

Vivado Design Suite User Guide

Design Flows Overview

UG892 (v2020.1) July 8, 2020

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
07/08/2020 Version 2020.1	
General updates.	Editorial updates only. No technical content updates.

Table of Contents

Revision History	2
Chapter 1: Vivado System-Level Design Flows	
Overview	5
Industry Standards-Based Design	6
Design Flows	7
Chapter 2: Understanding Use Models	
Vivado Design Suite Use Models	14
Working with the Vivado Integrated Design Environment (IDE)	15
Working with Tcl	17
Understanding Project Mode and Non-Project Mode	18
Using Third-Party Design Software Tools	23
Interfacing with PCB Designers	23
Chapter 3: Using Project Mode	
Overview	25
Project Mode Advantages	27
Creating Projects	27
Understanding the Flow Navigator	30
Performing System-Level Design Entry	32
Working with IP	35
Creating IP Subsystems with IP Integrator	42
Logic Simulation	45
Running Logic Synthesis and Implementation	51
Viewing Log Files, Messages, Reports, and Properties	55
Opening Designs to Perform Design Analysis and Constraints Definition	58
Device Programming, Hardware Verification, and Debugging	69
Using Project Mode Tcl Commands	70
Chapter 4: Using Non-Project Mode	
Overview	73
Non-Project Mode Advantages	74

Reading Design Sources	75
Working with IP and IP Subsystems	76
Running Logic Simulation	77
Running Logic Synthesis and Implementation	77
Generating Reports	77
Using Design Checkpoints	78
Performing Design Analysis Using the Vivado IDE	78
Using Non-Project Mode Tcl Commands	80

Chapter 5: Source Management and Revision Control Recommendations

Interfacing with Revision Control Systems	83
Upgrading Designs and IP to the Latest Vivado Design Suite Release	101

Appendix A: Additional Resources and Legal Notices

Xilinx Resources	103
Solution Centers	103
Documentation Navigator and Design Hubs	103
References	104
Training Resources	105
Please Read: Important Legal Notices	105

Vivado System-Level Design Flows

Overview

This user guide provides an overview of working with the Vivado® Design Suite to create a new design for programming into a Xilinx® device. It provides a brief description of various use models, design features, and tool options, including preparing, implementing, and managing the design sources and intellectual property (IP) cores.

The Vivado Design Suite offers multiple ways to accomplish the tasks involved in Xilinx device design, implementation, and verification. You can use the traditional register transfer level (RTL)-to-bitstream FPGA design flow, as described in [RTL-to-Bitstream Design Flow](#). You can also use system-level integration flows that focus on intellectual property (IP)-centric design and C-based design, as described in [Alternate RTL-to-Bitstream Design Flows](#).

Design analysis and verification is enabled at each stage of the flow. Design analysis features include logic simulation, I/O and clock planning, power analysis, constraint definition and timing analysis, design rule checks (DRC), visualization of design logic, analysis and modification of implementation results, programming, and debugging.

The following documents and QuickTake videos provide additional information about Vivado Design Suite flows:

- [Vivado Design Suite QuickTake Video: Vivado Design Flows Overview](#)
- *Vivado Design Suite Tutorial: Design Flows Overview* (UG888) [\[Ref 10\]](#)
- [Vivado Design Suite QuickTake Video: Getting Started with the Vivado IDE](#)
- [Xilinx Video Training: UltraFast Vivado Design Methodology](#)

The entire solution is integrated within a graphical user interface (GUI) known as the Vivado Integrated Design Environment (IDE). The Vivado IDE provides an interface to assemble, implement, and validate the design and the IP. In addition, all flows can be run using Tcl commands. Tcl commands can be scripted or entered interactively using the Vivado Design Suite Tcl shell or using the Tcl Console in the Vivado IDE. You can use Tcl scripts to run the entire design flow, including design analysis, or to run only parts of the flow.

Industry Standards-Based Design

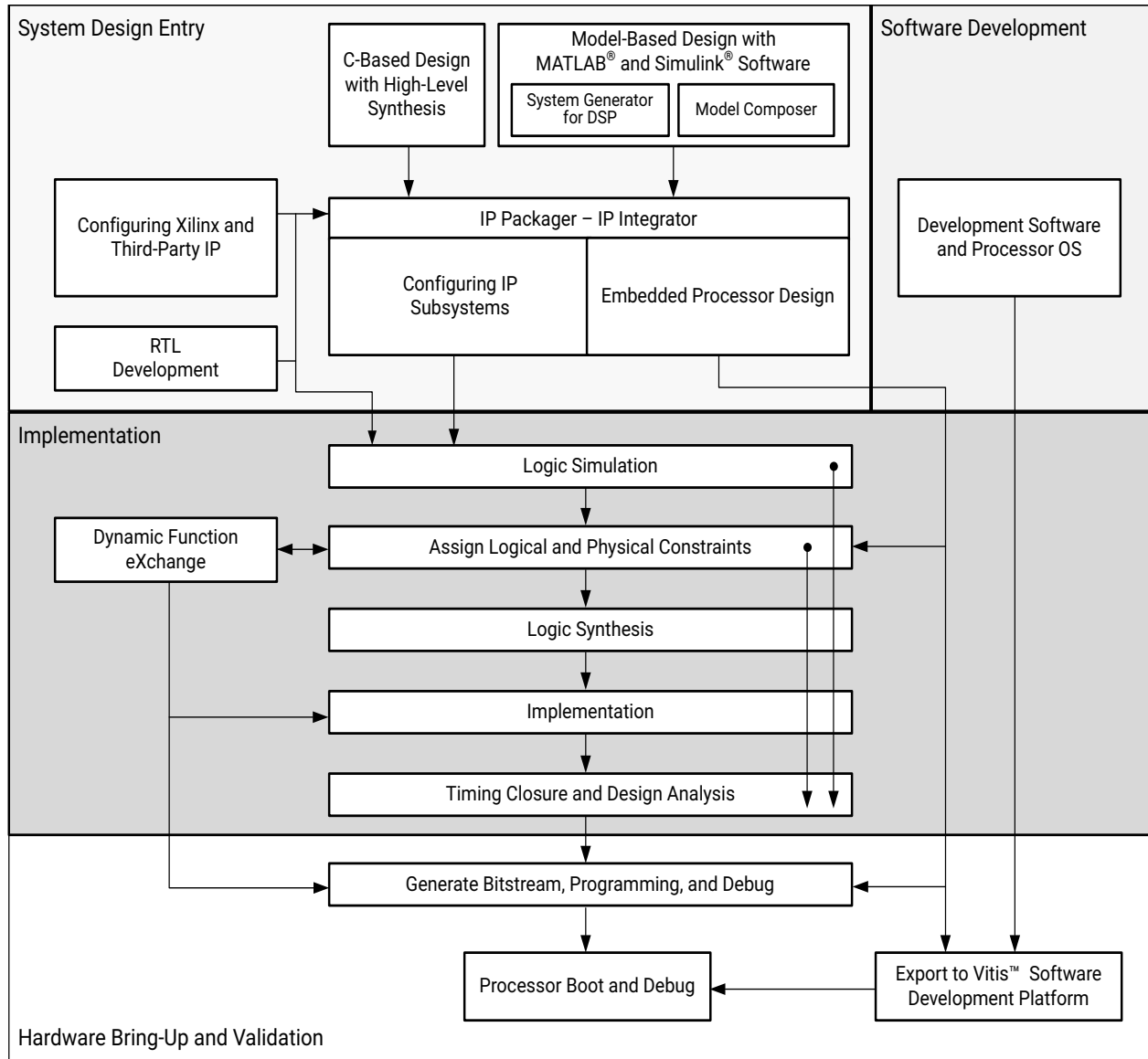
The Vivado Design Suite supports the following established industry design standards:

- Tcl
- AXI4, IP-XACT
- Synopsys design constraints (SDC)
- Verilog, VHDL, VHDL-2008, SystemVerilog
- SystemC, C, C++

The Vivado Design Suite solution is native Tcl based with support for SDC and Xilinx design constraints (XDC) formats. Extensive Verilog, VHDL, and SystemVerilog support for synthesis enables easier FPGA adoption. Vivado High-Level Synthesis (HLS) enables the use of native C, C++, or SystemC languages to define logic. Using standard IP interconnect protocol, such as AXI4 and IP-XACT, enables faster and easier system-level design integration. Support for these industry standards also enables the electronic design automation (EDA) ecosystem to better support the Vivado Design Suite. In addition, many new third-party tools are integrated with the Vivado Design Suite.

Design Flows

Figure 1-1 shows the high-level design flow in the Vivado Design Suite. Xilinx Design Hubs provide links to documentation organized by design tasks and other topics. On the Xilinx website, see the [Design Hubs](#) page.



X15150-070120

Figure 1-1: Vivado Design Suite High-Level Design Flow

RTL-to-Bitstream Design Flow

RTL Design

You can specify RTL source files to create a project and use these sources for RTL code development, analysis, synthesis and implementation. Xilinx supplies a library of recommended RTL and constraint templates to ensure RTL and XDC are formed optimally for use with the Vivado Design Suite. Vivado synthesis and implementation support multiple source file types, including Verilog, VHDL, SystemVerilog, and XDC. For information on creating and working with an RTL project, see this [link](#) in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 11].

The *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949) [Ref 28] focuses on proper coding and design techniques for defining hierarchical RTL sources and Xilinx design constraints (XDC), as well as providing information on using specific features of the Vivado Design Suite, and techniques for performance improvement of the programmed design.

IP Design and System-Level Design Integration

The Vivado Design Suite provides an environment to configure, implement, verify, and integrate IP as a standalone module or within the context of the system-level design. IP can include logic, embedded processors, digital signal processing (DSP) modules, or C-based DSP algorithm designs. Custom IP is packaged following IP-XACT protocol and then made available through the Vivado IP catalog. The IP catalog provides quick access to the IP for configuration, instantiation, and validation of IP. Xilinx IP utilizes the AXI4 interconnect standard to enable faster system-level integration. Existing IP can be used in the design either in RTL or netlist format. For more information, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 12].

IP Subsystem Design

The Vivado IP Integrator environment enables you to stitch together various IP into IP subsystems using the AMBA® AXI4 interconnect protocol. You can interactively configure and connect IP using a block design style interface and easily connect entire interfaces by drawing DRC-correct connections similar to a schematic. Connecting the IP using standard interfaces saves time over traditional RTL-based connectivity. Connection automation is provided as well as a set of DRCs to ensure proper IP configuration and connectivity. These IP block designs are then validated, packaged, and treated as a single design source. Block designs can be used in a design project or shared among other projects. The IP Integrator environment is the main interface for embedded design and the Xilinx evaluation board interface. For more information, see the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 30].

I/O and Clock Planning

The Vivado IDE provides an I/O pin planning environment that enables I/O port assignment either onto specific device package pins or onto internal die pads, and provides tables to let you design and analyze package and I/O-related data. Memory interfaces can be assigned interactively into specific I/O banks for optimal data flow. You can analyze the device and design-related I/O data using the views and tables available in the Vivado pin planner. The tool also provides I/O DRC and simultaneous switching noise (SSN) analysis commands to validate your I/O assignments. For more information, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 15].

Xilinx Platform Board Support

In the Vivado Design Suite, you can select an existing Xilinx evaluation platform board as a target for your design. In the platform board flow, all of the IP interfaces implemented on the target board are exposed to enable quick selection and configuration of the IP used in your design. The resulting IP configuration parameters and physical board constraints, such as I/O standard and package pin constraints, are automatically assigned and proliferated throughout the flow. Connection automation enables quick connections to the selected IP. For more information see this [link](#) in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 11].

Synthesis

Vivado synthesis performs a global, or top-down synthesis of the overall RTL design. However, by default, the Vivado Design Suite uses an out-of-context (OOC), or bottom-up design flow to synthesize IP cores from the Xilinx IP Catalog and block designs from the Vivado IP integrator. You can also choose to synthesize specific modules of a hierarchical RTL design as OOC modules. This OOC flow lets you synthesize, implement, and analyze design modules of a hierarchical design, IP cores, or block designs, out of the context of, or independent from the top-level design. The OOC synthesized netlist is stored and used during top-level implementation to preserve results and reduce runtime. The OOC flow is an efficient technique for supporting hierarchical team design, synthesizing and implementing IP and IP subsystems, and managing modules of large complex designs. For more information on the out-of-context design flow, see [Out-of-Context Design Flow](#).

The Vivado Design Suite also supports the use of third-party synthesized netlists, including EDIF or structural Verilog. However, IP cores from the Vivado IP Catalog must be synthesized using Vivado synthesis, and are not supported for synthesis with a third-party synthesis tool. There are a few exceptions to this requirement, such as the memory IP for 7 series devices. Refer to the data sheet for a specific IP for more information.



IMPORTANT: *The ISE Netlist format (NGC) is supported for 7 series devices. It is not supported for UltraScale™ and later devices.*

Design Analysis and Simulation

The Vivado Design Suite lets you analyze, verify, and modify the design at each stage of the design process. You can run design rule and design methodology checks, logic simulation, timing and power analysis to improve circuit performance. This analysis can be run after RTL elaboration, synthesis, and implementation. For more information, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 21].

The Vivado simulator enables you to run behavioral and structural logic simulation of the design at different stages of the design flow. The simulator supports Verilog and VHDL mixed-mode simulation, and results can be displayed in a waveform viewer integrated in the Vivado IDE. You can also use third-party simulators that can be integrated into and launched from the Vivado IDE. Refer to [Running Logic Simulation](#) for more information.

Placement and Routing

When the synthesized netlist is available, Vivado implementation provides all the features necessary to optimize, place and route the netlist onto the available device resources of the target part. Vivado implementation works to satisfy the logical, physical, and timing constraints of the design.

For challenging designs the Vivado IDE also provides advanced floorplanning capabilities to help drive improved implementation results. These include the ability to constrain specific logic into a particular area, or manually placing specific design elements and fixing them for subsequent implementation runs. For more information, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 21].

Hardware Debug and Validation

After implementation, the device can be programmed and then analyzed with the Vivado logic analyzer, or within the standalone Vivado Lab Edition environment. Debug signals can be identified in the RTL design, or inserted after synthesis and are processed throughout the flow. You can add debug cores to the RTL source files, to the synthesized netlist, or in an implemented design using the using the Engineering Change Order (ECO) flow. You can also modify the nets connected to a debug probe, or route internal signals to a package pin for external probing using the ECO flow. For more information, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 22].

Alternate RTL-to-Bitstream Design Flows

The Vivado Design Suite also supports several alternate design flows, as described in the following sections. Each of these flows is derived from the RTL-to-bitstream flow, so the implementation and analysis techniques described above also apply to these other design flows.

Accelerated Kernel Flows

The Xilinx® Vitis™ unified software platform introduces acceleration use cases into Vivado flows. In this design methodology, Vivado is used to create a platform which is consumed by the Vitis software platform to add accelerated kernels. The hardware design is comprised of the platform and the accelerators. In this case, the final bitstream is created by the Vitis software platform since the complete design is not visible in Vivado. For more information on platform creation, see *Vitis Unified Software Platform Documentation (UG1393)* [Ref 34].

Embedded Processor Design

A slightly different tool flow is needed when creating an embedded processor design. Because the embedded processor requires software in order to boot-up and run effectively, the software design flow must work in unison with the hardware design flow. Data hand-off between the hardware and software flows, and validation across these two domains is critical for success.

Creating an embedded processor hardware design involves the IP integrator of the Vivado Design Suite. In a Vivado IP integrator block design, you instantiate, configure, and assemble the processor core and its interfaces. The IP Integrator enforces rules-based connectivity and provides design assistance. After it is compiled through implementation, the hardware design is exported to the Vitis software platform for use in the software development and validation. Simulation and debug features allow you to simulate and validate the design across the two domains.

The Vitis software platform is Xilinx's unified software suite that includes compilers for all embedded applications and accelerated applications on Xilinx platforms. Vitis supports developing in higher level languages, leverages open source libraries, and supports domain specific development environments.

VIDEO: For training videos on the Vivado IP integrator and the embedded processor design flow, see the [Vivado Design Suite QuickTake Video: Targeting Zynq Devices Using Vivado IP Integrator](#).

The embedded processor design flow is described in the following resources:

- *Vivado Design Suite User Guide: Embedded Processor Hardware Design (UG898)* [Ref 14]
- *Vivado Design Suite Tutorial: Embedded Processor Hardware Design (UG940)* [Ref 26]
- *UltraFast Embedded Design Methodology Guide (UG1046)* [Ref 31]

Model-Based Design Using Model Composer

Model Composer is a model-based graphical design tool that enables rapid design exploration within the MathWorks MATLAB® and Simulink® products and accelerates the path to production for Xilinx devices through automatic code generation. For information, see the *Model Composer User Guide* (UG1262) [Ref 1].

Model-Based DSP Design Using Xilinx System Generator

The Xilinx System Generator tool, which is installed as part of the Vivado Design Suite, can be used for implementing DSP functions. You create the DSP functions using System Generator as a standalone tool, and then package your System Generator design into an IP module that can be included in the Vivado IP catalog. From there, the generated IP can be instantiated into your Vivado design as a submodule. For more information, see the *Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator* (UG897) [Ref 13].

High-Level Synthesis C-Based Design

The C-based High-Level Synthesis (HLS) tools within the Vivado Design Suite enable you to describe various DSP functions in the design using C, C++, and SystemC. You create and validate the C code with the Vivado HLS tools. Use of higher-level languages allows you to abstract algorithmic descriptions, data type, specification, etc. You can create “what-if” scenarios using various parameters to optimize design performance and device area.

HLS lets you simulate the generated RTL directly from its design environment using C-based test benches and simulation. C-to-RTL synthesis transforms the C-based design into an RTL module that can be packaged and implemented as part of a larger RTL design, or instantiated into an IP Integrator block design.



VIDEO: For various training videos on Vivado HLS, see the *Vivado High-Level Synthesis video tutorials* available from the [Vivado Design QuickTake Video Tutorials page](#) on the Xilinx website.

The HLS tool flow and features are described in the following resources:

- *Vivado Design Suite User Guide: High-Level Synthesis* (UG902) [Ref 2]
- *Vivado Design Suite Tutorial: High-Level Synthesis* (UG871) [Ref 7]

Partial Reconfiguration Design

Partial Reconfiguration allows portions of a running Xilinx device to be reconfigured in real-time with a partial bitstream, changing the features and functions of the running design. The reconfigurable modules must be properly planned to ensure they function as needed for maximum performance.

The Partial Reconfiguration flow requires a strict design process to ensure that the reconfigurable modules are designed properly in order to enable glitch-less operation

during partial bitstream updates. This includes reducing the number of interface signals into the reconfigurable module, floorplanning device resources, and pin placement; as well as adhering to special partial reconfiguration DRCs. The device programming method must also be properly planned to ensure the configuration I/O pins are assigned appropriately.



VIDEO: Information on the Partial Reconfiguration flow is available from the [Vivado Design Suite QuickTake Video: Partial Reconfiguration](#).

The Partial Reconfiguration tool flow and features are described in the following resources:

- *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909) [[Ref 23](#)]
- *Vivado Design Suite Tutorial: Partial Reconfiguration* (UG947) [[Ref 27](#)]

Hierarchical Design

Hierarchical Design (HD) flows enable you to partition a design into smaller, more manageable modules to be processed independently. The hierarchical design flow involves proper module interface design, constraint definition, floorplanning, and some special commands and design techniques. For more information, see the *Vivado Design Suite User Guide: Hierarchical Design* (UG905) [[Ref 20](#)].

Using a modular approach to the hierarchical design lets you analyze modules independent of the rest of the design, and reuse modules in the top-down design. A team of users can iterate on specific sections of a design, achieving timing closure and other design goals, and reuse the results.

There are several Vivado features that enable a hierarchical design approach, such as the synthesis of a logic module outside of the context (OOC) of the top-level design. You can select specific modules, or levels of the design hierarchy, and synthesize them OOC. Module-level constraints can be applied to optimize and validate module performance. The module design checkpoint (DCP) will then be applied during implementation to build the top-level netlist. This method can help reduce top-level synthesis run time, and eliminate re-synthesis of completed modules.

Understanding Use Models

Vivado Design Suite Use Models



RECOMMENDED: Before beginning your first design with the Vivado tools, review the information in the Vivado Design Suite User Guide: Getting Started (UG910) [Ref 24].

Just as the Vivado® Design Suite supports many different design flows, the tools support several different use models depending on how you want to manage your design and interact with the Vivado tools. This section will help guide you through some of the decisions that you must make about the use model you want to use for interacting with the Vivado tools.

Some of these decisions include:

- Are you a script or command-based user; or do you prefer working through a graphical user interface (GUI)? See [Working with the Vivado Integrated Design Environment \(IDE\)](#) and [Working with Tcl](#).
- Do you want the Vivado Design Suite to manage the design sources, status, and results by using a project structure; or would you prefer to quickly create and manage a design yourself? See [Understanding Project Mode and Non-Project Mode](#).
- Do you want to configure IP cores and contain them within a single design project for portability; or establish a remote repository of configured IP cores outside of the project for easier management across multiple projects?
- Are you managing your source files inside a revision control system? See [Interfacing with Revision Control Systems](#).
- Are you using third-party tools for synthesis or simulation? See [Using Third-Party Design Software Tools](#).

Working with the Vivado Integrated Design Environment (IDE)

The Vivado Integrated Design Environment (IDE) can be used in both [Project Mode](#) and [Non-Project Mode](#). The Vivado IDE provides an interface to assemble, implement, and validate your design and IP. Opening a design loads the current design netlist, applies design constraints, and fits the design onto the target device. The Vivado IDE allows you to visualize and interact with the design as shown in the following figure.

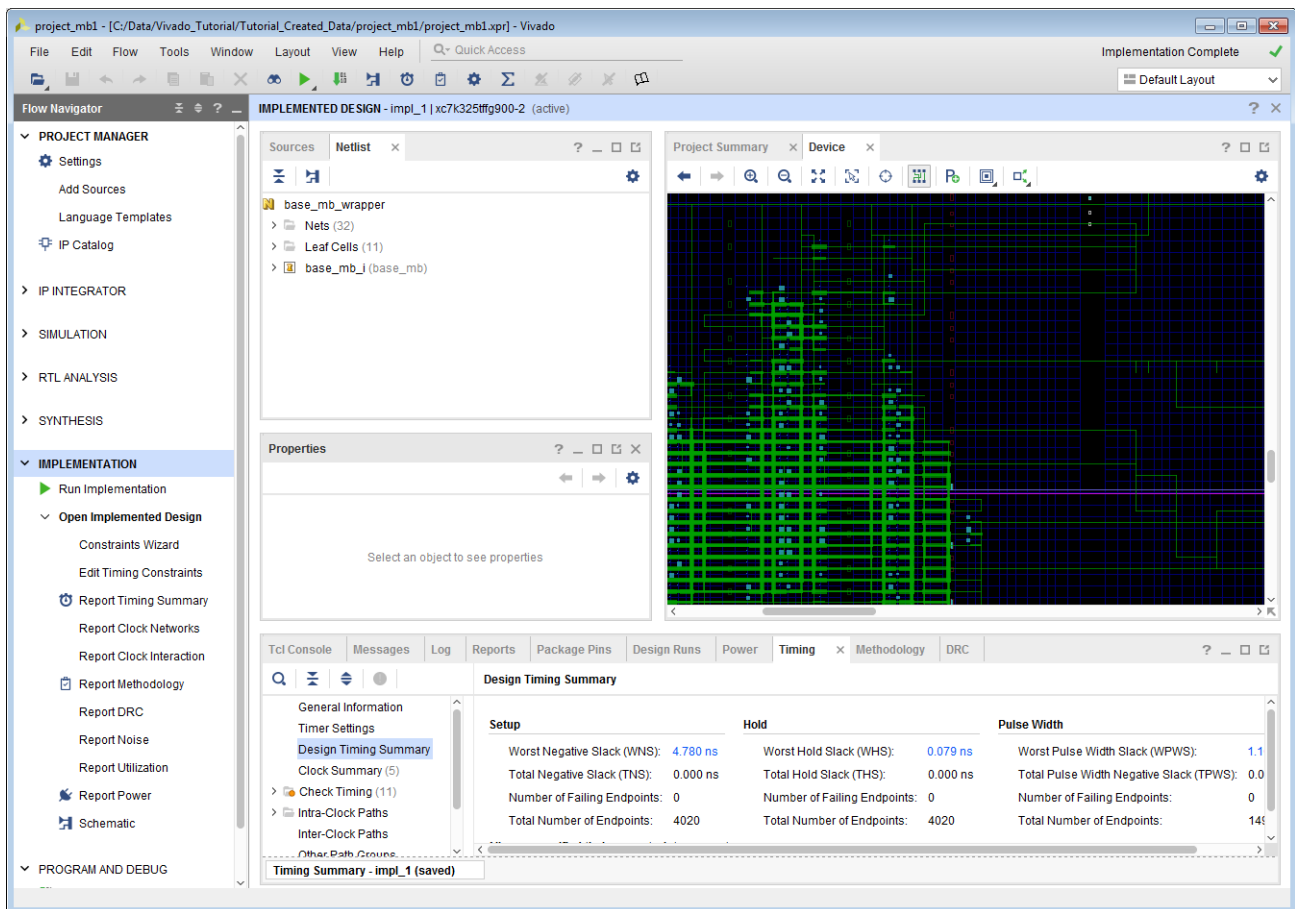


Figure 2-1: Opening the Implemented Design in the Vivado IDE

When using Project Mode, the Vivado IDE provides an interface called Flow Navigator, that supports a push-button design flow. You can open designs after RTL elaboration, synthesis, or implementation and analyze the design, make changes to constraints, logic or device configuration, and implementation results. You can also use design checkpoints to save the current state of any design. For more information on the Vivado IDE, see the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 9].



VIDEO: For more information, see the [Vivado Design QuickTake Video: Getting Started with the Vivado IDE](#).

Launching the Vivado IDE on Windows

Select **Start > All Programs > Xilinx Design Tools > Vivado <version> > Vivado <version>**.

Note: You can also double-click the Vivado IDE shortcut icon on your desktop.



Figure 2-2: Vivado IDE Desktop Icon



TIP: You can right-click the Vivado IDE shortcut icon, and select **Properties** to update the Start In field. This makes it easier to locate the project file, log files, and journal files, which are written to the launch directory.

Launching the Vivado IDE from the Command Line on Windows or Linux

Enter the following command at the command prompt:

```
vivado
```

When you enter this command, it automatically runs `vivado -mode gui` to launch the Vivado IDE. If you need help, type `vivado -help`.



TIP: To add the Vivado tools path to your current shell/command prompt, run `settings64.bat` or `settings64.sh` from the `<install_path>/Vivado/<version>` directory.

When launching the Vivado Design Suite from the command line, change directory to your project directory so that the Vivado tool will write its log and journal files to your project directory. This makes it easy to locate and review these files as needed.



RECOMMENDED: Launch the Vivado Design Suite from your project directory to make it easier to locate the project file, log files, and journal files, which are written to the launch directory.

Launching the Vivado IDE from the Vivado Design Suite Tcl Shell

When the Vivado Design Suite is running in Tcl mode, enter the following command at the Tcl command prompt to launch the Vivado IDE:

```
start_gui
```

Working with Tcl

All supported design flows and use models can be run using Tcl commands. You can use Tcl scripts to run the entire design flow, including design analysis and reporting, or to run parts of the design flow, such as design creation and synthesis. You can use either individual Tcl commands or saved scripts of Tcl commands.

If you prefer working directly with Tcl commands, you can interact with your design using a Vivado Design Suite Tcl shell, using the Tcl Console from within the Vivado IDE. For more information about using Tcl and Tcl scripting, see the *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894) [Ref 10] and *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 6]. For a step-by-step tutorial that shows how to use Tcl in the Vivado tools, see the *Vivado Design Suite Tutorial: Design Flows Overview* (UG888) [Ref 8].

For more information on using a Tcl-based approach using either the Project Mode or Non-Project Mode, see [Chapter 3, Using Project Mode](#) or [Chapter 4, Using Non-Project Mode](#).

Launching the Vivado Design Suite Tcl Shell

Use the following command to invoke the Vivado Design Suite Tcl Shell either at the Linux command prompt or within a Windows Command Prompt window:

```
vivado -mode tcl
```

Note: On Windows, you can also select **Start > All Programs > Xilinx Design Tools > Vivado <version> > Vivado <version> Tcl Shell**.

Launching the Vivado Tools Using a Batch Tcl Script

You can use the Vivado tools in batch mode by supplying a Tcl script when invoking the tool. Use the following command either at the Linux command prompt or within a Windows Command Prompt window:

```
vivado -mode batch -source <your_Tcl_script>
```

Note: When working in batch mode, the Vivado tools exit after running the specified script.

Using the Vivado IDE with a Tcl Flow

When working with Tcl, you can still take advantage of the interactive GUI-based analysis and constraint definition capabilities in the Vivado IDE. You can open designs in the Vivado IDE at any stage of the design cycle, as described in [Performing Design Analysis Using the Vivado IDE](#). You can also save the design database at any time as a checkpoint file, and open the checkpoint later as described in [Using Design Checkpoints](#).

Using the Xilinx Tcl Store

The Xilinx Tcl Store is an open source repository of Tcl code designed primarily for use in FPGA designs with the Vivado Design Suite. The Tcl Store provides access to multiple scripts and utilities contributed from different sources, which solve various issues and improve productivity. You can install Tcl scripts and also contribute Tcl scripts to share your expertise with others. For more information on working with Tcl scripts and the [Xilinx Tcl Store](#), see the *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894) [\[Ref 10\]](#).

Understanding Project Mode and Non-Project Mode

The Vivado Design Suite has two primary use models: Project Mode and Non-Project Mode. Both Project Mode and Non-Project Mode can be developed and used through either the Vivado IDE, or through Tcl commands and batch scripts. However, the Vivado IDE offers many benefits for the Project Mode, such as the Flow Navigator graphical workflow interface. Tcl commands are the simplest way to run the Non-Project Mode.

Project Mode

The Vivado Design Suite takes advantage of a project based architecture to assemble, implement, and track the state of a design. This is referred to as Project Mode. In Project Mode, Vivado tools automatically manage your design flow and design data.



TIP: *The key advantage of Project Mode is that the Vivado Design Suite manages the entire design process, including dependency management, report generation, data storage, etc.*

When working in Project Mode, the Vivado Design Suite creates a directory structure on disk in order to manage design source files, either locally or remotely, and manage changes and updates to the source files.



IMPORTANT: *Certain operating systems (for example, Microsoft Windows) restrict the number of characters (such as 256) that can be used for the file path and file name. If your operating system has such a limitation, Xilinx recommends that you create projects closer to the drive root to keep file paths and names as short as possible.*

The project infrastructure is also used to manage the automated synthesis and implementation runs, track run status, and store synthesis and implementation results and reports. For example:

- If you modify an HDL source after synthesis, the Vivado Design Suite identifies the current results as out-of-date, and prompts you for re-synthesis.
- If you modify design constraints, the Vivado tools prompt you to either re-synthesize, re-implement, or both.
- After routing is completed, the Vivado tool automatically generates timing, DRC, methodology, and power reports.
- The entire design flow can be run with a single click within the Vivado IDE.

For detailed information on working with projects, see [Chapter 3, Using Project Mode](#).

Non-Project Mode

Alternatively, you can choose an in-memory compilation flow in which you manage sources and the design process yourself, known as *Non-Project Mode*. In Non-Project Mode, you manage design sources and the design process yourself using Tcl commands or scripts. The key advantage is that you have full control over each step of the flow.

When working in Non-Project Mode, source files are read from their current locations, such as from a revision control system, and the design is compiled through the flow in memory. You can run each design step individually using Tcl commands. You can also use Tcl commands to set design parameters and implementation options.

You can save design checkpoints and create reports at any stage of the design process. Each implementation step can be tailored to meet specific design challenges, and you can analyze results after each design step. In addition, you can open the Vivado IDE at any point for design analysis and constraints assignment.

In Non-Project Mode, each design step is controlled using Tcl commands. For example:

- If you modify an HDL file after synthesis, you must remember to rerun synthesis to update the in-memory netlist.
- If you want a timing report after routing, you must explicitly generate the timing report when routing completes.
- Design parameters and implementation options are set using Tcl commands and parameters.
- You can save design checkpoints and create reports at any stage of the design process using Tcl.

As the design flow progresses, the representation of the design is retained in memory in the Vivado Design Suite. Non-Project Mode discards the in-memory design after each session and only writes data to disk that you instruct it to. For more information on Non-Project Mode, see [Chapter 4, Using Non-Project Mode](#).

Feature Differences

In Project Mode, the Vivado IDE tracks the history of the design and stores pertinent design information. However, because many features are automated, you have less control in the default flow. For example, only a standard set of report files is generated with each run. However, through Tcl commands or scripting, you have access to customize the flow and features of the tool in Project Mode.

The following automated features are only available when using Project Mode:

- Out-of-the-box design flow
- Easy-to-use, push-button interface
- Powerful Tcl scripting language for customization
- Source file management and status
- Automatically generated standard reports
- Storage and reuse of tool settings and design configuration
- Experimentation with multiple synthesis and implementation runs
- Run results management and status

Non-Project Mode, is more of a compilation methodology where you have complete control over every action executed through a Tcl command. This is a fully customizable

design flow suited to specific designers looking for control and batch processing. All of the processing is done in memory, so no files or reports are generated automatically. Each time you compile the design, you must define all of the sources, set all tool and design configuration parameters, launch all implementation commands, and generate report files. This can be accomplished using a Tcl run script, because a project is not created on disk, source files remain in their original locations and design output is only created when and where you specify. This method provides you with all of the power of Tcl commands and full control over the entire design process. Many users prefer this batch compilation style interaction with the tools and the design data.

Table 2-1 summarizes the feature differences between Project Mode and Non-Project Mode.

Table 2-1: Project Mode versus Non-Project Mode Features

Flow Element	Project Mode	Non-Project Mode
Design Source File Management	Automatic	Manual
Flow Navigation	Guided	Manual
Flow Customization	Unlimited with Tcl commands	Unlimited with Tcl commands
Reporting	Automatic	Manual
Analysis Stages	Designs and design checkpoints	Designs and design checkpoints

Command Differences

Tcl commands vary depending on the mode you use, and the resulting Tcl run scripts for each mode are different. In Non-Project Mode, all operations and tool settings require individual Tcl commands, including setting tool options, running implementation commands, generating reports, and writing design checkpoints. In Project Mode, wrapper commands are used around the individual synthesis, implementation, and reporting commands.

For example, in Project Mode, you add sources to the project for management using the `add_files` Tcl commands. Sources can be copied into the project to maintain a separate version within the project directory structure or can be referenced remotely. In Non-Project Mode, you use the `read_verilog`, `read_vhdl`, `read_xdc`, and `read_*` Tcl commands to read the various types of sources from their current location.

In Project Mode, the `launch_runs` command launches the tools with preconfigured run strategies and generates standard reports. This enables consolidation of implementation commands, standard reporting, use of run strategies, and run status tracking. However, you can also run custom Tcl commands before or after each step of the design process. Run results are automatically stored and managed within the project. In Non-Project Mode, individual commands must be run, such as `opt_design`, `place_design`, and `route_design`.

Many Tcl commands can be used in either mode, such as the reporting commands. In some cases, Tcl commands are specific to either Project Mode or Non-Project Mode. Commands that are specific to one mode must not be mixed when creating scripts. For example, if you are using the Project Mode you must not use base-level commands such as **synth_design**, because these are specific to Non-Project Mode. If you use Non-Project Mode commands in Project Mode, the database is not updated with status information and reports are not automatically generated.



TIP: Project Mode includes GUI operations, which result in a Tcl command being executed in most cases. The Tcl commands appear in the Vivado IDE Tcl Console and are also captured in the `vivado.jou` file. You can use this file to develop scripts for use with either mode.

Figure 2-3 shows the difference between Project Mode and Non-Project Mode Tcl commands.

Project Mode		Non-Project Mode
GUI	Tcl Script	Tcl Script
	<pre> create_project ... add_files ... import_files launch_run synth_1 wait_on_run synth_1 open_run synth_1 report_timing_summary launch_run impl_1 wait_on_run impl_1 open_run impl_1 report_timing_summary launch_run impl_1 -to_step_write_bitstream wait_on_run impl_1 </pre>	<pre> read_verilog ... read_vhdl ... read_ip ... read_xdc ... read_edif synth_design ... report_timing_summary write_checkpoint opt_design write_checkpoint place_design write_checkpoint route_design report_timing_summary write_checkpoint write_bitstream </pre>

Figure 2-3: Project Mode and Non-Project Mode Commands

Using Third-Party Design Software Tools

Xilinx has strategic partnerships with several third-party design tool suppliers. The following software solutions include synthesis and simulation tools only.

Running Logic Synthesis

The Xilinx FPGA logic synthesis tools supplied by Synopsys and Mentor Graphics are supported for use with the Vivado Design Suite. In the Vivado Design Suite, you can import the synthesized netlists in structural Verilog or EDIF format for use during implementation. In addition, you can use the constraints (SDC or XDC) output by the logic synthesis tools in the Vivado Design Suite.

All Xilinx IP and Block Designs use Vivado Synthesis. Use of third party synthesis for Xilinx IP or IP Integrator block designs is not supported, with a few exceptions, such as the memory IP for 7 series devices. Refer to the data sheet for a specific IP for more information.

Running Logic Simulation

Logic simulation tools supplied by Mentor Graphics, Cadence, Aldec, and Synopsys are integrated and can be launched directly from the Vivado IDE. Netlists can also be produced for all supported third-party logic simulators. From the Vivado Design Suite, you can export complete Verilog or VHDL netlists at any stage of the design flow for use with third-party simulators. In addition, you can export structural netlists with post-implementation delays in standard delay format (SDF) for use in third-party timing simulation. The Vivado Design Suite also generates simulation scripts for enterprise users. Using the scripts and compiled libraries, enterprise users can run the simulation without the Vivado Design Suite environment.



VIDEO: For more information, see the [Vivado Design Suite QuickTake Video: Simulating with Cadence IES in Vivado](#) and [Vivado Design Suite QuickTake Video: Simulating with Synopsys VCS in Vivado](#).

Note: Some Xilinx IP provides RTL sources in only Verilog or VHDL format. After synthesis, structural netlists can be created in either language.

Interfacing with PCB Designers

The I/O planning process is critical to high-performing systems. Printed circuit board (PCB) designers are often concerned about the relationship and orientation of the FPGA on the PCB. These large ball grid array (BGA) devices are often the most difficult routing challenge a PCB designer faces. Additional concerns include critical interface routing, location of

power rails, and signal integrity. A close collaboration between FPGA and PCB designers can help address these design challenges. The Vivado IDE enables the designer to visualize the relationship between the physical package pins and the internal die pads to optimize the system-level interconnect.

The Vivado Design Suite has several methods to pass design information between the FPGA, PCB, and system design domains. I/O pin configuration can be passed back and forth using a comma separated value (CSV) spreadsheet, RTL header, or XDC file. The CSV spreadsheet contains additional package and I/O information that can be used for a variety of PCB design tasks, such as matched length connections and power connections. An I/O Buffer Information Specification (IBIS) model can also be exported from the Vivado IDE for use in signal integrity analysis on the PCB.

For more information see:

- *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [\[Ref 15\]](#)
- [Vivado Design Suite QuickTake Video: I/O Planning Overview](#)
- [Vivado Design Hub: I/O and Clock Planning](#)

Using Project Mode

Overview

In Project Mode, the Vivado® Design Suite creates a project directory structure and automatically manages your source files, constraints, IP data, synthesis and implementation run results, and reports. In this mode, the Vivado Design Suite also manages and reports on the status of the source files, configuration, and the state of the design.

You can create RTL-based projects or synthesized, netlist-based projects. Netlist projects are primarily used with third-party synthesis tools, and the design process is managed from a post-synthesis perspective. You can analyze the netlist design, assign and manage constraints, implement and analyze the design, program and debug the device, and manage the sources and outputs for the entire flow.

In the Vivado IDE, you can use the Flow Navigator (Figure 3-1) to launch predefined design flow steps, such as synthesis and implementation. When you click **Generate Bitstream**, the Vivado IDE ensures that the design is synthesized and implemented with the most current design sources and generates a bitstream file. The environment provides an intuitive pushbutton design flow and also offers advanced design management and analysis features. Runs are launched with wrapper Tcl scripts that consolidate the various implementation commands and automatically generates standard reports. You can use various run strategies to address different design challenges, such as routing density and timing closure. You can also simultaneously launch multiple implementation runs to see which will achieve the best results.

Note: Run strategies only apply to Project Mode. In Non-Project Mode, all directives and command options must be set manually.

You can run Project Mode using the Vivado IDE or using Tcl commands or scripts. In addition, you can alternate between using the Vivado IDE and Tcl within a project. When you open or create projects in the Vivado IDE, you are presented with the current state of the design, run results, and previously generated reports and messages. You can create or modify sources, apply constraints and debug information, configure tool settings, and perform design tasks.



RECOMMENDED: *Project Mode is the easiest way to get acquainted with features of the Vivado tools and Xilinx® recommendations.*

Vivado has the unique capability to open the design at various stages of the design flow. You can open designs for analysis and constraints definition after RTL elaboration, synthesis, and implementation. When you open a design, the Vivado tools compile the netlist and constraints against the target device and show the design in the Vivado IDE. After you open the design, you can use a variety of analysis and reporting features to analyze the design using different criteria and viewpoints. You can also apply and save constraint and design changes. For more information, see *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 21].

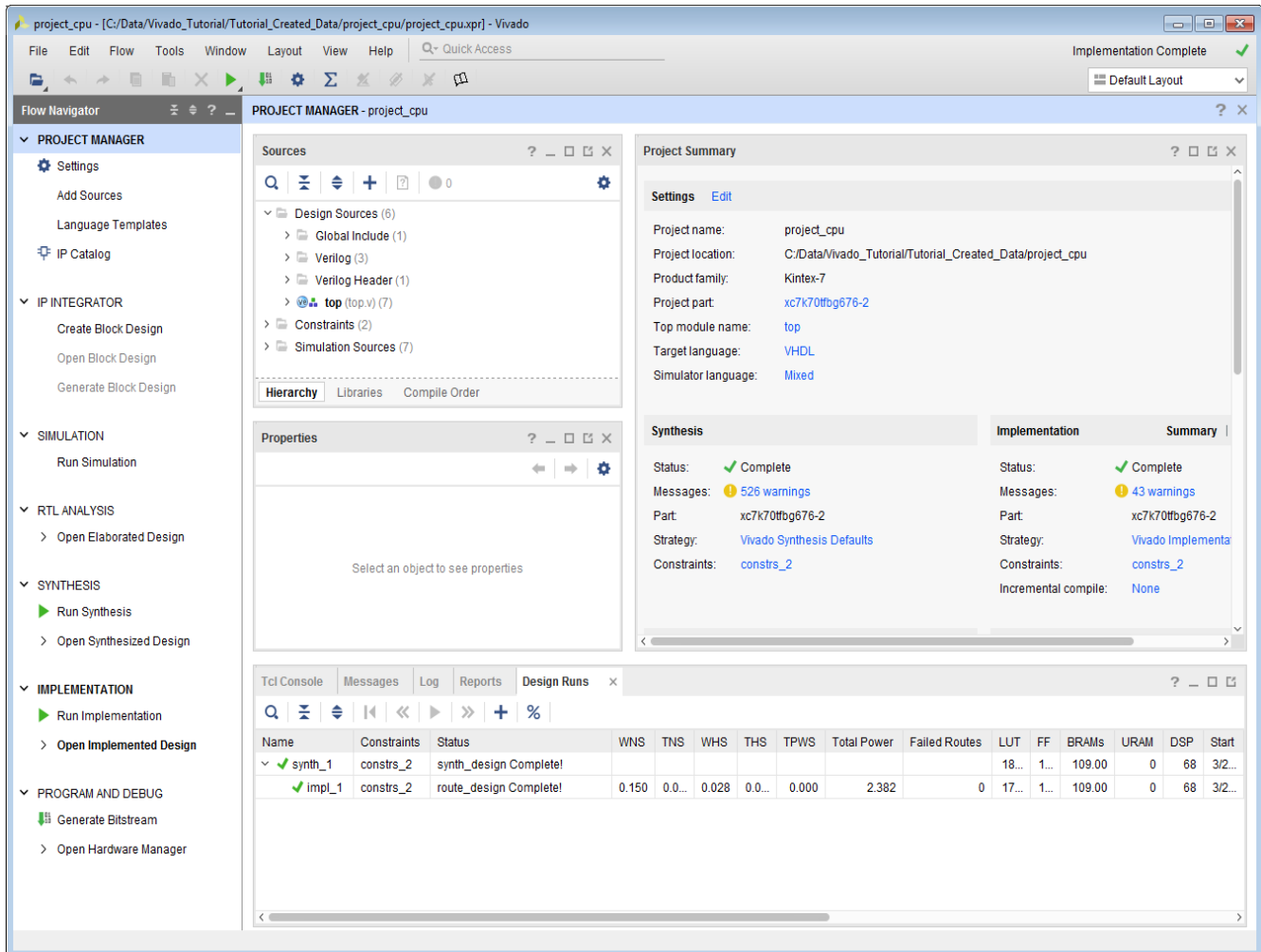


Figure 3-1: Flow Navigator in the Vivado IDE

Project Mode Advantages

Project Mode has the following advantages:

- Automatically manages project status, HDL sources, constraint files, IP cores and block designs.
- Generates and stores synthesis and implementation results
- Includes advanced design analysis capabilities, including cross probing from implementation results to RTL source files
- Automates setting command options using run strategies and generates standard reports
- Supports the creation of multiple runs to configure and explore available constraint or command options

Creating Projects

The Vivado Design Suite supports different types of projects for different design purposes. For example, you can create a project with RTL sources or synthesized netlists from third-party synthesis providers. You can also create empty I/O planning projects to enable device exploration and early pin planning. The Vivado IDE only displays commands relevant to the selected project type.

In the Vivado IDE, the Create Project wizard walks you through the process of creating a project. The wizard enables you to define the project, including the project name, the location in which to store the project, the project type (for example, RTL, netlist, and so forth), and the target part. You can add different types of sources, such as RTL, IP, Block designs, XDC or SDC constraints, simulation test benches, DSP modules from System Generator as IP, or Vivado High-Level Synthesis (HLS), and design documentation. When you select sources, you can determine whether to reference the source in its original location or to copy the source into the project directory. The Vivado Design Suite tracks the time and date stamp of each file and report status. If files are modified, you are alerted to out-of-date source or design status. For more information, see this [link](#) in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 11].



CAUTION! *The Windows operating system has a 260 character limit for path lengths which can affect the Vivado tools. To avoid this issue, use the shortest possible names and directory locations when creating projects, defining IP or managed IP projects, or creating block designs.*

Different Types of Projects

The Vivado Design Suite allows for different design entry points depending on your source file types and design tasks. Following are the different types of projects you can use to facilitate those tasks:

- **RTL Project:** You can add RTL source files and constraints, configure IP with the Vivado IP catalog, create IP subsystems with the Vivado IP integrator, synthesize and implement the design, and perform design planning and analysis.
- **Post-Synthesis Project:** You can import third-party netlists, implement the design, and perform design planning and analysis.
- **I/O Planning Project:** You can create an empty project for use with early I/O planning and device exploration prior to having RTL sources.
- **Imported Project:** You can import existing project sources from the ISE Design Suite, Xilinx Synthesis Technology (XST), or Synopsys Synplify.
- **Example Project:** You can explore several example projects, including example Zynq®-7000 SoC or MicroBlaze™ embedded designs with available Xilinx evaluation boards.
- **Partial Reconfiguration Project:** You can dynamically reconfigure an operating FPGA design by loading a partial bitstream file to modify reconfigurable regions of the device. The Partial Reconfiguration (PR) feature requires a separate license.

Managing Source Files in Project Mode

In Project Mode, source management is performed by the project infrastructure. The Vivado IDE manages different types of sources independently, including RTL design sources, IP, simulation sources, and constraint sources. It uses the concept of a source set to enable multiple versions of simulation or design constraints sets. This enables you to manage and experiment with different sets of design constraints in one design project. The Vivado IDE also uses the same approach for simulation, enabling management of module-level simulation sets for simulating different parts of the design.

When adding sources, you can reference sources from remote locations or copy sources locally into the project directory structure. Sources can be read from any network accessible location. With either approach, the Vivado IDE tracks the time and date stamps on the files to check for updates. If source files are modified, the Vivado IDE changes the project status to indicate whether synthesis or implementation runs are out of date. Sources with read-only permissions are processed accordingly.

When adding sources in the Vivado IDE, RTL files can optionally be scanned to look for include files or other global source files that might be in the source directory. All source file types within a specified directory or directory tree can be added with the **File > Add Sources** command. The Vivado IDE scans directories and subdirectories and imports any file with an extension matching the set of known sources types.

After sources are added to a project, the compilation order and logic hierarchy is derived and displayed in the Sources window. This can help you to identify malformed RTL or missing modules. The Messages window shows messages related to the RTL compilation, and you can cross probe from the messages to the RTL sources. In addition, source files can be enabled and disabled to allow for control over configuration.

Using Remote, Read-Only Sources

The Vivado Design Suite can utilize remote source files when creating projects or when read in Non-Project Mode. Source files can be read-only, which compiles the files in memory but does not allow changes to be saved to the original files. Source files can be saved to a different location if required.

Archiving Projects

In the Vivado IDE, the **File > Project > Archive** command creates a ZIP file for the entire project, including the source files, IP, design configuration, and optionally the run result data. If the project uses remote sources, the files are copied into the project locally to ensure that the archived project includes all files.

Creating a Tcl Script to Recreate the Project

In the Vivado IDE, the **File > Project > Write Tcl** command creates a Tcl script you can run to recreate the entire project, including the source files, IP, and design configuration. You can check this script into a source control system in place of the project directory structure.

Working with a Revision Control System

Many design teams use source management systems to store various design configurations and revisions. There are multiple commercially available systems, such as Revision Control System (RCS), Concurrent Versions System (CVS), Subversion (SVN), ClearCase, Perforce, Git, BitKeeper, and many others. The Vivado tools can interact with all such systems. The Vivado Design Suite uses and produces files throughout the design flow that you can manage with a revision control system. For more information on working with revision control software, refer to [Chapter 5, Source Management and Revision Control Recommendations](#).



VIDEO: For information on best practices when using revision control systems with the Vivado tools, see the [Vivado Design Suite QuickTake Video: Using Vivado Design Suite with Revision Control](#).

Understanding the Flow Navigator

The Flow Navigator (Figure 3-2) provides control over the major design process tasks, such as project configuration, synthesis, implementation, and bitstream generation. The commands and options available in the Flow Navigator depend on the status of the design. Unavailable steps are grayed out until required design tasks are completed.

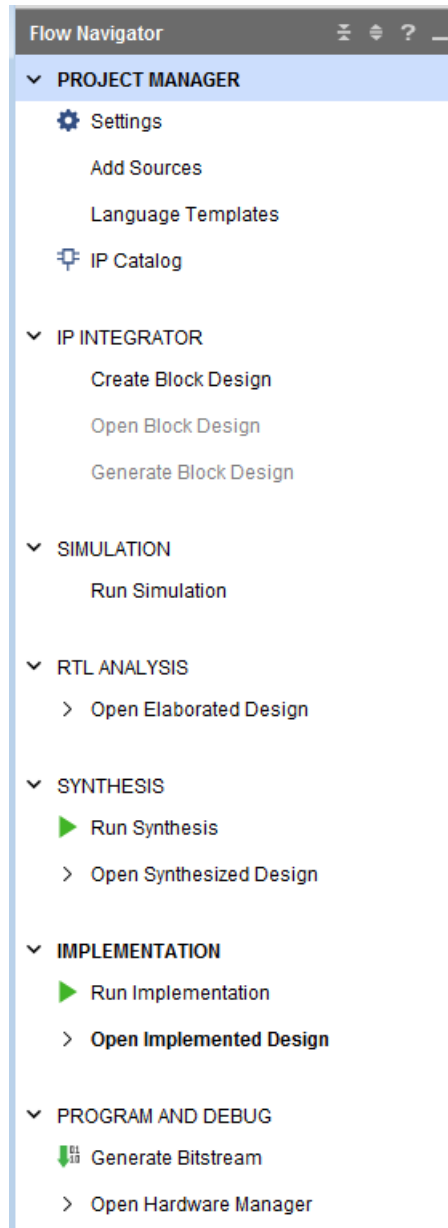


Figure 3-2: Flow Navigator

The Flow Navigator (Figure 3-3) differs when working with projects created with third-party netlists. For example, system-level design entry, IP, and synthesis options are not available.

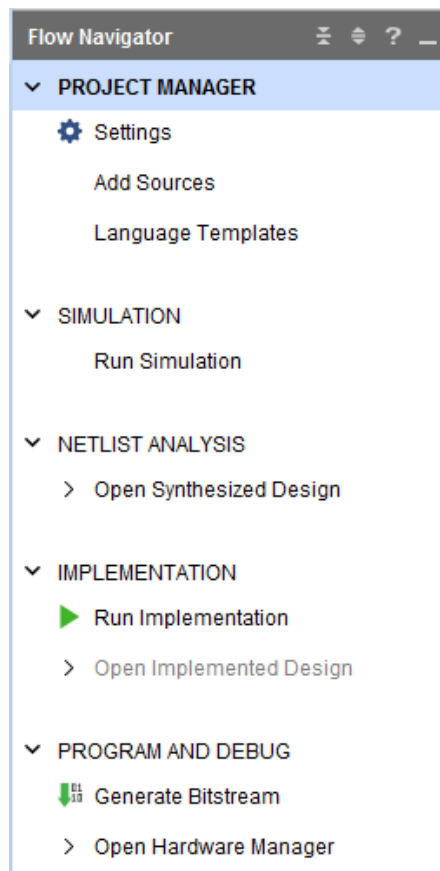


Figure 3-3: Flow Navigator for Third-Party Netlist Project

As the design tasks complete, you can open the resulting designs to analyze results and apply constraints. In the Flow Navigator, click **Open Elaborated Design**, **Open Synthesized Design**, or **Open Implemented Design**. For more information, see [Opening Designs to Perform Design Analysis and Constraints Definition](#).

When you open a design, the Flow Navigator shows a set of commonly used commands for the applicable phase of the design flow. Selecting any of these commands in the Flow Navigator opens the design, if it is not already opened, and performs the operation. For example, [Figure 3-4](#) shows the commands related to synthesis.

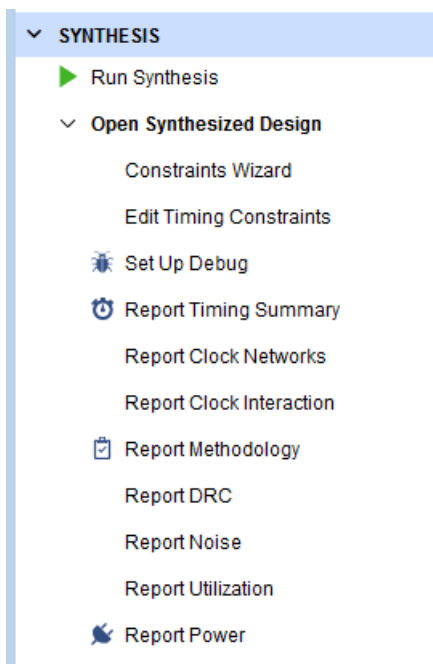


Figure 3-4: Synthesis Section in the Flow Navigator

Performing System-Level Design Entry

Automated Hierarchical Source File Compilation and Management

The Vivado IDE Sources window ([Figure 3-5](#)) provides automated source file management. The window has several views to display the sources using different methods. When you open or modify a project, the Sources window updates the status of the project sources. A quick compilation of the design source files is performed and the sources appear in the Compile Order view of the Sources window in the order they will be compiled by the downstream tools. Any potential issues with the compilation of the RTL hierarchy are shown as well as reported in the Message window. For more information on sources, see this [link](#) in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 11].



TIP: *If you explicitly set a module as the top module, the module is retained and passed to synthesis. However, if you do not explicitly set a top module, the Vivado tools select the best possible top module from the available source files in the project. If a file includes syntax errors and does not elaborate, this file is not selected as the top module by the Vivado tools.*

Constraints and simulation sources are organized into sets. You can use constraint sets to experiment with and manage constraints. You can launch different simulation sessions using different simulation source sets. You can add, remove, disable, or update any of the sources. For more information on constraints, see the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 18]. For more information on simulation, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

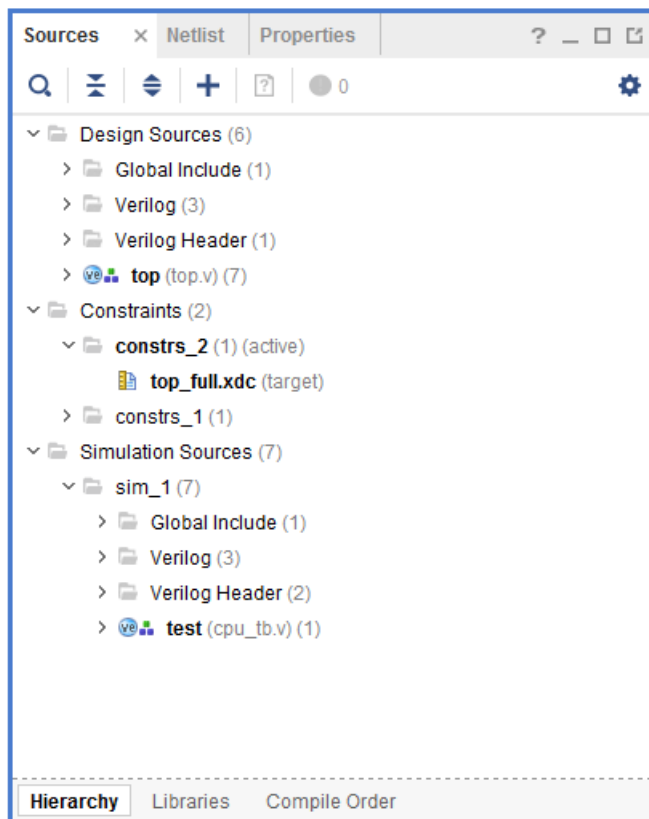


Figure 3-5: Hierarchical Sources View window

RTL Development

The Vivado IDE includes helpful features to assist with RTL development:

- Integrated Vivado IDE Text Editor to create or modify source files
- Automatic syntax and language construct checking across multiple source files
- Language templates for copying recommended example logic constructs
- Find in Files feature for searching template libraries using a variety of search criteria
- RTL elaboration and interactive analysis
- RTL design rule checks
- RTL constraints assignment and I/O planning

RTL Elaboration and Analysis

When you open an elaborated RTL design, the Vivado IDE compiles the RTL source files and loads the RTL netlist for interactive analysis. You can check RTL structure, syntax, and logic definitions. Analysis and reporting capabilities include:

- RTL compilation validation and syntax checking
- Run checks to ensure your RTL is compliant with the UltraFast Methodology rules
- Netlist and schematic exploration
- Design rule checks
- Early I/O pin planning using an RTL port list
- Ability to select an object in one view and cross probe to the object in other views, including instantiations and logic definitions within the RTL source files

For more information on RTL development and analysis features, see the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 18]. For more information on RTL-based I/O planning, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 15].

Timing Constraint Development and Verification

The Vivado IDE provides a Timing Constraints wizard to walk you through the process of creating and validating timing constraints for the design. The wizard identifies clocks and logic constructs in the design and provides an interface to enter and validate the timing constraints in the design. It is only available in synthesized and implemented designs, because the in-memory design must be clock aware post-synthesis. For more information, see the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 18].



TIP: The Vivado Design Suite only supports Synopsys design constraints (SDC) and Xilinx design constraints (XDC). It does not support Xilinx user constraints files (UCF) used with the ISE Design Suite nor does it directly support Synplicity design constraints. For information on migrating from UCF format to XDC format, see this [link](#) in the ISE to Vivado Design Suite Migration Guide (UG911) [Ref 25].

Working with IP

The Vivado Design Suite provides an IP-centric design flow that lets you configure, implement, verify, and integrate IP modules to your design from various design sources. The tool also provides an extensible IP catalog that includes Xilinx LogiCORE™ IP that can be configured and verified as a standalone module or within the context of a system-level design. For more information, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 12].

You can also package custom IP using the IP-XACT protocol and make it available through the Vivado IP catalog. Xilinx IP uses the AMBA® AXI4 interconnect standard to enable faster system-level integration. Existing IP can be added to a design as either RTL source or a netlist.

The available methods to work with IP in a design are as follows:

- Use the managed IP flow to customize IP and generate output products, including a synthesized design checkpoint (DCP) to preserve the customization for use in the current and future releases. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 12].
- Use IP in either Project or Non-Project modes by importing or reading the created Xilinx core instance (XCI) file. This is the recommended method for large projects with many team members.
- Access the IP catalog from a project to customize and add IP to a design. Store the IP files either local to the project, or save them externally from the project. This is the recommended method for small team projects.

Configuring IP

The Vivado IP catalog ([Figure 3-6](#)) lets you browse the available IP for the target device in the current project. The catalog shows version and licensing information about each IP and provides the applicable data sheet.

The Vivado IP catalog displays either **Included** or **Purchase** under the License column in the IP catalog. The following definitions apply to IP offered by Xilinx:

- **Included:** the Xilinx End User License Agreement includes Xilinx LogiCORE™ IP cores that are licensed within the Xilinx Vivado Design Suite software tools at no additional charge.
- **Purchase:** the Core License Agreement applies to fee-based Xilinx LogiCORE IP, and the Core Evaluation License Agreement applies to the evaluation of fee-based Xilinx LogiCORE IP.

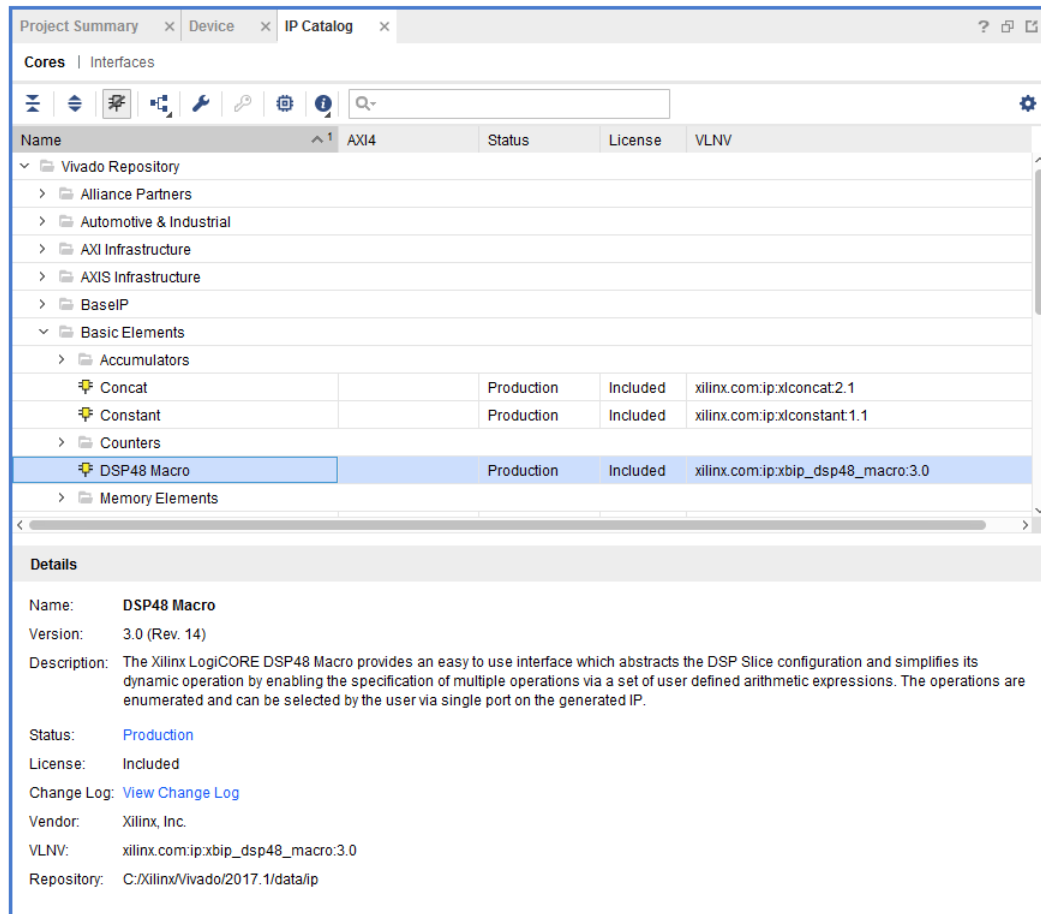


Figure 3-6: Vivado IP Catalog

This license status information is available for IP cores used in a project using Report IP Status by selecting **Reports > Report IP Status**. For additional information on how to obtain IP licenses, see the [Xilinx IP Licensing](#) page on the Xilinx website.

Xilinx and its partners provide additional IP cores that are not shipped as part of the default Vivado IP Catalog. For more information on the available IP, see the [Intellectual Property](#) page on the Xilinx website.

You can double-click any IP to launch the Configuration wizard to instantiate an IP into your design. After configuring the IP, a Xilinx Core Instance (.xci) file is created. This file contains all the customization options for the IP. From this file the tool can generate all output products for the IP. These output products consist of HDL for synthesis and simulation, constraints, possibly a test bench, C modules, example designs, etc. The tool creates these files based upon the customization options used.

Generating IP Output Products

IP output products are created to enable synthesis, simulation, and implementation tools to use a specific configuration of the IP. While generating output products, a directory structure is set up to store the various output products associated with the IP. The folders and files are fairly self-explanatory and should be left intact. The Vivado Design Suite generates the following output products:

- Instantiation template
- RTL source files and XDC constraints
- Synthesized design checkpoint (default)
- Third-party simulation sources
- Third-party synthesis sources
- Example design (for applicable IP)
- Test bench (for applicable IP)
- C Model (for applicable IP)



TIP: In Project Mode, missing output products are automatically generated during synthesis, including a synthesized design checkpoint (DCP) file for the out-of-context flow. In Non-Project Mode, the output products must be manually generated prior to global synthesis.

For each IP customized in your design, you should generate all available output products, including a synthesized design checkpoint. Doing so provides you with a complete representation of the IP that can be archived or placed in revision control. If future Vivado Design Suite versions do not include that IP, or if the IP has changed in undesirable ways (such as interface changes), you have all the output products required to simulate, and to use for synthesis and implementation with future Vivado Design Suite releases.

Using IP Core Containers

The optional Core Container feature helps simplify working with revision control systems by providing a single file representation of an IP. By enabling this option, you can store IP configuration files (XCI) and output products in a single, binary IP core container file (XCIX) rather than a loose directory structure. The XCIX file is similar to the XCI file and works in a similar way in the tool. For more information on using IP core containers, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 12]. For more information on revision control interaction with core containers, see [Managing IP Sources](#).

Out-of-Context Design Flow

By default, the Vivado Design Suite uses an out-of-context (OOC) design flow to synthesize IP from the IP catalog, and block designs from the Vivado IP integrator. This OOC flow lets you synthesize, implement, and analyze design modules in a hierarchical design, IP cores, or block designs, independent of the top-level design. The OOC flow reduces design cycle time, and eliminates design iterations, letting you preserve and reuse synthesis results.

IP cores that are added to a design from the Vivado IP catalog default to use the out-of-context flow. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 12]. Block designs created in the Vivado IP integrator also default to the OOC flow when generating output products. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 30].

The Vivado Design Suite also supports global synthesis and implementation of a design, in which all modules, block designs, and IP cores, are synthesized as part of the integrated top-level design. You can mark specific modules or IP for out-of-context synthesis, and other modules for inclusion in the global synthesis of the top-level design. In the case of a block design from Vivado IP integrator, the entire block design can be specified for OOC synthesis, or you can specify OOC synthesis for each individual IP, or per IP used in the block design. When run in global mode, Vivado synthesis has full visibility of design constraints. When run in OOC mode, estimated constraints are used during synthesis.

The Vivado synthesis tool also provides a cache to preserve OOC synthesis results for reuse in other designs that use the same IP customization. This can significantly speed synthesis of large complex designs.

A design checkpoint (DCP) is created for OOC IP or modules, which contains the synthesized netlist and design constraints. OOC modules are seen as black boxes in the top-level design until the synthesized design is open and all the elements are assembled. Before the top-level synthesized design is opened, resource utilization and analysis of the top-level design may not include netlist or resource information from the OOC modules, or black boxes, and so will not provide a complete view of the design.



IMPORTANT: *To obtain more accurate reports, you should open and analyze the top-level synthesized design, which will include all the integrated OOC modules.*

The OOC flow is supported in Vivado synthesis, implementation, and analysis. For more information refer to this [link](#) in the *Vivado Design Suite User Guide: Synthesis* (UG901) [Ref 17]. OOC synthesis can also be used to define a hierarchical design methodology and a team design approach as defined in the *Vivado Design Suite User Guide: Hierarchical Design* (UG905)[Ref 20].

IP Constraints

Many IP cores contain XDC constraint files that are used during Vivado synthesis and implementation. These constraints are applied automatically in both Project Mode and Non-Project Mode if the IP is customized from the Vivado IP catalog.

Many IP cores reference their input clocks in these XDC files. These clocks can come either from the user through the top level design, or from other IP cores in the design. By default, the Vivado tools process any IP clock creation and any user-defined top-level clock creation early. This process makes these clocks available to the IP cores that require them. Refer to this [link](#) in *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 12] for more information.

Validating the IP

You can verify Vivado IP by synthesizing the IP and using behavioral or structural logic simulation, and by implementing the IP module to validate timing, power, and resource utilization. Typically, a small example design is used to validate the standalone IP. You can also validate the IP within the context of the top-level design project. Because the IP creates synthesized design checkpoints, this bottom-up verification strategy works well either standalone or within a project.

Many of the Xilinx IP delivered in the Vivado IP catalog have an example design. You can determine if an IP comes with an example design by selecting the IP from the IP Sources area of the Manage IP or RTL project and see if the **Open IP Example Design** is selectable, as shown in [Figure 3-7](#). This can also be done using Tcl by examining the SUPPORTED_TARGETS property of the IP.

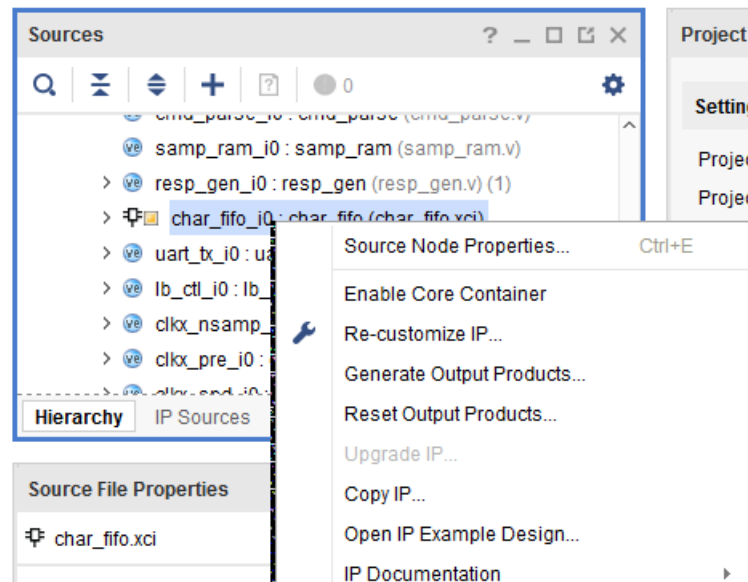


Figure 3-7: Opening an Example Design

Use the **Open IP Example Design** right-click menu command for a selected IP to create an example design to validate the standalone IP within the context of the example design project. For more details on working with example designs and IP output products, refer to the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 12].

Some IP deliver test benches with the example design, which you can use to validate the customized IP functionality. You can run behavioral, post synthesis, or post-implementation simulations. You can run either functional or timing simulations. In order to perform timing/functional simulations you will need to synthesize/implement the example design. For specific information on simulating an IP, refer to the product guide for the IP. For more detail on simulation, refer to the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Using Memory IP

Additional I/O pin planning steps are required when using Xilinx memory IP. After the IP is customized, you then assign the top-level I/O ports to physical package pins in either the elaborated or synthesized design in the Vivado IDE.

All of the ports associated with each memory IP are grouped together into an I/O Port Interface for easier identification and assignment. A Memory Bank/Byte Planner is provided to assist you with assigning Memory I/O pin groups to Byte lanes on the physical device pins.

For more information, see this [link](#) in the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 15].

If you have memory IP in your design, see the following resources:

- For details on simulation, see the *LogiCORE IP UltraScale Architecture-Based FPGAs Memory Interface Solutions Product Guide* (PG150) [Ref 3].
- For an example of simulating memory IP with a MicroBlaze™ processor design, see the *Reference System: Kintex-7 MicroBlaze System Simulation Using IP Integrator* (XAPP1180) [Ref 5].

Packaging Custom IP and IP Subsystems

The Vivado Design Suite lets you package custom IP or block designs into IP to list in the Vivado IP catalog for use in designs or in the Vivado IP integrator. You can package IP from a variety of sources, such as from a collection of RTL source files, a Vivado IP integrator block design, or an entire Vivado Design Suite project.

The location of the packaged IP can be added to the IP Repository section of the Settings dialog box which can be accessed through **Tools > Settings** menu in the Vivado IDE. After a repository of one or more IP has been added, the IP core(s) from the repository will be shown in the IP Catalog.



TIP: Before packaging your IP HDL, ensure its correctness by simulating and synthesizing to validate the design.

There are multiple ways to configure the IP and make it available for use within the Vivado IP catalog and IP integrator. For example, the Create and Package IP wizard takes you step-by-step through IP packaging and lets you package IP from a project, a block design, or a specified directory. You can also create and package a new template AXI4 peripheral for use in embedded processor designs.



IMPORTANT: Ensure that the desired list of supported device families is defined properly while creating the custom IP definition. This is especially important if you want your IP to be used with multiple device families.

For more information, see the *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) [Ref 32] and *Vivado Design Suite Tutorial: Creating and Packaging Custom IP* (UG1119) [Ref 33].

Upgrading IP

With each release of the Vivado Design Suite, new IP versions are introduced. It is recommended that you upgrade the IP used in your designs at each new release. However, you can also use the older version as a static IP that is already configured and synthesized to avoid introducing any unnecessary changes into your design. To use the static version of an existing IP, all of the output products must have been previously generated for the IP, and no changes to those generated output files will be possible. For more information refer to this [link](#) in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 12].

To report on the current status of the IP in a design, you can use the `report_ip_status` Tcl command. If changes are needed, you can selectively upgrade the IP in the design to the latest version. A change log for each IP details the changes made and lists any design updates that are required. For example, top-level port changes are occasionally made in newer IP versions, so some design modification might be required. If the IP version has changed in the latest release, the version used in the design becomes locked and must be used as a static IP with the available output products, or must be updated to support the current release. Locked IP are reported as locked, and appear with a lock symbol in the Sources window of the Vivado IDE.

Creating IP Subsystems with IP Integrator

The Vivado IP integrator enables the creation of Block Designs (.bd), or IP subsystems with multiple IP stitched together using the AXI4 interconnect protocol. The IP Integrator lets you quickly connect IP cores to create domain specific subsystems and designs, including embedded processor-based designs using Zynq® UltraScale+™ MPSoC, Zynq-7000 SoC, and MicroBlaze processors. It can instantiate High-Level Synthesis modules from Vivado HLS, DSP modules from System Generator, and custom user-defined IP as described in [Packaging Custom IP and IP Subsystems](#).

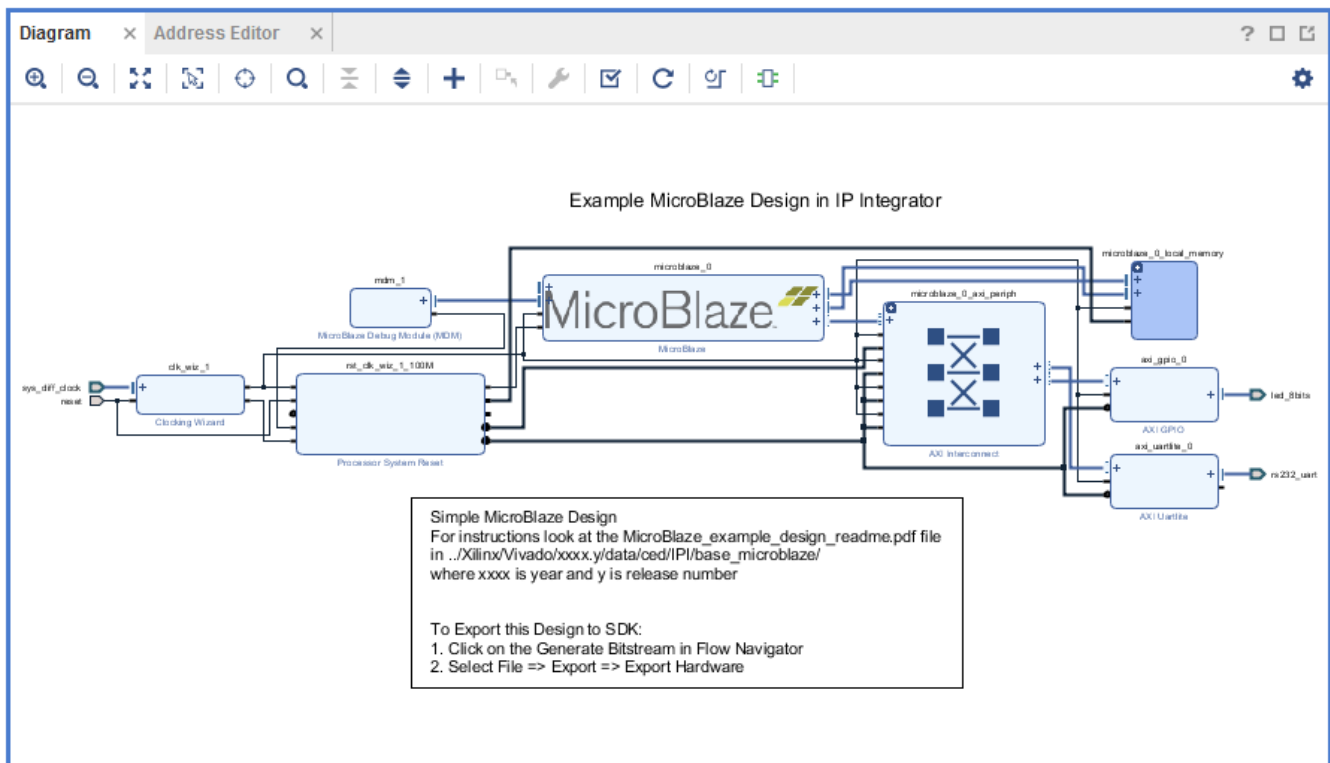


Figure 3-8: Vivado IP Integrator

Using Vivado IP Integrator you can drag and drop IP onto the design canvas, connect AXI interfaces with one wire, and place ports and interface ports to connect the IP subsystem to the top-level design. These IP block designs can also be packaged as sources (.bd) and reused in other designs. For more information, see the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 30] or *Vivado Design Suite User Guide: Embedded Processor Hardware Design* (UG898) [Ref 14].

Building IP Subsystems

The interactive block design capabilities of the Vivado IP Integrator make the job of configuring and assembling groups of IP easy.



TIP: *If you prefer to work with a Tcl script, it is suggested to create the block design interactively using the Vivado IDE and then capture and edit the script as needed to recreate the block design. The IP Integrator can create a Tcl script to re-create the current block design in memory.*

Referencing RTL Modules in Block Designs

The Module Reference feature of the Vivado IP Integrator lets you quickly add a module or entity definition from a Verilog or VHDL source file directly into your block design. This provides a means of quickly adding RTL modules without having to go through the process of packaging the RTL as an IP to be added through the Vivado IP catalog. The Module Reference flow is quick, but does not offer the benefits of the working through the IP catalog. Both flows have the benefits and associated limitations. Refer to this [link](#) in *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 30] for more information.

Designer Assistance

To expedite the creation of a subsystem or a design, the IP Integrator offers Block Automation and Connection Automation. The Block Automation feature can be used to configure a basic processor-based design and some complex IP subsystems, while the Connection Automation feature can be used to automatically make repetitive connections to different pins or ports of the design. IP integrator also supports all the Xilinx evaluation boards in the Platform Board Flow, as described below in [Using the Platform Board Flow](#). This lets the Connection Automation feature connect the I/O ports of the design to components on the target board. Designer assistance also helps with defining and connecting clocks and resets. Using Designer Assistance not only expedites the design process but also helps prevent unintended design errors.

Using the Platform Board Flow

The Vivado Design Suite is board aware and can automatically derive I/O constraints and IP configuration data from included board files. Through the board files, the Vivado Design Suite knows the various components present on the target boards and can customize and configure an IP to be connected to a particular board component. Several 7 series, Zynq-7000 SoC, and UltraScale™ device boards are currently supported. You can download support files for partner-developed boards from the partner websites.

The IP Integrator shows all the component interfaces on the target board in a separate tab called the Board tab. You can use this tab to connect to the desired components through the Designer Assistance feature. All the I/O constraints are automatically generated as a part of using this feature.

You can also generate board files for custom boards and add the repository that contains the board file to a project. For more information on generating a custom board file, see this [link](#) in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 11].

Validating IP Subsystems

IP Integrator runs basic design rule checks in real time as the design is being assembled. However, there is still a potential for design errors, such as the frequency on a clock pin may be set incorrectly. The tool can catch these types of errors by running a more thorough design validation. You can run design validation by selecting **Tools > Validate Design** or through the Tcl command `validate_bd_design`.

The Validate Design command applies design rule checks on the block design and reports warnings and/or errors found in the design. You can cross-probe the warnings and/or errors from the Messages window to locate objects in the block diagram. Xilinx recommends validating a block design to catch errors that would otherwise be found later in the design flow.

Running design validation also runs Parameter Propagation on the block design. Parameter Propagation enables IP Integrator to automatically update the parameters associated with a given IP based on its context and its connections in the design. You can package custom IP with specific parameter propagation rules, and IP Integrator applies these rules as the block diagram is generated and validated. See this [link](#) in *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 30] for more information.

Generating Block Design Output Products

After the block design or IP subsystem has been created, you can generate the block design including all source codes, necessary constraints for the IP cores, and the structural netlist of the block design. You can generate the block design by right-clicking on the block design (in the Sources window) and selecting **Generate Output Products** from the pop-up menu. In the Vivado Design Suite Flow Navigator you can also select **IP Integrator > Generate Block Design**.

There are two modes of OOC supported for block designs in the Vivado Design Suite: **Out of context per Block design** and **Out of context per IP**. Refer to [Out-of-Context Design Flow](#) for more information.

Integrating the Block Design into a Top-Level Design

An IP Integrator block design can be integrated into a higher-level design or it can be the highest level in the design hierarchy. To integrate the IP Integrator design into a higher-level design, instantiate the block design as a module in the higher-level HDL file.

You can perform a higher-level instantiation of the block design by selecting the block design in the Vivado IDE Sources window and selecting **Create HDL Wrapper**. This generates a top-level HDL file for the IP Integrator sub-system. See this [link](#) in *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 30] for more information.

Logic Simulation

The Vivado Design suite has several logic simulation options for verifying designs or IP. The Vivado simulator, integrated into the Vivado IDE, allows you to simulate the design, add and view signals in the waveform viewer, and examine and debug the design as needed.

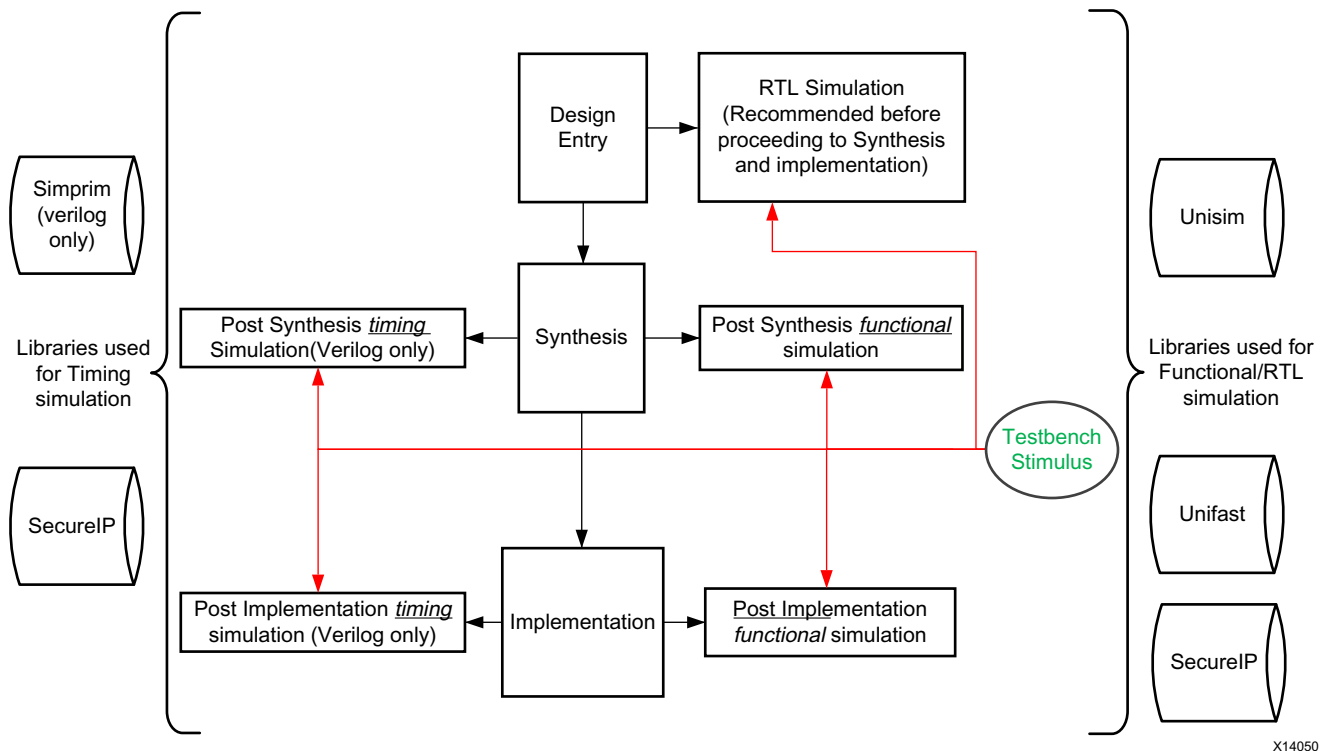


Figure 3-9: Simulation at Various Points in the Design Flow

You can use the Vivado simulator to perform behavioral and structural simulation of designs as well as full timing simulation of implemented designs. [Figure 3-9](#) shows all the places where Vivado simulation could be used for functional and timing simulation. You can also use third-party simulators by writing Verilog or VHDL netlists, and SDF files from the elaborated, synthesized, or implemented design. The Vivado IDE lets you configure and launch simulators from Mentor Graphics, Synopsys, Cadence, and Aldec. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Simulation Flow Overview

The following are some key suggestions related to simulating in the Vivado Design Suite. Many of these tips are described in greater detail in the text that follows, or in *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

1. Run behavioral simulation before proceeding with synthesis and implementation. Issues identified early will save time and money.
2. Infer logic wherever possible. Instantiating primitives adds significant simulation runtime cost.
3. Always set the Target Language to **Mixed** unless you do not have a mixed mode license for your simulator.
4. Turn off the waveform viewer when not in use to improve simulation performance.
5. In the Vivado simulator, turn off debug during `xe1ab` for a performance boost.
6. In the Vivado simulator, turn on multi-threading to speed up compile time.
7. When using third-party simulators, always target supported versions. For more information, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973) [Ref 29].
8. Make sure incremental compile is turned on when using third-party simulators.
9. Use the Xilinx Tcl command `export_simulation` to generate batch scripts for selected simulators.
10. Generate simulation scripts for individual IP, BDs, and hierarchical modules as well as for the top-level design.
11. If you are targeting a 7 series device, use UNIFAST libraries to improve simulation performance.

Note: The UNIFAST libraries are not supported for UltraScale device primitives.

Compiling Simulation Libraries

Vivado delivers precompiled simulation libraries for use with the Vivado simulator, as well as precompiled libraries for all the static files required by Xilinx IP. When simulation scripts are created, they reference these precompiled libraries.

When using third-party simulators, you must compile Xilinx simulation libraries prior to running simulation, as explained in the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16]. This is especially true if your design instantiates VHDL primitives or Xilinx IP, the majority of which are in VHDL form. The simulation tool will return “library binding” failures if you do not precompile simulation libraries.

You can run the `compile_simlib` Tcl command to compile the Xilinx simulation libraries for the target simulator. You can also issue this command from the Vivado IDE by selecting **Tools > Compile Simulation Libraries**.



IMPORTANT: *Simulation libraries are pre-compiled and provided for use with the Vivado simulator. However, you must manually compile the libraries for use with a third-party simulator. Refer to this [link](#) in the Vivado Design Suite User Guide: Logic Simulation (UG900) [Ref 16] for more information.*

Simulation Time Resolution

Xilinx recommends that you run simulations using a resolution of 1 ps. Some Xilinx primitive components, such as MMCM, require a 1 ps resolution to work properly in either functional or timing simulation.



TIP: *Because most of the simulation time is spent in delta cycles, there is no significant simulator performance gain by using coarser resolution with the Xilinx simulation models.*

There is no need to use a finer resolution, such as femtoseconds (fs) as some simulators will round the numbers while others will truncate the numbers.

Functional Simulation Early in the Design Flow

Use functional or register transfer level (RTL) simulation to verify syntax and functionality. This first pass simulation is typically performed to verify the RTL or behavioral code and to confirm that the design is functioning as intended.

With larger hierarchical designs, you can simulate individual IP, block designs, or hierarchical modules before testing your complete design. This simulation process makes it easier to debug your code in smaller portions before examining the larger design. When each module simulates as expected, create a top-level design test bench to verify that your entire design functions as planned. Use the same test bench again for the final timing simulation to confirm that your design functions as expected under worst-case delay conditions.



RECOMMENDED: *At this stage, no timing information is provided. Xilinx recommends performing simulation in unit-delay mode to avoid the possibility of a race condition.*

You should use synthesizable HDL constructs for the initial design creation. Do not instantiate specific components unless necessary. This allows for:

- More readable code
- Faster and simpler simulation
- Code portability (the ability to migrate to different device families)
- Code reuse (the ability to use the same code in future designs)



TIP: *You might need to instantiate components if the components cannot be inferred.*

Instantiation of components can make your design code architecture specific.

Using Structural Netlists for Simulation

After synthesis or implementation, you can perform netlist simulation in functional or timing mode. The netlist simulation can also help you with the following:

- Identify post-synthesis and post-implementation functionality changes caused by:
 - Synthesis attributes or constraints that create mismatches (such as `full_case` and `parallel_case`)
 - UNISIM attributes applied in the Xilinx Design Constraints (XDC) file
 - Differences in language interpretation between synthesis and simulation
 - Dual-port RAM collisions
 - Missing or improperly applied timing constraints
 - Operation of asynchronous paths
 - Functional issues due to optimization techniques
- Sensitize timing paths declared as false or multi-cycle during STA
- Generate netlist switching activity to estimate power
- Identify X state pessimism

For netlist simulation, you can use one or more of the libraries shown in the following table.

Table 3-1: Use of Simulation Library

Library Name	Description	VHDL Library Name	Verilog Library Name
UNISIM	Functional simulation of Xilinx primitives	UNISIM	UNISIMS_VER
UNIMACRO	Functional simulation of Xilinx macros	UNIMACRO	UNIMACRO_VER
UNIFAST	Fast simulation library	UNIFAST	UNIFAST_VER

The UNIFAST library is an optional library that you can use during functional simulation to speed up simulation runtime. UNIFAST libraries are supported for 7 series devices only. UltraScale and later device architectures do not support UNIFAST libraries, because all the optimizations are incorporated in the UNISIM libraries by default. For more information on Xilinx simulation libraries, see this [link](#) in the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Primitives/elements of the UNISIM library do not have any timing information except the clocked elements. To prevent race conditions during functional simulation, clocked elements have a clock-to-out delay of 100 ps. Waveform views might show spikes and glitches for combinatorial signals, due to lack of any delay in the UNISIM elements.

Timing Simulation

Xilinx supports timing simulation in Verilog only. You can export a netlist for timing simulation from an open synthesized or implemented design using the **File > Export > Export Netlist** command in the Vivado IDE, or by using the `write_verilog` Tcl command.

The Verilog system task `$sdf_annotate` within the simulation netlist specifies the name of the standard delay format (SDF) file to be read for timing delays. This directive is added to the exported netlist when the `-sdf_anno` option is enabled on the Netlist tab of the Simulation Settings dialog box in the Vivado IDE. The SDF file can be written with the `write_sdf` command. The Vivado simulator automatically reads the SDF file during the compilation step.



TIP: The Vivado simulator supports mixed-language simulation, which means that if you are a VHDL user, you can generate a Verilog simulation netlist and instantiate it from the VHDL test bench.

Many users do not run timing simulation due to high run time. However, you should consider using full timing simulation because it is the closest method of modeling hardware behavior. If your design does not work on hardware, it is much easier to debug the failure in simulation, as long as you have a timing simulation that can reproduce the failure.

If you decide to skip timing simulation, you should make sure of the following:

- Ensure that your STA constraints are absolutely correct. Pay special attention to exceptions.
- Ensure that your netlist is exactly equivalent to what you intended through your RTL. Pay special attention to any inference-related information provided by the synthesis tool.

Simulation Flow

The Vivado Design Suite supports both integrated simulation, which allows you to run the simulator from within the Vivado IDE, and batch simulation, which allows you to generate a script from the Vivado tools to run simulation on an external verification environment.

Integrated Simulation

The Vivado IDE provides full integration with the Vivado simulator, and all supported third-party simulators. In this flow, the simulator is called from within the Vivado IDE, and you can compile and simulate the design easily with a push of a button, or with the `launch_simulation` Tcl command.



IMPORTANT: *The `launch_simulation` command launches integrated simulation for project-based designs. This command does not support Non-Project Mode.*

For information on the steps involved in setting up the integrated simulation flow, see this [link](#) in the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Batch Simulation



RECOMMENDED: *If your verification environment has a self-checking test bench, run simulation in batch mode. There is a significant runtime cost when you view simulator waveforms using the integrated simulation.*

For batch simulation, the Vivado Design Suite provides the `export_simulation` Tcl command to generate simulation scripts for supported simulators, including the Vivado simulator. You can use the scripts generated by `export_simulation` directly or use the scripts as a reference for building your own custom simulation scripts.

The `export_simulation` command creates separate scripts for each stage of the simulation process (compile, elaborate, and simulate) so that you can easily incorporate the generated scripts in your own verification flow. For more information about generating scripts for batch simulation, see this [link](#) in the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Running Logic Synthesis and Implementation

Logic Synthesis

Vivado synthesis enables you to configure, launch, and monitor synthesis runs. The Vivado IDE displays the synthesis results and creates report files. You can select synthesis warnings and errors from the Messages window to highlight the logic in the RTL source files.

You can launch multiple synthesis runs concurrently or serially. On a Linux system, you can launch runs locally or on remote servers. With multiple synthesis runs, Vivado synthesis creates multiple netlists that are stored with the Vivado Design Suite project. You can open different versions of the synthesized netlist in the Vivado IDE to perform device and design analysis. You can also create constraints for I/O pin planning, timing, floorplanning, and implementation. The most comprehensive list of DRCs is available after a synthesized netlist is produced, when clock and clock logic are available for analysis and placement.

For more information, see the *Vivado Design Suite User Guide: Synthesis* (UG901) [Ref 17].

Implementation

Vivado implementation enables you to configure, launch, and monitor implementation runs. You can experiment with different implementation options and create your own reusable strategies for implementation runs. For example, you can create strategies for quick runtimes, improved system performance, or area optimization. As the runs complete, implementation run results display and report files are available.

You can launch multiple implementation runs either simultaneously or serially. On a Linux system, you can use remote servers. You can create constraint sets to experiment with various timing constraints, physical constraints, or alternate devices. For more information, see the *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 19] and *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 18],



TIP: You can add Tcl scripts to be sourced before and after synthesis, any stage of implementation, or bitstream generation using the `tcl.pre` and `tcl.post` files. For more information, see the *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894) [Ref 10].

Configuring Synthesis and Implementation Runs

When using Project Mode, various settings are available to control the features of synthesis and implementation. These settings are passed to runs using run strategies, which you set in the Settings dialog box. A run strategy is simply a saved set of run configuration parameters. Xilinx supplies several pre-defined run strategies for running synthesis and implementation, or you can apply custom run settings. In addition, you can use separate constraint sets for synthesis and implementation.

For information on modifying settings, see this [link](#) to the *Vivado Design Suite User Guide: Synthesis* (UG901) [Ref 17] and see this [link](#) in the *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 19].



TIP: You can create an out-of-context module run to synthesize the Vivado Design Suite IP in the project. If you generate a design checkpoint for the IP, the default behavior is to create an out-of-context run for each IP in the design.

Creating and Managing Runs

After the synthesis and implementation settings are configured in the Settings dialog box, you can launch synthesis or implementation runs using any of the following methods:

- In the Flow Navigator, select **Run Synthesis**, **Run Implementation**, or **Generate Bitstream**.
- In the Design Runs window, select a run, right-click, and select **Launch Runs**. Alternatively, you can click the **Launch Selected Runs** button.
- Select **Flow > Run Synthesis**, **Flow > Run Implementation**, or **Flow > Generate Bitstream**.

You can create multiple synthesis or implementation runs to experiment with constraints or tool settings. To create additional runs:

1. In the Flow Navigator, right-click **Synthesis** or **Implementation**.
2. Select **Create Synthesis Runs** or **Create Implementation Runs**.
3. In the Create New Runs wizard (Figure 3-10), select the constraint set and target part.

If more than one synthesis run exists, you can also select the netlist when creating implementation runs. You can then create one or more runs with varying strategies, constraint sets, or devices. There are several launch options available when multiple runs exist. You can launch selected runs sequentially or in parallel on multiple local processors.



TIP: You can configure and use remote hosts on Linux systems only.

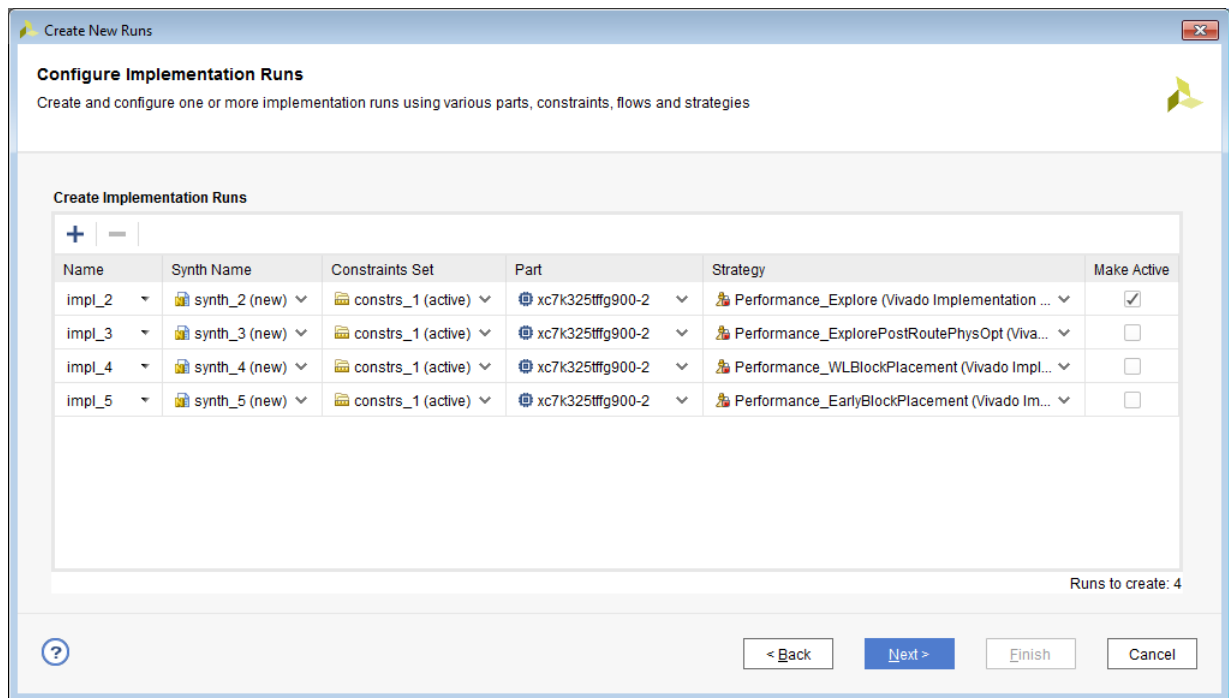


Figure 3-10: Creating Multiple Synthesis and Implementation Runs

Managing Runs with the Design Runs Window

The Design Runs windows (Figure 3-11) displays run status and information and provides access to run management commands in the popup menu. You can manage multiple runs from the Design Runs window. When multiple runs exist, the active run is displayed in bold. The Vivado IDE displays the design information for the active run. The Project Summary, reports, and messages all reflect the results of the active run.

The Vivado IDE opens the active design by default when you select **Open Synthesized Design** or **Open Implemented Design** in the Flow Navigator. You can make a run the active run using the **Make Active** popup menu command. The Vivado IDE updates results to reflect the information about the newly designated active run. Double-click any synthesized or implemented run to open the design in the Vivado IDE.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BR/
synth_1 (active)	constrs_1	synth_design Complete!								0	0	
impl_1 (active)	constrs_1	Running route_design...										
impl_2	constrs_1	Queued...										
impl_3	constrs_1	Queued...										
impl_4	constrs_1	Queued...										
impl_5	constrs_1	Not started										
impl_6	constrs_1	Not started										

Figure 3-11: Design Runs Window

Resetting Runs

In the Flow Navigator, you can right-click **Synthesis** or **Implementation**, and use the following popup menu commands to reset runs. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 19].

- **Reset Runs** resets the run to its original state and optionally deletes generated files from the run directory.
- **Reset to Previous Step** resets the run to the listed step.



TIP: To stop an in-process run, click the **Cancel** button in the upper right corner of the Vivado IDE.

Launching Runs on Remote Clusters

To launch runs on remote Linux hosts, you can directly access a load sharing facility (LSF) server farm. Vivado allows all cluster commands to be configured through Tcl. For information, see this [link](#) in the *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 19].

Performing Implementation with Incremental Compile

You can specify the incremental compile flow when running Vivado implementation to facilitate small design changes. Incremental compile can reduce place and route run times and preserve existing implementation results depending on the scope of the change and the amount of timing-critical logic that is modified.

You can specify the Set Incremental Compile option in the Implementation Settings dialog box in the Vivado IDE, or by using the Set Incremental Compile command from the right-click menu of the Design Runs window. You can also use the `read_checkpoint` Tcl command with the `-incremental` option, and point to a routed design checkpoint to use as a reference. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 19].

Implementing Engineering Change Orders (ECOs)

Engineering change orders (ECOs) are modifications to an implemented design, with the intent to minimize impact to the original design. The Vivado Design Suite provides an ECO flow, which lets you modify an existing design checkpoint to implement changes, run reports on the changed netlist, and generate the required bitstream files.

The advantage of the ECO flow is fast turn-around time by taking advantage of the incremental place and route features of the Vivado tool. The Vivado IDE provides a predefined layout to support the ECO flow. Refer to this [link](#) in the *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 19] for more information.

Viewing Log Files, Messages, Reports, and Properties

Viewing Log Files

In the Log window (Figure 3-12), you can click the different tabs to view the standard output for **Synthesis**, **Implementation**, and **Simulation**. This output is also included in the `vivado.log` file that is written to the Vivado IDE launch directory.

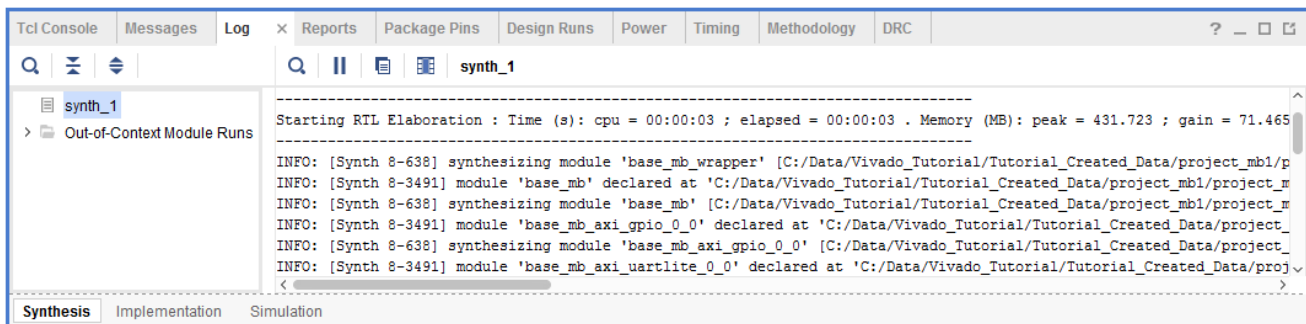


Figure 3-12: Viewing Log Files

Viewing Messages

In the Messages window (Figure 3-13), messages are categorized according to design step and severity level: Errors, Critical Warnings, Warnings, Info, and Status. To filter messages,

select the appropriate check boxes in the window header. You can expand the message categories to view specific messages. You can click the **Collapse All** icon to show only the main design steps. This enables better navigation to specific messages. Many messages include links that take you to logic lines in the RTL files. For more information, including advanced filtering techniques, see this [link](#) in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 9].



Figure 3-13: Viewing Messages

Viewing Reports

In the Reports window (Figure 3-14), several standard reports are generated using the `launch_runs` Tcl commands. You can double-click any report to display it in the Vivado IDE Text Editor. You can also create custom reports using Tcl commands in the Tcl Console or using report strategies. For more information, see this [link](#) and this [link](#) in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 9] and see this [link](#) and this [link](#) in *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 21].

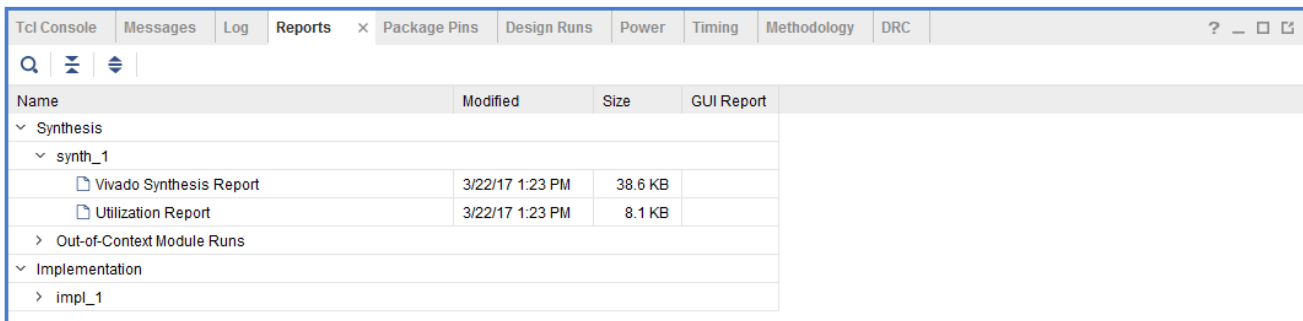


Figure 3-14: Viewing Reports

Viewing or Editing Device Properties

With the elaborated, synthesized, or implemented design open, you can use the **Tools > Edit Device Properties** command to open the Edit Device Properties dialog box (Figure 3-15) in which you can view and set device configuration and bitstream-related properties. This command is available only when a design is open. For information on each property, see the link in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 22]. For information on setting device configuration modes, see this link in the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 15].

Note: The **Edit > Device Properties** is only available when a design is open.

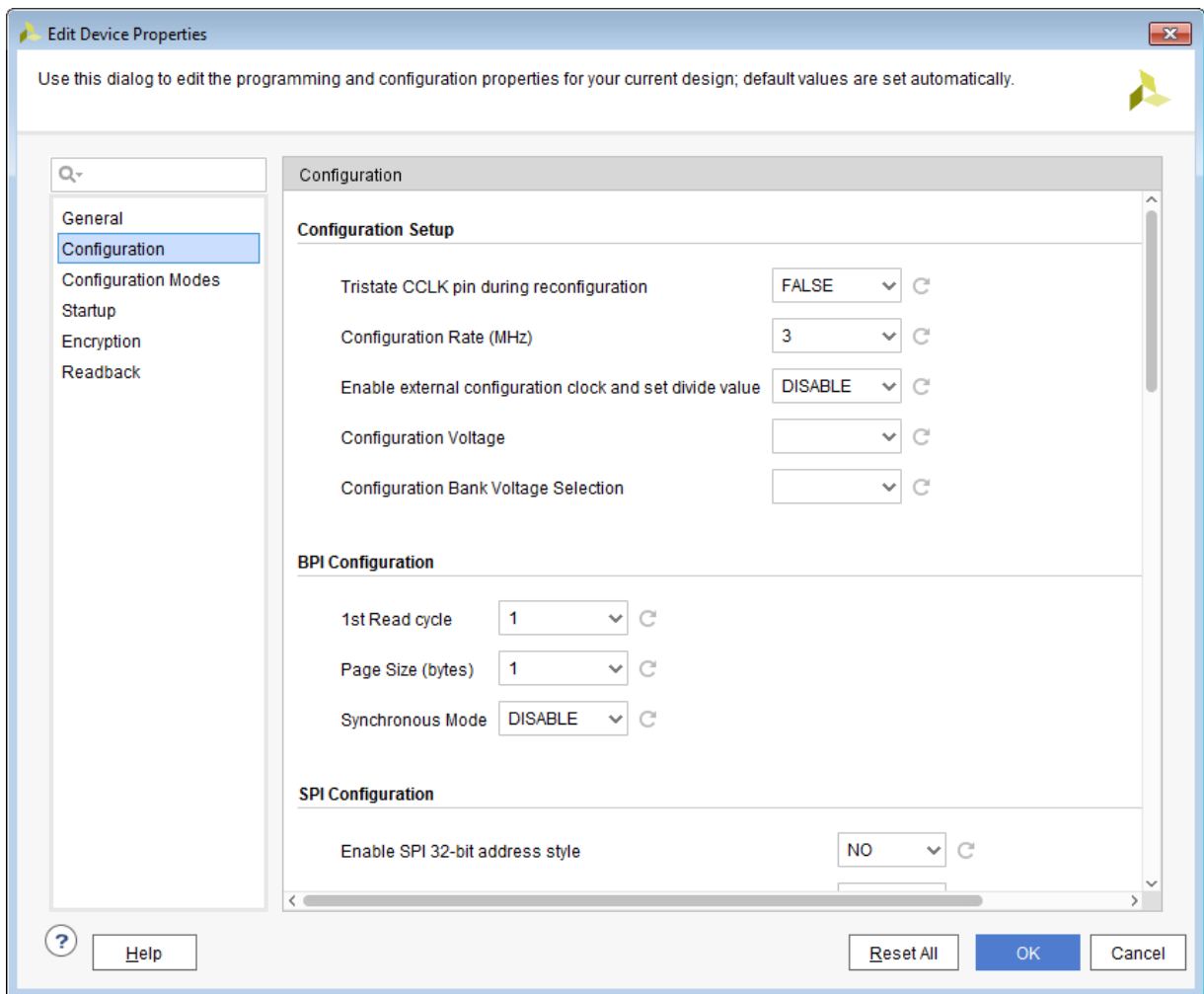


Figure 3-15: Viewing Device Properties

Opening Designs to Perform Design Analysis and Constraints Definition

You can perform design analysis and assign constraints after RTL elaboration, after synthesis, or after implementation. To identify design issues early, you can perform design analysis prior to implementation, including timing simulation, resource estimation, connectivity analysis, and DRCs. You can open the various synthesis or implementation run results for analysis and constraints assignment. This is known as opening the design.

When you open the design, the Vivado IDE compiles the netlist and applies physical and timing constraints against a target part. You can open, save, and close designs. When you open a new design, you are prompted to close any previously opened designs in order to preserve memory. However, you are not required to close the designs, because multiple designs can be opened simultaneously. When you open a synthesized design, the Vivado IDE displays the netlist and constraints. When you open an implemented design, the Vivado IDE displays the netlist, constraints, and implementation results. The design data is presented in different forms in different windows, and you can cross probe and coordinate data between windows.

After opening a design, many analysis and reporting features are available in the Vivado IDE. For example, you can analyze device resources in the graphical windows of the internal device and the external physical package. You can also apply and analyze timing and physical constraints in the design using the Netlist, Device, Schematic, or Hierarchy windows. For more information, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 21] and *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 18].

Note: If you make constraint changes while the design is open, you are prompted to save the changes to the original XDC source files or to create a new constraint set. For more information, see this [link](#) in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 11].

Opening an Elaborated RTL Design

When you open an elaborated design, the Vivado Design Suite expands and compiles the RTL netlist and applies physical and timing constraints against a target part. The different elements of the elaborated design are loaded into memory, and you can analyze and modify the elements as needed to complete the design. For more information, see this [link](#) in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 11].

The Vivado Design Suite includes linting DRCs and checking tools that enable you to analyze your design for logic correctness. You can make sure that there are no logic compilation issues, no missing modules, and no interface mismatches. In the Messages window, you can click links in the messages to display the problem lines in the RTL files in the Vivado IDE Text Editor. In the Schematic window, you can explore the logic interconnects and hierarchy in a variety of ways. The Schematic window displays RTL interconnects using RTL-based logic constructs. You can select logic in the Schematic window and see specific lines in the RTL files in the Vivado IDE Text Editor. For more information, see this [link](#) in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 11].

Note: There is no FPGA technology mapping during RTL elaboration.

Constraints that are defined on specific logic instances within the logic hierarchy, such as registers, might not be resolvable during RTL elaboration. The logic names and hierarchy generated during elaboration might not match those generated during synthesis. For this reason, you might see constraint mapping warnings or errors when elaborating the RTL design, if you have these types of constraints defined. However, when you run synthesis on the design, these issues are resolved.

Using the I/O planning capabilities of the Vivado IDE, you can interactively configure and assign I/O Ports in the elaborated RTL design and run DRCs. When possible, it is recommended that you perform I/O planning after synthesis. This ensures proper clock and logic constraint resolution, and the DRCs performed after synthesis are more extensive. For more information, see *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 15].



TIP: When you select the **Report DRC** command, the Vivado IDE invokes a set of RTL and I/O DRCs to identify logic issues such as asynchronous clocks, latches, and so forth. For more information, see this [link](#) in the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895) [Ref 11].

To open an elaborated design, use one of the following methods:

- In the RTL Analysis section of the Flow Navigator, select **Open Elaborated Design**.
- In the Flow Navigator, right-click **RTL Analysis**, and select **New Elaborated Design** from the popup menu.
- Select **Flow > Open Elaborated Design**.

Figure 3-16 shows the default view layout for an open elaborated RTL design. Notice the logic instance that was cross-selected from the schematic to the specific instance in the RTL source file and within the elaborated RTL netlist.

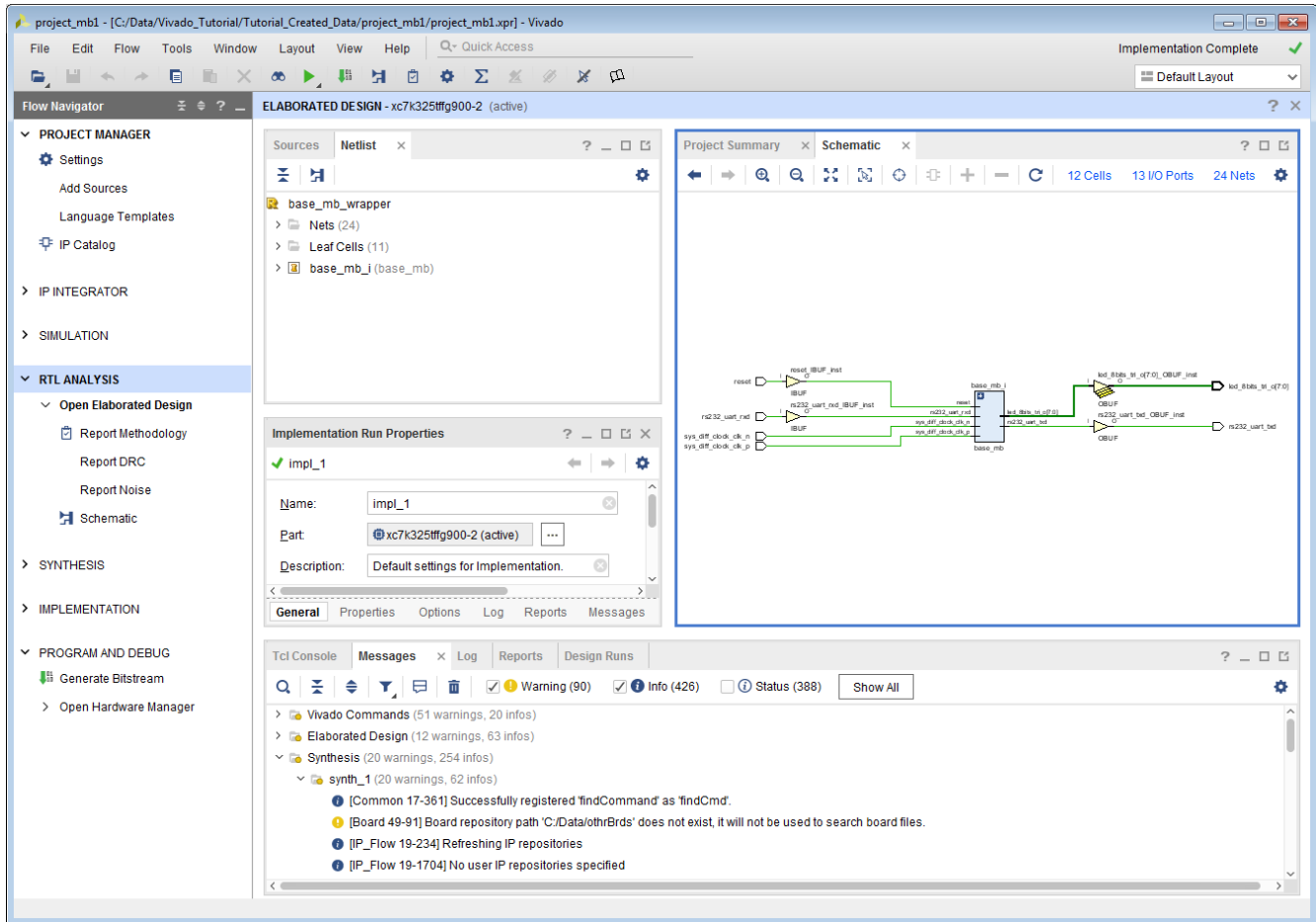


Figure 3-16: Elaborated RTL Design View Layout

Opening a Synthesized Design

When you open a synthesized design, the Vivado Design Suite opens the synthesized netlist and applies physical and timing constraints against a target part. The different elements of the synthesized design are loaded into memory, and you can analyze and modify these elements as needed to complete the design. You can save updates to the constraints files, netlist, debug cores, and configuration.

In a synthesized design, you can perform many design tasks, including early timing, power, and utilization estimates that can help you determine if your design is converging on desired targets. You can explore the design in a variety of ways using the windows in the Vivado IDE. Objects are always cross-selected in all other windows. You can cross probe to problem lines in the RTL files from various windows, including the Messages, Schematic, Device, Package, and Find windows. The Schematic window allows you to interactively explore the logic interconnect and hierarchy. You can also apply timing constraints and perform further timing analysis. In addition, you can interactively define physical constraints for I/O ports, floorplanning, or design configuration. For more information, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [\[Ref 21\]](#).

Using the I/O planning capabilities of the Vivado IDE, you can interactively configure and assign I/O ports in the synthesized design and run DRCs. Select the **Run DRC** command to invoke a comprehensive set of DRCs to identify logic issues. For more information, see this [link](#) in the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [\[Ref 15\]](#) and see this [link](#) in *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [\[Ref 21\]](#).

You can configure and implement debug core logic in the synthesized design to support test and debug of the programmed device. In the Schematic or Netlist windows, interactively select signals for debug. Debug cores are then configured and inserted into the design. The core logic and interconnect is preserved through synthesis updates of the design when possible. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [\[Ref 22\]](#).

To open a synthesized design, use one of the following methods:

- In the Synthesis section of the Flow Navigator, select **Open Synthesized Design**.
- In the Flow Navigator, right-click **Synthesis**, and select **New Synthesized Design** from the popup menu.
- Select **Flow > Open Synthesized Design**.
- In the Design Runs view, double-click the run name.

Figure 3-17 shows the default view layout for an open synthesized design.

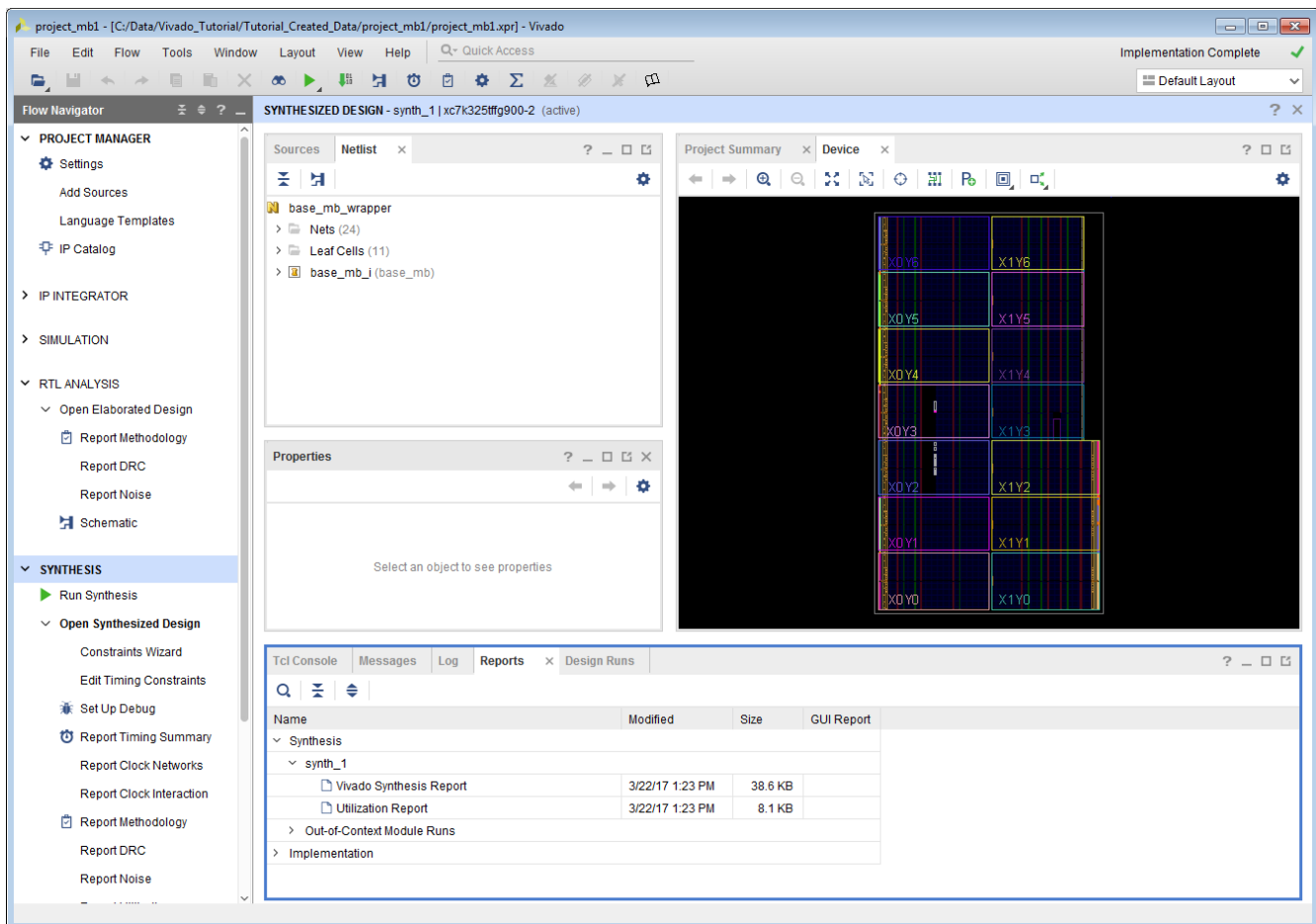



Figure 3-17: Synthesized Design View Layout

Opening an Implemented Design

When you open an implemented design in the Flow Navigator, the Vivado IDE opens the implemented netlist and applies the physical and timing constraints used during implementation, placement, and routing results against the implemented part. The placed logic and routed connections of the implemented design are loaded into memory, and you can analyze and modify the elements as needed to complete the design. You can save updates to the constraints files, netlist, implementation results, and design configuration. Because the Vivado IDE allows for multiple implementation runs, you can select any completed implementation run to open the implemented design.

In an implemented design, you can perform many design tasks, including timing analysis, power analysis, and generation of utilization statistics, which can help you determine if your design converged on desired performance targets. You can explore the design in a variety of ways using the windows in the Vivado IDE. Selected objects are always cross-selected in all related windows. You can cross probe to lines in the source RTL files from various windows, including the Messages, Schematic, Device, Package, and Find windows. The Schematic window allows you to interactively explore the logic interconnect and hierarchy. You can also apply timing constraints and perform further timing analysis. In addition, you can interactively apply floorplanning or design configuration constraints and save the constraints for future runs. For more information, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 21].

In the Device window, you can explore the placement or the routing results by toggling the **Routing Resources** button . As you zoom, the amount of detail shown in the Device window increases. You can interactively alter placement and routing as well as design configuration, such as look-up table (LUT) equations and random access memory (RAM) initialization. You can also select results in the Device or Schematic windows to cross probe back to problem lines in the RTL files. In the Schematic window, you can interactively explore the logic interconnect and hierarchy. For more information, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 21].

To open an implemented design, use one of the following methods:

- In the Implementation section of the Flow Navigator, click **Open Implemented Design**.
- Select **Flow > Open Implemented Design**.
- In the Design Runs view, double-click the run name.



TIP: Because the Flow Navigator reflects the state of the active run, the **Open Implemented Design** command might be disabled or greyed out if the active run is not implemented. In this case, use the **Implementation** popup menu in the Flow Navigator to open an implemented design from any of the completed implementation runs.

Figure 3-18 shows the default layout view for an open implemented design.

Note: The Device window might display placement only or routing depending on the state the window was in when it was last closed. In the Device window, click the **Routing Resources** button to toggle the view to display only placement or routing.

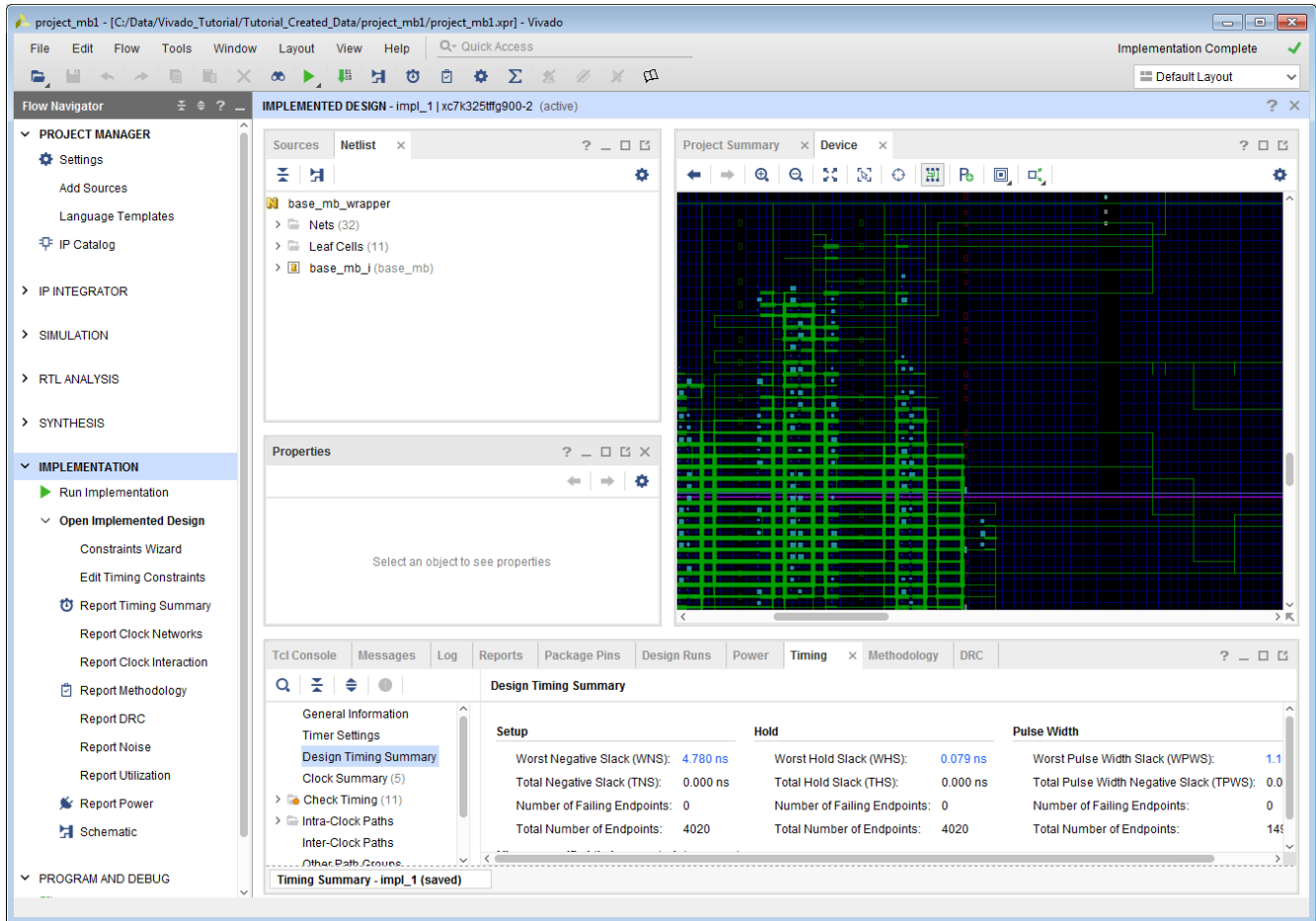


Figure 3-18: Implemented Design View Layout

Updating Out-of-Date Designs

During the design process, source files or constraints often require modification. The Vivado IDE manages the dependencies of these files and indicates when the design data in the current design is out of date. For example, changing settings, such as the target part or active constraint set, can make a design out of date. As source files, netlists, or implementation results are updated, an out-of-date message is displayed in the design window banner of an open synthesized or implemented design to indicate that the run is out of date (Figure 3-19). Click the associated more info link to view which aspects of the design are out of date.

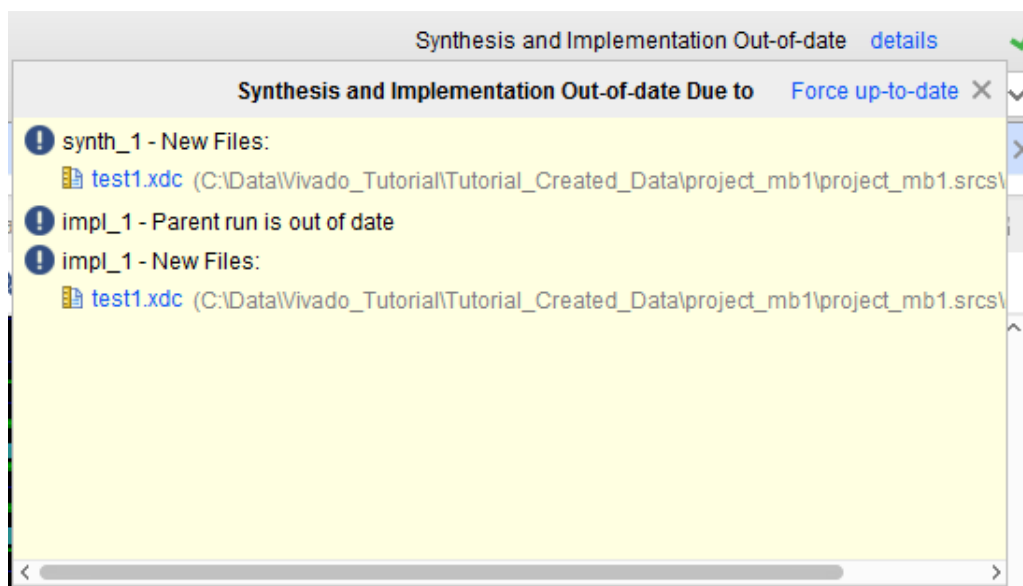


Figure 3-19: Design Out-of-Date and Reload Banner

From the design window banner, use any of the following actions to resolve an out-of-date design:

- Click **More Info**, and click the **Force up-to-date** link in the Out-of-Date Due to window that appears.

Force up-to-date resets the `NEEDS_REFRESH` property on the active synthesis or implementation runs as needed to force the runs into an up-to-date state. The associated Tcl command is shown in the following sample code:

```
set_property NEEDS_REFRESH false [get_runs synth_2]
```

Note: Use this command to force designs up to date when a minor design change was made, and you do not want to refresh the design.

- Click **Reload** to refresh the in-memory view of the current design, eliminating any unsaved changes you made to the design data.
- Click **Close Design** to close the out-of-date design.

Using View Layouts to Perform Design Tasks

When a design is open, several default view layouts (Figure 3-20) are provided to enable you to more easily work on specific design tasks, such as I/O planning, floorplanning, and debug configuration. Changing view layouts simply alters the windows that are displayed, which enables you to focus on a particular design task. You can also create custom view layouts using the **Save Layout As** command.

Note: Default view layouts are available only when a design is open.

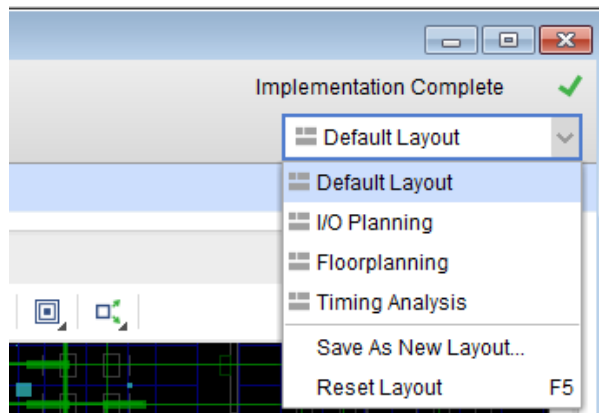


Figure 3-20: Selecting a View Layout

Saving Design Changes

In the Vivado IDE, you interactively edit the active design in memory. It is important to save the design when you make changes to constraints, netlists, and design parameters, such as power analysis characteristics, hardware configuration mode parameters, and debug configuration. For changes made while interactively editing an open design, you can save the changes either back to your original XDC constraint files or to a new constraint set as described in the following sections.

Saving Changes to Original XDC Constraint Files

To save any changes you made to your design data back to your original XDC constraint files, select **File > Constraints > Save**, or click the **Save Constraints** button .

The Save Constraints command saves any changes made to the constraints, debug cores and configuration, and design configuration settings made in the open design. The Vivado IDE attempts to maintain the original file format as much as possible. Additional constraints are added at the end of the file. Changes to existing constraints remain in their original file locations.

Saving Changes to a New Constraint Set

To save changes to the design to a new constraint set, select **File > Constraints > Save As** to create a new constraint file.

This saves any changes while preserving your original constraints source files. The new constraint set includes all design constraints, including all changes. This is one way to maintain your original XDC source files. You can also make the new constraint set the active constraint set, so that it is automatically applied to the next run or when opening designs.

Closing Designs

You can close designs to reduce the number of designs in memory and to prevent multiple locations where sources can be edited. In some cases, you are prompted to close a design prior to changing to another design representation. To close individual designs, do either of the following:

- In the design title bar, click the close button (**X**).
- In the Flow Navigator, right-click the design, and select **Close**.

Analyzing Implementation Results

When you open an implemented design, placement and routing results are displayed in the Device window. In the Timing Results window, you can select timing paths to highlight the placement and routing for the selected path in the Device window. You can also interactively edit placement and routing to achieve design goals and change design characteristics, such as LUT equations, RAM initialization, and phase-locked loop (PLL) configuration. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 19].



IMPORTANT: *Changes are made on the in-memory version of the implemented design only. Resetting the run causes changes to be lost. To save the changes, use the **Save Checkpoint** command, as described in [Saving Design Changes to Design Checkpoints in Chapter 4](#).*

Running Timing Analysis

The Vivado IDE provides a graphical way to configure and view timing analysis results. You can experiment with various types of timing analysis parameters using **Tools > Timing** commands. You can use the Clock Networks and Clock Interaction report windows to view clock topology and relationships. You can also use the Slack Histogram window to see an overall view of the design timing performance. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 21].

In addition, the Vivado IDE has many timing analysis options available through the Tcl Console and SDC constraint options. Many standard report Tcl commands are available to

provide information about the clock structure, logic relationships, and constraints applied to your design. For more information, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 6], or type `help report_*`.

Running Reports: DRC, Power, Utilization Analysis

The Vivado IDE provides a graphical way to configure and view power, utilization, and DRC analysis results. The `report_power` command lets you experiment with power parameters and quickly estimate power at any stage of the design. The `report_utilization` command lets you analyze the utilization statistics of various types of device resources. The `report_design_analysis` command lets you analyze critical path characteristics and the complexity of the design to help identify and analyze problem areas that are prone to routing congestion and timing closure issues. The `report_drc` command let you configure and run a comprehensive set of DRCs to identify problems that must be solved prior to generating the bitstream for the design.

In the Vivado IDE, report results are provided with links to select problem areas or offending objects. In addition, many reports can write an RPX file to save the report results in an interactive report file that can be reloaded into memory, with links to design objects. Reloading the report reconnects the object links so that cross-selection between the report in the Vivado IDE and the design is enabled. For more information, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 21], or refer to the `report_xxx` commands in the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 6].

Device Programming, Hardware Verification, and Debugging

In the Vivado IDE, the Vivado logic analyzer includes many features to enable verification and debugging of the design. You can configure and implement IP debug cores, such as the Integrated Logic Analyzer (ILA) and Debug Hub core, in either an RTL or synthesized netlist. Opening the synthesized or implemented design in the Vivado IDE enables you to select and configure the required probe signals into the cores. You can launch the Vivado logic analyzer on any run that has a completed bitstream file for performing interactive hardware verification. In addition, you can create programming bitstream files for any completed implementation run. Bitstream file generation options are configurable. Launch the Vivado device programmer to configure and program the part. You can launch the Vivado logic analyzer directly from the Vivado IDE for further analysis of the routing or device resources. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 22].

Implementing Engineering Changes (ECOs) for Debugging

Engineering change orders (ECOs) are modifications to an implemented design. The Vivado Design Suite provides an ECO flow that lets you implement changes to an existing design checkpoint (DCP) and generate updated bitstream files. After implementing an ECO on the design, you might also need to modify, add, or delete debug cores or probes to the implemented design. Refer to this [link](#) in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 22] for information on the debug ECO flow.

Using Project Mode Tcl Commands

Table 3-2 shows the basic Project Mode Tcl commands that control project creation, implementation, and reporting.



TIP: The best way to understand the Tcl commands involved in a design task is to run the command in the Vivado IDE and inspect the syntax in the Tcl Console or the `vivado.jou` file.

Table 3-2: Basic Project Mode Tcl Commands

Command	Description
<code>create_project</code>	Creates the Vivado Design Suite project. Arguments include project name and location, design top module name, and target part.
<code>add_files</code>	<p>Adds source files to the project. These include Verilog (.v), VHDL (.vhd or .vhdl), SystemVerilog (.sv), IP and System Generator modules (.xco or .xci), IP Integrator subsystems (.bd), and XDC constraints (.xdc or .sdc).</p> <p>Individual files can be added, or entire directory trees can be scanned for legal sources and automatically added to the project.</p> <p>Note: The .xco file is no longer supported in UltraScale device designs.</p>
<code>set_property</code>	Used for multiple purposes in the Vivado Design Suite. For projects, it can be used to define VHDL libraries for sources, simulation-only sources, target constraints files, tool settings, and so forth.
<code>import_files</code>	Imports the specified files into the current file set, effectively adding them into the project infrastructure. It is also used to assign XDC files into constraints sets.
<code>launch_runs</code> <code>launch_runs -to_step</code>	Starts either synthesis or implementation and bitstream generation. This command encompasses the individual implementation commands as well as the standard reports generated after the run completes. It is used to launch all of the steps of the synthesis or implementation process in a single command, and to track the tools progress through that process. The <code>-to_step</code> option is used to launch the implementation process, including bitstream generation, in incremental steps.
<code>wait_on_run</code>	Ensures the run is complete before processing the next commands in a Tcl script.
<code>open_run</code>	Opens either the synthesized design or implemented design for reporting and analysis. A design must be opened before information can be queried using Tcl for reports, analysis, and so forth.
<code>close_design</code>	Closes the in-memory design.
<code>start_gui</code> <code>stop_gui</code>	Opens or closes the Vivado IDE with the current design in memory.

Note: This document is not a complete reference for the available Tcl commands. Instead, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 6].

Project Mode Tcl Script Examples

The following examples show a Tcl script for an RTL project and a netlist project. The first example script, `run_bft_kintex7_project.tcl`, is available in the Vivado Design Suite installation at:

```
<install_dir>/Vivado/2016.2/examples/Vivado_Tutorial
```

You can source these scripts from the Vivado Tcl shell, or the Tcl Console inside of the Vivado IDE.

RTL Project Tcl Script

```
# run_bft_kintex7_project.tcl
# BFT sample design
#
# NOTE: Typical usage would be "vivado -mode tcl -source run_bft_kintex7_project.tcl"
# To use -mode batch comment out the "start_gui" and "open_run impl_1" to save time
#
create_project project_bft ./Tutorial_Created_Data/project_bft -part
xc7k70tfbg484-2
add_files {./Sources/hdl/FifoBuffer.v ./Sources/hdl/async_fifo.v
./Sources/hdl/bft.vhdl}
add_files -fileset sim_1 ./Sources/hdl/bft_tb.v
add_files ./Sources/hdl/bftLib
set_property library bftLib [get_files {./Sources/hdl/bftLib/round_4.vhdl \
./Sources/hdl/bftLib/round_3.vhdl ./Sources/hdl/bftLib/round_2.vhdl
./Sources/hdl/bftLib/round_1.vhdl \
./Sources/hdl/bftLib/core_transform.vhdl ./Sources/hdl/bftLib/bft_package.vhdl}]
import_files -force
import_files -fileset constrs_1 -force -norecurse ./Sources/bft_full_kintex7.xdc
# Mimic GUI behavior of automatically setting top and file compile order
update_compile_order -fileset sources_1
update_compile_order -fileset sim_1
# Launch Synthesis
launch_runs synth_1
wait_on_run synth_1
open_run synth_1 -name netlist_1
# Generate a timing and power reports and write to disk
report_timing_summary -delay_type max -report_unconstrained -check_timing_verbose \
-max_paths 10 -input_pins -file ./Tutorial_Created_Data/project_bft/syn_timing.rpt
report_power -file ./Tutorial_Created_Data/project_bft/syn_power.rpt
# Launch Implementation
launch_runs impl_1 -to_step write_bitstream
wait_on_run impl_1
# Generate a timing and power reports and write to disk
# comment out the open_run for batch mode
open_run impl_1
report_timing_summary -delay_type min_max -report_unconstrained
-check_timing_verbose \
-max_paths 10 -input_pins -file ./Tutorial_Created_Data/project_bft/imp_timing.rpt
report_power -file ./Tutorial_Created_Data/project_bft/imp_power.rpt
# comment out the for batch mode
start_gui
```

Netlist Project Tcl Script

```
# Kintex-7 Netlist Example Design
#
# STEP#1: Create Netlist Project, add EDIF sources, and add constraints
#
create_project -force project_K7_netlist
./Tutorial_Created_Data/project_K7_netlist/ -part xc7k70tfbg676-2
# Property required to define Netlist project
set_property design_mode GateLvl [current_fileset]
add_files {./Sources/netlist/top.edif}
import_files -force
import_files -fileset constrs_1 -force ./Sources/top_full.xdc

#
# STEP#2: Configure and Implementation, write bitstream, and generate reports
#
launch_runs impl_1
wait_on_run impl_1
launch_runs impl_1 -to_step write_bitstream
wait_on_run impl_1
open_run impl_1
report_timing_summary -delay_type min_max -report_unconstrained
-check_timing_verbose \
-max_paths 10 -input_pins -file
./Tutorial_Created_Data/project_K7_netlist/imp_timing.rpt
report_power -file ./Tutorial_Created_Data/project_K7_netlist/imp_power.rpt
#
# STEP#3: Start IDE for design analysis
#
start_gui
```


Using Non-Project Mode

Overview

This chapter highlights the differences between Non-Project Mode and Project Mode. To fully understand Non-Project Mode in the Vivado[®] Design Suite, you should be familiar with Project Mode as described in [Chapter 3, Using Project Mode](#).

In Non-Project Mode, you use Tcl commands to compile a design through the entire flow. In this mode, an in-memory project is created to let the Vivado tools manage various properties of a design, but the project file is not written to disk, and the project status is not preserved.



TIP: An in-memory project is also generated in Non-Project Mode for the Vivado tool to use. However, it is not preserved as part of the design.

Tcl commands provide the flexibility and power to set up and run your designs and perform analysis and debugging. Tcl commands can be run in batch mode, from the Vivado[®] Design Suite Tcl shell, or through the Vivado IDE Tcl Console. Non-Project Mode enables you to have full control over each design flow step, but you must manually manage source files, reports, and intermediate results known as design checkpoints. You can generate a variety of reports, perform DRCs, and write design checkpoints at any stage of the implementation process.

Unlike Project Mode, Non-Project Mode does not include features such as runs infrastructure, source file management, or design state reporting. Each time a source file is updated, you must rerun the design manually. Default reports and intermediate files are not created automatically in this mode. However, you can create a wide variety of reports and design checkpoints as needed using Tcl commands. In addition, you can still access the GUI-based design analysis and constraints assignment features of the Vivado IDE. You can open either the current design in memory or any saved design checkpoint in the Vivado IDE.

When you launch the Vivado IDE in Non-Project Mode, the Vivado IDE does not include Project Mode features such as the Flow Navigator, Project Summary, or Vivado IP catalog. In Non-Project Mode, you cannot access or modify synthesis or implementation runs in the Vivado IDE. However, if the design source files reside in their original locations, you can cross probe to design objects in the different windows of the Vivado IDE. For example, you

can select design objects and then use the **Go To Instantiation**, **Go To Definition**, or **Go To Source** commands to open the associated RTL source file and highlight the appropriate line.



IMPORTANT: *Some of the features of Project Mode, such as source file and run results management, saving design and tool configuration, design status, and IP integration, are not available in Non-Project Mode.*

You must write reports or design checkpoints to save the in-memory design as it progresses. The design checkpoint (DCP) refers to a file that is an exact representation of the in-memory design. You can save a design checkpoint after each step in the design flow, such as post synthesis, post optimization, post placement. The DCP file can be read back into the Vivado Design Suite to restore the design to the state captured in the checkpoint file.

You can also open a DCP in the Vivado IDE to perform interactive constraints assignment and design analysis. Because you are viewing the active design in memory, any changes are automatically passed forward in the flow. You can also save updates to new constraint files or design checkpoints for future runs.

While most Non-Project Mode features are also available in Project Mode, some Project Mode features are not available in Non-Project Mode. These features include source file and run results management, saving design and tool configuration, design status, and IP integration. On the other hand, you can use Non-Project mode to skip certain processes, thereby reducing the memory footprint of the design, and saving disk space related to projects.

Non-Project Mode Advantages

Non-Project Mode enables you to have full control over each design flow step. You can take advantage of a compile-style design flow.

In this mode, you manage your design manually, including:

- Manage HDL Source files, constraints, and IP
- Manage dependencies
- Generate and store synthesis and implementation results

The Vivado Design Suite includes an entire suite of Vivado Tcl commands to create, configure, implement, analyze, and manage designs as well as IP. In Non-Project Mode, you can use Tcl commands to do the following:

- Compile a design through the entire flow
- Analyze the design and generate reports

Reading Design Sources

When using Non-Project Mode, the various design sources are read into the in-memory design for processing by the implementation tools. Each type of Vivado Design Suite source file has a `read_*` Tcl command to read the files, such as `read_verilog`, `read_vhdl`, `read_ip`, `read_edif`, or `read_xdc`. Sources must be read each time the Tcl script or interactive flow is started.



TIP: *Because there is no project structure to add the files or import the files into, you should not use the `add_files` or `import_files` Tcl commands to add files to a non-project based design.*

Managing Source Files

In Non-Project Mode, you manage source files manually by reading the files into the in-memory design in a specific order. This gives you full control over how to manage the files and where files are located. Sources can be read from any network accessible location. Sources with read-only permissions are processed accordingly.

Working with a Revision Control System

Many design teams use source management systems to store various design configurations and revisions. There are multiple commercially available systems, such as Revision Control System (RCS), Concurrent Versions System (CVS), Subversion (SVN), ClearCase, Perforce, Git, BitKeeper, and many others. The Vivado tools can interact with all such systems. The Vivado Design Suite uses and produces files throughout the design flow that you may want to manage under revision control.

Working with revision control software is simple when using the Non-Project mode. The designer checks out the needed source files into a local directory structure. The sources are then instantiated into a top-level design to create the design. New source files might also need to be created and read into the design using various `read_*` Tcl commands. The design files are passed to the Vivado synthesis and implementation tools. However, the source files remain in their original locations. The checked-out sources can be modified interactively, or with Tcl commands during the design session using appropriate code editors. Source files are then checked back into the source control system as needed. Design results, such as design checkpoints, analysis reports, and bitstream files, can also be checked in for revision management. For more information on working with revision control software, see [Chapter 5, Source Management and Revision Control Recommendations](#).



VIDEO: *For information on best practices when using revision control systems with the Vivado tools, see the [Vivado Design Suite QuickTake Video: Using Vivado Design Suite with Revision Control](#).*

Using Third-Party Synthesized Netlists

The Vivado Design Suite supports implementation of synthesized netlists, such as when using a third-party synthesis tool. The external synthesis tool generates a Verilog or EDIF netlist and a constraints file, if applicable. These netlists can be used standalone or mixed with RTL files in either Project Mode or Non-Project Mode.

Working with IP and IP Subsystems

In Non-Project Mode, output products must be generated for the IP or block designs prior to launching the top-level synthesis. You can configure IP to use RTL sources and constraints, or use the OOC netlist from a synthesized design checkpoint as the source in the top-level design. The default behavior is to generate an OOC design checkpoint for each IP.

In Non-Project Mode, you can add IP to your design using any of the following methods:

- IP generated using the Vivado IP catalog (.xci format or .xcix format for core container)

If the out-of-context design checkpoint file exists in the IP directory, it is used for implementation and a black box is inserted for synthesis. If a design checkpoint file does not exist in the IP directory, the RTL and constraints sources are used for global synthesis and implementation.

- Use Tcl commands to configure and generate the IP or block design.

Using Tcl ensures that the IP is configured, generated, and synthesized with each run.



IMPORTANT: *When using IP in Project Mode or Non-Project Mode, always use the XCI file not the DCP file. This ensures that IP output products are used consistently during all stages of the design flow. If the IP was synthesized out-of-context and already has an associated DCP file, the DCP file is automatically used and the IP is not re-synthesized. For more information, see this [link](#) in the Vivado Design Suite User Guide: Designing with IP (UG896) [Ref 12].*

For more information, see this [link](#) in the Vivado Design Suite User Guide: Designing with IP (UG896) [Ref 12], or this [link](#) in the Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator (UG994) [Ref 30].

Running Logic Simulation

The Vivado simulator, integrated with the Vivado IDE, allows you to simulate the design, and view signals in the waveform viewer, and examine and debug the design as needed. The Vivado simulator is a fully integrated mixed-mode simulator with analog waveform display capabilities. Using the Vivado simulator, you can perform behavioral and structural simulation of designs and full timing simulation of implemented designs.

You can also use third-party simulators to write the Verilog, VHDL netlists, and SDF format files from the open design. You can launch the Mentor Graphics ModelSim and Questa simulators from the Vivado IDE. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Running Logic Synthesis and Implementation

In Non-Project Mode, each implementation step is launched with a configurable Tcl command, and the design is compiled in memory. The implementation steps must be run in a specific order, as shown in the [Non-Project Mode Tcl Script Example](#). Optionally, you can run steps such as `power_opt_design` or `phys_opt_design` as needed. Instead of run strategies, which are only supported in Project Mode, you can use various commands to control the tool behavior. For more information, see the *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 19].

It is important to write design checkpoints after critical design steps for design analysis and constraints definition. With the exception of generating a bitstream, design checkpoints are not intended to be used as starting points to continue the design process. They are merely snapshots of the design for analysis and constraint definition.



TIP: After each design step, you can launch the Vivado IDE to enable interactive graphical design analysis and constraints definition on the active design, as described in [Performing Design Analysis Using the Vivado IDE](#).

Generating Reports

With the exception of the `vivado.log` and `vivado.jou` reports, reports must be generated manually with a Tcl command. You can generate various reports at any point in the design process. For more information, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 6] or *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 19].

Using Design Checkpoints

Design checkpoints enable you to take a snapshot of your design in its current state. The current netlist, constraints, and implementation results are stored in the design checkpoint. Using design checkpoints, you can:

- Restore your design if needed
- Perform design analysis
- Define constraints
- Proceed with the design flow

You can write design checkpoints at different points in the flow. It is important to write design checkpoints after critical design steps for design analysis and constraints definition. You can read design checkpoints to restore the design, which might be helpful for debugging issues. The design checkpoint represents a full save of the design in its current implementation state. You can run the design through the remainder of the flow using Tcl commands. However, you cannot add new sources to the design.

Note: You can also use the `write_checkpoint <file_name>.dcp` and `read_checkpoint <file_name>.dcp` Tcl commands to write and read design checkpoints. To view a checkpoint in the Vivado IDE, use the `open_checkpoint <file_name>.dcp` Tcl command. For more information, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 6].

Performing Design Analysis Using the Vivado IDE

In Non-Project Mode, you can launch the Vivado IDE after any design step to enable interactive graphical design analysis and constraints definition on the active design.

Opening the Vivado IDE From the Active Design

When working in Non-Project Mode, use the following commands to open and close the Vivado IDE on the active design in memory:

- `start_gui` opens the Vivado IDE with the active design in memory.
- `stop_gui` closes the Vivado IDE and returns to the Vivado Design Suite Tcl shell.



CAUTION! *If you exit the Vivado Design Suite from the GUI, the Vivado Design Suite Tcl shell closes and does not save the design in memory. To return to the Vivado Design Suite Tcl shell with the active design intact, use the `stop_gui` Tcl command rather than the `exit` command.*

After each stage of the design process, you can open the Vivado IDE to analyze and operate on the current design in memory (Figure 4-1). In Non-Project Mode, some of the project features are not available in the Vivado IDE, such as the Flow Navigator, Project Summary, source file access and management, and runs. However, many of the analysis and constraint modification features are available in the **Tools** menu.



IMPORTANT: *Be aware that any changes made in the Vivado IDE are made to the active design in memory and are automatically applied to downstream tools.*

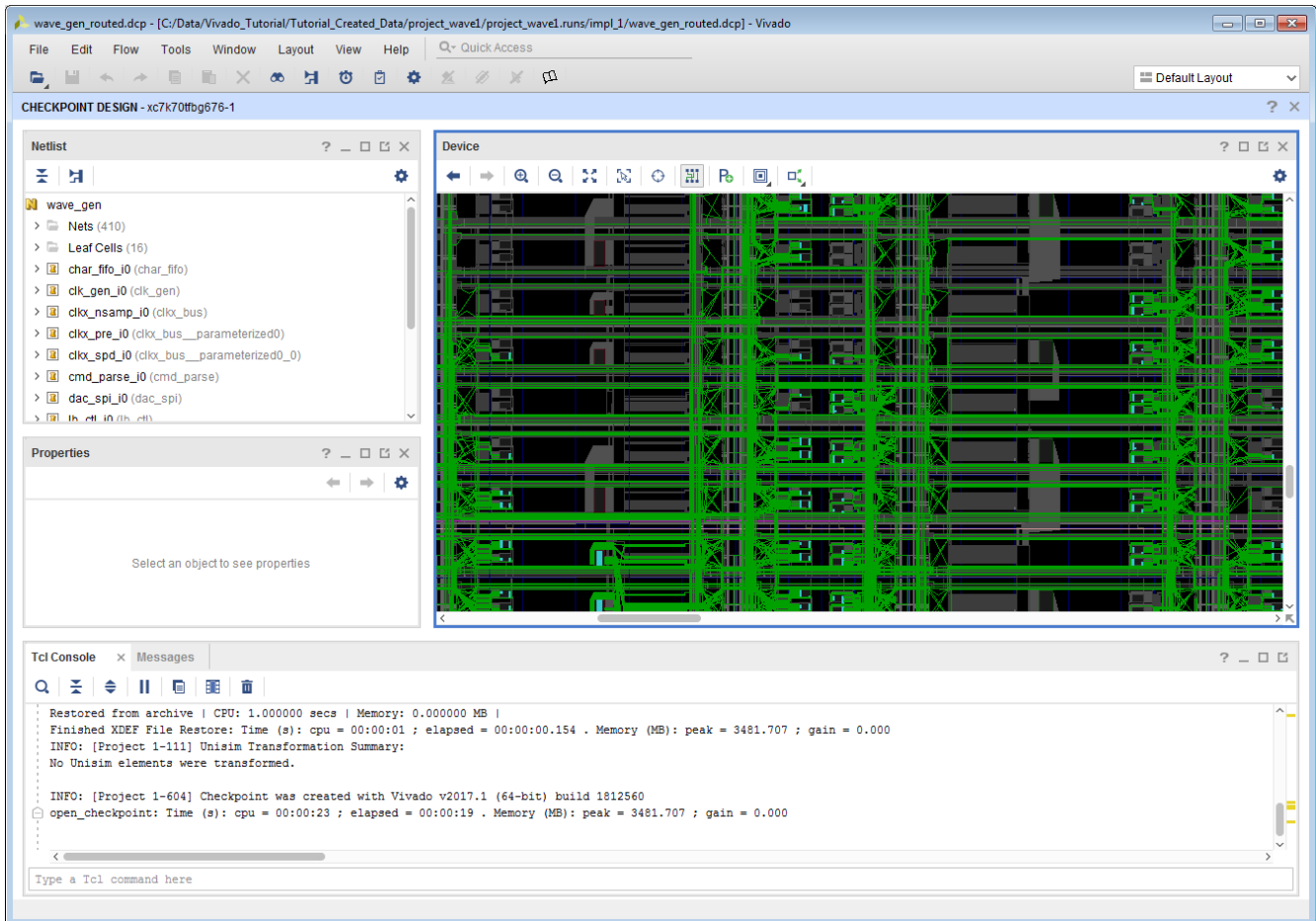


Figure 4-1: Opening Vivado IDE with the Active Design

Saving Design Changes to the Active Design

Because you are actively editing the design in memory, changes are automatically passed to downstream tools for the remainder of the Vivado IDE Tcl session. This enables you to reflect the changes in the active design and to save the changes for future attempts. Select **File > Export > Export Constraints** to save constraints changes for future use. You can use this command to write a new constraints file or override your original file.

Note: When you export constraints, the `write_xdc` Tcl command is run. For more information, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 6].

Opening Design Checkpoints in the Vivado IDE

You can use the Vivado IDE to analyze designs saved as design checkpoints. You can run a design in Non-Project Mode using Tcl commands (`synth_design`, `opt_design`, `power_opt_design`, `place_design`, `phys_opt_design`, and `route_design`), store the design at any stage, and read it in a Vivado IDE session. You can start with a routed design, analyze timing, adjust placement to address timing problems, and save your work for later, even if the design is not fully routed. The Vivado IDE view banner displays the open design checkpoint name.

Saving Design Changes to Design Checkpoints

You can open, analyze, and save design checkpoints. You can also save changes to a new design checkpoint:

- Select **File > Checkpoint > Save** to save changes made to the current design checkpoint.
- Select **File > Checkpoint > Write** to save the current state of the design checkpoint to a new design checkpoint.

Using Non-Project Mode Tcl Commands

Table 4-1 shows the basic Non-Project Mode Tcl commands. When using Non-Project Mode, the design is compiled using `read_verilog`, `read_vhdl`, `read_edif`, `read_ip`, `read_bd`, and `read_xdc` type commands. The sources are ordered for compilation and passed to synthesis. For information on using the Vivado Design Suite Tcl shell or using batch Tcl scripts, see [Working with Tcl in Chapter 2](#).

Note: This document is not a complete reference for the available Tcl commands. Instead, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 6] and *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894) [Ref 10].

Table 4-1: Basic Non-Project Mode Tcl Commands

Command	Description
<code>read_edif</code>	Imports an EDIF or NGC netlist file into the Design Source fileset of the current project.
<code>read_verilog</code>	Reads the Verilog (.v) and System Verilog (.sv) source files for the Non-Project Mode session.
<code>read_vhdl</code>	Reads the VHDL (.vhd or .vhdl) source files for the Non-Project Mode session.
<code>read_ip</code>	Reads existing IP (.xci or .xco) project files for the Non-Project Mode session. For Vivado IP (.xci), the design checkpoint (.dcp) synthesized netlist is used to implement the IP if the netlist is in the IP directory. If not, the IP RTL sources are used for synthesis with the rest of the top-level design. The .ngc netlist is used from the .xco IP project. Note: The .xco file is no longer supported in UltraScale device designs.

Table 4-1: Basic Non-Project Mode Tcl Commands (Cont'd)

Command	Description
<code>read_checkpoint</code>	Loads a design checkpoint into the in-memory design.
<code>read_xdc</code>	Reads the .sdc or .xdc format constraints source files for the Non-Project Mode session.
<code>read_bd</code>	Reads existing IP Integrator block designs (.bd) for the Non-Project session.
<code>set_param</code> <code>set_property</code>	Used for multiple purposes. For example, it can be used to define design configuration, tool settings, and so forth.
<code>link_design</code>	Compiles the design for synthesis if netlist sources are used for the session.
<code>synth_design</code>	Launches Vivado synthesis with the design top module name and target part as arguments.
<code>opt_design</code>	Performs high-level design optimization.
<code>power_opt_design</code>	Performs intelligent clock gating to reduce overall system power. This is an optional step.
<code>place_design</code>	Places the design.
<code>phys_opt_design</code>	Performs physical logic optimization to improve timing or routability. This is an optional step.
<code>route_design</code>	Routes the design.
<code>report_*</code>	Runs a variety of standard reports, which can be run at different stages of the design process.
<code>write_bitstream</code>	Generates a bitstream file and runs DRCs.
<code>write_checkpoint</code>	Saves the design at any point in the flow. A design checkpoint consists of the netlist and constraints with any optimizations at that point in the flow as well as implementation results.
<code>start_gui</code> <code>stop_gui</code>	Opens or closes the Vivado IDE with the current design in memory.

Non-Project Mode Tcl Script Example

The following example shows a Tcl script for the BFT sample design included with the Vivado Design Suite. This example shows how to use the design checkpoints for saving the database state at various stages of the flow and how to manually generate various reports. This example script, `run_bft_kintex7_project.tcl`, is available in the Vivado Design Suite installation at:

```
<install_dir>/Vivado/2019.2/examples/Vivado_Tutorial
```

You can source the script from the Vivado Tcl shell, or the Tcl Console inside of the Vivado IDE.

```
# run_bft_kintex7_batch.tcl
# bft sample design
# A Vivado script that demonstrates a very simple RTL-to-bitstream non-project batch
flow
```

```
#
# NOTE: typical usage would be "vivado -mode tcl -source run_bft_kintex7_batch.tcl"
#
# STEP#0: define output directory area.
#
set outputDir ./Tutorial_Created_Data/bft_output
file mkdir $outputDir
#
# STEP#1: setup design sources and constraints
#
read_vhdl -library bftLib [ glob ./Sources/hdl/bftLib/*.vhdl ]
read_vhdl ./Sources/hdl/bft.vhdl
read_verilog [ glob ./Sources/hdl/*.v ]
read_xdc ./Sources/bft_full_kintex7.xdc
#
# STEP#2: run synthesis, report utilization and timing estimates, write checkpoint
design
#
synth_design -top bft -part xc7k70tfbg484-2
write_checkpoint -force $outputDir/post_synth
report_timing_summary -file $outputDir/post_synth_timing_summary.rpt
report_power -file $outputDir/post_synth_power.rpt
#
# STEP#3: run placement and logic optimization, report utilization and timing
estimates, write checkpoint design
#
opt_design
place_design
phys_opt_design
write_checkpoint -force $outputDir/post_place
report_timing_summary -file $outputDir/post_place_timing_summary.rpt
#
# STEP#4: run router, report actual utilization and timing, write checkpoint design,
run drc, write verilog and xdc out
#
route_design
write_checkpoint -force $outputDir/post_route
report_timing_summary -file $outputDir/post_route_timing_summary.rpt
report_timing -sort_by group -max_paths 100 -path_type summary -file
$outputDir/post_route_timing.rpt
report_clock_utilization -file $outputDir/clock_util.rpt
report_utilization -file $outputDir/post_route_util.rpt
report_power -file $outputDir/post_route_power.rpt
report_drc -file $outputDir/post_imp_drc.rpt
write_verilog -force $outputDir/bft_impl_netlist.v
write_xdc -no_fixed_only -force $outputDir/bft_impl.xdc
#
# STEP#5: generate a bitstream
#
write_bitstream -force $outputDir/bft.bit
```

Source Management and Revision Control Recommendations

Interfacing with Revision Control Systems

The methodologies for source management and revision control can vary depending on user and company preference, as well as the software used to manage revision control. This section describes some of the fundamental methodology choices that design teams need to make to manage their active design projects. Specific recommendations on using the Vivado® Design Suite with revision control systems are provided later in this section. Throughout this section, the term “manage” refers to the process of checking source versions in and out using a revision control system.

Many design teams use source management systems to manage various design configurations and revisions. There are many systems available, such as Git, RCS, CVS, Subversion, ClearCase, Perforce, and Bitkeeper. No single system is predominant. The Vivado Design Suite can interact with all of these systems and is intended to work equally well with each while stopping short of direct integration with any individual tool.



VIDEO: For information on best practices when using revision control systems with the Vivado tools, see the [Vivado Design Suite QuickTake Video: Using Vivado Design Suite with Revision Control](#).

Most users follow a methodology in which sources are checked out into a local repository as a staging area for local changes. This is called a *sandbox* or a local working version of the master repository. The purpose of this area is to group and test local changes by one user before pushing them back into the master repository where other users see the latest changes. The sources in this local repository can be modified to complete the design. These modified sources can be checked back into the source control system as new revisions at any time. Design results can also optionally be checked in for revision storage. Many users use a directory structure to store and manage sources and results.

Revision control systems use different mechanisms to update only those sources that have been modified since the last check out.

There is a defined subset of Vivado Design Suite files that should be managed using revision control. There are a number of intermediate files generated by the Vivado Design

Suite that it would be inefficient and unnecessary to manage. The list shown in [Vivado Design Suite Source Types](#) highlights the key files.



RECOMMENDED: *The run script and tool settings should also be checked in for revision control. This information can be extracted into a Tcl script using the `write_project_tcl` command. Because `vivado_init.tcl` is not included in `write_project_tcl`, `vivado_init.tcl` must also be checked in. Checking in all these files enables the design to be recreated using the current sources and tool configuration settings.*

Single User Versus Multi-User Access

Design sources and IP are typically stored in a library or repository that can be referenced by any number of designers working on a project. However, using design sources or IP from a common repository can make it subject to modification by anyone using these files in their design. Any modification to a shared source or IP is going to be incorporated by all other designers referencing that same source. This can have unintended consequences if design specific modifications impact multiple designs.

Multi-user access works best if the design sources and IP are managed by a single designer and shared exactly as is to all other designs. These sources should be read-only, which prevents any one designer from inadvertently updating a common source.

In addition, Vivado Design Suite IP and IP Subsystems target a specific device. Consequently, it is important to consider device compatibility before sharing them between designs.

Using Read-Only Managed Sources Directly

The Vivado Design Suite allows the use of read-only source files and IP to assemble designs. If sources are not expected to change during the process of the design, they can easily be referenced from their managed read-only locations. If sources need to be modified, they can be checked out into the local working area.

The area where Vivado Design Suite is invoked should have write privileges in order to write log and report files, certain caches, and design results.

Using a Local Working Area

A common methodology for design source management is to make a local copy of the managed sources into an active project working area. Designers can make modifications at will in that area to complete their designs. Sources can be checked back into revision control at any milestone event. Note that all your changes are made in the local copy of the file lying deep inside the project directory structure. So, be sure of the file that you are checking back into revision control system.

This method could be applied only for design sources that are expected to change. Static sources could be referenced from their originally managed read-only locations.

Managing Minimum Sets of Sources

Some design teams want to manage the minimum set of design files needed to reproduce the design. For details, see [Minimum Sets of Source Files to Manage](#).

The Vivado Design Suite lets you manage the project with a minimum set of files with the following restrictions:

- The output products for IP and IP Integrator Subsystems need to be regenerated, which require using the same version of software that the IP was originally created with, to get the exact same replica. If all of the IP output products are managed, the IP can be referenced by future software versions.
- It takes significant added time to generate the output products during design compilation.



RECOMMENDED: For best results, Xilinx recommends managing all of the sources listed in the following section, [Recommended Source Files to Manage](#).

Recommended Source Files to Manage

You can recreate a Vivado Design Suite design and associated IP and BD files using a set of source files. To mitigate the risk of recreation using future versions of the Vivado Design Suite and to reduce compile times, Xilinx recommends managing the following source files. Checking in all these files enables the design to be recreated using the current sources and tool configuration settings.

- Scripts
 - Run scripts used to compile the design including any pre and post scripts used when launching runs
 - If modifications were made to the Vivado `vivado_init.tcl` file, it must also be checked in
 - Project recreation scripts created with the `write_project_tcl` command

- Project .xpr files can be used to recreate a design project
- RTL and simulation test benches
 - All RTL-based source files including .include files
 - Simulation test benches and stimulus files
- Constraints
 - All XDC constraint files and Tcl commands used as constraints
- IP
 - The IP source file (.xci)
 - All IP directories and output product files with names intact
 - If using IP core containers, the IP container file (.xcix) with all third-party simulation and synthesis files in the `ip_user_files` and `ip_static_files` subdirectories
 - Any .coe, .csv, .bmm, and .elf files that are used with the IP
- IP Integrator
 - The block design source file (.bd)
 - The entire block design location with all of the IP subdirectories, files, and names intact
- HLS
 - All C sources files
 - Packaged HLS IP for use in Vivado synthesis
 - Scripts
- Vivado Hardware Manager
 - All .bit files needed to program the device
 - The .ltx file containing tool defaults and custom user settings
- Hardware Software Interface
 - The Handoff Design File (.hdf extension) contains all the information required by Xilinx Vitis to create the corresponding software project for your design. The HDF file is created when you export your design either through the `write_hwdef` or `write_sysdef` Tcl command or the **File > Export > Export Hardware** command.
- Documentation
 - Any design related docs, specs, and reports.

Minimum Sets of Source Files to Manage

A Vivado Design Suite design along with its associated IP and BDs can be recreated using a minimum set of source files. There are limitations to this approach, as described in [Managing Minimum Sets of Sources](#).

Following are the minimum set of files needed to recreate the design:

- Scripts
 - Run scripts used to compile the design including any pre and post scripts used when launching runs
 - Project recreation scripts created with the `write_project_tcl` command
 - Project .xpr files can be used to recreate a design project
- RTL and Test benches
 - All RTL based source files including .include files
 - Simulation test benches and stimulus files
- Constraints
 - All XDC constraint files and Tcl commands used as constraints
- IP
 - The .xci files associated for each IP
 - If using IP core containers, the IP core container .xcix file



IMPORTANT: *When working with IP, it is important to manage the original IP customization file (.xci) at minimum, rather than the output netlist (.dcp) for the customized IP. From the XCI file, the IP can be recustomized and regenerated as needed.*

- IP Integrator
 - The BD recreation Tcl command created using the `write_bd_tcl` command
 - The .bd file used for each BD
 - The UI directory for graphics locations for blocks and comments

Vivado Design Suite Source Types

The Vivado Design Suite environment references source files that contain design descriptions. Options in the Vivado Design Suite control how you create, use, and manage the source types. These sources include:

- HDL and netlist files: Verilog (.v), SystemVerilog (.sv), VHDL (.vhd), and EDIF (.edf)
- C based source files (.c)

- Tcl files, run scripts, and vivado_init.tcl (.tcl)
- Logical and Physical Constraints (.xdc)
- IP core files (.xci) and IP core container files (.xcix)
- IP integrator block design files (.bd)
- Design Checkpoint files (.dcp)
- Side files for use by related tools (e.g. "do" files for simulator)
- Block Memory Map files (.bmm, .mmi)
- Executable and Linkable Format files (.elf)
- Coefficient files (.coe)
- Archived projects (.zip)

How Vivado Design Suite Recognizes Source Modifications

The Vivado Design Suite uses the time/date stamps on the majority of the source files to indicate whether the source file has been updated. Taking any action on the various sources that may trigger a new time/date stamp may indicate that the design is out of date and needs to be updated.



IMPORTANT: *Ensure that common revision control actions do not alter the file time/date stamps.*

Some Revision Control Systems might alter the timestamps even if there were no changes to the contents of a file, or they might alter the timestamps based on the time of check-in/check-out. To retain the relative time modification ordering of source and generated products, sort your files in reverse order of the timestamps and then perform check-in/check-out. The following is an indicative command (in Unix) to add files to git with timestamp ordering maintained:

```
find . -type f -print0 | xargs -0 ls-rt | xargs git add
```

When you check files out of a revision control system, they are often time-stamped with the time they are checked out of the repository. This can cause output files for IP and synthesis and implementation runs to appear out-of-date in the Vivado tool due to the changed timestamp, even though the content of the files is not actually out-of-date. This will require the output products to be regenerated, or synthesis and implementation to run again, even though the design is current.

Some revision control systems let you preserve the timestamp on the file as the time it was checked into the system, rather than the time it was checked out. This can prevent the output products and design runs from going out-of-date, and prevent unnecessary iterations. Check your specific revision control system to see if such a feature is available.

Defining a Design Source Directory Structure

To effectively distinguish and manage the various types of design sources, Xilinx recommends creating a directory structure for each active design. The various directories store specific types of sources, as shown in the following figure.

Name	Type	Date modified
IP	File folder	1/22/2015 2:42 PM
HLS	File folder	1/16/2015 3:21 PM
Testbenches	File folder	1/16/2015 3:15 PM
Work_Dir	File folder	1/16/2015 12:56 PM
Doc	File folder	1/16/2015 12:44 PM
Constraints	File folder	1/16/2015 12:30 PM
RTL	File folder	1/16/2015 12:28 PM
IPI-BDs	File folder	1/16/2015 11:54 AM
DSP	File folder	1/16/2015 11:54 AM
Scripts	File folder	1/16/2015 11:53 AM

Figure 5-1: Design Source Directory Structure Example

The idea is to categorize and organize the various types of Vivado design sources. Define a naming convention and subdirectory structure that best works for your design and design team. The area itself could be managed by a revision control system or it could be a design specific checked out copy.

Vivado IP and IP Subsystems are device specific, so thought should be given when considering storage directory names. This can help identify them for later use in other designs.

Using an Active Working Area

Xilinx recommends creating an active working directory in order to provide a writable area to create projects, compile the design, write results, and experiment to close the design. This area could also contain the writable versions of the design sources and IP, if desired.

It is good design practice to put some thought into how to name and organize design specific sources, scripts, and results for easy identification. Devise subdirectory and source file naming schemes in order to help understand their contents later. Remove stale or unsuccessful design attempts and stale sources.



CAUTION! *The Windows operating system has a 260-character limit for path lengths, which can affect the Vivado tools. To avoid this issue, use the shortest possible names and directory locations when creating projects, defining IP or managed IP projects, and creating block designs.*

Managing Vivado Design Suite Projects

Using a Vivado Design Suite project can complicate the interaction with a source control system. Certain steps should be followed when using Vivado Design Suite projects with revision control. The project can maintain its own copy of the source files or reference remote sources and provides its own source management. However, there are methods to use Projects with revision control. The following methods are recommended.

Using Remote or Local Sources

When projects are created and managed through the Vivado tools, sources can either be referenced in their original locations or copied local to the project. Both local and remote sources can be interactively manipulated using the Vivado IDE. The text editor can be used to edit the sources, and the results can be analyzed and modified in open designs.

Design sources can be read-only protected, and stored anywhere that is network accessible from the design directory. Read-only sources limit the advantages of the interactive design features of the Vivado IDE by not allowing modifications.

For easiest interfacing with revision control systems when you are using Project Mode, you should create projects using remote design sources and not copy sources into the project directory. Managing sources remotely lets you easily maintain the design sources in the project while using the interactive Vivado IDE capabilities. You can check in new versions to the revision control system as modifications are made to the source files.

Alternately, you can elect to copy sources locally into the project directory which makes the entire design project more portable and self-contained, but it also buries the design source files down inside the project subdirectories.



TIP: *You can save most source modifications with a new name.*

Recreating Projects with only the .xpr file

You can recreate and manage Vivado Design Suite projects with a single file (`<project_name>.xpr`). The project file, along with the various project source files, are the only files that you need to manage under revision control. The entire project can be recreated by opening this project file, along with its associated source files, provided that the source files are at their original locations and were added as remote sources to the project. If sources are internal to the project, the standalone XPR file will not be able to reproduce the project correctly.

Recreating Projects with the `write_project_tcl` command

When using projects with source control systems, Xilinx recommends rebuilding the project from scratch using a Tcl-scripted approach. The Vivado `write_project_tcl` command provides the ability to create a Tcl script that you can use to recreate the current design using the same source files and settings.

```
write_project_tcl <script_file_name>.tcl
```

The resulting script file should be managed with revision control and can be used to recreate the project as follows:

```
source <script_file_name>.tcl
```

The new project is written to the current working directory (CWD) where you source the `<script_file_name>.tcl` file.



RECOMMENDED: *Xilinx recommends sourcing the Tcl script from the same directory in which the project was created. However, if you source the Tcl script from a different directory, set the `origin_dir_loc` variable in the Tcl shell to the new directory, or edit the `origin_dir` variable in the Tcl script to maintain the relative path between the CWD and the source files.*

Using Vivado Design Suite Script-Based Flows

The easiest way to interact with source control systems is to use the Non-Project scripted flow. The designer can check out the desired sources into a local directory structure or they can be read directly from their managed locations. New source files may also need to be created. Once the files are ready, the `read_<source_type>` Tcl commands pass the files to the Vivado synthesis and implementation commands. The source files remain in their original locations. The checked-out sources can be modified interactively, or with Tcl commands during the design session, using appropriate code editors. A common example of such a modification is a timing constraint change.

Note: Although source files can be read-only protected, this disables them from being modified.

Source files are then checked back into the source control system at the designer's discretion. Design results such as design checkpoints, reports, and bit files can also be checked in for revision management.

Using IP and IP Subsystems

IP and IP Integrator Subsystems are best configured interactively within the Vivado IDE. The IP customization wizards and the interactive IP Integrator canvas make the job very easy. Once the IP or IP Subsystem is configured and the output products are generated, the sources can be read by Vivado Tcl commands. For IP, the `.xci` should be used for the source. For IP Integrator, the `.bd` should be used as the source.

Note: You can also use the design checkpoint files (.dcp) for IP or IP Integrator as sources in the script based flow. Xilinx recommends using the .xci and .bd files, because they are guaranteed to be the latest version.

Managing Scripts and Reports

Designers should also manage any Tcl scripts and modified .ini startup files required to recreate the design. Managing the various output report files can also help to identify the design state for future reference.

Managing IP

Xilinx recommends that you create project-specific storage locations for the IP used in the design project. Because IP can be re-configured from any design project that uses the IP, creating design-specific IP prevents unwanted updates by other designers.

Vivado Design Suite IP and IP Subsystems are device specific, so thought should be given when considering sharing them between designs and when setting up management areas.

Vivado Design Suite IP can be configured, stored, and managed using the following methods:

- Create a Manage IP Project (Recommended)
- Project Local IP
- Project Remote IP

Creating Manage IP Projects (Recommended)

With centralized IP management, the customized IP and their output products are stored in a centralized repository outside of the current design or project directory structure. Each IP that is customized will be stored in its own self contained directory. The IP can be referenced from this repository in either Project Mode or Non-Project Mode, either by a Tcl script or by adding it to a project as a remote source.

The Manage IP project creates IP storage locations to configure, validate, and store multiple IP cores. They take advantage of the Vivado IDE to enable IP configuration using the IP Catalog, source management, analysis, and the runs infrastructure to validate and store the results for each IP core. This capability allows IP designers to select IP cores from the IP Catalog, configure and customize them, and then generate output products in the desired design configuration. The environment also allows you to validate the IP by performing synthesis and implementation.



RECOMMENDED: *Xilinx recommends that you manage the entire IP Location directory structure intact, including the `managed_ip_project` directory. This ensures that the Vivado Design Suite maintains the status of the IP runs and output products.*

Project Local IP

The IP can be configured and stored within the Vivado Design Suite project. The IP output products can reside within the project directory structure. Storing the IP output products local to the design project creates a standalone entity for the entire design, which can easily be packaged and shared. This is also useful if the design uses a unique IP configuration that you may not want made available in a shared IP repository.

Using the IP Catalog within a project to customize and add IP is straightforward. The project is self-contained and easily managed in one location. When an IP is not used in multiple projects or by multiple people, this is a simple approach to take. The IP is simply another part of a project that is managed along with all other design data, such as RTL sources and run results.

The entire project directory structure should be managed intact.

Project Remote IP

The Vivado Design Suite also enables individual IP to be configured standalone and remotely for use in both Project Mode and Non-Project Mode. While configuring any Vivado Design Suite IP, you can specify the IP Location, which can be set to a location outside of the project directory as shown in the following figure.

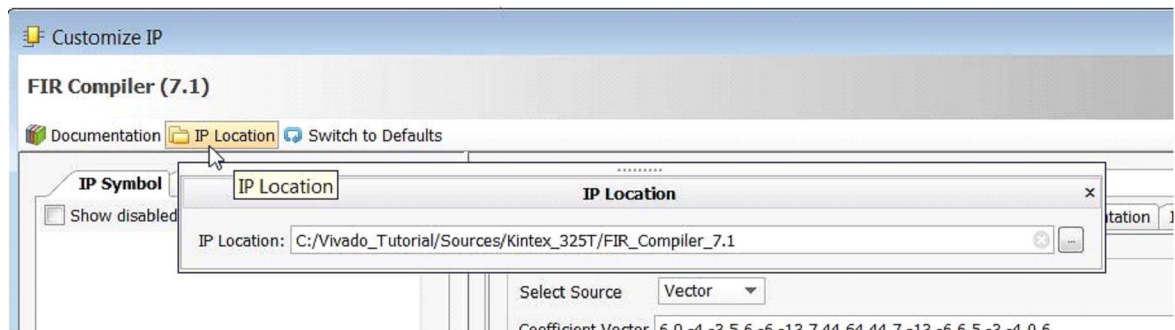


Figure 5-2: Defining a Remote Location for an Individual IP

Setting the IP Location to a directory outside the Vivado Design Suite project writes the IP configuration file (`.xc1`), and the various IP output products including the RTL sources and constraints into a standalone directory with a name matching the IP customization name.

Remote IP can be used in any number of design projects. However, since the features of the Vivado IDE enable modification and version updates to remote IP, you must take care when modifying or updating shared remote IP, as it may affect other users and designs. Setting up design specific remote IP locations prevent unwanted modification from other designers.

When using Remote IP Locations, follow these guidelines:

- Design an IP storage directory structure that differentiates device types, IP types, and other elements.
- Store example designs outside the IP directory to ensure that they are preserved when IP is updated.

Manage the entire remote IP directory structure intact. You can specify a location, but each IP is always placed in its own directory, named after the IP, just as with in project and manage IP projects.

Managing IP Sources

With Vivado Design Suite IP, each IP core is stored in a separate subdirectory containing the main .xci IP configuration file, along with RTL, XDC, and other related output product files required to synthesize and implement the IP. Xilinx recommends managing all of these files with their directory structure intact in order to mitigate any risk of reproducing the results in future versions of software.

Using IP Core Containers (Recommended)

To facilitate interactions with revision control systems, you can store IP configuration files and output products in a single, binary IP core container file rather than a directory structure. The Vivado Design Suite interacts directly with the IP core container files. When using IP core containers, you only need to manage the IP .xcix file. The .xcix file contains all of the files required for simulation, synthesis, and implementation.

For more information on using IP core containers, see *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 12]. For more information on simulating with IP core container files, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Note: All of the output product source files, including the .xci file or .xcix file, can be read only.



IMPORTANT: Create IP example designs in an unmanaged location to enable compilation and experimentation.

Some IP have additional associated files, such as COE, CSV, ELF, and BMM files. These files need to be managed with revision control systems. In a project flow, these files are registered with the project and might appear in the Design Sources area. Although the files are not imported into the project, they remain at the location they were referenced.

For example, when you specify a .coe file for the FIR Compiler, the file is added to a Coefficient Files folder under the Design Sources. The .coe file location is stored as a relative path in the IP XCI file. When placing the IP and the COE file in revision control, the relative paths need to be maintained. If the relative path needs to be changed, the path is a property of the IP. You can update this path property using the following Tcl command:

```
set_property CONFIG.Coefficient_File {/location/of/coe/file} [get_ips my_FIR_compiler]
```



TIP: You can also update this path property by re-customizing the IP in the GUI and specifying the COE file location.

Note: Archive Project correctly handles these file by copying the files into the project directory structure and updating the path references for the IP. When using the `write_project_tcl` command, the files are either referenced from their current location or imported depending on the options used.

Managing IP Core Containers and External Third-Party Files (Recommended)

When using IP core containers, the following IP output files are duplicated outside of the core container for easier access by third-party tools:

- Instantiation template
- Synthesis stub files
- Simulation source files
- Scripts
- Test benches

You can find these duplicated files in the `ip_user_files` and `ip_static_files` subdirectories for the IP.



RECOMMENDED: When using third-party tools, Xilinx recommends managing the IP `.xcix` file as well as these intact external subdirectories.

Managing All IP Output Products (Recommended)

It is recommend you manage the entire IP directory containing all IP output products with revision control. This gives the flexibility to decide when and if to upgrade the IP at a future point. It also provides better run time as the IP does not have to be regenerated every time it is used.

The Vivado IDE only supports one version of an IP in the IP Catalog. If you upgrade the Vivado IDE and the IP is no longer current, it is still usable. The IP is locked and you are not able to re-customize or generate output products. However, if all the output products are present, the IP can be used.

By default, all IP are synthesized out-of-context from the rest of the design. The synthesized design checkpoint produced is an output product of the IP. To ensure the IP can be used in future versions of the Vivado IDE these files must be present if the default out-of-context flow was used when creating the IP. As with other output products they cannot be created if the IP is not the current version.

Managing the IP .xci File Only

To restore the IP customizations used in a design, you must at a minimum preserve the .xci file for the specific configuration of the IP. From the .xci, the IP can be regenerated using the same Vivado IDE release with which it was created. If you plan to stay with this software version, or plan to always upgrade the IP to the latest version, the IP .xci is sufficient. However, to use an IP core (including its current customization and the RTL, XDC, etc.) with future versions of the Vivado IDE, you should place the entire IP output product directory under revision control and maintain the directory structure.

IP directory structures should be maintained in order to ensure locations to create the output products. Do not store multiple .xci files in a single directory as output products will collide when generated.

Managing the IP .xcix Core Container File Only

When you want to manage a minimum set of files and still include the IP output products needed for simulation and synthesis, use IP core containers. With core containers, you only need to manage the IP .xcix file. You can use the following command to export all of the IP source files needed for simulation, synthesis, and implementation into any unmanaged area for use in any design project:

```
export_ip_user_files
```


Managing IP Integrator Subsystems

IP Integrator Subsystems contain multiple IP, along with their customized parameters, and interconnect. These IP subsystems are created with the IP Integrator and are referred to as Block Designs (BD) within the Vivado IDE.

Creating Remote Block Designs (Recommended)

When you create a Vivado IP Integrator block design (.bd extension), you can specify a remote location. This is the easiest way to manage and store revisions of the block designs. The Vivado tools store all of the files and IP used in the block designs in a directory remote from the project directory, which is similar to [Project Remote IP](#).

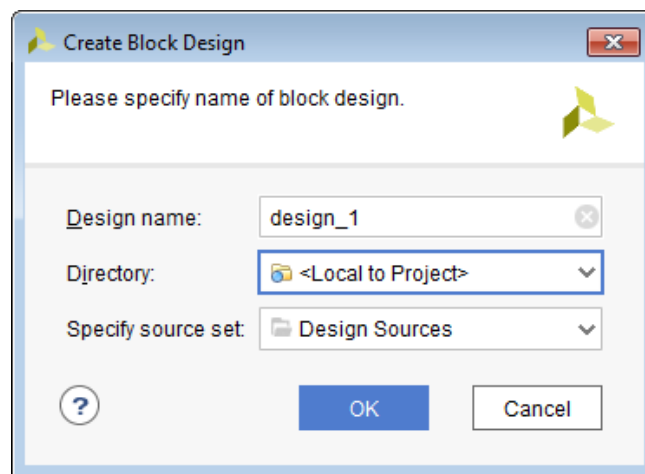


Figure 5-3: Defining a Remote Location for a Block Design

For more information, see this [link](#) on creating block designs in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 30].

Creating Project-Based Block Designs

By default, Vivado IP Integrator will create and store the block design data inside the local project directory structure. This can make interaction with a revision control system tricky as the Vivado IDE enables interactive modifications and management of the sources. Storing the IP output products within the design project creates a standalone entity for the entire design, which can easily be packaged and shared. This is also useful if the design uses a unique IP configuration that you may not want made available in a shared IP repository.

Managing IP Integrator Sources

With Vivado IP Integrator, each IP core used in the BD is stored in a separate subdirectory under the BD, containing the main .xci IP configuration file, along with RTL, XDC, and other related output product files required to implement each IP. Xilinx recommends managing all of the BD files with the directory structure and file names intact in order to mitigate any risk of reproducing the results on future versions of software.

All of the output product source files including the .bd file can be read only, however it limits the block design capabilities of the Vivado IDE.

You should take care when using IP example designs to ensure they are created in an unmanaged location to enable compilation and experimentation. By default they are created inside the IP subdirectory.

Recreating the Block Design using only the .bd File

At a minimum, the .bd source file associated with the BD should be managed with revision control. Adding the .bd as a project source will trigger the regeneration of the block design and related IP output products. This approach can be time consuming, because all of the IP and block design output products will need to be regenerated and compiled. It also limits the recreation of the block design to the version of software it was created with. Xilinx recommends checking in the entire bd directory. Ensure the BD has been generated successfully with all of the associated IP output products.

Recreating the Block Design Using the `write_bd_tcl` Command

The Vivado `write_bd_tcl` command provides the ability to create a Tcl script that you can use to recreate the current IP integrator block design (`.bd`) using the same IP, connectivity, and settings. Ensure the BD has been generated successfully with all of the associated IP output products.

```
write_bd_tcl <script_file_name>.tcl
```

You can use this command with both Project and Non-Project flows to recreate the block design providing the sources are still located in the same places.

```
source <script_file_name>.tcl
```

You should also manage the resulting script file with revision control.

Managing Simulation Sources

Simulation sources are often the majority of the design source files along with test benches and simulation scripts. All of these files should be managed by revision control. They can be exported into a single directory from the Vivado Design Suite.

For more information on exporting files for logic simulation, see [Batch Simulation](#).

Managing System Generator Sources

The entire project directory created by the Xilinx System Generator should be managed intact by revision control.

Managing HLS Sources

All of the C based design sources, run scripts, and potentially the output packaged RTL IP should be managed by revision control.

Managing Design Results

Depending on your design team needs, additional files may be desired to be managed. These may include the design bitstream file, implementation results, design checkpoints, log files and reports, custom Tcl commands and run scripts. Manage these files appropriately.

Design Checkpoints

The Vivado Design Suite uses Design Checkpoint files (`.dcp`) to store the current state of the design being processed through the flow. These checkpoints include the netlist, constraints, and design results at the stage when the checkpoint was written.

Checkpoints should be written after each stage of the design process. They are automatically created when using a project to process synthesis and implementation runs.

Checkpoints can be opened in the Vivado IDE for design analysis and constraints modification. Constraint changes can be written to new constraints files for use during the next run through the flow. Checkpoints are images of a design at a given state in the flow. Checkpoints do not contain the entire project or the source files. They can typically be passed on to the remaining design steps, such as Generate Bitstream.

Your design team may elect to also store milestone design checkpoints.

Archiving Designs

The `archive_design` command can package up an entire project into a compressed zip file. The command has several options for storing sources and run results. Essentially the entire project is copied locally in memory, and then zipped into a file on disk while leaving the original project intact. This command also copies any remote sources into the archive. This feature is useful for sending your design description to another person or to store as a self contained entity. You might also need to send your version of `vivado_init.tcl` if you are using this file to set specific parameters or variables that affect the design.

For more information, see the following resources:

- [Vivado Design Suite User Guide: System-Level Design Entry \(UG895\) \[Ref 11\]](#)
- [Vivado Design Suite QuickTake Video: Creating Different Types of Projects](#)
- [Vivado Design Suite QuickTake Video: Managing Sources with Projects](#)

Managing Hardware Manager Projects and Sources

Project `.bit` file and `.ltx` files are the primary output files required for using the Vivado Design Suite Debug and Programming features in Vivado hardware manager. Xilinx recommends you manage these files under revision control if you want to use this project in Vivado Lab Edition.

When Using Vivado Design Suite Projects

The `project_name.hw` directory in your Vivado Design Suite project stores information about custom dashboards, trigger, capture conditions, waveform configuration files etc created as part of using the Debug and Programming in the Vivado hardware manager. Xilinx recommends you manage the `project_name.hw` directory in your Vivado Design Suite project under revision control. This also helps if you want to hand off this project to be used in Vivado Lab Edition.

Managing Vivado Lab Edition Sources

Xilinx recommends you manage the project directory that was created for the project in Vivado Lab Edition under revision control. The `hw_*` directories in the Lab Edition project directory stores information about custom dashboards, trigger, capture conditions, waveform configuration files, etc., in the Vivado hardware manager as part of using the Debug and Programming. The `.lpr` file in the Lab Edition project directory is the project file that you need to manage under revision control. The entire project can be recreated by opening this project file, provided that the `hw_*` directory are at their original locations.

Upgrading Designs and IP to the Latest Vivado Design Suite Release

It is highly likely that a new release of the Vivado Design Suite will become available during your design process. You can update to this new release, or stay with your current release. Although Xilinx recommends that you use the latest release, it is not mandatory. You should be aware though, that Xilinx does not support versions prior to the last two major releases. New releases may contain:

- Software bug fixes
- New IP versions
- Updated device files (including speed files for various devices)
- New device offerings from Xilinx
- New software features
- Performance improvements

The Vivado Design Suite project may be upgraded when migrating to a newer release. The project and device file upgrades typically happen automatically without user interaction.

You can elect to update the latest IP version, or lock the IP at the version with which it was configured. To use a locked version of an IP, generate the output products for the IP, and then use those during subsequent software updates. This does prevent you from reconfiguring the IP on the latest release unless you use the software version with which it was originally configured. Although Xilinx recommends that you use the latest IP versions, doing so is not mandatory.

IP Subsystems (created using IP integrator) also require you to manage the IP contained in them. You can either stay with the locked version of the IP output products, or update them to the latest version. All IP included in a particular IP Subsystem must be updated simultaneously.

For more information, see [Working with IP](#) as well as the following resources:

- *Vivado Design Suite User Guide: Designing with IP* (UG896) [\[Ref 12\]](#)
- *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [\[Ref 30\]](#)
- [Vivado Design Suite QuickTake Video: Managing Vivado IP Version Upgrades](#)

For information on migrating your design to the new software release, see *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973) [\[Ref 29\]](#).

Updating Projects

When opening a project, you are prompted to upgrade if it was last saved with an older version of the Vivado Design Suite. It does allow you to open the project in a read-only manner. Project updates are related to new project formats only. It does not automatically update IP or IP Subsystems. It is recommend to update to the latest project format with each release.

Updating IP

If the IP is updated in a future Vivado Design Suite release, Xilinx recommends that you upgrade. When using the Vivado IDE you can usually upgrade to the latest IP. The Change Log describes the upgrade. If you do not wish to upgrade, you must generate and archive all the output products for an IP. For more information, see [Managing IP](#).

These saved targets may be used, however, they cannot be recustomized using newer version of the Vivado Design Suite. Neither can additional output products be created in a new Vivado IDE release.

Updating IP Subsystems

If any of the IP used in the subsystem is updated in a future Vivado Design Suite release, Xilinx recommends that you upgrade the entire block design. If you do not wish to upgrade, you must generate and archive all the output products for an IP. Editing the block design is disabled. For more information, see [Managing IP](#).

These saved targets may be used, however, they cannot be recustomized using newer version of the Vivado Design Suite. Neither can additional output products be created in a new Vivado IDE release.

For more information, see the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [\[Ref 30\]](#).

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

Note: For more information on Documentation Navigator, see the [Documentation Navigator](#) page on the Xilinx website.

References

1. *Model Composer User Guide* ([UG1262](#))
2. *Vivado Design Suite User Guide: High-Level Synthesis* ([UG902](#))
3. *UltraScale Architecture-Based FPGAs Memory IP* ([PG150](#))
4. *AXI BFM Cores LogiCORE IP Product Guide* ([PG129](#))
5. *Reference System: Kintex-7 MicroBlaze System Simulation Using IP Integrator* ([XAPP1180](#))
6. *Vivado Design Suite Tcl Command Reference Guide* ([UG835](#))
7. *Vivado Design Suite Tutorial: High-Level Synthesis* ([UG871](#))
8. *Vivado Design Suite Tutorial: Design Flows Overview* ([UG888](#))
9. *Vivado Design Suite User Guide: Using the Vivado IDE* ([UG893](#))
10. *Vivado Design Suite User Guide: Using Tcl Scripting* ([UG894](#))
11. *Vivado Design Suite User Guide: System-Level Design Entry* ([UG895](#))
12. *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
13. *Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator* ([UG897](#))
14. *Vivado Design Suite User Guide: Embedded Processor Hardware Design* ([UG898](#))
15. *Vivado Design Suite User Guide: I/O and Clock Planning* ([UG899](#))
16. *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))
17. *Vivado Design Suite User Guide: Synthesis* ([UG901](#))
18. *Vivado Design Suite User Guide: Using Constraints* ([UG903](#))
19. *Vivado Design Suite User Guide: Implementation* ([UG904](#))
20. *Vivado Design Suite User Guide: Hierarchical Design* ([UG905](#))
21. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* ([UG906](#))
22. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
23. *Vivado Design Suite User Guide: Partial Reconfiguration* ([UG909](#))
24. *Vivado Design Suite User Guide: Getting Started* ([UG910](#))
25. *ISE to Vivado Design Suite Migration Guide* ([UG911](#))
26. *Vivado Design Suite Tutorial: Embedded Processor Hardware Design* ([UG940](#))
27. *Vivado Design Suite Tutorial: Partial Reconfiguration* ([UG947](#))
28. *UltraFast™ Design Methodology Guide for the Vivado Design Suite* ([UG949](#))

29. *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* ([UG973](#))
 30. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* ([UG994](#))
 31. *UltraFast Embedded Design Methodology* ([UG1046](#))
 32. *Vivado Design Suite User Guide: Creating and Packaging Custom IP* ([UG1118](#))
 33. *Vivado Design Suite Tutorial: Creating and Packaging Custom IP* ([UG1119](#))
 34. *Vitis Unified Software Platform Documentation* ([UG1393](#))
 35. [Vivado Design Suite Documentation](#)
-

Training Resources

Xilinx provides a variety of training courses and QuickTake videos to help you learn more about the concepts presented in this document. Use these links to explore related training resources:

1. [Designing FPGAs Using the Vivado Design Suite 1 Training Course](#)
 2. [Designing FPGAs Using the Vivado Design Suite 2 Training Course](#)
 3. [Vivado Design Suite QuickTake Video: Vivado Design Flows Overview](#)
 4. [Vivado Design Suite QuickTake Video: Getting Started with the Vivado IDE](#)
 5. [Vivado Design Suite QuickTake Video: Targeting Zynq Devices Using Vivado IP Integrator](#)
 6. [Vivado Design Suite QuickTake Video: Partial Reconfiguration in Vivado Design Suite](#)
 7. [Vivado Design Suite QuickTake Video: Simulating with Cadence IES in Vivado](#)
 8. [Vivado Design Suite QuickTake Video: Simulating with Synopsys VCS in Vivado](#)
 9. [Vivado Design Suite QuickTake Video: I/O Planning Overview](#)
 10. [Vivado Design Suite QuickTake Video: Using Vivado Design Suite with Revision Control](#)
 11. [Vivado Design Suite QuickTake Video: Creating Different Types of Projects](#)
 12. [Vivado Design Suite QuickTake Video: Managing Sources with Projects](#)
 13. [Vivado Design Suite QuickTake Video: Managing Vivado IP Version Upgrades](#)
 14. [Vivado Design Suite QuickTake Video Tutorials](#)
-

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF

MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2012–2020 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.