Vivado Design Suite User Guide

Design Flows Overview

Vivado Design Suite

UG892 (v2022.1) April 20, 2022

AMD
XILINX

Table of Contents

Chapter 5: Source Management and Revision Control Recommendations

Appendix A: Additional Resources and Legal Notices

Vivado System-Level Design Flows

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)
- 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.

Related Information

RTL-to-Bitstream Design Flow
Alternate RTL-to-Bitstream Design Flows

Send Feedback

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal® ACAP design process Design Hubs and the Design Flow Assistant materials can be found on the Xilinx.com website. This document covers the following design processes:

- **System and Solution Planning:** Identifying the components, performance, I/O, and data transfer requirements at a system level. Includes application mapping for the solution to PS, PL, and AI Engine. Topics in this document that apply to this design process include:

  - Design Flows

  - RTL-to-Bitstream Design Flow

  - Alternate RTL-to-Bitstream Design Flows

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:

  - Accelerated Kernel Flows

- **System Integration and Validation:** Integrating and validating the system functional performance, including timing, resource use, and power closure. Topics in this document that apply to this design process include:

  - Running Logic Simulation

  - Logic Simulation

- **Board System Design:** Designing a PCB through schematics and board layout. Also involves power, thermal, and signal integrity considerations. Topics in this document that apply to this design process include:

  - Xilinx Platform Board Support

  - Board Files

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

The following figure 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.

AMD XILINX

Figure 1: **System-Level Design Flow for FPGAs and SoCs**

System Design Entry

- C-Based Design with High-Level Synthesis
- Model-Based Design with MATLAB® and Simulink® Software
  - Vitis™ Model Composer
- Configuring Xilinx® and Third-Party IP
- RTL Development
- IP Packager – IP Integrator
  - Configuring IP Subsystems
  - Embedded Processor Design

Software Development

- Development Software and Processor OS

Implementation

- Logic Simulation
- Dynamic Function eXchange
- Assign Logical and Physical Constraints
- Logic Synthesis
- Implementation
- Timing Closure and Design Analysis

- Generate Bitstream, Programming, and Debug
- Processor Boot and Debug
- Export to Vitis Software Development Platform

Hardware Bring-Up and Validation

X15150-063021

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).

The *UltraFast Design Methodology Guide for Xilinx FPGAs and SoCs* (UG949) 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).

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).

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).

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).

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.

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

Send Feedback

Related Information

Out-of-Context Design Flow

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).

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.

Related Information

Running Logic Simulation

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).

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).

Send Feedback

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 because the complete design is not visible in Vivado. For more information on platform creation, see *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393).

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 Xilinx Vitis™ for use in software development and validation. Simulation and debug features allow you to simulate and validate the design across the two domains.

The Vitis Design Suite 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)

- *Vivado Design Suite Tutorial: Embedded Processor Hardware Design* (UG940)

- *UltraFast Embedded Design Methodology Guide* (UG1046)

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).

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).

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)

- *Vivado Design Suite Tutorial: High-Level Synthesis* (UG871)

Dynamic Function Exchange Design

Dynamic function exchange (DFx) 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 DFx flow requires a strict design process to ensure that the reconfigurable modules are designed properly 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 DFx DRCs. The device programming method must also be properly planned to ensure the configuration I/O pins are assigned appropriately.

VIDEO: *Information on the DFx flow is available from the* Vivado Design Suite QuickTake Video: DFx*.*

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

- *Vivado Design Suite User Guide: Dynamic Function eXchange* (UG909)

- *Vivado Design Suite Tutorial: Dynamic Function eXchange* (UG947)

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).

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).*

Just as the Vivado 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.

Related Information

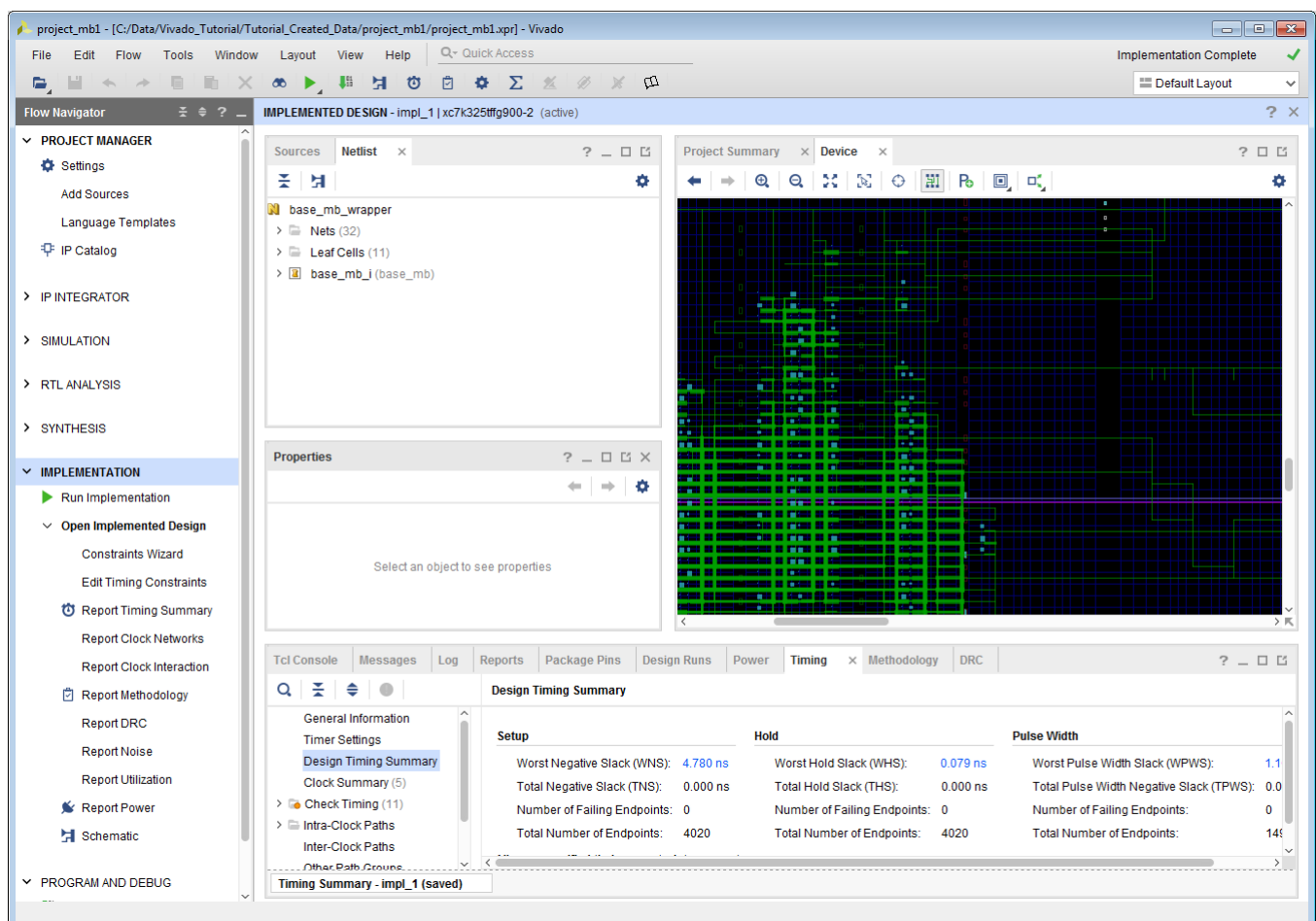Working with the Vivado Integrated Design Environment (IDE)
Working with Tcl
Understanding Project Mode and Non-Project Mode
Using Third-Party Design Software Tools

Send Feedback

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: **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).

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 3: **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
```

Send Feedback

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) and *Vivado Design Suite Tcl Command Reference Guide* (UG835). 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).

For more information on using a Tcl-based approach using either the Project Mode or Non-Project Mode, see Understanding Project Mode and Non-Project Mode.

Related Information

Using Project Mode
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.

Send Feedback

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.

Related Information

Performing Design Analysis Using the Vivado IDE
Using Design Checkpoints

Using Xilinx Vivado Store

The Xilinx® Vivado Store enables you to download Tcl apps, board files, and example designs from Xilinx's public GitHub repository. The download path for both boards and example designs can be defined in your Tool→Settings. Third-parties can also contribute to these repositories by submitting GitHub pull requests. For more information on submitting, please refer to the documentation on the GitHub for the following repositories:

- Xilinx/XilinxTclStore

- Xilinx/XilinxBoardStore

- Xilinx/XilinxCEDStore

Xilinx Tcl Apps

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).

Board Files

Board files define external connectivity for Vivado. Board files information is available in the IP integrator when you select a board, as opposed to a part, when creating the project. Board interfaces can be enabled in the IP integrator by selecting the appropriate interface in the Boards tab in Vivado. For more information, see *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) and *Integrated Interlaken up to 150G LogiCORE IP Product Guide* (PG169).

Send Feedback

Example Design

Example designs are available in Vivado to demonstrate a particular functionality. By hosting example designs on GitHub, they are updated asynchronously to the Vivado release. Example designs are accessed through the new project wizard when installed through the Xilinx Vivado Store.

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.

Note: 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.

Related Information

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-memory compilation enables project settings to be applied to Non-Project based designs. 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.

Related Information

Using Non-Project Mode

Send Feedback

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.

The following table summarizes the feature differences between Project Mode and Non-Project Mode.

Table 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.
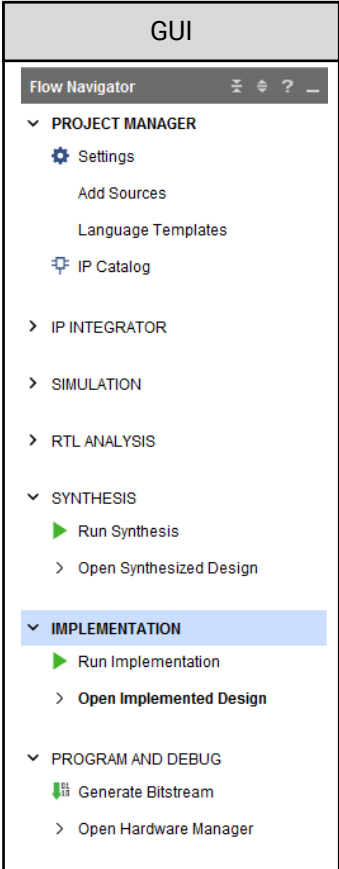
In Project Mode, the `launch_runs` command launches the tools with preconfigured run strategies and generates standard reports. This enables consolidation of implementation commands, standard reporting, use of run strategies, and run status tracking. However, you can also run custom Tcl commands before 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.

Note: 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.

The following figure shows the difference between Project Mode and Non-Project Mode Tcl commands.

Send Feedback

AMD
XILINX

Figure 4: **Project Mode and Non-Project Mode Commands**

Project Mode		Non-Project Mode
GUI	**Tcl Script**	**Tcl Script**
Flow Navigator ⤓ ⬍ ? ▬ ∨ **PROJECT MANAGER** ⚙ Settings Add Sources Language Templates ⊹ IP Catalog › IP INTEGRATOR › SIMULATION › RTL ANALYSIS ∨ SYNTHESIS ▶ Run Synthesis › Open Synthesized Design ∨ **IMPLEMENTATION** ▶ Run Implementation › **Open Implemented Design** ∨ PROGRAM AND DEBUG ⬇ Generate Bitstream › Open Hardware Manager	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	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

X12974-070621

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.

Send Feedback

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)

- Vivado Design Suite QuickTake Video: I/O Planning Overview

- [Vivado Design Hub: I/O and Clock Planning](#)

Send Feedback

Using Project Mode

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.

In the Vivado IDE, you can use the Flow Navigator (shown in the following figure) to launch predefined design flow steps, such as synthesis and implementation. When you click **Generate Bitstream** or **Generate Device Image** for Versal® ACAP, 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 push button 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.
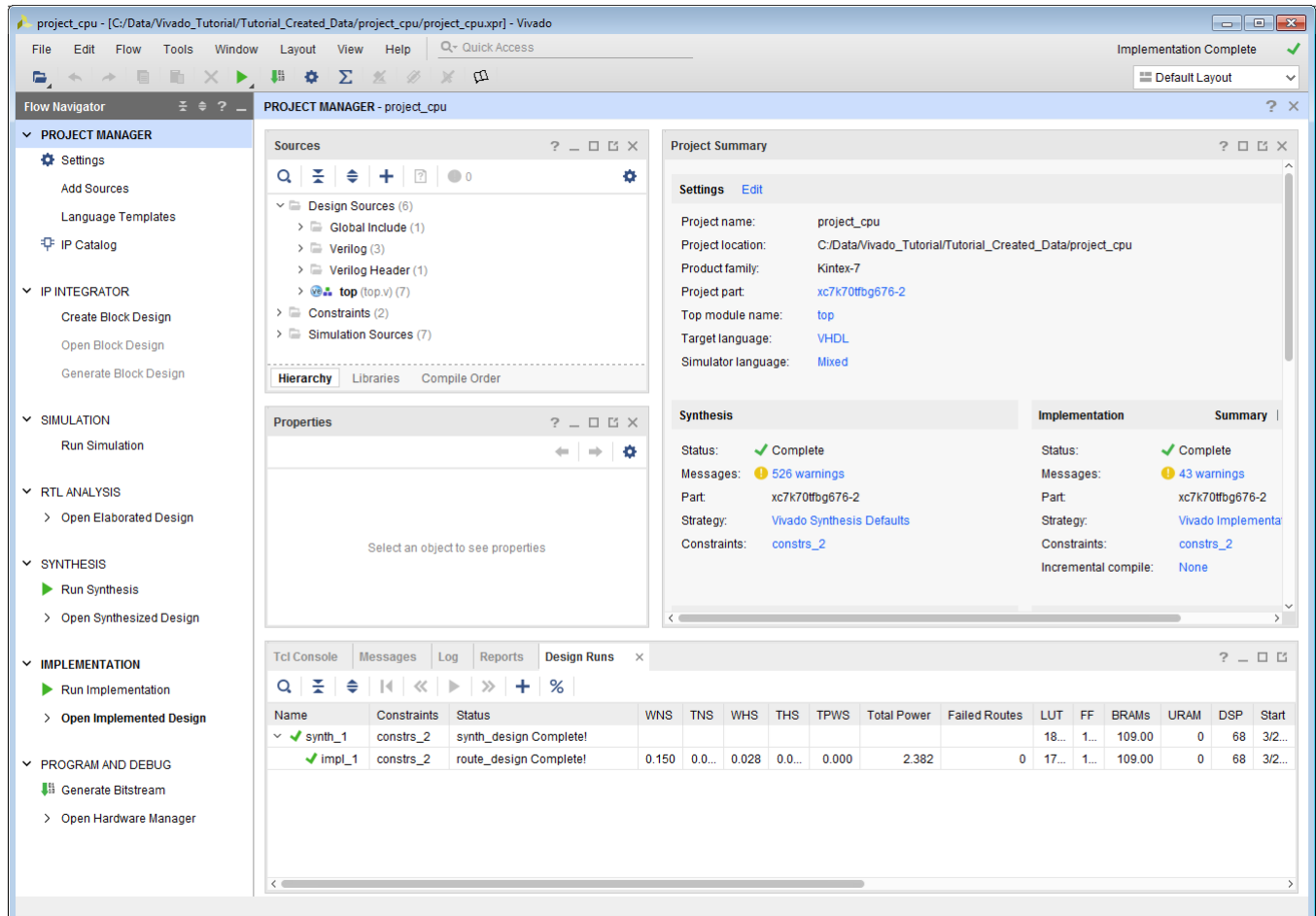
You can run Project Mode using the Vivado IDE or using Tcl commands or scripts. In addition, you can alternate between using the Vivado IDE and Tcl within a project. When you open or create projects in the Vivado IDE, you are presented with the current state of the design, run results, and previously 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).

Send Feedback

Figure 5: **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).

> ⚠ **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.

- **DFx:** You can dynamically reconfigure an operating FPGA design by loading a partial bitstream file to modify reconfigurable regions of the device.

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 Source Management and Revision Control Recommendations.
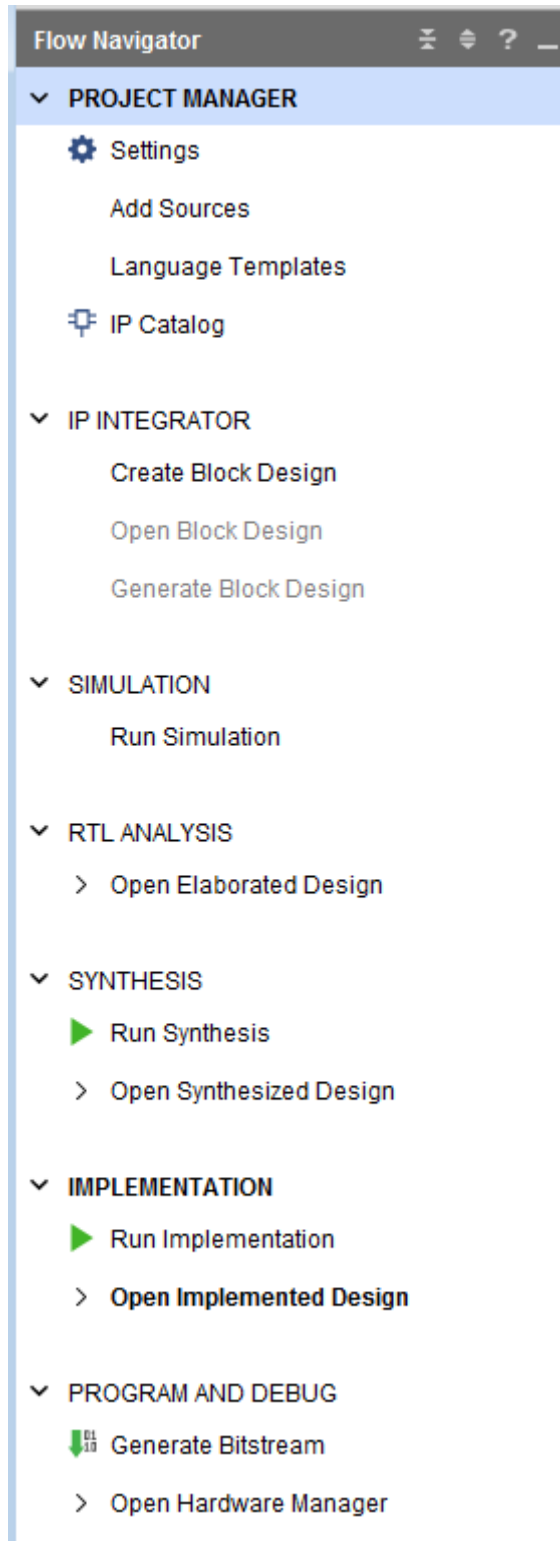

VIDEO: *For information on best practices when using revision control systems with the Vivado tools, see the Vivado Design Suite QuickTake Video: Using Vivado Design Suite with Revision Control.*
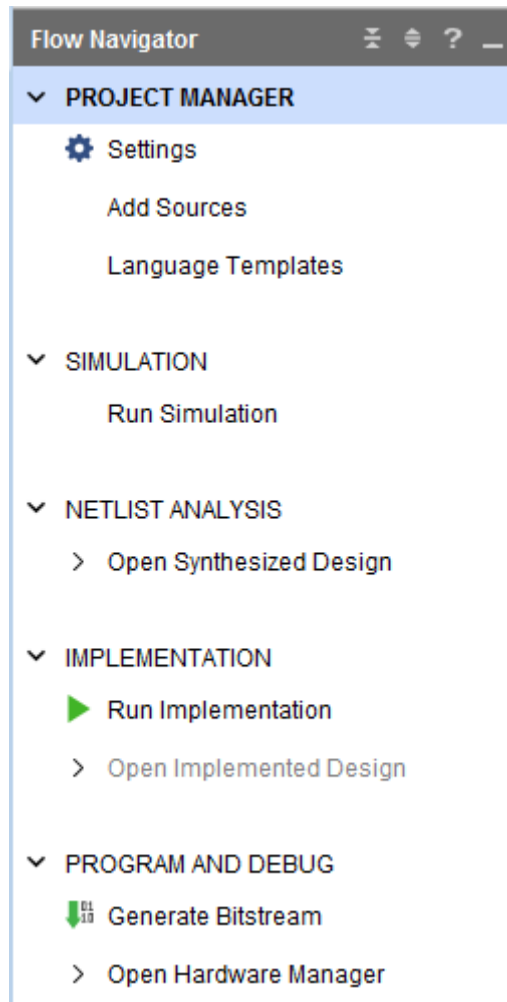

Understanding the Flow Navigator

The Flow Navigator (shown in the following figure) 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 6: **Flow Navigator**

The Flow Navigator (shown in the following figure) differs when working with projects created with third-party netlists. For example, system-level design entry, IP, and synthesis options are not available.
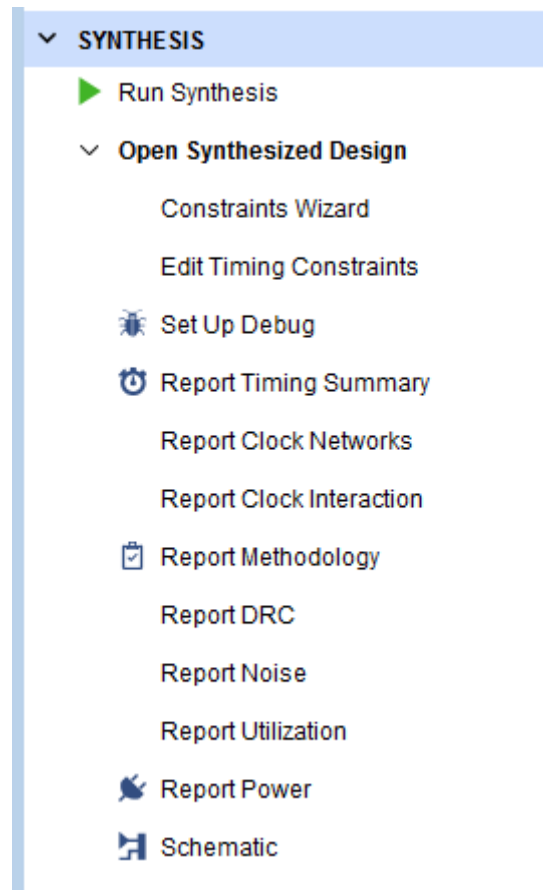
Figure 7: **Flow Navigator for Third-Party Netlist Project**

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, the following figure shows the commands related to synthesis.

Send Feedback

Figure 8: **Synthesis Section in the Flow Navigator**

Related Information

[Opening Designs to Perform Design Analysis and Constraints Definition](#)

Performing System-Level Design Entry

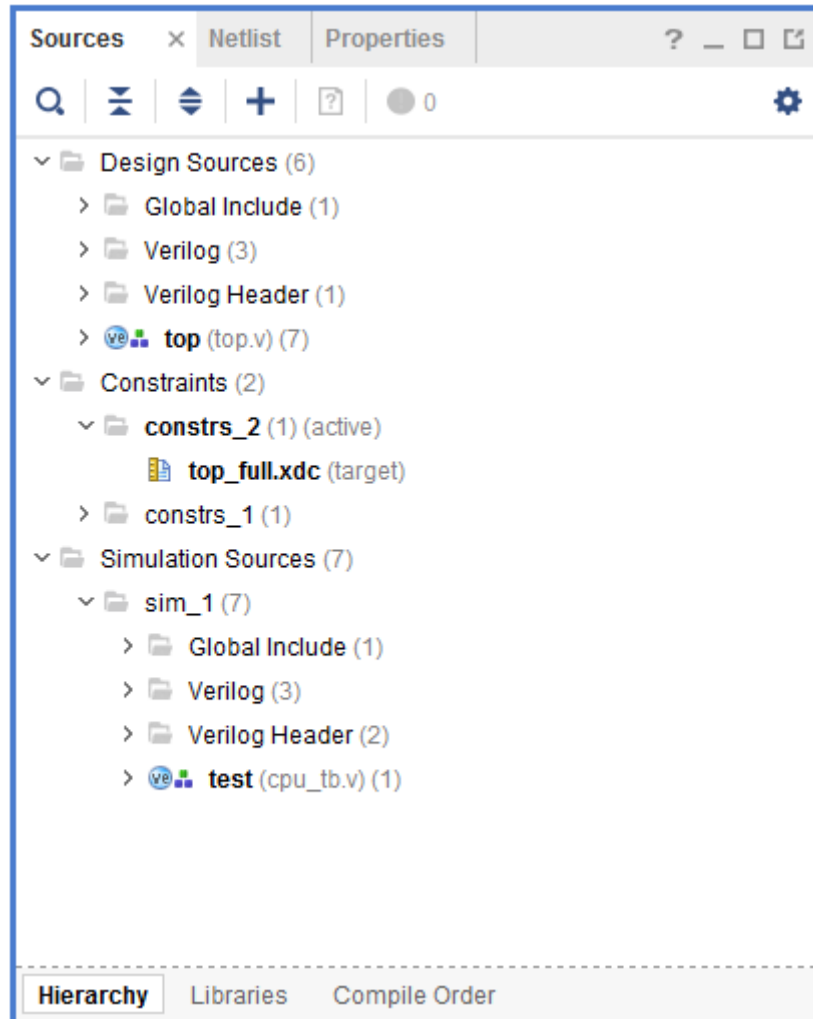## Automated Hierarchical Source File Compilation and Management

The Vivado IDE Sources window (shown in the following figure) 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).

> **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). For more information on simulation, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

Figure 9: **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

Send Feedback

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). For more information on RTL-based I/O planning, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899).

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).

> **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 ISE to Vivado Design Suite Migration Guide (UG911).*

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).

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: Design Flows Overview* (UG892).

- 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.
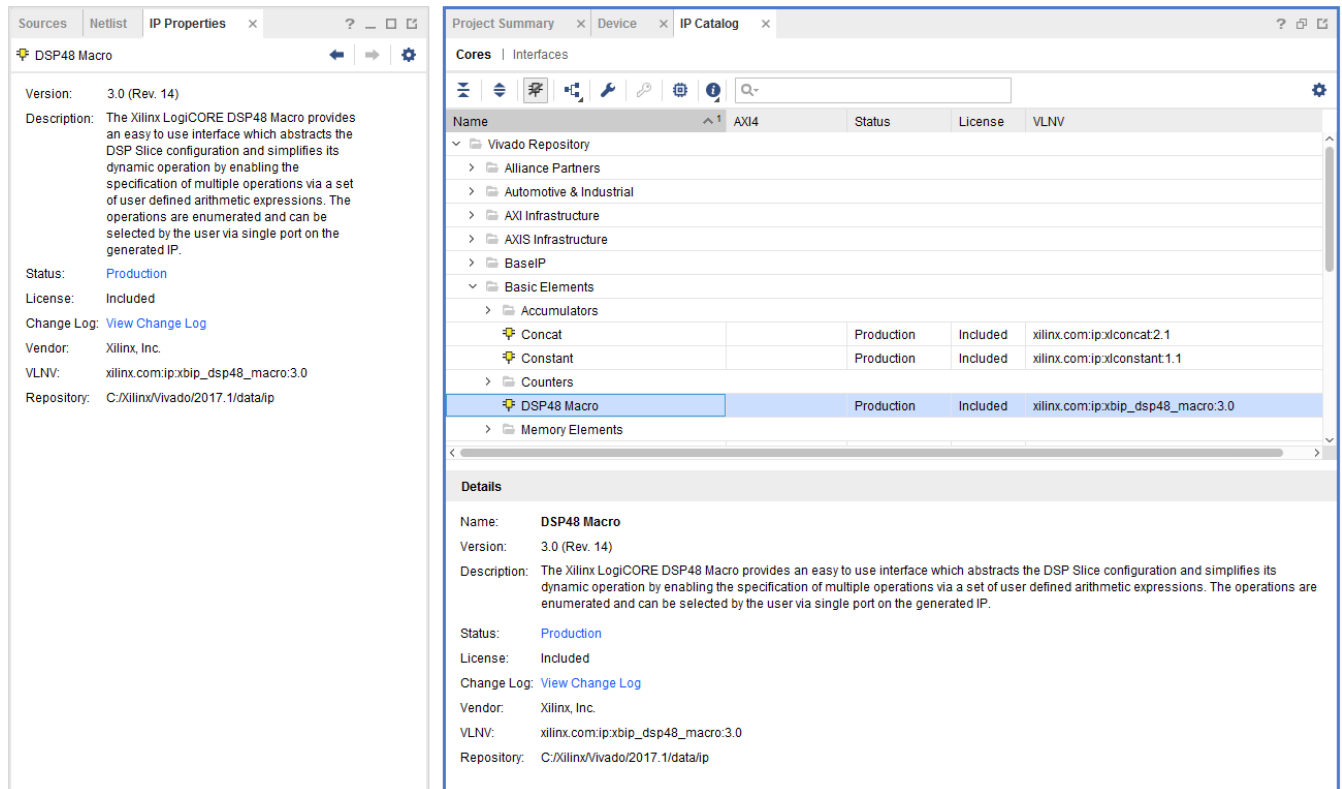
Configuring IP

The Vivado IP catalog (shown in the following figure) 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 IP.

Figure 10: **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.

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.

Send Feedback

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).

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). 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).

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). 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).

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.
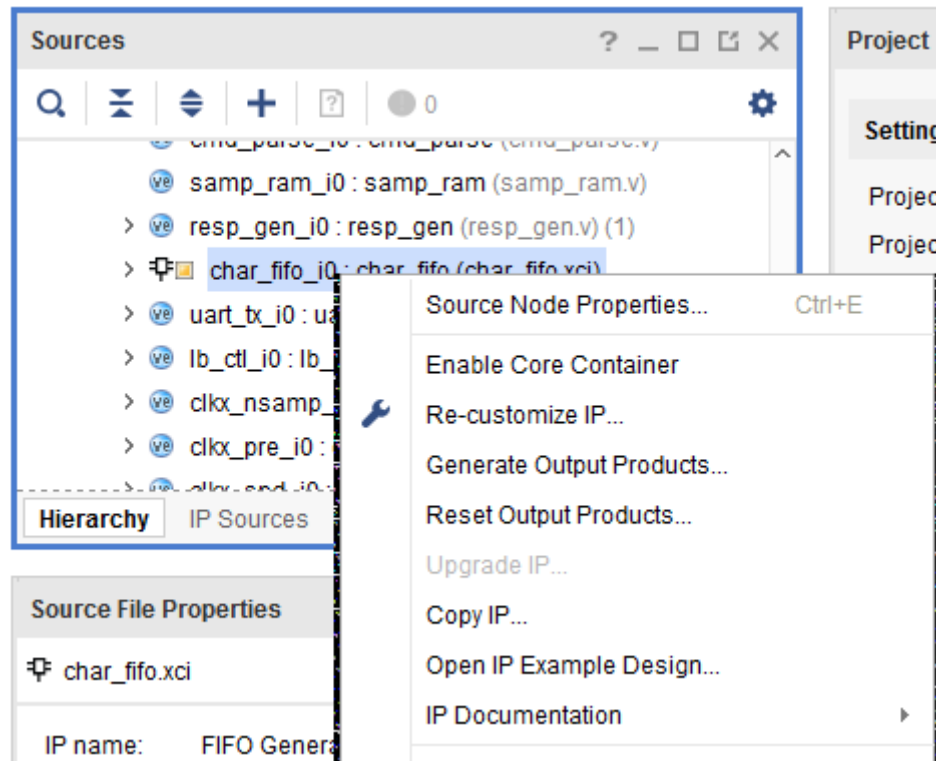
Many IP cores reference their input clocks in these XDC files. These clocks can come either from the user through the top level design, or from other IP cores in the design. By default, the Vivado tools process any IP clock creation and any user-defined top-level clock creation early. This process makes these clocks available to the IP cores that require them. Refer to this link in *Vivado Design Suite User Guide: Designing with IP* (UG896) 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 the following figure. This can also be done using Tcl by examining the `SUPPORTED_TARGETS` property of the IP.

Figure 11: **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).

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).

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.

Send Feedback

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

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

- For details on simulation, see the *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150).

- 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).

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) and *Vivado Design Suite Tutorial: Creating, Packaging Custom IP* (UG1119).
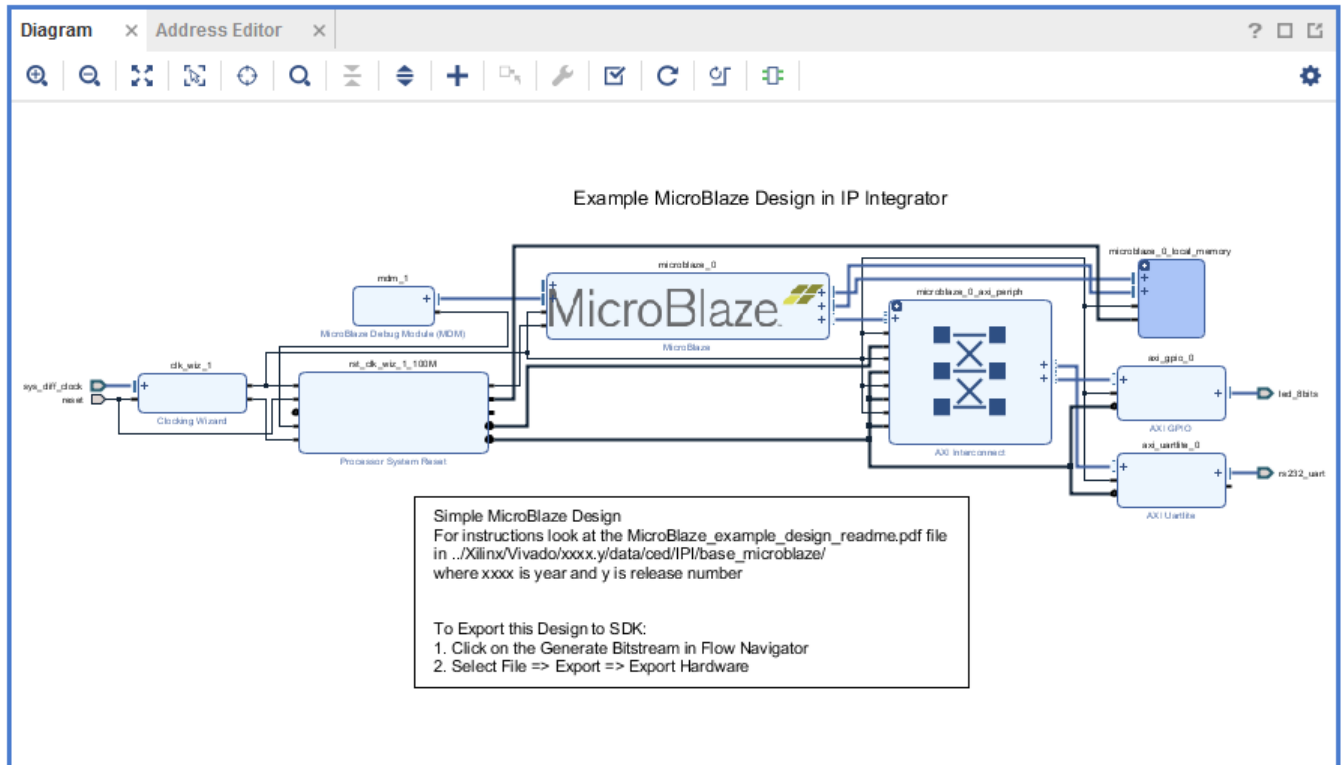
Send Feedback

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).

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 12: **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) or *Vivado Design Suite User Guide: Embedded Processor Hardware Design* (UG898).

Related Information

Packaging Custom IP and IP Subsystems

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.*

Send Feedback

Block Design Containers

Block design containers allow a block diagram to reference a secondary block diagram. This enables a design to be partitioned into several block diagrams. Block design containers also support several instances of child block diagrams in a parent block diagram. In this manner, a logic can be replicated even if each instance has different parametrization. For more information on block design containers, see *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994).

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) 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.

Related Information

Using the Platform Board Flow

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 or from the Xilinx Vivado Store.

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).

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) 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.

Related Information

Out-of-Context Design Flow

Send Feedback

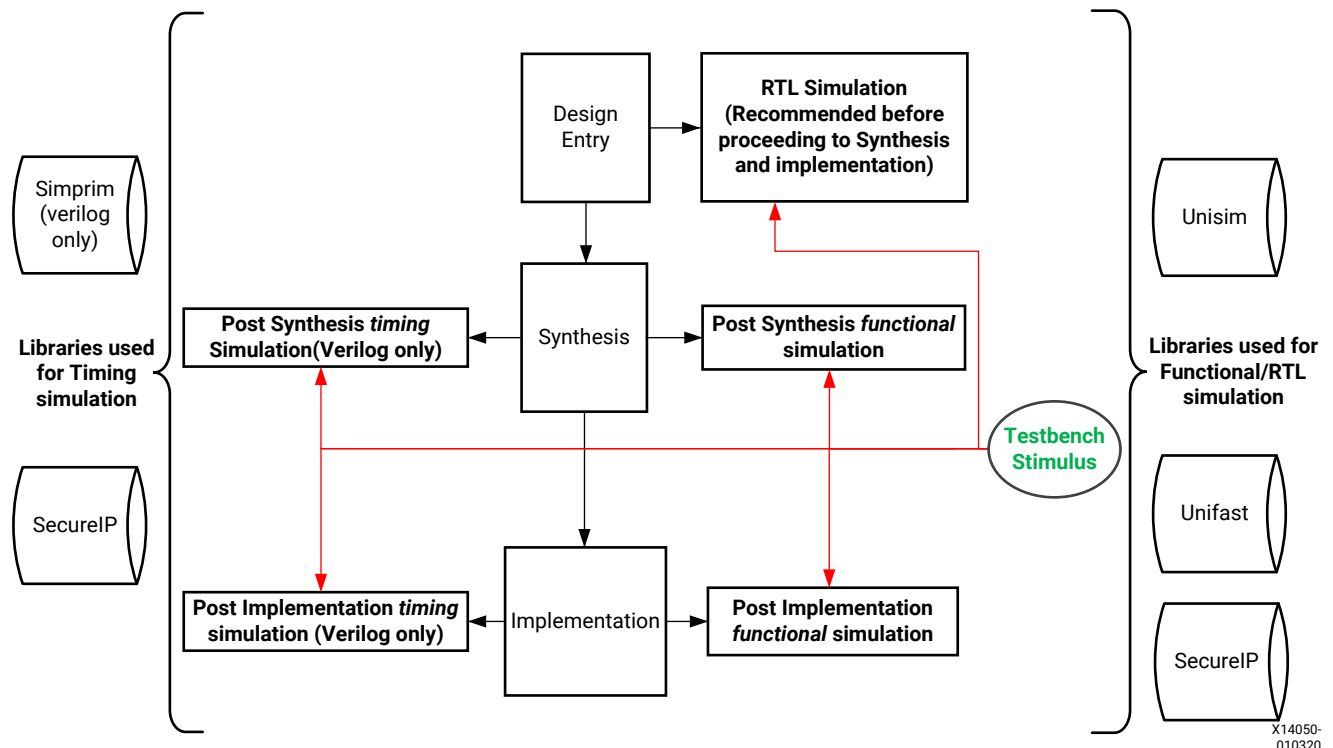# Integrating the Block Design into a Top-Level Design

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 the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) 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 13: **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. The previous figure 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).

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).

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 xelab 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).

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). 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) 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 2: **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

Send Feedback

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).

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.

Send Feedback

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).

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).

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).

Note: Launching multiple jobs simultaneously on the same machine can exhaust memory, resulting in random Vivado crashes. Ensure to reserve enough memory for all jobs running on a single machine.

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 run times, 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) and *Vivado Design Suite User Guide: Using Constraints* (UG903).

> **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).*

Note: Launching multiple jobs simultaneously on the same machine can exhaust memory, resulting in random Vivado crashes. Ensure to reserve enough memory for all jobs running on a single machine.

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) and see this link in the *Vivado Design Suite User Guide: Implementation* (UG904).

Send Feedback

> 💡 **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** or **Generate Device Image** for Versal ACAP.

- 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** or **Generate Device Image** for Versal ACAP.

You can create multiple synthesis or implementation runs to experiment with constraints or tool settings. To create additional runs:
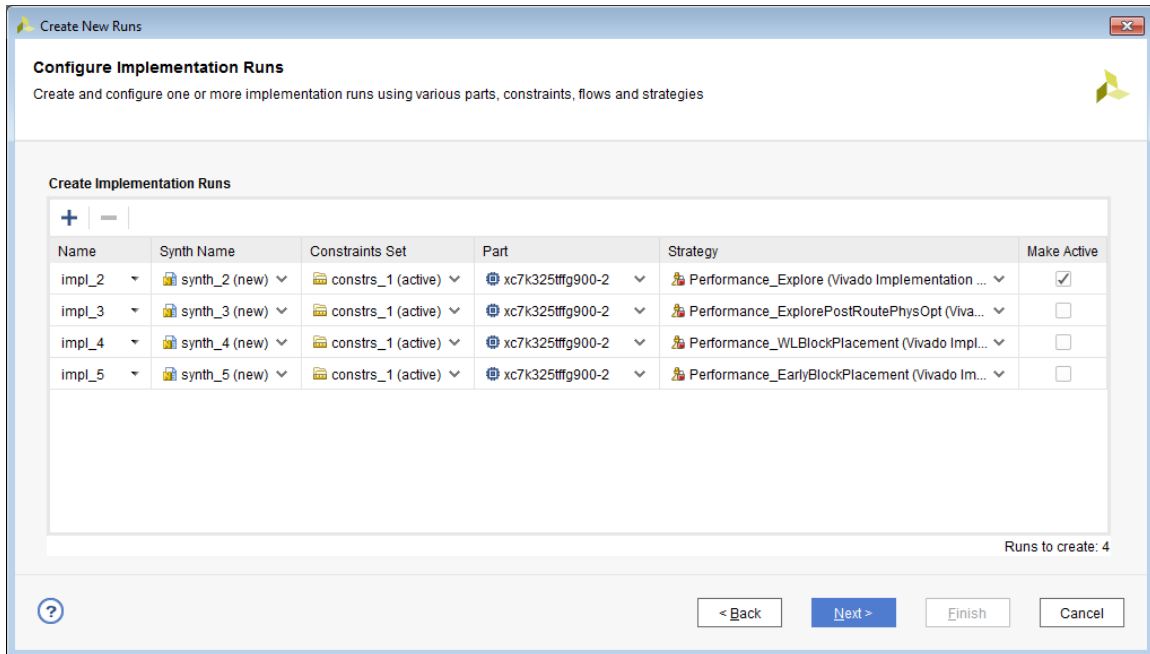
1. In the Flow Navigator, right-click **Synthesis** or **Implementation**.

2. Select **Create Synthesis Runs** or **Create Implementation Runs**.

3. In the Create New Runs wizard (see the following figure), 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.*

Send Feedback

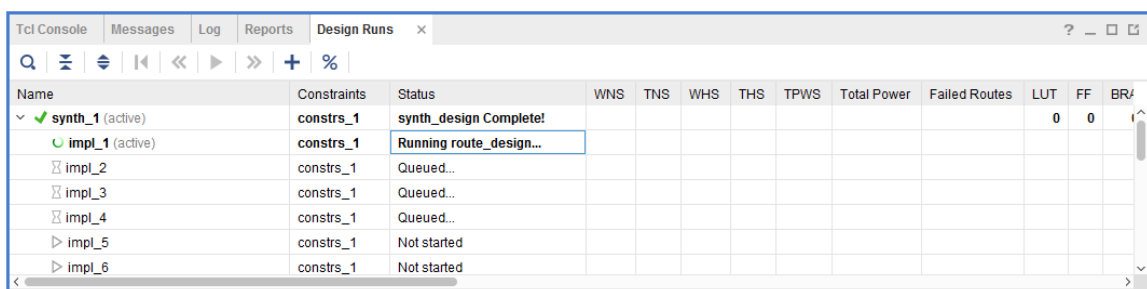*Figure 14:* **Creating Multiple Synthesis and Implementation Runs**

Managing Runs with the Design Runs Window

The Design Runs windows (shown in the following figure) 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.

Figure 15: **Design Runs Window**

Send Feedback

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).

- **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 more information, see Using Remote Hosts and Compute Clusters Appendix in the *Vivado Design Suite User Guide: Implementation* (UG904).

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).

Closing Timing Using Intelligent Design Runs

Intelligent design runs (IDR) uses a multi-stage run approach to automatically close timing on a design. This flow can be invoked in the GUI by right clicking on **implementation run** in the design runs window and selecting "**Close Timing Using Intelligent Design Runs**" or in Tcl, by creating a new run using the IDR flow and properly setting the reference run. For information on Intelligent design runs, see Chapter 8 of *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906).

Send Feedback

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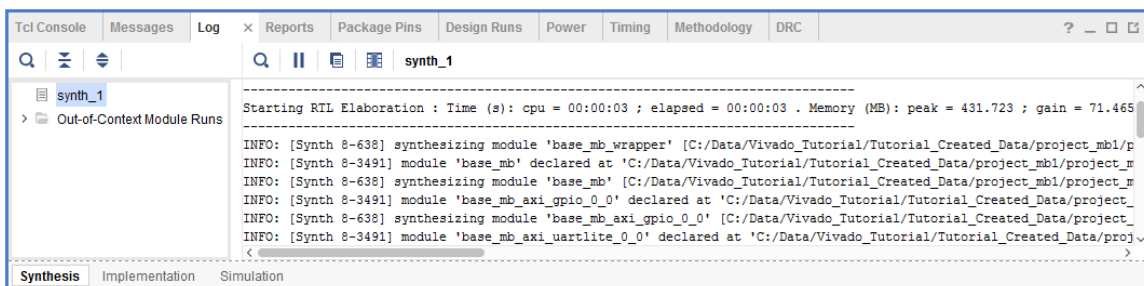
The advantage of the ECO flow is fast turn-around time by taking advantage of the incremental place and route features of the Vivado tool. The Vivado IDE provides a predefined layout to support the ECO flow. Refer to this link in the *Vivado Design Suite User Guide: Implementation* (UG904) for more information.

Viewing Log Files, Messages, Reports, and Properties

Viewing Log Files

In the Log window (shown in the following figure), 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 16: **Viewing Log Files**

Send Feedback

Viewing Messages

In the Messages window (shown in the following figure), 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).
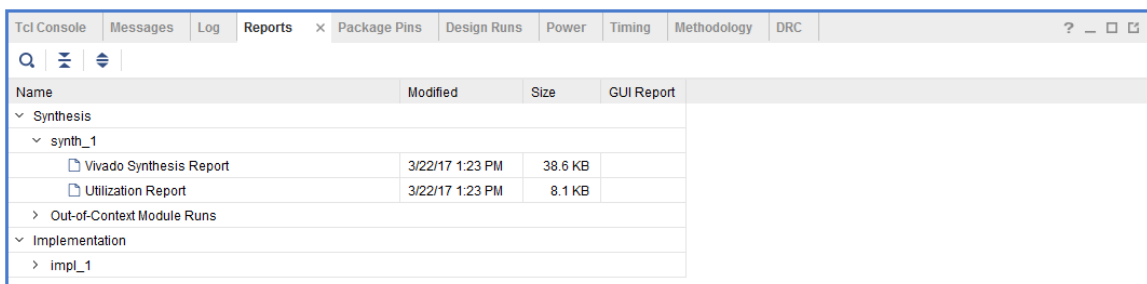
Figure 17: **Viewing Messages**

Viewing Reports

In the Reports window (shown in the following figure), 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) and see this link and this link in *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906).
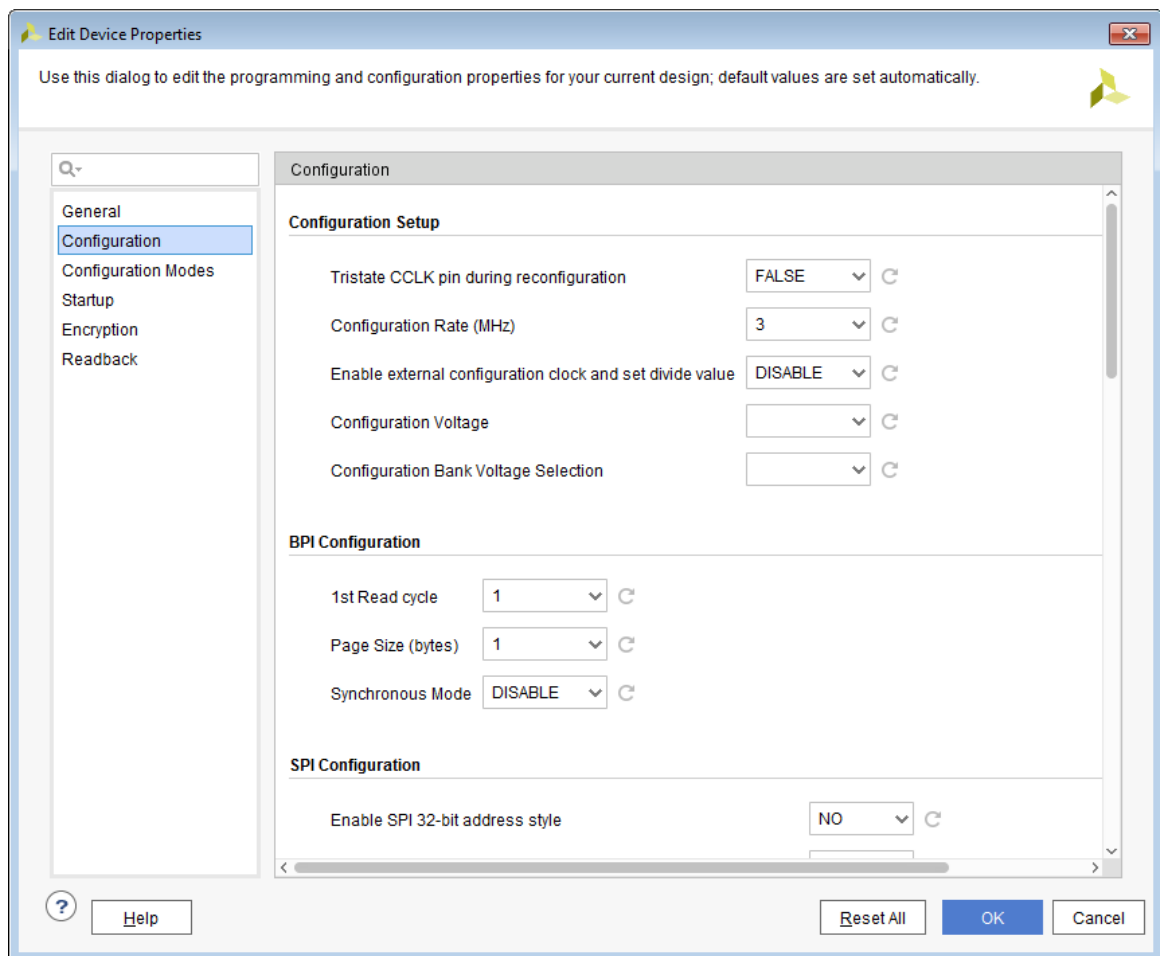
Figure 18: **Viewing Reports**

Send Feedback

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 (shown in the following figure) 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). For information on setting device configuration modes, see this link in the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899).

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

Figure 19: **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) and *Vivado Design Suite User Guide: Using Constraints* (UG903).

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).

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).

Send Feedback

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).

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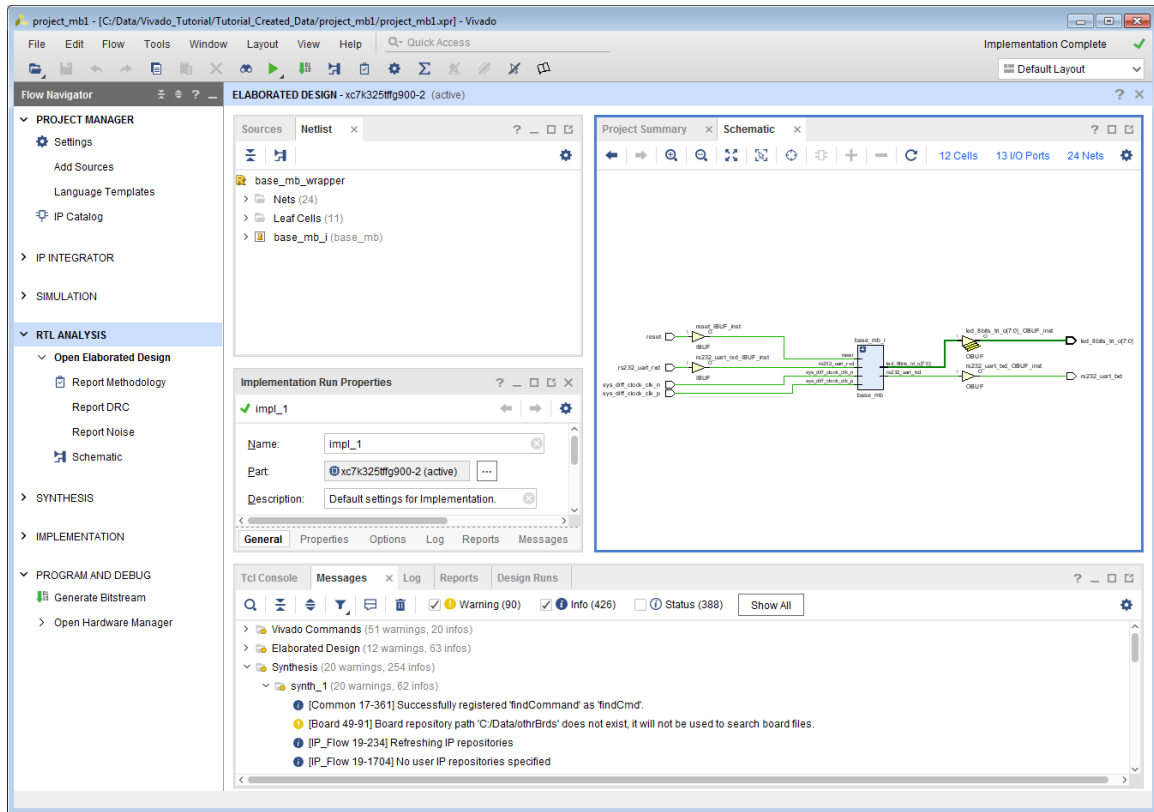
> **TIP:** *When you select the **Report DRC** command, the Vivado IDE invokes a set of RTL and I/O DRCs to identify logic issues such as asynchronous clocks, latches, and so forth. For more information, see this link in the Vivado Design Suite User Guide: System-Level Design Entry (UG895).*

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**.

The following figure 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 20: **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).

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) and see this link in *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906).
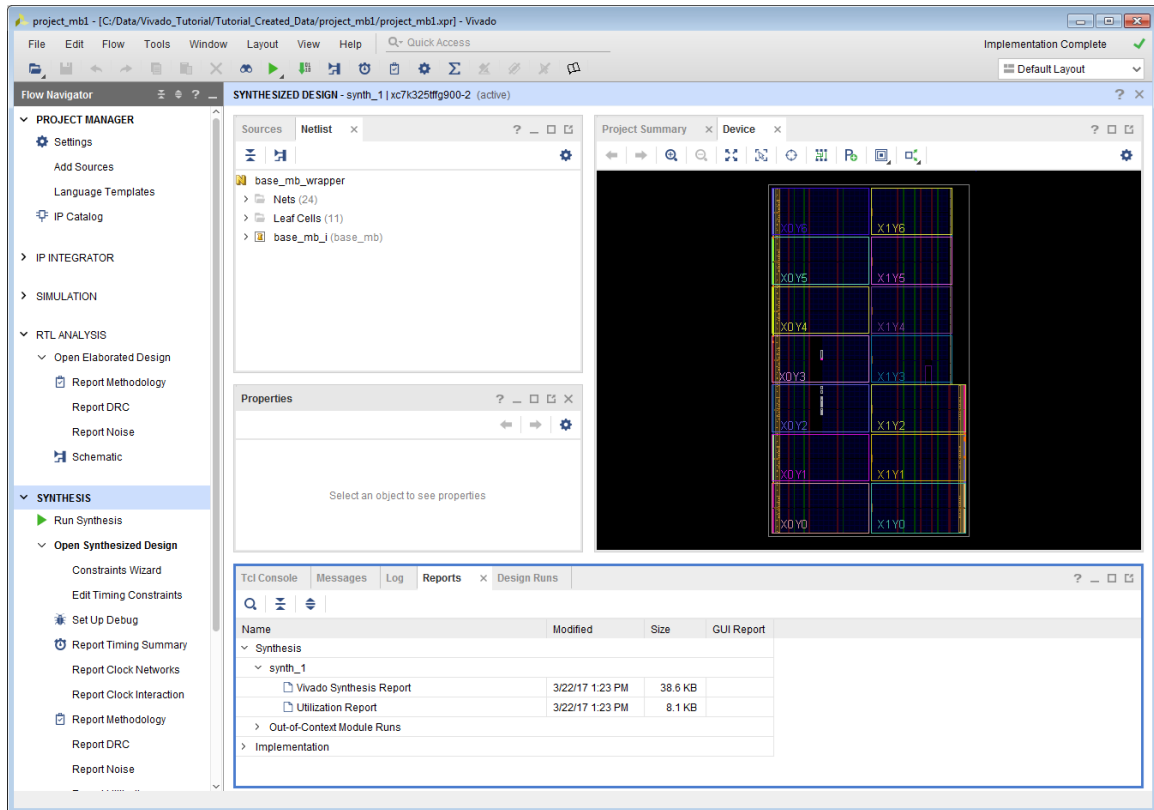
You can configure and implement debug core logic in the synthesized design to support test and debug of the programmed device. In the Schematic or Netlist windows, interactively select signals for debug. Debug cores are then configured and inserted into the design. The core logic and interconnect is 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).

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.

The following figure shows the default view layout for an open synthesized design.

Figure 21: **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

Send Feedback

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).

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).

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**.

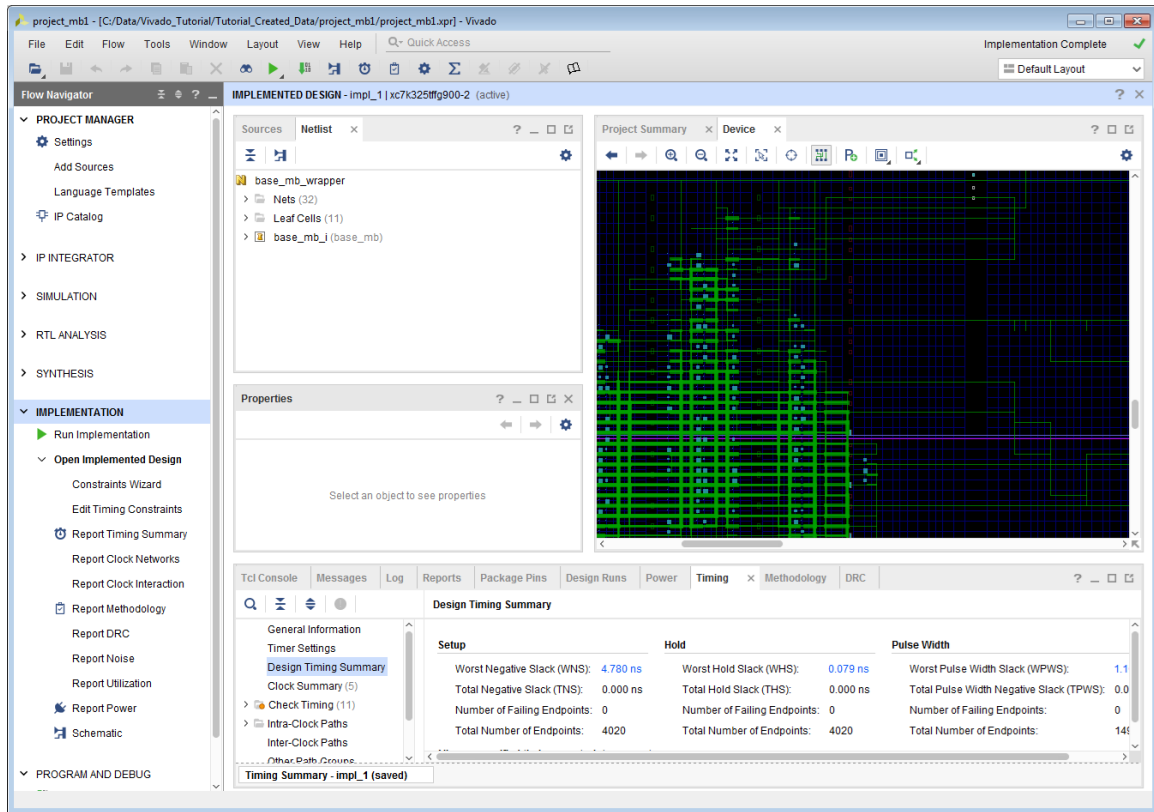- In the Design Runs view, double-click the run name.

💡 **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.*

The following figure 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.
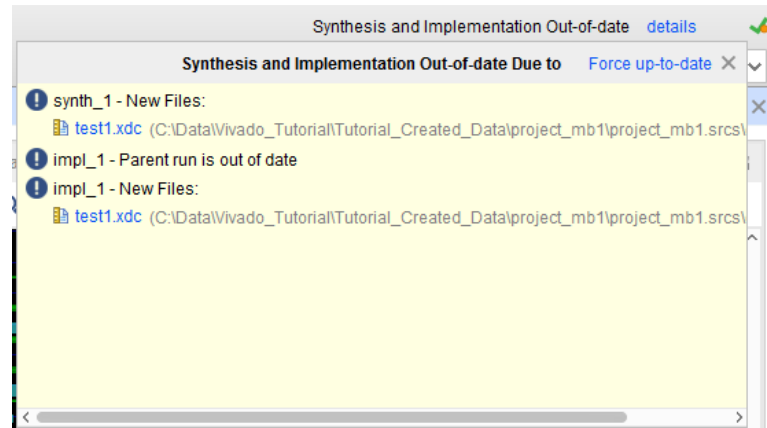
Figure 22: **Implemented Design View Layout**

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 (shown in the following figure). Click the associated **more info** link to view which aspects of the design are out of date.

Figure 23: **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.
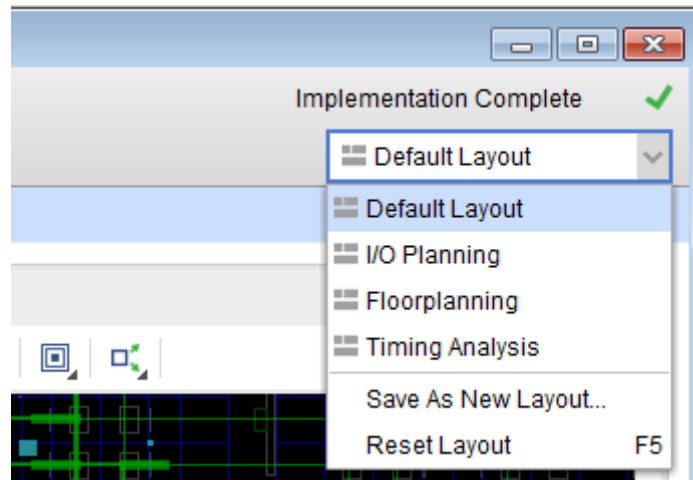
- Click **Reload** to refresh the in-memory view of the current design, eliminating any unsaved changes you made to the design data.

- 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 (shown in the following figure) 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.

Send Feedback

Figure 24: **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).

> **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.*

Related Information

Saving Design Changes to Design Checkpoints

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).

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), 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), or refer to the `report_xxx` commands in the *Vivado Design Suite Tcl Command Reference Guide* (UG835).

Report strategies enable you to define groups of reports and associate all the reports with a particular run. Report strategies can be created by using Tools→Settings→Strategies→Report Strategies. Individual reports can be specified for each step of a run. Report strategy is a property of a run. Setting report strategy on the run generates all specified reports when the run is launched.

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).

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) for information on the debug ECO flow.

Using Project Mode Tcl Commands

The following table 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: **Basic Project Mode Tcl Commands**

Command	Description
create_project	Creates the Vivado Design Suite project. Arguments include project name and location, design top module name, and target part.
add_files	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`). Individual files can be added, or entire directory trees can be scanned for legal sources and automatically added to the project. **Note:** The `.xco` file is no longer supported in UltraScale device designs.
set_property	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.
import_files	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.
launch_runs launch_runs -to_step	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 -to_step option is used to launch the implementation process, including bitstream generation, in incremental steps.
wait_on_run	Ensures the run is complete before processing the next commands in a Tcl script.
open_run	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.
close_design	Closes the in-memory design.
start_gui stop_gui	Opens or closes the Vivado IDE with the current design in memory.

Send Feedback

Note: This document is not a complete reference for the available Tcl commands. Instead, see the

Vivado Design Suite Tcl Command Reference Guide (UG835).

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/2020.2/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
```

Send Feedback

```
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

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 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.

Related Information

[Using Project Mode](#)

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

Send Feedback

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).*

For more information, see this link in the *Vivado Design Suite User Guide: Designing with IP* (UG896), or this link in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994).

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).

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).

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.*

Related Information

Non-Project Mode Tcl Script Example
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) or *Vivado Design Suite User Guide: Implementation* (UG904).

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).

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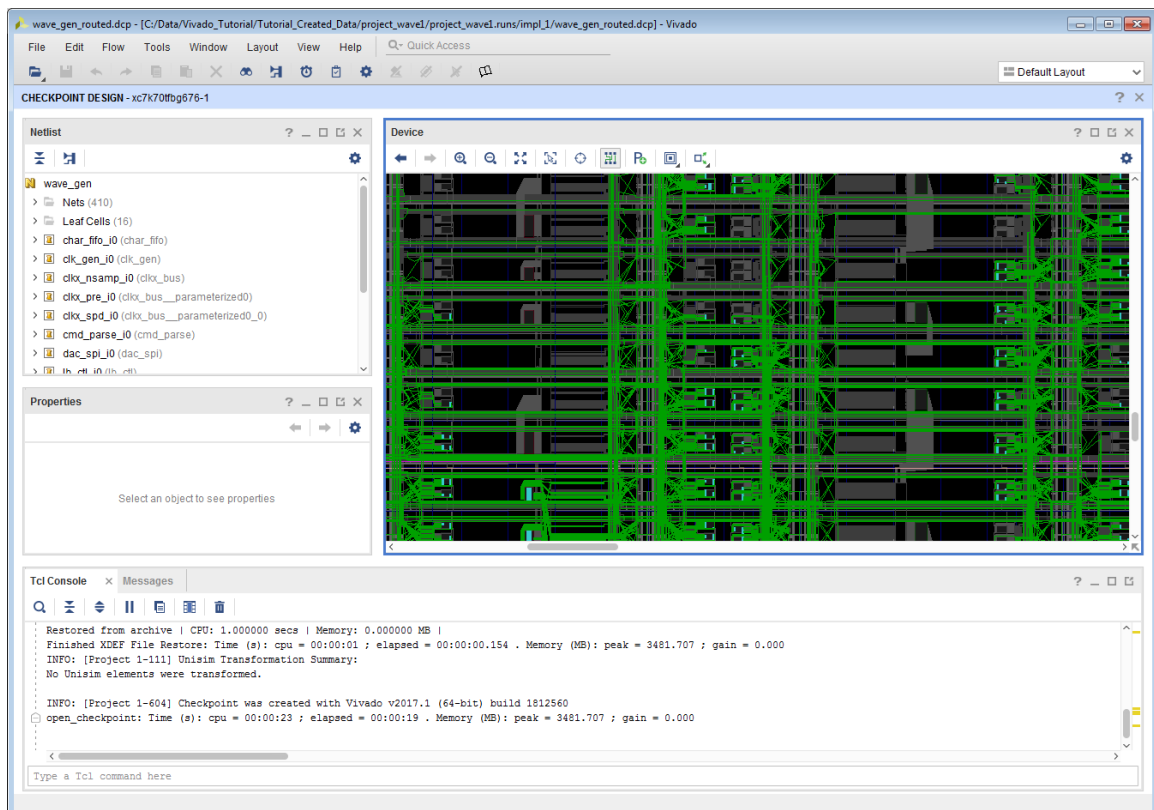
> ⚠️ **CAUTION!** *If you exit the Vivado Design Suite from the GUI, the Vivado Design Suite Tcl shell closes and does not save the design in memory. To return to the Vivado Design Suite Tcl shell with the active design intact, use the* `stop_gui` *Tcl command rather than the exit command.*

After each stage of the design process, you can open the Vivado IDE to analyze and operate on the current design in memory (shown in the following figure). 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 25: **Opening Vivado IDE with the Active Design**

Send Feedback

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).

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

The following table 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.

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

Send Feedback

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

Command	Description
read_edif	Imports an EDIF or NGC netlist file into the Design Source fileset of the current project.
read_verilog	Reads the Verilog (`.v`) and System Verilog (`.sv`) source files for the Non-Project Mode session.
read_vhdl	Reads the VHDL (`.vhd` or `.vhdl`) source files for the Non-Project Mode session.
read_ip	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.
read_checkpoint	Loads a design checkpoint into the in-memory design.
read_xdc	Reads the `.sdc` or `.xdc` format constraints source files for the Non-Project Mode session.
read_bd	Reads existing IP Integrator block designs (.bd) for the Non-Project session.
set_param set_property	Used for multiple purposes. For example, it can be used to define design configuration, tool settings, and so forth.
link_design	Compiles the design for synthesis if netlist sources are used for the session.
synth_design	Launches Vivado synthesis with the design top module name and target part as arguments.
opt_design	Performs high-level design optimization.
power_opt_design	Performs intelligent clock gating to reduce overall system power. This is an optional step.
place_design	Places the design.
phys_opt_design	Performs physical logic optimization to improve timing or routability. This is an optional step.
route_design	Routes the design.
report_*	Runs a variety of standard reports, which can be run at different stages of the design process.
write_bitstream	Generates a bitstream file and runs DRCs.
write_checkpoint	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.
start_gui stop_gui	Opens or closes the Vivado IDE with the current design in memory.

Related Information

Working with Tcl

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/2020.2/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 optimzation, 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
```

Send Feedback

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.

Vivado generates many intermediate files as it compiles a design. This chapter defines the minimum set of files necessary to recreate the design. In some cases, you might want to keep intermediate files to improve compile time or simplify their analysis. You can always optionally manage additional files.

Revision Control Philosophy from 2020.2 Onwards

In 2020.2, significant improvements are made to the Vivado project directory structure to improve the ability to interact with the revision control systems. For new projects created in 2020.2, an additional project directory called `project.gen` is automatically created in the project directory. This directory stores all output products created by the IP and Block Diagram (BD). The result of this change is that the `project.srcs` directory contains only the sources used to create the design. Moving forward, endorse a flow with the following behavior:

1. Create the project and add all the sources to the project

2. Manage `project.xpr` file

3. Manage `project.srcs` directory

4. Test your methodology. Revision control holds good depending on how well you test and maintain your methodology. Ideally, to ensure no files are missed and the design rebuilds completely from design sources using a script, the design would be regressed at a regular cadence. By rebuilding the design regularly, any issues with the revision control methodology can be caught and addressed in a timely manner.

The project can be re-created by restoring the `project.srcs` directory and `project.xpr` file. Opening the `project.xpr` file and proceeding with synthesis and implementation flows. Separating the output IP and BD output products from the `project.srcs` directory.

Revision Control Philosophy Pre 2020.2

The overall philosophy for revision controlling a design is to recreate the design from its sources using a Tcl script. The following steps outline how this is achieved:

1. Use a scripted flow for revision control. Scripted flows enable repeatability, the design can be recreated by sourcing the Tcl script.

2. Keep source files external to the project. Ideally, the source files are kept outside of the Vivado build directory. This helps to ensure separation of the source files and the tool generated files.

3. Revision control the source repository. All sources should be managed by the revision control system. It is important to note that when Vivado is using the source files, they should be writable.

4. Generate a script to recreate the design. Non-project flows are, by definition, scripted flows because the design is compiled strictly using Tcl commands. You would manually create this Tcl script. A project flow can be driven from the Vivado IDE or through a project based Tcl script. The recommendation is to use Tcl commands to recreate a project flow.

5. Revision control the script. Once the script is created, it is important to manage this file as a source too. As the design changes, this script is updated accordingly to accommodate new sources or to capture new design configurations. It is important that this script is managed like any other design source.

6. Test your methodology. Revision control holds good depending on how well you test and maintain your methodology. Ideally, to ensure no files are missed and the design rebuilds completely from design sources using a script, the design would be regressed at a regular cadence. By rebuilding the design regularly, any issues with the revision control methodology can be caught and addressed in a timely manner.

The subsequent sections of this chapter describe how this revision control philosophy should be applied to the scripted project flows. Non-project flow users should be aware of exactly which files are sources and which files are generated by the flow. They are also, rebuilding the design from scratch on each design iteration.

Send Feedback

Generating a Script to Recreate a Design

For a project flow, a script to recreate your design can be generated manually or by using the `write_project_tcl` command. The advantages of manually creating this script is that it remains short and well organized, but it could potentially miss some project settings and fail to recreate the complete design. Alternatively, the `write_project_tcl` script is robust in ensuring all files are captured appropriately. But its versatility results in a more complicated and more verbose script. Regardless of how this script is generated, it must be maintained as the design evolves.

Note: `write_project_tcl` recreates the design as originally created by the user. For designs using IP integrator, propagated parameters do not reflect in the recreated design until `validate_bd_design` is run.

Revision Controlling Projects with Only RTL Sources

An example of revision controlling an entirely RTL based design is shown in the following figure. In this case, there is a defined repository where all the RTL sources reside along with the script to rebuild the design. The script references the sources and rebuilds the design into your workspace. Sourcing the script from the workspace directory reproduce the complete design.

Figure 26: **Example of Revision Controlling**

Send Feedback

Revision Controlling Xilinx IP

The Xilinx® IP repository resides in the Vivado install area. For each IP there is a `component.xml` file that contains the VLNV of the IP and all the user customizable parameters for the IP. The parametrized RTL and associated IP constraints also reside in the IP repository. When you customize an IP, an `XCI` file is generated that contains the user desired parameters for the IP. When you generate the output products for an IP, the RTL and associated constraints are copied from the IP repository to the project directory and customized using the parameters specified in the `XCI` file. Thus, for a given version of Vivado, because the IP repo resides in the install area, an `XCI` file is the only file necessary to recreate the design. In this case, the `XCI` file should be managed by the revision control system and referenced in the script used to regenerate the design.

Note: Projects created prior to 2020.2, the IP output products are written to the `project.srcs` directory where the `XCI` file is residing. In order to facilitate a clear delineation between project sources and output products, for any new projects created using 2020.2, a `project.gen` directory is automatically created in parallel to the `project.srcs` directory. All IP output products are written to the `project.gen` directory.

Managing Custom IP Repositories

When you package a custom IP and share it among several projects, you should use a custom IP repository. In this case, the project used to create and package the custom IP should be revision controlled following the methodology specified in this chapter. The packaged IP with the `component.xml` file should reside in a custom revision controlled IP repository managed by you. The project that uses the IP should contain a pointer to your custom IP repository and an `XCI` file with the desired customizations applied to the IP. The script to recreate the project should set the IP repository to the custom IP repository path and add the `XCI` file as a source to the project. When the project is recreated, similar to a project with Xilinx IP, Vivado would copy the RTL and constraints from your custom IP repository to your local project directory and applies the parameters from the `XCI` file. The `XCI` files along with your custom IP repository are sufficient to regenerate the complete project.

Revision Controlling Block Diagrams

Block diagrams (BD) can contain instances of IPs from Xilinx repositories, IPs from custom IP repositories, references to RTL, or block design containers (references to other BDs). When a BD is validated, the customization of a single IP can affect how connected IPs are customized. The process of applying parameters from one IP to connected IP is coined parameter propagation and occurs when the BD is validated. To revision control a BD, the entire BD directory in the `project.srcs` should be managed by the revision control system. The directory contains the BD file, the `XCI` files for the IP post-parameter propagation and some meta data files.

Note: Projects created prior to 2020.2, the BD output products are written to the `project.srcs` directory where the BD file is residing. In order to help facilitate a clear delineation between project sources and output products, for any new projects created using 2020.2, a `project.gen` directory is automatically be created in parallel to the `project.srcs` directory. All BD output products are written to the `project.gen` directory. In turn, this significantly reduces the size of the BD directory from previous Vivado releases.

In the case of block design containers, there is a BD directory in the `project.srcs` directory for each BD source in the design. If there are several instances of a block design container on a parent BD, each instance of the block design container is generated in the `project.gen` directory. Each block design container instance, even though derived from the same source BD, can be unique due to parameter propagation. Therefore, the instances of the each block design containers reside in the `project.gen` directory, but the source from which they are all derived reside in the `project.srcs` directory. Any BD directories that reside in the `project.srcs` directory should be fully revision controlled.

Note: Projects created in 2020.2, IP and BD output products are no longer written to the `project.srcs` directory. The `project.srcs` directory should contain the bare minimum number of sources necessary to recreate the project with the exception of files that are referenced from directories external to the project. The cleanup of the `project.srcs` directory should tremendously improve the delimitation between files that are necessary to be revision controlled and tool generated files.

Note: To view the differences between two versions of a block diagram, see *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) to learn more about the `diffbd` (check spelling) utility.

Other Files to Revision Control

The project manages many other types of files required to rebuild a design. Following are a few examples:

- XDC files containing design constraints
- Simulation test benches
- HLS IP
- Pre/post Tcl hook scripts used for synthesis or implementation
- Incremental compile DCPs
- ELF files

These files reside in the `project.srcs` directory or the `project.util` directory. It is important to manage these files to accurately reproduce your design. Routinely rebuilding your designs from your revision control system should help you catch any of these files that you could inadvertently miss in your build script.

Output Files to Optionally Revision Control

Following is a list of additional files you may consider revision controlling:

- Simulation scripts for third-party simulators generated by `export_simulation`. Because these are typically hand-off files between design and verification, you might want to snapshot them at different stages of the design process.

- XSA files. These are hardware hand-off files between Vivado and Vitis™ software platform.

- Bitsteams/PDIs.

- LTX files for hardware debug

- Intermediate DCP files created during the flow

- IP output products. These are usually considered output products, but if you do not want to upgrade an IP when migrating between Vivado releases, then you must manage the output files. Vivado only provides one version of each IP per release. If the IP you are using is updated between Vivado releases then there are only two choices:

  - Upgrade the IP to the latest version in the latest Vivado release. This may cause you to change your design to accommodate IP RTL changes.

  - Revision control the output products of the IP. The IP will be locked in the newer version of Vivado because you will not be able to re-customize the IP. But, this allows you to carry the IP forward to a future release because you are essentially capturing all the RTL sources for the IP.

Archiving Designs

The `archive_design` command can compress your entire project into a zip file. This command has several options for storing sources and to run results. Essentially, the entire project is copied locally in the memory and then zipped into a file on the disk while leaving the original project intact. This command also copies any remote source 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)

- Vivado Design Suite QuickTake Video: Creating Different Types of Projects

- Vivado Design Suite QuickTake Video: Managing Sources with Projects

Send Feedback

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.

Send Feedback

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 (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open 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 DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

References

These documents provide supplemental material useful with this guide:

1. *Vitis Model Composer User Guide* (UG1483)
2. *Vivado Design Suite User Guide: High-Level Synthesis* (UG902)
3. *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (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. *Vitis Model Composer User Guide* (UG1483)
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: Dynamic Function eXchange* (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)

Send Feedback

27. *Vivado Design Suite Tutorial: Dynamic Function eXchange* (UG947)

28. *UltraFast Design Methodology Guide for Xilinx FPGAs and SoCs* (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 Guide* (UG1046)

32. *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118)

33. *Vivado Design Suite Tutorial: Creating, Packaging Custom IP* (UG1119)

34. 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

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
04/20/2022 Version 2022.1	
General Updates	Editorial updates only. No technical content updates.

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