ESE382 Digital System Design Using VHDL and PLDs

Spring 24, Ken Short

Laboratory 02b: VHDL/PLD Design Flow - Timing Simulation, and Programming a SPLD
This laboratory is to be performed the week starting Feb. 4th. along with Laboratory 02a.

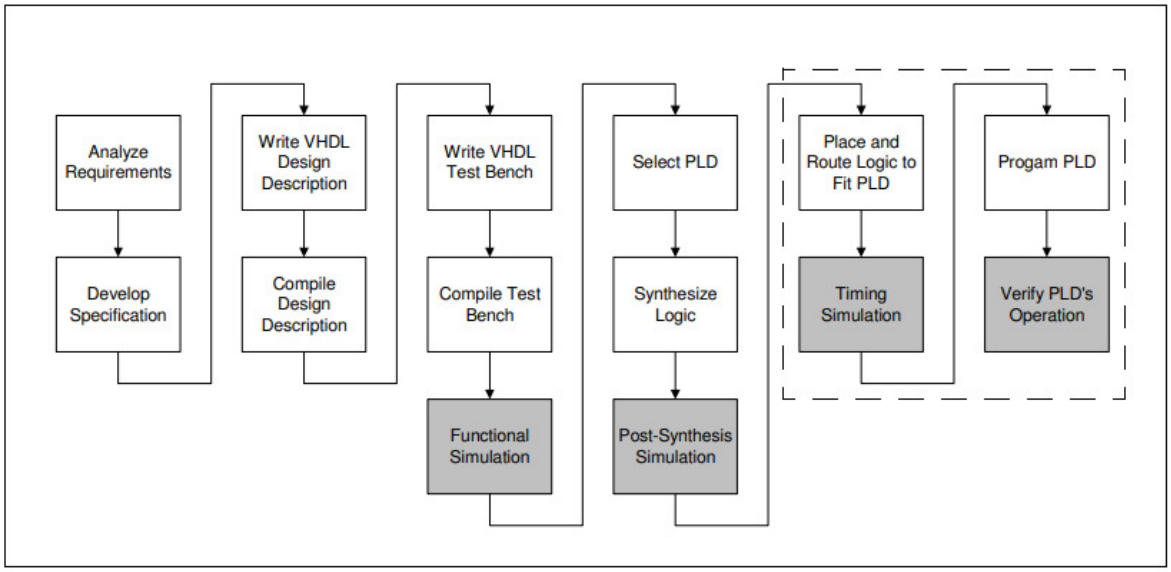
Prerequisite Reading

1. Sections 1.10 through 1.15 of text

Purpose

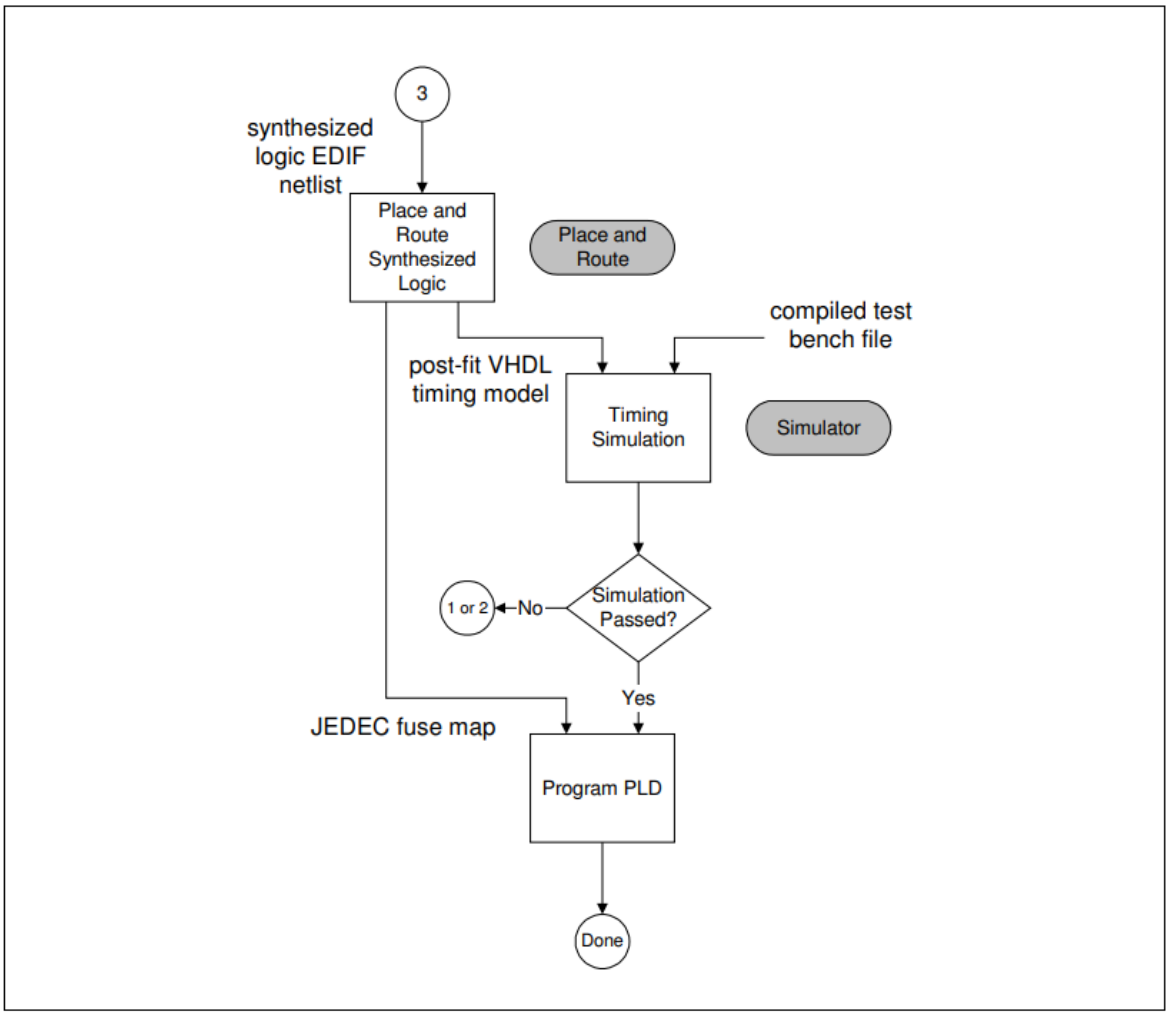
The purpose of this laboratory is to complete the VHDL design flow example from Laboratories 01 and 02a by performing a timing simulation of the design fitted to a PLD and programming and testing the PLD. The focus will be on the phases inside the dashed rectangle of Fig. 1.

Figure 1: Design flow for VHDL/PLD design methodology.



The portion of the design flow that is of interest in this laboratory is shown in more detail in the following figure.

Figure 2: The third portion of the complete VHDL/PLD design flow including timing simulation.



You will use three software tools during this laboratory:

1. Active-HDL: Compiler and Simulator
2. Synplify: Synthesizer
3. ispLEVER Classic: Place-and-route tool

The first two tools you have used previously. The third tool, ispLEVER Classic, is a place-and-route tool that can fit a synthesized design to some SPLDs and CPLDs from technology vendor Lattice Semiconductor. This Place-and-route tool takes the EDIF netlist from the synthesizer as its input. It generates a VHDL timing model of the logic fitted to the target PLD. Such a model represents both the functionality of the logic and its physical delays. Accordingly, during simulation, when the value of an input to the timing model is changed, the corresponding output(s) do not change immediately. Instead, each output changes after an appropriate delay. This delay corre-

sponds to the physical delay caused by the logic elements and interconnect in the PLD through which the signal must pass from input to output.

Design Tasks

The VHDL testbench will have to be slightly modified later during this laboratory. The testbench will be modified twice.

1. So that the input stimuli change slowly enough to allow the output signals to be stable at their new values for a short period of time.
2. So that the timing simulation can easily be duplicated by applying electrical signals to a programmed SPLD.

Examine the VHDL testbench file from Laboratory 1. In the architecture body there is a process with the label `tb`. This process assigns values to signals `a_tb` and `b_tb` so that these signals change every 20 ns. These signals connect to the UUT input signals `a` and `b`. Therefore, these inputs are successively assigned the values 00, 01, 10, and 11.

The **wait for** statements cause signals `a` and `b` to remain at their most recently assigned values for 20 ns. The **assert** statements specify the values the outputs should have at the end of the 20 ns period. The **report** statement causes an error message to be printed if the simulated output signals don't have the specified values at the appropriate times.

Your objective in the first modification of the testbench is to increase the time interval between the changes in input signals to 100 ns.

Your objective in the second modification of the testbench is to create a stimulus that can easily be duplicated with actual electrical signals on a hardware testbench. The desired input stimuli should cause signal `a_tb` to remain constant at logic 1 and cause signal `b_tb` to be a square wave with a 200 ns period. In the laboratory, you can realize the corresponding electrical signals with a +5V constant voltage for `a` and a 0 to +5 V square wave at a frequency of 5 MHz for `b`.

To make each of these modifications you only need to change a few values in the existing code. You don't need to remove or add statements.

Laboratory Tasks

Overview

In the laboratory you will go through all the previously covered phases in the VHDL/PLD design flow, except the post-synthesis simulation will be skipped. The phases you will carry out are:

1. Create a new workspace and a new project, both named lab02b. Add the design description and testbench from Laboratory 1 to this project. Modify the entity declaration of the design description to add attributes that specify pin assignments for port signals.

2. Use Aldec's Active-HDL simulator to compile and functionally simulate this design, just as you have done before.

3. Use Synopsys' Synplify Pro synthesizer tool to synthesize logic corresponding to your VHDL design description. You did this previously in Laboratory 2a.

4. Use Lattice ispLEVER Classic to place and route (fit) the logic to a 22V10 PLD. This place-and-route tool will produce a timing model of the half-adder logic fitted to the target PLD as well as a JEDEC file that can be used by a programmer instrument to program the PLD.

5. Modify the testbench so that inputs are changed at 100 ns intervals.

6. Simulate the timing model to produce a timing simulation.

7. Modify the testbench to create a different set of stimuli and perform a second timing simulation.

8. Program a 22V10 and test its functional performance against what was predicted by your functional and timing simulations.

Create a new workspace and project

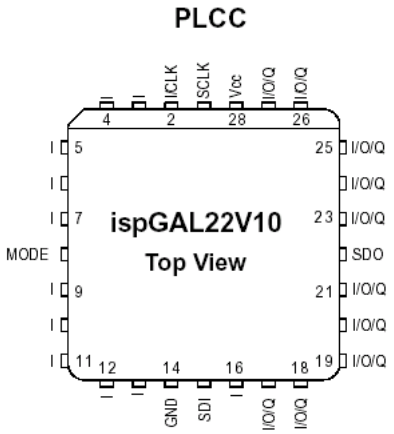
1. Follow the procedures from Laboratory 1 to launch Active-HDL and create a new folder, workspace, and project all named lab02b. Add the design description and testbench files from Laboratory 1 to this project.

2. Use the HDL editor to add the following attribute statements to the original entity declaration in half_adder.vhd, so that the resulting entity declaration is:

```
entity half_adder is
      port (a, b : in std_logic;
      sum, carry_out: out std_logic);

      attribute loc : string;
      attribute loc of a : signal is "P3";
      attribute loc of b : signal is "P4";
      attribute loc of sum : signal is "P27";
      attribute loc of carry_out : signal is "P26";

end entity half_adder;
```

These VHDL statements are user attributes that allow you to provide pin-locking information to the place-and-route tool to control the processing of this design description. The information provided assigns the input and output signals to pins of the 22V10 PLD used in this design. The "P*n*" specifies the pin number. The P must be upper case. From the data sheet pin out diagram of the 28 pin PLCC version of the ispGAL22V10, verify that the pin numbers assigned to input signals are

input pins of the ispGAL22V10 and the pin numbers assigned to output signals are I/O pins of the ispGAL22V10.



This pin number information is used by the place-and-route tool, which is the last tool in the tool chain. This information is passed along to the place-and-route tool in the EDIF file generated by the synthesizer. If you do not specify pin assignments, the place-and-route tool makes its own choice of pin assignments. However, since we provide a test station with a zero-insertion force (ZIF) socket to plug in your programmed PLD, you must use the same pin assignments that were used in the test station.

Compilation and Functional Simulation

1. Follow the procedures in Laboratory 1 to compile and functionally simulate your half-adder design description. Note that the folder to use is lab02b. The output waveforms are the same as in previous laboratories. As might be expected, the addition of the pin number attributes has no effect on the functionality of the VHDL design description.

Save your design by selecting **Save All** and then **Close Workspace** from the **File** menu. Close Active-HDL.

Synthesizing Logic and Creating an EDIF file

1. Follow the procedures in Laboratory 2a to synthesize your design description file. The device to be selected for this synthesis is the ispGAL22V10C-**10** LJ SPLD. Note that the folder to use is lab02b. However, you are ***not*** to perform a post synthesis simulation.

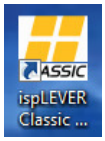
2. After the synthesis is complete, click **Close Project,** and save the changes to your project.

3. Close Synplify.

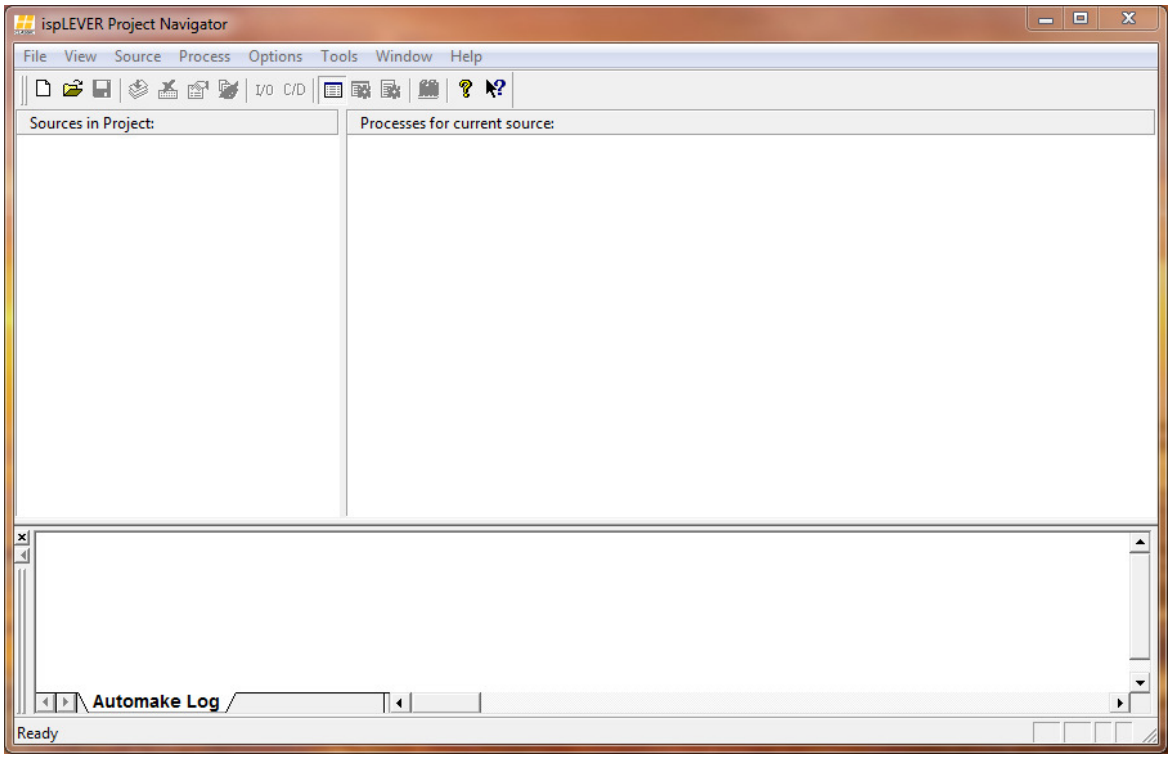
Fitting a Design and Creating a simulation model and JEDEC file

Next you will use the ispLEVER Classic place-and-route tool to fit the logic just synthesized (half_adder.edf) to the ispGAL22V10C-10 SPLD.

1. Using the desktop icon launch the ispLEVER Classic software. This will open ispLEVER Clas-

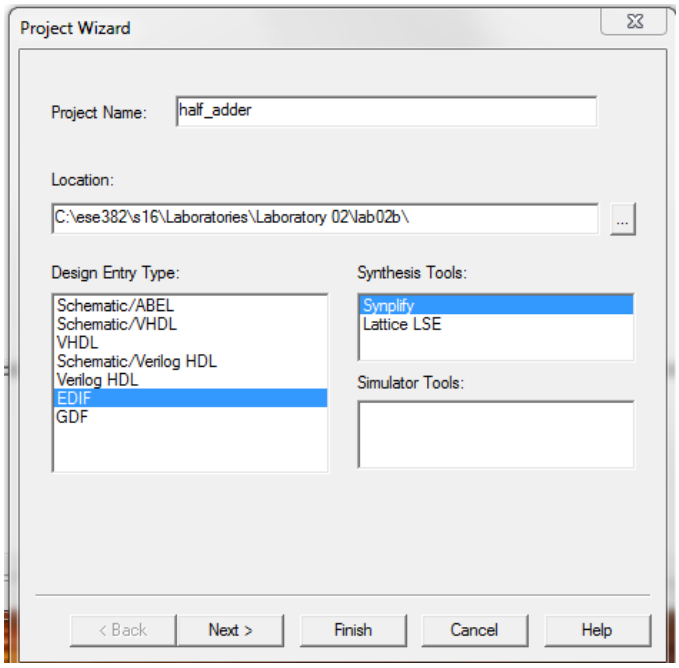


sic's main window.

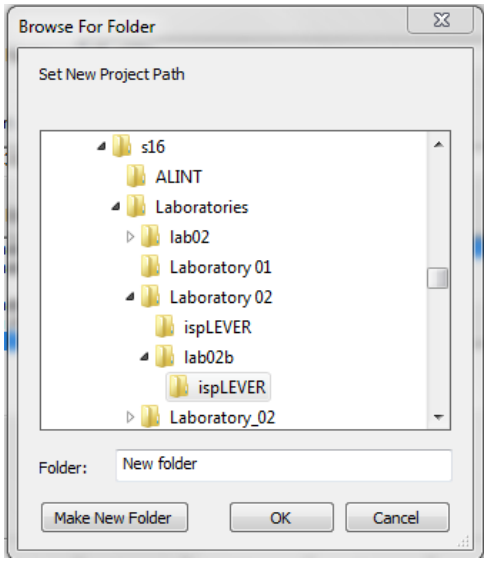


2. If a project is loaded automatically on startup, and it is not the project you need, close the project by selecting the **File** menu and clicking **Close Project**.

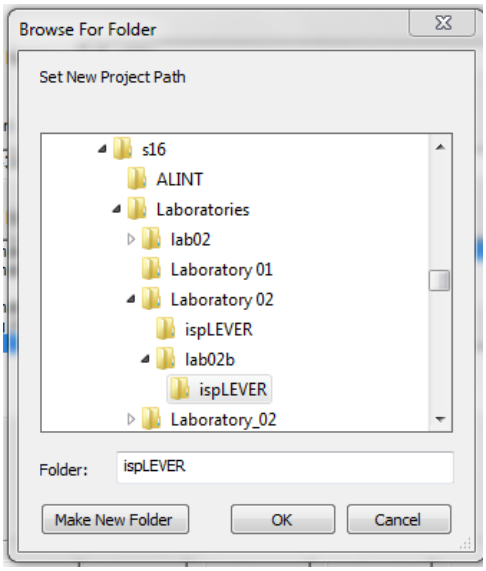
3. Create a new project by selecting **New Project** from the **File** menu**.** This opens the **Project Wizard** dialog box.



Enter the **Project Name** half_adder. In the **Location** box browse to your original design folder for lab02b, in the workspace lab02b, and click the new **Make New Folder** button. When the new folder appears type in the name ispLEVER.

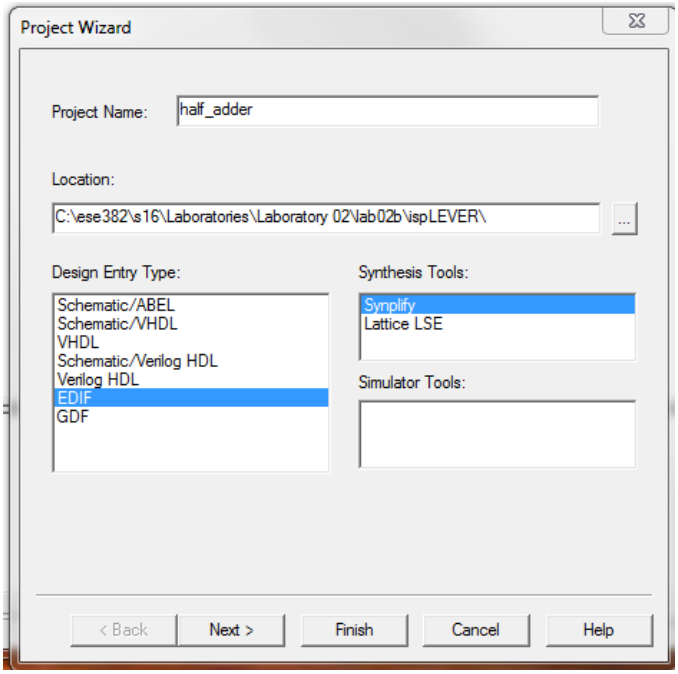


Double click the folder ispLEVER to open it. You will see the name ispLEVER in the **Folder** text box.



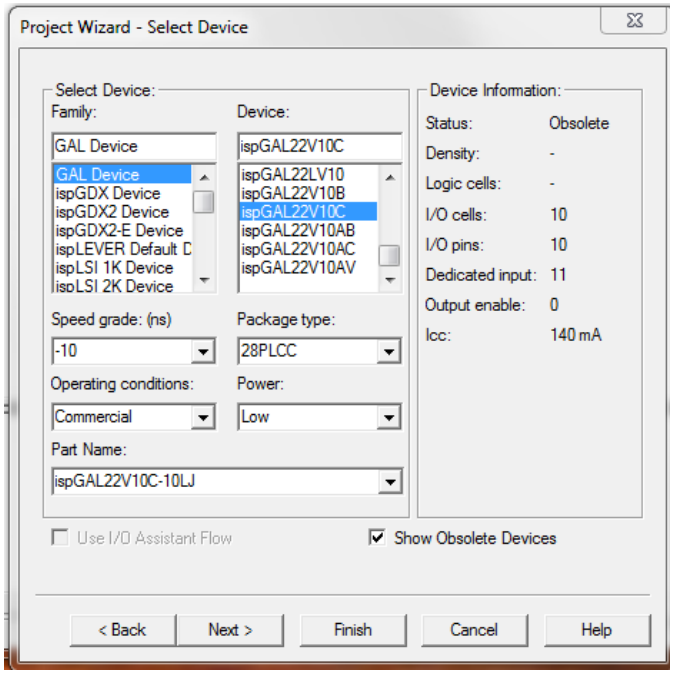
Then click **OK**.

A very important step is to define the design entry type. Select **EDIF** for the design entry type.



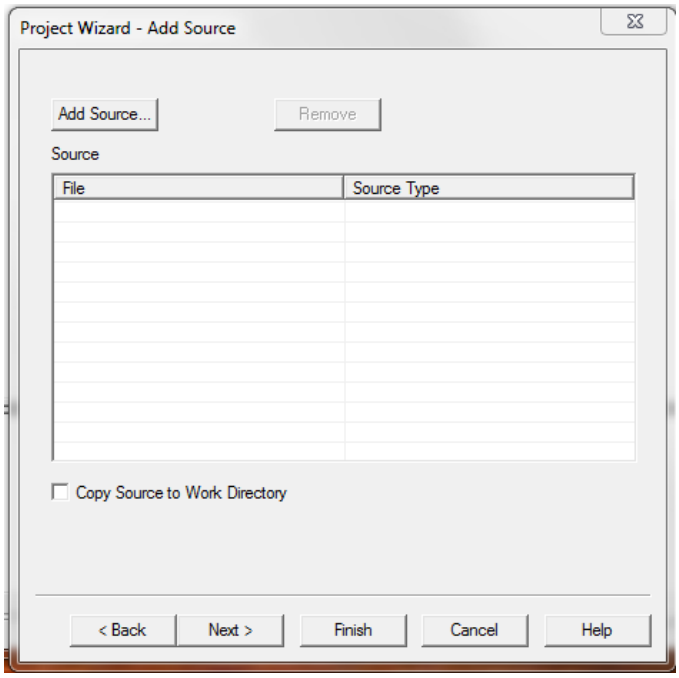
Click **Next**.

The Select Device dialog box appears. In this dialog box select, the PLD to which the design is to be fitted.

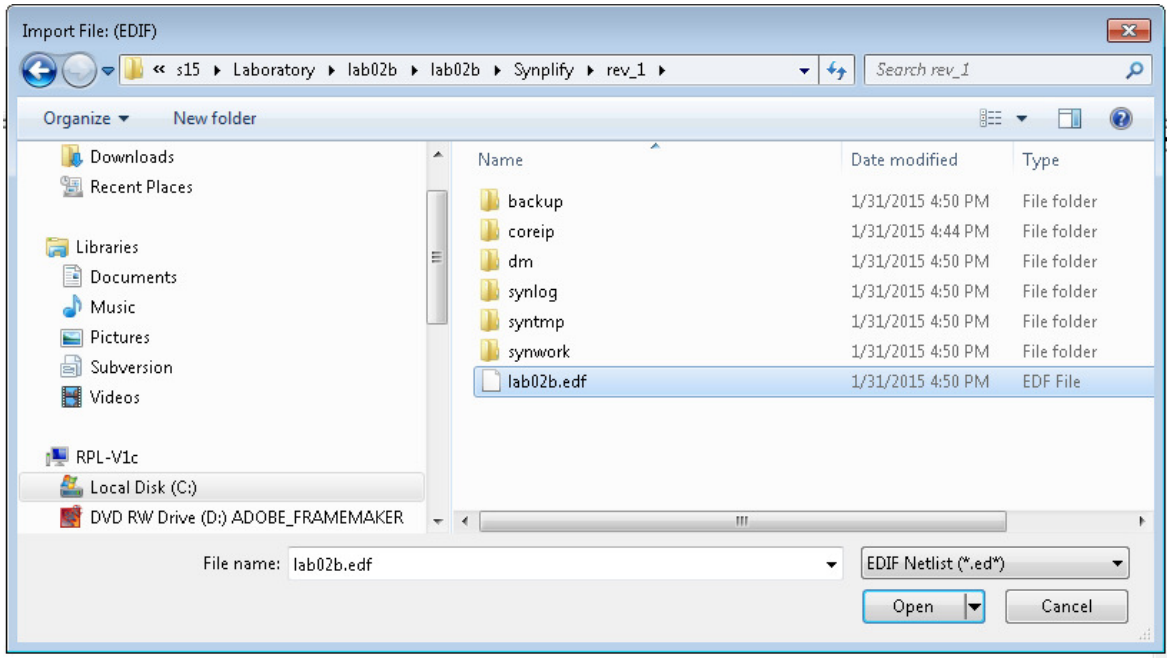


Specifying a part requires several selections. First click on the box for **Show Obsolete Devices**. Second, select the device family (GAL Device) using the **Family** menu. Next, select ispGAL22V10C as the device via the **Device** window. Make sure the **Speed grade**, **Package type**, and **Part Name** are correct. Click **Next**.

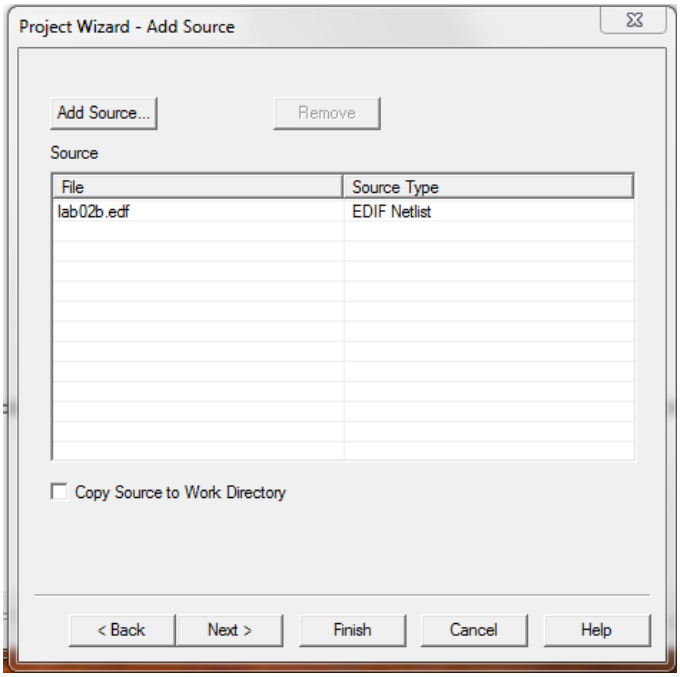
The **Add Source** dialog box opens. Click the **Add Source**.



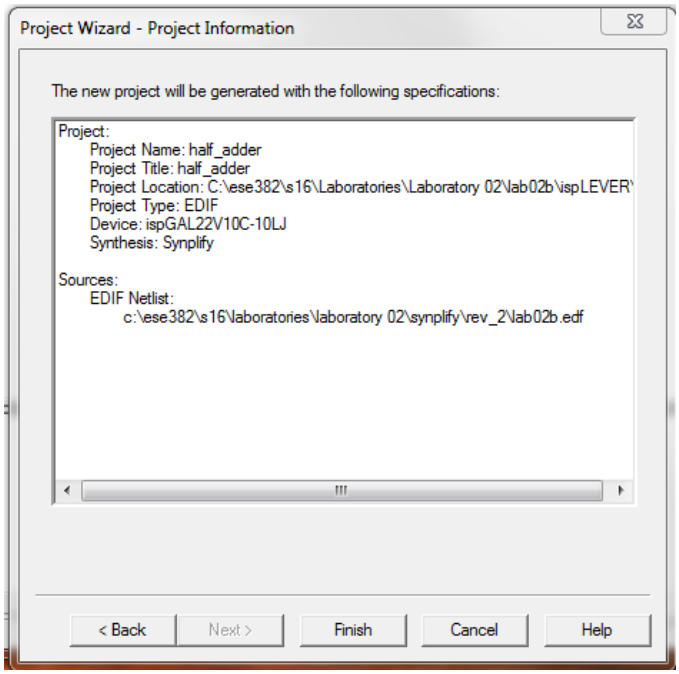
The Import File window opens.



Browse to the Synplify folder that you created in lab02b. Open the folder rev_1. Make sure that the **File type** drop down menu at the bottom right of this dialog box has EDIF Netlist (.*.ed*) selected, or you may not be able to see the file half_adder.edf. Select the file lab02b.edf. Click **Open**.

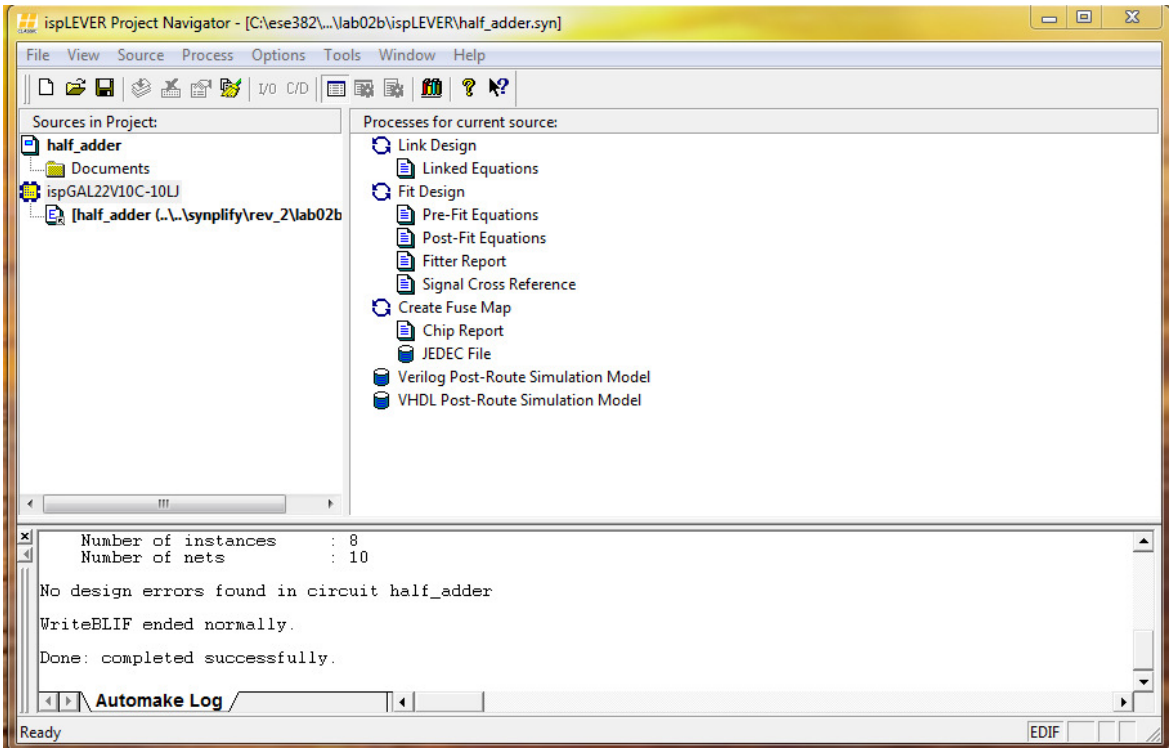


The file lab02b.edf is added to the project.
Click **Next**. The project information is displayed.

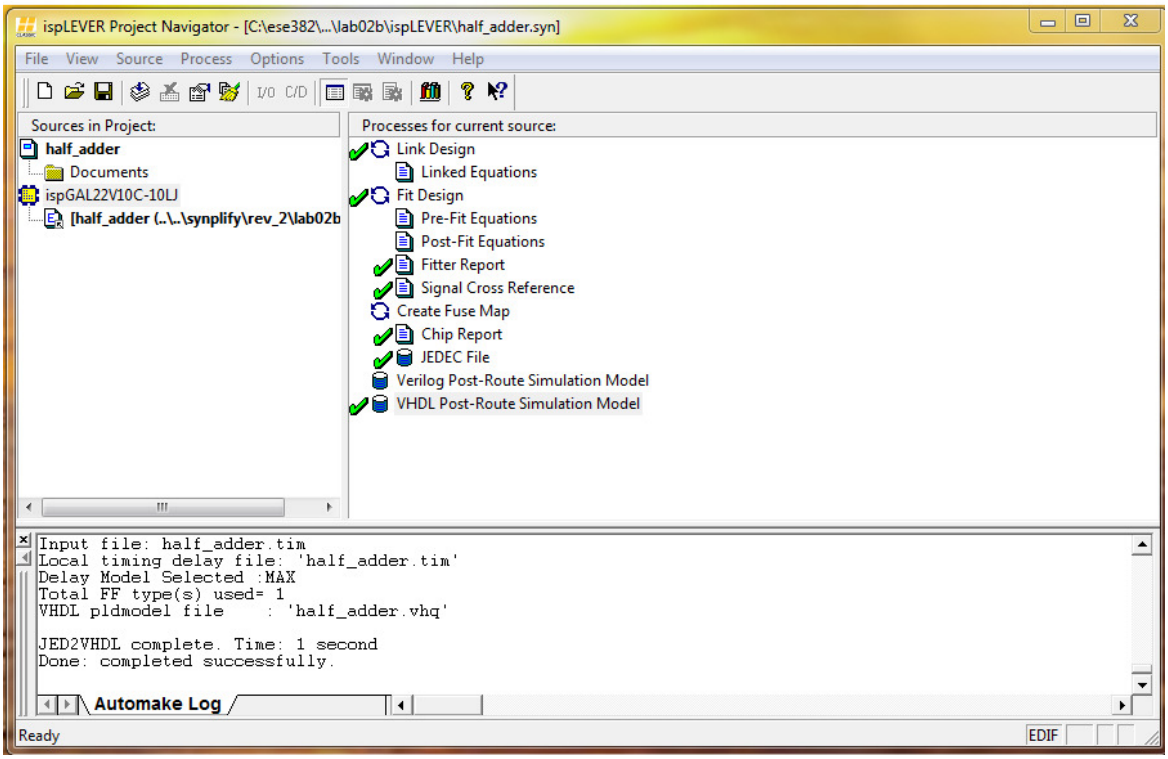


Click **Finish**.

Select the ispGAL22V10. The Sources in Project and Processes for current source information appears.



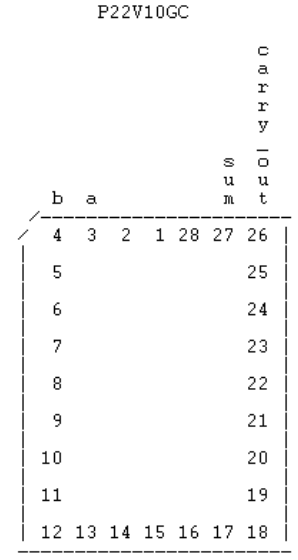
In the **Processes for current source** window double click on VHDL Post-Route Simulation Model (at the bottom of the flow). The place-and-route tool processes the half_adder.edf file and generates the timing model. The timing model is given the name half_adder.vhq. The green check marks that appear indicate that the associated steps were completed successfully.



Next, double click on **Chip Report** in the Processes for current sources window. The Chip Report will appear in the Output Window (bottom window) of the Project Navigator. Right click the mouse in this window and select **Print** to print the Chip Report. This window can also be detached from the Project Navigator and expanded for easier viewing.

Part of what you will see in the Chip Report are the equations corresponding to the outputs. These equations are written using logic symbols which may be unfamiliar to you, but are easy to figure out.

You will also see a logic symbol for the ispGAL22V10C-10LJ. Examine the logic symbol and verify that the signals were assigned to the pins specified by the attribute statements.



Verify the .jed file has been created. Recall the file name will be the same as the ispLEVER Classic project name. Close your project by selecting **Close Project** from the **File** menu. Save the changes to your project. Then close ispLEVER Classic

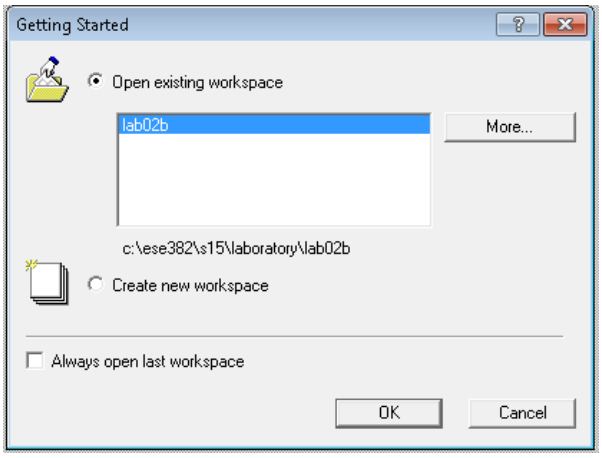
Timing Simulation

To perform a timing simulation, the UUT used by the testbench must be changed. The UUT in the functional simulation was the design entity half_adder contained in the file half_adder.vhd. This file contained our original VHDL behavioral design description of the half-adder.

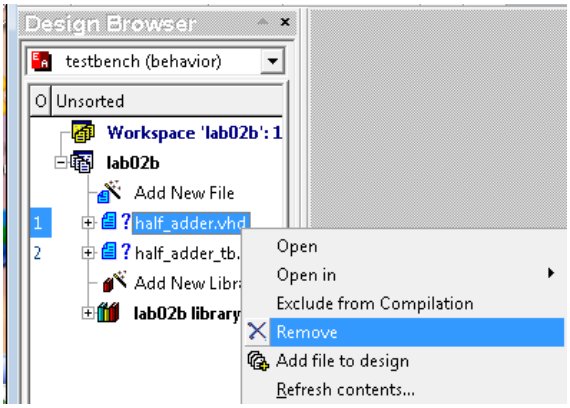
For the timing simulation, we need the VHDL structural model of the design entity half_adder that was automatically produced by the fitter (ispLEVER Classic). This design entity is contained in the file half_adder.vhq that was automatically generated and stored in the folder ispLEVER.

We must put the file half_adder.vhq into our original Active-HDL project in place of the file half_adder.vhd. This is similar to the process in Laboratory 2a, where the VHDL design description model was replaced by the VHDL netlist model.

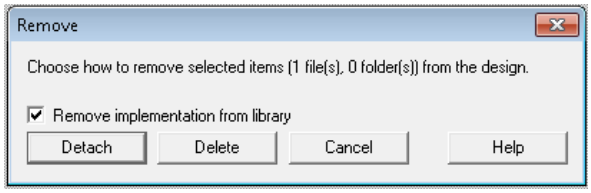
Launch Active-HDL. In the **Getting Started window** check **Open existing design** and select lab02b. Click **OK**.



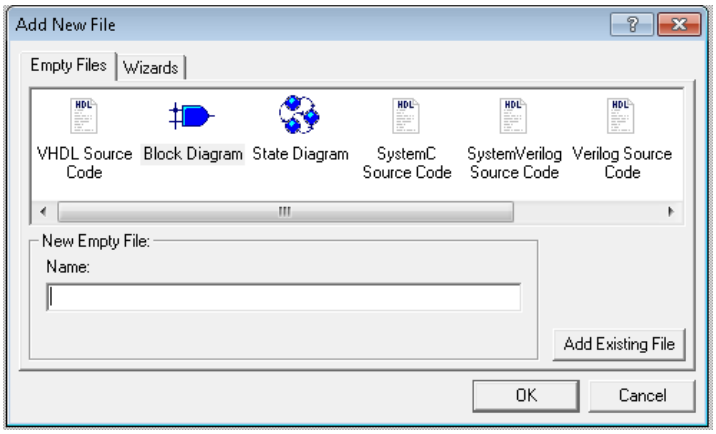
In the **Design Browser,** right click on the file half_adder.vhd. In the pop-up window click **Remove.**



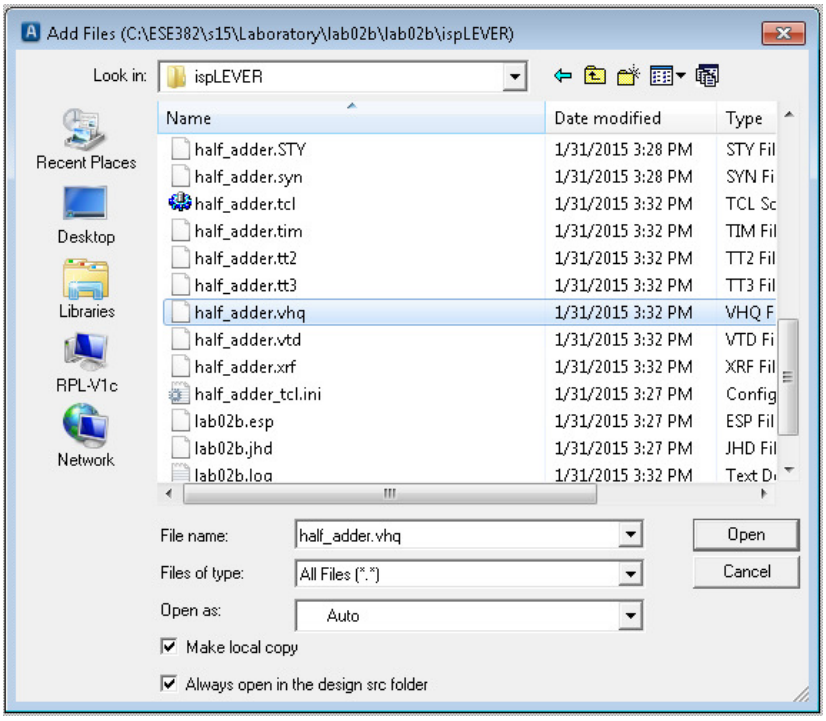
The Remove window opens. ***Do not*** click Delete. We want to ***detach*** the file half_adder.vhd from the project, click **Detach.**



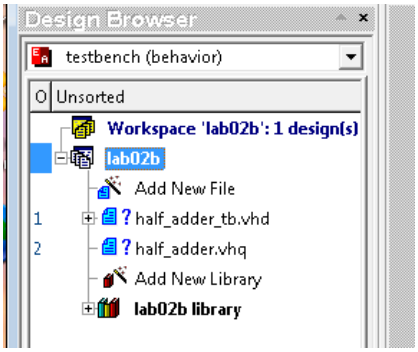
In The Design Browser, double click **Add New File**. The **Add New File** window opens. Click on **Add Existing File.**



Browse to the ispLEVER folder and click on half_adder.vhq. Then click **Add.**



The file half_adder.vhq is added to the project.



Double click on the file half_adder.vhq. The HDL editor will open. Print this file. Examine this complete file. This file is a structural (netlist) VHDL timing model of the logic that was synthesized for the half-adder fitted to the ispGAL22V10C-10. Note that the entity defined in this file is the entity half-adder. However, in this entity declaration for half-adder we have a generic clause that specifies all of the delays in the ispGAL22V10C-10.

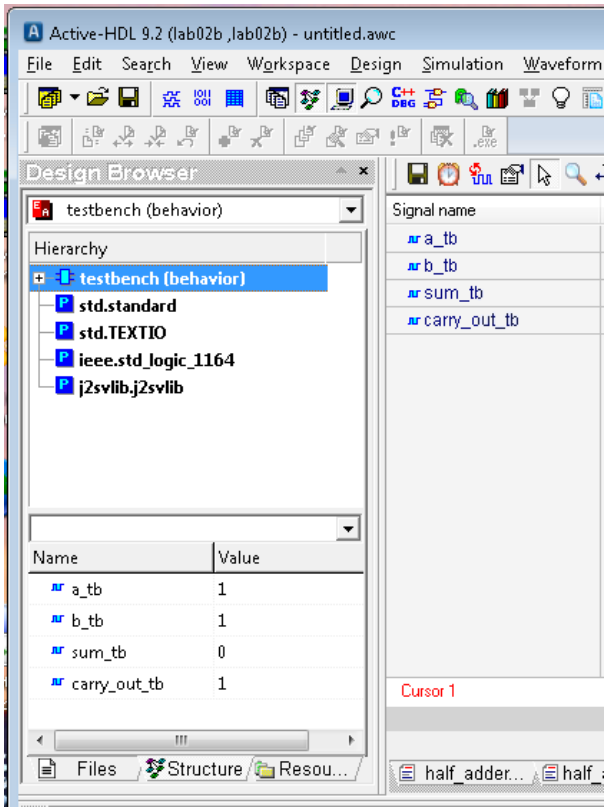
In the port clause we have the inputs and outputs that we specified in the design description assigned to the desired pins. We also have all of the other pins on the ispGAL22V10C-10 specified.

```
ENTITY half_adder IS
   GENERIC (
-- Delay Parameters:
--   MAX delay used
      tpLHO9 : time := 0 ns;
      tpHLO9 : time := 0 ns;
      tpLH11 : time := 0 ns;
      tpHL11 : time := 0 ns;
      tpLH_iob : time := 0 ns;
      tpHL_iob : time := 0 ns;
      tpLHO3 : time := 0 ns;
      tpHLO3 : time := 0 ns;
      tpLHO5 : time := 0 ns;
      tpHLO5 : time := 0 ns;
      tpLH_inco_lump : time := 10 ns;
      tpHL_inco_lump : time := 10 ns;
      tpLH_oe : time := 0 ns;
      tpHL_oe : time := 0 ns;
      tpLH_oe_pterm : time := 10 ns;
      tpHL_oe_pterm : time := 10 ns;
      t_preset : time := 0 ns;
      t_reset : time := 13 ns;
      t_cnt : time := 10 ns;
      T_ckout : time := 7 ns;
      t_setup : time := 7 ns;
      t_hold  : time := 0 ns;
      t_co    : time := 7 ns;
      t_cf    : time := 3 ns;
      tpLH_fo_lump : time:= 4 ns;
      tpHL_fo_lump : time := 4 ns);
   PORT (

    PIN02: IN STD_LOGIC := '0';
    a: IN STD_LOGIC := '0';
    b: IN STD_LOGIC := '0';
    PIN05: IN STD_LOGIC := '0';
    PIN06: IN STD_LOGIC := '0';
    PIN07: IN STD_LOGIC := '0';
    PIN09: IN STD_LOGIC := '0';
    PIN10: IN STD_LOGIC := '0';
    PIN11: IN STD_LOGIC := '0';
    PIN12: IN STD_LOGIC := '0';
    PIN13: IN STD_LOGIC := '0';
    PIN16: IN STD_LOGIC := '0';
    PIN17: INOUT STD_LOGIC;
    PIN18: INOUT STD_LOGIC;
    PIN19: INOUT STD_LOGIC;
    PIN20: INOUT STD_LOGIC;
    PIN21: INOUT STD_LOGIC;
    PIN23: INOUT STD_LOGIC;
    PIN24: INOUT STD_LOGIC;
    PIN25: INOUT STD_LOGIC;
    carry_out: OUT STD_LOGIC;
    sum: OUT STD_LOGIC);

--Pin Assignments:
    -- alias PIN03 is: STD_LOGIC a;
    -- alias PIN04 is: STD_LOGIC b;
    -- alias PIN26 is: STD_LOGIC carry_out;
    -- alias PIN27 is: STD_LOGIC sum;

END half_adder;
```

Compile the file half_adder.vhq.

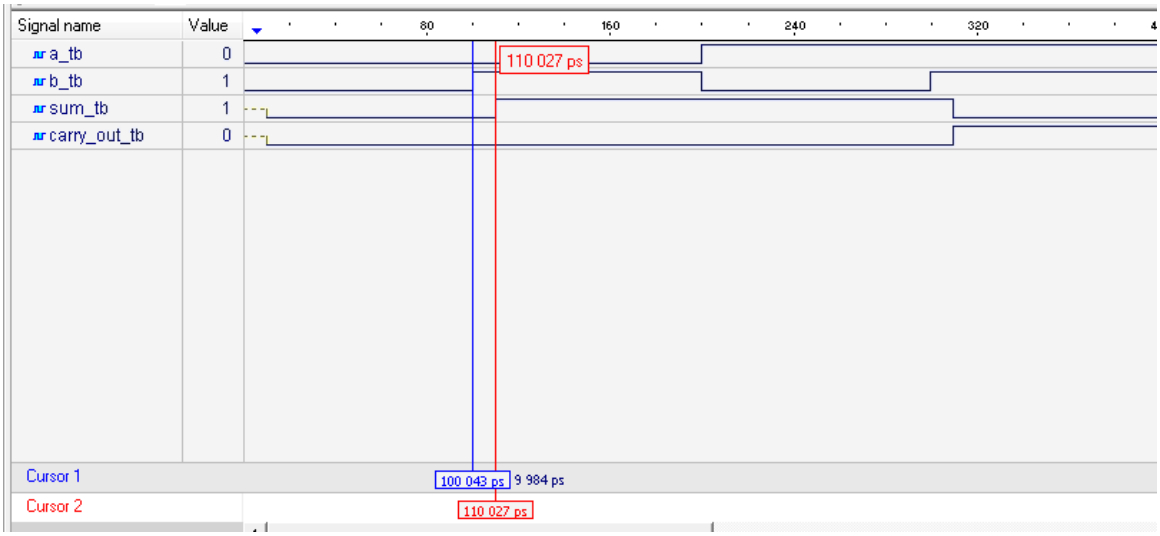
Next use the editor to open the testbench (half_adder_tb.vhd). The ispGAL22V10C-10 has a delay of 10 ns. Change the value of the constant period in the testbench from 20 ns to 100 ns. Compile the testbench.

Next select the **Structure** tab at the bottom of the **Design Browser.** Follow the procedure from Laboratory 1 to perform a simulation. This simulation will be a timing simulation, because the model of half-adder that we are simulating is a timing model (in the file half_adder.vhq) that has internal delays for the PLD specified.



Initialize and run the simulation for 400 ns. Examine the timing waveforms. Note that when input `b_tb` first changes from 0 to 1, that output sum does not change from 0 to 1 immediately. Instead there is a delay of 10 ns.

Normally, there is a single waveform cursor in the waveform window that you can move to any point in the window you would like, to determine the simulated time at that point. If you right click in the waveform window, you can then select **Add Cursor** to add a second cursor. You can then position the cursors so that the time between any two points is displayed. In the timing waveform that follows, the waveform cursors were used to display the exact delay between the second transition of `b_tb` from 0 to 1 and the transition of `sum_tb` from 1 to 0.



Modify the testbench so that `a_tb` remains constant at logic 1 and `b_tb` is a square wave with a 200 ns period. Before running the timing simulation, draw out the waveforms you expect from the simulation.

Recompile the testbench and carry out a new timing simulation. Verify that the waveforms are as expected. If you have any errors, modify your design description or testbench as appropriate to remove the errors.

Programming the ispGAL22V10C-10

Follow the instructions provided to you in the laboratory to program an ispGAL22V10C-10LJ using the .jed file. Test the ispGAL22V10C-10LJ using the test station provided and verify that it operates as specified.

Questions

1. What model of the half-adder design entity is used for timing simulation? What is the source of this model?

2. Why does performing a timing simulation obviate the need to perform a post synthesis simulation?

3. What is the purpose of the loc attributes in the entity declaration? What do the simulation and synthesis tools do with the loc attributes?

4. What does the place-and-route tool do with the loc attributes?

5. How are propagation delays denoted in the file half_adder.vhq?