Verilog Design Entry, Synthesis, and Behavioral Simulation

PURPOSE - This lab will present a brief overview of a typical design flow, walk through some typical tasks, and introduce the user interface of Xilinx Vivado 2016.2. You will design a very simple circuit during the Design Entry, Synthesis, and Functional Simulation. You will also learn how to do simple behavioral simulation of Verilog designs. This lab, along with the following two labs, will walk through a HDL design flow from design entry to implementation.

Introduction

Figure 1 presents the basic steps we have to perform when we design circuits we want to implement in a Field Programmable Gate Arrays (FPGAs) or Complex Logic Devices (CPLDs).

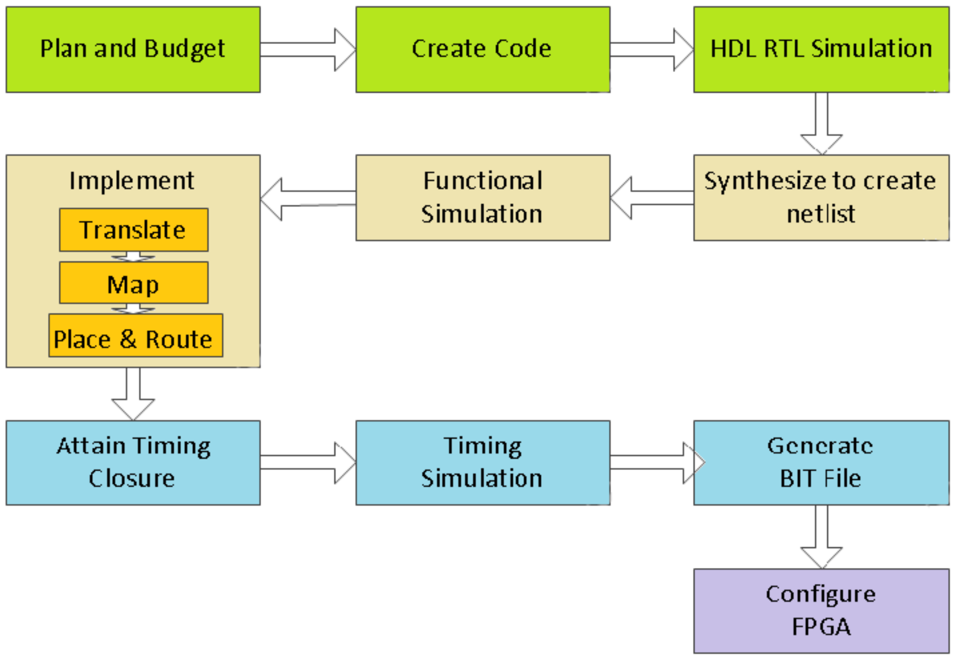


Figure 1. Typical design flow

The Vivado Design Suit was introduced by Xilinx as a design tool for their most recent families of FPGAs. The Vivado Integrated Development Environment (IDE) provides a graphical user interface (GUI) for various design flows which target programmable logic devices.

For the EE4301 labs we will use Verilog, a Hardware Definition Language (HDL), to do Register Transfer Level (RTL) design targeting an Artix-7 FPGA on a Digilent Basys 3 development board. Vivado will be used to synthesize, simulate, implement and finally generate a bitstream to download to the Basys 3.

Vivado can be used to implement several design flows and can be operated in project or non-project mode. For the EE4301 labs we will follow an RTL design flow as shown in figure 1.1. This RTL design flow consists of three main steps.

1. **System Design Entry**

   The design entry step involved creating a description of the design, verifying it, and doing a behavioral simulation. This involves describing the design with HDL code or using schematic entry and may involve third party IP.

2. **Implementation**

   The implementation step involves specifying constraints and synthesizing the design to generate a netlist. Functional simulation is used to verify the synthesized design which is then placed and routed for the FPGA.

3. **Hardware Bring-Up and Validation**

   Finally, a bitstream is generated which can be downloaded to the FPGA to realize the design. Hardware testing is then used to validate the final design in the hardware.

We will use Xilinx Vivado Suite (version 2016.2) to do these steps.

Organizing Your Work

To organize your work, ISE groups all related files into separate logic units called projects. A project is a collection of design files stored in a separate subfolder, called the project working directory, which is treated as a separate logic unit and includes:

– source files
– output and intermediate files (netlists, report and log files, bit stream files)
– configuration files

Your design is represented in various ways. Those representations are called sources. A source is any element in the design such as an HDL file, schematic, or simulation file. Sources are displayed in the Sources in Project window. Sources include not only the description of the logic circuits, as represented by schematics and hardware description languages, but also include simulation test fixtures and documentation of the design. All these pieces are part of the whole design.

The design description (logic) for a project is contained within the following types of sources

- HDL files (VHDL, Verilog, or HDL ABEL)
- EDIF
- Schematics
- IP

One source file in a project is the top-level source for the design. The top-level source defines the inputs and outputs that will be mapped into the device, and references the logic descriptions contained in lower-level sources. The referencing of another source is called "instantiation." Lower-level sources can also instantiate sources to build as many levels of logic as necessary to describe your design.

During this lab you will describe a very simple circuit, an 8-bit adder, using Verilog. You will learn about Verilog, a Hardware Definition Language (HDL), during the class. Further information about Verilog can be found in the references at the end of this lab as well as on the class web page.

NOTE: To organize your work during the laboratories of this class, create a directory called **labs** in your account and then create directories **lab1**, **lab2**, **lab3,** etc. (one separate folder for each lab). During each lab save your projects and any file to the corresponding directory.

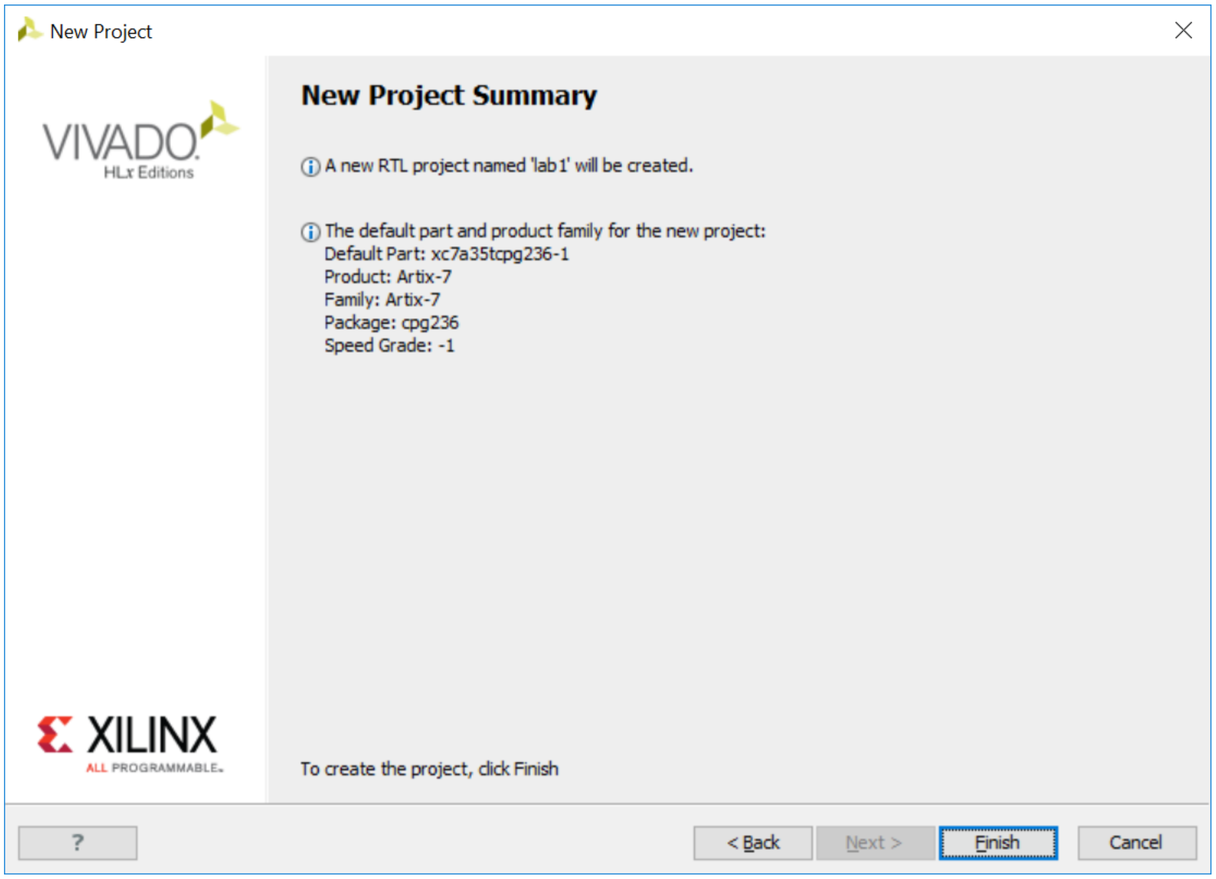
Creating a New Project with Vivado

1. Start Vivado by clicking on the Vivado icon (or select the Start button and then type Vivado). The Start Page appears. Click on **Create a New Project**, then click **Next** to invoke the New Project Wizard.

2. In the **Project Name** screen enter the project name, `lab1`, and the location, `C:/…/EE4301_Labs`. Verify that the **Create a Project subdirectory** is checked. Then click **Next**.

3. In the Project Type screen select RTL project and check Do not specify project sources at this time, we will add these later. Click **Next**.

4. In the Default Part screen select xc7a35tcpg236-1. This corresponds to the Artix-7 device on the Basys 3 board. Click **Next**. Assuming you have entered the previous items correctly the New Project Summary appears and should be the following:



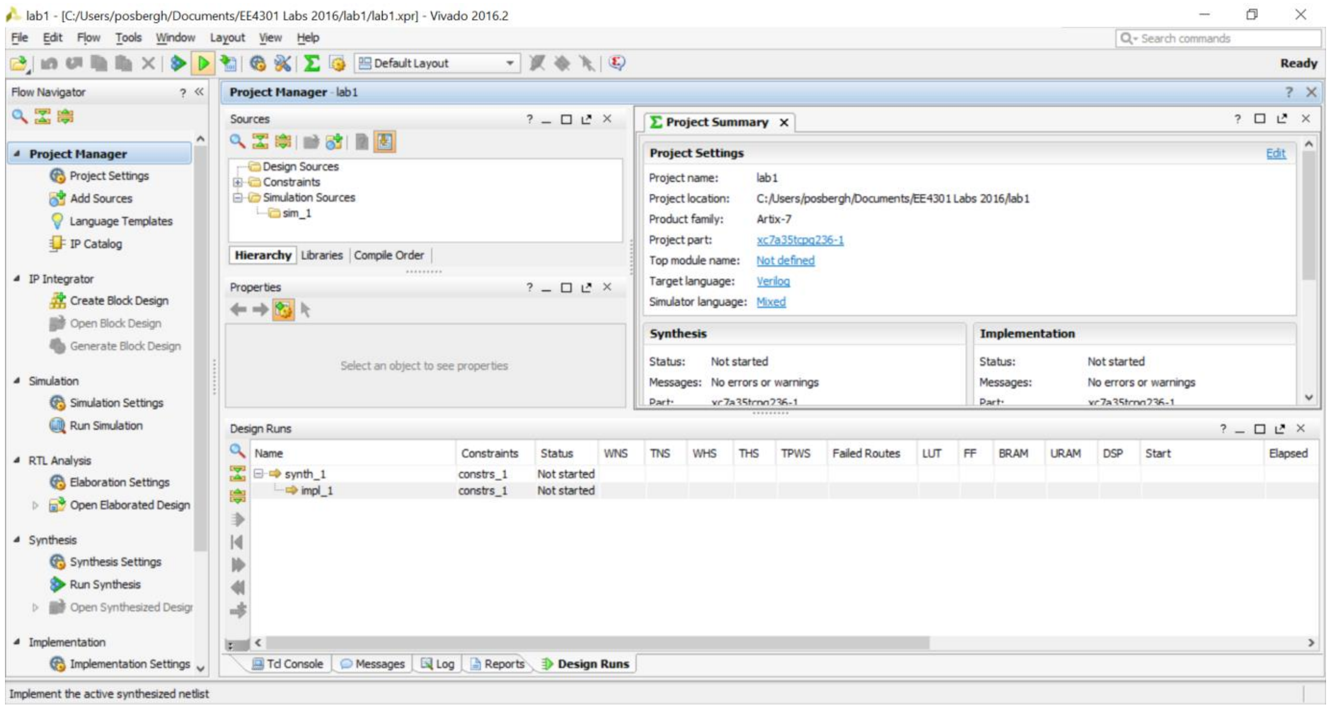
New Project Summary

A new RTL project named 'lab1' will be created.

The default part and product family for the new project:
Default Part: xc7a35tcpg236-1
Product: Artix-7
Family: Artix-7
Package: cpg236
Speed Grade: -1

To create the project, click Finish

5. To complete the specification of the new project and create it click **Finish**. The project will then be created and the Vivado project window appears.



Project Settings

Project name: lab1
Project location: C:/Users/posbergh/Documents/EE4301 Labs 2016/lab1
Product family: Artix-7
Project part: xc7a35tcpg236-1
Top module name: Not defined
Target language: Verilog
Simulator language: Mixed

Synthesis

Status: Not started
Messages: No errors or warnings
Part: xc7a35tcpg236-1

Implementation

Status: Not started
Messages: No errors or warnings
Part: xc7a35tcpg236-1

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1	constrs_1	Not started													
impl_1	constrs_1	Not started													

We are now ready to enter our design.

Entering the Design

For this project, lab1, we create an 8-bit adder. The adder will be specified with two Verilog source files which we will create using the Vivado editor.

In the Project Manager pane of the Vivado Project Window click on Add Sources (alternatively we could select in the menu bar File > Add Sources or just type Alt-A). The Add Sources window should appear. In this window choose Add or create design sources and then click Next. The Add or create design sources window now appears. Choose Create File to invoke the Create Source File pop-up window. The file type should be Verilog, name this file full_adder, and verify the file location is <Local to Project>. Click OK. This then appears in the list of source files, click Finish.

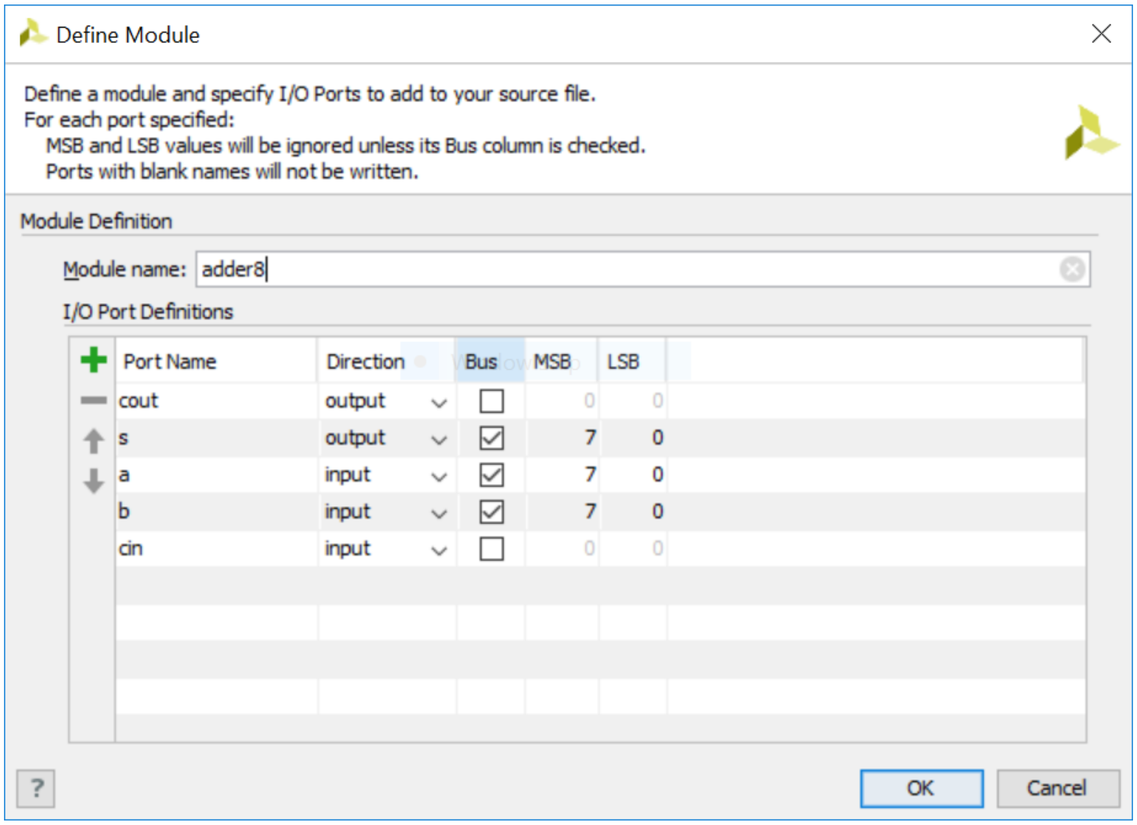
The Define Module window now appears and can be used to specify the ports of the module. For the full_adder we will specify these manually in our source code. So click OK to skip the port declarations.

In the editor window, enter the following Verilog code for the full_adder module

```
module full_adder(
   output cout,
   output sum,
   input  ain,
   input  bin,
        input  cin
        );

        assign sum = ain^bin^cin;
        assign cout= (ain&bin)|(ain&cin)|(bin&cin);

    endmodule
```

This Verilog source code will be contained in the source file full_adder.v.

Next we repeat the process to create a higher level module which implements an 8-bit ripple carry adder. This module uses the full adder module as a component. For this module we will use Vivado to specify the ports. Name the file `adder8`. This module has one scalar output `cout`, an 8-bit output `s`, and two 8-bit inputs, `a` and `b`. These are specified in the Define Module window as shown below.



Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name: adder8

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
cout	output	☐	0	0
s	output	☑	7	0
a	input	☑	7	0
b	input	☑	7	0
cin	input	☐	0	0

Type in the remaining code to create the following Verilog module:

```
    module adder8(
       output cout,
       output [7:0] s,
       input [7:0] a,
       input [7:0] b,
       input cin
       );
```

```
        wire [7:1] carry;

        full_adder FA0(carry[1],s[0],a[0],b[0],cin);
        full_adder FA1(carry[2],s[1],a[1],b[1],carry[1]);
        full_adder FA2(carry[3],s[2],a[2],b[2],carry[2]);
        full_adder FA3(carry[4],s[3],a[3],b[3],carry[3]);
        full_adder FA4(carry[5],s[4],a[4],b[4],carry[4]);
        full_adder FA5(carry[6],s[5],a[5],b[5],carry[5]);
        full_adder FA6(carry[7],s[6],a[6],b[6],carry[6]);
        full_adder FA7(cout,s[7],a[7],b[7],carry[7]);

    endmodule
```

When completed, the two modules can operate as an 8-bit adder for the top-level module. They will appear indented under the Design Sources item in the **Sources** window in the **Project Manager**. You should make sure you understand how this design hierarchy is used. Make sure you can explain how this circuit functions to the TA and in your lab notebook.

In addition to regular editing features, the editor provides syntax coloring. The syntax coloring feature supports three languages: Verilog, ABEL and Verilog. The HDL Editor will operate as a standard text editor as well.

Double-click on the name of either of the above two files to open it with the HDL Editor.

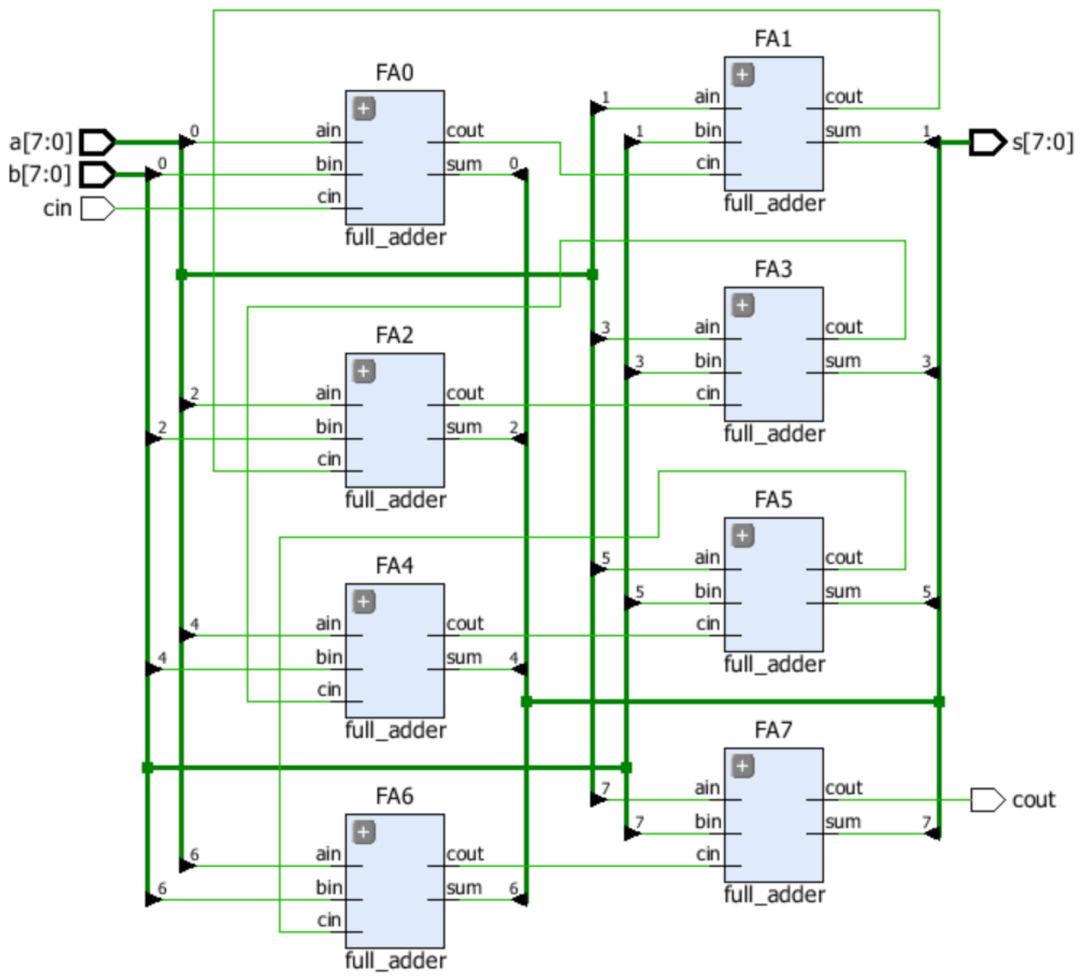
NOTE: Take a little time to get familiar with HDL Editor.

Register Transfer Level Analysis

With the design sources entered the next step is the compile the Verilog code and verify the RTL model. The steps in the design flow that will do this are in the **Flow Navigator** panel in the left area of GUI. First, you must select the correct design module (top-level) in the Hierarchy panel at the upper left. Click on Adder8 and in the menu that pops up select Set as Top. We are now ready to compile the Verilog code.

Under **Synthesis** in the **Flow Navigator** click on Run Synthesis (or click on the corresponding menu item in the toolbar or select Flow > Run Synthesis or press F11). This initiates the synthesis step which compiles the Verilog code, this step may take some time. Messages will appear in the Tcl Console at the bottom of the GUI. Assuming no errors you can view the Vivado synthesis report by selecting the Reports Tab in this window. If these are errors you will need to go back, correct your code, and recompile. Note that syntax errors will generally indicate the line in your code where the error was reported which is where you should look. Typically syntax errors arise from misspelt keyword, missing semicolons, or invalid syntax. When you have found and corrected such errors recompile by repeating Run Synthesis. When synthesis completes the Synthesis Completed pop-up window appears. Select View Reports and click OK.

Take a look at the report. It describes the synthesis process and the resources required to implement your design. We can further analyze the design by selecting Open Elaborated Design under the RTL Analysis item in the Flow Navigator. If you select the schematic item under this entry you can view a schematic of your design. It should look like the following



You can view the corresponding logic in one of the full_adder components by clicking on the box. You should do this and verify the logic implements a full adder.

Performing a HDL Behavioral Simulation

When writing Verilog descriptions, you usually verify their correctness (to see that they behave in the desired fashion) via simulation. Vivado contains a tool called XSim which allows you to run simulations. For a behavioral simulation the first step is to create a testbench module, which creates another level of the design hierarchy only used by the simulator. The testbench will instantiate your top-level module and connect all of its input and output signals, allowing you to create stimulus for the inputs and observe the outputs.

Using Vivado we can create a testbench and run a behavioral simulation. This testbench, add8_tb.v, can be found with lab1. You should download it and place it in the lab1 project folder at this time. As before in the Project Manager pane of the Vivado Project Window click on Add Sources (alternatively we could select in the menu bar File > Add Sources or just type Alt-A). In the Add Sources window choose Add or create simulation sources and then click Next. The Add or create design sources window now appears. Choose Add Files to invoke the Add Files File pop-up window. You should be able to find add8_tb.v in the pop-up window that appears, select it and click OK. This then appears in the list of source files, click Finish.

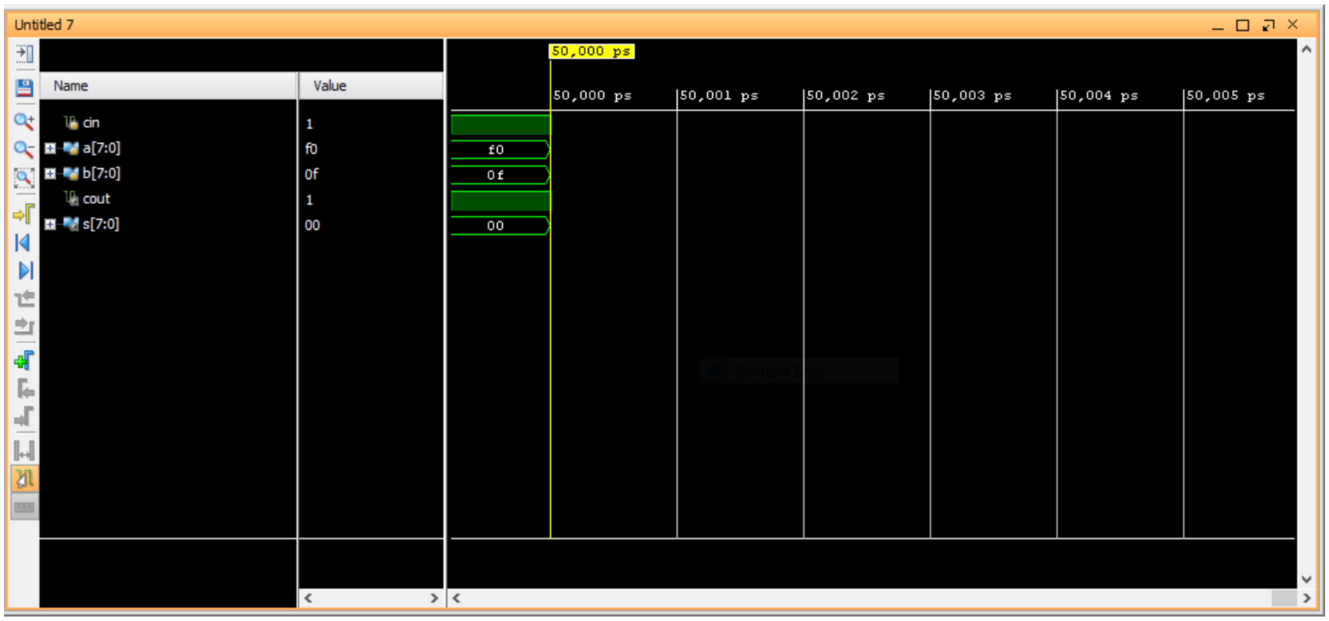
A listing of this testbench is as follows

```
`timescale 1ns / 1ps

module adder8_tb(
    );

    reg cin;
    reg [7:0] a,b;
    wire cout;
    wire [7:0] s;

    // instantiate the unit under test uut
    adder8 uut(cout,s,a,b,cin);

    // stimulus (inputs)
    initial begin
       #10 // wait for global reset
       a=8'b00000000; b=8'b00000000; cin=1'b0;
       #10
       a=8'b11111111; b=8'b00000001; cin=1'b0;
       #10
       a=8'b10101010; b=8'b01010101; cin=1'b0;
       #10
       a=8'b11110000; b=8'b00001111; cin=1'b1;
    end

    // results (outputs)
    initial begin
       #5
       $display("a = %b, b = %b, cin = %b",a,b,cin);
       $display("s = %b, cout = %b",s,cout);
       #10
       $display("a = %b, b = %b, cin = %b",a,b,cin);
```

```
        $display("s = %b, cout = %b",s,cout);
        #10
        $display("a = %b, b = %b, cin = %b",a,b,cin);
        $display("s = %b, cout = %b",s,cout);
        $display("End Simulation");
    end

endmodule
```

The testbench should appear along with add8 under sim_1 under Simulation Sources in the Sources window. In the Flow Navigator under Simulation open Simulation Settings and verify that add8_tb is the Simulation top module name, then click OK.

You are now ready to run the simulation. Under Simulation in the Flow Navigator click Run Simulation. A pop-up menu will present several options. Select Run Behavorial Simulation. The simulation will run for 1000 ns (default) and then open the simulation window which will look like



Name	Value
cin	1
a[7:0]	f0
b[7:0]	0f
cout	1
s[7:0]	00

When it works properly, you will see the input values and the result on the s[3:0] and cout signals. Everything should show in green, if there is red or blue waveforms you have some type of error in the design and you need to debug it. If an output has blue and the value 'z', it means the signal is floating, which means you probably have a typo in a name so a connection is not made. If it is red with the value 'x', it is unknown so there may be multiple gates driving the same signal, for example.

Each time you add #10 you cause the sim to advance 10ns, and then you can change the values on the input pins. Every signal keeps the last value assigned until you change it, so you don't have to list very input every time. Create several different input combinations, so that you can verify that proper addition and subtraction occurs.

Explore the simulation. Experiment with different stimulus and other features of the simulation, and its interaction with Vivado. Print out a good view of the waveforms, and paste it into your lab notebook to show that your adder/subtractor functions properly.

— End of Lab 1 —

Close the Project Navigator. In Lab 2 you will continue with the implementation step.

NOTE: Vivado is a quite big software package with many tools and features. It is not possible to cover everything about it within the labs. However, the labs will cover the basics. That is why it is highly recommended to go through **Help> Software Manuals and Help> Tutorials > Tutorials on Web** during your off-labs time. You will find plenty of extra information!

SUMMARY -- This laboratory wanted to teach you how to perform Design Entry (using the HDL Editor tool), Analyze the RTL, and do a Behavioral using Xilinx Vivado 2016.2.

REFERENCES

[1] Xilinx, *Vivado Design Suite User Guide: Using the Vivado IDE*, UG893(v.2016.2) June 8, 2016.

[2] Xilinx, *Vivado Design Suite User Guide: Vivado Design Flows*, UG892(v.2016.2) June 8, 2016.

[3] Xilinx, *Vivado Design Suite User Guide: Logic Simulation*, UG911(v.2016.2) June 8, 2016.

[4] *Introduction to Verilog*, Supplemental Notes for EE2301, Fall Semester 2015