Lab Instructions: Revision Control

Contents

[Introduction 1](#_Toc411524398)

[Lab Files 2](#_Toc411524399)

[Working in Shells 2](#_Toc411524400)

[About the Lab Questions 3](#_Toc411524401)

[Lab 1 : Simple RTL Project 3](#_Toc411524402)

[Lab Procedure 3](#_Toc411524403)

[Conclusion 5](#_Toc411524404)

[Lab 2: Simple IP Project 6](#_Toc411524405)

[Lab Procedure 6](#_Toc411524406)

[Conclusion 7](#_Toc411524407)

[Lab 3: Custom IP Project 7](#_Toc411524408)

[Lab Procedure 7](#_Toc411524409)

[Conclusion 9](#_Toc411524410)

[Lab 4: HLS-Based IP 10](#_Toc411524411)

[Lab Procedure 10](#_Toc411524412)

[Conclusion 11](#_Toc411524413)

[Lab 5: IPI Block Design Project 11](#_Toc411524414)

[Lab Procedure 11](#_Toc411524415)

[Conclusion 13](#_Toc411524416)

[Lab 6: Top Level Integration 13](#_Toc411524417)

[Lab Procedure 13](#_Toc411524418)

[Conclusion 14](#_Toc411524419)

# Introduction

This document includes the instructions for the Revision Control lab exercises. There are 6 labs:

* Lab 1: Simple RTL Project
* Lab 2: Simple IP Project
* Lab 3: Custom IP Project
* Lab 4: HLS-Based IP
* Lab 5: IPI Block Design Project
* Lab 6: Top Level Integration

We will be using the following tools for these labs:

* Git revision control – included with Vivado
* MinGW make utility - included with Vivado System Edition
* Vivado 2015.1 System Edition and SDK

Please be sure to review the separate Pre-Work document if you have not already done so. It contains information about setup and installation of Git and a brief introduction to the make utility.

## Lab Files

The labs are created to run on Windows. Attendees can modify scripts and Makefiles to run on Linux if necessary.

Since revision control is an essential concept for the labs, all 6 labs will all use the same working directory and revision control repository. First create a “root” location where you will store the lab source files and working directory. For example:

* C:\training\revCtrl
* C:\users\your\_username\revCtrl

Copy or extract the lab files into this directory. The following files are provided with the labs, organized in these directories:

* **doc**: lab instructions and presentation
* **scripts**: Vivado run scripts and Makefiles
* Design source code is further organized in the following directories:
  + **hdl**: Verilog HDL including the top-level design
  + **xdc**: Xilinx Design Constraints for the top-level design
  + **hls**: C++ design
  + **dsp**: SysGen design

Next create the working directory called **work** inside the root directory. It is not under revision control but is where you will run Vivado and store files temporarily before check-ins.

## Working in Shells

We will using the Windows command shell for these labs, this gives you the ability to:

* Run Vivado
* Run make
* Use Git commands for managing files under revision control

Note: If you like you can install GitHub for Windows which will install a Git Shell which you can use. An introduction to GitHub for Windows is included in the Prework document (VivadoRevCtrlPrework.docx) in the doc directory.

Open a Windows command shell and change to the directory you extracted the zip to and run the **env.bat** script to set up the environment to be able to run Vivado, make and Git commands. You will see some commands echoed to the shell (Figure 1).

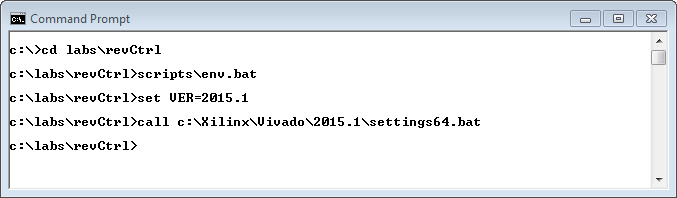


Figure 1: Setting up the shell environment

This adds the MinGW UNIX-like commands to your path so you can run the **make** utility to call Makefiles in the Windows shell. You can also run other common UNIX commands such ls, cp, rm, mv, and the UNIX find command within the Windows command shell.

After calling the script, test your setup by verifying that you can run Vivado, make utility and Git from the command line:

C:\labs\revCtrl>vivado -version

Vivado v2014.4 (64-bit)

SW Build 1071353 on Tue Nov 18 18:29:27 MST 2014

IP Build 1070531 on Tue Nov 18 01:10:18 MST 2014

Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.

C:\labs\revCtrl>make

make: \*\*\* No targets specified and no makefile found. Stop.

C:\labs\revCtrl>git --version

git version 1.9.5.msysgit.0

If you encounter problems, ask your instructor for assistance.

## About the Lab Questions

The lab questions intend to provoke thought and help enhance the learning experience. Although some space is provided you need not record written answers unless you wish. The answers are provided in a separate document.

# Lab 1 : Simple RTL Project

In this lab we will start with a very simple design **top** to see how revision control works with Vivado. Our overall goal is to determine the minimum set of files to regenerate top and place those files under revision control. Throughout the labs we will use the project-based flow for simplicity but the concepts can be extended to fit non-project flows.

## Lab Procedure

1. Navigate to the scripts directory and view the file **setup\_simple.tcl**.

**Question 1-1a**: What is the primary purpose of this script?

To verify your answer, change to the work directory and try running the script. At the command prompt run:

C:\labs\revCtrl\work>vivado –mode batch -source ..\scripts\setup\_simple.tcl

**Question 1-1b**: The script uses **add\_files** to refer to source files. What if **import\_files** was used instead, would it create any complications for revision control?

1. Inspect the contents of the script **compile.tcl** in the scripts directory.

**Question 1-2a**: What is the purpose of this script?

**Question 1-2b**: What **two** important files are generated by this script? (Hint: one is for **make**)

1. Next copy the makefile **Makefile\_simple** from the scripts directory into the work directory and rename it to **Makefile**.

**Question 1-3a**: What 4 targets are contained within the Makefile? Which targets require Vivado run scripts as rules to generate them?

**Question 1-3b**: Name a variable used in the makefile, and at least one advantage of using variables. What is the syntax for dereferencing a variable?

**Question 1-3c**: What is the first character of a line containing a makefile rule?

1. The entire build process can be managed using make since it describes the dependency relationships and the “rules” to make the targets.

**Question 1-4**: What are the dependencies for the compile target?

1. Let’s experiment with make and generate a few targets. Clean the working directory then make the **setup** target:

make clean

make setup

**Question 1-5a**: Note the compile target depends on the setup target. If we make the compile target next, will the setup target be generated first?

Next make the **compile** target, followed by **all**:

make compile

make all

**Question 1-5b**: Why does make report that “Nothing is to be done for ‘all?’”

1. Check the files into the Git repository. Change to the root directory of the lab (cd ..).
   1. Create the Git repository in the lab root folder using the git init command.
   2. Check the status using the git statuscommand. It should reflect a newly created repository with **Untracked files**.
   3. Check in the following directories: (never check in **work**):

git add dsp hdl hls scripts xdc

git commit -m “Initial checkin”

It is good to run git status after each step to ensure the project state is as expected. Once all has been checked in successfully, clean the work directory by making the clean target. Also remove the file **Makefile** since it is only used for Lab1.

**Question 1-6**: Bonus question: List the minimum set of files required to regenerate the top bitstream. These would also be the minimum files to be placed under revision control.

## Conclusion

This first lab illustrates the process of managing design files under revision control including:

1. Working with Tcl scripts to create and run Vivado projects.
2. Using a Makefile to run those scripts to build “targets.”
3. Creating and using a Git repository to keep recommended files under revision control.

We will build on the concepts learned in this lab to learn how to handle different types of design data under revision control.

# Lab 2: Simple IP Project

The goal of this lab is to become familiar with managing standalone IP from the IP Catalog under revision control. We generate the files for an AXI IIC interface IP using a scripted flow and place the results under revision control where they can be reused in other designs.

## Lab Procedure

1. Inspect the **ip.tcl** script inside the scripts directory. This script generates an IP from scratch.

**Question 2-1a**: What would be a good starting point, if you needed to generate such a script?

**Question 2-1b**: Which command chooses the IP to be configured?

**Question 2-1c**: Which command configures the IP with the user-desired settings?

**Question 2-1d**: In which directory will you find the results once IP generation is finished?

**Question 2-1e**: What is the purpose of the .ip.done file?

1. Change to the work directory and copy the file Makefile from the scripts to the work directory. This Makefile is used for the entire design and is therefore more complicated than the Makefile used in Lab1. View the Makefile to become more familiar with its structure and conventions.

**Question 2-2**: What is the command to make the AXI IIC IP target?

1. Make the IP target: make ip

**Question 2-3a**: Assuming that other designs wish to use the generated IP and its output products, what files are needed?

**Question 2-3b**: If only the IP source is reused but not the output products, then what files are needed?

1. Now we will place the generated IP and its output products under revision control. In the lab root directory you will see a directory called **ip**. In the **work** directory there should now be a directory also called **ip** which contains two sub-directories; **axi\_iic\_0** and **managed\_ip\_project**. To add the IP directory to our revision control you need to copy the **axi\_iic\_0** directory from the **work/ip/** directory to the **<labroot>/ip/** directory as our sources are preserved outside of our working directory. We do not copy the **managed\_ip\_directory** as it is not recommended to put Vivado projects into revision control systems.
2. Next, add the **ip** directory (and hence the IP directory inside it) in the lab root directory to revision control as you did in lab #1 step 6. Be sure to change back to the lab root directory first.
   1. First use git add to track the files.
   2. Next use git commit –m “your comment”to check in the files.
   3. Run git status to check the status of the repository. Note that the **work** directory should still be reported as **Untracked.**
3. Bonus points: Create a project to view the generated IP. Try reconfiguring the IP and regenerating the output products and note the Tcl commands that appear in the console.

## Conclusion

This lab extends the basic concepts of scripting and revision control to cover generated IP.

# Lab 3: Custom IP Project

The goal of this lab is to become familiar with managing custom IP under revision control. Custom IP is different from IP Catalog with a different set of files to manage. We package the familiar bft example design as an IP and place the results under revision control. The IP can then be added to the project IP repositories to be reused in other designs.

## Lab Procedure

1. Inspect the **cip.tcl** script inside the scripts directory. This script packages bft as a custom IP.

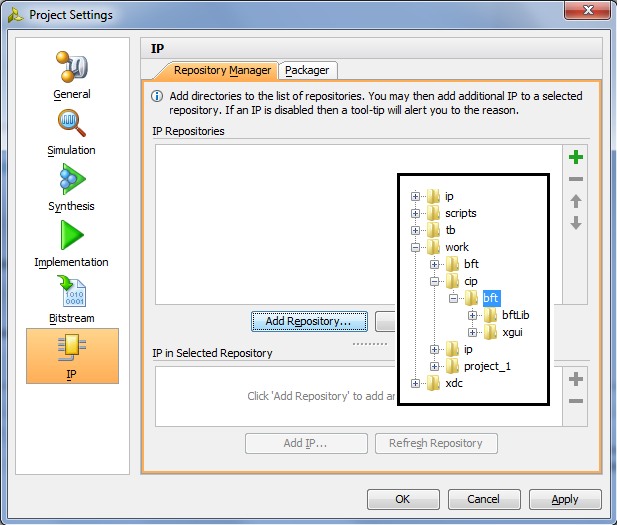
Question 3-1a: What is the purpose of copying the bft source structure?

Question 3-1b: What is the command that packages the project as an IP?

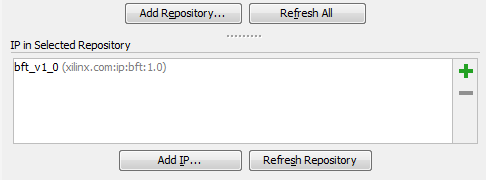
1. Use make to generate the Custom IP target in the work directory.

Question 3-2: What are the three types of files needed to reuse a custom IP?

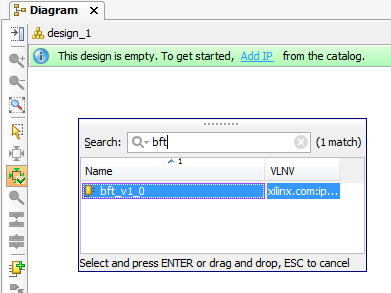
1. Later we plan to add this IP to a block design. Test the generated IP by adding it to a project.
   1. In the work directory, create a new Vivado project, project\_1.
   2. Choose an RTL project with no sources and target the device xc7z020clg484-1.
   3. Open IP Catalog, right-click and choose **IP Settings**. On the Repository Manager, choose **Add Repository** and browse to the location of the bft custom IP:



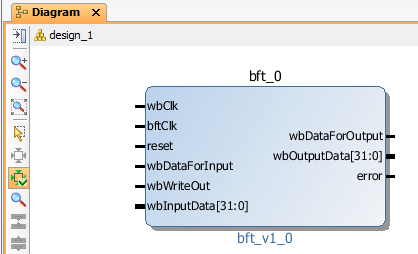
* 1. After selecting bft, the bft IP should be visible under “IP in Selected Repository”



* 1. Create a new block design and add the bft from the catalog by searching for it:



* 1. This should place the bft symbol in the block design where it can be connected to other IP.



You may exit the project without saving.

1. Finally place the files under revision control:
   1. Copy the custom IP into a new folder **cip** which is at the same level as other sources such as hdl and xdc. The **bft** directory containing the custom IP files should reside inside the **cip** directory.
   2. Check in the necessary files for the bft custom IP into the repository using git add, and git commit. Review status with git status.

## Conclusion

This lab extends the basic concepts of scripting and revision control to cover a custom IP built from the bft HDL example design.

# Lab 4: HLS-Based IP

The goal of this lab is to become familiar with packaging an HLS project into a custom IP **rgb\_mux** and placing it under revision control. The IP generation includes scripting the Vivado HLS run to generate it from the C++ source files. The IP can then be added to the project IP repositories to be reused in other designs.

## Lab Procedure

1. Inspect the **hls.tcl** script inside the scripts directory. This script synthesizes rgb\_mux and packages it as a custom IP.

Question 4-1a: What program is used to run this script?

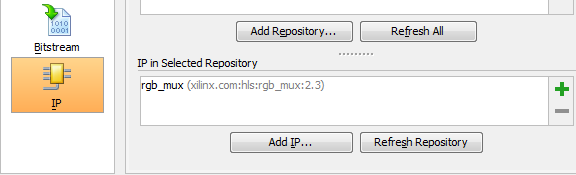
Question 4-1b: What command packages the design as a custom IP?

Question 4-1c: What is a good starting point for creating this script, particularly if you are not familiar with Vivado HLS? (Hint: check the contents of the **hls** directory).

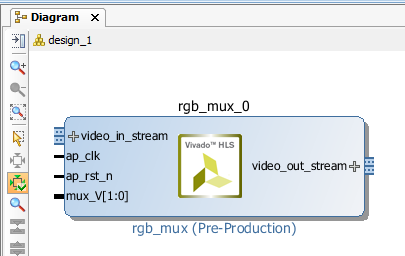
1. Use make to generate rgb\_mux in the work directory. Upon completion, locate the rgb\_mux custom IP.

Question 4-2: In which directory are the custom IP files located, relative from **work**?

1. We plan to also use this IP in a block design. Using the same procedure in Lab 3, Step 3, add the custom IP to a test project such as the previously created project **project\_1**. You should see the rgb\_mux IP in the newly added repository.



And you should be able to instantiate the IP in a block design:



1. Finally place the files under revision control:
   1. The rgb\_mux is a custom IP so it will be stored in the **cip** directory next to **bft**. Create a new directory inside cip called **rgb\_mux** and place its custom IP files into this directory. It should also be structurally similar to bft so that the component.xml file and xgui directory are found inside rgb\_mux.
   2. Check in the necessary files for the rgb\_mux custom IP into the repository using git add, and git commit. Review status with git status.

## Conclusion

This lab illustrates that creating custom IP from HLS is a straightforward process that can be entirely scripted and generated using make, and the results managed under revision control for reuse in other designs.

# Lab 5: IPI Block Design Project

In this lab we will focus on an IPI block design: automating the generation of the design **zynq\_bd** using scripts and placing the needed files under revision control. The IPI block design reuses the IP created in previous labs as well as other IP from the IP Catalog. Similar to an IP, output products are generated for the block design so that it can be reused in other designs.

## Lab Procedure

1. The block design generation may take a bit longer to run than the previous labs. First use the makefile to generate the block design then proceed with the next steps while Vivado runs in the background. Also if your working directory is becoming cluttered, you may want to clean it.
2. Familiarize yourself with the block design scripts. First review the **bd\_gen.tcl** script which generates the zynq\_bd block design that can be reused in another design. The block design itself includes many IP blocks including the IP generated in previous labs.

Question 5-2a: How are the custom IPs made available for the block design project?

Question 5-2b: How is the block design built? (This may require a bit of digging.)

* What command is used to add IP blocks including the bft and rgb\_mux?
* What command is used to connect IP blocks?

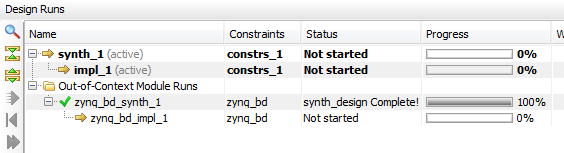
Question 5-2c: Which commands package the block design and allow it to be reused, similar to IP?

1. After the block design is generated, check the working directory. There are two new directories created in the process: **zynq** and **zynq\_bd**. Inspect the contents of each directory.

Question 5-3 If we want to reuse the block design, which directory contents should be kept under revision control: zynq, zynq\_bd, or both?

1. Open the **zynq** project which was created during the block design generation process.

Question 5-4: Why is there no top design for this project? (Hint: view the Design Runs)



1. Open the zynq\_bd block design and view the contents. Next run write\_bd\_tcl block.tcl and view the block.tcl file contents.

Question 5-5: What purpose does this file serve in a scripted flow environment?

1. Finally place the files under revision control:
   1. The **zynq\_bd** block design will be stored in a new directory **bd**. Create a new directory **bd** and copy the entire zynq\_bd directory from the working directory into bd, so that the block design source file **zynq\_bd.bd** is located in **bd/zynq\_bd**.
   2. Check in the necessary files for the block design into the repository using git add, and git commit. Review status with git status.

## Conclusion

This lab demonstrates the automated process for creating a block design and placing the contents under revision control. The block design is more complex than the IPs from the previous labs as it contains many IP itself, and requires a block design description. Output products are generated for the block design in a manner similar to that of IP, and the output products can be placed under revision control to be reused in other designs.

# Lab 6: Top Level Integration

This final lab progresses further in complexity beyond the IPI block design. The design **top** contains:

* The zynq\_bd block design which itself contains the IP and custom IP from the previous labs
* Several Verilog RTL modules including the top design
* A SysGen design generated standalone and integrated as a DCP

Similar to the previous labs we will automate the generation of the top design and place necessary files under revision control.

## Lab Procedure

1. The entire top design generation is automated using Vivado scripts and the makefile. Open the Makefile for editing.

Question 6-1a: What targets are used for generating the top design?

Question 6-1b Find the variable REUSEGOLDEN. What is the difference in makefile behavior between a TRUE value and a FALSE value?

Question 6-1c: How does setup.tcl use the tclargs **reuseGolden** variable?

1. Change the value of REUSEGOLDEN to TRUE. Then clean the work directory and make the top design. (This may run for a few minutes.)

Question 6-2a: What value of REUSEGOLDEN results in potentially faster compile time?

Question 6-2b: What value of REUSEGOLDEN requires fewer files to be placed under revision control?

1. Once the top generation is complete, you may open the top project and view the results.

Question 6-3: If we cleaned the working directory, then set REUSEGOLDEN to false and made the top design, what would you expect to see in the work directory?

If you have time you may try setting the REUSEGOLDEN variable to FALSE and see in detail how the make process differs from the previous run.

This is the final step for the lab so there is nothing further to place into the Git repository.

## Conclusion

This lab covers the design file types most likely to be encountered in real designs and illustrates how the entire flow can be managed to bitstream generation with Vivado scripts and make. Variables can be used to control the make process as we saw with the different values of REUSEGOLDEN and the tclargs option. One can choose to regenerate as much of the design as possible or to reuse as much as possible.

These concepts are applicable to all types of designs created using Vivado, and other design steps such as simulation. Although we used Git for revision control, the general strategies apply equally to other revision control systems which have similar features.