Chapter 2
Vivado Design Tools

Sudipto Chakraborty

The Vivado suite of design tools contain services that support all phases of FPGA designs—starting from design entry, simulation, synthesis, place and route, bitstream generation, debugging, and verification as well as the development of software targeted for these FPGAs.

You can interact with the Vivado environment in multiple ways. This includes a GUI-based interface for interactive users, as well as a command-line interface if you prefer to use batch mode. Vivado also supports a scripting interface with a rich set of Tcl commands. These multiple modes of interaction can also be combined in different ways to suit the exact needs of users. These are explained in detail below.

2.1 Project vs. Non-project Mode

There are two primary ways to invoke design flows in Vivado—using a project or a non-project mode. In the first case, you start by creating a project to manage all your design sources as well as output generated from executing design flows. When a project is created, Vivado creates a predetermined directory structure on disk, which contains folders for source files, your configurations, as well as output data. Once a project has been created, you can enter and leave the Vivado environment as needed, and each time you can start from where you left off, without having to start from scratch each time. The project-based environment supports the notion of *runs* which allow users to invoke design flows like synthesis and implementation. You are allowed to customize the design environment in multiple ways, and these configurations are also persisted in the project environment in the form of "metadata."

S. Chakraborty (✉)
Xilinx, Longmont, CO, USA
e-mail: sudipto@xilinx.com

© Springer International Publishing Switzerland 2017

S. Churiwala (ed.), *Designing with Xilinx® FPGAs*,
DOI 10.1007/978-3-319-42438-5_2

The directory structure created for a project is as follows:

```
<project>/
<project>.xpr              : the main project file in text format
<project>.srcs/            : directory for sources local to a project
<project>.ip_user_files/   : directory for user accessible IP files
<project>.runs/            : directory for output data from synth/impl
<project>.sim/             : directory for output data from simulation
<project>.hw/              : directory for hardware debug related data
<project>.cache/           : directory for locally cached data
<project>.ipdef/           : directory for local IP definitions
```

Not all of the above mentioned directories will always be created. For example, a Vivado project supports referring to design sources remotely from their original location or copying them locally inside the project directory structure, based on user preference. The <project>.srcs directory is only created if there are such local copies of source files present.

In the non-project mode, you interact more directly with the Vivado environment using lower level commands. This mode is called *non-project* because you do not directly create a project to get your design flows to complete. However, it is important to note that a project *object* does exist in this case also; it is created automatically to manage certain aspects of the design flows. This project object exists only in memory while your session is active and does not create the on-disk structure described above. Since there is no automatic persistence of data on disk, all data is maintained only in memory and available only during the current session. Hence, you need to make sure that all necessary output is generated before you exit the current non-project session of Vivado.

One interesting note here is that the project mode of Vivado is actually built on top of the non-project mode, as explained in Sect. 2.2.1.

2.2 GUI, Command Line, and Tcl

Vivado offers a fully interactive graphical user interface to allow you to more easily manage your design sources and go through all phases of the design flow. Vivado also supports doing all these operations in a non-GUI, command-line environment. The common connection between these two interfaces is the Tcl commands that drive Vivado. Almost all operations performed during the GUI mode end up issuing a Tcl command to the core Vivado *engine*. These commands are shown in the Tcl console in the GUI and are also captured in a journal file, which is typically located where Vivado was started from, and the file is named *vivado.jou*. When working in command-line mode, these Tcl commands can be issued directly without needing the presence of a GUI.

2.2.1 Interaction with Project/Non-Project

While it is common for GUI-based users to typically use the project mode, it is also possible to execute the flows in non-project mode while being in the GUI. Similarly, command-line users can choose to use either project mode or non-project mode.

The Tcl commands supported for project mode are higher level, macro style commands which perform many functionalities under a single command. The Tcl commands for the non-project mode, on the other hand, are more granular WYSIWYG (what you see is what you get) type of commands which only perform the specified operation, no more no less. Some project mode commands actually use many non-project commands internally to perform the desired operation. This explains the comment in Sect. 2.1 that project mode in Vivado is actually built on top of the non-project mode.

Scripts 1 and 2 are example scripts for project mode and non-project mode, which both perform the same operation, but the non-project script is more verbose since it uses more granular commands.

```
Script 1: Project mode example Tcl script
   create_project project_1
   add_files top.v child.v
   launch_runs -to_step write_bitstream impl_1
   close_project

Script 2: Non-Project Mode Tcl Script
   read_verilog top.v
   read_verilog child.v
   synth_design -top top
   opt_design
   place_design
   route_design
   report_timing_summary
   write_checkpoint top_routed.dcp
   write_bitstream top.bit
```

2.2.2 Runs Infrastructure

In the Script 1 and Script 2 examples, the *launch_runs* command is a macro command which is part of the Vivado *runs infrastructure*. This command internally creates a Tcl script which looks similar to the non-project example Script 2 and automatically launches this script with a new Vivado session to execute the flow.

Runs infrastructure allows managing the output products from design flow automatically. It also maintains status of the flow execution, such that if a design source file changes, it automatically lets you know that the previously generated output product is now *out-of-date* and if you relaunch the end step of a run, it automatically determines which previous steps need to be performed first and executes them automatically.

The *runs* infrastructure also allows parallel execution of independent portions of the design flows to complete the overall flow quicker. These parallel runs can be executed on multiple processors in the same host machine, or if a compute farm like LSF or GRID is available, the multiple runs can be executed on different host machines in the compute farm.

2.3 Overview of Vivado GUI

This section provides a high level overview of the Vivado GUI and some recommendation for first-time users. Vivado is designed based on a concept of layered complexity. This means using the tool for common tasks and designs is made as automated and easy as possible without having to have detailed knowledge of the tool. However, once you get more familiarized with the tool and want to use advanced features to control your design flows in a customized manner, Vivado allows you with higher control with finer granularity.

Vivado GUI and project-based mode is highly recommended for first-time users or those who want to get quickly up and running. Using the GUI makes it easy to use the various wizards (like *New Project* wizard) to get started. First-time users can leave all settings at default and let the tool decide best automatic options. There are several example projects included with Vivado which you can readily open and use to try out the design flows. If you want to try your own design, the only two minimum required pieces of input are an HDL file to describe the design and a constraint file to specify the timing intent and pin mapping of the in/out signals to specific FPGA pins.

Figure 2.1 shows the screenshot of the Vivado GUI with some of the key areas highlighted:

1. This area is called the *Flow Navigator*. It provides easy, single click access to the common design flow steps and configuration options.
2. This area shows the sources in the design. The first tab here shows a graphical view of the sources with modules and instance relationships. The other tabs in this area show other key aspects of design sources.
3. This area shows the properties of the items selected in the GUI.
4. This area shows the Tcl console in the GUI as well as various reports and design run related details.
5. This area shows the built-in text editor, information related to project summary, etc.
6. This is a view of a design open in the GUI, which is key to all the design implementation steps.

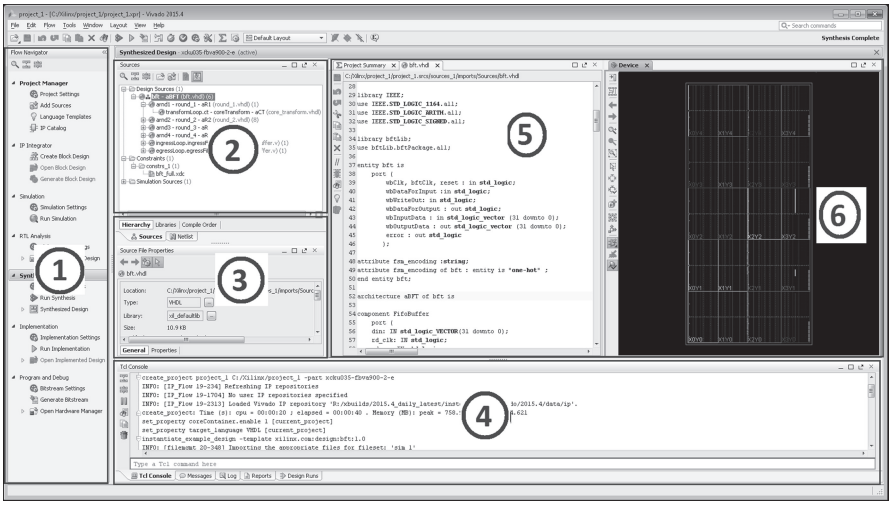

Fig. 2.1 Overall organization of Vivado GUI

Starting in the GUI and following the wizards make it easy to get started with the Vivado design flow. At the same time, as the various operations are being performed in the GUI, Vivado generates equivalent Tcl commands for those operations in the Tcl console area, as well as in the journal file as mentioned in Sect. 2.2. Using these Tcl commands, you can later customize the flow or build other similar flows.

http://www.springer.com/978-3-319-42437-8

Designing with Xilinx® FPGAs
Using Vivado
Churiwala, S. (Ed.)
2017, X, 260 p. 141 illus., 3 illus. in color., Hardcover
ISBN: 978-3-319-42437-8