



# Design Portability, Configurations and Constraints

## Summary

Article

AR0124 (v2.0) March 03, 2008

This article describes what is required for design portability, and the role of configurations and constraints in achieving this portability.

The design for an FPGA is captured in a set of schematic and/or VHDL source files. As well as the symbols, wiring and VHDL source that makes up the design, there is other essential information that must be captured.

This can be broadly labeled as implementation information and could include: the target FPGA, pin configuration details, device constraints such as place and route information, and design timing requirements. Rather than locking the design to a specific FPGA-on-PCB combination, separating this information allows design portability.

## Maintaining Design Portability

Designs that are created within the system have a high degree of portability, allowing them to be retargeted to different device and board environments with the minimum of difficulty.

This portability is achieved by a combination of:

- designing with generic components, that can be implemented in all supported device families
- Altium Designer's plug-in device driver model that allows support for new devices to be dropped in
- integrated PCB libraries that support bi-directional data linking from the FPGA design to the PCB design
- separating the design – captured in the source documents – from the device-specific and implementation-specific requirements, stored in constraint files.

## Generic Components

All of the logical components that are available for designing within an FPGA are targeted for all of the FPGAs supported by the system.

This includes everything from simple gates, all the way up to FPGA microprocessors and other high-level functional blocks. These components are pre-synthesized in a way that is optimized for the target architecture. In some cases, the implementation may be quite different, depending on the capabilities of the underlying target architecture. This allows a new 'high-level' device-independent design methodology to be adopted.

The system also supports designing from vendor-specific primitive libraries, but this is inherently device specific and will result in the design being 'trapped' within the target architecture. This is not recommended unless the highest level of optimization is required. If specific areas of the design do need to use features that are specific to a particular device, then these should be isolated as much as possible in order to avoid unnecessarily compromising the portability of the design.

## Nexus Driver Files

There is a large amount of information that must be known about each FPGA device family that is supported by the system. This information is 'plugged into' Altium Designer's underlying DXP integration platform in the form of a Nexus file. These files serve as 'drivers' within the system and contain all of the information, or references to it, that are required to utilize and interact with the chosen device. The Nexus file includes physical information about the device, such as details about the capabilities of each pin (including pin-banking data), boundary scan (JTAG) information and details on how to program the device – everything needed to interact with the device while it is online.

## Integrated PCB Libraries

Each device family also includes a PCB-level integrated library. This library contains schematic symbols, PCB footprints and 3D models for all of the devices within the family. For programmable devices such as CPLDs and FPGAs the library includes generic 'un-programmed' symbols that serve as templates for the final fully-programmed device.

## ***Design Portability, Configurations and Constraints***

These libraries are linked to the device descriptions with the Nexus files to allow all aspects of the device, from the internals of FPGAs to the high-level board level design process, to be fully integrated into the design and debug flow.

### **Constraining the Design**

Rather than storing device and implementation data in the source documents, this information is stored in separate files. These files constrain the design to the target device and PCB layout, allowing the same design to be re-targeted quickly and easily.

### **Configuring the Constraints**

---

Configurations provide a way for an Altium Designer project to be managed parametrically. For embedded projects this could allow a separate set of settings to be maintained for a debug version and a release version of the product. For a PCB project configurations create the scope for rules to be specified in a separate file, allowing different versions of the board to be developed with different rules.

Focusing specifically on FPGA projects, configurations allow the schematic/VHDL design to be developed without the need to specify either tool settings or target-device settings, such as project port to device pin mappings, on the schematic/VHDL source documents.

### **Classes of Configuration Information**

There are a number of classes of configuration information within an FPGA project.

#### **Specific to the Device and Board**

This is where pin information is considered. The physical location of a pin on a device is specific to the combination of a given device and a given board.

Each logical port at the top level of an FPGA design is given a logical name which can be used to reference the pin in a device-independent way. This logical port must be mapped to a physical pin on the target device. The final location of this pin on the device will be driven by board-level issues, such as how the pin best connects to other resources on the board.

For example a data pin, say DATA3, for a particular board may be implemented as pin 21 on a Xilinx QFP package, and as pin H7 on an Altera BGA package. So the same port may be connected through a different pin on a different device, or even if the device is the same, the port may be connected through a different pin on a different board layout. From the logical design perspective, there would be multiple configurations, one for each of these scenarios. Each would have a pin description that included the logical name DATA3, but the physical pin being described would be different.

Another example of device/board specific requirements is pin configuration constraints. For example, if the FPGA device is connected to a 2.5 volt device on one board and a 3.3 volt device on another, then the setup data for the pin can be defined as a constraint for that device and target board.

These constraints could be defined as a description of how a specific device is connected to the resources on a specific PCB.

#### **Specific to a Certain Device**

This would include specifications for how to use internal features of the particular device. A good example would be place and route controls for FPGA tools – these are very device specific, but do not have any impact on the way the device is connected to the outside world.

These constraints could be defined as a description of how to configure a specific device's internal resources for a specific FPGA design.

#### **Specific to a Project or Design**

This would include requirements which are associated with the logic of the design, as well as constraints on its timing. For example, specifying that a particular logical port must be allocated to a global clock net, and must be able to run at a certain speed. This class of constraint could be added directly to the source documents of the FPGA design. Defining them as separate from the source documents allows the design itself to be more parametric.

These constraints could be defined as a description of how to configure a specific FPGA design.

## How it Works – Configurations and Constraint Files

All of the three classes of constraints described above are implemented as constraints within constraint files. Constraint files can contain any number of different constraints, for any of the classes mentioned above. To ensure the most portable FPGA design, the most logical approach is to break the constraints into these three classes and store these in separate constraint files.

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

For example, consider an FPGA project that is targeted to:

- a Xilinx Spartan-XC2S300E QFP208 on a NanoBoard
- an Altera Cyclone QFP240 on a NanoBoard
- and a Xilinx Spartan-XC2S100E QFP144 on the user's own board design.

This would require three configurations – one for each target.

Assuming good design practice (from a portability perspective) this would theoretically require seven constraint files – separate constraint files to control the pin-outs for each of the three devices in their target boards, separate constraint files to control any internal place and route constraints for each of the three target devices, and one constraint file for logical design constraints. If there were no place and route constraints (which would usually be the case), then there would only be four constraint files.

Each configuration would then include three constraint files, two which are specific to the configuration and one which is common to all configurations.

If there were constraints that were common to the two different Xilinx devices (which are internally very similar) then it may be beneficial to create a constraint file for these. In this case, the extra constraint file would be added to two of the configurations.

## Diagrams of these Configurations and their Constraint Files

Figure 1 below shows the relationships between the constraint files and the configurations.

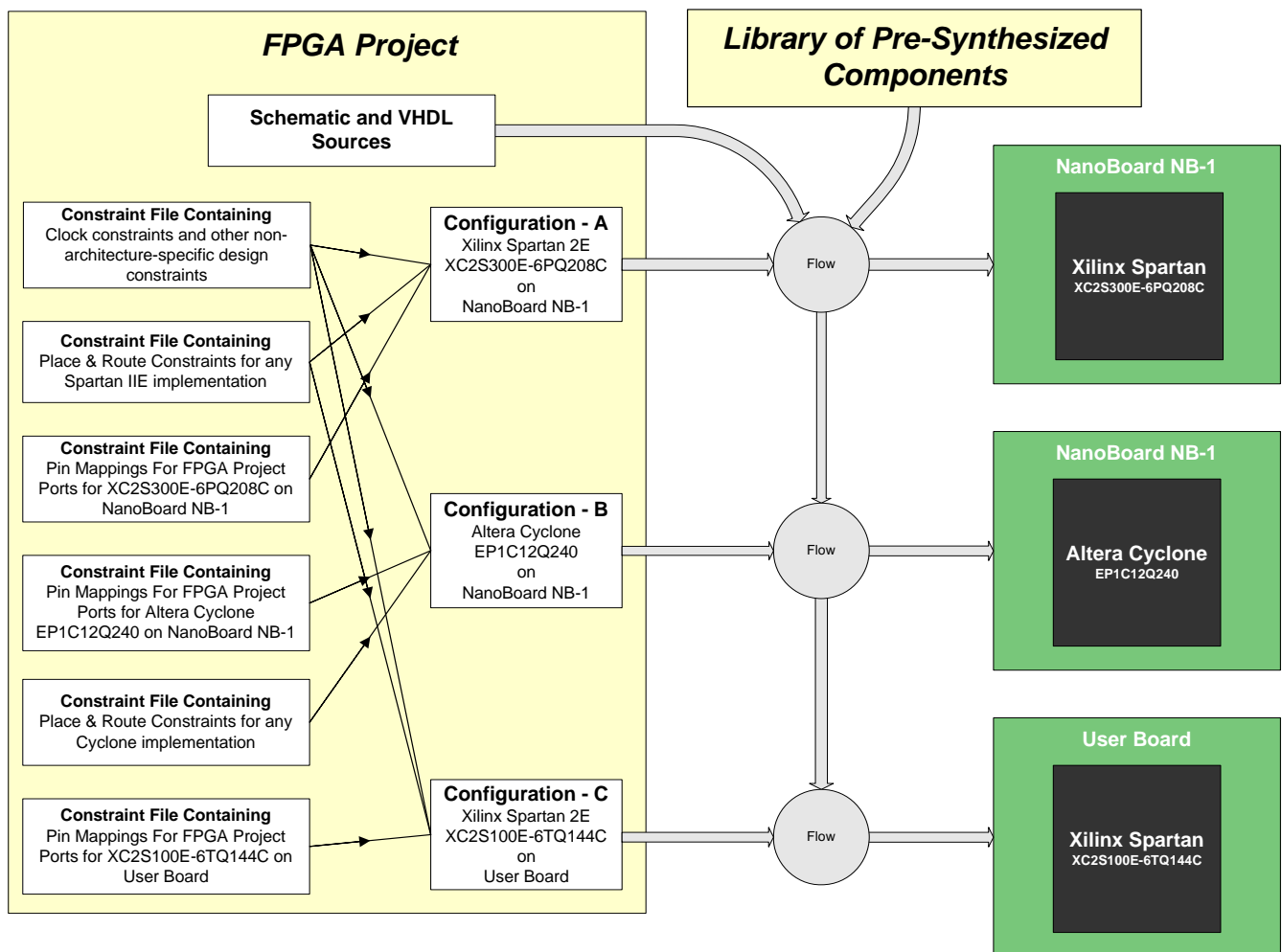


Figure 1. A single set of schematic and VHDL source documents, targeted to three different implementations by different sets of constraint files.

## Design Portability, Configurations and Constraints

The actual setups for each of the three configurations are shown below.

### For the Spartan 2E300 on the NanoBoard

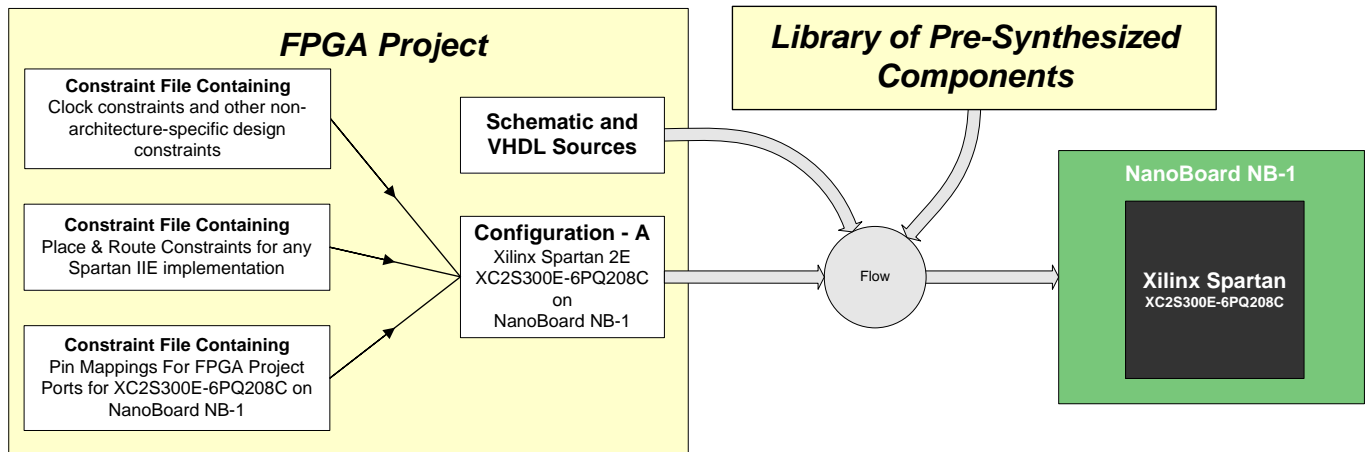


Figure 2. Design targeting a Xilinx Spartan2E 300, on the NanoBoard.

### For the Cyclone EP1C12 on the NanoBoard

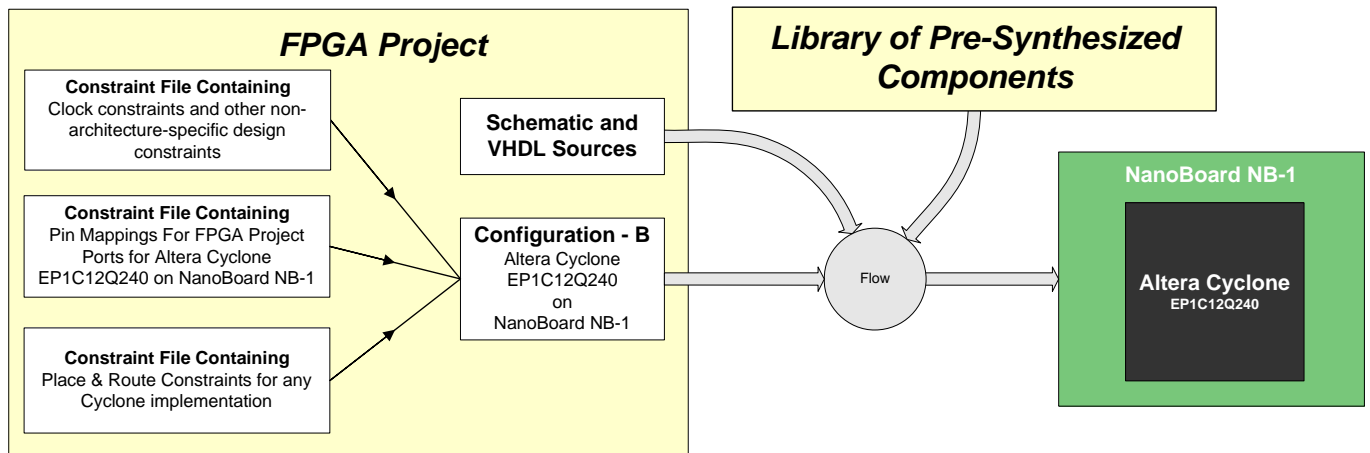


Figure 3. Design targeting an Altera Cyclone EP1C12, on the NanoBoard.

### For the Spartan 2E100 on a User Board

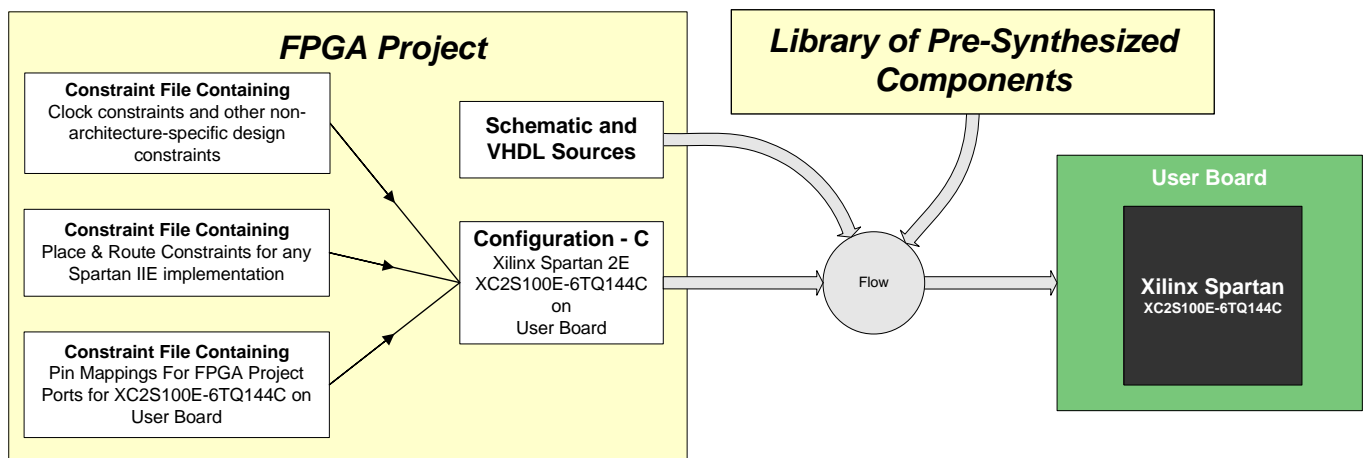


Figure 4. Design targeting a Xilinx Spartan2E 100, on a User Board.

## Using Configurations

Configurations and constraint files can be considered from two different perspectives.

### Existing Board

In the case where the FPGA designer is working with an existing (constructed) PCB, the resources of the board and how they are connected to the FPGA are already defined. This would include Altium NanoBoards and third party development or evaluation boards. The board would come with a constraint file that describes its resources and how they are connected. In the case of boards with plug-in devices, the board would come with a number of constraint files, one for each plug-in device.

### New Board being Developed

In this case, the design of the FPGA may be driving the design and layout of the PCB. Here the designer may let the FPGA tools allocate pins for the design and then use this information to drive the layout of the PCB. Alternatively, the PCB designer may allocate the pins on the FPGA and then pass this information to the FPGA designer.

In all of these cases, a constraint file holds the current assignments of these pins. This information is used to update either the PCB design or to drive the FPGA place and route tools. The information in this file can also be updated by importing from the FPGA tools, or importing from the PCB design after board-level optimization of pin assignments.

### An Iterative Approach

These two perspectives should not be considered as mutually exclusive. With the first case of an existing constructed board, there will most likely (in the case of NanoBoards and other development boards) be general purpose I/O headers that can be used as required. In this case, the constraint file that comes with the board should be considered as a template for the engineer's own design. The constraint file could be extended to include the engineer's new connections through the general purpose I/O connectors on the board, or an extra constraint file could be created to hold this information.

## Defining a Configuration

A configuration is essentially just a named list of constraint files. Configurations belong to the project, and are stored in the project file. Configurations are created and managed in the *Configuration Manager*, accessible through the **Project** menu.

Controls in the *Configuration Manager* allow the creation of new configurations, and the allocation of constraint files to a configuration. Add constraint files to the FPGA project to use them in a configuration.

## Understanding Constraints

Constraint files are simple documents within the system and can be compiled to add more intelligence to the whole process.

A constraint document contains a list of statements, known as constraint groups, each of which targets one or more objects and contains one or more constraints.

This example constraint group, shown in the typical constraint group syntax, specifies the target FPGA device:

```
Record=Constraint | TargetKind=Part | TargetId=XC2S300E-6PQ208C
```

Each constraint group can include multiple constraints, for example a constraint group can be added that targets a bus (a collection of nets). Multiple constraints can then be added to target the individual items within the bus.

The file syntax of this would be:

```
Record=Constraint | TargetKind=Port | TargetId=DB[7..0] | FPGA_PINNUM=P8,P7,P6,P5,P4,P3,P206,P205 |
FPGA_SLEW=FAST,FAST,FAST,FAST,FAST,FAST,FAST,FAST
```

This example specifies a set of constraints that target the Ports DB7..DB0. It then supplies eight FPGA\_PINNUM constraints and eight FPGA\_SLEW constraints.

Note that bus ports can be described in a constraint and then later, after compiling, these can be mapped to individual pins. There is no need to specify both levels (bus and single) as separate constraints.

Multiple constraints can target the same Port. For example the following constraint groups target the port CLK\_REF; the first specifies the pin number, the second specifies that this pin must be configured as a clock pin, the third specifies that this net must use a global clock resource, the fourth specifies that the place and route process should attempt to route this to achieve the specified

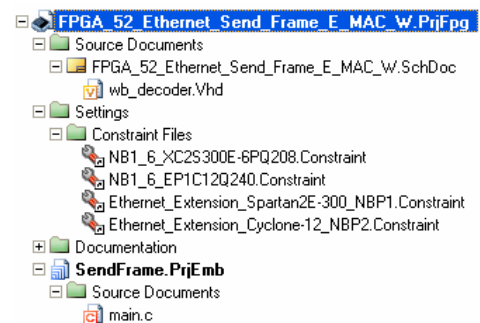


Figure 5. Example project with 4 constraint files.

## Design Portability, Configurations and Constraints

frequency.

```
Record=Constraint | TargetKind=Port | TargetId=CLK_REF | FPGA_PINNUM=P185
Record=Constraint | TargetKind=Port | TargetId=CLK_REF | FPGA_CLOCK_PIN=TRUE
Record=Constraint | TargetKind=Port | TargetId=CLK_REF | FPGA_CLOCK=TRUE
Record=Constraint | TargetKind=Port | TargetId=CLK_REF | FPGA_CLOCK_FREQUENCY=50 Mhz
```

Alternatively, these constraint requirements could be specified in the one constraint group:

```
Record=Constraint | TargetKind=Port | TargetId=CLK_REF | FPGA_PINNUM=P185 | FPGA_CLOCK_PIN=TRUE |
FPGA_CLOCK=TRUE | FPGA_CLOCK_FREQUENCY=50 Mhz
```

A more appropriate way to group them would be a group of those that are design-type information, and another for those that are device-on-PCB type information, in separate constraint files.


For example, the design-type constraint file would include:

```
Record=Constraint | TargetKind=Port | TargetId=CLK_REF | FPGA_CLOCK=TRUE
Record=Constraint | TargetKind=Port | TargetId=CLK_REF | FPGA_CLOCK_FREQUENCY=50 Mhz
```

And the device-on-PCB type constraint file would include:

```
Record=Constraint | TargetKind=Port | TargetId=CLK_REF | FPGA_PINNUM=P185
Record=Constraint | TargetKind=Port | TargetId=CLK_REF | FPGA_CLOCK_PIN=TRUE
```

Note that they have been kept as separate entries in the constraint files for readability.

 For more information on supported constraints, see the [Constraint File Reference](#).

## Mapping the FPGA Project to a PCB

Implementing a design into an FPGA on a PCB is done in the **Devices** view. Here the FPGA project, in combination with a configuration, is targeted to a board. An example is shown in Figure 6.

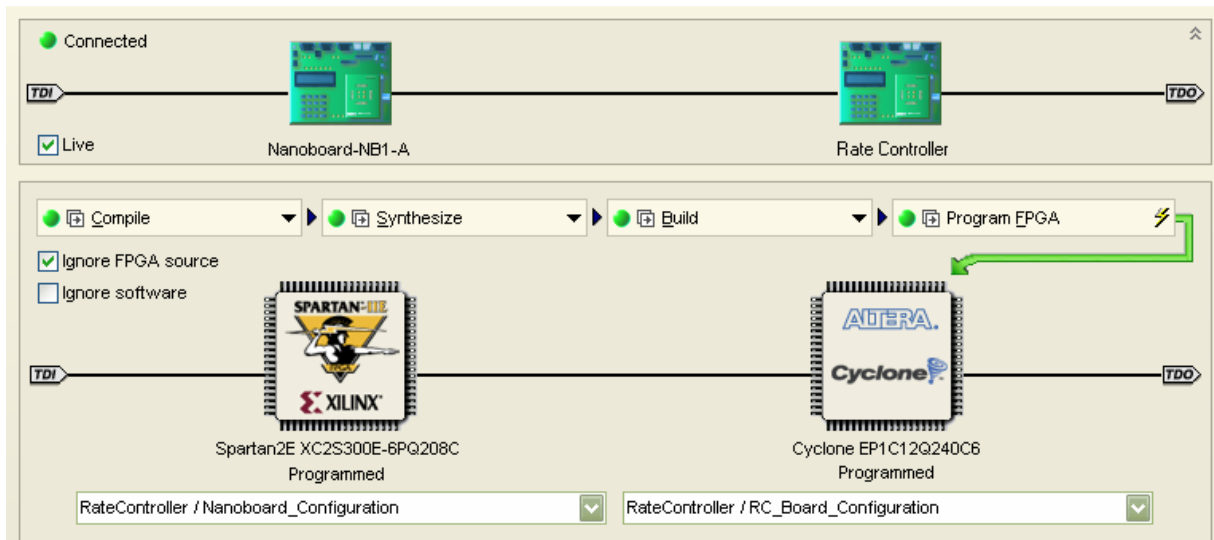



Figure 6. The one FPGA project (RateController), targeted to different FPGAs, on two different boards.

For the Configuration to be valid, it must include:


- at least one constraint file
- a constraint group in the file specifying the same device as the one available on the target board.

When the **Live** option is enabled the system searches for a physical device that matches the device specified in the project configuration and assigns that project to that FPGA/board. If there are no valid configurations available in the project, you will not be able to select an FPGA project/configuration in the drop-down list.

 For information on using the **Devices** view to process the design, see the [Processing the Captured FPGA Design](#) application note.

## Synchronization of the FPGA Design to a PCB

Since the FPGA design is target-independent, the synchronization process actually happens between the constraint file and the PCB project, rather than the actual design captured in the FPGA project source files.

 For information on FPGA project to PCB project synchronization, see the [Linking an FPGA Project to a PCB Project](#) application note.

## Revision History

Date	Version No.	Revision
15-Jan-04	1.0	New product release
06-Jul-2005	1.1	Updated for Altium Designer SP4
03-Mar-2008	2.0	Updated for Altium Designer Summer 08

Software, hardware, documentation and related materials:

Copyright © 2008 Altium Limited.

All rights reserved. You are permitted to print this document provided that (1) the use of such is for personal use only and will not be copied or posted on any network computer or broadcast in any media, and (2) no modifications of the document is made. Unauthorized duplication, in whole or part, of this document by any means, mechanical or electronic, including translation into another language, except for brief excerpts in published reviews, is prohibited without the express written permission of Altium Limited. Unauthorized duplication of this work may also be prohibited by local statute. Violators may be subject to both criminal and civil penalties, including fines and/or imprisonment. Altium, Altium Designer, Board Insight, Design Explorer, DXP, LiveDesign, NanoBoard, NanoTalk, 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.