Lab Instructions: Revision Control

Contents

[Introduction 2](#_Toc395717025)

[Lab Files 2](#_Toc395717026)

[Files Provided with the Labs 2](#_Toc395717027)

[Where to put the lab files 2](#_Toc395717028)

[Lab 1 : RTL Project Scripts 3](#_Toc395717029)

[Working in shells 3](#_Toc395717030)

[Lab Setup 4](#_Toc395717031)

[Lab Procedure 5](#_Toc395717032)

[Conclusion 6](#_Toc395717033)

[Lab 2 Managed IP 6](#_Toc395717034)

[Lab file 6](#_Toc395717035)

[Lab Procedure 7](#_Toc395717036)

[Conclusion 9](#_Toc395717037)

[Lab 3: IPI Block Design 9](#_Toc395717038)

[Lab files 10](#_Toc395717039)

[Lab Procedure 10](#_Toc395717040)

[Summary 12](#_Toc395717041)

[Lab 4: Packaged Custom RTL IP 12](#_Toc395717042)

[Lab Files 12](#_Toc395717043)

[Lab Procedure 13](#_Toc395717044)

[Summary 15](#_Toc395717045)

[Lab 5: HLS-Based Packaged IP 15](#_Toc395717046)

[Lab Files 16](#_Toc395717047)

[Lab Procedure 16](#_Toc395717048)

[Summary 18](#_Toc395717049)

[Lab 6: System Generator DSP Subsystem (under constr) 19](#_Toc395717050)

[Lab Procedure 19](#_Toc395717051)

# Introduction

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

* Lab 1: RTL project
* Lab 2: IP project
* Lab 3: IPI project
* Lab 4: Custom IP project
* Lab 5: HLS-based IP
* Lab 6: !!! Needs Updated !!!

We will be using the following tools for these labs:

* Git revision control
* MinGW make utility - included with Vivado System Edition
* Vivado 2014.2 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 essential to the labs, the labs will all use the same working directory and revision control repository.

### Files Provided with the Labs

The following files are provided with the labs, arranged in these directories:

* doc: lab instructions and presentation
* lab\_files: Use these as starting points or templates and complete them as necessary during the labs. Inside lab\_files are directories lab1, lab2, lab3, and lab? which contain the files for each lab.
* lab\_solutions: These contain completed versions of the files modified during the labs. Please do not jump directly to the lab\_solutions without first trying the labs. Notify your instructor if any lab instructions are unclear.

### Where to put the lab files

First create a “root” location where you will store the lab source files and working directory. For example:

* C:\tsc14\revision\_control\_labs
* C:\users\your\_username\revision\_control\_labs

This is where you will create the Git repository and where you will put the directories for the different types of lab files. Copy the files from lab\_files into this root directory. Example: lab1 contains these directories, and some of those directories contain files:

* lab1
  + hdl
  + scripts
  + xdc
  + work

Copy the lab1 contents (not lab1 itself) into your root location, for example C:\tsc14\revision\_control\_labs. Then when you begin Lab 1, your directory structure looks like this:

* C:\tsc14\revision\_control\_labs
  + hdl
  + scripts
  + xdc
  + work

Subsequent labs contain files that you add to these directories when you begin the lab. The lab2 directory contains a single file:

lab2\**scripts\ip.tcl**

so to begin lab2, you copy ip.tcl to:

C:\tsc14\revision\_control\_labs\**scripts\ip.tcl**

Do not overwrite the scripts directory with lab2\scripts. Subsequent labs follow similar patterns.

# Lab 1 : RTL Project Scripts

In this lab we will start with a very basic set of files to get an introduction to the basic processes involved in creating a project script for Vivado and maintaining files under revision control. This includes creating:

1. Vivado run scripts to create and run projects
2. A Makefile that calls the run scripts to build projects
3. A Git repository that keeps files under revision control

Please review the introductions to Git and make if they are not familiar to you.

## Working in shells

We will use two different command shells for the labs:

* Windows command shell - to run Vivado and make
* Git bash shell - for everything else

Upon opening a Windows command shell, use the **env.bat** script to set up Vivado to run in the shell:

call <path to scripts>\env.bat

This also 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.

Test the make command. In the Windows command shell, run **make test**. It should fail with a message similar to this:



If you get a different message or experience some other difficulty, ask your instructor for assistance.

To launch the Git shell, double-click the icon as described earlier during the Git introduction:



The Git shell functions just like a bash shell and is pre-configured to run git commands. On Windows the pathname to the **C:\** drive is **/c/**, so for example the folder **C:\tsc14** is accessed using **/c/tsc14**.

## Lab Setup

As described in the introduction, we will start with a few files in a simple directory structure and add to it as the design grows with each successive lab. In the lab1 directory you’ll find these files to begin:

* work
* hdl
  + top
    - top.v
  + threeFlop
    - threeFlop.v
* xdc
  + top.xdc
* scripts
  + setup.tcl
  + Makefile
  + env.bat
  + utils.tcl

Copy these files to your root directory, for example **C:\tsc14\revision\_control\_labs**. Inside root we will place the Git repository and the **work** directory will store intermediate Vivado results. The directory hierarchy should now resemble:

* (root directory)
  + hdl
  + xdc
  + scripts
  + work

When working on files in this lab, it is recommended to keep script files in the **script** directory and test them by running vivado or make in the **work** directory. To do this copy the Makefile into the work directory for editing and test it by calling **make** on the command line. When finished the Makefile can be checked back in to the scripts directory.

## Lab Procedure

1. Complete the Tcl script named **setup.tcl** to recreate a Vivado project for the **threeFlop** design, using the commented hints inside the file. Recall the ways to create a project script such as:
   * Using write\_project\_tcl
   * Creating a project using the GUI and capturing .jou commands or command history into a Tcl script.

Make sure:

* 1. The touch proc is called at the end of the script to create a Makefile target - This tells **make** that the setup script ran successfully. If setup fails then no target file is generated.
  2. The sources are referenced remotely, not added to the project.
  3. When testing the script, exit any active Vivado sessions, run Vivado in batch mode in the **work** directory, and source the script.

1. Create a new Tcl script called **compile.tcl** in the **scripts** directory.
   1. It should open the **project** from the previous setup step and generate a bitstream. Hints:
      1. Use the project Tcl commands for launching and waiting on runs.
      2. There is a command option for generating a bitstream after implementation.
   2. Use the touch proc to create a target file **.compile.done** similar to the target file used for the setup target.
   3. Test the script from the **work** directory to ensure it creates the Vivado project and Makefile target as expected.
2. Complete the Makefile to build a bitstream using setup and compile targets.
   1. The setup target has been completed as an example.
   2. Add a target **compile** that calls compile.tcl to generate the bitstream.
   3. Review the **clean** target to make sure you understand the syntax. Questions to ask yourself may include:
      1. What files are removed and what files are not removed?
      2. What happens to the Makefile target files such as .setup.done?
   4. Run **make clean** to begin with a clean work directory.
   5. Test the script to ensure it creates the bitstream and Makefile target as expected. Run **make compile** to check that the setup target is completed first. The setup target must run first to generate the project, otherwise compile will fail.
3. Check the files into the Git repository.
   1. Create the Git repository in the root folder using the **git init** command.
   2. Check the status using the **git status** command. It should reflect a newly created repository with **Untracked files**.
   3. Check in the **hdl**, **scripts**, and **xdc** directories, but not work. We will not save the bitstream for this lab. The commands to use are:
      1. **git add <filename(s) or directory name(s)>**  to stage files for a commit.
      2. **git commit -m “Comments on the checkin”** (add comments of your choice)

It is good to run **git status** after each step to ensure the project state is as expected.

1. Once all has been checked in successfully, clean the work directory using **make clean**.

## Conclusion

This first lab has taken you through the complete process of:

1. Creating Tcl scripts to create and run Vivado projects.
2. Preparing and calling a Makefile to run those scripts to build projects.
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 Managed IP

This lab covers the process of generating a simple managed IP AXI IIC and placing it under revision control. The lab objectives include:

* Creating a Managed IP project for AXI IIC.
* Create a script to fully generate the IP with OOC flow.
* Update the Git repository with the IP products and updated scripts into revision control.
* Iterate and change an IP customization option and update the Git repository as needed.

## Lab file

These files are provided to get started:

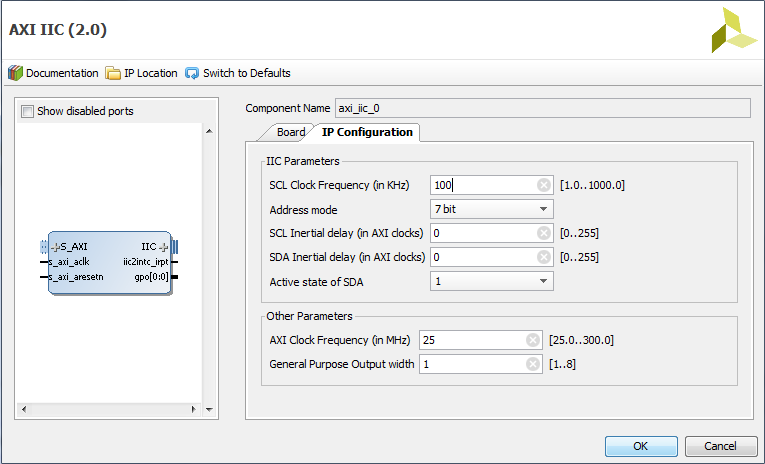
* scripts/ip.tcl: This is a template file where you will add the Tcl required to build the IP from scratch.
* hdl/top/top.v: This is a new top level that contains the AXI IIC IP and other logic.
* hdl/top/iicWrapper.v: This is the Verilog wrapper around the AXI IIC IP.
* hdl/top/shift.v: Shift register logic used inside the IP wrapper.
* xdc/top.xdc: An updated XDC constraints file.

We will edit the ip.tcl script to generate the IP from scratch, updating the Makefile to include the IP as a target, and checking the IP output products into the ip directory. Assuming that you are unfamiliar with the Tcl commands required to build IP using scripts, we will use the GUI which will reveal the equivalent Tcl commands.

In the later portions of the lab, we will update our design environment to add the IP to the new top level design.

## Lab Procedure

1. Copy the lab file **scripts/ip.tcl** to your scripts directory.
2. Create a new directory called **ip** in the root directory, same level as **hdl**. This is where we will store the AXI IIC IP output products.
3. Launch Vivado in the work directory and create a new Managed IP project. The IP to be created is **AXI IIC**.
   1. Choose the ZC702 as the target board (part xc7z020clg484-1)
   2. The target language is Verilog.
   3. Use the default settings for the IP as shown below. The component name can be left as is: **axi\_iic\_0**.
   4. When prompted, **Generate** the output products and move the synthesis run to the Background.



1. Complete the **ip.tcl** Tcl script that recreates the IP from scratch, all the way to output product generation.
   1. With synthesis in the background, review the commands used in the IP generation in the Tcl console.
   2. In the script, note the suggestion of storing results locally in the **ip** directory, referenced by the ipDir variable. Translate the absolute paths in the Tcl commands to instead use the ipDir variable to point to the local directory **ip** which will be created inside of **work**. Be sure to update all the directories and IP filenames. This enhances the portability of scripts so that the IP can be recreated in a different location in the future.
   3. Make sure the utils.tcl script is sourced and use the touch proc to create a file **.ip.done** that can be used for a Makefile target similar to **compile** and **setup**.
   4. Since ip.tcl may be called in a sequence with other scripts, remember to wait on a run before creating .ip.done.
   5. When finished, exit any active Vivado sessions.
2. Add the IP target to the Makefile:
   1. If necessary copy the Makefile to the work directory.
   2. Add the new target called **ip** that regenerates the axi\_iic\_0 output products from scratch. It should depend on the Makefile target **.ip.done**.
   3. Add the rule to create **.ip.done**. Assume that the IP will be generated unconditionally when the **ip** target is made.
   4. The other targets remain unchanged for this lab.
3. Test both the Makefile and ip.tcl by making the **clean** target followed by **ip**. If you encounter errors, revise as necessary or ask for assistance.
4. Check that the IP output products are in the new directory **ip** which should be located in **work**. The **ip** directory should contain the **axi\_iic\_0** directory for the generated output products.
5. If satisfied that the necessary output products were generated, copy the necessary files to the new directory **ip** which is in the root directory, at the same level as **work**. Recall the recommendation of files to check in for Managed IP: the IP directory and everything underneath which would be the **axi\_iic\_0** directory. Other files such as the managed IP project are a byproduct of IP generation and are not relevant for revision control.
6. Check in the IP, the Makefile and the new script:
   1. Use **git status** to check the project status.
   2. Use **git add** to add the ip directory contents and changed files.
   3. Use **git status** to check the project status again.
   4. Use **git commi**t to check in the files.
7. Once the IP is under revision control, make a change. Change the AXI Clock Frequency from 25 MHz to **100 MHz**. This can be done by either using the GUI to recustomize the IP or if you are comfortable with Tcl, by modifying the appropriate IP property which is **CONFIG.AXI\_ACLK\_FREQ\_MHZ**.



1. Incorporate this change in the IP. This requires an update to **ip.tcl**.
2. Close any open Vivado sessions and use the Makefile to regenerate the IP output products. Note that it may be best to clean the work directory before **make ip**.
3. Once the IP output products are regenerated, copy the necessary files to the **ip** directory. Keep in mind that files are under revision control at this point. Overwrite the entire **axi\_iic\_0** directory with the new version.
4. Run **git status**. It should indicate that the **axi\_iic\_0** and **ip.tcl** files are modified.
5. Run **git add** to stage these modified files for checkin.
6. Run **git status** again to check that the files are staged, and they should be displayed in green.
7. Run **git commit** to check in the IP update.
8. Once all has been checked in successfully, clean the work directory.

## Bonus Lab Procedure

In the next part of this lab we will use the IP in an updated version of the top design.

**NOTE**: if you do not have time to complete this section, perform these steps before continuing to Lab 3:

* Copy the lab\_solutions/lab2 contents into your root directory to OVERWRITE all files.
* Run **git add .** in your root directory and **git commit** to check in all files to bring your repository up-to-date.

1. Copy the necessary lab files:
   1. Copy the **hdl/top** to your hdl directory to OVERWRITE the existing **hdl/top**.
   2. Copy **xdc/top.xdc** to your xdc directory to OVERWRITE the existing **xdc/top.xdc.**
2. Update the **setup.tcl** script. Our project now consists of a couple more HDL files and an IP.
   1. Add the missing hdl files.
   2. Add the AXI IIC **.xci** file as a remote source.
3. Review the **compile.tcl** script. Does it require any changes?
4. Update the **Makefile**. There are a few updates to consider:
   1. The dependencies for the **setup** target have changed.
      1. Update the **RTL** variable to reflect the new HDL files.
      2. The project now depends on the .xci file. Add that dependency and for better readability consider using a new **IP** variable.
   2. Does **clean** need to be modified?
5. Test the Makefile in stages from the work directory by making the different targets:
   1. First run **make clean** and ensure all intermediate Vivado results are gone.
   2. Run make **setup** and verify that the project is created as expected.
   3. Run make **compile** and verify that it generates a bitstream.
6. Run **git status** to see what has changed since the last checkin.
   1. The new source files should be reported as **Untracked.**
   2. The modified scripts should be reported as **Modified.**
   3. Use **git add** and **git commit** to check in the changes.
   4. Verify that the repository is up-to-date.

## Conclusion

In this lab you have covered how to manage IP under revision control. You should now be comfortable performing the following:

* Creating and modifying Tcl scripts to fully generate the IP from scratch.
* Using a Makefile to build IP.
* Checking in IP files under revision control with Git.
* Modifying IP and updating the Git repository.

# Lab 3: IPI Block Design

This lab focuses on block designs from IP Integrator. In terms of revision control, block designs are somewhat similar to IP as they have associated output products. Block design management may also include recreating the block design itself from scratch. In this lab we will cover:

1. Use of write\_bd\_tcl to generate a Tcl script to recreate a block design.
2. Creating a script to generate the block design output products.
3. Using a Makefile to build block design output products from scratch.
4. Checking in the block design under revision control.
5. Instantiating the block design in another block design.
6. Revising and repackaging the IP

## Lab files

The lab files contain the following new files to get started:

* scripts/bd\_gen.tcl: This is a template file where you will add the Tcl required to build the block design from scratch.
* zynq\_bd\_project.xpr.zip : contains a project **zynq.xpr** with the block design used for the lab
* hdl/top/top.v : This is a new top level design that instantiates the block design.
* xdc/top.xdc: This is an updated XDC constraints file.
* xdc/top\_io.xdc : This XDC file contains the IOSTD and PACKAGE\_PIN constraints for generating a bitstream.

This lab is a bit more complex than previous labs. This lab sets up a scenario where a customer has a block design inside a Vivado project that they would like to reuse in other block designs. Therefore as a source, the block design must be placed under revision control. The original block design is a local source in a project in zynq\_bd\_project.xpr. The sequence of updating the block design is:

* The block design is maintained and updated in IP Integrator in a Vivado project zynq.xpr.
* The block design is saved and packaged into a separate source directory called **bd** which is used to store block designs. (Recall the recommendation for block designs is to check in the directory containing the .bd file: that directory and all files and directories underneath are placed under revision control).
* Other designs instantiate the block design which implies that those designs depend on the corresponding .bd file.
* When the block design is updated and saved, the .bd file changes and the entire block design package must be regenerated from scratch and checked back into the **bd** directory.

## Lab Procedure

1. Copy the lab files to their proper locations:
   1. Copy **zynq\_bd\_project.xpr.zip** into the **work** directory. This contains the block design we wish to extract.
   2. Copy **scripts/bd\_gen.tcl** into the **scripts** directory.
2. In work, unzip the archive, and open the project. Then open the block design, which should resemble the following:



1. Use **write\_bd\_tcl** to write a script that creates the block design from scratch. Name the file **bd.tcl** and place it in the scripts directory. Edit the bd.tcl script to change the create\_bd\_design options to include **-dir .** so the necessary project files are deposited into the current working directory ‘.’

**create\_bd\_design -dir . $design\_name**

1. Update the Tcl script **bd\_gen.tcl** in the **scripts** directory. This generates the block design output products, similar to IP generation. Some hints:
   1. Note that a project is required to hold the generated block design, although the project itself may not be useful for revision control.
   2. The target part is that of the ZC702: xc7z020clg484-1.
   3. Check what Tcl commands are issued when launching **Generate Block Design** in IP Integrator.
   4. Replace absolute file pathnames to increase script portability.
   5. Ensure the Makefile target **.bd.done** is the final step.
   6. Test the script by exiting any interactive Vivado sessions and running vivado in batch mode, sourcing the script. Verify that it generates the expected output.
      1. The output products are located in the project srcs directory.
      2. Recall the files to check in: the directory containing the .bd file, and all files underneath.
2. Next update the Makefile. If necessary copy the Makefile from scripts to work.
   1. Include a target **bd\_gen** that results in the entire block design being generated from scratch. Similar to other targets, **bd\_gen** is dependent on **.bd.done.**
   2. In this lab we will assume that a final block design has been handed to us, so we do not need to monitor the original zynq\_bd\_project as a dependency. But we may choose to modify the block design in the future as it is now a self-contained package. In the Makefile however the bd\_gen target should just be generated unconditionally when made.
3. Test the Makefile:
   1. First **make clean**. Note that this may remove the zynq project which contains the block design so it is important to make sure the script runs successfully.
   2. Next make **bd\_gen**.
4. Create a new folder **bd** in the root directory, at the same level as the other source directories. Copy the bd output products into **bd** from work. This should be the entire **zynq\_bd** directory.
5. Run **git status** to see what files have been added or modified since the last checkin. This should include:
   1. The bd directory and the zynq\_bd block design.
   2. In scripts: bd\_gen.tcl, bd.tcl, and Makefile.
   3. The original zynq project as well as work contents can be discarded.
6. Check in the necessary files using **git add** and **git commit.** At each step run **git status** to ensure the correct files are staged and checked in.
7. Once all has been checked in successfully, clean the work directory.

## Bonus Lab Procedure

In the next part of this lab we will use the block design in an updated version of the top design.

**NOTE**: if you do not have time to complete this section, perform these steps before continuing to Lab 4:

* Copy the lab\_solutions/lab3 contents into your root directory to OVERWRITE all files.
* Run **git add .** in your root directory and **git commit** to check in all files to bring your repository up-to-date.

1. Copy the necessary lab files for this part of the lab:
   1. hdl/top/top.v : This is a new top level design that instantiates the block design.
   2. xdc/top.xdc: This is an updated XDC constraints file.
   3. xdc/top\_io.xdc : This XDC file contains the IOSTD and PACKAGE\_PIN constraints for generating a bitstream.
2. Update the **setup.tcl** file to set up the project.
   1. Add the **.bd** file as a remote source.
   2. Add the new **top\_io.xdc** constraints file.
3. Similar to the previous lab, note that **compile.tcl** does not require any changes.
4. Update the **Makefile**. The **setup** target now depends on the block design.
   1. Add a new variable **BD** for the block design file.
   2. Use the BD variable as a dependency for the **.setup.done** target.
5. Test the Makefile:
   1. Generate the **clean** target.
   2. Generate the **compile** target and verify that it results in a bitstream.
6. Run **git status** to see what has changed.
   1. Check in the new hdl sources.
   2. Check in the new and updated XDC constraints.
   3. Check in the new and updated scripts.
   4. Check in the new bd directory.

## Summary

This lab demonstrates how IP Integrator Block Designs can be managed using revision control. Some key points to remember:

* A block design .bd file can be generated using the write\_bd\_tcl command.
* The generate\_target command generates block design output products from a .bd file, similar to Managed IP.
* A block design can be extracted from an existing project, placed under revision control, and used in other designs.

# Lab 4: Packaged Custom RTL IP

This lab covers RTL that is packaged into a custom IP for the IP Catalog. We will package the familiar bft example design. We will cover:

1. Creating a script to package the RTL project from scratch.
2. Placing the necessary packaged IP files under revision control.
3. Revising the IP within IP Integrator, re-packaging, and checking in the updated IP.

## Lab Files

The lab files contain the following new files to get started:

* scripts/bft\_cip.tcl: This is a template file where you will add the Tcl required to build the packaged IP from scratch.
* scripts/Makefile: The Makefile has been updated with a hint for setting up the relevant dependencies.
* hdl/bft: Contains the RTL source for the IP

At a high level, the procedure for packaging IP involves:

* Choosing a directory for the IP Definition.
* Copying all necessary source files under the IP Definition directory, so that the sources are kept with the packaged IP.
* Creating a project and adding the sources from the IP Definition directory.
* Configuring IP packaging settings such as version info.
* Generating the packaged IP.
* Creating scripts and Makefiles that can generate the IP from scratch.

## Lab Procedure

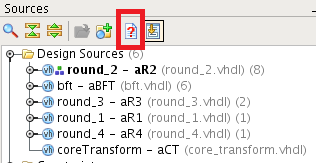
The goal is to create a script that can generate IP from scratch. It is easiest to work through the steps in the GUI and review the command history after a successful packaging run.

1. Copy the **bft\_cip.tcl** and **Makefile** files to your local **scripts** directory and copy the **bft** directory to your local **hdl** directory.
2. Set up the HDL source files to be packaged with the IP.
   1. Inside **work**, create a directory called **cip** which will be used to store Custom IP.
   2. Inside the **cip** directory, create a directory **bft** to be used as the IP Definition directory for the bft custom IP.
   3. Inside **bft** create a directory **hdl** to store the bft source files.
   4. Copy the **bft** directory ***contents*** from **hdl** to **cip/bft/hdl**. From work:

**$ cp -r ../hdl/bft/\* cip/bft/hdl**

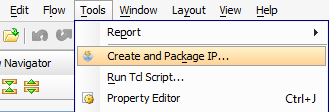
This ensures that the source files will be packaged with the IP.

1. Begin with a clean work directory, launch Vivado, and create a new RTL project:
   1. Project name is not important.
   2. Target is the ZC702 board (xc7z020clg484-1 device)
   3. Add the bft source files from the **cip/bft/hdl** directory in work (don’t import).
   4. The design top is **bft**.
2. Before packaging, the RTL must synthesize correctly. Sometimes you may be given RTL from a customer that doesn’t synthesize cleanly out-of-the-box, as this design demonstrates. If you do not want to try to figure out and fix the problem with this design, then skip to the next step, otherwise the following hints may help.
   1. Notice the active icon in the sources toolbar, it gives a clue that instances are missing, as would the **report\_compile\_order** command:



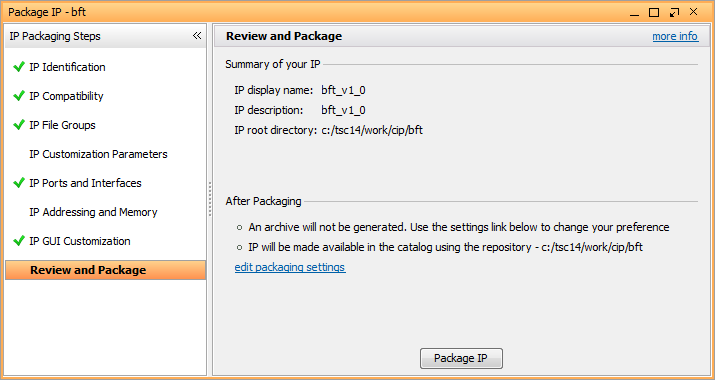
* 1. The design should successfully complete elaboration or **synth\_design -rtl** but this design gives errors.
  2. It involves VHDL libraries. Check the Compile Order and the library settings of the VHDL files.

1. Make sure the Library of each of the VHDL files in the bftLib directory is set to **bftLib**.
2. Verify that the design is able to elaborate successfully.
3. To package the design into an IP, launch the Create and Package IP Wizard from the Tools menu.



Step through the wizard and:

* 1. Choose to package your current project.
  2. Verify that the correct directory is chosen for the IP Definition (…work/cip/bft).
  3. This design has no .xci or IP generated files.
  4. Finish to launch the Package IP window where the IP can be further configured.
  5. Keep all default settings. Skip to the last IP Packaging step which is **Review and Package** select **Package IP** at the bottom:



1. After Generating the IP, check the contents of the **cip/bft** directory for the files to be placed under revision control. This should include:
   1. the **hdl** directory containing HDL files
   2. the **xgui** directory
   3. the **component.xml** file

Note that the Vivado project is a byproduct of IP generation and not needed for revision control.

1. Complete the script **bft\_cip.tcl** to generate the IP from scratch.
   1. Review the first completed parts of the script that include sourcing **utils.tcl** and copying the bft hdl files into the appropriate locations.
   2. The next part of the script should include creating and setting up the project. You can use write\_project\_tcl but that may be overkill for this lab example. It may be easier to just check to commands used to create the project in the Tcl Console or vivado.jou file.
   3. It is not necessary to launch any runs.
   4. The remainder of the script up to the generation of **.bft\_cip.done** includes the commands to package the IP. Notice some of these commands are in the **ipx** namespace. Strictly speaking, we only need to include the single command to package the project. Since our goal is only to generate the IP from a script, we do not need the other commands to configure the IP Catalog to use the IP.
2. Edit the Makefile and add the **cip** target. It follows the same structure as the other targets, except for the dependency. The **BFT\_SRC** variable in the Makefile has already been defined, and can be used as the dependency for **.bft\_cip.done**.
3. Copy the Makefile to **work** and test it.
   1. Begin with **make clean**.
   2. Run **make cip** and verify that the expected files were generated in the work directory.
4. Relocate the entire **cip** directory to the root directory, such that it resides at the same level as other source directories. The cip directory should contain the **bft** directory which contains everything needed to place the IP under revision control.
5. Run **git status** to see what has changed since the last checkin.
   1. The Makefile should be reported as modified.
   2. There are many new untracked files:
      1. scripts/bft\_cip.tcl
      2. The new **cip** directory
      3. hdl/bft
6. Check in the new files and verify the repository status is up to date.
7. Clean the work directory.

## Summary

This lab demonstrates how Custom IP can be managed using revision control. Some key points to remember:

* The IP Definition directory is where the packaged IP is stored.
* For packaged IP, the source files must be placed in the IP Definition directory.

# Lab 5: HLS-Based Packaged IP

This lab covers the generation of an IP from an HLS design. We will cover the process of generating a packaged IP from and HLS source which includes:

1. Generating IP output products from Vivado HLS.
2. Creating a script and Makefile to generate the IP from scratch.
3. Placing the appropriate files under revision control.

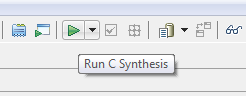
## Lab Files

The lab files contain the following new files to get started:

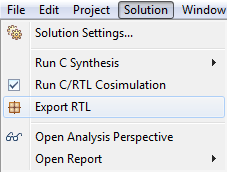
* hls/rgb\_mux.cpp : This is the C++ source file for the HLS-based IP
* hls/rgb\_mux.h: This is the C++ header file required by the C++ source.

## Lab Procedure

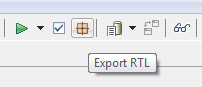
1. Copy the hls directory to the root directory, at the same level as the other source directories.
2. Launch Vivado HLS and create a new Vivado HLS project with the following settings:
   1. Project name: **rgb\_mux**
   2. Location: the **work** directory
   3. Top function: **rgb\_mux**
   4. Add file **rgb\_mux.cpp**, no testbench (Note: the header file is not added, it is included automatically).
   5. Solution Name: **solution\_zc702**
   6. Clock Period: **5**, Uncertainty: **1**
   7. Choose the ZC702 board as a target.
3. Click the Run button on the toolbar to run C Synthesis.



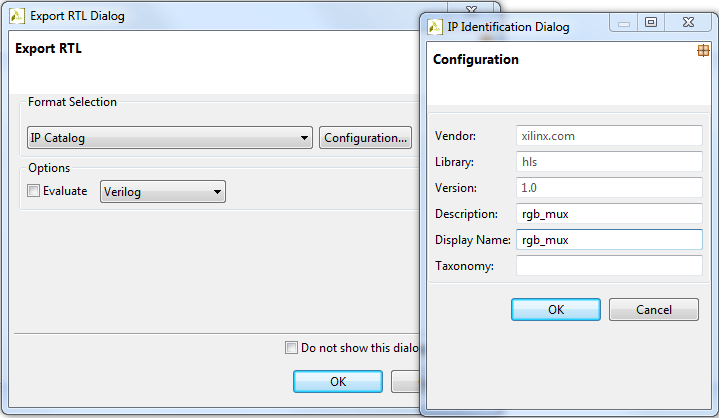
1. After C synthesis completes, export the RTL using Solution -> Export RTL



or by clicking the equivalent toolbar button



1. When prompted, choose to export to IP Catalog and select **Configuration…** The configuration should include updating the **Description** and **Display Name** values to **rgb\_mux**



Once IP generation is complete, the next step is to locate the important pieces generated by Vivado HLS to be placed under revision control:

* + A run script that generates the packaged IP from scratch from the source files.
  + The generated IP output products.

1. IP generation script: Vivado HLS automatically generates a script in the solution directory with the name **script.tcl**. Locate this file in the work directory and use it as a starting point to generate **hls.tcl**.
   1. Copy **script.tcl** to scripts with the file name **hls.tcl** and update hls.tcl.
   2. Note the HLS script uses relative pathnames. Also note that it stores synthesis **directives** in **directives.tcl** and sources that file before running C Synthesis. Although there are no directives in this design, it is good practice to include this file under revision control so that it is not overlooked when HLS directives are used.
      1. Locate the directives.tcl file and copy it to the **scripts** directory as **hls\_directives.tcl**.
      2. Update the reference to the file in hls.tcl to reflect its new location and file name.
   3. Similar to other targets, add the creation of **.hls.done** for the Makefile. For **touch** you need to source the **utils.tcl** script which is found at the beginning of other Vivado run scripts in the labs.
   4. Vivado HLS does not currently have an equivalent of the **vivado -mode batch** option, so the very last command in the script must be **exit**.
2. IP output products: Vivado HLS generates the IP output products in the solution **impl/ip** directory. If you completed the Packaged Custom IP Lab, you will recall the recommendation of which files to place under revision control:
   1. HDL files
   2. xgui directory
   3. component.xml file
3. Test the hls.tcl file. The command for running Vivado HLS in Tcl mode is:

**vivado\_hls -f ../scripts/hls.tcl -l hls.log**

1. Update the Makefile. Copy the Makefile from scripts to work if necessary and add the following:
   1. A new target **hls** dependent on **.hls.done**.
   2. A rule to make **.hls.done** by running Vivado HLS in Tcl mode, sourcing the **hls.tcl** script.
   3. Create a Makefile variable **HLS** with the value of the dependent files, then use the HLS variable as the dependency in the rule to make **.hls.done.**
2. Test the Makefile:
   1. First run make clean.
   2. Next run make hls.
   3. Verify that the IP output products are generated.
3. Since we are treating the HLS IP as packaged IP, copy the IP output products to the **cip** directory.
   1. Create a new directory **rgb\_mux** inside **cip**.
   2. From the HLS-exported **impl/ip** directory, copy **component.xml,** **hdl**, and **xgui** to the **cip/rgb\_mux** directory.
4. Run **git status** to see what has changed since the last checkin.
   1. The **hls** directory should be new and untracked.
   2. In **scripts** are the new files **hls\_directives.tcl** and **hls.tcl**, and the modified **Makefile**.
   3. In **cip** is the new directory **rgb\_mux.**
5. Use **git add** and **git commit** to check in these files. Use **git status** to make sure the repository is up-to-date.

## Summary

This lab has demonstrated the process of generating an HLS-based packaged IP. Some key points to remember:

* + Vivado HLS automates the IP generation process by packaging IP and generating output products.
  + Vivado HLS can be scripted and generates Tcl scripts to recreate projects from scratch.
  + The Vivado HLS sources, directives, and scripts should be kept under revision control as well as the generated output products of the exported IP.

# Lab 6: System Generator DSP Subsystem (under constr)

This lab covers the revision control of a SysGen subsystem. This lab does not cover the generation of the subsystem and does not require a MATLAB license. Instead we will begin with a fully-generated subsystem.

1. (TBE) What steps to learn here?
2. (TBE) What files to begin with?

## Lab Procedure

1. start here…