HUNT ENGINEERING
Chestnut Court, Burton Row,
Brent Knoll, Somerset, TA9 4BP, UK
Tel: (+44) (0)1278 760188,
Fax: (+44) (0)1278 760199,
Email: sales@hunteng.co.uk
www.hunteng.co.uk
www.hunt-dsp.com

Using the VHDL Support for HERON Modules with Design Flows other than ISE (VHDL-XST)

Rev 1.3 R. Williams 09-05-05

Starting in February 2002, second generation support for HERON modules with FPGAs was introduced, with this support being entirely based around VHDL. VHDL is a portable language popularly used for logic synthesis. The main development tool sold by Xilinx for their FPGAs is ISE Foundation which is entirely VHDL based. All of the examples shipped by HUNT ENGINEERING are written for the ISE Foundation flow, using the Xilinx Synthesis Tool (XST) included in ISE.

There are, however, third party tools that can be used to synthesise VHDL designs, and as VHDL is a portable language this should present no problems to our users. There are also older versions of Xilinx tools that can be used if the correct procedure is followed.

This document describes how to approach each of these other flows, detailing the actions you need to take in order to use our standard examples.

History

Revision 1.0	19-02-02	First written
Revision 1.1	11-12-02	Changed text to remove references to ISE 4.1 as we now support ISE 5.x
Revision 1.2	02-04-03	Updated example scripts and projects for Leonardo Spectrum and Synplify.
Revision 1.3	09-05-05	Updated to reference 'Using Different ISE Versions' app note.

What We Are Trying To Do

We need to support users of our HERON modules in making their FPGA design. Most of our users will follow the “standard” ISE design flow of VHDL-XST. We provide full projects for those users in order to make it simple for them to re-create and modify our examples, then go on to design their own FPGA while still being able to use a simple VHDL interface to our hardware.

We aim to provide consistent support for all of our modules, so users that interface to hardware through our Hardware Interface Layer are not only protected from hardware details and concerns of timing etc, but also are helped if they want to migrate to a newer module type, or larger FPGA etc.

For users of design flows other than VHDL-XST, we provide a starting point through this document and example scripts and projects. These example files illustrate how the Example1 project from the HUNT ENGINEERING CD can be synthesized, placed, and routed under a variety of different flows. This document also details how to adapt the Example1 scripts and projects for use with other HUNT ENGINEERING CD examples, or for use with your own application.

The Basics of Synthesis

Whether you are using Xilinx XST, Leonardo Spectrum, Synplify from Synplicity or any other VHDL synthesis tool there are a common set of issues that must be considered when synthesizing.

The first consideration is to provide a list to the synthesizer of all the design source modules that make up a particular design. Typically, the order in which these source modules is listed is very important, as some parts of the design tree will require other parts to have been synthesized first in order for synthesis to complete successfully.

Whether you are creating your own synthesis project file or script from scratch, or whether you are using the supplied example files, the rules for source module ordering are the same.

The first group of source modules that must be listed are the modules that make up the Hardware Interface Layer. These modules are always located in the 'Common' directory for any particular board type. Please note, this group does NOT include the file 'top.vhd'.

The second group of source modules is the 'user' design source. If you are synthesizing one of the standard examples, or a project derived from one of these examples, then this source will be located in the 'Src' sub-directory.

The third group contains one source module 'top.vhd' (located in the 'Common' directory). This source module is the top level of the design and must always be listed as the last design source file.

Whether you are using Leonardo Spectrum or another synthesizer you can refer to the TCL script provided for Example1 (the '.tcl' file in the 'Leo_Syn' sub-directory) as a point of reference for how to correctly list all source modules for a particular board type.

With the design source correctly listed, the next concern is the application of timing constraints and pin constraints. The application of constraints takes two forms. Firstly, the synthesizer will work to a particular set of timing constraints. These constraints must be specified in the synthesis project or script you are using and are fairly simple in that a single operating frequency is specified. Secondly, the Place-and-Route tool must be given a set of timing constraints AND pin constraints that are correct for a particular FPGA module type.

The Place-and-Route tools must be set up to use the UCF file supplied with the example projects. Please note however, that for several tool flows such as Leonardo Spectrum and Synplify, the UCF file will need to be edited to reflect differences in the way signal names are generated. Please refer to the section on Place and Route Tools for a description of how to handle the differences in signal naming.

The last consideration is that all EDIF netlists required by the design can be found when the Place-and-Route tool is used to build the bitstream. This means that the working directory for the Place-and-Route tool must contain the EDIF netlist generated as output by the synthesizer, along with any EDIFs for black-box components such as CoreGenerator generated modules.

The following two sections describe how users of Leonardo Spectrum and Synplify can use the example scripts and projects provided. For users of any other synthesis tool, you will need to create your own projects, but in doing so must follow the guidelines given in this section.

Synthesis with Leonardo Spectrum

If you are not familiar with the synthesis tool and with the VHDL language, then you need to acquire some proficiency first before attacking complex high-speed designs. We assume in the following that this is not the case, and that you are familiar with VHDL projects and your synthesizer.

For users of Leonardo Spectrum a TCL script is provided with Example1. This TCL script should be used as a starting point for synthesis of all projects under Leonardo Spectrum. For each application, this script may need to be modified to reflect differences between Example1 and that project. However, the majority of this script will be correct for all users of a particular FPGA module type.

The script supplied with Example1 can be found in the **Leo_Syn** sub-directory of example1.

It is recommended that all users first synthesize the Example1 project using the supplied script. By doing so this will provide a point of reference before modifying the script for use with another project. When you have successfully synthesized Example1, you can then modify the script to work with other supplied example projects or with your own project, as detailed in the following section.

To synthesize Example1 in Leonardo Spectrum, simply set the working directory to the 'Leo_Syn' directory of Example1. Then run the supplied TCL script using the menu option '`File -> Run Script`…' and direct the tool to use the TCL file in the 'Leo_Syn' directory. Leonardo Spectrum will then begin synthesizing the Example1 design, creating a single EDIF netlist called 'Ex1_top.edf'.

Please note, getting many warning messages during synthesis is normal. The synthesis tools usually remove all logic that is defined but not really used, and warn you about the removed instances. *However, you should always review all the warning messages, understand each of them, and make sure they are acceptable in your specific case.*

The EDIF netlist created after synthesis can then be used in the ISE Alliance tools as the source for an EDIF flow project. We also provide an example project that can be used for placing-and-routing an EDIF netlist generated by Leonardo Spectrum. Please refer to the later section about Place and Route that details using the supplied example EDIF-flow ISE project.

As a final point when using Leonardo Spectrum, path names cannot use the backslash character, and must use the forward slash instead. Just be aware of this fact when editing the example files provided.

Adapting the Example Script

The example TCL script provided with Example1 performs a few simple steps necessary to synthesize Example1. This script can be easily adapted for use with other HUNT ENGINEERING example projects or your own project.

The first group of 'set' commands define basic timing information suitable for 100MHz designs. If you need to synthesize for a different system clock frequency then simply change these time constraint lines to match your new requirements.

The next group of commands set-up the Xilinx part type that the project is being targeted at. Please check that the part type defined here matches the Xilinx FPGA you have purchased.

The next command is the 'read' command. This command is very important in that it defines the entire design source that comprises a particular project. There are three parts to the source list. The first group lists all of the component modules from the Hardware Interface Layer required for that FPGA module type. These are the lines that start with '`../../Common`' (except the line that lists '`Top.vhd`'). The second group ('`../Src`') lists all of the modules from the Src directory that make up the user-application part of the design. You will need to add or remove the files listed here according to the differences between the project you are using and Example1. The third group is the line listing '`Top.vhd`'. This reference must always be the last line of the read command.

The final part of the script sets the optimisation process and names the output file to be created.

Synthesis with Synplify

If you are not familiar with the synthesis tool and with the VHDL language, then you need to acquire some proficiency first before attacking complex high-speed designs. We assume in the following that this is not the case, and that you are familiar with VHDL projects and your synthesizer.

For users of Synplify an example project is provided with Example1. This example project should be used as a starting point for synthesis of all projects under Synplify. For each application, this project may need to be modified to reflect differences between Example1 and that project. However, the majority of this project will be correct for all users of a particular FPGA module type.

The project supplied with Example1 can be found in the **Leo_Syn** sub-directory of example1.

It is recommended that all users first synthesize Example1 using the supplied project. By doing so this will provide a point of reference before modifying the project for use with another application. When you have successfully synthesized Example1, you can then modify the project file to work with other supplied example projects or with your own project, as detailed in the following section.

However, please be careful **not to save the project under the original name!** Since the `prj` file is our script and it would not be wise to overwrite it. We suggest that you edit manually the '`.prj`' file. If you want to save the project as such from the GUI, then save it under a **different name**.

To synthesize Example1 in Synplify, simply open the example project and click on the RUN button. Synplify will then begin synthesizing the Example1 design, creating a single EDIF netlist called 'Ex1_top.edf'.

Please note, getting many warning messages during synthesis is normal. The synthesis tools usually remove all logic that is defined but not really used, and warn you about the removed instances. *However, you should always review all the warning messages, understand each of them, and make sure they are acceptable in your specific case.*

The EDIF netlist created after synthesis can then be used in the ISE Alliance tools as the source for an EDIF flow project. We also provide an example project that can be used for placing-and-routing an EDIF netlist generated by Synplify. Please refer to the later section about Place and Route that details using the supplied example EDIF-flow ISE project.

As a final point when using Synplify, path names cannot use the backslash character, and must use the forward slash instead. Just be aware of this fact when editing the example files provided.

Adapting the Example Project

The example Synplify project provided with Example1 performs a few simple steps necessary to synthesize Example1. This project file can be easily adapted for use with other HUNT ENGINEERING example projects or your own project.

You are free to modify the project file as required by your project, but when doing so you must remember to both correctly define the part type and the design source as required by your application.

For each new project you make, please ensure that the device options list the correct Xilinx FPGA type that is fitted to the module your FPGA module. Also, please ensure that all of the design source modules are correctly listed at the top of the project file.

There are three parts to the source module list. The first part lists all of the component modules from the Hardware Interface Layer required for that FPGA module type. These are the lines that start with '`../../Common`' (except the line that lists '`Top.vhd`'). The second part ('`../Src`') lists all of the modules from the Src directory that make up the user-application part of the design. You will need to add or remove the files listed here according to the differences between the project you are using and Example1. The third part is the line listing '`Top.vhd`'. This reference must always be the last line of source module 'add_file' commands.

Place-and-Route for Third Party Synthesis Tools like Leonardo Spectrum & Synplify

When you have successfully synthesized your design, you will have an EDIF netlist (such as ‘Ex1_top.edf’ for the Example1 project) that contains the synthesized design. This netlist must then be passed through the Xilinx Place-and-Route tool from the ISE Alliance tools, in order to generate a bit-stream that can be downloaded to your FPGA.

As part of the Example1 for each FPGA module type, there is an EDIF-flow ISE project that can be used to build the netlist generated by your Synthesizer.

To use this project first start ISE and then open the ISE project file located in the Leo_Syn directory of Example1 for your module type.

Before building the design you will first need to check that the UCF file has been set up correctly. For users of Leonardo Spectrum and Synplify the UCF file will need to be modified to take account of differences in signal naming between ISE and those tools.

The ISE EDIF-flow project requires that the User Constraints File exists in the same directory as the ISE project. Therefore, you will first need to copy the supplied UCF file from the ISE sub-directory to the Leo_Syn sub-directory.

After you have copied the UCF file, you must then edit the file to take account of the different way that bus elements are referenced.

The XST synthesis tool translates bus elements using angle (<>) brackets, and both Leonardo Spectrum and Synplify translate them using round (( )) brackets. This means that you must edit the ‘.ucf’ file to replace angle brackets with round brackets. Do this carefully because if you change any of the pin location constraints you can stop the design from working. In extreme cases you can damage the hardware.

So take the .ucf file, (be sure to select the correct file for the FPGA option that you have on your module) and use your text editor to globally change all ‘<’ to ‘(‘ and then all ‘>’ to ‘)’ then save it.

You may also encounter some casing issues on pin names. Again, XST has a curious behaviour in that it transforms uppercase pin names into lower case… most of the time. To minimise the impact of this issue, we have created top level names using lower case. Casing issue should however be automatically resolved, the only inconvenience being irritating warning messages.

Creating Your Own ISE-EDIF Project

If you do want to create your own ISE-EDIF project from scratch, follow these guidelines:

1. Open ISE, create a new project, and select the Leo_Syn directory as project location.
2. Select the Device Family, device and Design Flow = EDIF.
3. Use Project > Add Source : select Ex1_top.edf (from the upper directory).
4. Edit > Preferences | Processes > Process settings > Property display level = advanced
5. Implement Properties > Translate Properties > Ignore LOC constraints on invalid object names > ON
6. Implement Properties > Translate Properties > Implementation User Constraints File : ../../../Common/xxx.UCF.
   Beware : make sure to have edited the UCF file as described in the synthesis with sections above.

Select the proper option(s) for BitGen (create ASCII configuration file)

Run the implementation.

When the place and Route has finished, you should check that some pins are properly placed (this ensures that the proper UCF file has been used). Compare the Pin report and the original UCF file.

Using Previous Versions of Xilinx ISE

The ready prepared example projects provided on the CD are written to use the latest version of ISE. If you are using a different version of ISE to the CD projects then you can work through the application note ‘Using Different Versions of ISE’.

Using Xilinx Foundation Classic

Although it is probably better to upgrade your Foundation Classic to the latest Foundation ISE, it is possible to have VHDL projects in Foundation Classic.

The VHDL Synthesis tool in Foundation Classic is FPGA Express, which has been removed from the Xilinx tools that are shipping now. This means it is probably not a good candidate for new designs.

The instructions below have been used on a particular version of Foundation Classic to successfully produce a bitstream for example1. While these guidelines are provided to help users of Foundation Classic, we cannot provide low level support for this tool flow.

Creating the Project

Create the following directory structure:

1. `md .\Common`, and copy the files located under \Common from the examples for your module type. This directory must contain the VHDL source files for the hardware interface layer, the constraint file (extension .ucf), the simulation models (sim_*.vhd), and the template for your design.

2. `md .\example1`, and copy the files located under \example1 from the examples for your module type.
This directory must contain the VHDL source files for the example.

3. Open Project Manager and choose “Create a New Project”.

4. Select the directories that you just copied the files to using browse, and give your project a name.

5. Select the flow to be HDL.

6. `Use Project > Add source`

The order that you add files to the project is VERY important.

First add the user_ap file from the \src directory

Then add the other files from this directory (except the tb_ file which is a testbench).

Then from the common directory add all of the files except top.vhd and those beginning with SIM_ and the xxx_tpl file.

Last of all add the top.vhd file from the common directory.

7. Click on synthesis and select “top” for the top level from the drop down menu. Choose a version name, and select the correct device type, package and speed (to match your module type).

8. Click Run to perform the synthesis. Some warnings are normal but check them carefully.

Now Click on Implementation, then click options next to the revision number. Select “edit options” next to configuration. Tick the “Produce ASCII configuration file” option, and if you are using the Virtex-II , you also need to uncheck the Power Down Status pin, and DCM shut down options.

9. In ISE 4.1 or later you click on the "`Translate properties`" tab, check (activate) the option named "`Ignore LOC constraints on invalid Object Names`". If this option is omitted, fatal errors will happen on unused logic (which is removed by the synthesis but remains specified in the constraint file). Under Foundation Classic this option is not available, so you need to edit the .ucf file (very carefully) to remove the pins that are not in the final design. We recommend that actually they are simply commented

out in case you need them later. Do this carefully because if you change any of the pin allocations you can stop the example working. In extreme cases you can damage the hardware.

It will probably be necessary to run the tools, to get a list of pins that need to be deleted.

You need to select the ucf file that is used by the tools. To do this you need to be in the in the pop up window (that you should be in after completing the steps above) where you click the “set” button next to “Control files”. Then set “Use constraints from” to custom and browse to select the correct ucf file for your module type from the common directory. Validate this setting with OK, and finally “run” the implementation.

Beware that Foundation classic makes a copy of your User Constraints so if you want to edit them after selecting them, you have to be careful to edit the right file.