XAPP1165 (v1.0) August 5, 2013

Using Vivado Design Suite with Version Control Systems

Author: Jim Wu

Summary

This application note provides recommendations for using version control systems with the Vivado® Design Suite in both Non-Project Mode and Project Mode. The recommended methodology is to use the Vivado Design Suite Manage IP feature for IP management, keep design sources as remote files, and use Tool Command Language (Tcl) scripts to manage design sources, design runs, and project creation. This application note is accompanied by a complete example design that follows this methodology and runs from register transfer level (RTL) to bitstream. It is also accompanied by Tcl scripts for creating an initial design repository using Git in both Non-Project Mode and Project Mode.

Introduction

The Vivado Design Suite is a revolutionary intellectual property (IP)- and system-centric design environment, which simplifies design integration and implementation by bringing various design tools into a common design environment. This type of design environment poses a challenge to version control systems designed to facilitate project management and collaboration in a team design environment. For example, a complex FPGA project can involve multiple design sources and configuration files. Only a subset of these files require version control to recreate a project and reproduce implementation results. Following is an example of the types of files used in the Vivado Design Suite:

- RTL and netlist files, including Verilog, SystemVerilog, VHDL, and EDIF
- System Generator models (.mdl, .slx)
- IP-XACT core files (.xci, .mem, .coe)
- Vivado IP integrator block diagram files (.bd)
- Embedded subsystem (.xmp) and files (.elf, .bmm)
- Xilinx® design constraint files (.xdc)
- Scripts for creating and compiling designs (.tcl)
- Configuration files, including Vivado simulator and Vivado logic analyzer configuration files (.wcfg)

The Vivado Design Suite does not have native integration with a particular version control system. Instead, it is designed to work with all version control systems and includes the following features:

- Updates project files and timestamps only when files are modified

  Opening a Vivado Design Suite project does not cause version control-based flows to go stale unless a change is made to a file.
- Supports ASCII-based project files

  This includes files related to projects, source management, IP configurations, flow control, and run management. In most cases, files are in XML format, which is helpful for custom parsing. Binary files are kept to a minimum, such as for storing proprietary information.
- Supports extensive Tcl scripting capabilities

  Object-oriented Tcl extensions work with any complex, custom, and automated version control system.

Vivado supports two design flow modes: Non-Project Mode and Project Mode. Non-Project Mode is a Tcl script-based compilation style method in which you manage sources and the design process yourself. Project Mode is a project-based method that automatically manages your design process and design data using projects and project states. Either of these methods can be run using a Tcl scripted batch mode or run interactively in the graphical user interface (GUI) known as the Vivado Integrated Design Environment (IDE). For more information on the different design flow modes, see the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 1].

Table 1 shows high-level recommendations for working with version control systems in both modes.

Table 1: **High-Level Recommendations for Working with Version Control Systems**

Mode	Recommendation
Non-Project Mode	Create Tcl scripts to manage design sources, control tool options, and design runs as well as generate implementation results.
Project Mode	Use remote project sources with a project creation script generated by the Vivado Design Suite `write_project_tcl` Tcl command.
Both Modes	Use the standalone Manage IP feature to create, configure, and generate IP so that all IP output products are generated in a separate directory and can be recursively checked in.

Note: For more information on Tcl commands, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 2].

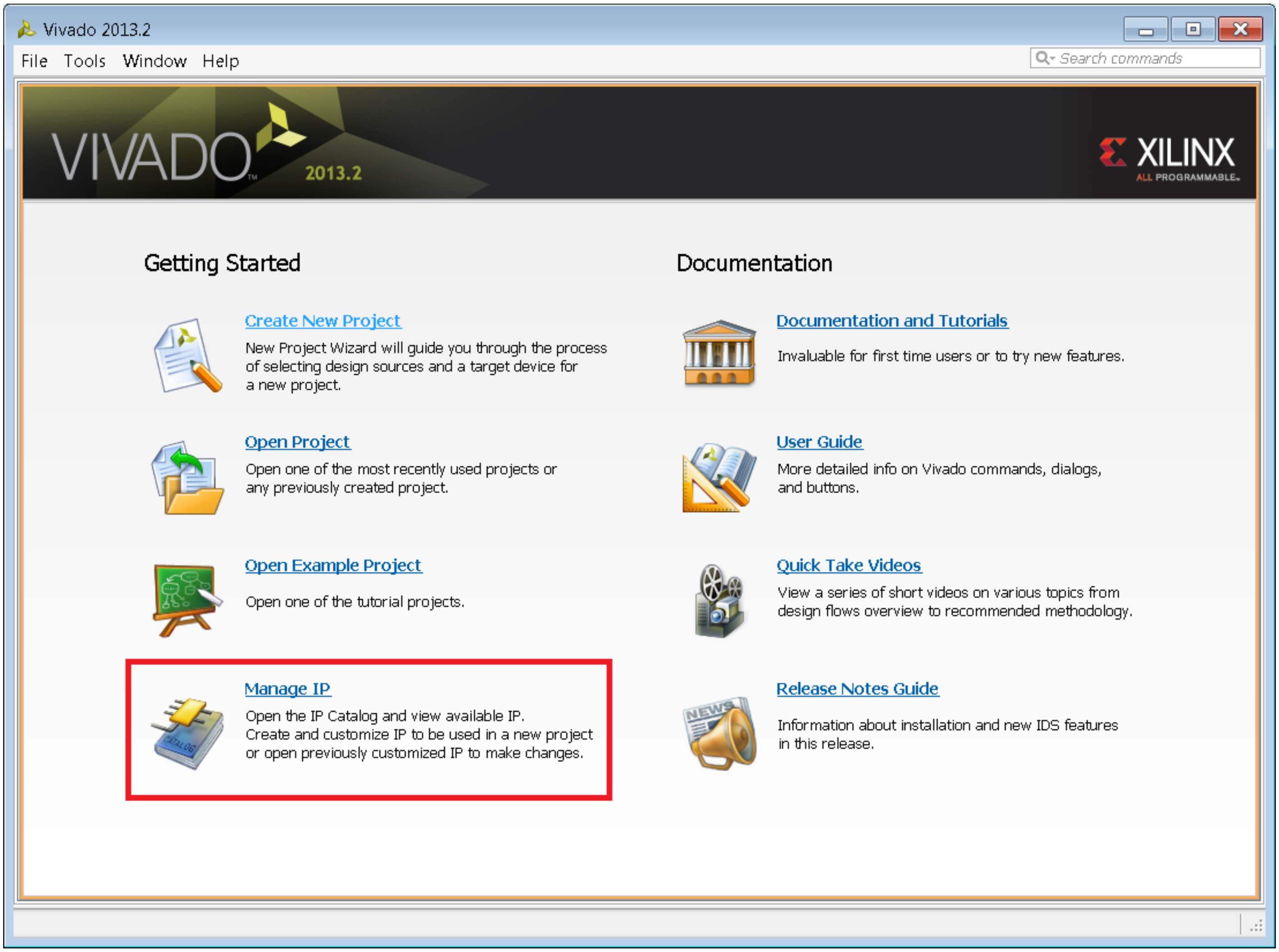
The following sections cover Project Mode and Non-Project Mode and use the example design to discuss the project file system. These sections also provide step-by-step instructions for how to create repositories using Git. Git is an open source-distributed version control and source code management system. It has gained popularity in recent years due its distributed model, speed, and efficiency. Some of the projects available from the Xilinx Open Source Wiki [Ref 3] use Git for source control. For more information on Git, see the Git website [Ref 4].

Version Control and IP

Managing IP

In both Non-Project Mode and Project Mode, the standalone Manage IP feature in the Vivado IDE (Figure 1) is recommended for creating, configuring, and generating IP and IP output products in a separate directory for designs under version control. To manage IP in both Non-Project Mode and Project Mode:

1. In the Vivado IDE Getting Started page, select **Manage IP**.



Getting Started

Create New Project

New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

Open Project

Open one of the most recently used projects or any previously created project.

Open Example Project

Open one of the tutorial projects.

Manage IP

Open the IP Catalog and view available IP. Create and customize IP to be used in a new project or open previously customized IP to make changes.

Documentation

Documentation and Tutorials

Invaluable for first time users or to try new features.

User Guide

More detailed info on Vivado commands, dialogs, and buttons.

Quick Take Videos

View a series of short videos on various topics from design flows overview to recommended methodology.

Release Notes Guide

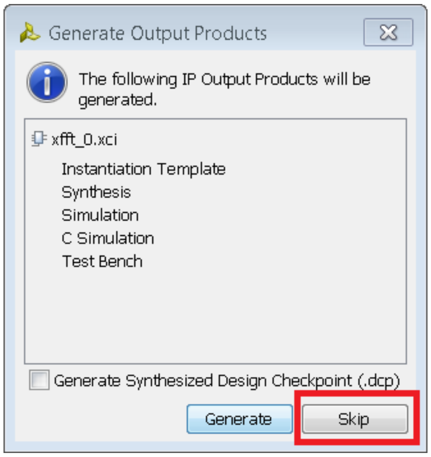
Information about installation and new IDS features in this release.

Figure 1: **Manage IP in the Vivado IDE**

2. In the Vivado IDE, create new IP using the Vivado IP catalog.

3. After configuring an IP, click the **Skip** button in the Generate Output Products dialog box (Figure 2) to minimize the number of files in the IP repository.

   This generates only the IP catalog file (.xml), IP configuration file (.xci), and the VHDL (.vho) or Verilog (.veo) IP instantiation file for the IP.



Generate Output Products

The following IP Output Products will be generated.

xfft_0.xci
- Instantiation Template
- Synthesis
- Simulation
- C Simulation
- Test Bench

□ Generate Synthesized Design Checkpoint (.dcp)

Figure 2: **Generate Output Products Dialog Box**

4. To get a list of all files associated with an IP that must be placed in version control, run the `get_files` command. For example:

```
get_files -all -of [get_files <IP_NAME>.xci]
```

   Note: When using Vivado Design Suite 2013.2, you must run the `git_manage_ip.tcl` script included with the design files for this application note. This Tcl script returns a list of all files associated with the IP in the managed IP locations, gets all IP files that require version control, and generates a batch file to check the files into a Git repository.

5. To import existing IP to the IP repository:

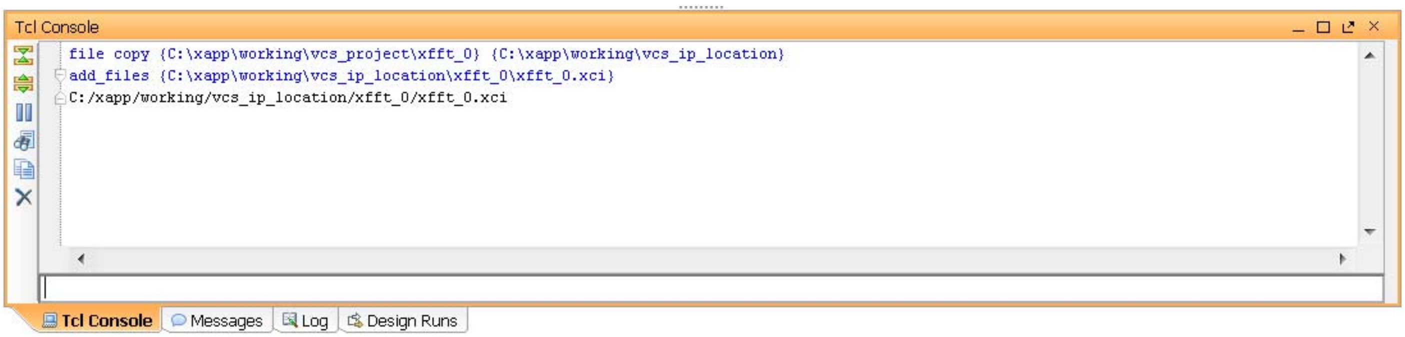
   a. Copy the IP directory to the current IP location. For example:

```
file copy {C:\xapp\working\vcs_project_flow\xfft_0}
{C:\xapp\working\vcs_ip_location}
```

   b. Use the `add_files` Tcl command to add IP configuration files (.xci). For example:

```
add_files {C:\xapp\working\ip_location\xfft_0\xfft_0.xci}
```

   Figure 3 shows these commands used to add an existing FFT core to the managed IP location.



Tcl Console

```
file copy {C:\xapp\working\vcs_project\xfft_0} {C:\xapp\working\vcs_ip_location}
add_files {C:\xapp\working\vcs_ip_location\xfft_0\xfft_0.xci}
C:/xapp/working/vcs_ip_location/xfft_0/xfft_0.xci
```

Figure 3: **add_files Tcl Command**

6. After IP is created or added, recursively check in the same directory structure, IP configuration files (.xci), and data files (.coe, .mem, and so forth) to the design repository.

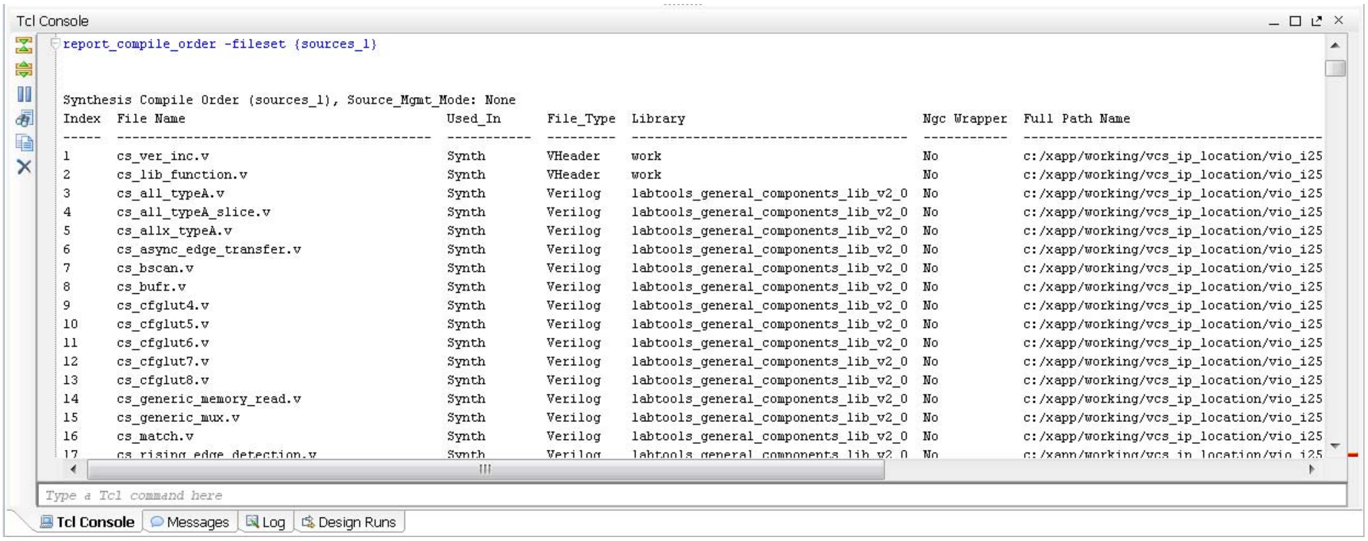
Simulating with IP from the Managed IP Location

Following are two ways to generate an ordered list of IP output files to be used with a simulator or other electronic design automation (EDA) tool.

report_compile_order Tcl Command Method

To return the compile order for all IP in the current Manage IP project, enter the following command (Figure 4):

```
report_compile_order -fileset {sources_1}
```



```
report_compile_order -fileset {sources_1}


Synthesis Compile Order (sources_1), Source_Mgmt_Mode: None
Index  File Name                                  Used_In      File_Type  Library                                Ngc Wrapper  Full Path Name
-----  -----------------------------------------  -----------  ---------  ------------------------------------   -----------  ----------------------------------------
1      cs_ver_inc.v                               Synth        VHeader    work                                   No           c:/xapp/working/vcs_ip_location/vio_i25
2      cs_lib_function.v                          Synth        VHeader    work                                   No           c:/xapp/working/vcs_ip_location/vio_i25
3      cs_all_typeA.v                             Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
4      cs_all_typeA_slice.v                       Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
5      cs_allx_typeA.v                            Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
6      cs_async_edge_transfer.v                   Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
7      cs_bscan.v                                 Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
8      cs_bufr.v                                  Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
9      cs_cfglut4.v                               Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
10     cs_cfglut5.v                               Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
11     cs_cfglut6.v                               Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
12     cs_cfglut7.v                               Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
13     cs_cfglut8.v                               Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
14     cs_generic_memory_read.v                   Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
15     cs_generic_mux.v                           Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
16     cs_match.v                                 Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
17     cs_rising_edge_detection.v                 Synth        Verilog    labtools_general_components_lib_v2_0   No           c:/xapp/working/vcs_ip_location/vio_i25
```

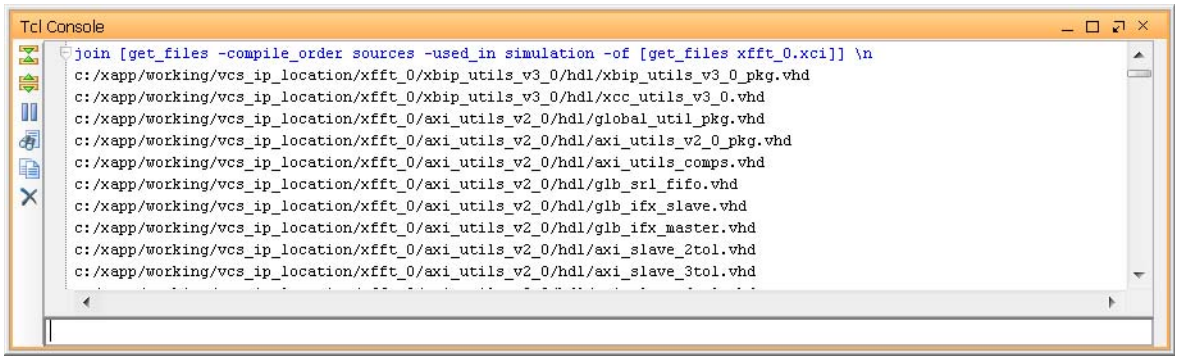
Figure 4: **report_compile_order Tcl Command**

get_files Tcl Command Method

To return the compile order for a specific IP in the Manage IP location, enter the following command (Figure 5):

```
join [get_files -compile_order sources -used_in simulation -of [get_files
xfft_0.xci]] \n
```

Note: Use the `join` command to break the returned list into one file per line.

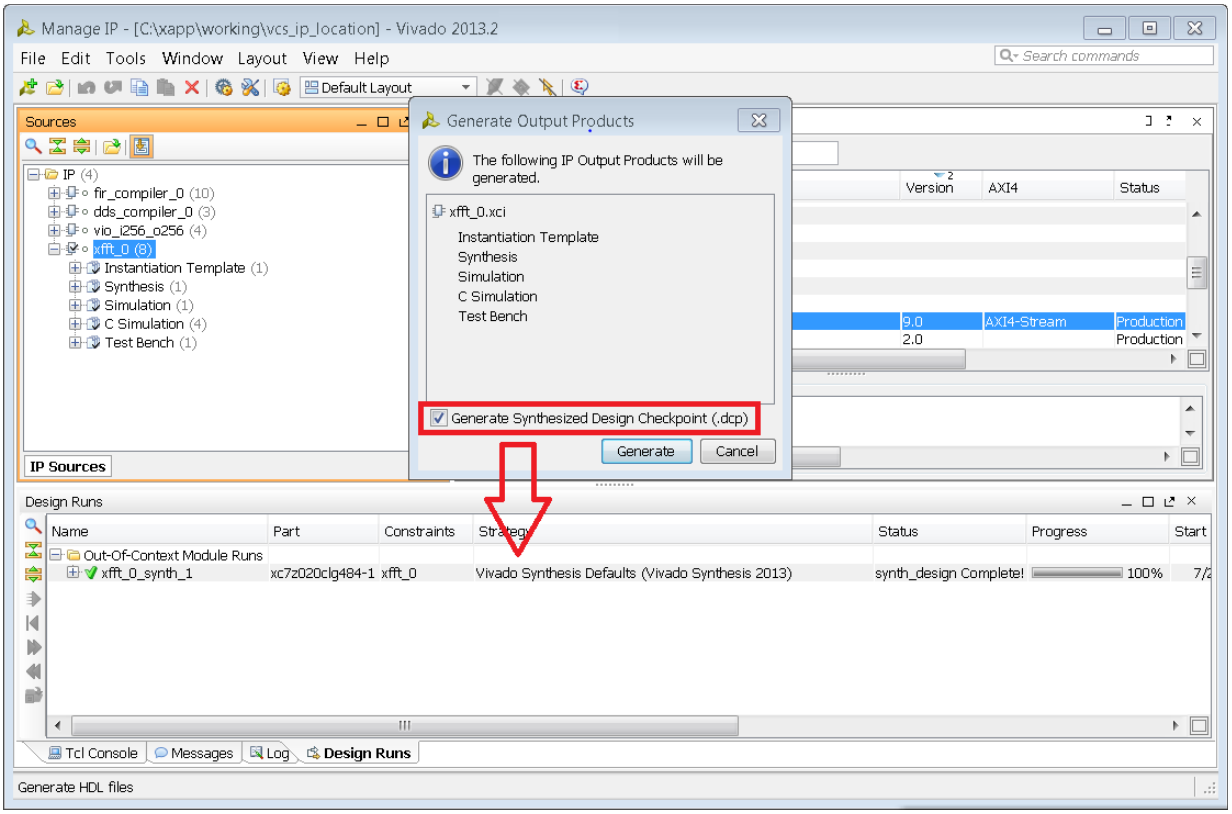


```
join [get_files -compile_order sources -used_in simulation -of [get_files xfft_0.xci]] \n
c:/xapp/working/vcs_ip_location/xfft_0/xbip_utils_v3_0/hdl/xbip_utils_v3_0_pkg.vhd
c:/xapp/working/vcs_ip_location/xfft_0/xbip_utils_v3_0/hdl/xcc_utils_v3_0.vhd
c:/xapp/working/vcs_ip_location/xfft_0/axi_utils_v2_0/hdl/global_util_pkg.vhd
c:/xapp/working/vcs_ip_location/xfft_0/axi_utils_v2_0/hdl/axi_utils_v2_0_pkg.vhd
c:/xapp/working/vcs_ip_location/xfft_0/axi_utils_v2_0/hdl/axi_utils_comps.vhd
c:/xapp/working/vcs_ip_location/xfft_0/axi_utils_v2_0/hdl/glb_srl_fifo.vhd
c:/xapp/working/vcs_ip_location/xfft_0/axi_utils_v2_0/hdl/glb_ifx_slave.vhd
c:/xapp/working/vcs_ip_location/xfft_0/axi_utils_v2_0/hdl/glb_ifx_master.vhd
c:/xapp/working/vcs_ip_location/xfft_0/axi_utils_v2_0/hdl/axi_slave_2to1.vhd
c:/xapp/working/vcs_ip_location/xfft_0/axi_utils_v2_0/hdl/axi_slave_3to1.vhd
```

Figure 5: **get_files Tcl Command**

Bottom-Up Synthesis Flow with IP from the Managed IP Location

Some design environments use bottom-up synthesis to save IP recompile time and preserve synthesis results. This methodology is supported and automated in the Manage IP project. When generating output products, enable **Generate Synthesized Design Checkpoint (.dcp)** in the Generate Output Products dialog box (Figure 6). This automatically creates and synthesizes an out-of-context module run. The synthesized design checkpoint is needed for check in into the design repository in the bottom-up synthesis flow.



Manage IP - [C:\xapp\working\vcs_ip_location] - Vivado 2013.2

Generate Output Products

The following IP Output Products will be generated.

xfft_0.xci
- Instantiation Template
- Synthesis
- Simulation
- C Simulation
- Test Bench

☑ Generate Synthesized Design Checkpoint (.dcp)

Sources

- IP (4)
  - fir_compiler_0 (10)
  - dds_compiler_0 (3)
  - vio_i256_o256 (4)
  - xfft_0 (8)
    - Instantiation Template (1)
    - Synthesis (1)
    - Simulation (1)
    - C Simulation (4)
    - Test Bench (1)

Version	AXI4	Status
9.0	AXI4-Stream	Production
2.0		Production

Design Runs

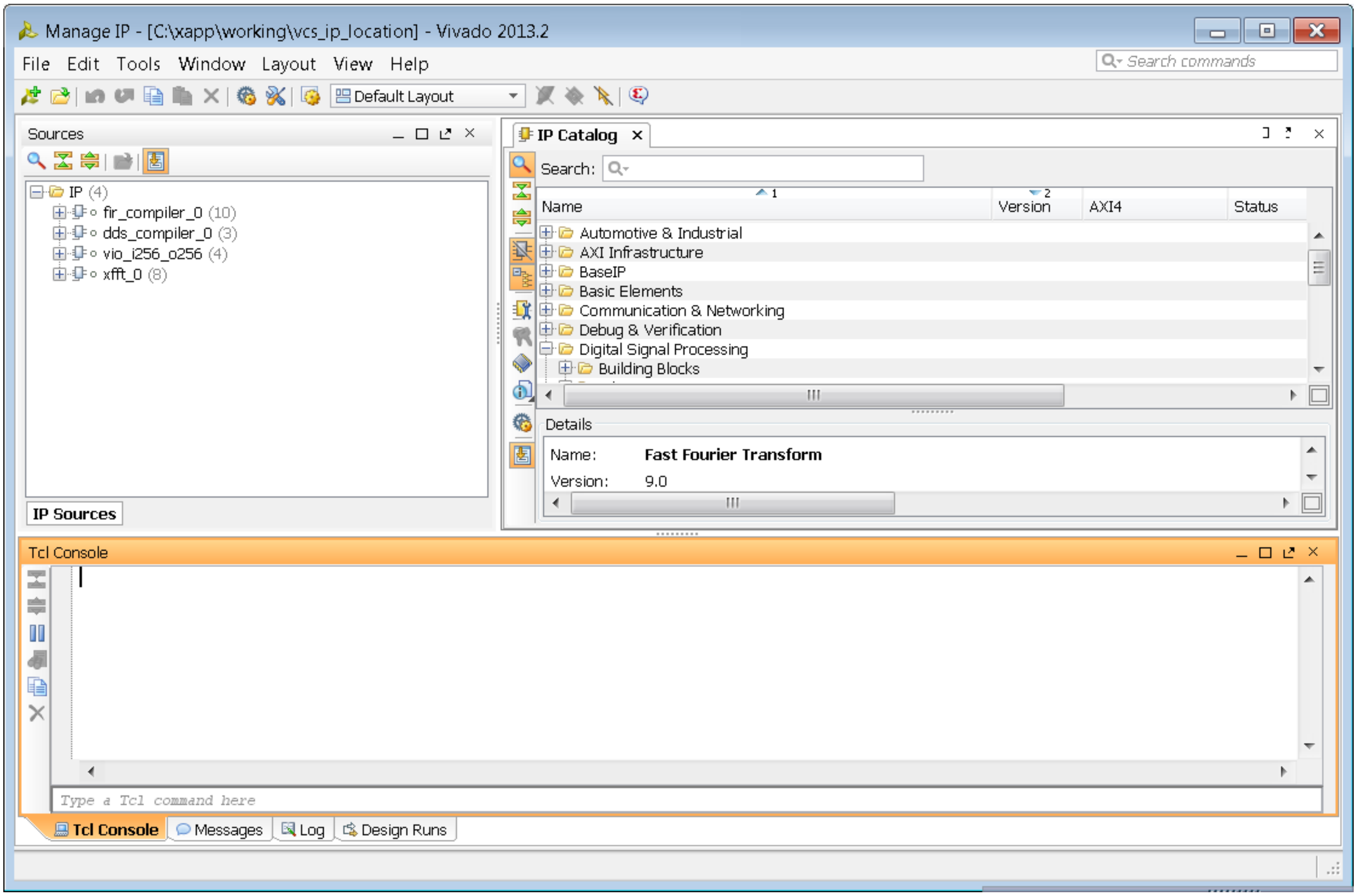
Name	Part	Constraints	Strategy	Status	Progress	Start
Out-Of-Context Module Runs						
xfft_0_synth_1	xc7z020clg484-1	xfft_0	Vivado Synthesis Defaults (Vivado Synthesis 2013)	synth_design Complete!	100%	7/2

Generate HDL files

Figure 6: **Generate Synthesized Design Checkpoint**

IP Location for Example Projects

Figure 7 shows the IP location for the example designs in Non-Project Mode and Project Mode.



- IP (4)
  - fir_compiler_0 (10)
  - dds_compiler_0 (3)
  - vio_i256_o256 (4)
  - xfft_0 (8)

Name	Version	AXI4	Status
Automotive & Industrial			
AXI Infrastructure			
BaseIP			
Basic Elements			
Communication & Networking			
Debug & Verification			
Digital Signal Processing			
Building Blocks			

Details

Name: **Fast Fourier Transform**

Version: 9.0

Figure 7: **IP Location for Example Designs**

Following is the directory structure for the example IP location:

```
vcs_ip_location
|-- git_manage_ip.tcl
|-- coeffs
|   `-- fir_p2_1db_s3_50db.coe
|-- dds_compiler_0
|   |-- dds_compiler_0.vho
|   |-- dds_compiler_0.xci
|   `-- dds_compiler_0.xml
|-- fir_compiler_0
|   |-- fir_compiler_0.vho
|   |-- fir_compiler_0.xci
|   `-- fir_compiler_0.xml
|-- ila_0
|   |-- ila_0.xci
|   `-- ila_0.xml
|-- vio_i256_o256
|   |-- vio_i256_o256.xci
|   `-- vio_i256_o256.xml
|-- xfft_0
         |-- xfft_0.vho
         |-- xfft_0.xci
         `-- xfft_0.xml
```

Non-Project Mode

In Non-Project Mode, the Vivado Design Suite takes design sources and constraints as direct input and executes the entire flow in memory. Configuration files are not saved on the hard disk, and there is no project management or flow navigation in Non-Project Mode. However, you can create design checkpoints and reports at any stage and open them in the Vivado IDE for design analysis. Non-Project Mode maintains control for command line users and also provides powerful features available in the Vivado IDE, such as cross probing and constraints assignments using a design checkpoint.

Non-Project Mode is commonly used and highly recommended for working with version control systems. When using Non-Project Mode, you control the input that is passed to the tool and the options that are used. It is recommended that you place all the inputs to the compilation under version control with a Tcl script that controls the compilation from those inputs to produce the desired final outputs. Following are files that must be checked in the repository in this flow:

- Tcl scripts for compiling and controlling tool options
- Input files passed to `read_*` or `add_source` Tcl commands
- Optional tool reports and output products (for example, bitstream, design checkpoints, and reports)

Note: The entire project must *not* be placed under version control.

Example Design Directory Tree in Non-Project Mode

Following is an example directory structure for a Non-Project Mode design. This is the minimum fileset that must be version controlled in the repository to implement and reproduce synthesis and implementation run results. All the IP are from the IP location generated using the Manage IP feature.

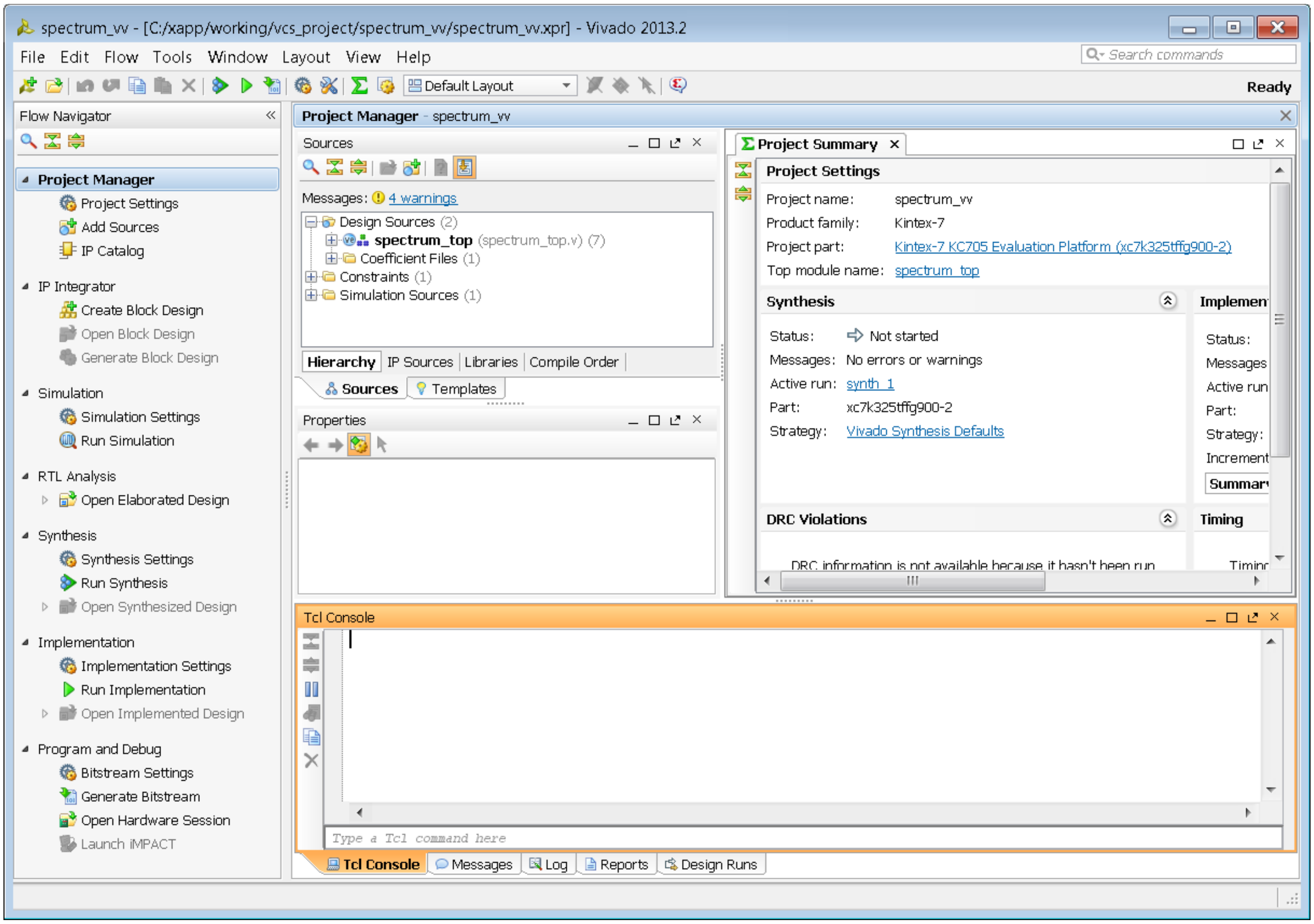
```
|-- script
|   |-- file_list.txt
|   `-- vivado_non-project.tcl
|-- sim
|   `-- spectrum_top_tb.v
`-- src
    |-- constrs
    |   |-- spectrum_top_physical.xdc
    |   `-- spectrum_top_timing.xdc
    |-- verilog
    |   |-- cplx_mult.v
    |   `-- spectrum_top.v
    `-- vhdl
        `-- heartbeat_gen.vhd
```

Project Mode

In Project Mode, the Vivado IDE creates a project file system and manages project data and status. You can manage the project using the Vivado IDE, using Project Mode Tcl commands interactively in the Vivado IDE, or using Tcl scripts. The Vivado IDE provides a robust design environment with a consistent interface for the entire design cycle. The following automated features are available when using Project Mode:

- Configuration and management of design sources and runs
- IP configuration, implementation, and packaging
- Design analysis and verification with comprehensive cross probing capabilities
- Constraint creation and assignment for I/O, clock, timing, and floorplanning
- Vivado logic analyzer core insertion for debug and analysis
- Bitstream generation and programming

Figure 8 shows the example design in the Vivado IDE.



spectrum_vv - [C:/xapp/working/vcs_project/spectrum_vv/spectrum_vv.xpr] - Vivado 2013.2

Messages: 4 warnings

- Design Sources (2)
  - spectrum_top (spectrum_top.v) (7)
  - Coefficient Files (1)
- Constraints (1)
- Simulation Sources (1)

Project Settings

Project name: spectrum_vv
Product family: Kintex-7
Project part: Kintex-7 KC705 Evaluation Platform (xc7k325tffg900-2)
Top module name: spectrum_top

Synthesis

Status: Not started
Messages: No errors or warnings
Active run: synth_1
Part: xc7k325tffg900-2
Strategy: Vivado Synthesis Defaults

DRC Violations

DRC information is not available because it hasn't been run

Figure 8: **Example Design**

Version Control Recommendations in Project Mode

Following are recommendations when integrating a version control system with Project Mode:

- Keep all input sources, especially IP sources, in remote source directories.

  Add the sources to the project as referenced sources instead of importing them into the project.

- Create and maintain a Tcl script to create the project, tool options, and settings.

  Vivado Design Suite 2013.1 and newer provides the `write_project_tcl` Tcl command that exports a Tcl script for creating the project. You can use this script as a project creation script template and can modify it to customize the flow or location of input sources.

- Change project settings in the project creation Tcl script as much as possible.

  Changes made to the project interactively in the Vivado IDE must be written out to a new Tcl script using the `write_project_tcl` command. Use the merge function provided by the version control system to resolve the differences between the new script and the one in the repository.

- Check in and check out files using the method supported by the version control system.

  You can also check in and check out files using command line commands in the Tcl console of the Vivado IDE.

Example Project Directory Tree in Project Flow

Following is an example directory structure for a Project Mode project. The `create_specturm_prj.tcl` script in the script directory is generated with the `write_project_tcl` command, which creates the project and all the settings used in Project Mode.

```
vcs_project
|-- debug
|   `-- hw_ila_data_1.wcfg
|-- script
|   |-- file_list.txt
|   |-- git_project.tcl
|   `-- spectrum_vv_prj_flow.tcl
|-- sim
|   |-- spectrum_top_tb.v
|   `-- spectrum_top_tb.wcfg
|-- script
|   `-- spectrum_top_tb.wcfg
`-- src
    |-- constrs
    |   |-- spectrum_top_physical.xdc
    |   `-- spectrum_top_timing.xdc
    |-- verilog
    |   `-- cplx_mult.v
    |   `-- spectrum_top.v
    `-- vhdl
        `-- heartbeat_gen.vhd
```

write_project_tcl Tcl Command

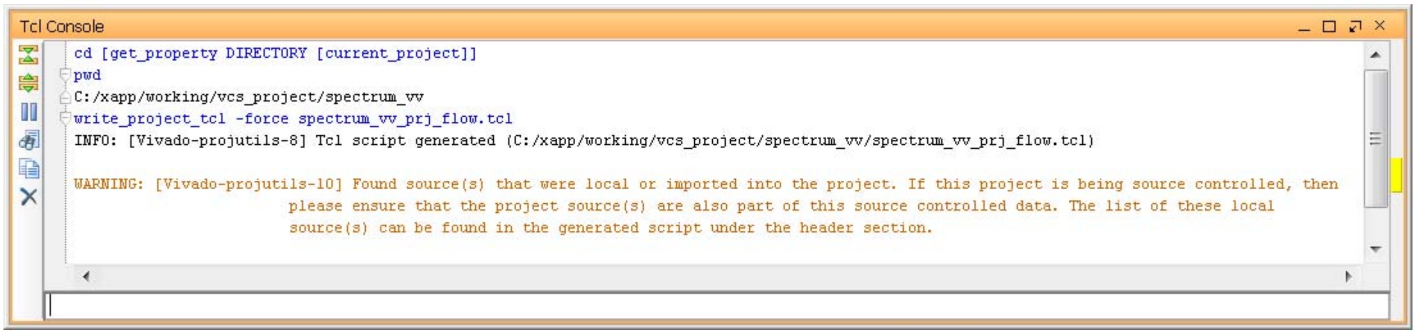
Vivado Design Suite 2013.1 and newer provides the `write_project_tcl` command, which is designed to integrate a Vivado Design Suite project with any version control system. This command exports a Tcl script that recreates the current project. You can do either of the following:

- Create a project manually using the Vivado IDE.
- Use an existing project, and export a Tcl script that recreates the project using the input sources and the project settings.

Note: It is recommended that the script is version controlled so that the project can be cleanly recreated. Attempting to place actively managed projects under version control is *not* recommended.

Before running Tcl commands that export files, it is recommended that you change the current working directory to the desired destination directory if needed. The following commands change to the current project directory and run the `write_project_tcl` command (Figure 9):

```
pwd
cd [get_property DIRECTORY [current_project]]
pwd
write_project_tcl -force spectrum_vv_prj_flow.tcl
```



```
cd [get_property DIRECTORY [current_project]]
pwd
C:/xapp/working/vcs_project/spectrum_vv
write_project_tcl -force spectrum_vv_prj_flow.tcl
INFO: [Vivado-projutils-8] Tcl script generated (C:/xapp/working/vcs_project/spectrum_vv/spectrum_vv_prj_flow.tcl)

WARNING: [Vivado-projutils-10] Found source(s) that were local or imported into the project. If this project is being source controlled, then
                      please ensure that the project source(s) are also part of this source controlled data. The list of these local
                      source(s) can be found in the generated script under the header section.
```

Figure 9: **write_project_tcl Tcl Command**

After the commands are run, the following occurs:

- Warnings are issued for any input sources that are local or imported to the project.

  It is recommended that all input sources reside in a remote source directory.
- The `spectrum_vv_prj_flow.tcl` Tcl script is saved to the current working directory.

  Following is detailed information on important commands in the script, which you can use to customize the script:

```
# Create project
create_project spectrum_vv ./spectrum_vv

# Set project properties (parts, boards, etc)
set obj [get_projects spectrum_vv]
set_property "board" "xilinx.com:kintex7:kc705:1.0" $obj

# Create 'sources_1' fileset
if {[string equal [get_filesets sources_1] ""]} {
  create_fileset -srcset sources_1
}

# Add files to 'sources_1' fileset. All files including IP configuration
files added with add_files command need to be checked in to the design
repository
set obj [get_filesets sources_1]
set files [list \
 "C:/dds_compiler_v5_0_0/dds_compiler_v5_0_0.xci"\

"C:/spectrum_vv/spectrum_vv.srcs/sources_1/ip/xfft_v8_0_0/xfft_v8_0_0.xci
"\
 "C:/xapp/working/vcs_project/src/coeff/fir_p2_1db_s3_50db.coe"\
 "C:/xapp/working/vcs_project/src/vhdl/heartbeat_gen.vhd"\
]
add_files -norecurse -fileset $obj $files

# Set the top module for the project
set obj [get_filesets sources_1]
set_property "top" "spectrum_top" $obj
```

```
# Create 'constrs_1' fileset (if not found)
if {[string equal [get_filesets constrs_1] ""]} {
  create_fileset -constrset constrs_1
}

# Add files to 'constrs_1' fileset All files added with add_files command
need to be checked in to the design repository
set obj [get_filesets constrs_1]
set files [list \
 "C:/xapp/working/vcs_project/src/constrs/spectrum_top_physical.xdc"\
 "C:/xapp/working/vcs_project/src/constrs/spectrum_top_timing.xdc"\
]
add_files -norecurse -fileset $obj $files

# Set 'constrs_1' fileset file properties for local files
set file "constrs/spectrum_top_physical.xdc"
set file_obj [get_files "*$file" -of_objects constrs_1]
set_property "file_type" "XDC" $file_obj

# Add files to 'sim_1' fileset All files added with add_files command need
to be checked in to the design repository
set obj [get_filesets sim_1]
set files [list \
 "C:/xapp/working/vcs_project/sim/spectrum_top_tb.v"\
]
add_files -norecurse -fileset $obj $files

# Set top module for the testbench
set obj [get_filesets sim_1]
set_property "top" "spectrum_top_tb" $obj

# Create 'synth_1' run (if not found)
if {[string equal [get_runs synth_1] ""]} {
  create_run -name synth_1 -part xc7k325tffg900-2 -flow {Vivado Synthesis
2012} -strategy "Vivado Synthesis Defaults" -constrset constrs_1
}

# Create 'impl_1' run (if not found)
if {[string equal [get_runs impl_1] ""]} {
  create_run -name impl_1 -part xc7k325tffg900-2 -flow {Vivado
Implementation 2012} -strategy "Vivado Implementation Defaults" -constrset
constrs_1 -parent_run synth_1
```

}

Git Version Control

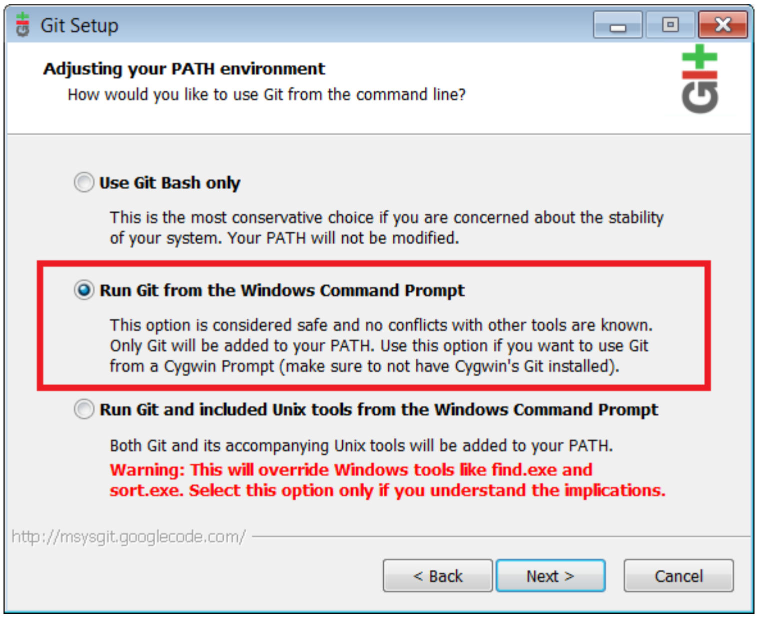
Git is an open source-distributed version control and source code management system. Each Git working directory is a full repository and does not require a central server to store history and version tracking data. The Git repository is stored in a single `.git` directory in the root directory of a project. This works well with EDA tools that dynamically add or delete working directories.

Git Installation

To install Git:

1. Download the latest version of Git from the Git website [Ref 4].
2. Install Git with default options *except* enable the **Run Git from the Windows Command Prompt** option in the Adjusting your PATH environment dialog box (Figure 10).

   Note: This application note uses Git-1.8.3-preview20130601 on a Windows 7 64-bit machine.



Adjusting your PATH environment

How would you like to use Git from the command line?

○ **Use Git Bash only**

This is the most conservative choice if you are concerned about the stability of your system. Your PATH will not be modified.

● **Run Git from the Windows Command Prompt**

This option is considered safe and no conflicts with other tools are known. Only Git will be added to your PATH. Use this option if you want to use Git from a Cygwin Prompt (make sure to not have Cygwin's Git installed).

○ **Run Git and included Unix tools from the Windows Command Prompt**

Both Git and its accompanying Unix tools will be added to your PATH.
Warning: This will override Windows tools like find.exe and sort.exe. Select this option only if you understand the implications.

http://msysgit.googlecode.com/

Figure 10: **Git Setup Options**

Git Repository Creation

This section provides step-by-step instructions for creating an initial design repository for the example design in Non-Project Mode.

Note: Repository creation for Project Mode is similar and also uses a project creation Tcl script for project management.

Initializing a Git Repository in Non-Project Mode

To initialize a Git repository:

1. In Windows Explorer, create a directory for the repository. For example: `C:/xapp/repository/vcs_non_project`
2. Navigate to the directory, right-click, and select **Git Bash** (Figure 11).

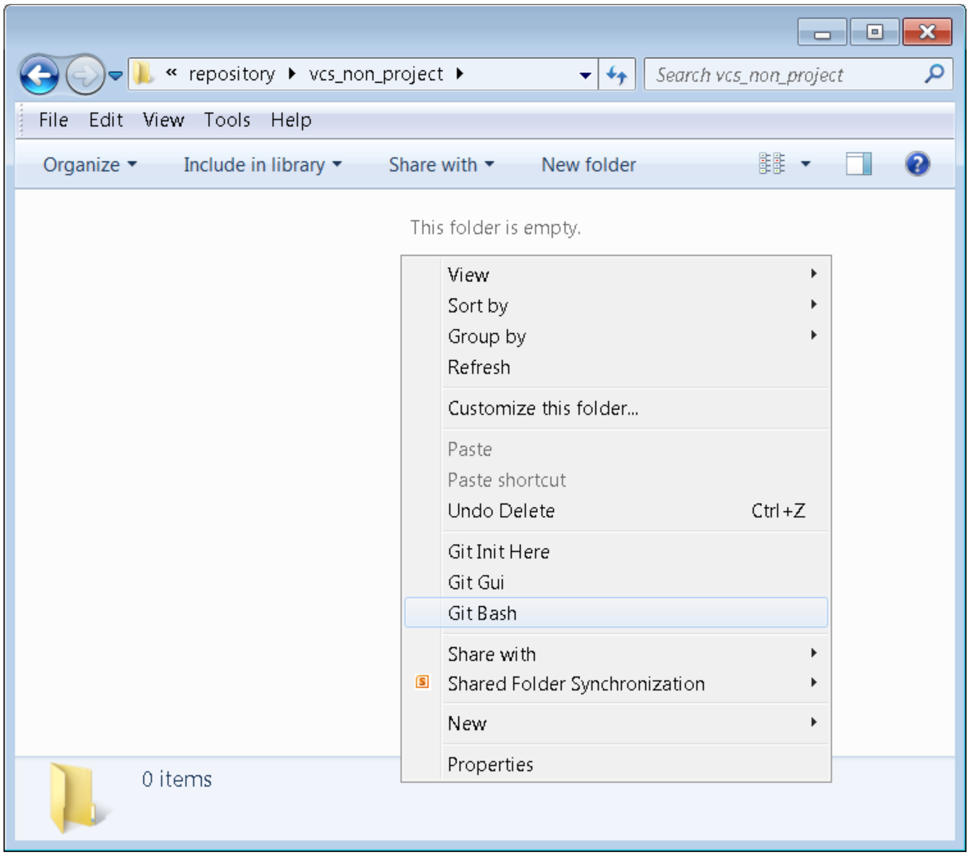
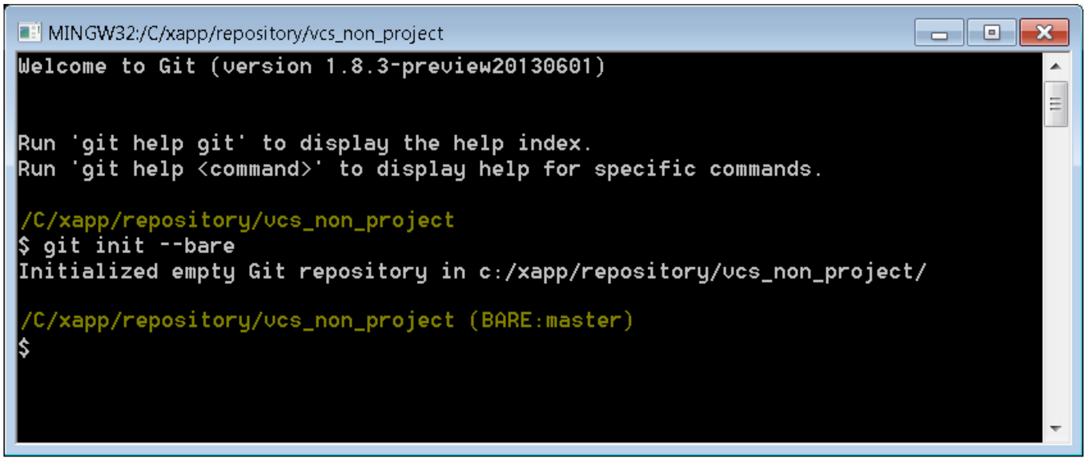


Figure 11: **Git Bash Command in vcs_non_project Directory**

3. In the Git Bash window, run the following command to create a bare repository (Figure 12):

```
$ git init --bare
```

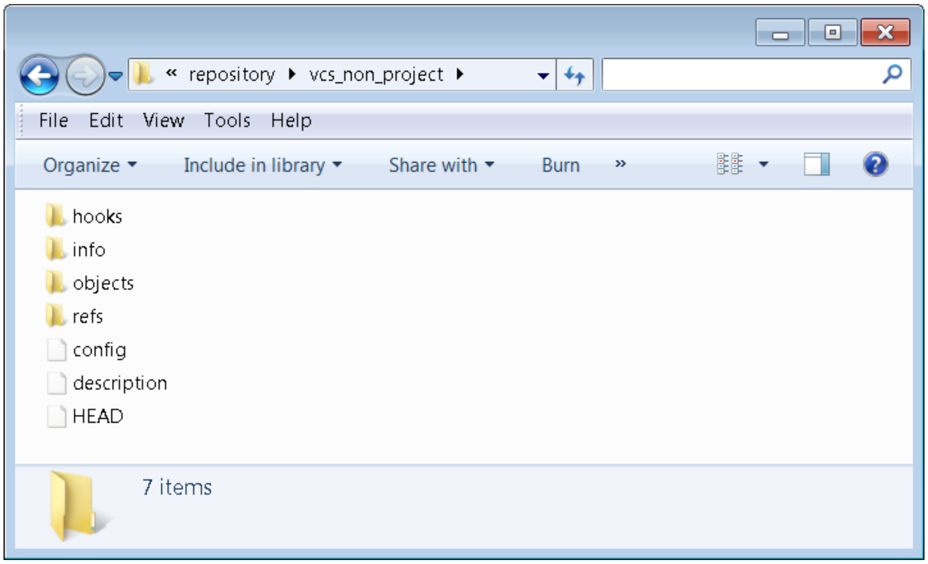

MINGW32:/C/xapp/repository/vcs_non_project

```
Welcome to Git (version 1.8.3-preview20130601)


Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

/C/xapp/repository/vcs_non_project
$ git init --bare
Initialized empty Git repository in c:/xapp/repository/vcs_non_project/

/C/xapp/repository/vcs_non_project (BARE:master)
$
```

Figure 12: **Bare Repository Creation**

Similar to a central repository in the Concurrent Versions System (CVS) or Apache Subversion (SVN) version control systems, each team member can pull changes from and push changes to the bare repository. Figure 13 shows the directory and file structure of the bare repository.

Note: The bare repository can only contain version control information and does not contain working files.


« repository ▸ vcs_non_project ▸
File Edit View Tools Help
Organize ▾ Include in library ▾ Share with ▾ Burn »

hooks
info
objects
refs
config
description
HEAD

7 items

Figure 13: **Bare Repository**

Creating a Working Repository in Non-Project Mode

Git is a distributed version control system. The working area for each team member is a full clone of the Git bare repository. To clone the bare repository:

1. In Windows Explorer, create a directory for the working area. For example: `C:/xapp/working`
2. Navigate to the directory, right-click, and select **Git Bash** (Figure 14).

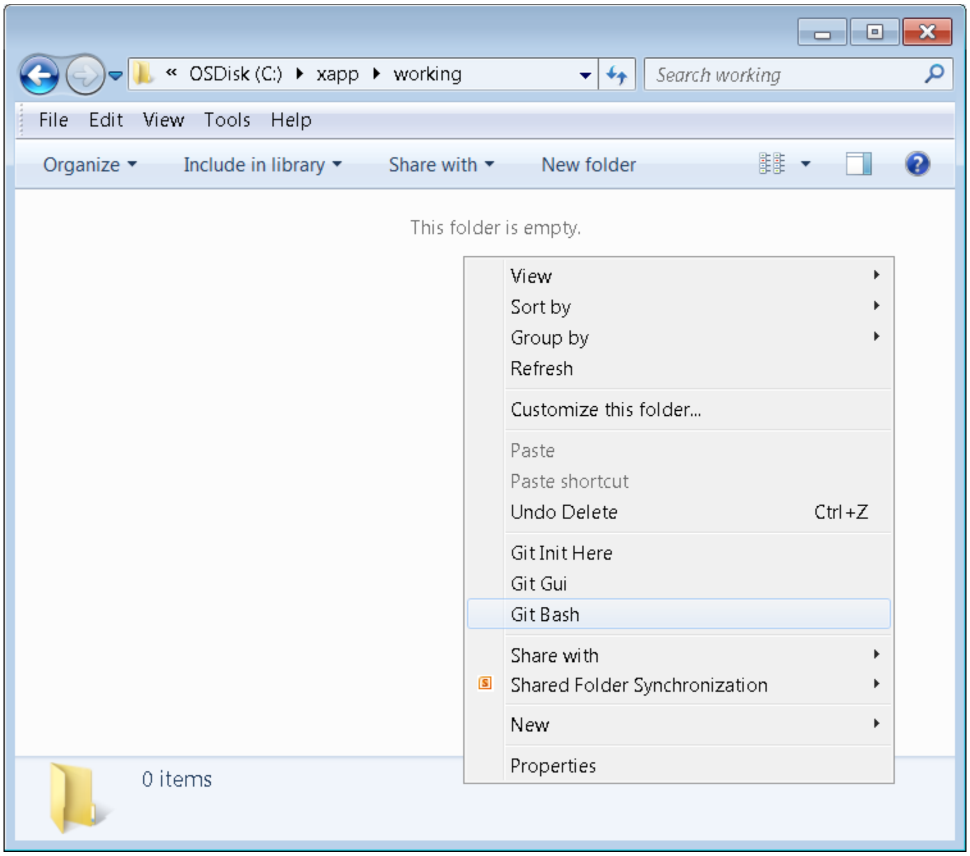
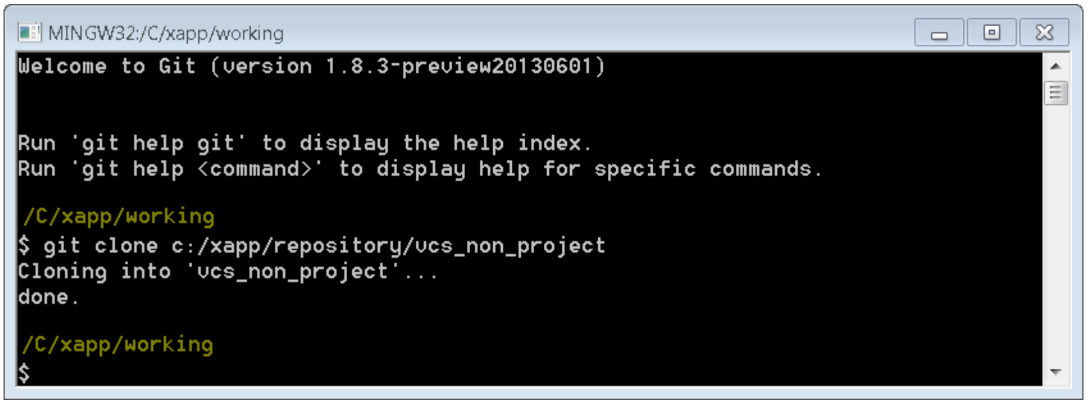


Figure 14: **Git Bash Command in working Directory**

3. In the Git Bash window, run the following command to clone the bare repository (Figure 15):

   ```
   $ git clone c:/xapp/repository/vcs_non_project
   ```

   This command creates the `vcs_non_project` directory as the working repository, which includes a hidden directory named `.git` to store version control information.



```
Welcome to Git (version 1.8.3-preview20130601)


Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

/C/xapp/working
$ git clone c:/xapp/repository/vcs_non_project
Cloning into 'vcs_non_project'...
done.

/C/xapp/working
$
```

Figure 15: **Bare Repository Cloning to Working Repository**

4. Unzip the example design archive `xapp1165.zip` to a directory.

5. Copy all subdirectories in the `vcs_non_project` directory to the working repository. For example: `C:/xapp/working/vcs_non_project`

   Figure 16 shows the directory and file structure of the working repository.

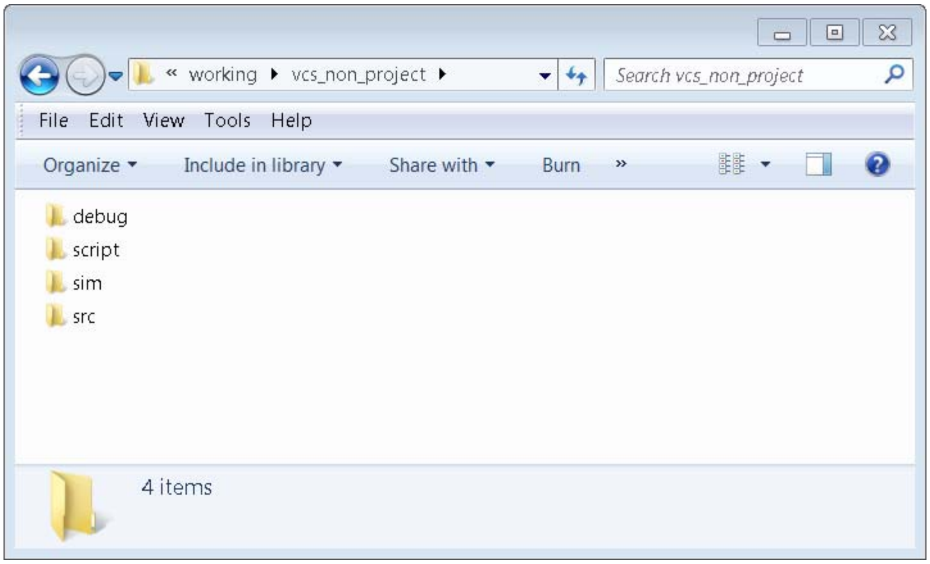

« working ▸ vcs_non_project ▸
Search vcs_non_project
File Edit View Tools Help
Organize ▾ Include in library ▾ Share with ▾ Burn »
debug
script
sim
src
4 items

Figure 16: **Working Repository**

6. From a Vivado Design Suite command prompt window, create a `run` directory. For example: `C:/xapp/working/vcs_non_project/run`
7. Navigate to the `run` directory, and run the following command to generate the `git_non_project.bat` file:

```
vivado -mode batch -source ../script/vivado_non_project.tcl -notrace
-tclargs -cmd vcs -ifn ../script/file_list.txt -top spectrum_top -part
xc7k325tffg900-2 -out git_non_project.bat
```

8. In Windows Explorer, navigate to the `run` directory. For example: `C:/xapp/working/vcs_non_project/run`
9. Right-click, and select **Git Bash**.
10. In the Git Bash window, run the following command to check in all design sources and push them to the shared repository:

```
git_non_project.bat
```

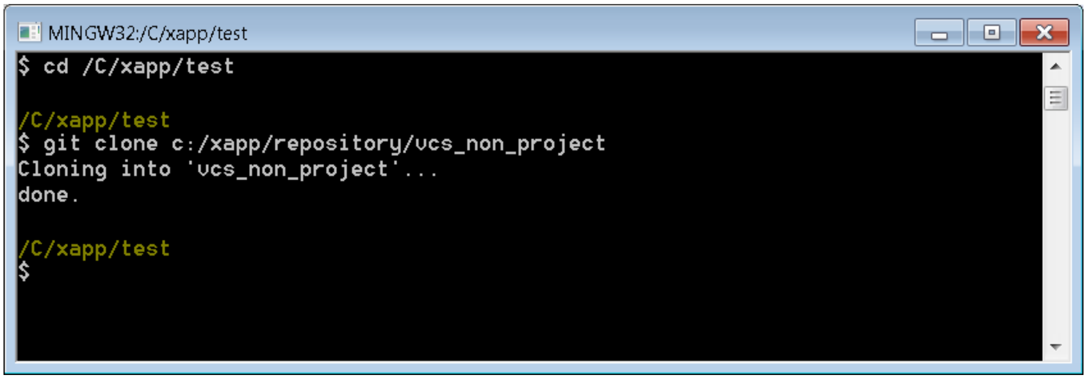
Testing the Shared Repository in Non-Project Mode

It is important to verify that the minimum fileset in the shared repository includes all the files required to implement the design in Non-Project Mode. To verify:

1. In the Git Bash window, run the following commands to clone the shared repository and create a test repository (Figure 17):

   ```
   $ cd /C/xapp/test
   $ git clone c:/xapp/repository/vcs_non_project
   ```

   Note: The root of the test repository must be in the same directory as the `vcs_ip_location` directory.



```
$ cd /C/xapp/test

/C/xapp/test
$ git clone c:/xapp/repository/vcs_non_project
Cloning into 'vcs_non_project'...
done.

/C/xapp/test
$
```

Figure 17: **Working Repository Cloning to Test Repository**

Figure 18 shows the directory and file structure of the test repository.

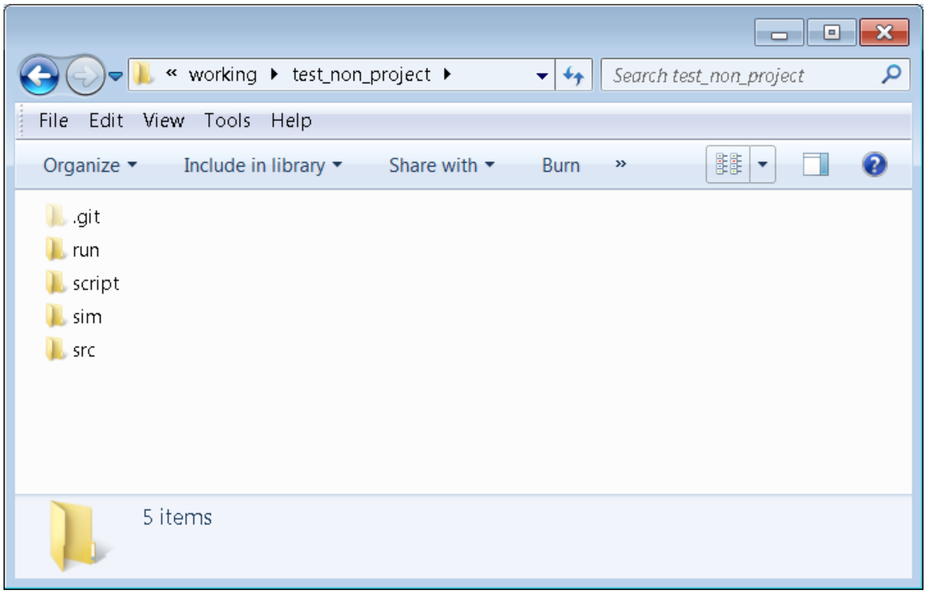
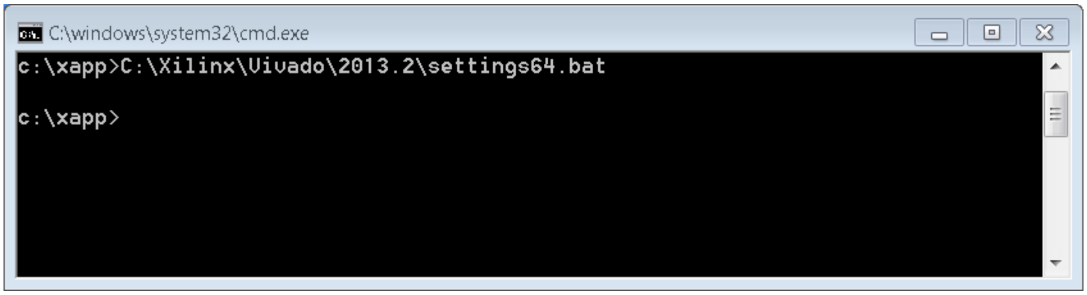

« working ▸ test_non_project ▸
Search test_non_project
File Edit View Tools Help
Organize ▾ Include in library ▾ Share with ▾ Burn »
.git
run
script
sim
src
5 items

Figure 18: **Test Repository**

2. In Windows Explorer, create a directory called `run` in the test directory. For example: `C:/xapp/test/test_non_project`

3. To open a Vivado Design Suite command prompt, open a Windows command prompt window and execute the Vivado Design Suite environment setup batch file (Figure 19). For example:

   ```
   C:\Xilinx\Vivado\2013.2\settings64.bat
   ```



```
c:\xapp>C:\Xilinx\Vivado\2013.2\settings64.bat

c:\xapp>
```

Figure 19: **Vivado Design Suite Environment Setup Batch File**

4. Navigate to the `run` directory, and run the following command to implement the design in Non-Project Mode:

   ```
   vivado -mode batch -source ../script/non-project_run.tcl -notrace
   -tclargs -cmd run -ifn ../script/file_list.txt -top spectrum_top -part
   xc7k325tffg900-2
   ```

   The run script generates all IP, runs synthesis and implementation, and generates a bitstream.

Reference Design

The example project included in this application note is a complete spectrum analysis design that runs from RTL to bitstream to hardware debug in the Vivado logic analyzer targeting the KC705 evaluation board. The design generates a sinusoidal signal using a direct digital synthesizer (DDS) compiler core, which is then processed in a finite impulse response (FIR) compiler core and fast Fourier transform (FFT) core. The digital signal processing (DSP) output is passed to a complex multiplier implemented in behavioral RTL for magnitude calculation. The project is a typical design environment that pulls in design sources from both user and tool-managed directories and uses multiple source types, including RTL, IP-XACT IP, simulation test bench and configuration files, and Vivado logic analyzer core inserter files. The example project is verified in Vivado Design Suite 2013.2.

The reference design files for this application note can be downloaded from: https://secure.xilinx.com/webreg/clickthrough.do?cid=344210

The reference design matrix is shown in Table 2.

Table 2: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Jim Wu
Target devices	Kintex-7 FPGAs
Source code provided	Yes
Source code format	VHDL/Verilog
Design uses code and IP from existing Xilinx application note and reference designs, CORE Generator™ tool, or third party	No
Simulation	
Functional simulation performed	Yes
Timing simulation performed	N/A
Test bench used for functional and timing simulations	Yes
Test bench format	Verilog
Simulator software/version used	Vivado simulator 2013.2
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/versions used	Vivado synthesis 2013.2
Implementation software tools/versions used	Vivado implementation 2013.2
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	KC705 evaluation board

References

This document includes the following references:

1. *Vivado Design Suite User Guide: Design Flows Overview* (UG892)
2. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
3. Xilinx Open Source Wiki (http://wiki.xilinx.com/)
4. Git website (http://git-scm.com/)

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
08/05/2013	1.0	Initial Xilinx release.

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.