Lab Answers: Revision Control

Contents

[Lab 1 : Simple RTL Project 1](#_Toc411524376)

[Lab 2: Simple IP Project 2](#_Toc411524377)

[Lab 3: Custom IP Project 3](#_Toc411524378)

[Lab 4: HLS-Based IP 4](#_Toc411524379)

[Lab 5: IPI Block Design Project 4](#_Toc411524380)

[Lab 6: Top Level Integration 5](#_Toc411524381)

# Lab 1 : Simple RTL Project

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

Answer: Creates a Vivado project and adds the source files.

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?

Answer: import\_files will copy the file into the project instead of referring to the files from their locations external to the project. That results in multiple copies of the same file which must be managed under revision control which is not desired.

Question 1-2a: What is the purpose of **compile.tcl**?

Answer: Opens the project created by setup.tcl and runs synthesis and implementation, all the way to bitstream generation.

Question 1-2b: What **two** important files are generated by compile.tcl?

Answer: top.bit and .compile.done. The touch proc creates .compile.done at the end of the compile.tcl script. The timestamp of .compile.donw is used by make to manage dependencies. If .compile.done is missing or older than .setup.done, the compile target is made.

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

Answer: Targets are: all, setup, compile, clean. Requires run script: setup, compile.

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?

Answer: One example is RTL. RTL is assigned the value of the RTL files. This allows the RTL file list to be defined in a single location and also makes the overall makefile more readable. The syntax for dereferencing the RTL variable in make is $(RTL).

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

Answer: a tab character. This is the default character that make uses to identify rules. Sometimes copy and paste will replace the tab with multiple spaces which can cause errors.

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

Answer: compile depends on .compile.done which in turn depends on .setup.done, which in turn depends on $(RTL) and $(XDC).

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?

Answer: No, the setup target is already made at this point, so .setup.done should exist. So then make will only generate the .compile.done target which runs **compile.tcl**.

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

Answer: The **all** target consists of both setup and compile, both of which are up-to-date at this point. So there is no action for **make all**.

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.

Answer: These files are needed to regenerate the design:

1. scripts: setup\_simple.tcl, compile.tcl, Makefile\_simple
2. hdl: top\_simple.v, threeflop.v
3. xdc: top.xdc, top\_io\_simple.xdc

# Lab 2: Simple IP Project

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

Answer: Like many processes, the actions can be performed in the GUI, which results in the equivalent Tcl commands being executed and output to the vivado.jou journal file. This includes creating, customizing, and generating IP.

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

Answer: create\_ip

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

Answer: set\_property -dict. The ‑dict option applies the arguments as key-value pairs like the Tcl dict structure, a series of property names and property values.

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

Answer: In the working directory is **ip** ($ipDir) and inside ip is **axi\_iic\_0**.

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

Answer: This is a convenient target for make. Since .ip.done is a single file created at the end of IP generation, make can check this single file to determine whether or not dependencies need to be recreated.

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

Answer: make ip

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

Answer: All files inside **axi\_iic\_0**. These would be placed under revision control.

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

Answer: Only the **axi\_iic\_0.xci** file is needed. The rest can be regenerated.

# Lab 3: Custom IP Project

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

Answer: The bft source is copied into the IP root directory for synthesis and simulation of the custom IP.

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

Answer: ipx::package\_project

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

Answer: the source files (e.g. HDL, XDC), **component.xml**, and the **xgui** directory and contents.

# Lab 4: HLS-Based IP

Question 4-1a: What program is used to run **hls.tcl**?

Answer: Vivado HLS. Notice that some Tcl commands are similar but this script is for HLS.

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

Answer: export\_design -format ip\_catalog

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).

Answer: The script to recreate an HLS project is automatically generated when you run HLS. The file **hls/script.tcl** shows in its header that it is automatically generated.

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

Answer: from work: ./rgb\_mux/solution\_zc702/impl/ip. This is where you will find hdl, component.xml, and xgui.

# Lab 5: IPI Block Design Project

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

Answer: By these commands:

set\_property ip\_repo\_paths {../cip/bft ../cip/rgb\_mux} [current\_fileset]

update\_ip\_catalog

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

Answer: the reason it takes a bit of effort is that the detailed block design generation is sourced from another script **bd\_cip\_<Vivado version>.tcl.** The commands for the following questions are found in that script.

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

Answer: create\_bd\_cell. Specifically:

set bft\_0 [ create\_bd\_cell -type ip -vlnv xilinx.com:ip:bft:1.0 bft\_0 ]

set rgb\_mux\_0 [ create\_bd\_cell -type ip -vlnv xilinx.com:hls:rgb\_mux:2.3 rgb\_mux\_0 ]

* What command is used to connect IP blocks?

Answer: connect\_bd\_net

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

Answer:

# Prepare as out-of-context run

create\_fileset -blockset -define\_from zynq\_bd zynq\_bd

# Generate output products similar to IP

generate\_target all [get\_files \*/zynq\_bd.bd]

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?

Answer: zynq\_bd only. Only zynq\_bd contains the relevant files to reuse the generated block design. zynq only contains a project that is used to run the block design as an out-of-context module.

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

Answer: As noted in the previous answer, the zynq project is only created to launch the out-of-context run for zynq\_bd.

Question 5-5: What purpose does block.tcl serve in a scripted flow environment?

Answer: The file contains all the Tcl commands to recreate the block design from scratch. It is a convenient way to manage block designs under revision control.

# Lab 6: Top Level Integration

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

Answer: all, compile.

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

Answer: When REUSEGOLDEN is true:

1. The setup target dependencies include only the $(IP) and $(BD) sources. If the sources are not out of date, then the generated, golden results are reused.
2. The **setup.tcl** script is called with the option **-tclargs reuseGolden.**

When REUSEGOLDEN is false:

1. The setup target dependencies include not only the $(IP) and $(BD) sources but also the **done** target files. It is assumed that the generated results are never stored and always generated from the source files.
2. The **setup.tcl** script is called without the option **-tclargs reuseGolden.**

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

Answer: Without reuseGolden, the script looks for the **ip** and **bd** in ‘.’ (the work directory). When reuseGolden is 0, the script looks in ‘..’ which is the level above work, and also the locations where the generated IP and block design are checked in.

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

Answer: TRUE. Always reuse the generated products if they are up-to-date.

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

Answer FALSE. Do not store the generated products, and instead generate them from the source files.

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?

Answer: In addition to the files generated for the top design, you would also see the files for the IP and block design generation.