Vivado Design Suite
User Guide

Implementation

UG904 (v2012.2) August 20, 2012
Notice of Disclaimer
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 the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; 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 Critical Applications: http://www.xilinx.com/warranty.htm#critapps.
© Copyright 2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History
The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>07/25/2012</td>
<td>2012.2</td>
<td>Initial Xilinx release.</td>
</tr>
<tr>
<td>08/20/2012</td>
<td>2012.2</td>
<td>Added details of route_design command regarding re-entrant routing and timing constraints.</td>
</tr>
</tbody>
</table>
# Table of Contents

## Chapter 1: Implementation Process

- **Implementation Overview** .................................................. 5
- **Getting to Implementation** ................................................. 7
  - Project and Non-Project Modes ........................................ 7
  - RTL and Synthesized Design ............................................. 8
  - Constraints ............................................................... 10
  - Design Checkpoints ..................................................... 11
- **Running Implementation in Non-Project Mode** ....................... 12
  - Non-Project Mode Sample Script ..................................... 12
  - Details of Sample Script ............................................... 13
- **Running Implementation in Project Mode** .............................. 15
  - Creating Implementation Runs ....................................... 15
  - Customizing Implementation Strategies ............................ 24
  - Launching Runs ......................................................... 28
  - Running Implementation in Steps .................................... 30
  - Monitoring the Implementation Run .................................. 31
  - Determining the Project Status ....................................... 33
- **Moving Forward After Implementation** .................................. 34
  - Viewing Messages ....................................................... 35
  - Viewing Implementation Reports ..................................... 36

## Chapter 2: Implementation Commands

- **Introduction** ................................................................... 40
- **Opening the Synthesized Design** ...................................... 41
  - synth_design ............................................................. 41
  - read_checkpoint ......................................................... 42
  - open_run ..................................................................... 42
  - link_design ............................................................... 43
- **Logic Optimization** .......................................................... 44
  - opt_design ................................................................. 44
  - Logic Optimization Constraints ...................................... 45
- **Power Optimization** .......................................................... 45
  - power_opt_design .......................................................... 46
- **Placement** ..................................................................... 47
  - place_design ............................................................... 48
- **Physical Synthesis** ........................................................... 49
  - phys_opt_design .......................................................... 49
- **Routing** ...................................................................... 50
  - route_design ............................................................... 50
Appendix A: Using Remote Hosts

Launching Runs on Remote Linux Hosts ............................................. 53
Configuring Remote Hosts ................................................................. 53
Setting Up SSH Key Agent Forward .................................................... 56

Appendix B: Additional Resources

Xilinx Resources ................................................................................. 57
Solution Centers .................................................................................. 57
References ............................................................................................ 57
Implementation Process

Implementation Overview

The Vivado™ Design Suite enables implementation of Xilinx® 7 series FPGA designs from a variety of design sources, such as RTL designs, netlist designs, and IP centric design flows, as illustrated in Figure 1-1. For a complete understanding of the different design flows supported by the Vivado tools, refer to the Vivado Design Suite User Guide: Design Flows Overview (UG892). Vivado implementation encompasses all of the design steps required to place and route the netlist onto the FPGA device resources while meeting the logical, physical, and timing constraints of the design.

Vivado implementation is a timing-driven flow with native support of industry standard Synopsys Design Constraints (SDC) commands to specify design requirements and restrictions. Xilinx has added additional commands in the Xilinx Design Constraints format (XDC).

Figure 1-1: Vivado Design Suite High-Level Design Flow
The Vivado implementation process includes logical as well as physical transformations of the design, and consists of the following sub-processes:

1. **Opt Design**: Optimize the logical design to make it easier to fit into the target Xilinx FPGA.
2. **Power Opt Design**: Optionally optimize elements of the design to reduce power demands of the implemented FPGA.
3. **Place Design**: Place the design onto the target Xilinx device.
4. **Phys Opt Design**: Optionally optimize the timing of the design by replicating drivers of high-fanout nets to distribute the loads.
5. **Route Design**: Route the design onto the target Xilinx device.
6. **Write Bitstream**: Generate a bitstream for Xilinx device configuration.

The complete design flow is integrated within the Vivado Integrated Design Environment (IDE) which provides a standardized interface called the **Flow Navigator** to assemble, implement and validate the design and IP of the FPGA design. The Flow Navigator provides a push-button interface to the entire implementation process, as shown in Figure 1-2, to simplify the design flow. This guide does not provide a detailed examination of the Vivado IDE, except as it applies to implementation. For a complete understanding of the Vivado IDE as it relates to the entire FPGA design flow, refer to the *Vivado Design Suite User Guide: Using the Integrated Design Environment (UG893)*.

![Figure 1-2: Flow Navigator - Implementation Section](image)

Vivado design tools also provide a tool command language (Tcl) application programming interface (API). The Tcl API enables scripting support for all design flows, allowing you to customize the design flow to meet your specific requirements.
Getting to Implementation

The Vivado Design Suite offers a variety of design flows, and supports an array of design sources. However, to get to a bitstream that can be downloaded into an FPGA, the design must pass through implementation. This is a series of steps that takes the logical netlist and maps it into the physical array of the target Xilinx device. These steps include logic optimization, placement of logic cells, and routing of connections between cells. The following section describes what is needed to get your design into implementation in the Vivado Design Suite.

Project and Non-Project Modes

The Vivado tools let you create a project file (\texttt{.xpr}), and directory structure, to manage the design source files, store the results of different synthesis and implementation runs, and track the project status through the design flow. However, the Vivado Design Suite also let you work strictly in memory, without the need for a project file and local directory.

In Project Mode, or a Project-based design, a directory structure is created on disk to help you manage design sources, run results, and project status. The automated management of the design data, process and status requires a project infrastructure which is stored in the Vivado project file (\texttt{xpr}). In addition, the Project Mode automatically writes checkpoint files into the local project directory at key points in the design flow.

Working without a project file, in the compilation style flow, is referred to as Non-Project Mode, or the Non-Project batch flow. A powerful feature of the Non-Project Mode is the ability to work with the design in memory. Source files and design constraints are read into memory from their current locations, and the in-memory design is stepped through the design flow without having to be written to intermediate files. In Non-Project Mode, you must run each design step individually and set design parameters and implementation options using Tcl commands. You can apply design changes and proceed through the design flow without having to save changes and rerun steps. You can run reports and save design checkpoints (\texttt{.dcp}) at any stage of the design flow.

\textbf{IMPORTANT:} In Non-Project Mode, when you exit the Vivado design tools, the in-memory design is lost. For this reason, you should write design checkpoints after major steps such as synthesis, placement, or routing.

You can save design checkpoints in both Project Mode and Non-Project Mode. You can only read design checkpoints in Non-Project mode.

There are many distinctions between Project Mode and Non-Project Mode in the Vivado Design Suite. Vivado implementation can be run using either project-based designs, or non-project based designs. The Vivado IDE and Tcl API can also be used with both.
project-based, and non-project based designs. However, some of the features of the Vivado IDE, and Vivado implementation, are not available in Non-Project Mode:(1)

- Flow Navigator
- Design status indicators
- IP integration
- Implementation Runs and Run Strategies
- Design Runs window
- Messages window
- Reports window

You must implement the non-project based design by running the individual Tcl commands: `opt_design`, `place_design`, and `route_design`. You can run implementation steps interactively in the Tcl Console or the Vivado IDE, or by using a custom Tcl script. You can also customize the design flow as needed to include reporting commands and additional optimizations. See Running Implementation in Non-Project Mode, page 12.

The details of running implementation in Project Mode and Non-Project Mode are described in this guide. However, for more information on running the Vivado Design Suite using either Project Mode or Non-Project Mode, refer to the Vivado Design Suite User Guide: Design Flows Overview (UG892) and Vivado Design Suite User Guide: Using the Integrated Design Environment (UG893).

**RTL and Synthesized Design**

You can use the Vivado Design Suite to manage the entire FPGA design process including RTL development, IP customization, synthesis, implementation through to programming and validating the device. You can add HDL sources such as Verilog, SystemVerilog, and VHDL files. You can add previously defined and configured (IP) cores from the Xilinx IP catalog. You can add digital signal processing (DSP) modules from System Generator, C-based DSP modules from Vivado High-level Synthesis (HLS), and embedded processor modules from Xilinx Platform Studio (XPS).

Vivado Design Suite also supports netlist-driven design by importing previously synthesized netlists from Xilinx or third-party tools. The netlist input formats include structural Verilog or SystemVerilog, EDIF, or Xilinx NGC. For more information on the different source files and project types supported by the Vivado Design Suite, refer to Vivado Design Suite User Guide: System Design Entry (UG895).

---

1. This list is representative of the features that are not supported in Non-Project Mode. It is not intended to be an exhaustive list.
At a minimum, Vivado implementation requires a synthesized netlist. A design can start from a synthesized netlist, or it can start from RTL source files. When starting from RTL sources, the Vivado synthesis, or XST, must be run before implementation can begin. The Vivado IDE will manage this automatically if you attempt to run implementation on an un-synthesized design, and give you the option to run synthesis first. For information on running Vivado synthesis, see the Vivado Design Suite User Guide: Synthesis (UG901).

In non-Project Mode, you must use the Tcl command, synth_design, to create and open the synthesized design. You can also use the Tcl command, link_design, to open the design. See Opening the Synthesized Design in Chapter 2 for more information.

In Project Mode, after synthesis of an RTL design, or with a netlist-based project open, you can load the design netlist prior to implementation. You can open a synthesized design using one of the following methods in the Vivado IDE:

- Select **Flow > Open Synthesized Design** from the main menu.
- In the Flow Navigator, select **Open Synthesized Design** from the Synthesis section.
- In the Design Runs window, using the **Open Synthesized Design** command from the popup menu.

**IP Centric Design**

The Vivado IP catalog feature lets you configure, implement, and verify IP. The IP can be configured and verified as a standalone module, or within the context of a larger system level design.

The IP Catalog displays all available Xilinx LogicCORE™ IP, as well as any user-defined IP or third party IP that has been added to the catalog. The catalog includes data related to the IP type, version, datasheet, and license information. You can add an IP core to an RTL design by defining the instantiation template into the system-level design. IP is created as RTL sources, not netlists. Running synthesis and implementation will implement the IP along with the rest of the design. However, you can also synthesize the IP as a standalone module and add the netlist to a netlist design. The supported IP netlist formats include Xilinx formats (.xco, .xci, .ngc), Verilog (.v), and EDIF (.edf). For more information on how Vivado tools support IP centric design, refer to the Vivado Design Suite User Guide: IP Design (UG896).
Constraints

In addition to a synthesized netlist, design constraints should be provided to guide implementation. A constraint set is a set of constraints files, containing design constraints captured in XDC files, that can be applied to your design. There are two types of design constraints:

• Physical constraints define pin placement, and absolute, or relative, placement of cells such as BRAMs, LUTs, and Flip Flops, and device configuration settings.

• Timing constraints, written in industry standard SDC, define the frequency requirements for the design. Without timing constraints, the Vivado Design Suite will optimize the design solely for wire length and routing congestion.

**Note:** Without timing constraints, Vivado implementation will make no effort to assess or improve the performance of the design.

**IMPORTANT:** The Vivado Design Suite does not support the UCF format. For information on migrating UCF constraints to XDC commands refer to the Vivado Design Suite Migration Methodology Guide (UG912) for more information.

Within a constraint set, you can have multiple constraints files. You can have constraint sets with separate physical and timing constraint files. You can have a master constraints file, and direct design changes to a new constraints file. Separating constraints by function into different constraint files, can make your overall constraint strategy clearer, and facilitate targeting timing and implementation changes.

You can also have multiple constraint sets for a project, and make different constraint sets active for different implementation runs to test different approaches. You can have separate constraint sets for synthesis and for implementation. You can experiment by applying different constraints during synthesis, simulation, and implementation to help meet your design objectives. Organizing design constraints into multiple constraint sets can help you:

• Target different Xilinx FPGAs for the same project. Different physical and timing constraints may be needed for different target parts.

• Perform what-if design exploration. Using constraint sets to explore different scenarios for floorplanning and over-constraining the design.

• Manage constraint changes. Override master constraints with local changes in a separate constraint file.

**TIP:** A good way to validate the timing constraints is to run the `report_timing_summary` command on the synthesized design. Problematic constraints should be fixed before implementation.

For more information on defining and working with constraints, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).
Design Checkpoints

In addition to the design netlist and constraints, which provide inputs to the implementation process, the Vivado Design Suite uses a physical design database to store placement and routing data. The Vivado tools provide design checkpoint files (.dcp) as a mechanism to save and restore this physical database at key steps in the design flow. Checkpoints are merely a snapshot of a design at a specific point in the flow.

The current netlist, including any optimizations made during implementation, the design constraints, and any implementation results, are stored in the design checkpoint file. Checkpoint designs can be run through the remainder of the design flow using Tcl commands, but cannot be modified with new design sources.

You can write and read checkpoint files using the following commands from the Vivado IDE:

- **File > Write Checkpoint**: Captures a snapshot of the design database at any stage in the flow. This creates a file with a .dcp file extension. The related Tcl command is `write_checkpoint`.
- **File > Open Checkpoint**: Opens the specified checkpoint in the Vivado Design Suite. The design checkpoint is opened as a separate project in the Vivado design tools, and cannot be read into an existing project. The related Tcl command is `read_checkpoint`.
Running Implementation in Non-Project Mode

To implement the synthesized design or netlist onto the targeted Xilinx FPGA, you must run the netlist and the design constraints through a series of steps: optimization, placement, and routing, collectively known as implementation.

In the Non-Project Mode, you must run implementation using a series of Tcl commands, or a Tcl script which defines the overall design flow. The Tcl commands can be entered into the Tcl Console from within the Vivado IDE, or can be entered from the Tcl prompt in the Vivado Design Suite Tcl shell.

Non-Project Mode Sample Script

The following script provides an example of running implementation in Non-Project Mode.

```
# Step 1: Read in top-level EDIF netlist from synthesis tool
read_edif c:/top.edf
# Read in lower level IP core netlists
read_edif c:/core1.edf
read_edif c:/core2.edf

# Step 2: Specify target device and link the netlists
# Merge lower level cores with top level into single design
link_design -part xc7k325tfbg900-1 -top top.edf

# Step 3: Read XDC constraints to specify timing requirements
read_xdc c:/top_timing.xdc
# Read XDC constraints that specify physical constraints such as pin locations
read_xdc c:/top_physical.xdc

# Step 4: Optimize the design with default settings
opt_design

# Step 5: Place the design with effort level set to high
place_design -effort_level high

# Step 6: Route the design with effort level set to high
route_design -effort_level high

# Step 7: Run Timing Summary Report to see timing results
report_timing_summary -file post_route_timing.rpt
# Run Utilization Report for device resource utilization
report_utilization -file post_route_utilization.rpt

# Step 8: Write checkpoint to capture the design database;
# The checkpoint can be used for design analysis in Vivado IDE or TCL API
write_checkpoint post_route.dcp
```
Details of Sample Script

The key steps of the preceding script can be broken down as follows:

1. Read design source files.

   In this case the design sources are EDIF netlist files. However, the Non-Project Mode also supports an RTL design flow, reading source files and running synthesis prior to implementation.

   The read_* Tcl commands are designed for use with the Non-Project Mode, as it allows a file on the disk to be read by the Vivado Design Suite to build the in-memory design, without copying the file or creating a dependency on the file in any way. The advantages of this approach make the Non-Project Mode extremely flexible with regard to design. However, a limitation is that you are required to monitor any changes to the source design files, and to update the design accordingly.

2. Build the in-memory design.

   In this example script, the Vivado tools build an in-memory view of the design using the link_design command. This command takes the netlist based source files read into the tool, and combine them with the Xilinx part information to create a design database in memory. All actions taken in the Non-Project Mode are directed at the in-memory database within the Vivado tools.

   The in-memory design resides in the Vivado tool, whether running in batch mode, Tcl shell mode for interactive Tcl commands, or in the Vivado IDE for interaction with the design data in a graphical form.

3. Read design constraints.

   The Vivado Design Suite uses design constraints to define requirements for both the physical and timing characteristics of the design, as explained in Constraints, page 10. The read_xdc command reads an XDC constraints file and applies it to the in-memory design.

   **TIP:** While the Project Mode supports the definition of constraint sets, containing multiple constraints files for different purposes, the Non-Project Mode uses multiple read_xdc commands to achieve the same effect.

4. Perform Logic Optimization.

   The example script performs logic optimization in preparation for placement and routing. The objective of optimization is to simplify the logic design before committing to physical resources on the target part. The Vivado logic optimizer provides many different types of optimizations to meet different design requirements. See Logic Optimization, page 44 for more information.
5. Place the Basic Logic Elements.

The example script performs a general placement of the overall design. However, placement can also be accomplished in stages, according to the hierarchy of the design, or the complexity of the placement challenge. See Placement, page 47 for more information.

6. Route the Design.

This command completes the required routing for the design. The Vivado router provides a general purpose route for all occasions, and also provides a great deal of control for re-entrant routing to complete challenging designs. See Routing, page 50 for more information.

7. Run required reports.

The example script generates two of the many reports available from the Vivado Design Suite. In the Non-Project Mode, you must specify each of the commands you want to create, using the appropriate Tcl command. You can output reports to files, for later review, or you can direct the reports to the Vivado IDE for more interactive examination. See Viewing Implementation Reports, page 36 for more information.

8. Save the Design Checkpoint prior to exiting the Vivado tool.

Finally, the example script saves the in-memory design, with its optimized netlist, the physical and timing related constraints, the Xilinx part data, and placement and routing information, into a design checkpoint file. In Non-Project Mode, the design checkpoint file saves the design and allows it to be reloaded for further analysis and modification. See Design Checkpoints, page 11 for more information.
Running Implementation in Project Mode

In Project Mode, the Vivado IDE allows you to define implementation runs that are configured to use specific synthesis results and design constraints, and to run multiple strategies on a single design. You can also customize implementation strategies to meet specific design requirements, and save those strategies for use in other designs.

**IMPORTANT:** The Non-Project Mode does not support predefined implementation runs and strategies. Non-project based designs must be manually moved through each step of the implementation process using Tcl commands. See Running Implementation in Non-Project Mode, page 12 for more information.

Creating Implementation Runs

You can create and launch new implementation runs to explore design alternatives and find the best results. You can queue and launch the runs serially, or in parallel using multiple local CPUs. On Linux systems, you can also launch runs on remote servers, see Appendix A, Using Remote Hosts.

An implementation run can be defined using one of the following methods:

- Select **Flow > Create Runs** from the main menu.
- In the Flow Navigator, select **Create Implementation Runs** from the Implementation popup menu.
- In the Design Runs window, using the **Create Runs** command from the popup menu.

The Create New Runs wizard opens. The first page of the wizard is a summary of the command. Click **Next** to proceed.

**Note:** If you used **Flow > Create Runs**, you will need to select Implementation on the first page of the Create New Runs wizard.

The Set-Up Implementation Runs dialog box opens, as shown in Figure 1-3, page 16.
1. Select a **Synthesized netlist**: 

   This lets you select a synthesis run that will be used to generate, or has already generated, the synthesized netlist to be implemented. Alternatively, you can select a synthesized netlist that was imported into the project from a third party synthesis tool. See the *Vivado Design Suite User Guide: Synthesis (UG901)* for more information.

   The default will be the currently active synthesis run in the Design Runs window, see Using the Design Runs window, page 19.

2. Select a **Constraints set**: 

   This specifies the constraint set to apply during implementation. The optimization, placement, and routing are largely directed by the physical and timing constraints in the specified constraint set. See the *Vivado Design Suite User Guide: Using Constraints (UG903)* for more information.

3. Select a target **Part**: for the run.

   The default values for Constraints Set and Part are defined by the Project Settings at the time the Create New Runs command is run. See the *Vivado Design Suite User Guide: Using the Integrated Design Environment (UG893)* for more information on the Project Settings.
To create runs with different constraint sets or target parts, you must use the Create New Runs command. You can also change these values in the Run Properties window, by selecting the run in the Device Runs window as explained later. See Changing Implementation Run Settings, page 20.

4. Click **Next**.

The Choose Implementation Strategies dialog box opens, as shown in Figure 1-4.

![Choose Implementation Strategies](image)

*Figure 1-4: Choose Implementation Strategies*

5. Enter a **Name** for the run, or accept the default name.

6. Select a **Strategy** for the new run.

   The strategies are a defined set of Vivado implementation feature options controlling the implementation results. Vivado Design Suite offers a set of pre-defined strategies to select from, and you can also create your own implementation strategies. For more information see Defining Strategies, page 25.

   - **Vivado Implementation Defaults**: Balances runtime with trying to achieve timing closure.
   - **HighEffort**: Puts more focus on timing closure, with increased runtime.
   - **HighEffortPhySynth**: Increased focus on timing closure, with increased runtime. Enables physical synthesis after placement.
   - **LowEffort**: Low placement and routing effort, useful for early implementation passes.
**QuickEffort**: Non-timing driven place and route. This strategy ignores all timing constraints, and should only be used to evaluate the ability to place and route the unconstrained design.

Prior to launching a run, you can change the settings for each step in the implementation process, overriding the default settings for the selected strategy. You can also save those new settings as a new strategy. See Changing Implementation Run Settings, page 20 for more information.

7. You can optionally choose **Make Active** for the run to make a new run the active run. If you are creating multiple new runs, only one run can be made the active run. The Vivado IDE will display run results information for the active run.

8. Click the **More** button to define additional runs. Specify names and strategies for the added runs as shown in **Figure 1-4, page 17**.

9. Click **Next**.

The Launch Options dialog box opens as shown in Implementation Launch Options, **Figure 1-5, page 18**.

10. **Specify the Launch Directory**: to create and store the implementation run data.

    The default directory is contained within the local project directory structure. Files for implementation runs are stored by default at:

    `<project_name>/project_name.runs/running_name`
11. Specify the Launch Options:
   - **Launch Runs on Local Host**: Launch the run on the local machine.
     - **Number of Jobs**: Define the number of local processors to use when launching multiple runs simultaneously. Individual runs are launched on each processor. Vivado does not support multi-threading for implementation.
   - **Launch Runs on Remote Hosts** (Linux only): Use remote hosts to launch one or more jobs. See Appendix A, Using Remote Hosts.
     - **Configure Hosts**: Select this option to configure remote hosts.
   - **Generate scripts only**: Export and create the run directory and run script, but do not launch the run at this time. The script can be run at a later time outside of the Vivado IDE tool.
   - **Do not launch now**: Save the new runs, but do not launch or create run scripts at this time.

12. Click **Next** and review the Create New Runs Summary.

13. Click **Finish** to create the defined runs and execute the specified launch options.

New runs are added to the Design Runs window.

**Using the Design Runs window**

The Design Runs window displays all of the synthesis and implementation runs created in a project, and provides commands to configure, manage, and launch the runs.

Select **Window > Design Runs** to open the Design Runs window if it is not already displayed. Figure 1-6, page 20 shows the Design Runs window.

Each implementation run appears indented beneath the synthesis run it is a child of. A synthesis run can have multiple implementation runs. You can expand and collapse synthesis runs using the tree widgets in the window. The Design Runs window is a tree table window. Refer to *Design Suite User Guide: Using the Integrated Design Environment (UG893)*, for more information on working with the columns to sort the data in this window.
Running Implementation in Project Mode

The Design Runs window reports the run status, including when the run has not been started, is in progress, is complete, or is out-of-date. Runs can become out-of-date when source files, constraints or project settings are modified. You can reset and delete stale run data in the Design Runs window.

Only one synthesis run and one implementation run can be “active” in the Vivado IDE at any time. All views in the Vivado IDE will reference the active run. The Log and Report views, Status Bar, and Project Summary display information for the active run. The Project Summary window only displays compilation, resource, and summary information for the active run.

The active run is displayed with bold text. To make a run active, select the run in the Design Runs window and use the Make Active command from the popup menu.

Changing Implementation Run Settings

When you select a run in the Design Runs window, the Run Properties window displays the current configuration of the selected run, as shown in Figure 1-7. In the Run Properties window you can change:

- The Name of the run.
- The Xilinx Part targeted by the run.
- The run Description.
- The Constraints set that both drives the implementation and is the target of new constraints from implementation.

You can also change the options used by Vivado implementation features. Select a run in the Design Runs window, and use the **Change Run Settings** command from the popup menu to open the Design Run Settings dialog box, as shown in Figure 1-8, page 21.

**TIP:** You can only change the settings for a run that has a **Not Started** status. You can use Reset Run to return a run to the Not Started status. See **Resetting and Deleting Runs**, page 23.

---

![Design Run Settings](image-url)
The Design Run Settings dialog box displays the implementation strategy currently employed by the run, and the command options associated with that strategy for each step of the implementation process:

- **Strategy:** Select the strategy to use for the implementation run. Vivado Design Suite offers a set of pre-defined strategies to select from, and you can also create your own implementation strategies. For more information see Defining Strategies, page 25.

- **Description:** Displays a textual description of the selected implementation strategy.

- **Options:** When you select a strategy, each step of the Vivado implementation process displays in a table in the lower part of the dialog box: opt_design, power_opt_design, place_design, phys_opt_design, route_design, and write_bitstream.

When you click on the name of a specific command option, a brief description of the option displays at the bottom of the Design Run Settings dialog box. See Chapter 2, Implementation Commands for a detailed discussion of the different implementation steps, and their available options.

You can modify command options by clicking on the right-side column of a specific option to modify:

- Options with predefined settings can be selected from a pull down menu.
- Options can be enabled or disabled by a check box.
- Options accepting a user-defined value can be defined by typing a value.
- Options accepting a file name and path will open a file browser dialog to let you locate and specify the file.
- A custom Tcl script can also be inserted before (tcl.pre) and after (tcl.post) each step, allowing you to perform specific tasks either before or after the implementation step. An example of this would be to generate a timing report before and after Place Design to compare timing results.

- **Save Strategy As:** This command, which appears as an icon next to the Strategy field, saves any changes to the strategy as a new strategy for future use. If you do not use the Save Strategy As command, any changes are saved to the current implementation run, but are not preserved for future use.
Understanding Run Status

The Vivado IDE processes the run, and launches implementation, depending on the status of the run. The status is displayed in the Design Runs window, as shown in Figure 1-6, page 20. If the status of the run is:

- **Not Started**: The run will begin immediately.
- **Error**: The Vivado IDE will first reset the run to remove any incomplete run data, and then will restart the run.
- **Complete** (or **Out-of-Date**): The Vivado IDE will prompt you to confirm that the run should be reset prior to proceeding with the run.

Resetting and Deleting Runs

To reset a run, select a run in the Design Runs window, and select **Reset Runs** from the popup menu. Resetting an implementation run will return it to the first step of implementation, opt_design, for the selected run.

![Reset Runs](image1.png)

*Figure 1-9: Reset Run*

The Vivado IDE will prompt you to confirm the Reset Runs command, and provide the option of deleting generated files from the run directory. The default setting is to delete the generated files. Disable this check box if you to preserve the generated run files for any reason.

You can also delete runs from the Design Runs window by selecting the run, and selecting **Delete** from the popup menu. The Vivado IDE will prompt you to confirm the Delete Runs command, and provide the option of deleting generated files from the run directory. This is on by default.

![Delete Runs](image2.png)

*Figure 1-10: Reset Run*
Customizing Implementation Strategies

Implementation Settings define the default options used when defining new implementation runs. These options can be configured using the Vivado IDE.

Figure 1-11, page 25 shows the Implementation Settings of the Project Settings dialog box. This dialog box can be accessed from the Vivado IDE by using the Tools > Project Settings command from the main menu.

TIP: The Project Settings command is not available in the Vivado IDE when running in Non-Project Mode. In this case, you can define and preserve implementation strategies as Tcl scripts that can be used in batch mode, or interactively in the Vivado IDE.

Implementation Settings for the active implementation run can also be accessed directly from the Flow Navigator.

The following fields are found in Implementation Settings:

- **Default Constraint Set**: Select the constraint set to be used by default for the implementation run.

- **Strategy**: Select the strategy to use for the implementation run. Vivado Design Suite offers a set of pre-defined strategies to select from, and you can also create your own implementation strategies. For more information see Defining Strategies, page 25.

- **Save Strategy As**: This command, which appears as an icon next to the Strategy field, saves any changes to the strategy as a new strategy for future use.

- **Description**: Displays a textual description of the selected implementation strategy.

The description of Vivado tools standard implementation strategies cannot be changed. However, if you change the standard strategy, you will be offered the chance to save the new strategy, and change the description at that time.

The description of user-defined strategies can be changed by entering a new description.
Defining Strategies

A strategy is a defined approach for resolving the synthesis or implementation challenges of the design. The strategy is defined in a pre-configured set of options for the Vivado implementation features. Strategies are tool and version specific. Each major release has version-specific strategies.

Vivado implementation provides several commonly used strategies that are tested against internal benchmarks. Changes to the settings for predefined implementation strategies cannot be saved. However, you can copy and modify supplied strategies to create your own.

The currently defined strategies can be accessed through the Vivado IDE from the Tools > Options command in the main menu. Figure Figure 1-12 shows the default strategies provided with the Vivado tools.
To review, copy, and modify strategies:

1. Select **Tools > Options** from the main menu.

2. Select **Strategies** in the left-side panel. The Strategies dialog box, shown in Figure 1-12, page 26, contains a list of pre-defined strategies for different tools and release versions.

---

Figure 1-12: Default Implementation Strategies
3. In the Flow pulldown select the appropriate Vivado Implementation version for the available strategies. A list of provided strategies displays.

4. To create a new strategy, use the Add Strategy command on the toolbar or from the popup menu.

Alternatively, you can copy an existing strategy by using the Create a copy of this strategy command on the toolbar or from the popup menu. The Vivado design tool creates a copy of the currently selected strategy and adds it to the User Defined Strategies list, and displays the strategy options on the right side of the dialog box for you to modify.

5. Provide a name and description for the new strategy as follows:
   - **Name**: Enter a strategy name to assign to a run.
   - **Type**: Specify Synthesis or Implement.
   - **Tool Version**: Specify the XST or ISE tool version.
   - **Description**: Enter the strategy description which is displayed in the Design Run results table.

6. Edit the Options for the different implementation steps: opt_design, power_opt_design, place_design, phys_opt_design, route_design, and write_bitstream.

When you click on the name of a specific command option, a brief description of the option displays at the bottom of the Design Run Settings dialog box. See Chapter 2, Implementation Commands for a detailed discussion of the different implementation steps, and their available options.

You can modify command options by clicking on the right-side column of a specific option to modify:

- Select predefined options from the pull down menu.
- Enable or disable some options with a check box.
- Type a user-defined value for options with a text entry field.
- Use the file browser to specify a file for options accepting a file name and path.
- Insert a custom Tcl script before (tcl.pre) and after (tcl.post) each implementation step. This allows you to perform specific tasks either before or after
the implementation step, such as generating a report to compare timing before and after optimization.

7. Click **Apply** and **OK** to save the new strategy.

The new strategy is listed under User Defined Strategy. The Vivado tool saves user-defined strategies to the following locations:

- **Linux OS**
  
  
  
  
  $HOME/.Xilinx/Vivado/strategies

- **Windows 7**
  
  
  
  
  C:\Users\<username>\AppData\Roaming\Xilinx\Vivado\strategies

- **Windows XP**
  
  
  
  
  C:\Documents and Settings\username\Application Data\Xilinx\Vivado\strategies

**TIP:** Design teams that want to create and share strategies can copy any user-defined strategy from the user directory to the `<InstallDir>/Vivado/<version>/strategies` directory, where `<InstallDir>` is the installation directory of the Xilinx software, and `<version>` is the release version.

**Launching Runs**

To launch the active implementation run in the Design Runs window, you can:

- Click the **Run Implementation** button on the Flow Navigator.

- Use the **Flow > Run Implementation** command from the main menu.

- Use the **Run Implementation** command from the toolbar menu.

- Select a run in the Design Runs window and use the **Launch Runs** command from the popup menu.

When you launch a single implementation run, the Vivado tools spawn a separate process for the implementation. You can also launch a run other than the active run, or launch multiple implementation runs at the same time by selecting one or more runs in the Design Runs window.

1. Use **Shift+click** or **Ctrl+click** to select multiple runs.

You can choose both synthesis and implementation runs when selecting multiple runs in the Design Runs window. The Vivado IDE will manage run dependencies, and launch runs in the correct order.
2. Use the **Launch Runs** command from the popup menu, or from the Design Runs window toolbar menu, to open the Launch Selected Runs dialog box as shown in Figure 1-13, page 29.

![Launch Selected Runs Dialog Box](image)

**Figure 1-13:** Launch Selected Implementation Runs

- **Launch Directory:** The default launch directory is contained within the local project directory structure. Files for implementation runs are stored at:

  \(<\text{project\_name}>/\text{project\_name}.\text{runs}/\text{run\_name}>\)

**TIP:** Defining any non-default location outside of the project directory structure makes the project non-portable because absolute paths are written into the project files.

- **Specify Options:**
  - **Launch Runs on Local Host:** Launch the run on the local machine.
    - **Number of Jobs:** Define the number of local processors to use when launching multiple runs simultaneously. Individual runs are launched on each processor. Vivado does not support multi-threading for implementation.
  - **Launch Runs on Remote Hosts** (Linux only): Use remote hosts to launch one or more jobs. See Appendix A, Using Remote Hosts.
    - **Configure Hosts:** Select this option to configure remote hosts.
  - **Generate scripts only:** Export and create the run directory and run script, but do not launch the run at this time. The script can be run at a later time outside of the Vivado IDE tool.
Moving Processes into the Background

As the Vivado IDE spawns the process to run synthesis or implementation, it begins by reading design files and constraint files in preparation for the run. The Starting Run dialog box, as shown in Figure 1-14, lets you make this preparation into a background process to free up CPU resources.

When you put this process into the background, it releases the Vivado IDE to perform certain other functions, like viewing reports, or opening design files, while it completes the background task. You can make use of this time to review previous runs for instance, or examine reports. However, the Tcl Console is blocked, and you will not be able to use Tcl commands, or perform tasks that require Tcl commands, such as switching to another open design.

Running Implementation in Steps

Vivado implementation consists of a number of smaller processes such as logic optimizer, placer, and router. The Vivado tools let you run implementation as a series of steps, rather than as a single process.

1. Select a run in the Design Runs window.
2. Use the Launch Next Step: <Step> command from the popup menu. Valid <Step> values are:
   - **Opt Design**: Optimize the logical design and fit it onto the target Xilinx FPGA.
   - **Power Opt Design**: Optimize elements of the design to reduce power demands of the implemented FPGA.
   - **Place Design**: Place the design onto the target Xilinx device.
   - **Phys Opt Design**: Optimizes the timing of the design by replicating drivers of high-fanout nets to distribute the loads.
   - **Route Design**: Route the design onto the target Xilinx device.
   - **Write Bitstream**: Generates a bitstream for Xilinx device configuration. Although not technically part of an implementation run, the BitGen step is available as an incremental step.
3. Repeat **Launch Next Step: <Step>** as needed to move the design through implementation.

You can run any reports or analysis needed between implementation steps to explore design options.

4. To back up from a completed step, you can use the **Reset to Previous Step: <Step>** command from the Design Runs window popup menu.

The **Reset to Previous Step** command resets the selected run from its current state to the prior incremental step. This allows you to step backward through a run, to make any needed changes, then step forward again to incrementally complete the run.

### Monitoring the Implementation Run

You can monitor the status of a Synthesis or Implementation run in the Compilation window, reading the compilation information, reviewing warnings and errors in the Messages window, viewing the Project Summary, or opening the Design Runs window.

### Using the Run Status Display

The status of a run that is in-progress can be displayed in two ways for synthesis and implementation runs. These status displays, as shown in **Figure 1-15**, indicate that a run is in progress, and provide an opportunity to cancel the run if needed.

1. The Run Status indicator in the project status bar at the upper right corner of the Vivado IDE displays a scrolling bar to indicate the run is in process. You can use the **Cancel** button to end the run.

2. The Run Status indicator in the Design Runs window, as shown at the bottom of **Figure 1-15, page 31**, displays a circular arrow to indicate the run is in process. You can select the run and use the **Reset Run** command from the popup menu to cancel the run.

**Figure 1-15:** Implementation Run Status

1. The Run Status indicator in the project status bar at the upper right corner of the Vivado IDE displays a scrolling bar to indicate the run is in process. You can use the **Cancel** button to end the run.

2. The Run Status indicator in the Design Runs window, as shown at the bottom of **Figure 1-15, page 31**, displays a circular arrow to indicate the run is in process. You can select the run and use the **Reset Run** command from the popup menu to cancel the run.
Cancelling/Resetting the Run

If you cancel a run that is in-progress, either through the **Cancel** button, or through the **Reset Run** command, the Vivado IDE will prompt you to delete any run files created during the cancelled run.

![Cancel Implementation](image)

*Figure 1-16: Cancel Run*

Selecting **Delete Generated Files** will clear the run data from the local project directories. It is a good idea to delete any data created as a result of a cancelled run to avoid any problems with future runs.

Viewing the Log

The Log window opens in the Vivado IDE after you launch a run, and shows the standard output messages. *Figure 1-17* shows an example of the Log window.

The Log window provides details about the progress of each individual implementation process, such as place_design or route_design. The Log window can also help you understand where different messages originate, to aid in debugging the implementation run.

![Compilation Log](image)

*Figure 1-17: Compilation Log*

The **Pause** button lets you pause the output to the Log window so that you can scroll and read the log while Vivado implementation continues running.
Determining the Project Status

The Vivado IDE provides several visual indicators about the overall status of a project as well as methods to take in the next step of the process. Only the results of the major design tasks in the design process are reported in the project status.

The overall project status displays in the Project Summary and in the Status Bar so you can quickly visualize the status of a project upon opening the project, or while you are running the design flow commands. These include RTL Elaboration, Synthesis, Implementation, and Bitstream Generation.

Project Status Bar

The overall project status displays in the project status bar in the upper-right corner of the Vivado IDE.

As you run the Elaborate, Synthesize, Implement, and Write Bitstream commands, the Project Status Bar changes to indicate either a successful or failed attempt. Failures are displayed with red text.

If source files or design constraints change, and either synthesis or implementation was previously completed, the project can be marked Out-of-Date as shown in Figure 1-18. The project status bar indicates an Out-of-Date status. Click the more info link to display what aspects of the design are out of date. In this case it may be necessary to rerun implementation, or both synthesis and implementation.

Figure 1-18 also displays the Force-up-to-date link that you can use to force the implementation or synthesis runs up to date. You can use this feature if you have made a change to the design or constraints, but still want to analyze the results of the current run. The Force-up-to-date command is also available from the popup menu of the Design Runs window when an out-of-date run is selected. See the Vivado Design Suite User Guide: Using the Integrated Design Environment (UG893) for more information.
Moving Forward After Implementation

Moving Forward After Implementation

After implementation has completed, for both Project Mode and Non-Project Mode, the direction you will take the design next depends on the results of the implementation. Is the design fully placed and routed, or are there issues that need to be resolved? Have the timing constraints and design requirements been met, or are there additional changes required to complete the design? Are you ready to generate the bitstream for the Xilinx part?

In the case of a non-project based design, the Vivado Design Suite has generated messages for the design session, and written them to the Vivado log file (vivado.log). You will need to examine this log file, as well as generate reports from the design data, to get an accurate assessment of the current project state.

For project-based designs, the Vivado IDE reflects the messages from the log file in the Messages window, and automates the creation and delivery of many of the reports you will need to examine. In Project Mode, after an implementation run is complete in the Vivado IDE, a dialog box opens that prompts you for the next step, as shown in Figure 1-19.

![Figure 1-19: Project Mode - Implementation Completed](image)

In the Implementation Completed dialog box, select the appropriate option, then click OK:

- **Open Implemented Design** — This imports the netlist, design constraints, the target part, and the results from place and route into the Vivado IDE for design analysis and further work as needed.

- **Generate Bitstream** — Launches the Generate Bitstream dialog box. See *Vivado Design Suite User Guide: Programming and Debugging* (UG908) for more information.

- **View Reports** — Opens the Reports window for you to select and view the various reports produced by the Vivado tools during implementation. See *Viewing Implementation Reports*, page 36, for more information.
The recommended steps to follow after implementation are:

1. Viewing implementation messages.

2. Viewing implementation reports to validate key aspects of the design:
   - Timing constraints are met (report_timing_summary).
   - Utilization is as expected (report_utilization).
   - Power is as expected (report_power).

3. Writing the bitstream file. This will include a final DRC to validate the design does not violate any hardware rules.

If any design requirements have not been met, in Project Mode, you can open the implemented design for further analysis. In the Non-Project Mode, you can open a post-implementation design checkpoint. For more information on analysis of the implemented design, refer to the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques (UG906)*.

**Viewing Messages**

**IMPORTANT:** Review all messages. The messages may suggest ways to improve your design for performance, power, area, and routing. Critical warnings may also expose timing constraint problems that should be resolved.

In Non-Project Mode, you will need to review the messages in the Vivado tools log file (vivado.log). This is the file where Vivado stores all of the commands, and the results and messages from those commands, that you have used during a single design session. Open the log file, in the Vivado text editor for example, and look through the results of all commands for valuable insight.

For Project Mode, the Messages window in the Vivado IDE provides a filtered list of the Vivado log that includes only the main messages, warnings, and errors. The Messages window sorts by feature, and provides toolbar options to filter and display only specific types of messages. Figure 1-20 shows an example Messages window.
Click the expand and collapse tree widgets to view the individual messages. You can display Errors, Critical Warnings, Warnings, and Informational Messages in the Messages window by selecting the appropriate check box in the banner as shown at the top of Figure 1-20.

Selecting any of the messages in the Messages window that have a link, automatically opens the source file and highlights the appropriate line in the file. You can also use the **Search for Answer Record** command from the Messages window popup menu to search the Xilinx Customer Support database for answer records related to a specific message.

### Viewing Implementation Reports

The Vivado tool also generates many different types of reports from the design data. The Vivado Design Suite can report on timing, timing configuration, and timing summary, clocks, clock networks, and clock utilization, power, switching activity, and noise analysis. These are just some of the many reports available from within the Vivado tools.

When viewing reports, you can:

- Browse the report file using the scroll bar.
- Select the **Find** or **Find in Files** buttons to search for specific text.
- Use the **Go to the beginning** or **Go to the End** toolbar buttons to scroll to the beginning or end of the file.
Reporting in Non-Project Mode

In the case of Non-Project mode, you will need to run these reports manually, using Tcl commands, or a Tcl script to create a series of reports. For example, the following script runs a series of reports and saves them to a Reports folder:

```tcl
# Report the 15 longest carry chains
report_carry_chains -file C:/Reports/carry_chains.rpt -max_chains 15

# Run Timing Summary Report for post implementation timing
report_timing_summary -file C:/Reports/post_route_timing.rpt -name time1

# Run Utilization Report for device resource utilization
report_utilization -file C:/Reports/post_route_utilization.rpt
```

You can also have these reports opened in a window within the Vivado IDE. In the preceding example script, the `report_timing_summary` command uses both the `-file` option and the `-name` option to direct the output of the report to a file and to a window within the IDE. Figure 1-22, page 38 shows an example of a report opened in the Vivado IDE.

**TIP:** The directory the reports will be written to must exist prior to running the report, or the file cannot be saved.

Use the Tcl `help` command in the Vivado IDE or at the Tcl command prompt, or refer to the *Vivado Design Suite Tcl Command Reference Guide (UG835)* for a complete description of the Tcl reporting commands and their various options.

Reporting in Project Mode

In Project Mode, many of the reports are generated automatically as you work through the design flow. In the Vivado IDE you can view report files generated by the Vivado tools from within the Reports window, as shown in Figure 1-21, page 38. The window is usually opened automatically after synthesis or implementation commands are run. If the window is not open, select the Reports link in the Project Summary, or use the Windows > Reports command.

**TIP:** The `tcl.pre` and `tcl.post` options of an implementation run let you output custom reports at each step in the process. These reports will not be listed in the Reports window, but can be tailored to meet your specific needs. See *Changing Implementation Run Settings, page 20* for more information.
The reports available from the Reports window are text-based reports, containing information related to the run. You can select and view any of the listed reports, and the selected report opens in text form in the Vivado IDE, as shown in Figure 1-22.

Figure 1-21: Reports window

Figure 1-22: Control Sets Report
Cross Probing from Reports

For both Project Mode and Non-Project Mode, the Vivado IDE supports cross-probing between reports and the associated design data in different windows of the IDE, such as in the Device window for instance. However, text reports do not support cross-probing. You generate the report in the Vivado IDE, using a menu command or Tcl command.

TIP: The Vivado IDE supports cross-probing between reports and other design windows in both Project Mode and Non-Project Mode.

As an example, the Reports window includes a text-based Timing Summary Report under Route Design as shown in Figure 1-21. However, when analyzing timing it is very helpful to see the design data associated with critical paths, including placement and routing resources in the Device window. You can regenerate the report in the Vivado IDE, using the Tools > Timing > Report Timing Summary command, and the resulting report will allow you to cross-probe between the different views of the design.

Figure 1-23, page 39 shows an example of cross-probing between the Timing Summary report and the Device window. In this Non-Project Mode example, a post-route design checkpoint was opened in the Vivado IDE, and the Timing Summary report was generated and opened in the IDE (using the report_timing_summary -name command). The Routing Resources were enabled in the Device window. When the timing path was selected in the Timing Summary report, the path was automatically cross-probed in the Device window as shown.

For more information on analyzing reports and strategies for design closure, see the Vivado Design Suite User Guide: Design Analysis and Closure Techniques (UG906).
Chapter 2

Implementation Commands

Introduction

The Vivado IDE provides many features to manage and simplify the implementation process for project-based designs, including the ability to step manually through the implementation process as described in Running Implementation in Project Mode, page 15. However, Non-Project based designs must be manually taken through each step of the implementation process using Tcl commands, or Tcl scripts, as described in Running Implementation in Non-Project Mode, page 12.

Moving a design through the Vivado implementation process, whether in Project Mode or Non-Project Mode, consists of several sub-processes:

- **Open Synthesized Design**: Combine the netlist, the design constraints, and Xilinx target part data, to build the in-memory design to drive implementation.
- **Opt Design**: Optimize the logical design and fit it onto the target Xilinx FPGA.
- **Power Opt Design**: Optimize elements of the design to reduce power demands of the implemented FPGA. This is an optional step in the implementation process.
- **Place Design**: Place the design onto the target Xilinx device.
- **Phys Opt Design**: Optimizes the timing of the design by replicating drivers of high-fanout nets to distribute the loads. This is an optional step in the implementation process.
- **Route Design**: Route the design onto the target Xilinx device.
- **Write Bitstream**: Generates a bitstream for Xilinx device configuration. Although not technically part of an implementation run, Write Bitstream is available as a separate step.

To provide a better understanding of the individual steps in the implementation process, the details of each step, and the associated Tcl commands are documented here. Refer to the Vivado Design Suite Tcl Command Reference Guide (UG835) for a complete description of the Tcl commands and their various options.
Opening the Synthesized Design

The first step in implementation is reading the netlist from the synthesized design into memory and applying design constraints. You can open the synthesized design in different ways, depending on the flow used. The following Tcl commands can be used, in Project Mode or Non-Project Mode, to read the synthesized design into memory, depending on the source files in the design, and the state of the design:

- `synth_design`
- `read_checkpoint`
- `open_run`
- `link_design`

To create the in-memory design, the Vivado Design Suite uses the following process to combine the netlist files, constraints files, and the target part information:

1. Assembles the netlist, from multiple sources if needed. Designs can consist of a mix of structural Verilog, EDIF, and NGC.

2. Transforms legacy netlist primitives to the currently supported subset of Unisim primitives.

   **TIP:** You can use `report_transformed_primitives` to generate a list of transformed cells.

3. Builds shapes. The Vivado tools create implicit shapes of cells based on their connectivity or placement constraints to simplify placement. A relatively placed macro (RPM) is an example of an implicit shape. RPMs are placed as a group rather than as individual cells. Another example of an implicit shape is a long carry chain that needs to be placed in multiple slices. The CARRY4 elements making up the carry chains must belong to a single shape to ensure downstream placement aligns it into vertical slices.

The `synth_design` command runs Vivado synthesis on RTL sources, with the specified options, and reads the design into memory after synthesis. The `synth_design` command can be used in both Project Mode, and Non-Project Mode.

```
synth_design [-name <arg>] [-part <arg>] [-constrset <arg>] [-top <arg>]
             [-flatten_hierarchy <arg>] [-gated_clock_conversion <arg>] [-effort_level <arg>]
             [-quiet] [-verbose]
```

**Sample Script:**
The following is an excerpt of the create_bft_batch.tcl script found in the examples/Vivado_Tutorials directory of the software installation. See the Vivado Design Suite Tutorial: Design Flows Overview (UG888) details on using the script.

# 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.xdc

# Run synthesis, report utilization and timing estimates, write design checkpoint
synth_design -top bft -part xc7k70tfbg484-2 -flatten rebuilt
write_checkpoint -force $outputDir/post_synth

The sample script reads VHDL and Verilog files, and reads a constraints file, and then synthesizes the design on the specified part. The design is opened by the Vivado tool into memory when the synth_design command completes. Finally, a design checkpoint is written after completing synthesis.

Refer to the Vivado Design Suite Tcl Command Reference Guide (UG835) for a complete description of the Tcl commands and their various options.

read_checkpoint

Open a post-synthesis design checkpoint for use in a non-project based design.

read_checkpoint [-part <arg>] [-quiet] [-verbose] <file>

Sample Script:

# Read the specified design checkpoint and create an in-memory design.
read_checkpoint C:/Data/post_synth.dcp

The sample script opens the post synthesis design checkpoint file.

open_run

Opens a previously completed synthesis or implementation run and loads the in-memory design of the Vivado tool. This command only works in Project Mode, since design runs are not supported in Non-Project Mode.

Use this command before implementation, on an RTL design, to open a previously completed Vivado synthesis or XST run, and load the synthesized netlist into memory. You do not need to do this after synth_design, since the in-memory design is updated automatically. You only need to use this command to open a previously completed synthesis run, from an earlier design session.

Note: This command is for use with RTL designs. To open a netlist-based design, use the link_design command.

open_run [-name <arg>] [-quiet] [-verbose] <run>
Sample Script:

```bash
# Open named design from completed synthesis run
open_run -name synth_1 synth_1
```

The sample script opens a design, called synth_1, into the Vivado tool memory from the completed synthesis run, also called synth_1.

**link_design**

Creates an in-memory design from netlist sources, such as from a third-party synthesis tool, linking the netlists and design constraints with the target part. This command supports both Project Mode and Non-Project mode to create the netlist design in the Vivado tool.

```bash
link_design [-name <arg>] [-part <arg>] [-constrset <arg>] [-top <arg>]
[-quiet] [-verbose]
```

Sample Script:

```bash
# Open named design from netlist sources.
link_design -name netDriven -constrset constrs_1 -part xc7k325tfbg900-1
```

*Note:* If you use the `open_run` or `link_design` commands while a design is already in memory, the Vivado tool prompts you to save any changes to the current design before opening the new design.

After creating the in-memory synthesized design in the Vivado tool, you should review Errors and Critical Warnings for missing or incorrect constraints. After the design is successfully created, you can begin running analysis, generating reports, applying new constraints, or running implementation.

**RECOMMENDED:** Immediately after opening the in-memory, synthesized design, is an excellent time to check timing constraints by running the `report_timing_summary` command to ensure the design goals are complete and reasonable.
Logic Optimization

Logic Optimization is run to ensure the most efficient logic design prior to attempting placement. Logic optimization performs a netlist connectivity check to warn of potential design problems such as nets with multiple drivers, and un-driven inputs.

The Vivado tools perform the following logic optimizations on the in-memory design by default:

1. **Retargeting**: Replaces one cell type with another to ease optimization. For example, a MUXF7 is replaced by a LUT3 so that it can be combined with other LUTs. In addition, simple cells like inverters are absorbed into downstream logic.

2. **Constant propagation**: Propagates constant values through logic, which results in:
   - Eliminated logic - for example, an AND with a constant 0 input.
   - Reduced logic - for example, a 3-input OR with a constant 1 input is reduced to a 2-input OR.
   - Redundant logic - for example, a 2-input OR with a logic 0 input is reduced to a wire.

3. **Remap**: Combines multiple LUTs into a single LUT to reduce the depth of the logic.

4. **Sweep**: Removes cells that have no loads.

5. **Resynth area**: Perform re-synthesis in area mode to reduce the number of LUTs.

**opt_design**

**IMPORTANT**: Each use of logic optimization affects the in-memory design, not the synthesized design that was originally opened.

Optimizes the current netlist and performs all optimizations by default. When `opt_design` runs, it reads the in-memory design, optimizes it, and outputs the optimized design back into memory.

```
opt_design [-sweep] [-retarget] [-propconst] [-remap] [-resynth <arg>]
    [-mode <arg>] [-effort_level <arg>] [-quiet] [-verbose]
```

**Sample Script:**

```
opt_design -retarget -propconst -sweep
```

You can use command line options to restrict optimization to one or more of the listed types. For example, a congested design may benefit from skipping the remap optimization.
RECOMMENDED: To better analyze optimization results, use the -verbose switch to see additional details of the logic affected by opt_design optimization. The -verbose switch is off by default due to the potential for a large volume of additional messages.

You should use the -verbose option if you suspect it will be helpful, because running the opt_design command again with -verbose changes the optimization results:

```
opt_design
opt_design -verbose
```

Logic Optimization Constraints

The Vivado Design Suite respects the DONT_TOUCH and MARK_DEBUG properties during logic optimization, and does not optimize away nets with these properties. See the Vivado Design Suite User Guide: Synthesis (UG901) for more information.

- **MARK_DEBUG** is placed on nets that are candidates for probing with the ChipScope™ Pro tool. A net with MARK_DEBUG is connected to a slice boundary to ensure it can be probed.
- The DONT_TOUCH property is typically placed on leaf cells to prevent them from being optimized. DONT_TOUCH on a hierarchical cell preserves the cell boundary, but optimization may still occur within the cell.

IMPORTANT: KEEP and KEEP_HIERARCHY properties in designs migrated from the ISE Design Suite are automatically converted to DONT_TOUCH.

Power Optimization

Optionally optimize dynamic power using clock gating. The Power Optimization feature can be used in both Project Mode and Non-Project Mode. For more information on refer to the Vivado Design Suite User Guide: Power Analysis and Optimization (UG907).

The Vivado power optimizer can be run after logic optimization to reduce power demand in the design. Power optimization includes Xilinx intelligent clock gating solutions that can reduce dynamic power in FPGA designs, but do not change the clocks or logic of the design.

The Vivado tool performs an analysis on all portions of the design, including legacy and third-party IP blocks, identifying output logic from sourcing registers that does not contribute to the result for each clock cycle. Vivado power optimizer takes advantage of the abundant supply of Clock Enables (CEs) available in the logic of 7 series FPGAs. The tool creates fine-grain clock gating, or logic gating signals, that eliminate unnecessary switching activity in the register. In addition, at the flip-flop level, CEs are actually gating the clock
rather than selecting between the D input and feedback Q output of the flip-flop. This increases the performance of the CE input but also reduces clock power.

![Image of Intelligent Clock Gating](image1)

**Figure 2-1:** Intelligent Clock Gating

Intelligent clock gating also reduces power for dedicated block RAMs in either simple dual-port or true dual-port mode, as illustrated in Figure 2-2, page 46. These blocks provide several enables: an array enable, a write enable, and an output register clock enable. Most of the power savings comes from using the array enable, and the Vivado power optimizer implements functionality to reduce power when no data is being written and when the output is not being used.

![Image of Leveraging BRAM Enables](image2)

**Figure 2-2:** Leveraging BRAM Enables

`power_opt_design`

Prior to running power optimization, you can configure the command with the `set_power_opt` command. This lets you specify the appropriate cell types or hierarchy to include or exclude in the optimization. The syntax for this command is:

```
set_power_opt [-include_cells <args>] [-exclude_cells <args>] [-clocks <args>]
[-cell_types <args>] [-quiet] [-verbose]
```

The `power_opt_design` command analyzes the design and performs intelligent clock gating to optimize power, using the following Syntax:

```
power_opt_design [-quiet] [-verbose]
```

**Sample Script:**

```
power_opt_design
```

The Vivado power optimizer analyzes and optimizes the entire design as a default, unless the `set_power_opt` command changes its scope.
Placement

After logic optimization, and optionally power optimization, the next step in implementation is placement. The Vivado placer engine positions cells from the netlist onto specific sites in the target Xilinx part. Like logic optimization, the Vivado placer works from, and updates, the in-memory design.

The Vivado placer engine simultaneously optimizes the design placement for:

- **Timing slack**: Placement of cells in timing-critical paths is chosen to minimize negative slack.
- **Wirelength**: Overall placement is driven to minimize the overall wirelength of connections.
- **Congestion**: The Vivado placer monitors pin density and spreads cells to reduce potential routing congestion.

Prior to starting placement, Vivado implementation runs design rule checks (DRC), including user-selected DRCs from `report_drc`, and built-in DRCs internal to the Vivado placer engine. Internal DRCs report:

- Memory Interface Generator (MIG) cells without LOC constraints.
- I/O banks with conflicting IOSTANDARDS.

After DRC, the Vivado placer places clock and I/O cells before placing other logic cells. Clock and I/O cells are placed concurrently because they are often related through complex placement rules specific to the selected Xilinx device family or part. At this time the placer targets:

- I/O ports and logic.
- Global and local clock buffers.
- Clock management tiles (MMCMs and PLLs).
- Gigabit Transceivers (GT) cells.

The placer adheres to physical constraints, such as LOC and AREA_GROUP/PBLOCK properties, when placing unfixed logic during this stage of placement. The Vivado placer validates existing LOC constraints against the netlist connectivity and device sites. Certain IP, such as MIGs and GTs, are generated with device-specific placement constraints.

Due to the device I/O architecture, a LOC property often constrains cells other than the LOCed cell. A LOC on an input port also fixes the location of its related I/O buffer, IDELAY, and ILOGIC. Conflicting LOC constraints cannot be applied to individual cells in the input path. The same applies for outputs and GT-related cells.
Clock resources must follow placement rules as described in the 7 Series FPGAs Clocking Resources (UG472). For example, an input that drives a global clock must be located at a clock-capable I/O site and must be located in the same upper or lower half of the device. These clock placement rules are also validated against the logical netlist connectivity and device sites.

If the Vivado placer fails to find a solution for the clock and I/O placement, it reports the placement rules that were violated, along with brief descriptions and affected cells. In some cases, the placer provisionally places cells at sites and then attempts to place other cells as it attempts to solve the placement problem. The provisional placements often pinpoint the source of clock and I/O placement failure. Manually placing a cell that failed provisional placement may help placement converge.

**TIP:** Use the `place_ports` command to run the clock and I/O placement step first, and then run `place_design`. If port placement fails, the placement is saved to memory to allow failure analysis.

After Clock and I/O placement, the remaining placement phases consist of global placement, detailed placement, and packing and legalization. After placement, an estimated timing summary is output to the log file:

```
Phase 12 Placer Reporting
INFO: [Place-100] Post Placement Timing Summary | WNS=-0.08836 | TNS=-1.479 |
```

Where: WNS = Worst Negative Slack, TNS = Total Negative Slack.

**RECOMMENDED:** Run `report_timing` after placement to check the critical paths. Paths with very large negative slack may need manual placement, further constraining, or logic restructuring to achieve timing closure.

**place_design**

Automatically place ports and cells.

```
place_design [-effort_level <arg>] [-no_timing_driven] [-quiet] [-verbose]
```

**Sample Script:**

```
# Run logic optimization and placement, report utilization and timing estimates,
opt_design
place_design
write_checkpoint -force $outputDir/post_place
report_timing_summary -file $outputDir/post_place_timing_summary.rpt
```

The sample script performs logic optimization on the in-memory design, rewriting it in the process, then places the design, also rewriting the in-memory design. A design checkpoint is written after completing placement, then a timing summary report is generated and written to the specified file.
Physical Synthesis

Physical Synthesis optionally performs timing-based logic replication of high-fanout drivers and critical-path cells. Drivers are replicated, and loads are redistributed among the replicated drivers, and then the replicated drivers are automatically placed. The optimization process is as follows:

1. High fanout nets, with negative slack within 10% of the WNS, are considered for replication.
2. Loads are clustered based on proximity, and drivers are replicated and placed for each load cluster.
3. Timing is re-analyzed, and logical changes are committed if timing is improved.
4. After replication, the design is checked again for high fanout nets to replicate. If high fanout nets still exist, the replication process continues until there are no high fanout nets to optimize.

Physical Synthesis reports each replicated net, the number of times its driver was replicated, and the worst negative slack (WNS) before and after optimization. Replicated objects are named by appending _replica to the original object name, followed by the replicated object count.

Physical Synthesis also replicates cells in failing paths. If the loads on a specific cell are placed far apart, the cell may be replicated with new drivers placed closer to load clusters. High fanout is not a requirement for this optimization to occur, but the path must fail timing with slack within 10% of the worst negative slack.

phys_opt_design

Optionally performs physical optimizations such as timing-driven replication of high fanout nets to improve timing results. Running phys_opt_design on a routed design is not supported.

phys_opt_design [-quiet] [-verbose]

Sample Script:

phys_opt_design

IMPORTANT: The phys_opt_design command operates on the in-memory design. If run twice, the second run optimizes the results of the first run.
Routing

The Vivado router engine performs routing on the placed design, and performs optimization on the routed design to resolve hold time violations. By default, the Vivado router is timing driven, though this can be disabled.

The router can be run in two modes:

- **Normal mode**: The router starts with a placed design and attempts to route all nets. The router can start with placed design that is unrouted, partially routed, or fully routed. The `route_design` command is incremental in nature, meaning that for a partially routed design, the Vivado router will use the existing routes as the starting point instead of starting from scratch. This is the commonly-used default mode.

- **Re-Entrant mode**: Re-entrant routing is a specific mode to use when you intend to run multiple routing passes for a sequence of operations. Without re-entrant mode, the router exits and clears its memory after each routing operation, requiring the router to be initialized each time it is run. This can be time consuming for multiple routing steps. In re-entrant mode the router keeps its data structures in memory in anticipation of subsequent routing operations, which saves considerable time. Subsequent commands like unrouting and routing individual nets can be performed immediately.

Prior to starting routing, Vivado implementation runs design rule checks (DRC), including user-selected DRCs from `report_drc`, and built-in DRCs internal to the Vivado router engine.

The Vivado Design Suite routes global resources first, such as clocks, resets, I/O, and other dedicated resources. This default priority is built into the router. The router then prioritizes data signals according to timing criticality. Keep in mind when using either pre-route flows, or fixed routing constraints, that often the router cannot optimally route some signals because poor timing constraints are giving the router an incorrect timing picture. Common examples include cross-clock paths and multi-cycle paths where hold timing causes route delay insertion. Other examples include congested areas, which can be addressed by targeted fanout optimization in RTL synthesis.

If you observe nets that are routed sub-optimally, the problem is often incorrect timing constraints. Before you experiment with router settings, make sure that you have validated the constraints and the timing picture seen by the router. Validate timing and constraints by reviewing timing reports from the placed design, prior to routing. You will get better results by cleaning up constraints, or by considering RTL changes, rather than simply throwing more routing resources at the problem.

```
routing
```

Route the nets in the current design to complete logic connections on the target part. The syntax for the Vivado router is as follows:
route_design [-unroute] [-re_entrant <arg>] [-nets <args>] [-physical_nets]
[-pin <args>] [-effort_level <arg>] [-no_timing_driven] [-preserve] [-delay]
[-free_resource_mode] -max_delay <arg> -min_delay <arg> [-quiet] [-verbose]

Normal Routing Sample Script

route_design -no_timing_driven -effort_level low

Normal routing is performed as part of an implementation run or by running the
route_design command after place_design as part of a Tcl script. After routing is
complete, routing statistics summarize the routing resources used by resource type, and
provide a timing summary:

[Route-20] Post Routing Timing Summary | WNS=0.0585 | TNS=0 | WHS=0 | THS=0 |

Where:

- WNS = Worst Negative Slack.
- TNS = Total Negative Slack.
- WHS = Worst Hold Slack.
- THS = Total Hold Slack.

Re-Entrant Routing Sample Script #1

# route a few critical nets
% route_design -delay -nets [get_nets myPreRoutes*]
# Complete full route
% route_design

Re-entrant mode is implicitly entered when using a re-entrant routing option, such as
-nets or -pin, and is usually run interactively to address specific routing issues such as:

- Pre-routing critical nets and locking down resources before a full route.
- Manually unrouting non-critical nets to free up routing resources for more critical nets.

The first re-entrant route command initializes the router and routes essential nets, such
as clocks, so that the router can perform timing analysis, timing-driven routing, and
hold-fixing. After re-entrant mode is enabled, the results of iterative routing and
unroutinig reside in memory. Re-entrant mode is exited by issuing the full
route_design command, or by directly disabling re-entrant routing:

route_design -re_entrant off
Re-Entrant Routing Sample Script #2

```tcl
% set preRoutes [list]
% foreach net [get_nets -hier] {
    set weight [get_property weight $net]
    if {[weight > 5]} { ; # get nets with weight above 5
        lappend preRoutes $net
    } }
% route_design -nets [get_nets $preRoutes]

# Unroute all the nets in u0/u1, and route the critical nets first
% route_design -unroute [get_nets u0/u1/*]
% route_design -delay -nets [get_nets $myCritNets]
% route_design
```

The strategy in this sample is to create a list of nets, preRoutes, based on the weight property on the net, and then route the more heavily weighted nets first. This requires critical nets to be assigned weight properties, probably before placement.

**Note:** Refer to the *Vivado Design Suite User Guide: Using Constraints (UG903)* for more information on using the weight constraint.

The script continues after routing the weighted nets. After `route_design` completes, the Vivado router unroutes all of the nets in instance `u0/u1`, then re-routes identified critical nets first, `myCritNets`. Then the general router finishes any remaining unrouted nets.

The following commands can be used during routing for design analysis:

- **report_route_status**: Reports route status for nets.
- **report_timing**: Performs path endpoint analysis.

Refer to the *Vivado Design Suite Tcl Command Reference Guide (UG835)* for a complete description of the Tcl commands and their various options.
Using Remote Hosts

Launching Runs on Remote Linux Hosts

The Vivado™ IDE supports parallel execution of synthesis and implementation runs on multiple Linux hosts simultaneously. This is accomplished in the application with simplified versions of more robust load-sharing software, such as Oracle® Grid Engine and IBM® Platform™ LSF. Linux is the only operating system supporting remote hosts because of security handling and the lack of remote-shell capabilities on Microsoft Windows systems.

Job submission algorithms are implemented using a “greedy,” round-robin style with Tcl pipes within Secure Shell (SSH), a service provided by the Linux operating system. Prior to launching runs on multiple Linux hosts in the Vivado IDE, you should configure SSH so that the host does not require a password each time you launch a remote run. For instructions on configuring SSH, refer to Setting Up SSH Key Agent Forward, page 56.

The requirements for launching synthesis and implementation runs on remote Linux hosts are:

- Vivado tool installation is assumed to be available from the login shell, which means that $XILINX_VIVADO and $PATH are configured correctly in your .cshrc/.bashrc setup scripts. If you can log into a remote machine and enter vivado -help without sourcing any other scripts, this flow should work. If you do not have Vivado set up upon login (CSHRC or BASHRC), you can use the Run pre-launch script option, as describe below, to define an environment setup script to be run prior to all jobs.

- Vivado IDE installation must be visible from the mounted file systems on remote machines. If the Vivado IDE installation is stored on a local disk on your own machine, it may not be visible from remote machines.

- Vivado IDE project files (.xpr) and directories (.data and .runs) must be visible from the mounted file systems on remote machines. If the design data is saved to a local disk, it may not be visible from remote machines.

Configuring Remote Hosts

To configure the Vivado IDE to run synthesis or implementation on a remote Linux host, use the following procedure.
1. Select one of the following commands:
   - **Tools > Options > Remote Hosts**
   - **Synthesis > Launch Runs > Configure Hosts**
   - **Implementation > Launch Runs > Configure Hosts**
   - **Configure Hosts** in the Launch Selected Runs dialog box, as shown in Figure A-1.

![Figure A-1: Configure Hosts from Launch Runs](image)

The Vivado Options dialog box displays with the Remote Hosts section selected. The list of currently defined remote Linux hosts is displayed as shown in Figure A-1, page 55.

2. Click **Add** to enter the names of additional remote servers.

3. Specify the number of processors the remote machine has available to run simultaneous processes using the **Jobs** field next to the host name. Individual runs require a separate processor. Vivado IDE does not support multithreading of processors.

4. Toggle the **Enabled** checkbox to specify whether the server is available for use. You can use this field when launching runs to specify which servers to use for selected runs.

5. Optionally, modify the **Launch jobs with:** field to change the remote access command used when launching runs. The default command is: `ssh -q -o -BatchMode=yes`

***IMPORTANT:** Be careful when modifying this field. For example, removing ‘BatchMode =yes’ could cause the remote process to hang because the Secure Shell incorrectly prompts for an interactive password.

6. Optionally, enable the **Run pre-launch script:** checkbox and define a shell script to run prior to launching the runs. Use this option to run script to setup the host environment if you do not have Vivado IDE set up upon login.
7. Optionally, enable the **Run post-completion script**: checkbox and define a custom script to run after the run completes, to move or copy the results for instance.

8. Optionally, enable **Send email to**: checkbox, and enter an Email address to send a notification when the runs complete. You can choose to have notifications sent **After each Job**, or **After all jobs**.

9. Click **OK** to accept the Remote Host configuration settings.

![Remote Host Configuration](image)

*Figure A-1: Configuring Remote Hosts*

You can also test the connection to the remote host by selecting one or more hosts, and clicking **Test** to verify that the server is available and the configuration is properly set.

**RECOMMENDED:** Test each host to ensure proper set up prior to submitting runs to the host.

Click **Remove** to delete selected remote hosts.
Setting Up SSH Key Agent Forward

SSH configuration is accomplished with the following commands at a Linux terminal or shell:

**Note:** This is a one-time step, and when successfully set-up, does not need to be repeated.

1. Run the following command at a Linux terminal or shell to generate a public key on your primary machine. Though not required, it is a good practice to enter (and remember) a private key phrase when prompted for maximum security.

   ```bash
   ssh -keygen -t rsa
   ```

2. Append the contents of your publish key to an `authorized_keys` file on the remote machine. Change `remote_server` to a valid host name:

   ```bash
   cat ~/.ssh/id_rsa.pub | ssh remote_server "cat - &gt; ~/.ssh/authorized_keys"
   ```

3. Run the following command to prompt for your private key pass phrase, and enable key forwarding:

   ```bash
   ssh -add
   ```

You should now be able to `ssh` to any machine without typing a password. The first time you access a new machine, it prompts you for a password; upon subsequent access it does not prompt. If you are always prompted for a password, contact your System Administrator for further assistance.
Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:


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.

References

Vivado Design Suite 2012.2 Documentation
(www.xilinx.com/support/documentation/dt_vivado_vivado2012-2.htm)