT, **I/O Type:** Output
5. Wire the sheet symbol into the main circuit and next to the CLK_REF port component, as shown in Figure 23.

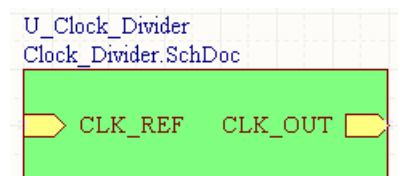


Figure 22. Sheet symbol referencing the underlying schematic sub-sheet.

Getting Started with FPGA Design

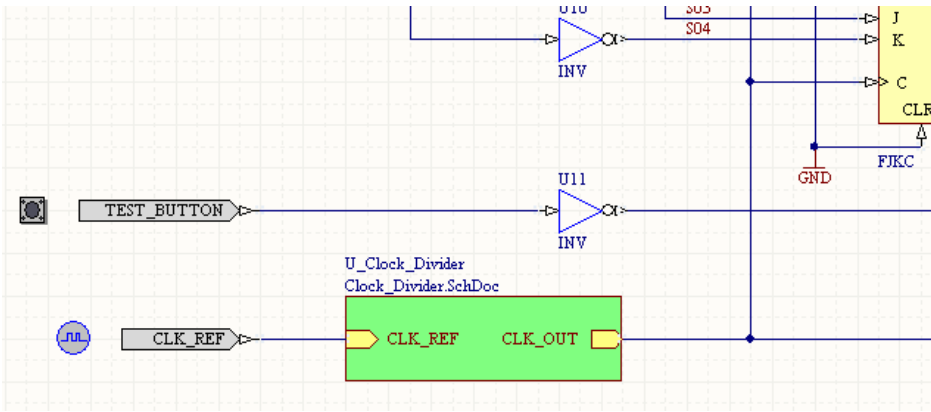


Figure 23. Simple_Counter schematic with sheet symbol for Clock_Divider.SchDoc sub-sheet placed and wired.

Now we have our parent sheet symbol, we must create the actual sub-sheet that it references and the required circuitry thereon.

Note: The subsequent steps 6-9 take you through the process of defining the schematic sub-sheet and its content from scratch. To skip this sequence of steps, right-click on the Simple_Counter.PrjFpg entry in the Projects panel and choose Add Existing to Project. In the Choose Documents to Add to Project dialog that appears, navigate to and open the file Clock_Divider.SchDoc, located in the \Examples\Tutorials\Getting Started with FPGA Design folder of the installation. Then save the project and schematic top-sheet and proceed with step 10.

6. Right-click on the sheet symbol and choose **Sheet Symbol Actions » Create Sheet From Symbol**. The new schematic document, Clock_Divider.SchDoc, is created and opened as the active document in the main design window. Initially, this document contains two ports – CLK_REF and CLK_OUT – which correspond (and connect upwards to) the respective sheet entries in the parent sheet symbol.
7. Access the **Libraries** panel and place six clock divider components (CDIV10DC50 – divide by 10 with 50% duty cycle output) from the FPGA Generic integrated library (FPGA Generic.IntLib) onto this new sheet. Connect the components in series and between the two ports as shown in Figure 24.

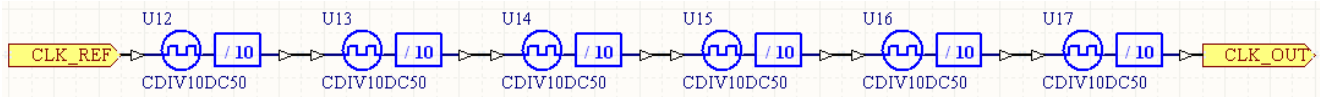


Figure 24. Clock division circuitry on the schematic sub-sheet Clock_Divider.SchDoc.

8. Annotate the components using the **Tools » Annotate Schematics Quietly**.
9. Use **File » Save All** to save both schematics and the parent project. Accept the default location (i.e. same folder as top-level schematic) when saving Clock_Divider.SchDoc.
10. Compile the project to check for any errors. If there are, resolve, save and recompile.
11. After compiling, you can check the sheet hierarchy of the project in the **Projects** panel. The project will now recognize the sub-sheet (Clock_Divider.SchDoc) as a child of the Simple_Counter schematic.

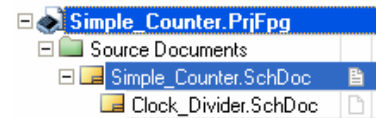


Figure 25. Hierarchy with a schematic sub-sheet.

Now we have finished adding the clock division circuitry, let's test it out.

12. Open the **Devices** view, ensuring that the **Live** option is still enabled and the **Connection** indicator is still Green.
13. The physical FPGA device is still programmed and the Process Flow still shows all stages as successfully completed (Green). How so, when we have modified the design's source documents? The answer is in the **Ignore FPGA source** option, which is enabled by default. We need to disable this option so that the modified source documents are taken into account when assessing the status of the Process Flow. Go ahead and disable the **Ignore FPGA source** option – each stage of the Process Flow will turn Yellow, indicating that it is out of date and must be run again.

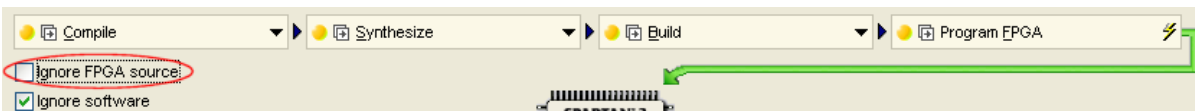


Figure 26. Disable the Ignore FPGA source option to take into account modified source when assessing the Process Flow.

14. Reprogram the daughter board's FPGA device. The simplest way to do this is to click directly on the **Program FPGA** stage in the Process Flow. All prior stages will be run automatically in sequence, because they are now marked as out of date. After the synthesis stage, a corresponding intermediate VHDL file for our new schematic sub-sheet will appear in the generated VHDL files for the project.

Once programmed, start the counter (set either switch 7 or switch 8 of the DIP-switch to ON) and observe the output on the NanoBoard's User LEDs has slowed down and we can now see the sequence more clearly. Use the DIP-switch to play with the direction now that it has meaning!

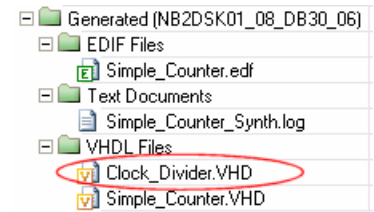


Figure 27. Additional VHDL file for the sub-sheet, generated on synthesis.

Clock Division using a HDL Sub-File

The concept of design hierarchy can easily be extended when capturing the design using a mixture of schematic sheets and HDL code. A VHDL or Verilog sub-document is referenced in the same way as a schematic sub-sheet, by specifying the sub-document filename in the sheet symbol that represents it.

When referencing a VHDL sub-document, the connectivity is from the sheet symbol to an entity declaration in the VHDL file. To reference an entity with a name that is different from the VHDL filename, include the `VHDLEntity` parameter in the sheet symbol, whose value is the name of the Entity declared in the VHDL file.

The process is similar when referencing a Verilog sub-document, where the connectivity is from the sheet symbol to a module declaration in the Verilog file. To reference a module with a name that is different from the Verilog filename, include the `VerilogModule` parameter in the sheet symbol, whose value is the name of the Module declared in the Verilog file.

Let's take a look at substituting the schematic sub-sheet we have just added, with a VHDL file that implements clock division by using a simple delay of one million.

First we will create the VHDL source file.

Note: The subsequent steps 1-3 take you through the process of defining the VHDL sub-file and its content from scratch. To skip this sequence of steps, right-click on the `Simple_Counter.PrjFpg` entry in the Projects panel and choose **Add Existing to Project**. In the *Choose Documents to Add to Project* dialog that appears, navigate to and open the file `Clock_Divider.vhd`, located in the `\Examples\Tutorials\Getting Started with FPGA Design` folder of the installation.

Then save the project and proceed with step 4.

1. Right-click on the `Simple_Counter.PrjFpg` entry in the **Projects** panel and choose **Add New to Project » VHDL Document**. A new VHDL document is created (`VHDL1.vhd`) and opened as the active document in the main design window. Save this document with the name `Clock_Divider.vhd`, in the same location as the project file.
2. Type the following VHDL code into the document:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Clock_Divider is
    port (
        CLK_REF : in std_logic;
        CLK_OUT : out std_logic
    );
end entity;

architecture RTL of Clock_Divider is
begin
    process(CLK_REF)
        variable i : integer range 0 to 999999;
    begin
```

Getting Started with FPGA Design

```

if rising_edge(CLK_REF) then
  if i = 0 then
    CLK_OUT <= '1';
    i := 999999;
  else
    CLK_OUT <= '0';
    i := i - 1;
  end if;
end if;
end process;
end architecture;

```

3. Save the document

Now we have our VHDL source file, we can create a sheet symbol directly from it. Before we do though, let's remove the existing sheet symbol from the top-level schematic and also its referenced schematic sub-sheet.

- Remove the schematic sub-sheet `Clock_Divider.SchDoc` from the FPGA project by right-clicking on its name in the **Projects** panel and choosing **Remove from Project** from the menu
- Open the `Simple_Counter` schematic (`Simple_Counter.SchDoc`). Click on the existing sheet symbol and press the **Delete** key.
- From the main menus, choose **Design » Create Sheet Symbol From Sheet Or HDL**. In the *Choose Document to Place* dialog that appears, select the `Clock_Divider.vhd` entry and click **OK**.
- Place and wire the new sheet symbol into the circuit as shown in Figure 28. Notice that the **Designator** and **Filename** have been automatically set to `U_clock_divider` and `Clock_Divider.vhd` respectively. The `VHDLENTITY` parameter has also been added, with value = `clock_divider` (the name of the entity in our VHDL sub-file).

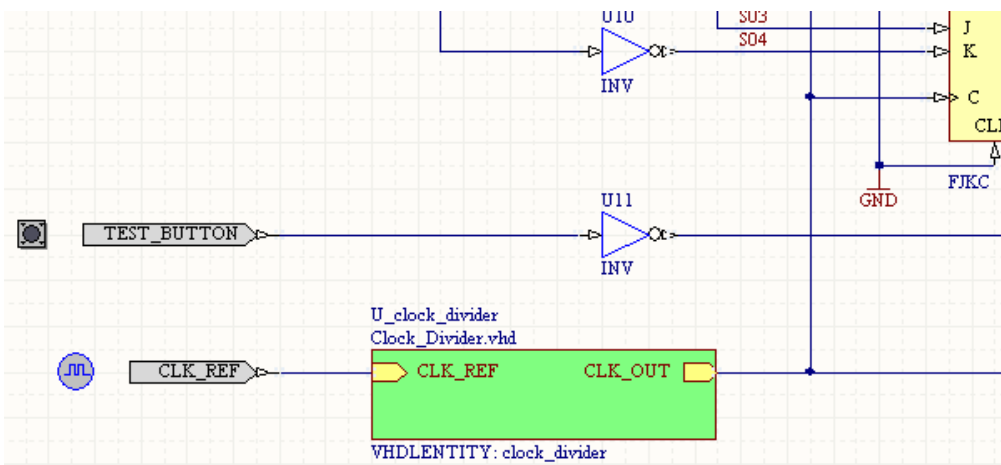


Figure 28. Simple_Counter schematic with sheet symbol for Clock_Divider.vhd sub-file placed and wired.

- Save the schematic and project.
- Compile the project to check for any errors. If there are, resolve them, save and recompile.
- After compiling, you can check the hierarchy of the project in the **Projects** panel. The project will now recognize the VHDL sub-file (`Clock_Divider.vhd`) as a child of the `Simple_Counter` schematic.

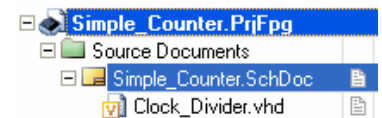


Figure 29. Hierarchy with a VHDL sub-file.

Now we have finished adding the VHDL version of the clock division circuitry, let's test it out.

- Open the **Devices** view and reprogram the daughter board's FPGA device by clicking the **Program FPGA** stage of the Process Flow.
- Once programmed, start the counter (set either switch 7 or switch 8 of the DIP-switch to ON). You should be able to clearly see the slowed-down counter output on the NanoBoard's User LEDs and control the output via the DIP-switch as before.

Monitoring the State of Device Pins – Live!

Once the design has been downloaded to the FPGA, the Hard Devices chain can be used to monitor the state of the FPGA pins. This is achieved using the device's associated **JTAG Viewer** panel (accessible from its instrument panel) set to operate in **Live Update** mode.

Where high-density component packaging makes physical probing of device pins impossible, the **JTAG Viewer** panel facilitates physical design debugging, 'virtual-style'. It uses the JTAG communications standard to interrogate the state of the pins in any JTAG compliant device in your design, not just the FPGAs. It presents the state of each pin, and includes an image of both the schematic symbol and the footprint, helping you to analyze and debug your design.

Now that we have introduced clock division to slow down the counter, we can observe the activity at the pins of the device much more clearly – so let's go ahead and have a peak 'under the bonnet' as it were.

1. From the **Devices** view, and with the program still running on the FPGA, double-click on the icon for the physical FPGA device in the Hard Devices chain. The **Instrument Rack – Hard Devices** panel appears (Figure 30).



Figure 30. Instrument panel for the physical FPGA device.

2. Click the **JTAG Viewer Panel** button to access the **JTAG Device Viewer** panel. Ensure that both **Live Update** and **Hide Unassigned I/O Pin** options are enabled (Figure 31).

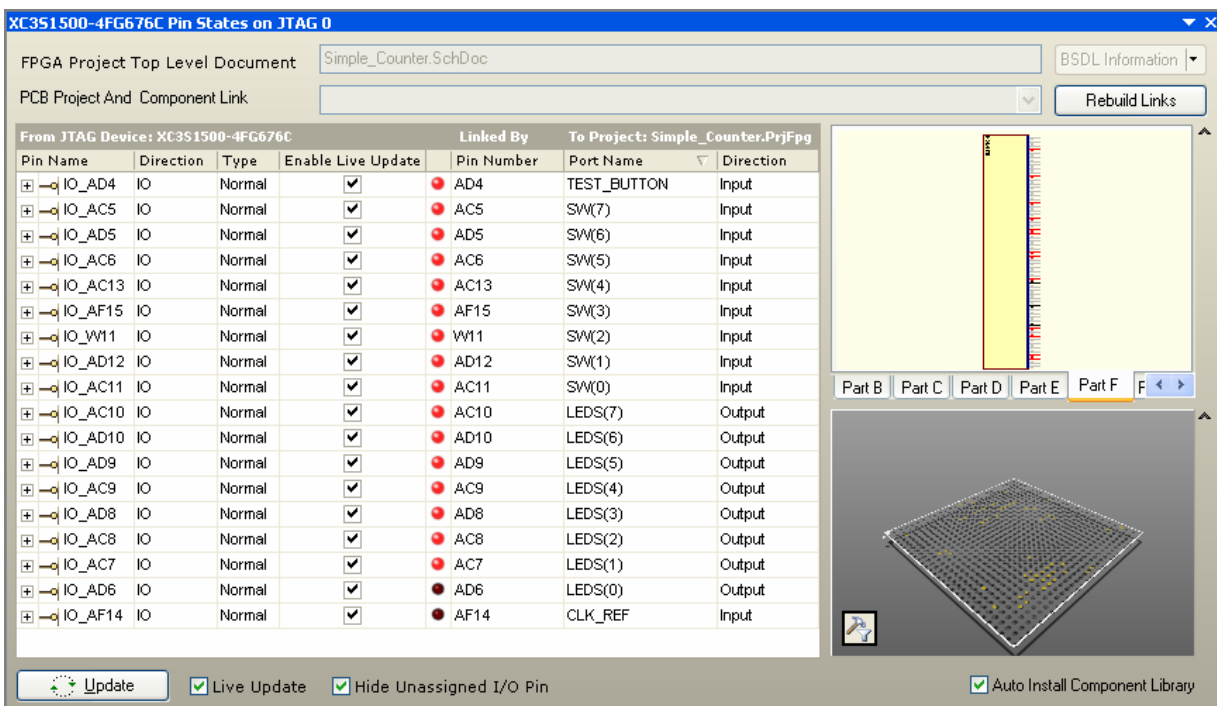


Figure 31. Live monitoring of device pins.

3. You are now viewing the pins of the physical FPGA device, while the circuit is in operation. Note how the LED icons light up next to the LEDS() ports as the circuit operates and reflect the distinctive output of the twisted-ring counter. You can also see the corresponding pins on the component symbol and footprint light up, as each pin is made active.
4. Change the direction of the count sequence using the NanoBoard's DIP-switch and see the change reflected in the panel.


Adding Virtual Instrumentation to the Mix

To test the state of internal nodes in the design you can 'wire in' virtual instruments. The 'hardware' portion of the instrument is placed and wired on the schematic like other components, and then synthesized into the FPGA. The interface to each instrument is accessed from within the **Devices** view.

Virtual instruments are placed from the `\Library\Fpga\FPGA Instruments.IntLib` library.

Let's add a couple of instruments into our design:

- A frequency counter (FRQCNT2) – which we will use to display the output frequency from our clock division circuit.
- A digital IO module (DIGITAL_IO) – which we will use to display the output of the counter and also the current state of the three switches associated to the NanoBoard's DIP-switch.

 For more detailed information on these instruments, refer to the documents [CR0101 FRQCNT2 Frequency Counter](#) and [CR0179 DIGITAL_IO Configurable Digital IO Module](#) respectively.

At this point, the current incarnation of the project has clock division supplied by an underlying VHDL sub-file. For this part of the tutorial we will leave this as is – there is no need to revert back to the schematic implementation of the clock division (although you could if you prefer it!).

Add the Frequency Counter

Let's start off by placing and wiring the frequency counter instrument into our simple counter design.

1. Open the Simple_Counter schematic (`Simple_Counter.SchDoc`).
2. Access the **Libraries** panel and place the FRQCNT2 component from the FPGA Instruments integrated library (`FPGA Instruments.IntLib`), to the bottom-right of the sheet symbol used to reference the VHDL sub-file.
3. Annotate using the **Tools » Annotate Schematics Quietly** command.

The signal we want to monitor is the output of the clock divider (`CLK_OUT`). We'll connect this to the `FREQA` input of the instrument. The `TIMEBASE` signal should be the original frequency from the NanoBoard (`CLK_REF`).

4. Go ahead and wire up the instrument as shown in Figure 32. Place a GND power port, connected direct to the instrument's `FREQB` pin, as we are not using this channel.

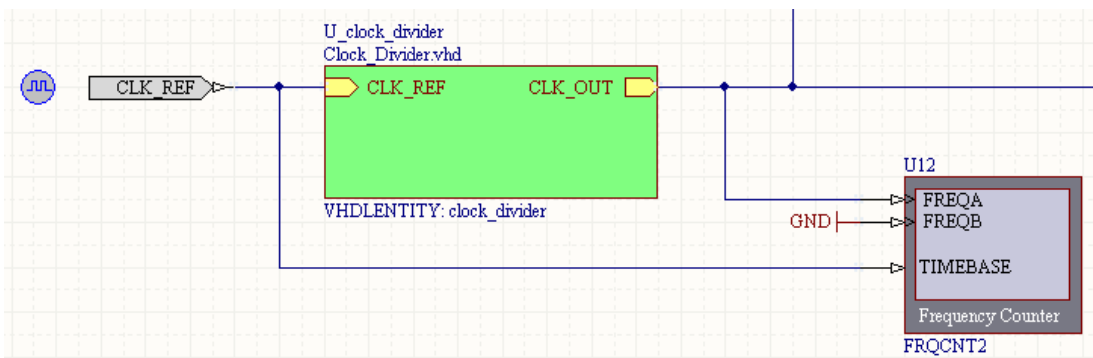


Figure 32. Frequency counter instrument placed and wired into the simple counter design.

Add the Digital IO Module

Now let's add the digital IO instrument.

1. Access the **Libraries** panel and place the DIGITAL_IO component, centrally and above the main circuitry in the design.
2. Annotate using the **Tools » Annotate Schematics Quietly** command.

Before we consider the wiring, let's configure the instrument with the signals we wish to monitor.

3. Right-click on the instrument and choose **Configure U13 (DIGITAL_IO)** from the menu to access the *Digital I/O Configuration* dialog. (Figure 33). Notice that by default, the instrument is configured to have a single 8-bit input bus (`AIN[7..0]`) and a single 8-bit output bus (`AOUT[7..0]`).

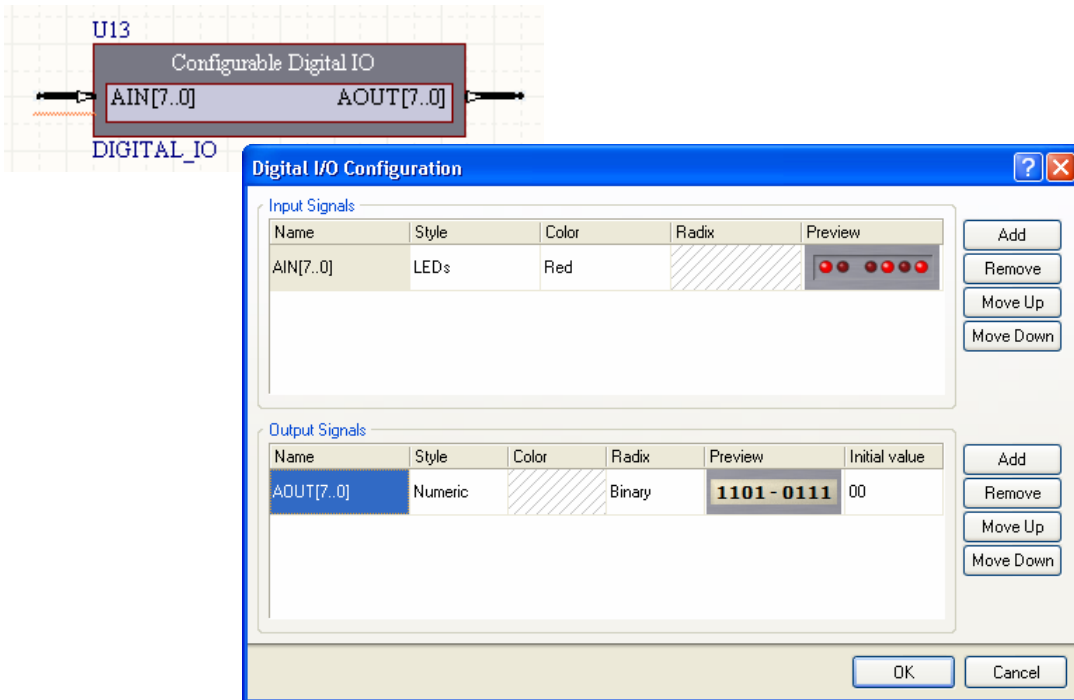


Figure 33. Access to controls for configuring the Digital IO instrument.

4. As we are not generating any outputs, we can simply remove the default entry. Click to select the `AOUT[7..0]` signal and click the associated **Remove** button for the section.
5. The default input signal is the right width for our purposes, so let's keep the signal, but give it a more meaningful name. Let's call it `Count_Output[7..0]`. We will keep the **Style** setting as **LEDs**, but change the **Color** setting to **Green** (just to mimic the color of the LEDs on the NanoBoard!).
6. To monitor the associated DIP-switch signals, let's now add a further three input signals to our configuration. Define these signals as:
 - Signal 1 – **Name:** `Shift_Left`, **Style:** LEDs, **Color:** Green.
 - Signal 2 – **Name:** `Shift_Right`, **Style:** LEDs, **Color:** Green.
 - Signal 3 – **Name:** `STOP`, **Style:** LEDs, **Color:** Red.

The Input Signals region of the dialog should now appear as shown in Figure 34.

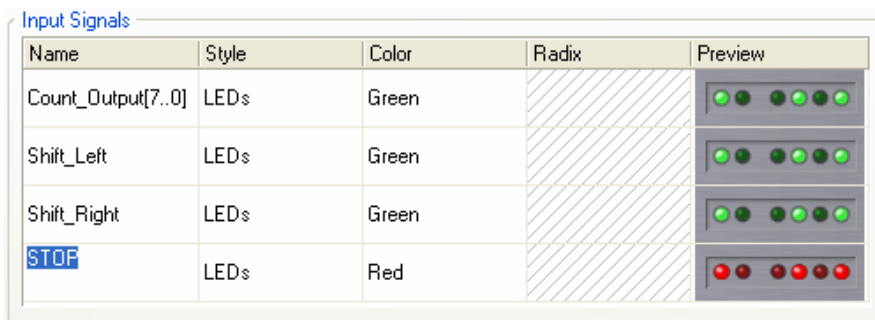


Figure 34. Fully configured Digital IO instrument.

7. Wire up the instrument as shown in Figure 35. Notice that we avoid messy wiring by using net labels to tap off the signals we wish to monitor (`S04`, `S03`, `S02`). These net labels already exist for the points we wish to interrogate, so it becomes a simple matter of copy and paste. Notice that we are monitoring the signals after they have passed through their respective inverters. This is because the switches associated with the DIP-switch are active Low, so it would be better to see a light come on when they are set to the ON position, rather than go off.

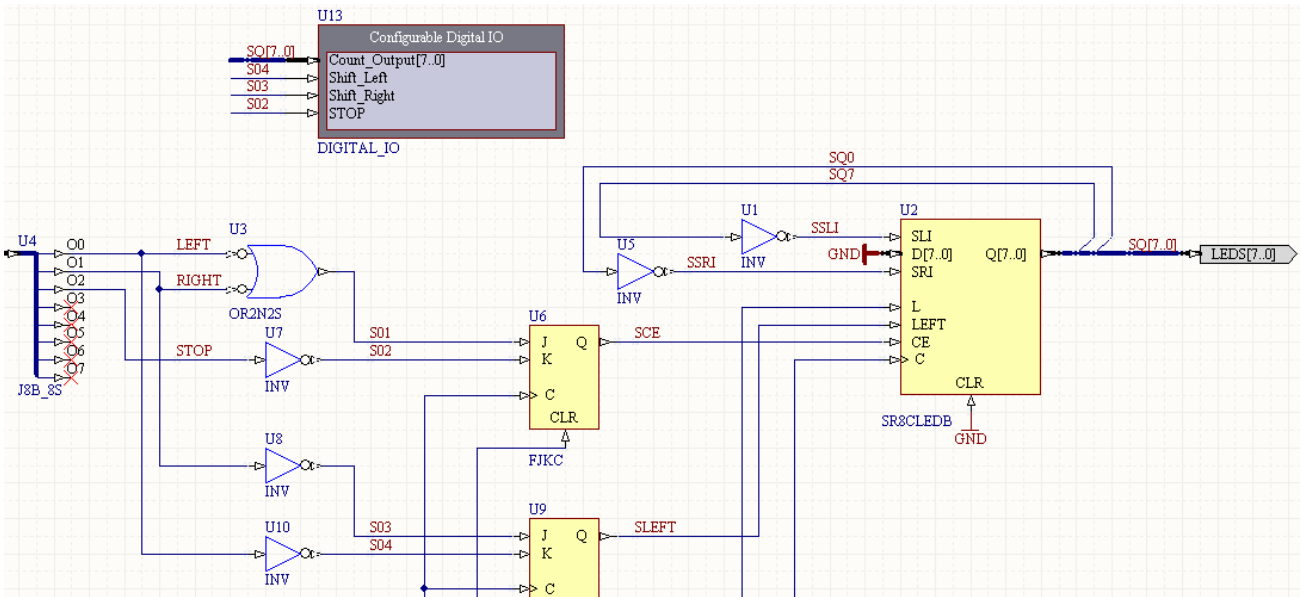


Figure 35. Digital IO instrument placed and wired into the simple counter design.

Enabling the Soft Devices JTAG Chain

Communications from the Altium Designer software environment to embedded processors and virtual instruments in an FPGA design, is carried out over a JTAG communications link. This is referred to on the Desktop NanoBoard as the Soft JTAG (or Nexus) chain.

The Soft JTAG chain signals (NEXUS_TMS, NEXUS_TCK, NEXUS_TDI and NEXUS_TDO) are derived in the Desktop NanoBoard's NanoTalk Controller (Xilinx Spartan-3). As part of the communications chain, these signals are wired to four pins of the daughter board FPGA. To interface to these pins, you need to place the NEXUS_JTAG_CONNECTOR design interface component (Figure 36). This can be found in the FPGA NB2DSK01 Port-Plugin integrated library (\Library\Fpga\FPGA NB2DSK01 Port-Plugin.IntLib).

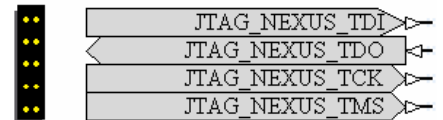


Figure 36. Nexus JTAG Connector.

This component 'brings' the Soft JTAG chain into the design. In order to wire all relevant Nexus-enabled devices (in our case the two virtual instruments) into this chain, you need to also place a NEXUS_JTAG_PORT component (Figure 37), and connect this directly to the NEXUS_JTAG_CONNECTOR. This component can be found in the FPGA Generic integrated library (\Library\Fpga\FPGA Generic.IntLib).

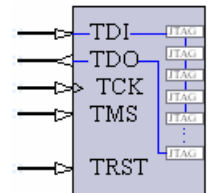


Figure 37. Nexus JTAG Port.

The presence of the NEXUS_JTAG_PORT component instructs the software to wire all components that possess the parameter NEXUS_JTAG_DEVICE=True into the Soft JTAG chain.

For information on the JTAG communications, refer to the document [AR0130 PC to NanoBoard Communications](#).

1. Place the NEXUS_JTAG_CONNECTOR and NEXUS_JTAG_PORT components onto the Simple_Counter schematic, connecting them together.
2. Place and connect a VCC Power Port to the TRST input of the NEXUS_JTAG_PORT component.

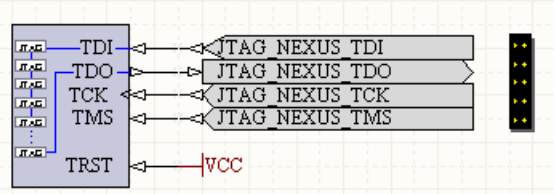


Figure 38. Connecting JTAG devices into the Soft JTAG chain.

Our design is now fully wired to include our two virtual instruments and should appear as shown in Figure 39.

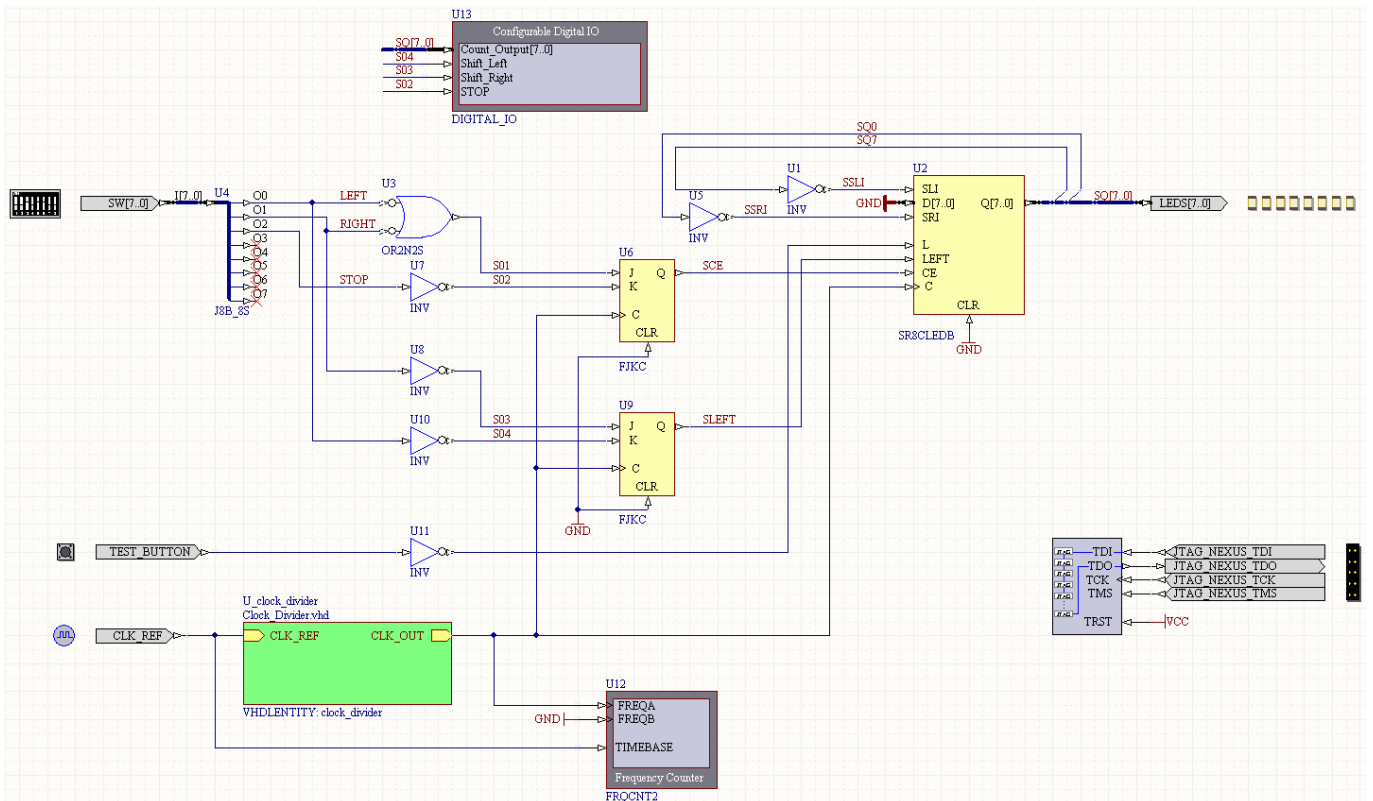


Figure 39. Final design, complete with the two virtual instruments.

3. Save the schematic and the parent project.
4. Recompile the project. This time you should see no messages in the **Messages** panel. This is because all SQ outputs are being used by the Digital IO instrument and so all now have a load!

Accessing Instrument Controls

In the previous section, we looked at the mechanics of adding the virtual instruments into the Soft JTAG chain. It is a good idea – before we reprogram the FPGA and actually access the instruments – to consider access to these instruments is made possible within Altium Designer.

The host computer is connected to the target instruments using the IEEE 1149.1 (JTAG) standard interface. This is the physical interface, providing connection to physical pins of the FPGA device in which the instruments are to be embedded.

The Nexus 5001 standard is used as the protocol for communications between the host and all devices that are debug-enabled with respect to this protocol. This includes the digital I/O and frequency counter, as well as other Nexus-compliant devices such as debug-enabled processors, frequency generators, logic analyzers and cross-point switches.

All such devices are connected in a chain – the Soft Devices chain – which is determined when the design has been implemented within the target FPGA device and presents within the **Devices** view. It is not a physical chain, in the sense that you can see no external wiring – the connections required between the Nexus-enabled devices are made internal to the FPGA itself.

1. Open the **Devices** view and reprogram the daughter board FPGA device.
2. Once programmed, the Soft Devices chain will appear as the last chain in the view, containing icons for our two virtual instruments (Figure 40).



Figure 40. Virtual instruments are presented in the Soft Devices chain, once the FPGA is programmed.

Getting Started with FPGA Design

- Start the counter (if not already) by toggling either switch 7 or switch 8 of the NanoBoard's DIP-switch.
- Double-click on the icons for both instruments, to display their associated instrument panels in the **Instrument Rack – Soft Devices** panel (Figure 41). These panels provide the necessary controls and displays for interacting with the instruments in real-time.

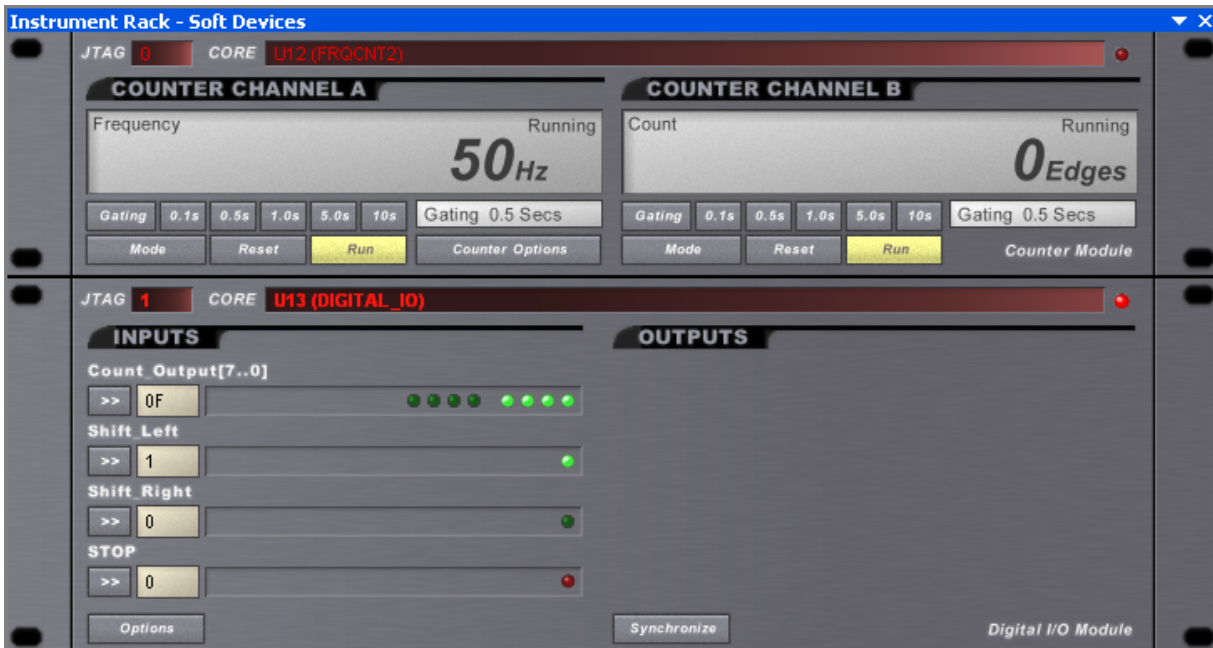


Figure 41. Respective instrument control panels for the frequency counter and digital IO module.

Looking at the panels we can see a couple of oddities. First, the frequency counter displays a frequency of 50Hz. Remember that our reference frequency from the NanoBoard is 20MHz and our clock divider provides divide-by-one million. The expected result of 20Hz is not there! Secondly, the LED display for the output of the counter, on the digital IO instrument's panel, is rather jittery – the LEDs are certainly not shifting in a smooth fashion as they are on the NanoBoard itself.

To remedy both of these situations, we need to further configure options relating to each instrument, from within their respective instrument panels.

- From the instrument panel for the frequency counter, click on the **Counter Options** button. In the *Counter Module – Options* dialog that appears, change the **Counter Time Base** entry from the default 50.000MHz to 20.000MHz (to be the same frequency as the signal we have wired to the instrument's TIMEBASE input). Close the dialog and notice, reassuringly, that the displayed frequency is now 20Hz.
- From the instrument panel for the digital IO module, click on the **Options** button. In the *Digital I/O Module – Options* dialog that appears, change the **Update Display From Core Every** entry from the default 250ms down to the minimum 100ms. Close the dialog. With this faster update, notice that the LEDs display more smoothly.

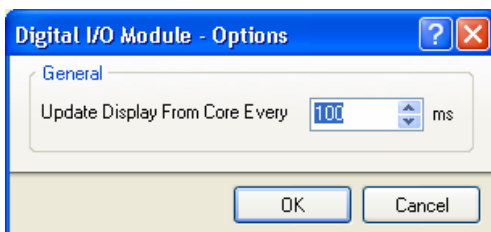


Figure 43. Increase the display update speed.

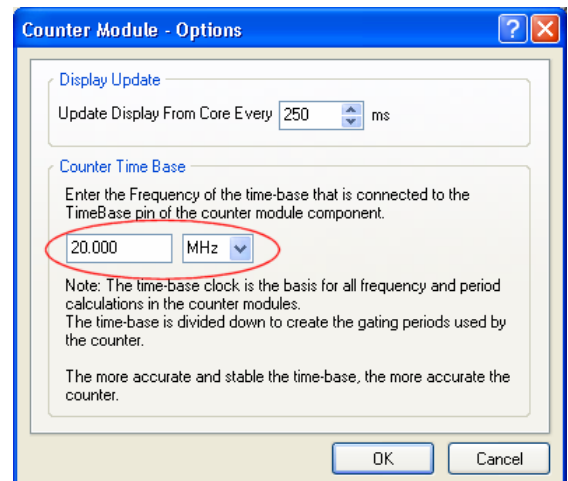



Figure 42. Change the counter time base for the instrument to be same as the frequency physically wired to the instrument's TIMEBASE input.

- Have a play with the design – toggling the count direction and stopping it – and observe that the digital IO module updates live and in accordance with your changes.

Where to Now?

That concludes this introductory tutorial into the basics of FPGA design using the Altium Innovation Station. Armed with a solid grounding of the base concepts covered during the implementation of our simple counter, you can now embark on your own FPGA designs.

The development environment provided by Altium Designer and the Desktop NanoBoard offers a far more extensive set of tools and features than those seen during the course of this tutorial. 32-bit processing, hardware acceleration and custom logic are just a few catchphrases to tempt the taste buds – not to mention a fully customizable virtual instrument where you even get to layout your own panel interface!

 For a high-level introduction to FPGA design and the Altium Innovation Station, refer to the document [GU0123 An Introduction to Embedded Intelligence](#).

For further, more advanced tutorials, covering more of the tools and features on offer, refer to the following documents:

[TU0122 Getting Started with Embedded Software](#)

[TU0123 Creating a Core Component](#)

[TU0126 Checking Signal Integrity on an FPGA Design](#)

[TU0128 Implementing a 32-bit Processor-based Design in an FPGA](#)

[TU0129 Converting an Existing FPGA Design to the OpenBus System](#)

[TU0130 Getting Started with the C-to-Hardware Compiler](#)

[TU0131 Capturing Video the Easy Way](#)

[TU0133 Designing Custom FPGA Logic using C](#)

[TU0135 Adding Custom Instrumentation to an FPGA Design](#)

Revision History

Date	Version No.	Revision
16-Jan-2004	1.0	Initial release
23-Sep-2004	1.1	Clock divider components updated
18-Jan-2005	1.2	Components in Johnson_Counter.SchDoc updated
13-Apr-2005	1.3	Updated for Altium Designer
22-Aug-2005	1.4	Verilog references included
28-Nov-2005	1.5	Updated for Altium Designer 6
12-Apr-2007	1.6	Updated for Altium Designer 6.7
17-May-2008	2.0	Updated for Altium Designer Summer 08. Project and schematic documents renamed to Simple_Counter.PrjFpg and Simple_Counter.SchDoc respectively. Tutorial content enhanced throughout. Section on virtual instrumentation added.

Software, hardware, documentation and related materials:

Copyright © 2008 Altium Limited. All Rights Reserved.

The material provided with this notice is subject to various forms of national and international intellectual property protection, including but not limited to copyright protection. You have been granted a non-exclusive license to use such material for the purposes stated in the end-user license agreement governing its use. In no event shall you reverse engineer, decompile, duplicate, distribute, create derivative works from or in any way exploit the material licensed to you except as expressly permitted by the governing agreement. Failure to abide by such restrictions may result in severe civil and criminal penalties, including but not limited to fines and imprisonment. Provided, however, that you are permitted to make one archival copy of said materials for back up purposes only, which archival copy may be accessed and used only in the event that the original copy of the materials is inoperable. Altium, Altium Designer, Board Insight, DXP, Innovation Station, LiveDesign, NanoBoard, NanoTalk, OpenBus, P-CAD, SimCode, Situs, TASKING, and Topological Autorouting and their respective logos are trademarks or registered trademarks of Altium Limited or its subsidiaries. All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same are claimed. v8.0 31/3/08.