Bonus(1)

-High level synthesis (Behavioral modeling):

Describe your design using C or C++ language and let Xilinix Vivado HLS convert your code to optimized hardware.

It turns functions into synthesised modules..

The first videos of the following playlist shows you the steps:

Vivado HLS Course Training: <https://www.youtube.com/playlist?list=PLo7bVbJhQ6qzK6ELKCm8H_WEzzcr5YXHC>

Bonus (2)

-Write a script that creates work library, compiles your modules, runs your test benchs and opens modelsim simulation to view important waves of your design using any scripting language.

You notice when using modelsim, that the transcript part has some written code like vsim, run etc..

If you wrote these instructions in a script and run it, it will do the same things you are doing manually.

The following playlist shows you how to write a code using TCL

TCL tutorial: Basics to Advanced: <https://www.youtube.com/playlist?list=PL1h5a0eaDD3rsGDFnVki_fFEtDWQfXjca>

Here is a sample code for what you will do, this code is very basic but it’s an excellent start, you could use while loops and other features if you want:

#global flag to avoid building and compiling multiple times

set build\_done 0

#define a function for building that runs one time

proc build {} {

#read the global build flag

global build\_done

#do the build routine, make sure to include all files

vlib work

vlog “file1.v”

vlog “file2.v”

#set build flag to 1

set build\_done 1}

#define a function for simulating each test bench ( this is repeated for each tb with different details’ names of course )

#tbmodule1 is the name of the test bench module not the test bench file

proc firstTb {}{

global build\_done

if {$build\_done ==0}{build}

#simulate, the “\*” symbol means all, however you could remove it and write the signals names if you want specific signals or order.

vsim work.tbmodule1

add wave \*

#run your simulation

run

}

Bonus(3) (Working in hardware is not only about design, you could be a verification engineer.. So, writing good test benches is necessary.)

-Dynamic Testing: It is some testing procedures that are done while executing the code, it contains:

1)Unit testing(1+ test bench (s)): write a tb for each module.

2)Integration testing(1+ test bench (s)): write a tb for each interacting modules to monitor the flow of data from one module to another.

3)System testing (1 test bench): write one tb that tests the behaviour of the system as a whole.

Note that while testing, you test the normal cases like 1+4 =5 and the corner cases like 4/0 to know the behaviour of the system in these cases so that it doesn't give you false answers, depending on your design you may have other corner cases too, test them all.

Bonus (4)

Floating point operations:

Design one/many verilog module/s that does operations (adds/ subtracts/ multiplies) on IEEE floating point numbers, the user enters the numbers in decimal and sees the result in decimal.

Bonus (5)

Pipelined structure for your project.