EGR 270
Fundamentals of Computer Engineering
Filename: Tutorial 1 - Aldec Active HDL and Xilinx Vivado

Tutorial 1: Combinational Logic Circuits using Aldec Active-HDL and Xilinx Vivado

Aldec Active-HDL software can be used to generate and simulate designs using VHDL. Aldec is not targeted to any one specific manufacturer of programmable logic devices (PLDs) or Field Programmable Gate Arrays (FPGAs), but produces a vhd file that can be implemented (or "synthesized") into various devices using various synthesis tools. We will use Xilinx Vivado to synthesize our design into the Xilinx Artix-7 FPGA. This FPGA is conveniently mounted on an FPGA board called by BASYS3 by Digilent. The BASYS3 provides easy access to input and output pins; slide switches and pushbutton switches for inputs; LEDs and 7-segment displays for outputs; USB and video ports; and PMOD connectors for interfacing with other devices.

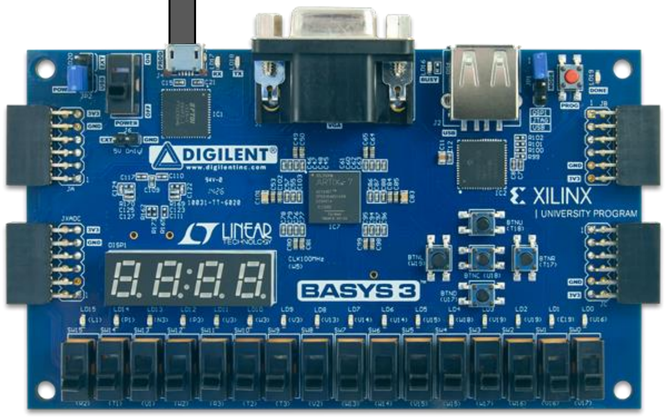
Aldec Active-HDL
- Define inputs and outputs
- Describe functional operation
- Create testbench and test design
- Generate truth tables and waveforms
- Compile and produce vhd file

vhd file (e.g., MyFile.vhd)

Xilinx Vivado
- Specify FPGA Package (Xilinx Artix-7, etc)
- Assign signals to pins
- Implement design
- Generate reports
- Produce bitstream for FPGA (bit file or bin file)
- Program the FPGA

bit file (e.g., MyFile.bit) or bin file (e.g., MyFile.bin)

USB cable



Digilent BASYS3 FPGA Board

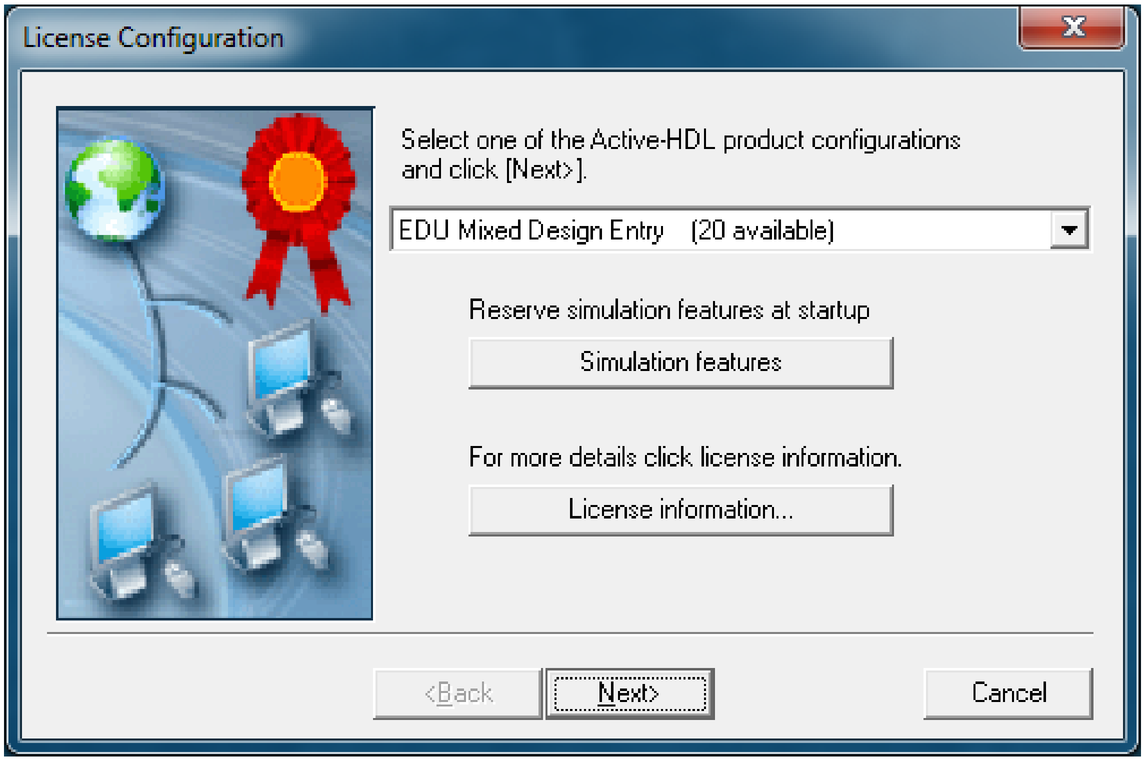
This tutorial will guide you through:

- Using Aldec Active-HDL software to:
  - Enter and compileVHDL code for a simple combinational logic circuit
  - Generate a testbench and simulate the design
  - Produce truth tables and waveforms verifying proper operation
- Using Xilinx Vivado software to:
  - Specify the type of FPGA to be programmed
  - Specify the source file (VHDL file just created using Aldec)
  - Create a configuration file that assigns input and output signals to FPGA pins
  - Implement the design
  - Create a bitstream using two methods:
    - Method A (Create a bit file to program the FPGA directly, but the design is lost when the BASYS3 is powered down.)
    - Method B (Create a bin file to program the onboard flash memory. The FPGA is reprogrammed every time the BASYS3 board is powered up.)
  - Program the FPGA
  - Generate reports
- Testing the design on the BASYS3 board.

Warning: Aldec and Xilinx can be particular about file names and folder names. Stick to using letters, numbers, and underscores! Avoid filenames such as \Lab 5\Lab#5.aws

1. Creating a Project with Aldec Active-HDL

- Launch **Aldec Active-HDL**
- The screen below should appear. Select **Next**.


License Configuration

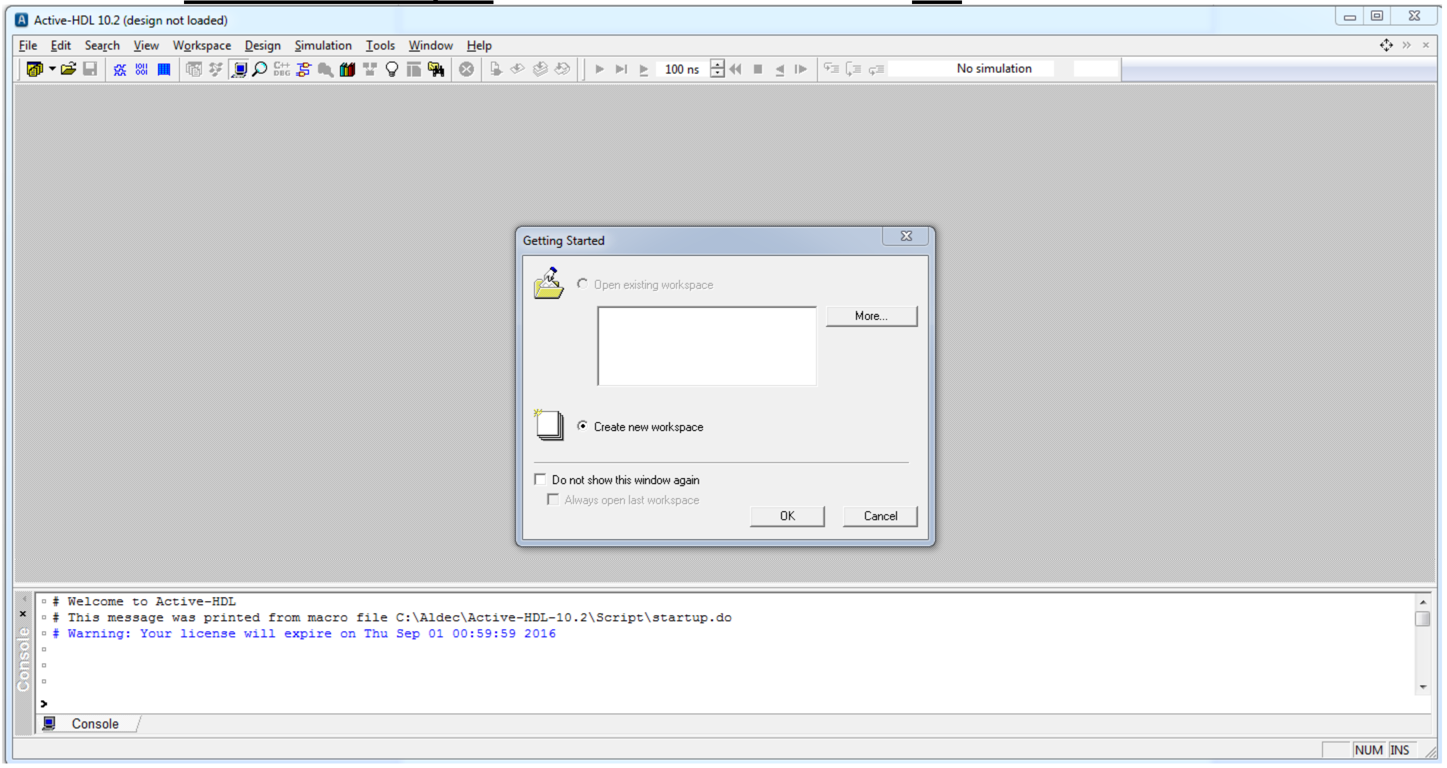
Select one of the Active-HDL product configurations and click [Next>].

EDU Mixed Design Entry (20 available)

Reserve simulation features at startup

For more details click license information.

- Select **Create new workspace** as shown below and then select **OK**.


Active-HDL 10.2 (design not loaded)
File Edit Search View Workspace Design Simulation Tools Window Help
100 ns
No simulation
Getting Started
More...
NUM INS

Open existing workspace

Create new workspace [✓]

Do not show this window again

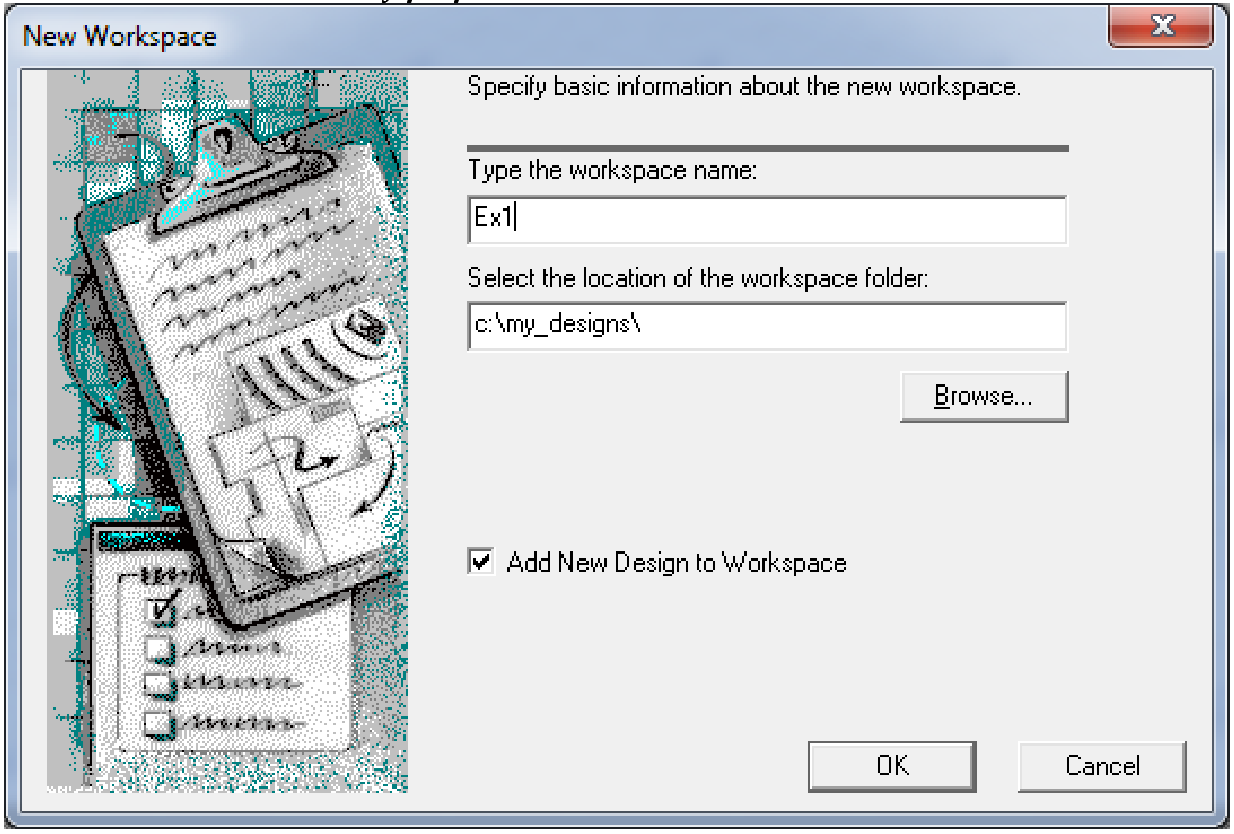
Always open last workspace

OK Cancel

```
# Welcome to Active-HDL
# This message was printed from macro file C:\Aldec\Active-HDL-10.2\Script\startup.do
# Warning: Your license will expire on Thu Sep 01 00:59:59 2016
```

Console

- Enter a name for the workspace (**Ex1** was entered below), change the location of the workspace folder (or use the default as below), and select **OK**. Note that the name for the project workspace, VHDL entity (to be entered later), and architecture (to be entered later) should be the same. ***Use letters, numbers and underscores only for filenames.***



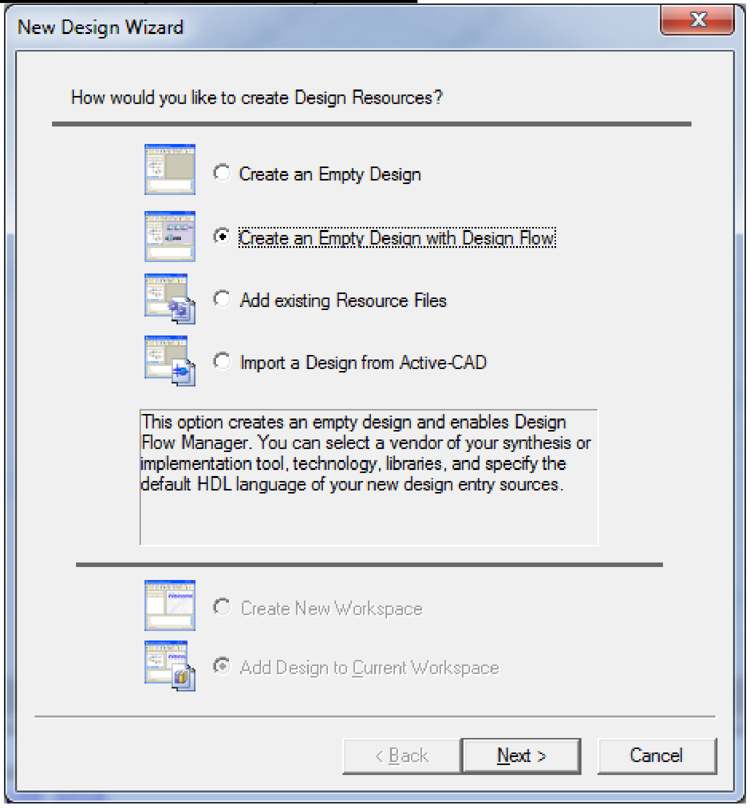
Specify basic information about the new workspace.

Type the workspace name: Ex1

Select the location of the workspace folder: c:\my_designs\

[✓] Add New Design to Workspace

- Select **Create an Empty Design with Design Flow** as shown below and select **Next**.



New Design Wizard

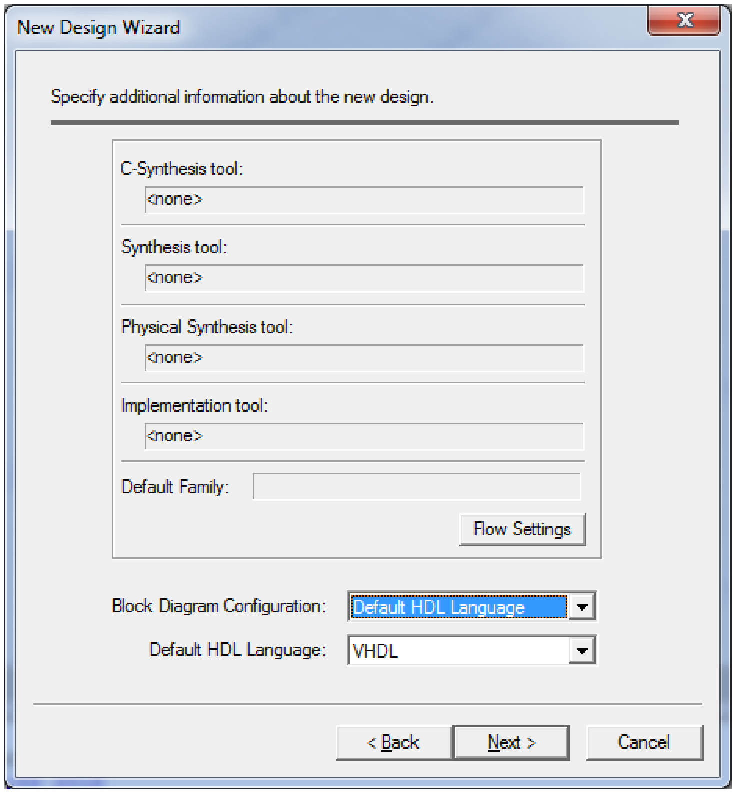
How would you like to create Design Resources?

( ) Create an Empty Design
(•) Create an Empty Design with Design Flow
( ) Add existing Resource Files
( ) Import a Design from Active-CAD

This option creates an empty design and enables Design Flow Manager. You can select a vendor of your synthesis or implementation tool, technology, libraries, and specify the default HDL language of your new design entry sources.

( ) Create New Workspace
(•) Add Design to Current Workspace

- The next window that appears shows information about synthesis tools and implementation tools that might be configured to launch automatically from Aldec Active-HDL. Synthesis tools can also be launched separately rather than integrating them into Aldec. We will launch Xilinx Vivado separately, so you can ignore most settings shown. Select **VHDL** for the Default HDL Language and then select **Next**.



New Design Wizard

Specify additional information about the new design.

C-Synthesis tool: <none>

Synthesis tool: <none>

Physical Synthesis tool: <none>

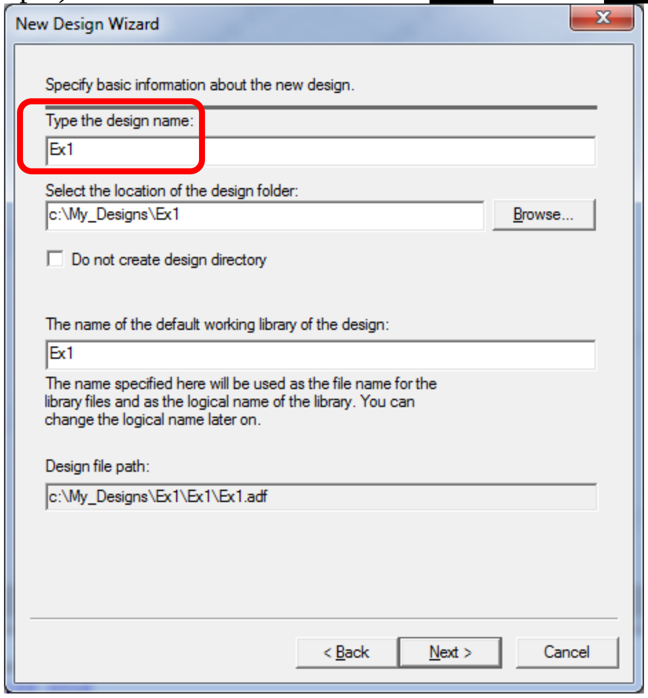
Implementation tool: <none>

Default Family:

Block Diagram Configuration: Default HDL Language

Default HDL Language: VHDL

- Enter the design name. Note that it should match the workspace name used earlier (Ex1 for this example). Enter the name and select **<u>Next</u>**. Select **<u>Finish</u>** on the final screen shown below.



New Design Wizard

Specify basic information about the new design.

Type the design name: Ex1

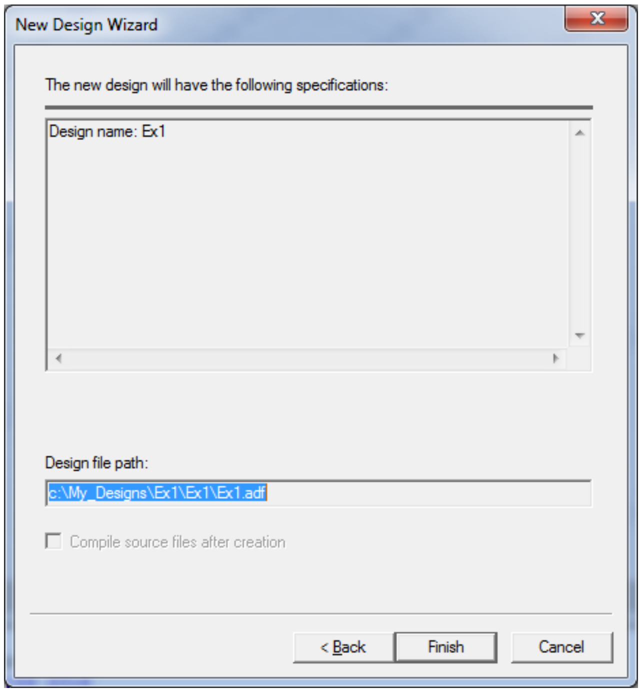
Select the location of the design folder: c:\My_Designs\Ex1

☐ Do not create design directory

The name of the default working library of the design: Ex1

The name specified here will be used as the file name for the library files and as the logical name of the library. You can change the logical name later on.

Design file path: c:\My_Designs\Ex1\Ex1\Ex1.adf



New Design Wizard

The new design will have the following specifications:

Design name: Ex1

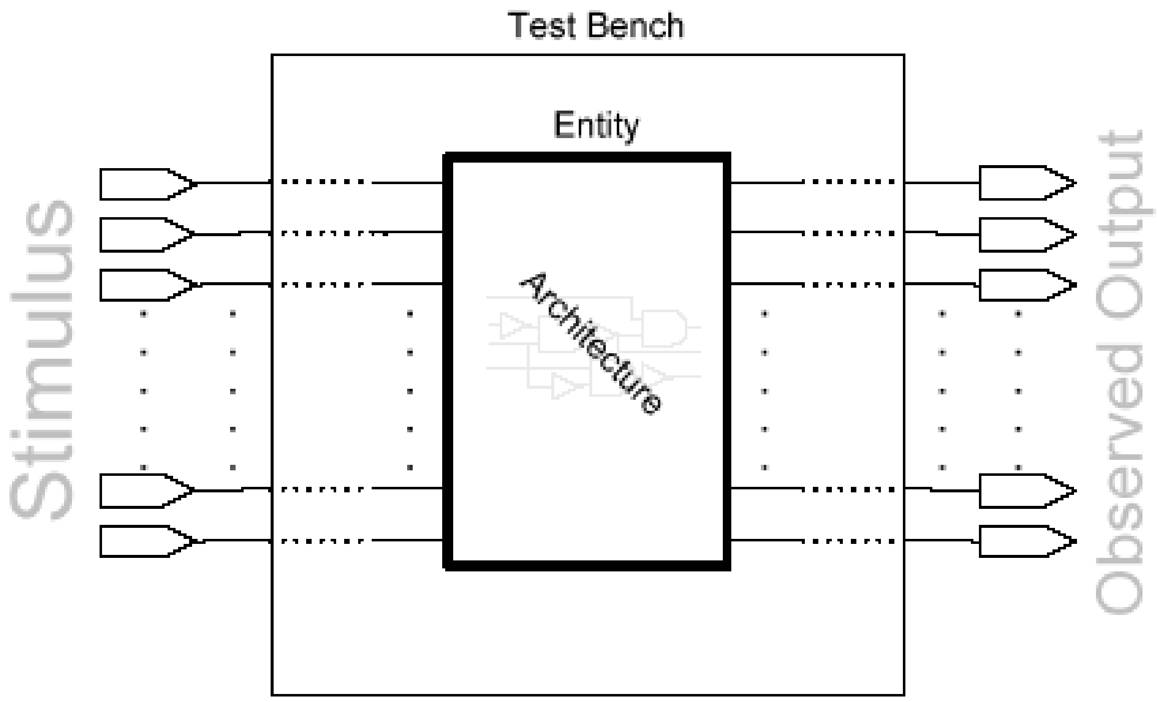
Design file path: c:\My_Designs\Ex1\Ex1\Ex1.adf

☐ Compile source files after creation

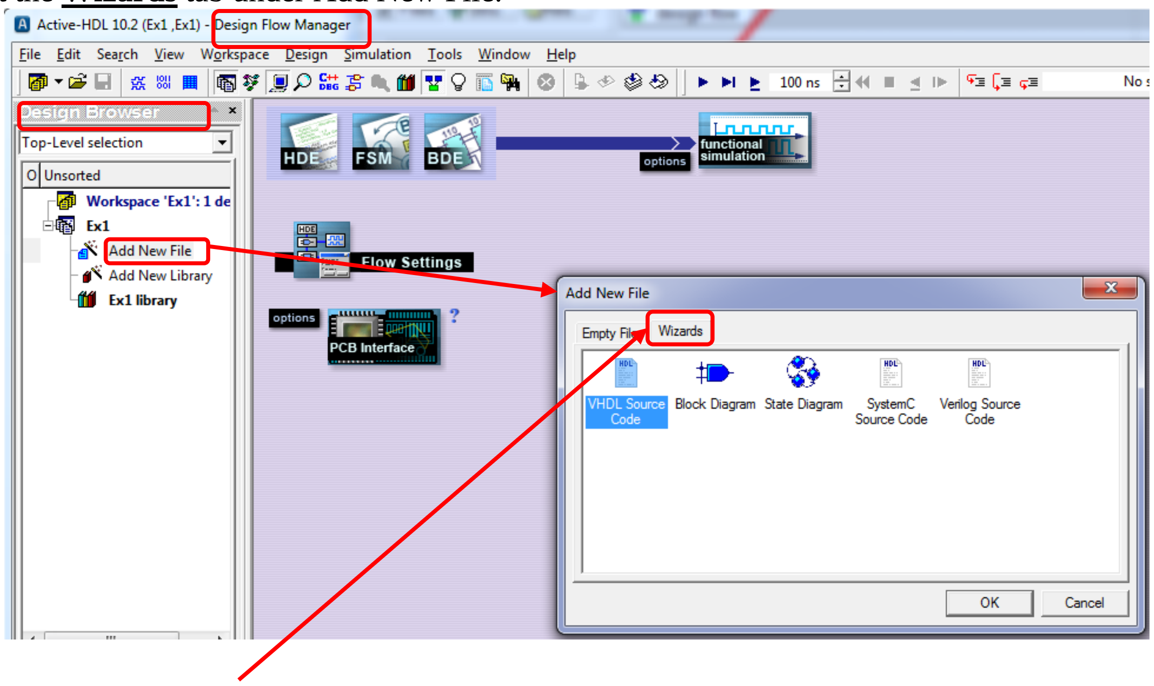
2. <u>Adding VHDL code to your design</u>

- Before adding VHDL code to the design, recall that VHDL designs typically have three parts:
  - **<u>Entity</u>** – basically defines the inputs and outputs to a black box
  - **<u>Architecture</u>** – defines the function of the black box
  - **<u>Testbench</u>** - defines stimulus signals as inputs to the design and allows us to observe the outputs (typically in the form of truth tables or timing diagrams).

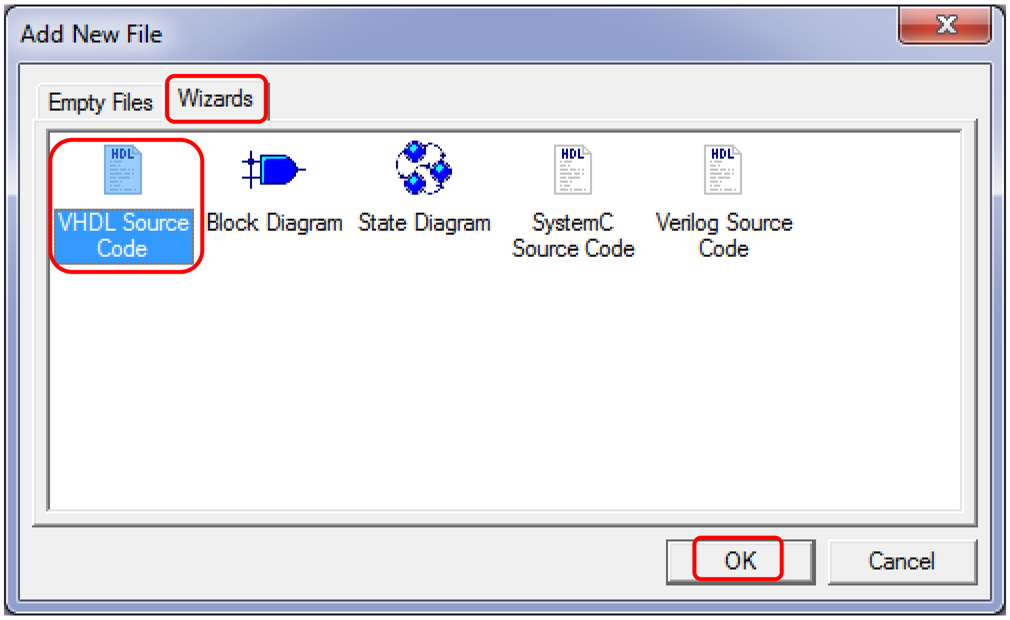
- The diagram below illustrates the relationship between the entity, architecture, and testbench.



- The Design Flow Manager screen should appear next.
- The Design Browser should appear on the left of the screen. If it does not appear, it can be toggled on and off using Alt + 1.
- Double-click on **Add New File** under the Design Browser and the Add New File window should appear.
- Note that there are several types of files that can be specified (VHDL Source Code, Block Diagram, etc). Additionally, some wizards are available to allow for easier entry of design information.
- Select the **Wizards** tab under Add New File.

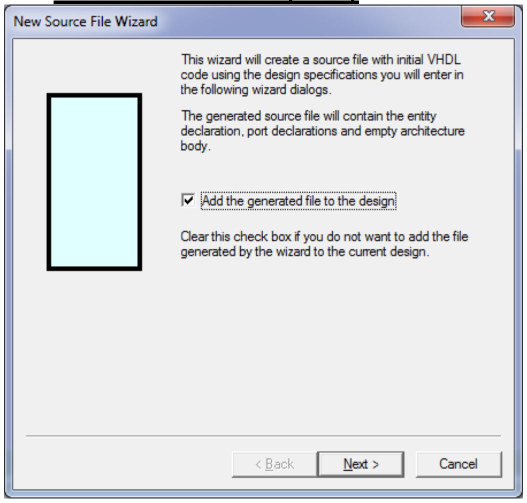


- After selecting the **Wizards** tab, select **VHDL Source Code** and then select **OK**.



The ***VHDL Source File Wizard*** is used to specify the inputs and outputs that will be used in the entity and architecture sections of the VHDL file to be created (Ex1.vhd). If the function to be implemented is F(A,B,C,D) = Σ(0,1,4,5,7,8,10,14) = A'C' + A'BD + ACD' + B'C'D', then we will need 4 inputs (A, B, C, D) and one output (F).

- Select **Next** when the screen below on the left appears.
- Enter the **source file name (Ex1)** in the window shown below on the right and select **Next**.

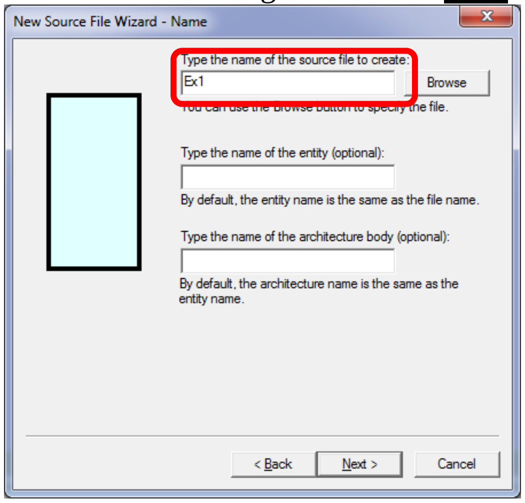

New Source File Wizard

This wizard will create a source file with initial VHDL code using the design specifications you will enter in the following wizard dialogs.

The generated source file will contain the entity declaration, port declarations and empty architecture body.

[✓] Add the generated file to the design

Clear this check box if you do not want to add the file generated by the wizard to the current design.


New Source File Wizard - Name

Type the name of the source file to create: Ex1

You can use the Browse button to specify the file.

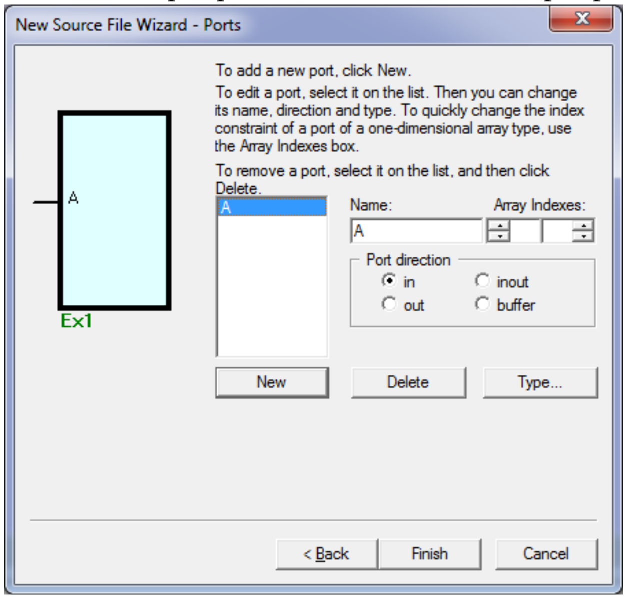
Type the name of the entity (optional):

By default, the entity name is the same as the file name.

Type the name of the architecture body (optional):

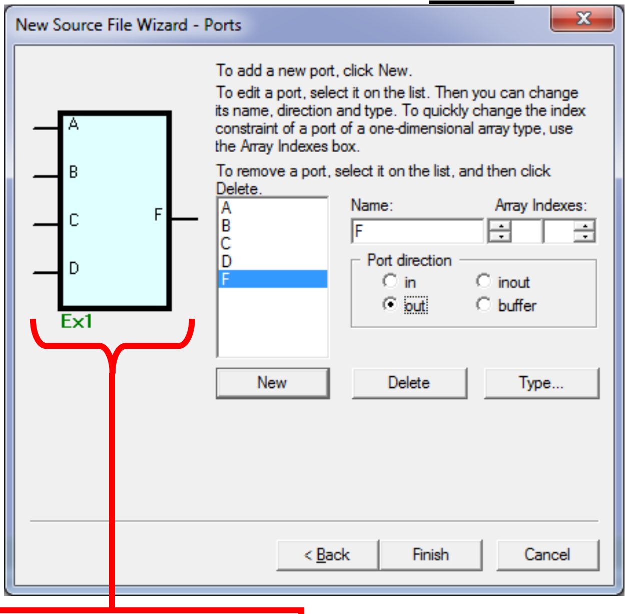
By default, the architecture name is the same as the entity name.

- Select **New** in the window below on the left to add a new port. Name it **A** with direction **in**.
- Also add input ports B, C, and D and output port F as shown below and then select **Finish**.


New Source File Wizard - Ports

To add a new port, click New.
To edit a port, select it on the list. Then you can change its name, direction and type. To quickly change the index constraint of a port of a one-dimensional array type, use the Array Indexes box.
To remove a port, select it on the list, and then click Delete.

Name: A
Array Indexes:
Port direction: [✓ in] out inout buffer

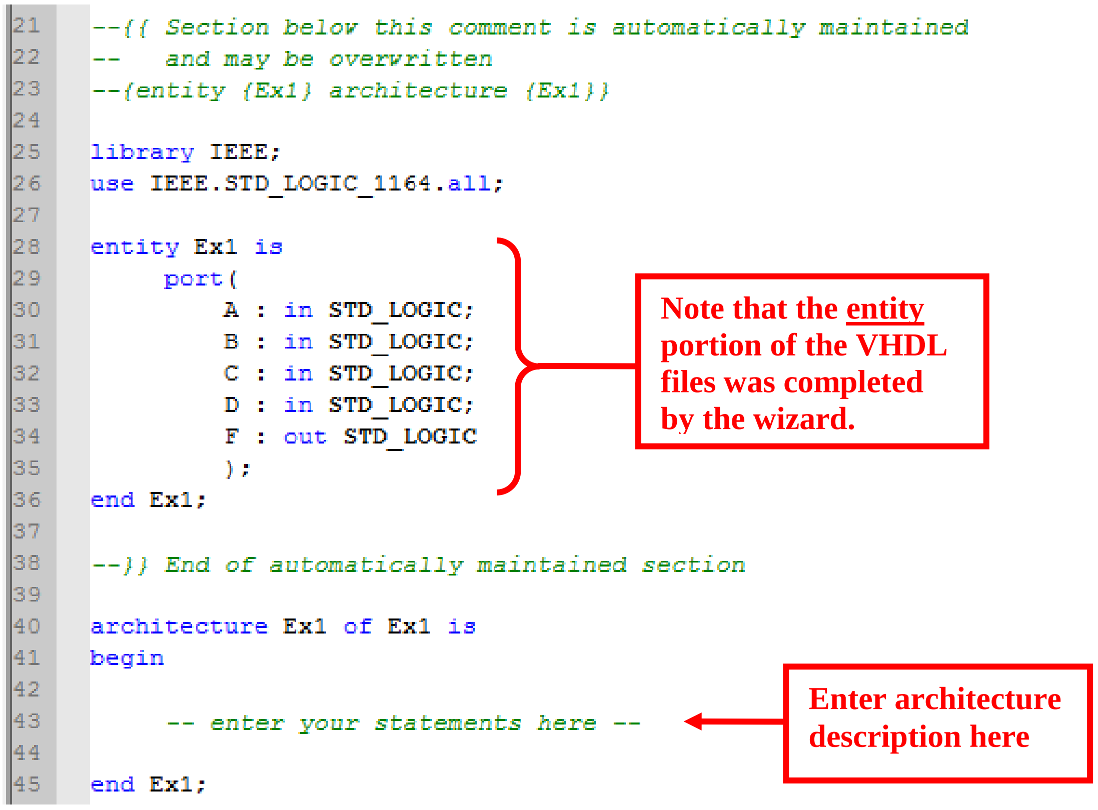

New Source File Wizard - Ports

To add a new port, click New.
To edit a port, select it on the list. Then you can change its name, direction and type. To quickly change the index constraint of a port of a one-dimensional array type, use the Array Indexes box.
To remove a port, select it on the list, and then click Delete.

Name: F
Array Indexes:
Port direction: in [✓ out] inout buffer

The diagram should clearly illustrate your inputs and outputs.

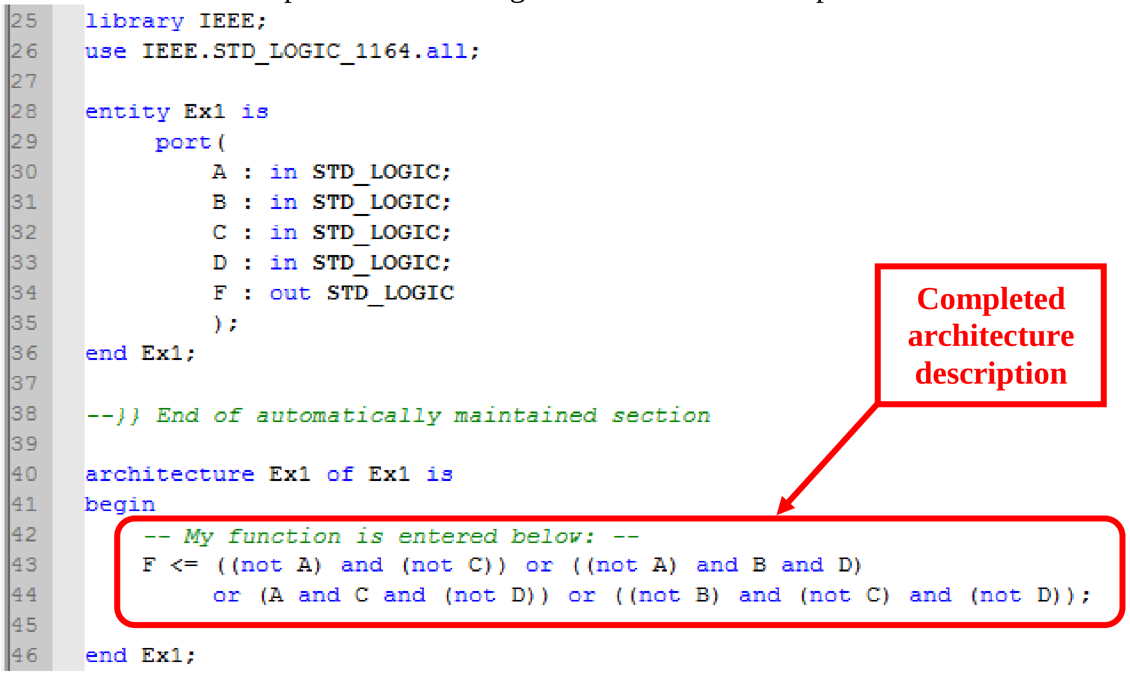
- Note that a VHDL file (Ex1.vhd) has now been generated with a complete entity section and the shell of the architecture section. Only the architecture description (Boolean equation for F in this example) needs to be entered.



```
--{{ Section below this comment is automatically maintained
--   and may be overwritten
--{entity {Ex1} architecture {Ex1}}

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Ex1 is
     port(
          A : in STD_LOGIC;
          B : in STD_LOGIC;
          C : in STD_LOGIC;
          D : in STD_LOGIC;
          F : out STD_LOGIC
          );
end Ex1;

--}} End of automatically maintained section

architecture Ex1 of Ex1 is
begin

     -- enter your statements here --

end Ex1;
```

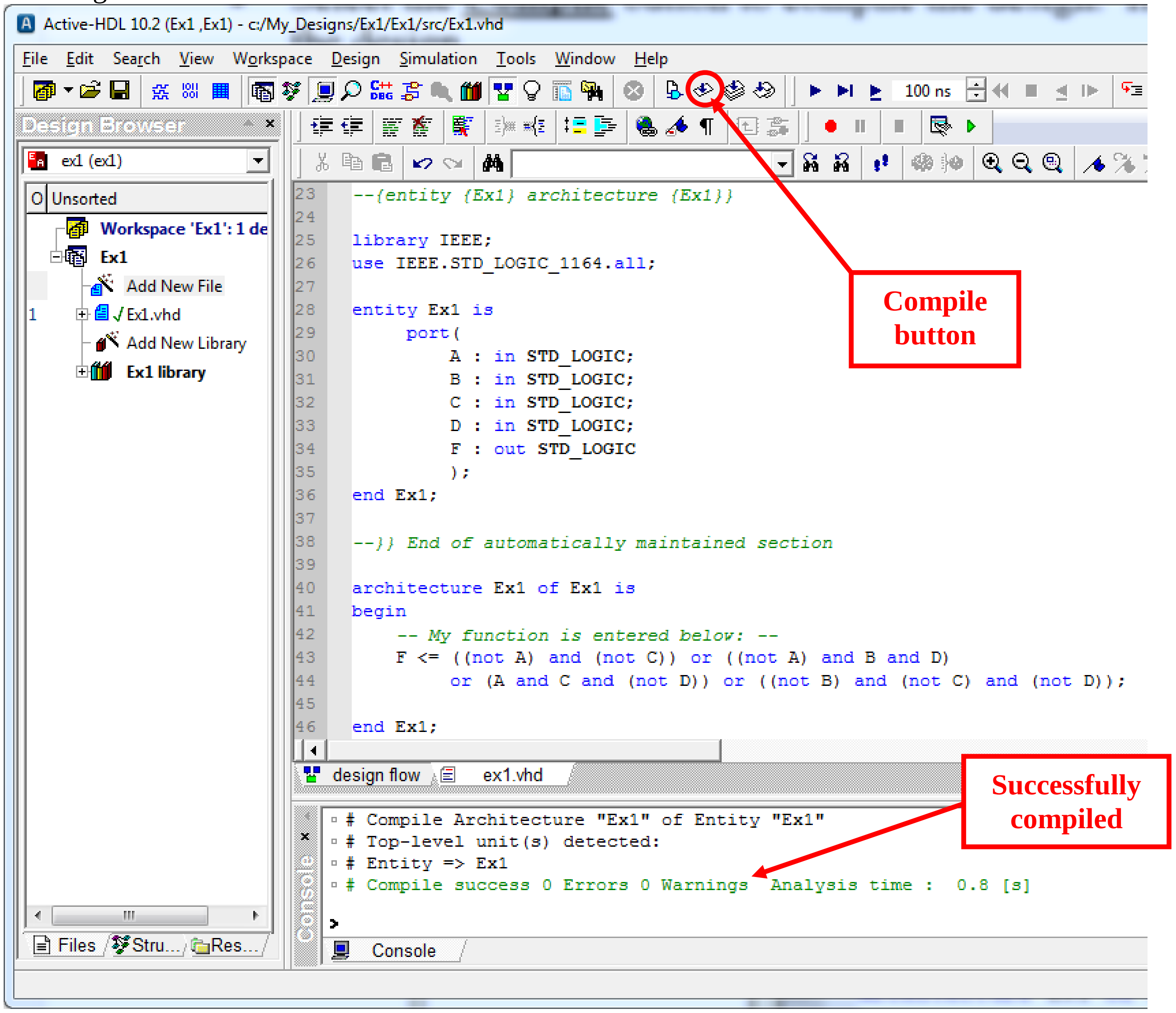
Note that the entity portion of the VHDL files was completed by the wizard.
Enter architecture description here

- The VHDL file is complete after entering the architecture description as shown below.



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Ex1 is
     port(
          A : in STD_LOGIC;
          B : in STD_LOGIC;
          C : in STD_LOGIC;
          D : in STD_LOGIC;
          F : out STD_LOGIC
          );
end Ex1;

--}} End of automatically maintained section

architecture Ex1 of Ex1 is
begin
    -- My function is entered below: --
    F <= ((not A) and (not C)) or ((not A) and B and D)
         or (A and C and (not D)) or ((not B) and (not C) and (not D));

end Ex1;
```

- Select the **Compile** button to compile the design. If there were any errors, correct them and recompile the design.

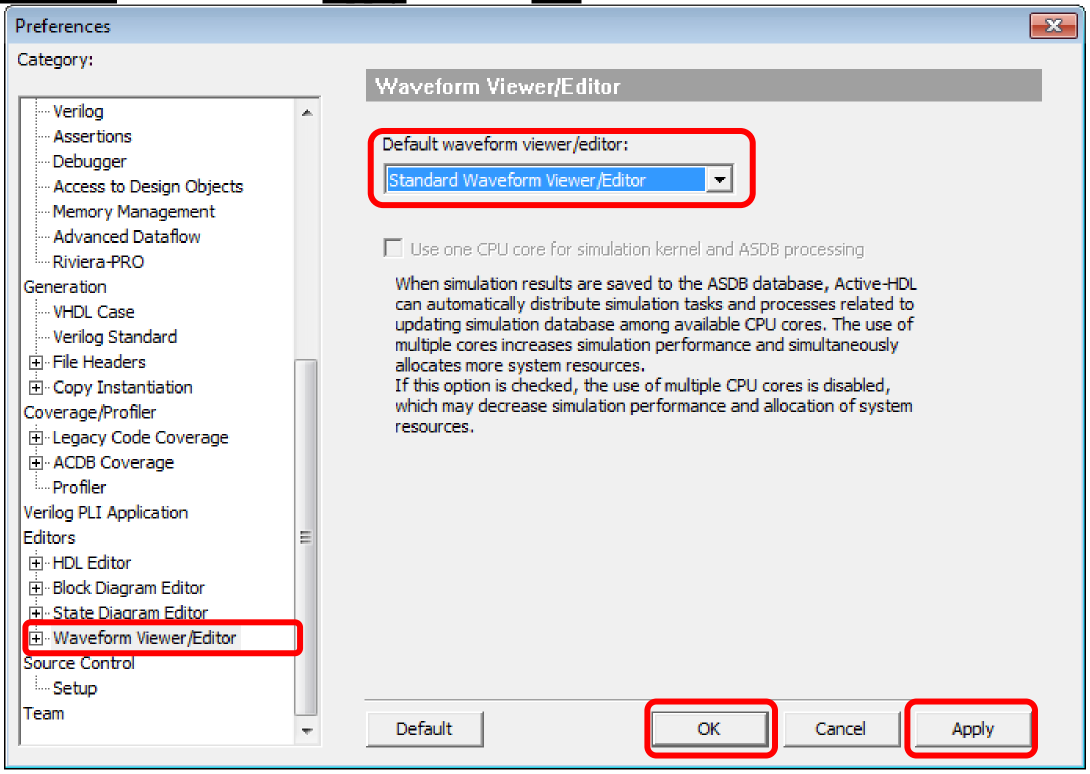


```
--{entity {Ex1} architecture {Ex1}}

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Ex1 is
      port(
            A : in STD_LOGIC;
            B : in STD_LOGIC;
            C : in STD_LOGIC;
            D : in STD_LOGIC;
            F : out STD_LOGIC
            );
end Ex1;

--}} End of automatically maintained section

architecture Ex1 of Ex1 is
begin
    -- My function is entered below: --
    F <= ((not A) and (not C)) or ((not A) and B and D)
         or (A and C and (not D)) or ((not B) and (not C) and (not D));

end Ex1;
```

```
# Compile Architecture "Ex1" of Entity "Ex1"
# Top-level unit(s) detected:
# Entity => Ex1
# Compile success 0 Errors 0 Warnings  Analysis time :  0.8 [s]
>
```

3. Simulating VHDL Code

- We have already defined the ***entity*** and the ***architecture*** for the design. Now we need to define the ***testbench*** to simulate the design to see if it works correctly.
- Selecting the following option will make later printing of truth tables and waveforms easier. Select **Tools – Preferences** and the Preferences window below should appear as shown below. Select **Waveform Viewer/Editor** and change the default waveform viewer/editor to **Standard Waveform Viewer/Editor** and then select **Apply** and then **OK**.



Preferences

Category:

Verilog
Assertions
Debugger
Access to Design Objects
Memory Management
Advanced Dataflow
Riviera-PRO
Generation
VHDL Case
Verilog Standard
File Headers
Copy Instantiation
Coverage/Profiler
Legacy Code Coverage
ACDB Coverage
Profiler
Verilog PLI Application
Editors
HDL Editor
Block Diagram Editor
State Diagram Editor
Waveform Viewer/Editor
Source Control
Setup
Team

Waveform Viewer/Editor

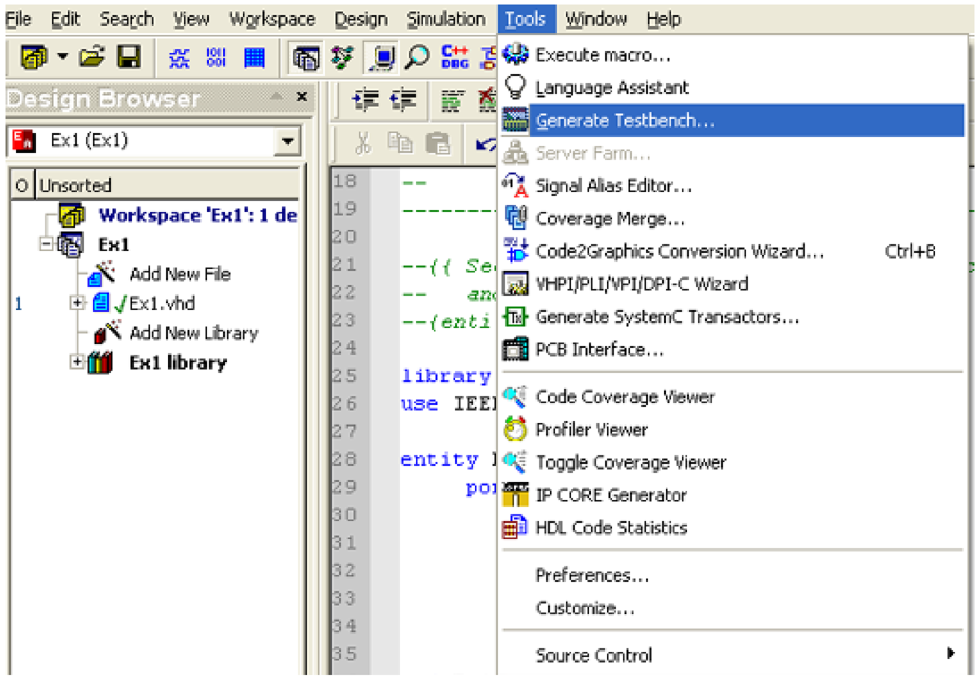
Default waveform viewer/editor: Standard Waveform Viewer/Editor

Use one CPU core for simulation kernel and ASDB processing

When simulation results are saved to the ASDB database, Active-HDL can automatically distribute simulation tasks and processes related to updating simulation database among available CPU cores. The use of multiple cores increases simulation performance and simultaneously allocates more system resources.
If this option is checked, the use of multiple CPU cores is disabled, which may decrease simulation performance and allocation of system resources.

Default | OK | Cancel | Apply

- To generate a testbench for your design, go to the **Tools** menu and select **Generate Testbench** as shown below.



File Edit Search View Workspace Design Simulation Tools Window Help

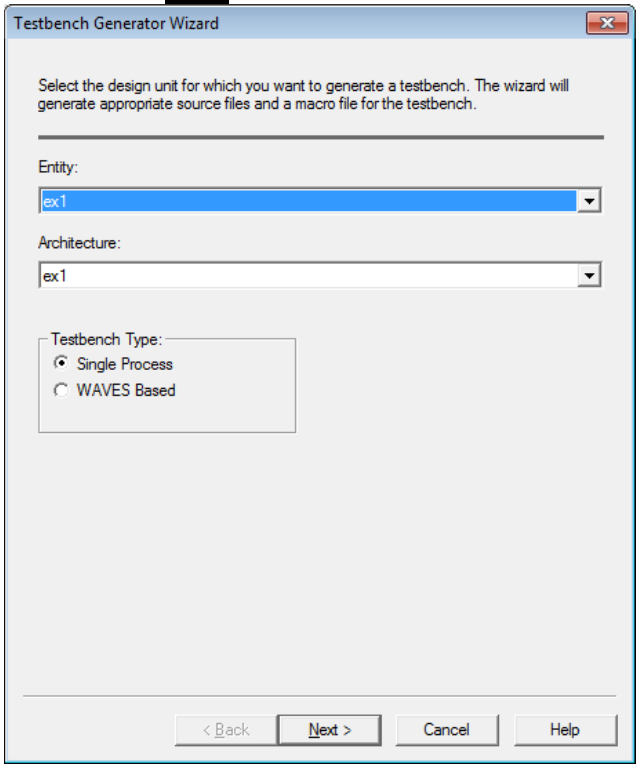
Design Browser

Ex1 (Ex1)

Unsorted
Workspace 'Ex1': 1 de
Ex1
Add New File
Ex1.vhd
Add New Library
Ex1 library

Execute macro...
Language Assistant
Generate Testbench...
Server Farm...
Signal Alias Editor...
Coverage Merge...
Code2Graphics Conversion Wizard... Ctrl+B
VHPI/PLI/VPI/DPI-C Wizard
Generate SystemC Transactors...
PCB Interface...
Code Coverage Viewer
Profiler Viewer
Toggle Coverage Viewer
IP CORE Generator
HDL Code Statistics
Preferences...
Customize...
Source Control

- Select the entity name and the architecture name in the window shown below on the left and then select **Next**. Also select **Next** in the window shown below on the right.

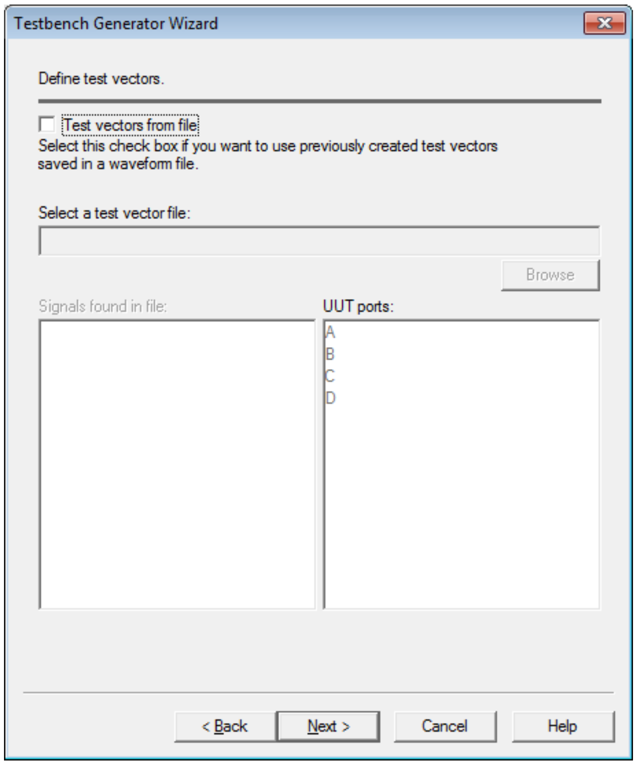

Testbench Generator Wizard

Select the design unit for which you want to generate a testbench. The wizard will generate appropriate source files and a macro file for the testbench.

Entity: ex1

Architecture: ex1

Testbench Type: [✓ Single Process] WAVES Based


Testbench Generator Wizard

Define test vectors.

Test vectors from file

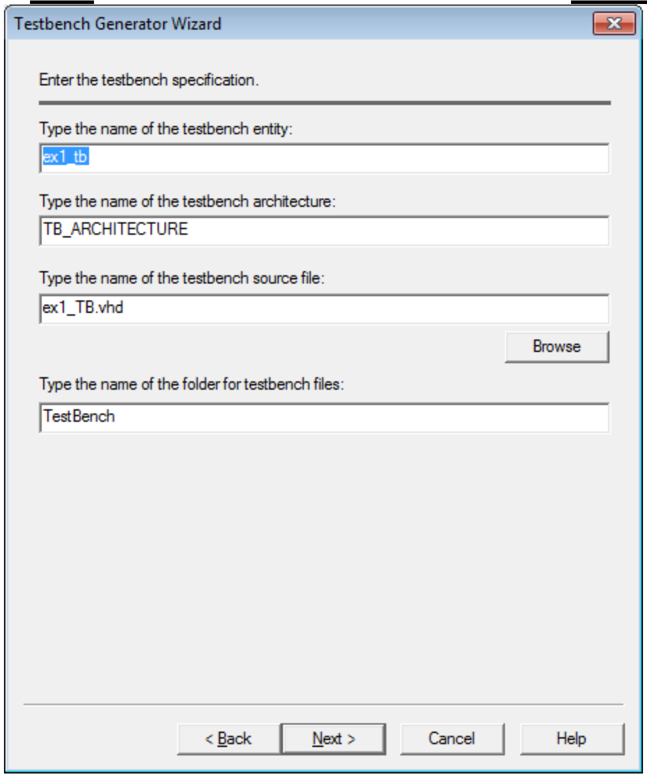
Select this check box if you want to use previously created test vectors saved in a waveform file.

Select a test vector file:

Signals found in file:

UUT ports:
A
B
C
D

- Select **Next** in the window below on the left and **Finish** in the window below on the right.


Testbench Generator Wizard

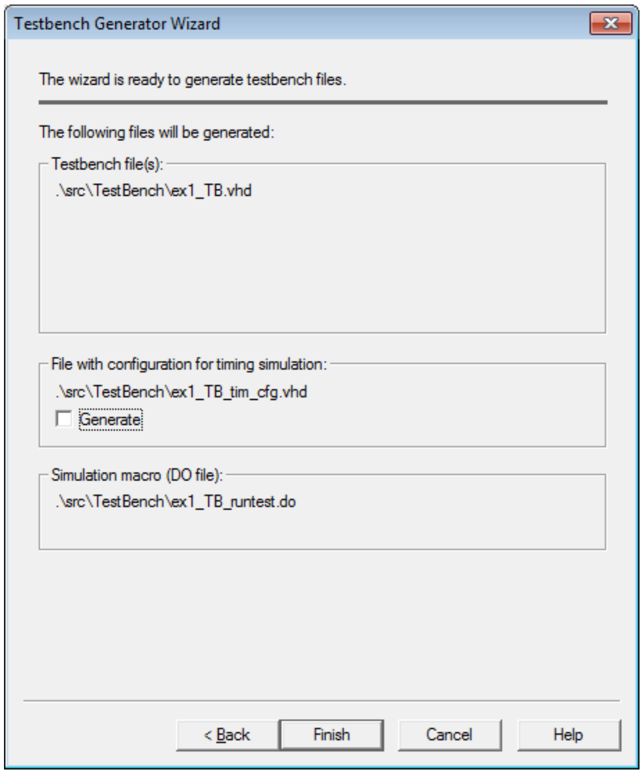
Enter the testbench specification.

Type the name of the testbench entity: ex1_tb

Type the name of the testbench architecture: TB_ARCHITECTURE

Type the name of the testbench source file: ex1_TB.vhd

Type the name of the folder for testbench files: TestBench


Testbench Generator Wizard

The wizard is ready to generate testbench files.

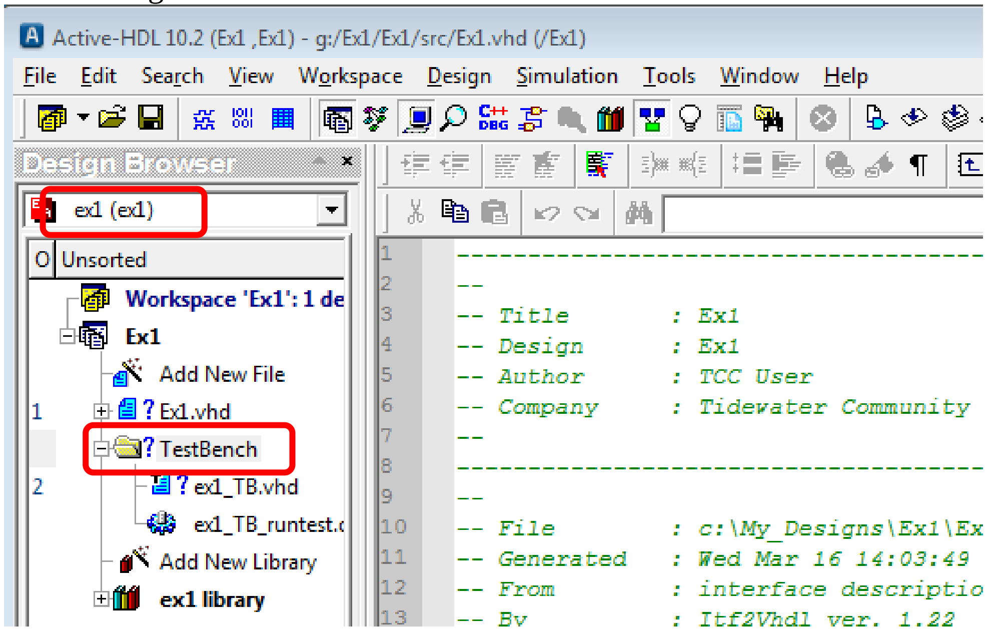
The following files will be generated:

Testbench file(s):
.\src\TestBench\ex1_TB.vhd

File with configuration for timing simulation:
.\src\TestBench\ex1_TB_tim_cfg.vhd
Generate

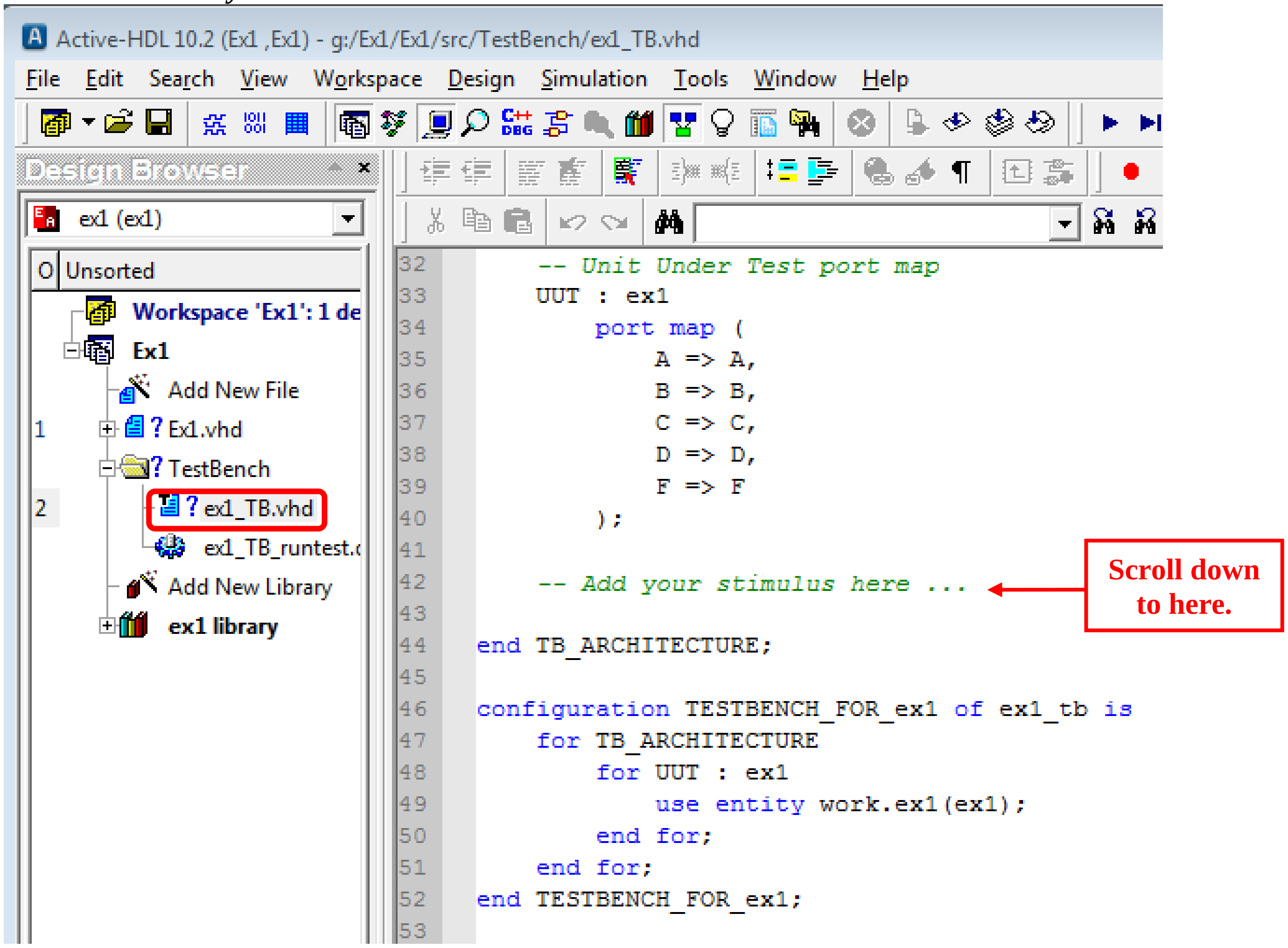
Simulation macro (DO file):
.\src\TestBench\ex1_TB_runtest.do

- Change the Design Browser so that ex1 (ex1) is displayed and expand the TestBench folder (click on the + symbol) in the Design Browser as illustrated below.



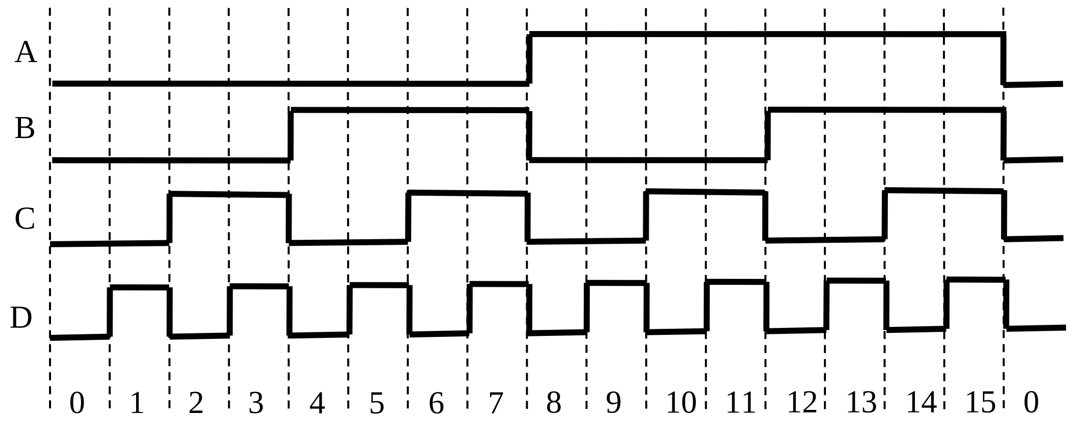
```
------------------------------------------
--
-- Title        : Ex1
-- Design       : Ex1
-- Author       : TCC User
-- Company      : Tidewater Community
--
------------------------------------------
--
-- File         : c:\My_Designs\Ex1\Ex
-- Generated    : Wed Mar 16 14:03:49
-- From         : interface descriptio
-- By           : Itf2Vhdl ver. 1.22
```

- Double-click on the file **ex1_TB.vhd** (or **yourdesignname_TB.vhd**) to open the file and scroll down to the comment ***"- - Add your stimulus here** ..."*

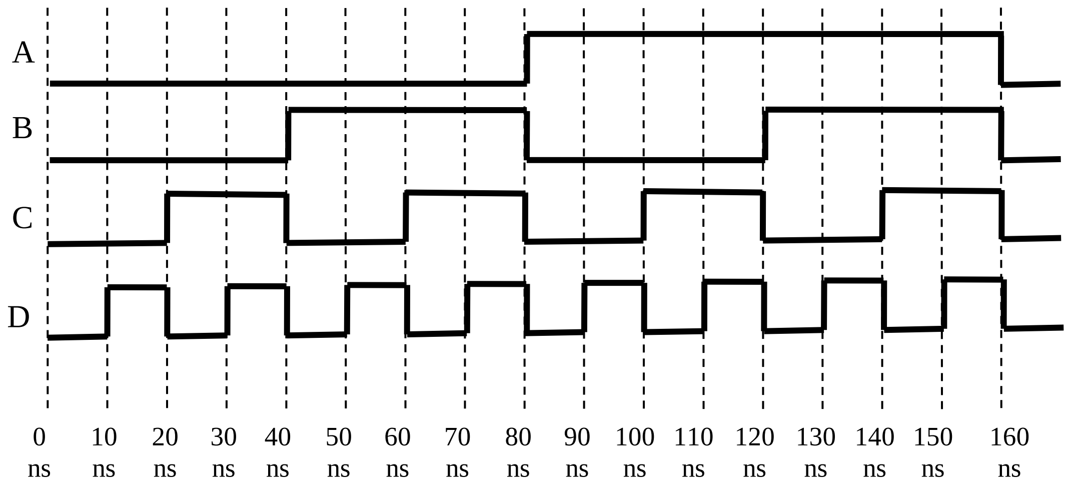


```
    -- Unit Under Test port map
    UUT : ex1
        port map (
            A => A,
            B => B,
            C => C,
            D => D,
            F => F
        );

    -- Add your stimulus here ...

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_ex1 of ex1_tb is
    for TB_ARCHITECTURE
        for UUT : ex1
            use entity work.ex1(ex1);
        end for;
    end for;
end TESTBENCH_FOR_ex1;
```

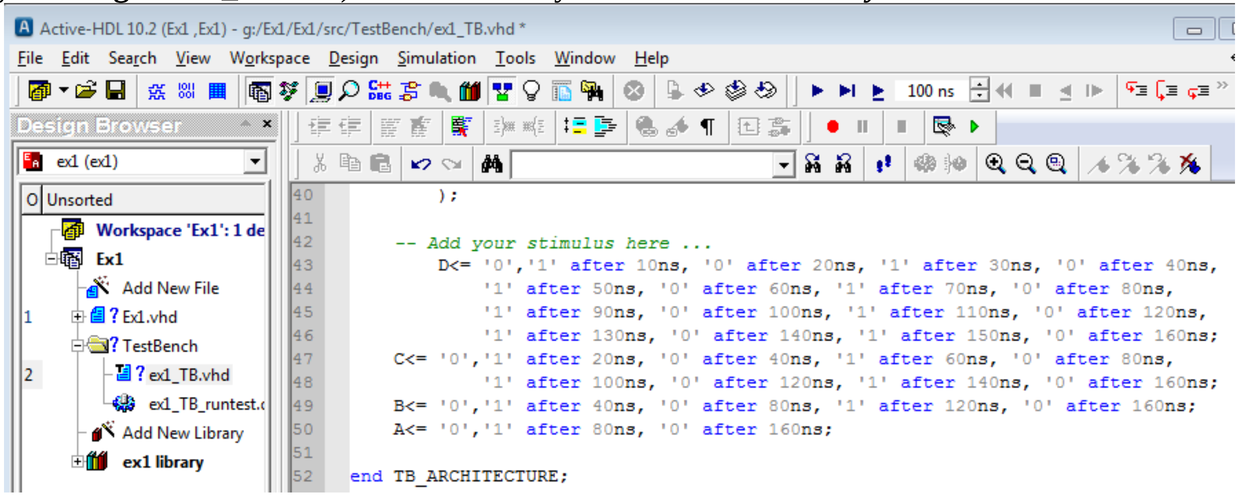
- Now we would like to generate a stimulus that will determine the output for all 16 possible input combinations of the four inputs (A,B,C, D). This can be done by generating waveforms as shown below.



- In order to use such waveforms in a computer simulation, each waveform must change at specific times. The exact amount of time for each count is somewhat arbitrary. Suppose that we select a time of 10 ns for each count. The waveforms can now be defined in terms of time as shown below.

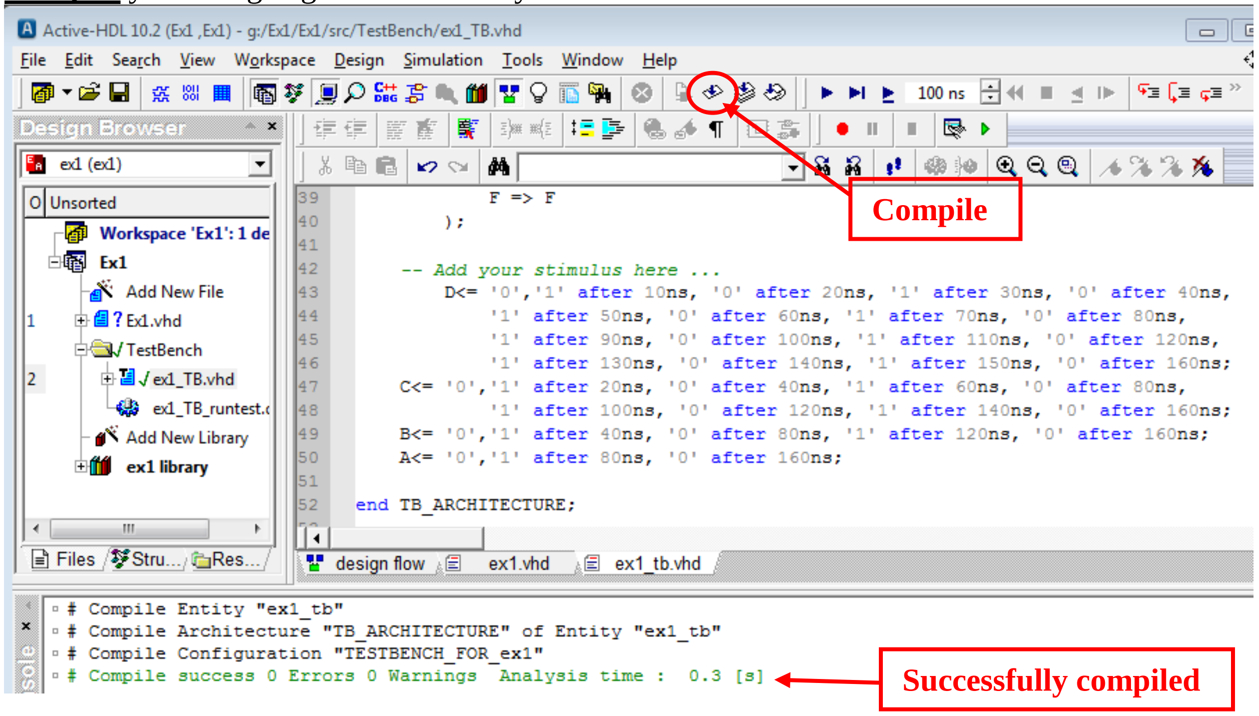


- Enter commands to describe the waveforms above into the file **ex1_TB.vhd** (or **yourdesignname_TB.vhd**) in the section by the comment ***"- - Add your stimulus here*** ..."



```
        );

    -- Add your stimulus here ...
        D<= '0','1' after 10ns, '0' after 20ns, '1' after 30ns, '0' after 40ns,
            '1' after 50ns, '0' after 60ns, '1' after 70ns, '0' after 80ns,
            '1' after 90ns, '0' after 100ns, '1' after 110ns, '0' after 120ns,
            '1' after 130ns, '0' after 140ns, '1' after 150ns, '0' after 160ns;
    C<= '0','1' after 20ns, '0' after 40ns, '1' after 60ns, '0' after 80ns,
            '1' after 100ns, '0' after 120ns, '1' after 140ns, '0' after 160ns;
    B<= '0','1' after 40ns, '0' after 80ns, '1' after 120ns, '0' after 160ns;
    A<= '0','1' after 80ns, '0' after 160ns;

end TB_ARCHITECTURE;
```

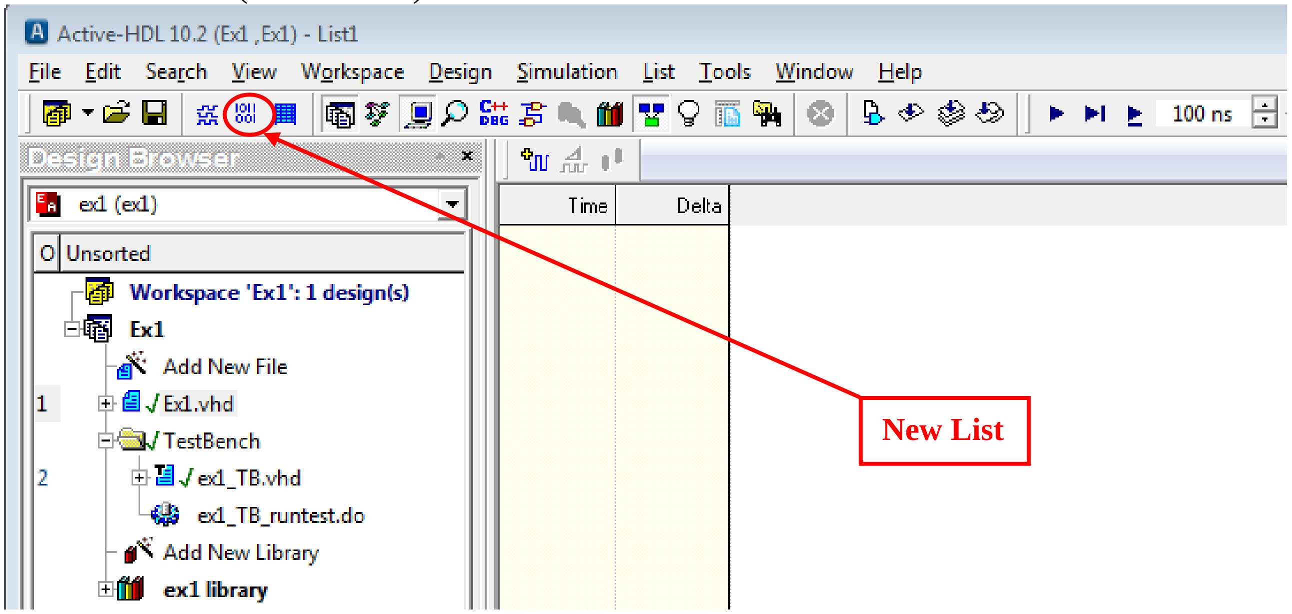
- **Compile** your design again. Correct any errors that occur in the VHDL code.



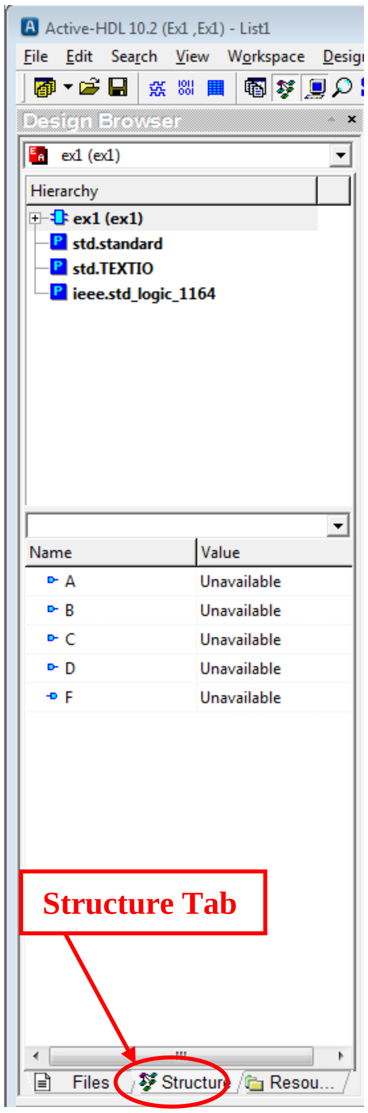
```
            F => F
        );

    -- Add your stimulus here ...
        D<= '0','1' after 10ns, '0' after 20ns, '1' after 30ns, '0' after 40ns,
            '1' after 50ns, '0' after 60ns, '1' after 70ns, '0' after 80ns,
            '1' after 90ns, '0' after 100ns, '1' after 110ns, '0' after 120ns,
            '1' after 130ns, '0' after 140ns, '1' after 150ns, '0' after 160ns;
    C<= '0','1' after 20ns, '0' after 40ns, '1' after 60ns, '0' after 80ns,
            '1' after 100ns, '0' after 120ns, '1' after 140ns, '0' after 160ns;
    B<= '0','1' after 40ns, '0' after 80ns, '1' after 120ns, '0' after 160ns;
    A<= '0','1' after 80ns, '0' after 160ns;

end TB_ARCHITECTURE;
```

```
# Compile Entity "ex1_tb"
# Compile Architecture "TB_ARCHITECTURE" of Entity "ex1_tb"
# Compile Configuration "TESTBENCH_FOR_ex1"
# Compile success 0 Errors 0 Warnings  Analysis time :  0.3 [s]
```

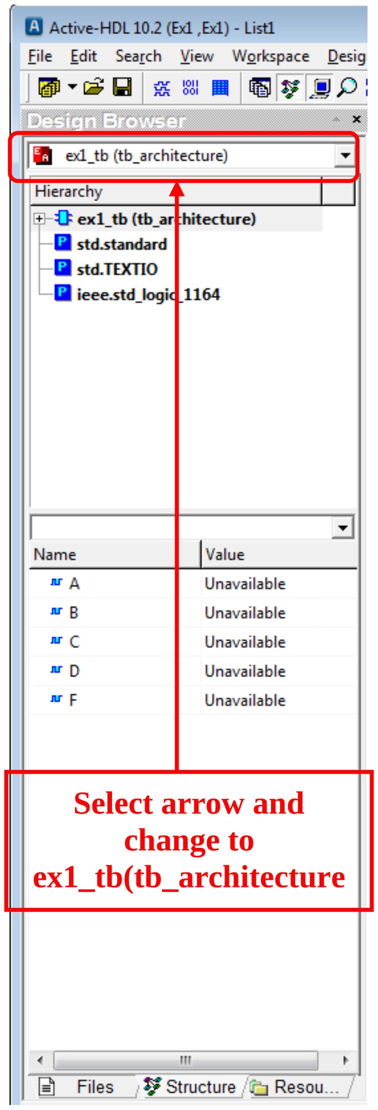
- Next we will generate a truth table to see if the design output is correct. Click on the **New List** button to create a new list (or truth table) based on the results observed from the simulation.



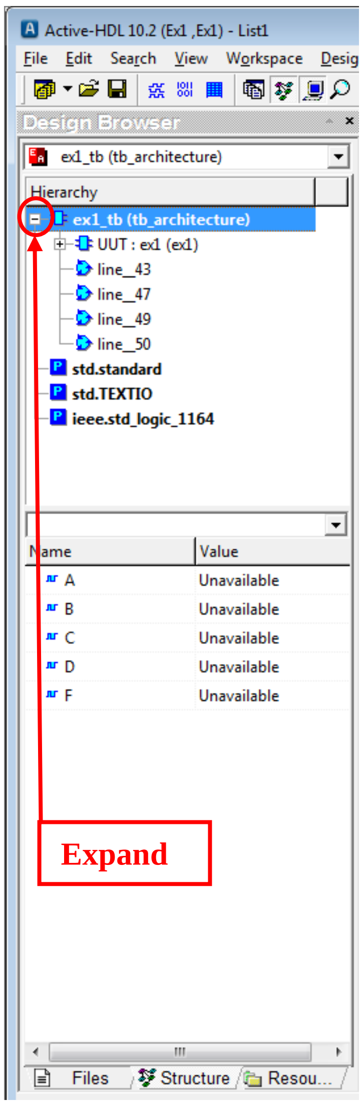
- The truth table can now be displayed as follows:
  - Click on the **Structure tab** in the **Design Browser** - see left window below.
  - Click on the **arrow** at the top of the browser and select **ex1_tb(tb_architecture)** from the pull down list – see center window below.
  - **Expand** ex1_tb(tb_architecture) and all of the input and output signals should now appear in the Browser window – see right window below.



Name	Value
A	Unavailable
B	Unavailable
C	Unavailable
D	Unavailable
F	Unavailable

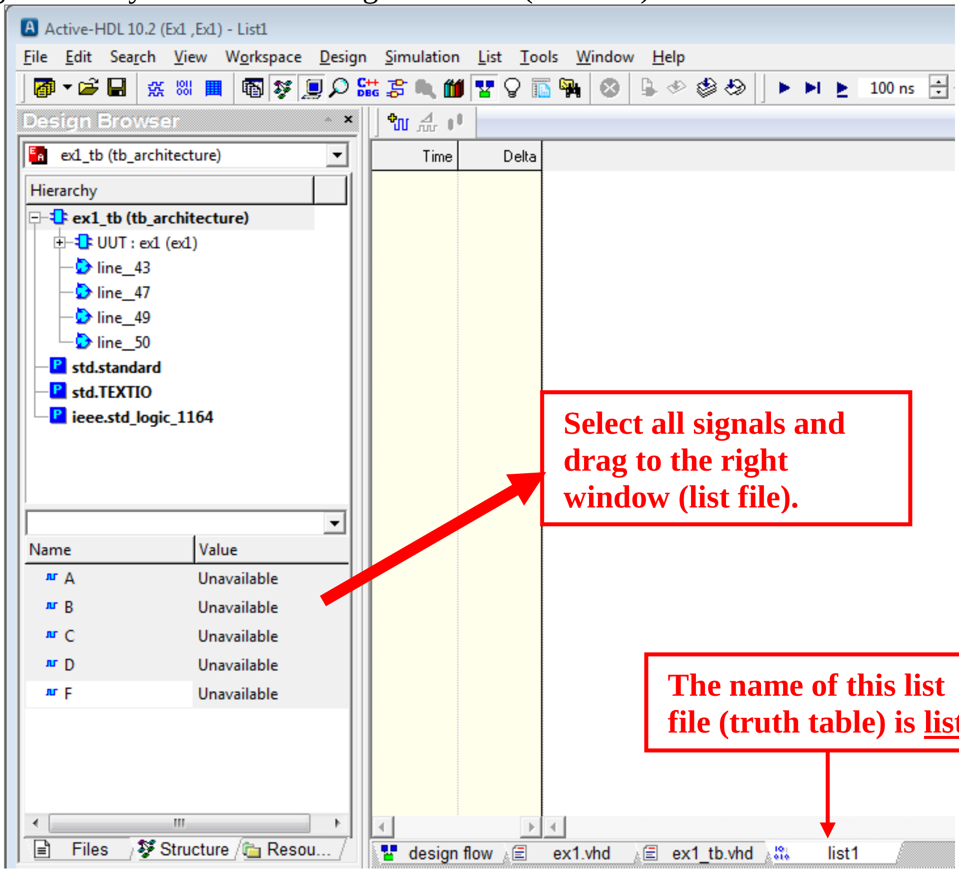


Name	Value
A	Unavailable
B	Unavailable
C	Unavailable
D	Unavailable
F	Unavailable

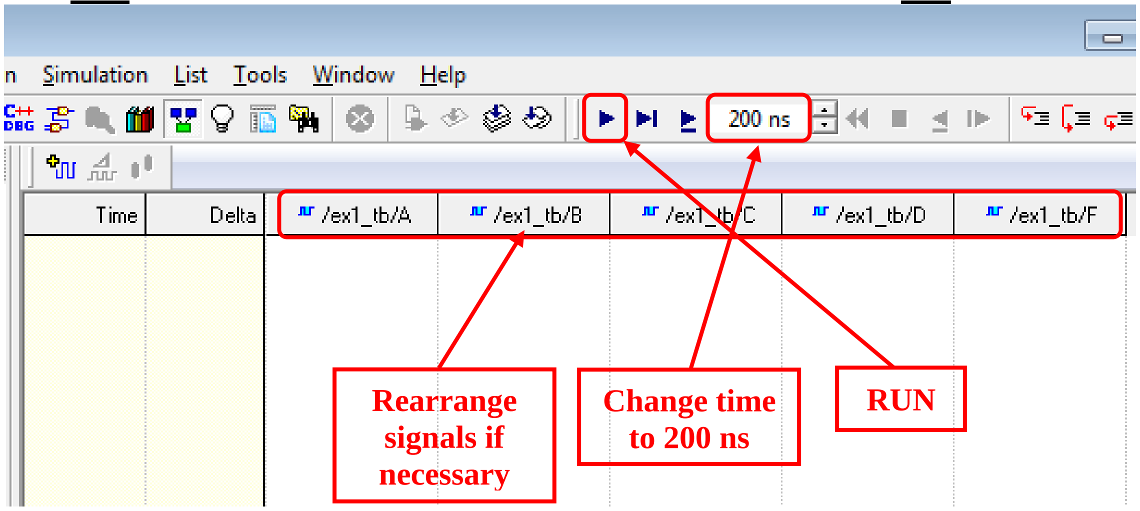


Name	Value
A	Unavailable
B	Unavailable
C	Unavailable
D	Unavailable
F	Unavailable

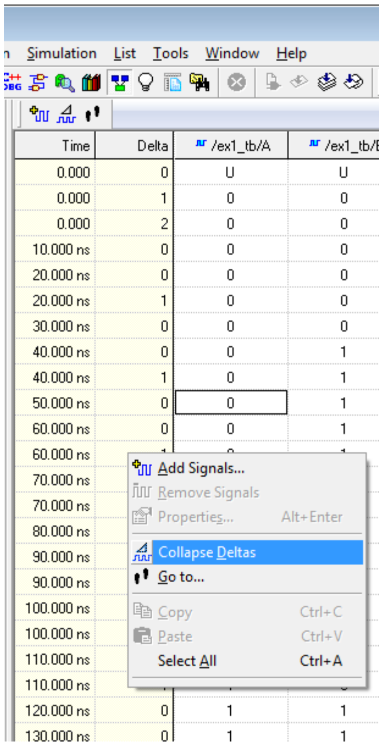
- Click on any signal in the browser (A, B, C, D, or F) and press Ctrl + A to select all of the signals. Next drag the signals to any location in the right window (List File).



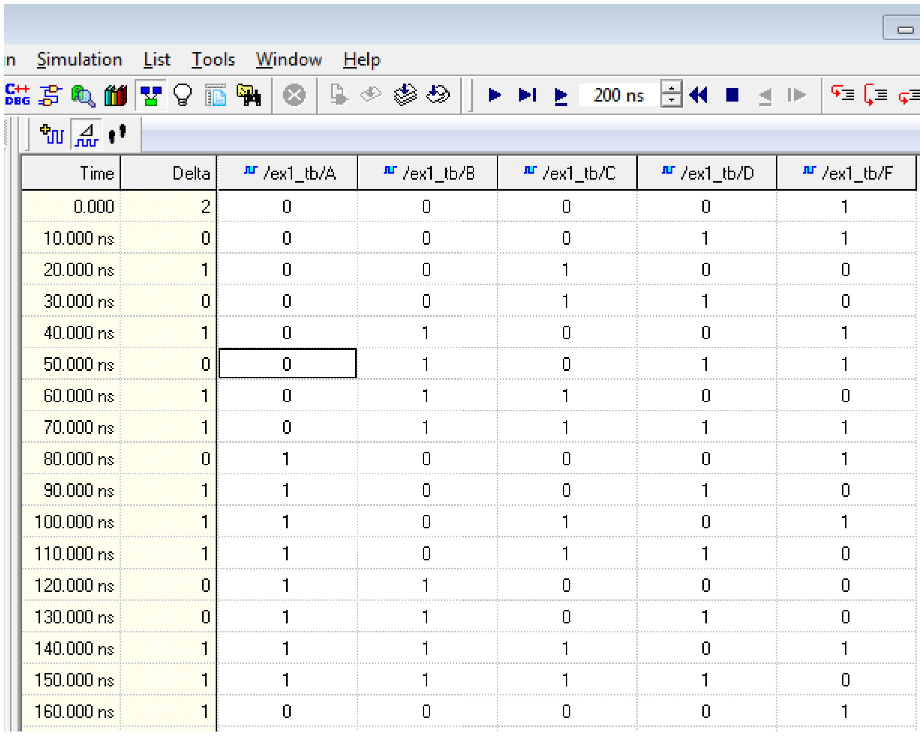
- If the signals are not in the desired order, rearrange (drag) the signals into the desired order (MSB to LSB for example) as shown below.
- Change the time for the simulation to **200ns** (since we defined our inputs from 0 to 160ns) as shown below.
- Click the **Run** button to run the simulation as shown below and then click **OK**.



- Note that some of the time increments occur more than once. Right click on any time value and select **Collapse Deltas** to remove the redundant time values.

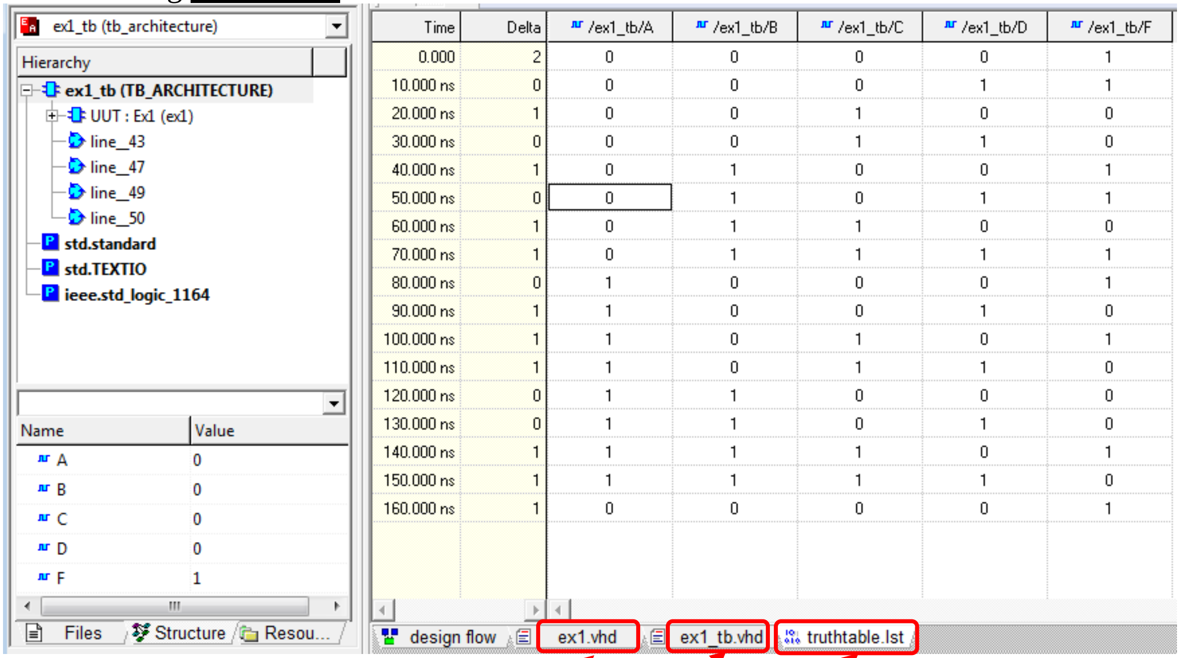


Time	Delta	/ex1_tb/A	/ex1_tb/B
0.000	0	U	U
0.000	1	0	0
0.000	2	0	0
10.000 ns	0	0	0
20.000 ns	0	0	0
20.000 ns	1	0	0
30.000 ns	0	0	0
40.000 ns	0	0	1
40.000 ns	1	0	1
50.000 ns	0	0	1
60.000 ns	0	0	1
60.000 ns			
70.000 ns			
70.000 ns			
80.000 ns			
90.000 ns			
90.000 ns			
100.000 ns			
100.000 ns			
110.000 ns			
110.000 ns			
120.000 ns	0	1	1
130.000 ns	0	1	1



Time	Delta	/ex1_tb/A	/ex1_tb/B	/ex1_tb/C	/ex1_tb/D	/ex1_tb/F
0.000	2	0	0	0	0	1
10.000 ns	0	0	0	0	1	1
20.000 ns	1	0	0	1	0	0
30.000 ns	0	0	0	1	1	0
40.000 ns	1	0	1	0	0	1
50.000 ns	0	0	1	0	1	1
60.000 ns	1	0	1	1	0	0
70.000 ns	1	0	1	1	1	1
80.000 ns	0	1	0	0	0	1
90.000 ns	1	1	0	0	1	0
100.000 ns	1	1	0	1	0	1
110.000 ns	1	1	0	1	1	0
120.000 ns	0	1	1	0	0	0
130.000 ns	0	1	1	0	1	0
140.000 ns	1	1	1	1	0	1
150.000 ns	1	1	1	1	1	0
160.000 ns	1	0	0	0	0	1

- Check the truth table. Recall that the output was defined as F(A,B,C,D) = Σ(0,1,4,5,7,8,10,14) for this example. Note that the truth table above is correct.
- Save the truth table by selecting **File – Save As – Truth Table.lst**
- Printing. You can print the truth table, VHDL code, or testbench code by selecting the appropriate tab and using **File - Print**.



Name	Value
A	0
B	0
C	0
D	0
F	1

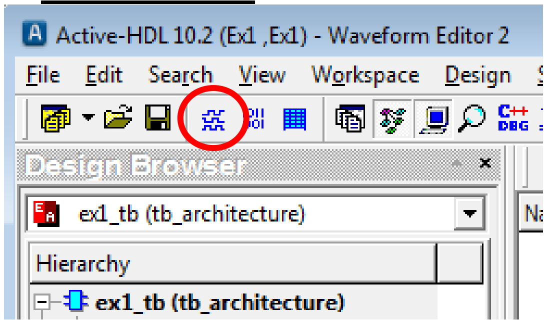
Time	Delta	/ex1_tb/A	/ex1_tb/B	/ex1_tb/C	/ex1_tb/D	/ex1_tb/F
0.000	2	0	0	0	0	1
10.000 ns	0	0	0	0	1	1
20.000 ns	1	0	0	1	0	0
30.000 ns	0	0	0	1	1	0
40.000 ns	1	0	1	0	0	1
50.000 ns	0	0	1	0	1	1
60.000 ns	1	0	1	1	0	0
70.000 ns	1	0	1	1	1	1
80.000 ns	0	1	0	0	0	1
90.000 ns	1	1	0	0	1	0
100.000 ns	1	1	0	1	0	1
110.000 ns	1	1	0	1	1	0
120.000 ns	0	1	1	0	0	0
130.000 ns	0	1	1	0	1	0
140.000 ns	1	1	1	1	0	1
150.000 ns	1	1	1	1	1	0
160.000 ns	1	0	0	0	0	1

Select the desired file to view or print.

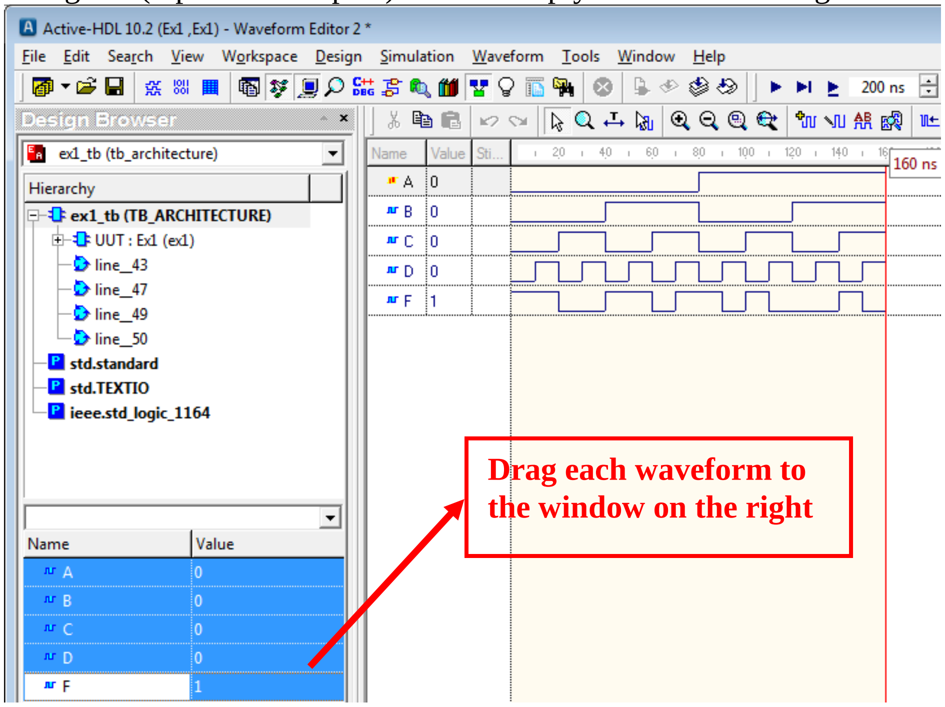
Generating Waveforms

The results of the simulation can also be displayed using waveforms.
If you have already run a simulation to generate a truth table (list), select **Simulation – End Simulation** from the main menu.

- Select the **New Waveform** button as shown below.



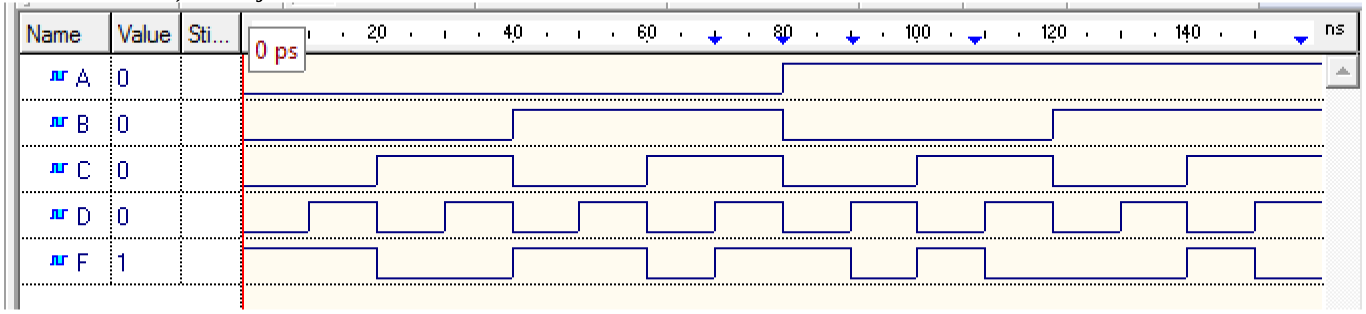
- Drag the available signals (inputs and outputs) into the empty window on the right.



Name	Value
A	0
B	0
C	0
D	0
F	1

- If necessary, rearrange the waveforms (by dragging them) into the desired order.

- Select the **RUN** ▶ button to run the simulation. Use the 🔍 symbols (Zoom In, Zoom Out, Zoom To Fit) to adjust the size of the waveform.

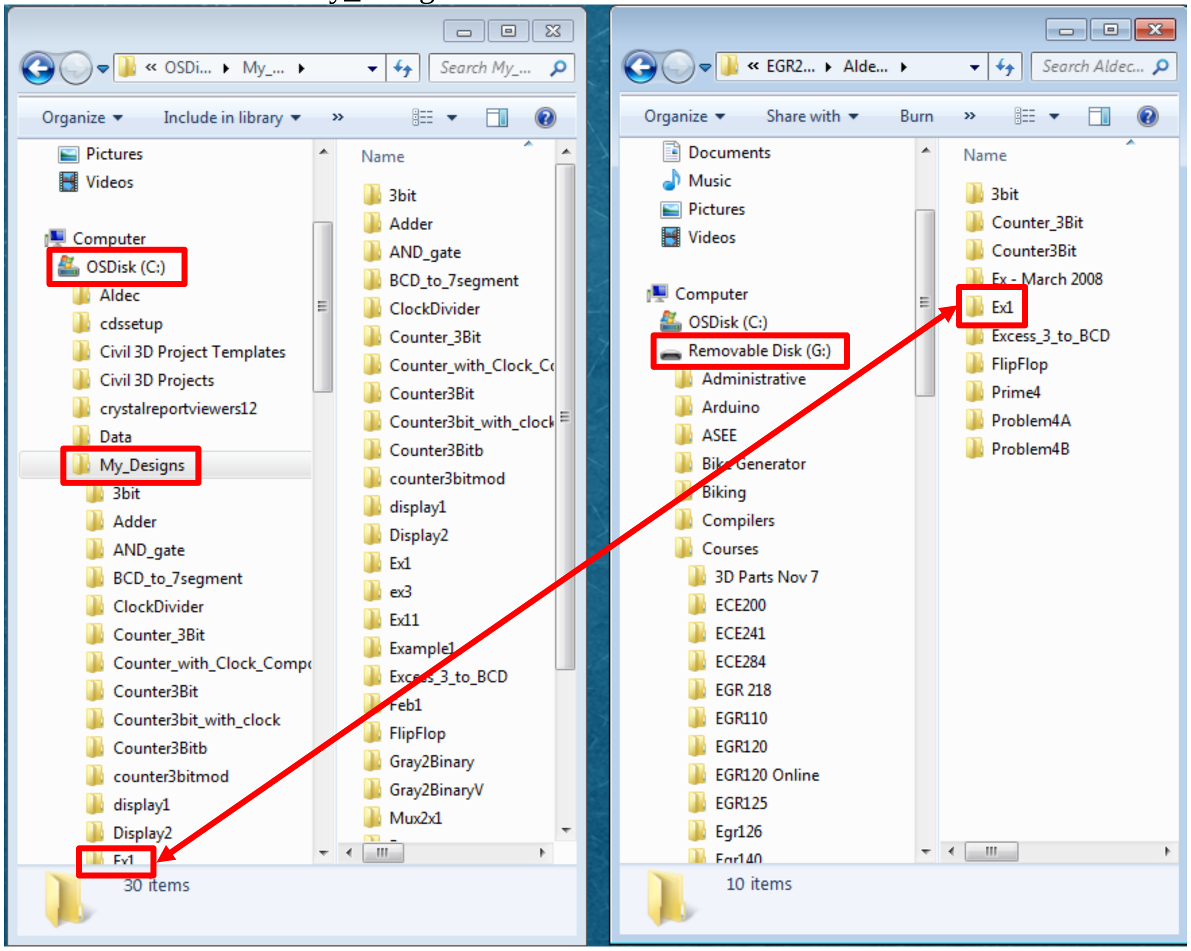


Name	Value
A	0
B	0
C	0
D	0
F	1

- Save the waveform file by selecting **File – Save As – Waveform.awf**
- Printing. You can print the waveform, truth table, VHDL code, or testbench code by selecting the appropriate tab and using **File - Print**. You will need to print all 4 of these files for your lab report.

File Locations

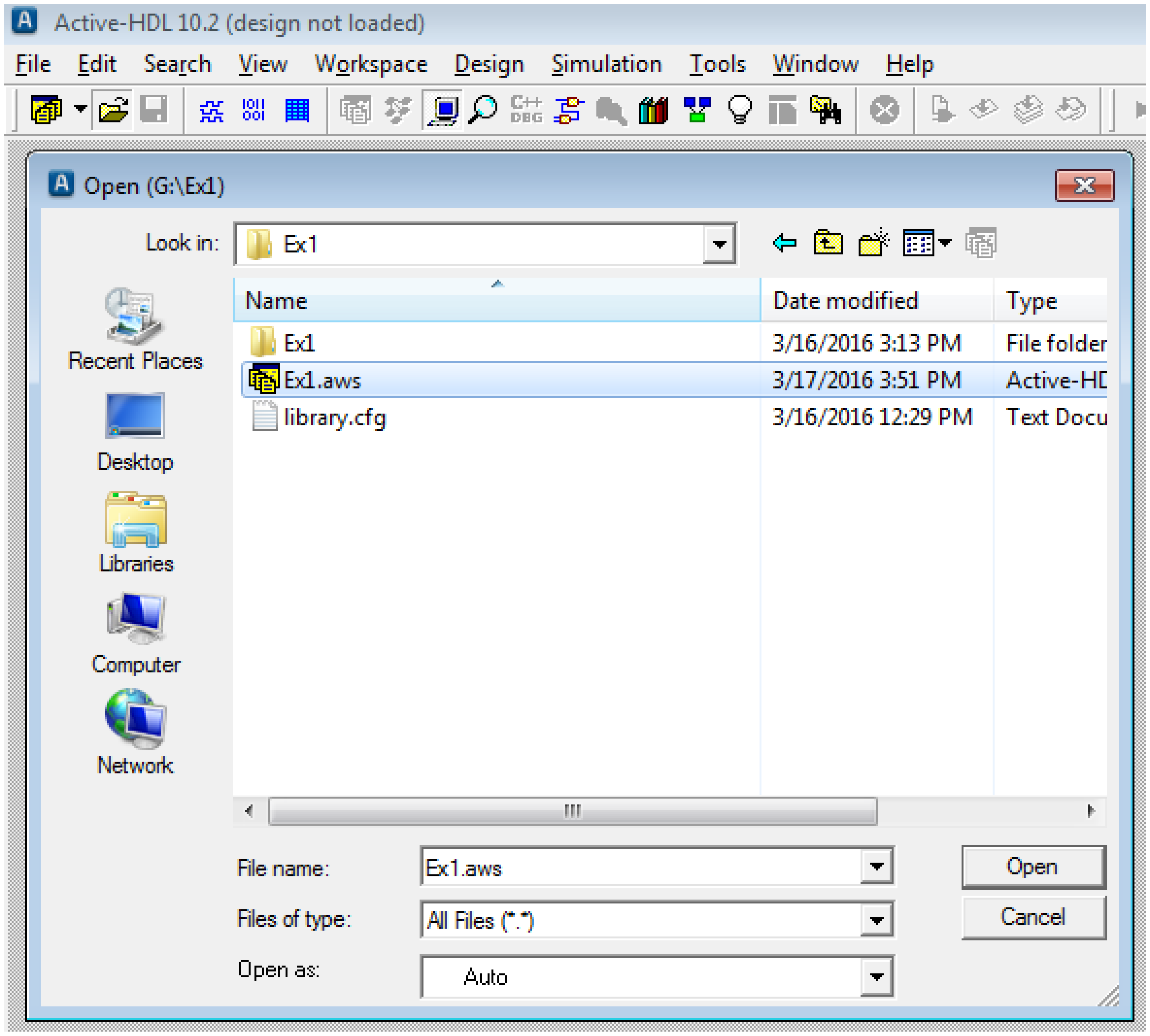
- Aldec Active-HDL will save your files by default in the C:\My_Designs folder.
- It is recommended that you leave this as the default setting.
- When you have finished your work in lab, copy the folder for your design onto a personal memory storage device as illustrated below.
- If you return to lab another day to work on the project again, copy the folder from your personal storage device back into the C:\My_Designs folder as illustrated below.



Drag design folder (Ex1 in this case) between C:\My_Designs and your personal memory device.

<u>Opening an existing design (workspace)</u> – Useful if you don't finish the lab and want to know how to re-open the design at a later date.

- Launch **<u>Aldec Active-HDL</u>**.
- Cancel the Getting Started window.
- Select **<u>File – Open</u>** and locate your workspace file (**aws** extension) as shown below.


Active-HDL 10.2 (design not loaded)
File Edit Search View Workspace Design Simulation Tools Window Help
Open (G:\Ex1)
Look in: Ex1
Recent Places
Desktop
Libraries
Computer
Network
Open
Cancel

Name	Date modified	Type
Ex1	3/16/2016 3:13 PM	File folder
Ex1.aws	3/17/2016 3:51 PM	Active-HD
library.cfg	3/16/2016 12:29 PM	Text Docu

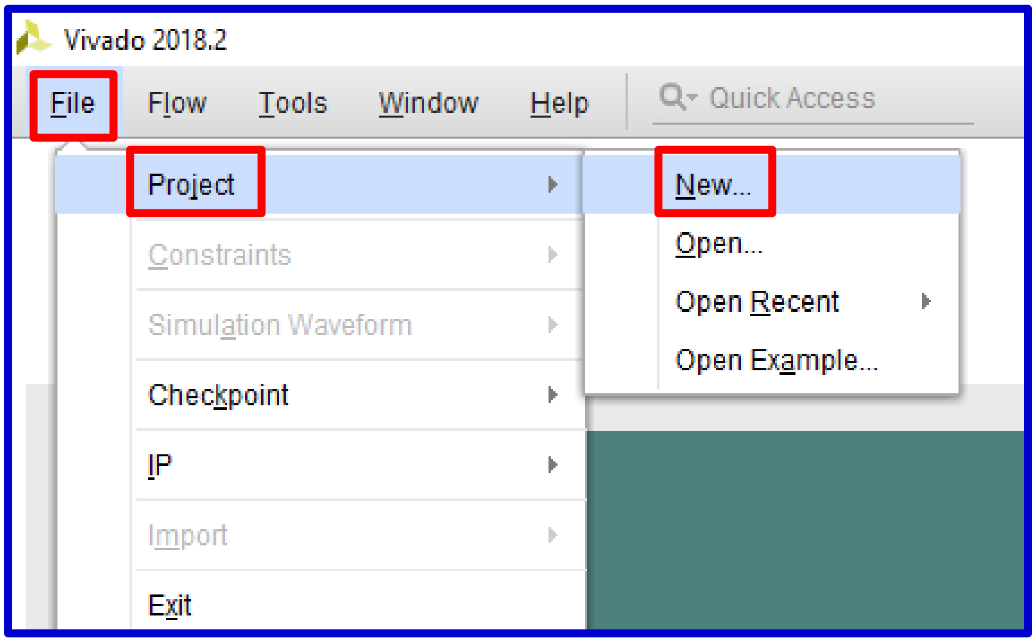
File name: Ex1.aws
Files of type: All Files (*.*)
Open as: Auto

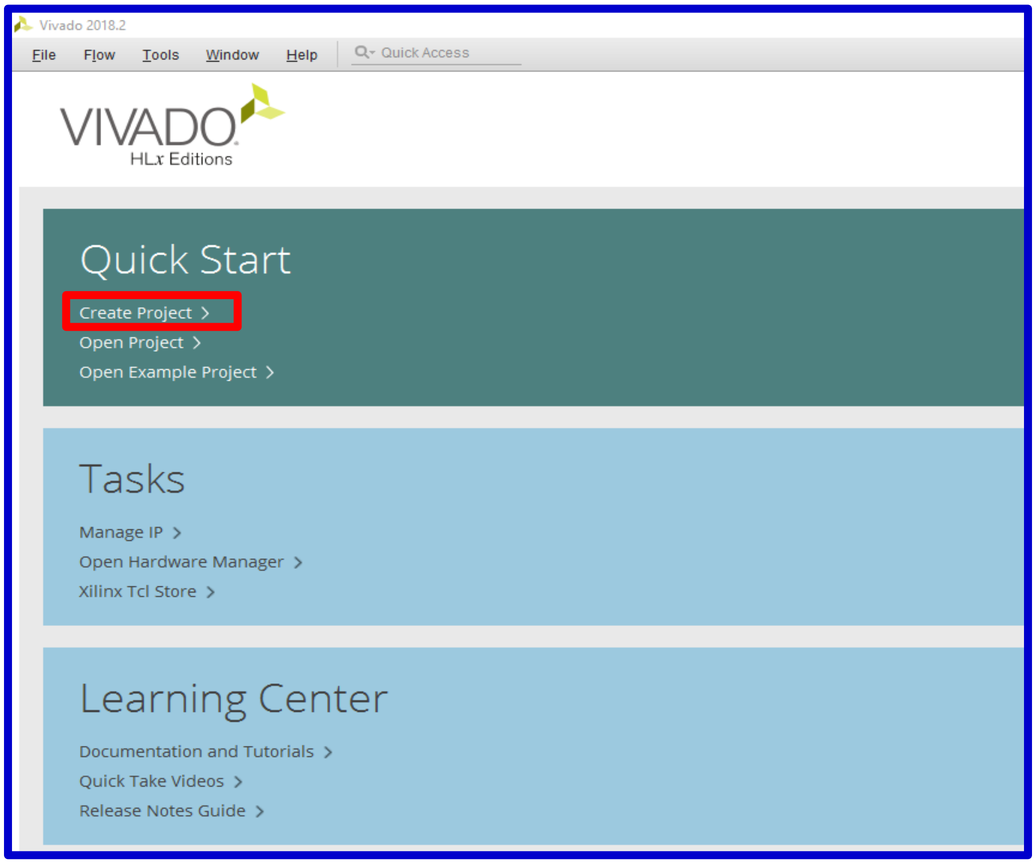
Xilinx Vivado

Xilinx Vivado will be used to implement a VHDL design into an FPGA. The design just created using Aldec Active-HDL is specified in the file **Ex1.vhd**.

1. Start Xilinx Vivado

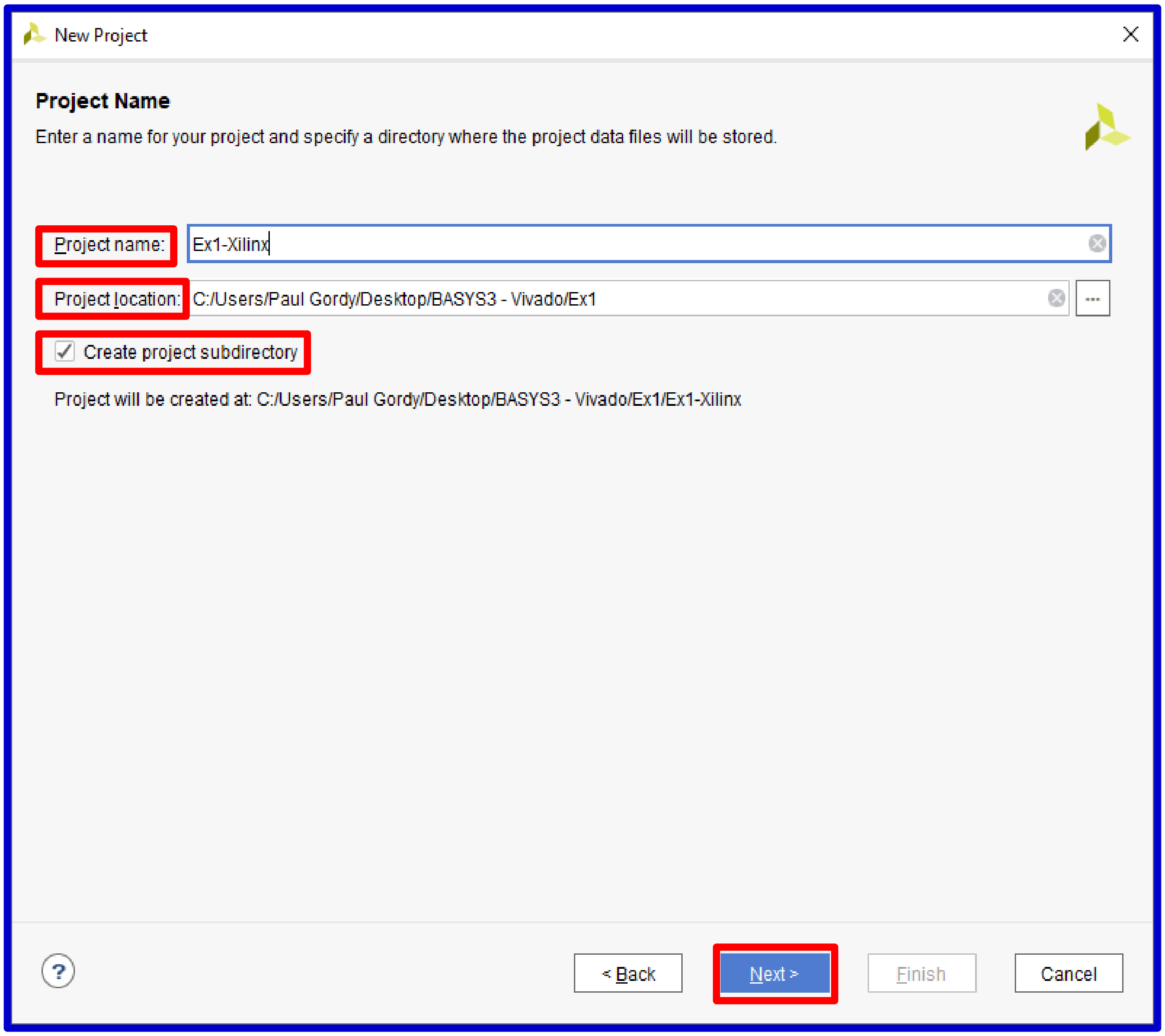
- **Launch Xilinx Vivado** using the shortcut on the desktop
- **Select File - Project – New** or **Create Project** (or select **Create Project** under the **Quick Start** window)





2. Project Name

- **Project Name**: Enter a name
- **Project Location**: Enter a location
- **Create project subdirectory**: Check box
- Select **Next**



New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: Ex1-Xilinx

Project location: C:/Users/Paul Gordy/Desktop/BASYS3 - Vivado/Ex1

✓ Create project subdirectory

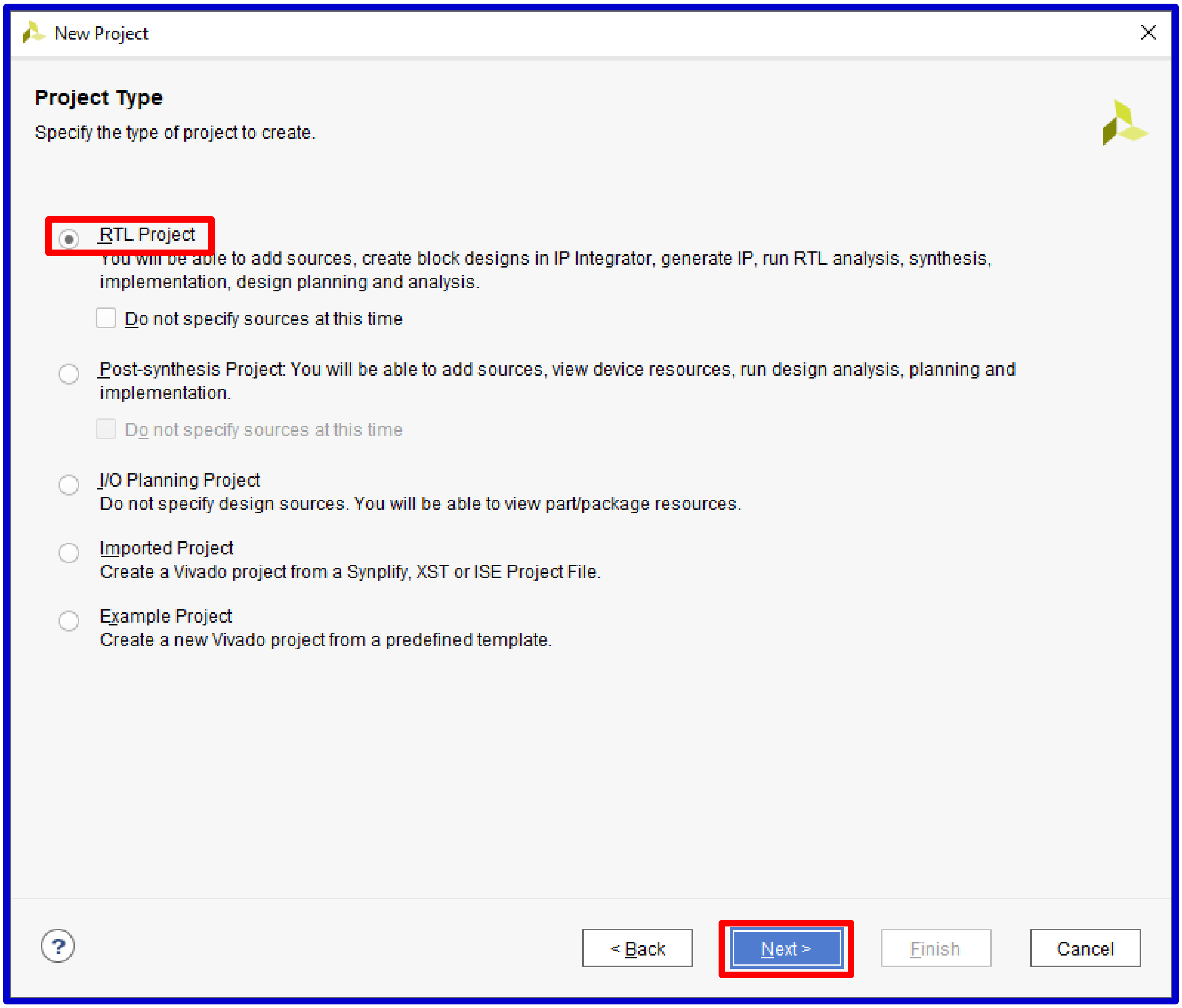
Project will be created at: C:/Users/Paul Gordy/Desktop/BASYS3 - Vivado/Ex1/Ex1-Xilinx

Recommendations:

- File and folder names: Use only letters, numbers and underscores
- Project location: Use the folder already containing the Aldec vhdl files (this keeps them together in the same location).

3. Project Type

- Select **RTL Project**
- Select **Next**



New Project

Project Type

Specify the type of project to create.

(●) RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

☐ Do not specify sources at this time

( ) Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.

☐ Do not specify sources at this time

( ) I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

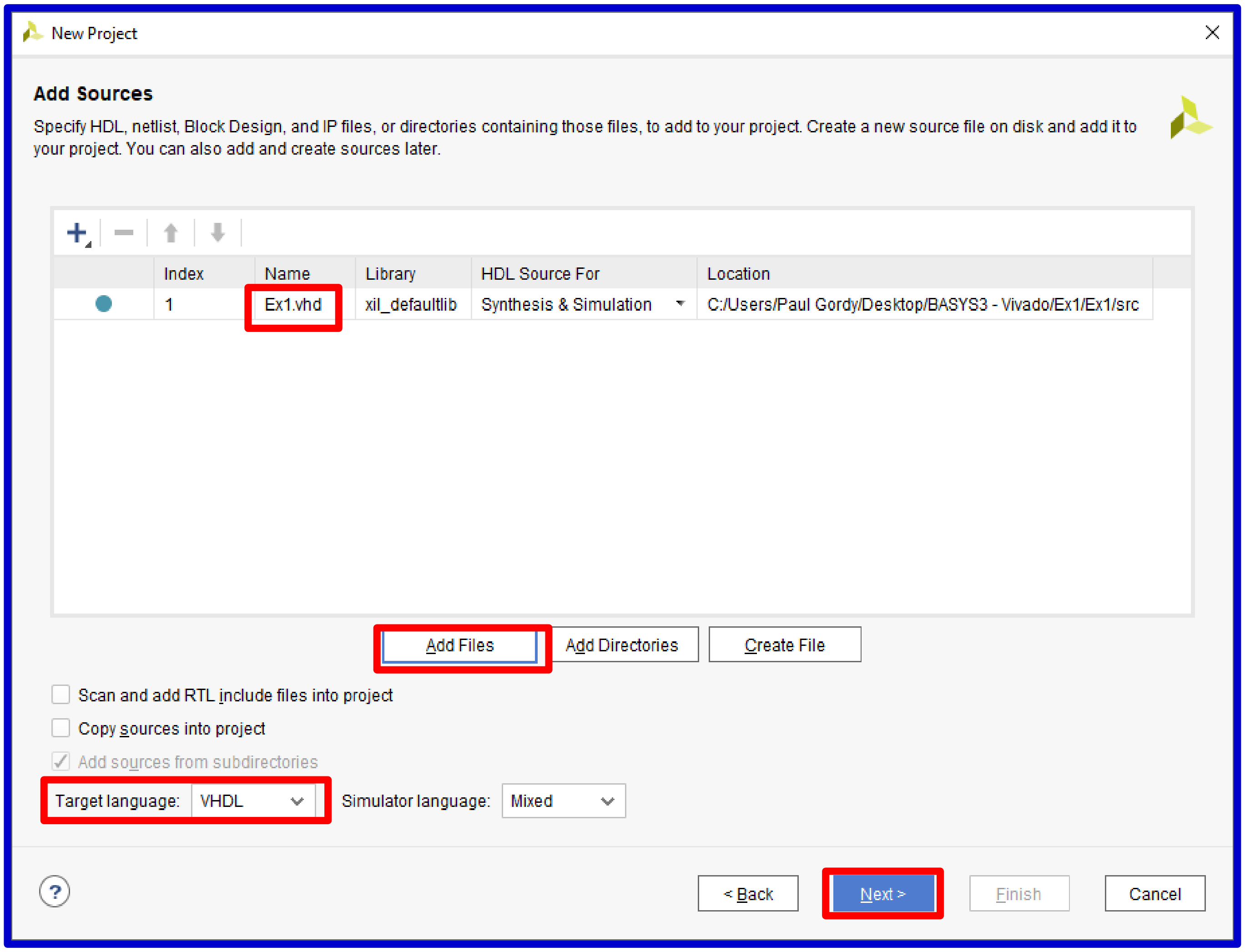
( ) Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

( ) Example Project
Create a new Vivado project from a predefined template.

Note: ***RTL*** stands for ***Register Transfer Level*** or ***Register Transfer Logic***) and refers to describing a design in terms of signals transferred between registers (flip-flops) using HDL. It can include signal descriptions for both synchronous and combinational circuits.
An RTL description may be converted into a gate-level design during the implementation phase.

4. Add Sources

- **Add Files** – Select Add Files and then browse to find the vhd file created using Aldec Active HDL. Ex1.vhd was added in this example.
- **Target language**: VHDL
- Select **Next**



New Project

Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

	Index	Name	Library	HDL Source For	Location
●	1	Ex1.vhd	xil_defaultlib	Synthesis & Simulation	C:/Users/Paul Gordy/Desktop/BASYS3 - Vivado/Ex1/Ex1/src

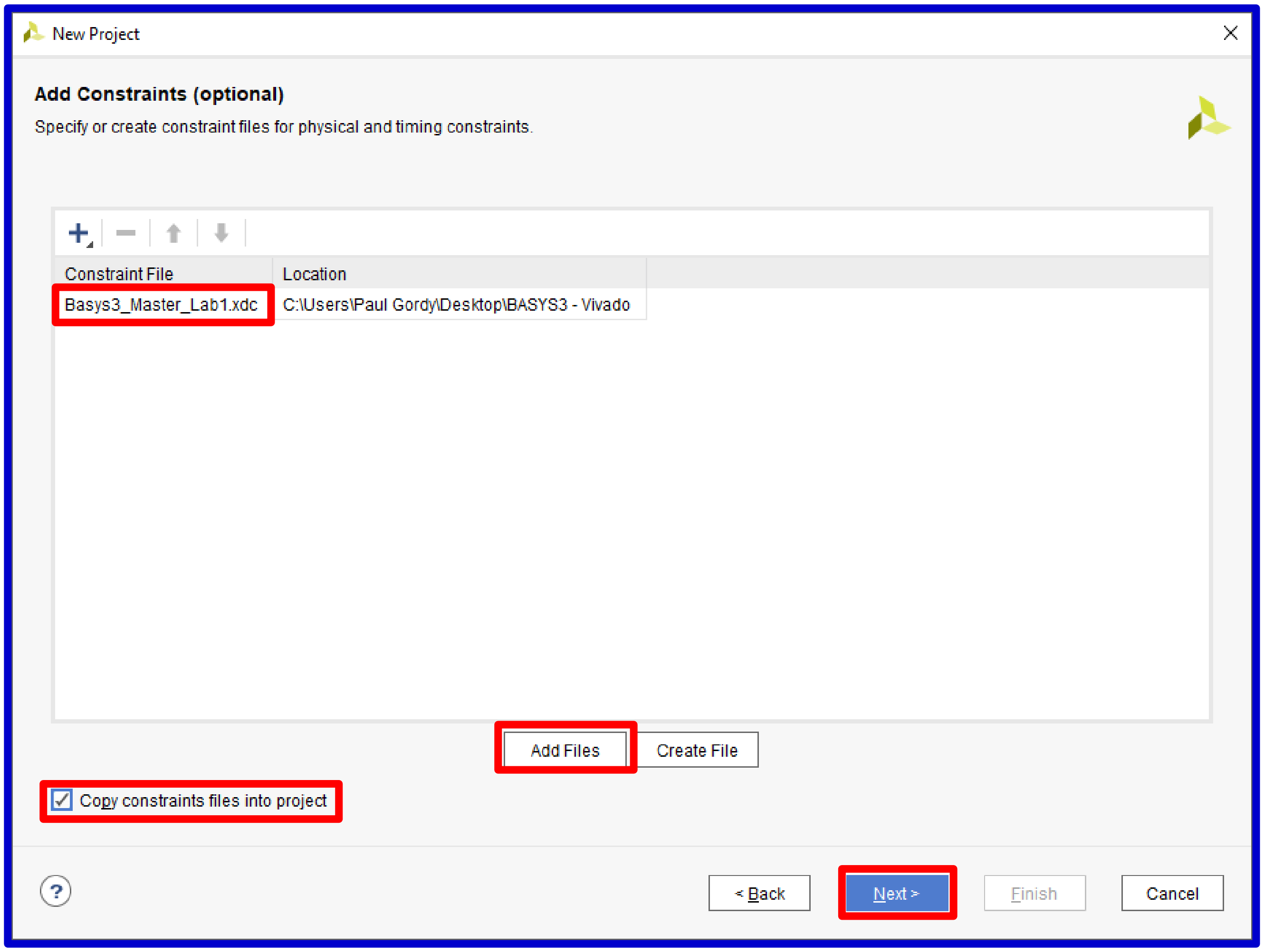
- [ ] Scan and add RTL include files into project
- [ ] Copy sources into project
- [x] Add sources from subdirectories

Target language: VHDL Simulator language: Mixed

5. <u>Add Constraints</u>

A constraints file (xdc) can be added now or later. If you have already created the constraints file (Preliminary Work for lab), it can be added as shown below. If you have not created it yet, you will need to create it first using Notepad. See the next pages to create the xdc file and then return to this page to add the file.

- **<u>Add Files</u>** – Select Add Files and then browse to find the xdc file that was created using Notepad. The file ***Basys3_Master_Ex1.xdc*** was added in this example.
- **<u>Check the box</u>** – Copy constraints files into project
- Select **<u>Next</u>**


New Project

Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

Constraint File	Location
Basys3_Master_Lab1.xdc	C:\Users\Paul Gordy\Desktop\BASYS3 - Vivado

Creating a Constraint File

- A ***constraint file*** (.xdc) is a file used to assigned signals to pins on the FPGA.
- The file ***Basys3_Master.xdc*** has been provided for the BASYS3. ***Download this file from the course website.*** The file needs to be modified (using NotePad) to assign pins to the inputs and outputs used in this example.
- This example uses a simple SOP circuit where

  F(A,B,C,D) = Σ(0,1,4,5,7,8,10,14) = A'C' + A'BD + ACD' + B'C'D'

  so we need 4 inputs switches (A,B,C,D) and one output LED (F).
- Note that there are 16 slide switches and 16 LEDs on the BASYS3 (Refer to the BASYS3 pinout shown or Figure 16 from the BASYS3 Reference Manual.
- We might select 4 of the switches and one of the LEDs as shown below. The pin numbers from the pinout on the right were used to complete the table below.

Input/ Output	BASYS3 Name	BASYS3 Pin
A	**SW3**	**W17**
B	**SW2**	**W16**
C	**SW1**	**V16**
D	**SW0**	**V17**
F	**LED0**	**U16**

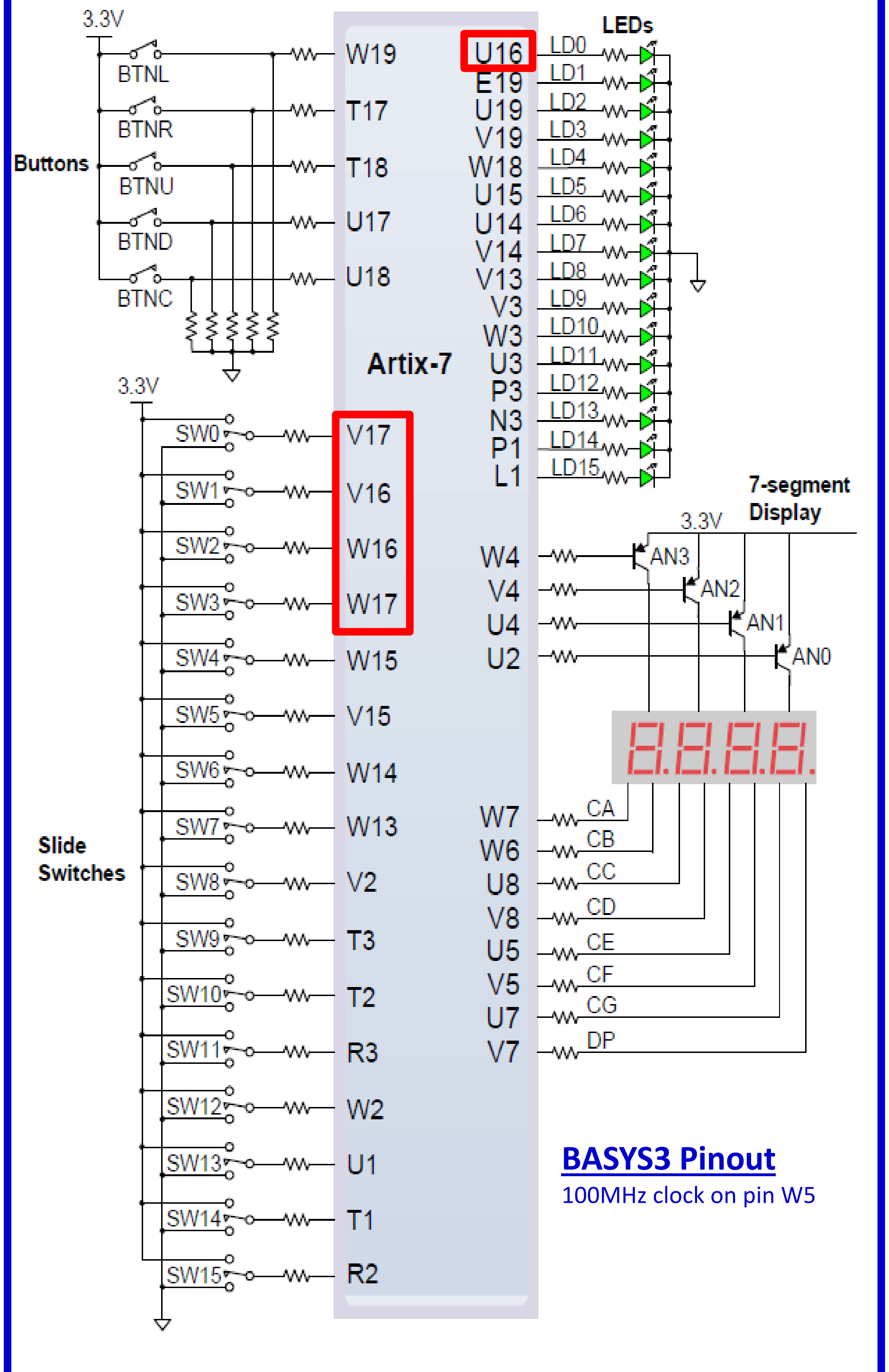
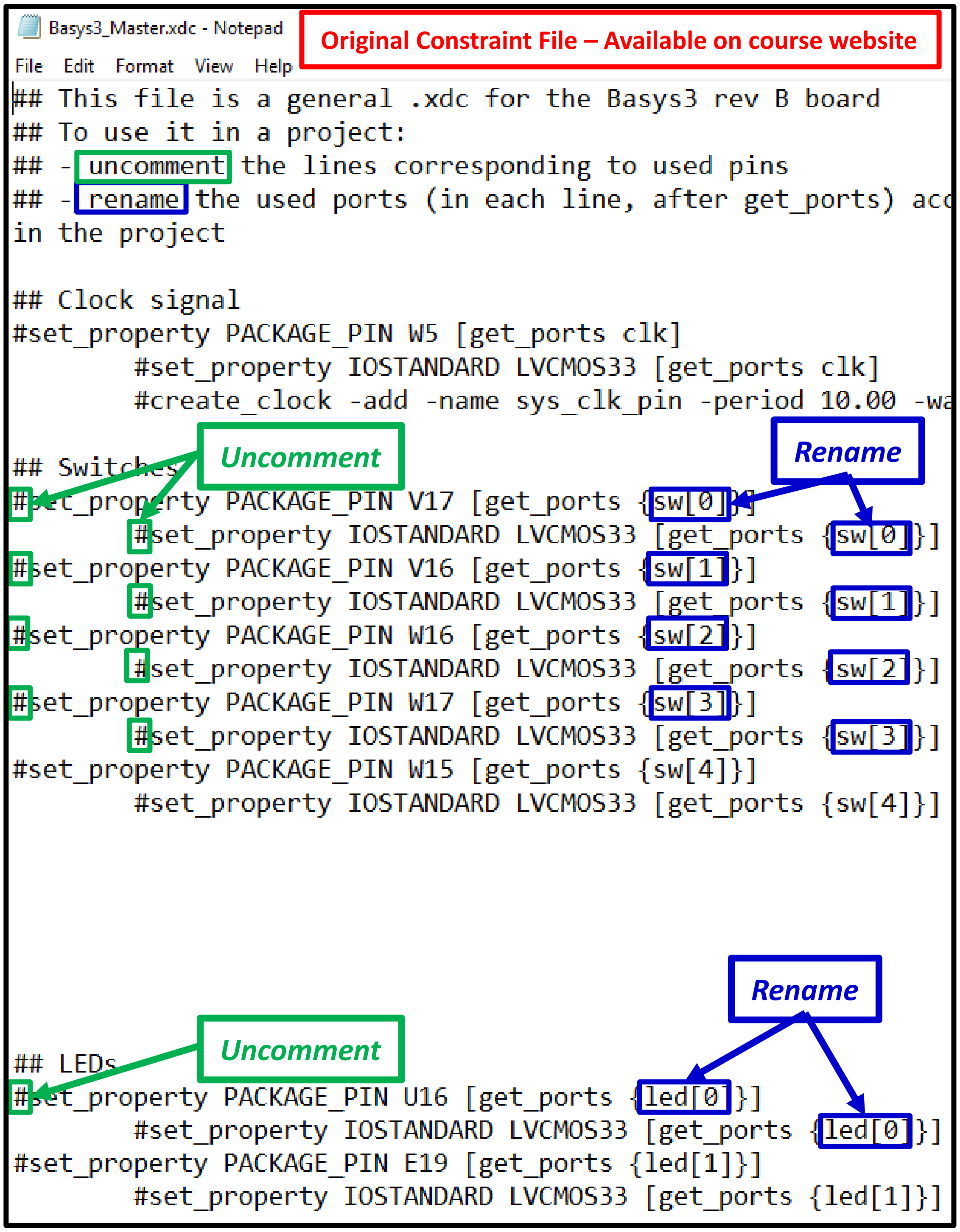


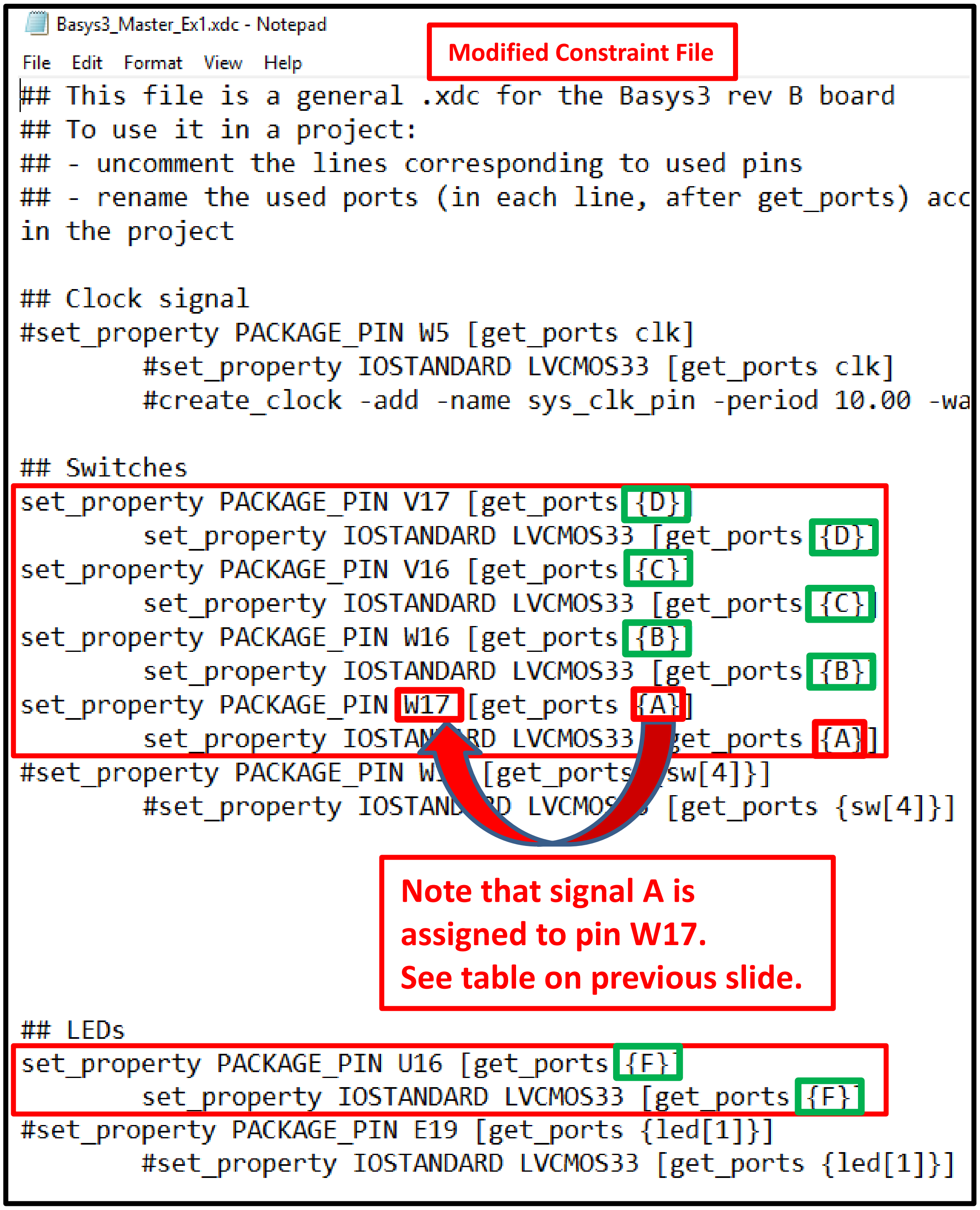
Figure 16. General Purpose I/O devices on the Basys3

Instructions for creating a constraints file:

- Download the file **Basys3_Master.xdc** from the course website.
- Modify it to remove comments (#) and assign pins.
- Save it using a new name (**Basys3_Master_Ex1.xdc** in this example).


Basys3_Master.xdc - Notepad
File Edit Format View Help
Original Constraint File – Available on course website
Uncomment
Rename
Uncomment
Rename

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) acc
in the project

## Clock signal
#set_property PACKAGE_PIN W5 [get_ports clk]
        #set_property IOSTANDARD LVCMOS33 [get_ports clk]
        #create_clock -add -name sys_clk_pin -period 10.00 -wa

## Switches
#set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
#set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
#set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
#set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
#set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]



## LEDs
#set_property PACKAGE_PIN U16 [get_ports {led[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
#set_property PACKAGE_PIN E19 [get_ports {led[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
```

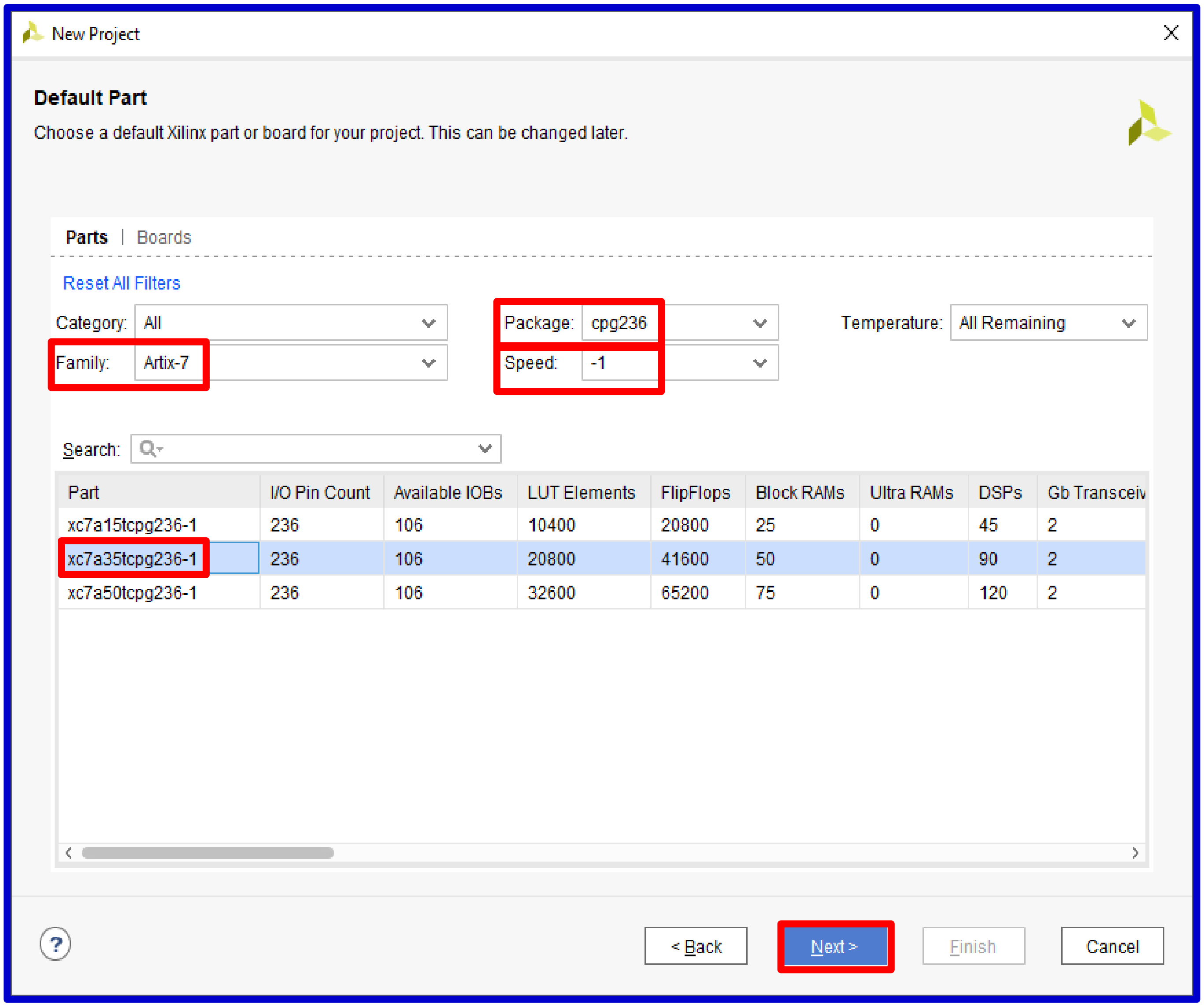

Basys3_Master_Ex1.xdc - Notepad

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) acc
in the project

## Clock signal
#set_property PACKAGE_PIN W5 [get_ports clk]
        #set_property IOSTANDARD LVCMOS33 [get_ports clk]
        #create_clock -add -name sys_clk_pin -period 10.00 -wa

## Switches
set_property PACKAGE_PIN V17 [get_ports {D}]
        set_property IOSTANDARD LVCMOS33 [get_ports {D}]
set_property PACKAGE_PIN V16 [get_ports {C}]
        set_property IOSTANDARD LVCMOS33 [get_ports {C}]
set_property PACKAGE_PIN W16 [get_ports {B}]
        set_property IOSTANDARD LVCMOS33 [get_ports {B}]
set_property PACKAGE_PIN W17 [get_ports {A}]
        set_property IOSTANDARD LVCMOS33 [get_ports {A}]
#set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {F}]
        set_property IOSTANDARD LVCMOS33 [get_ports {F}]
#set_property PACKAGE_PIN E19 [get_ports {led[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
```

Note that signal A is assigned to pin W17. See table on previous slide.

6. Default Part

- Add the following specifications for the BASYS3 FPGA board:
- **Family**: Artix-7
- **Package**: cpg236
- **Speed**: -1
- A list of parts meeting these specifications is shown. Select:
- **Part**: xca35tcpg236-1
- Select ***Next***



New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Parts | Boards

Reset All Filters

Category: All

Family: Artix-7

Package: cpg236

Speed: -1

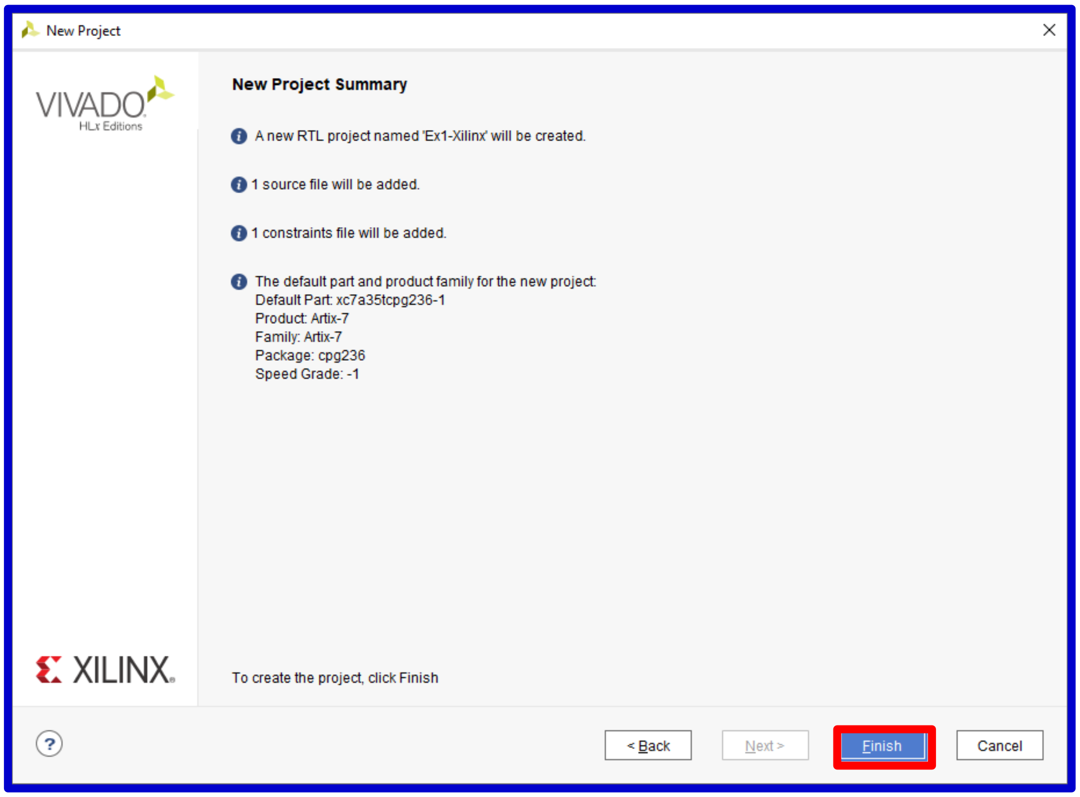
Temperature: All Remaining

Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceiv
xc7a15tcpg236-1	236	106	10400	20800	25	0	45	2
xc7a35tcpg236-1	236	106	20800	41600	50	0	90	2
xc7a50tcpg236-1	236	106	32600	65200	75	0	120	2

7. New Project Summary

- Read the summary to check for errors.
- Select ***Finish***



New Project Summary

A new RTL project named 'Ex1-Xilinx' will be created.

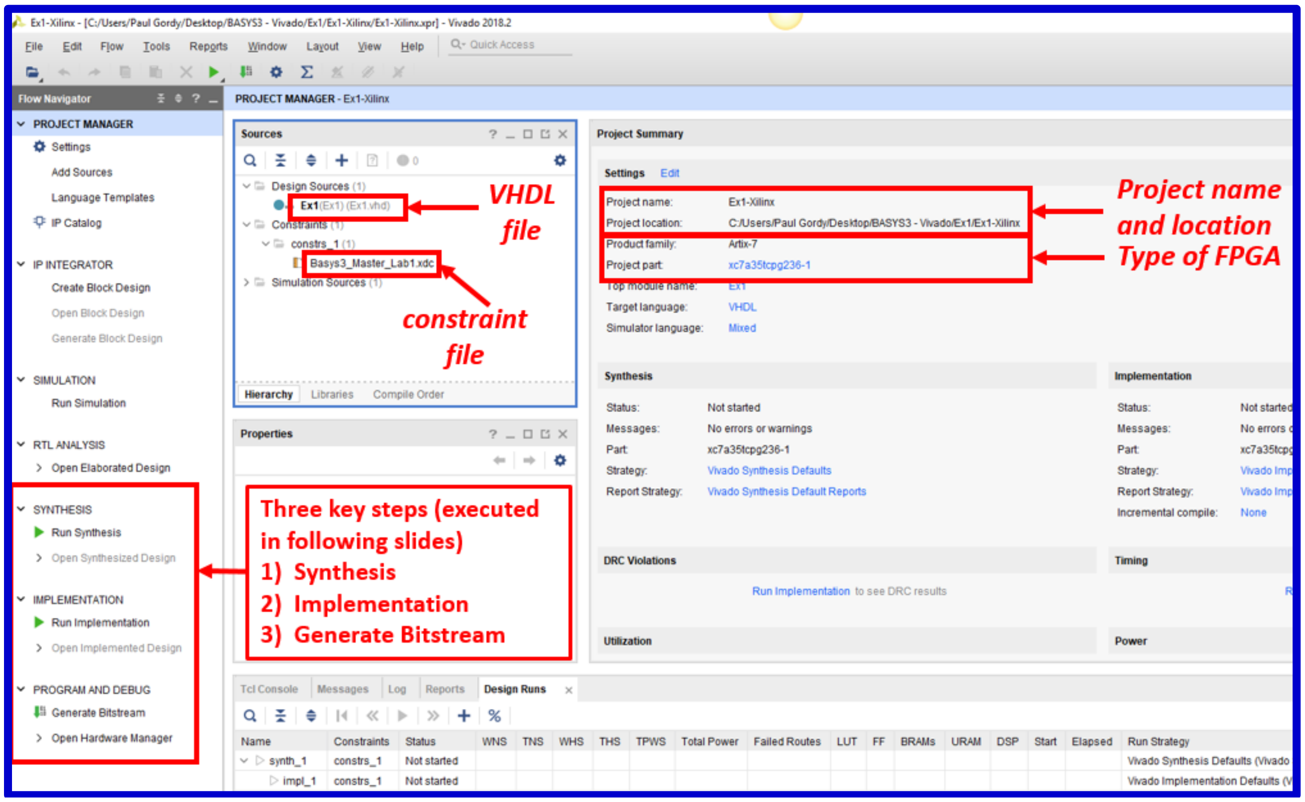
1 source file will be added.

1 constraints file will be added.

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

The main screen in Vivado now appears – a few key features are highlighted



Ex1-Xilinx - [C:/Users/Paul Gordy/Desktop/BASYS3 - Vivado/Ex1/Ex1-Xilinx/Ex1-Xilinx.xpr] - Vivado 2018.2

Project name:	Ex1-Xilinx
Project location:	C:/Users/Paul Gordy/Desktop/BASYS3 - Vivado/Ex1/Ex1-Xilinx
Product family:	Artix-7
Project part:	xc7a35tcpg236-1
Top module name:	Ex1
Target language:	VHDL
Simulator language:	Mixed

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7a35tcpg236-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports

Implementation

Status:	Not started
Messages:	No errors o
Part:	xc7a35tcpg
Strategy:	Vivado Imp
Report Strategy:	Vivado Imp
Incremental compile:	None

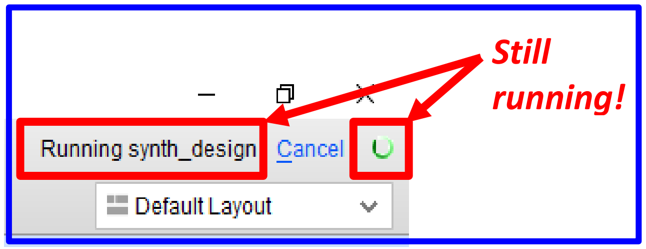
Run Implementation to see DRC results

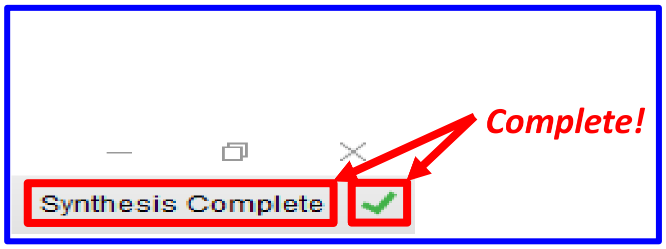
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado
impl_1	constrs_1	Not started															Vivado Implementation Defaults (V

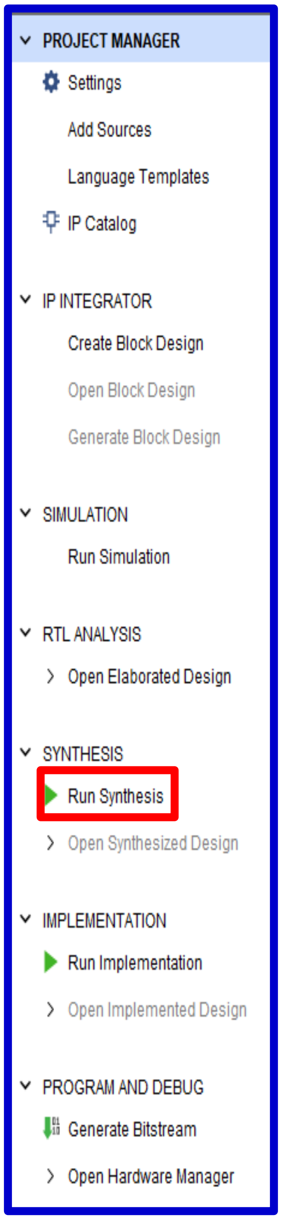
1) Synthesis

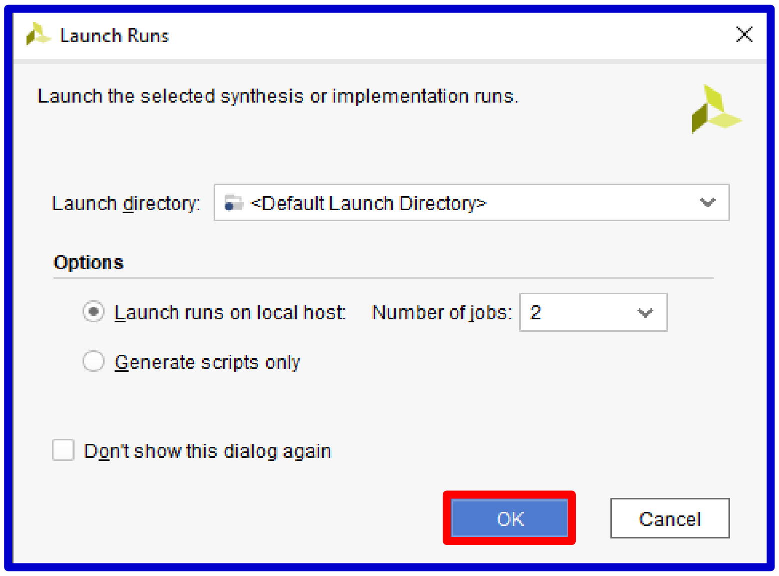
- Under **Synthesis**, select ***Run Synthesis***
- **Launch Runs** - select ***OK***
- **Synthesis Completed** (window will appear when completed) - select ***OK***

  (This may take a few minutes. Is it still running? Check the upper right corner of the main screen.)









Launch Runs

Launch the selected synthesis or implementation runs.

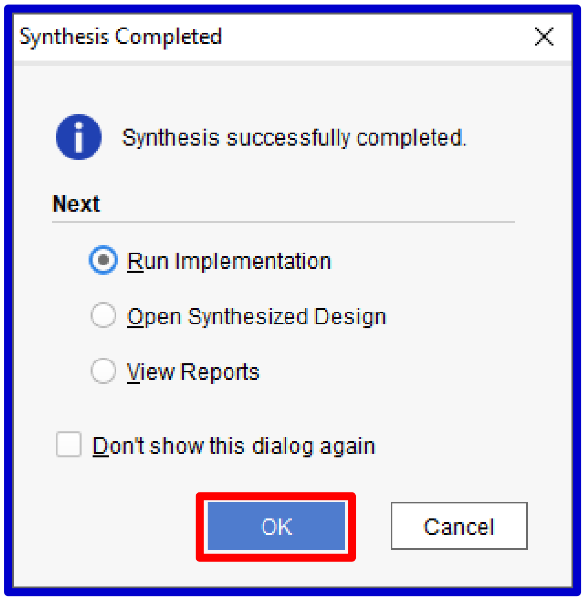
Launch directory: <Default Launch Directory>

Options

[✓ Launch runs on local host:] Number of jobs: 2

Generate scripts only

Don't show this dialog again



Synthesis Completed

Synthesis successfully completed.

Next

[✓ Run Implementation]

Open Synthesized Design

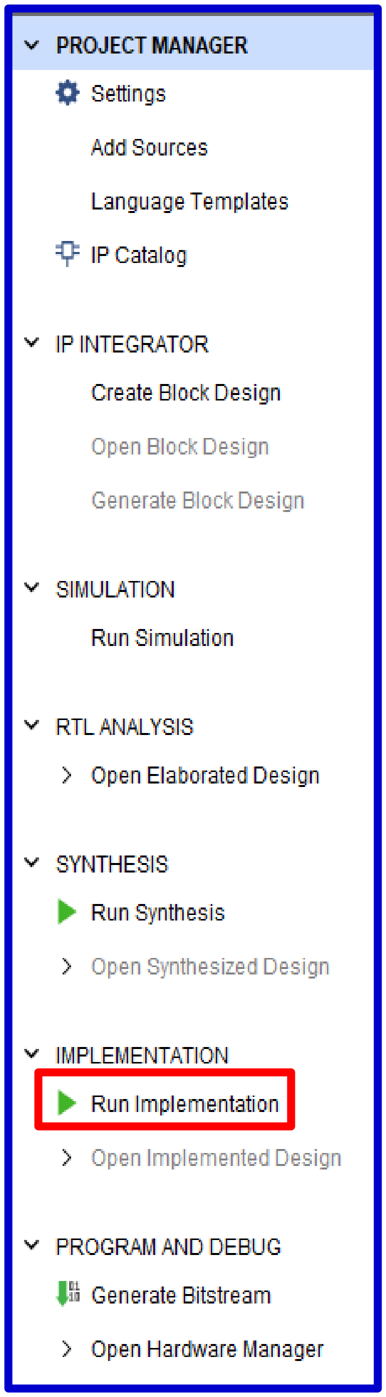
View Reports

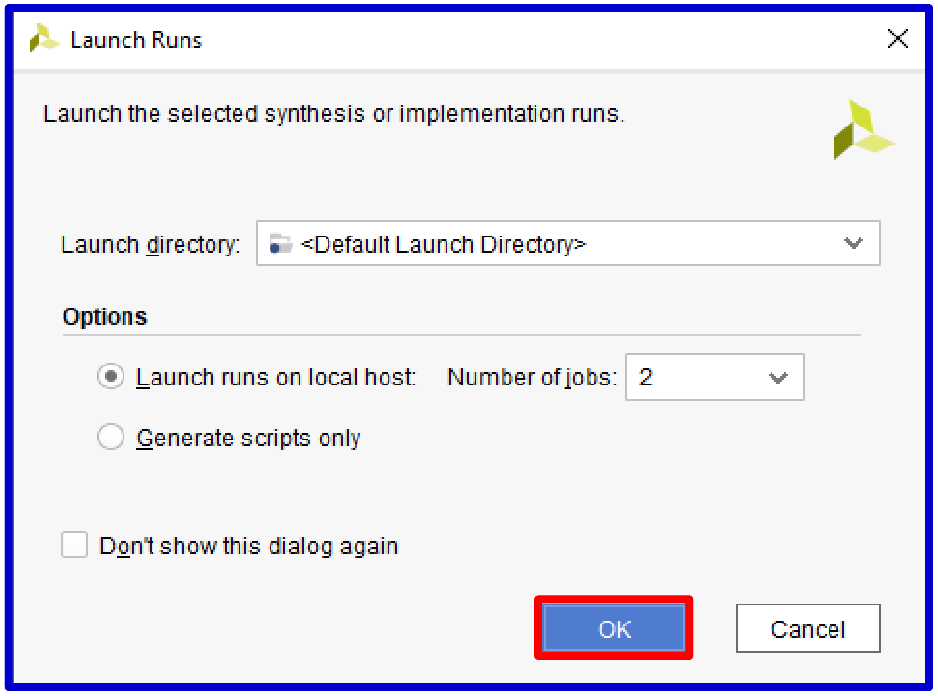
Don't show this dialog again

2) <u>Implementation</u>

- (This may run automatically after synthesis)
- Under **<u>Implementation</u>**, select ***Run Implementation***
- **<u>Launch Runs</u>** - select ***OK***
- **<u>Implementation Completed</u>** (window will appear when completed) - select ***OK***

(This may take a few minutes. Is it still running? Check the upper right corner of the main screen.)




Launch Runs

Launch the selected synthesis or implementation runs.

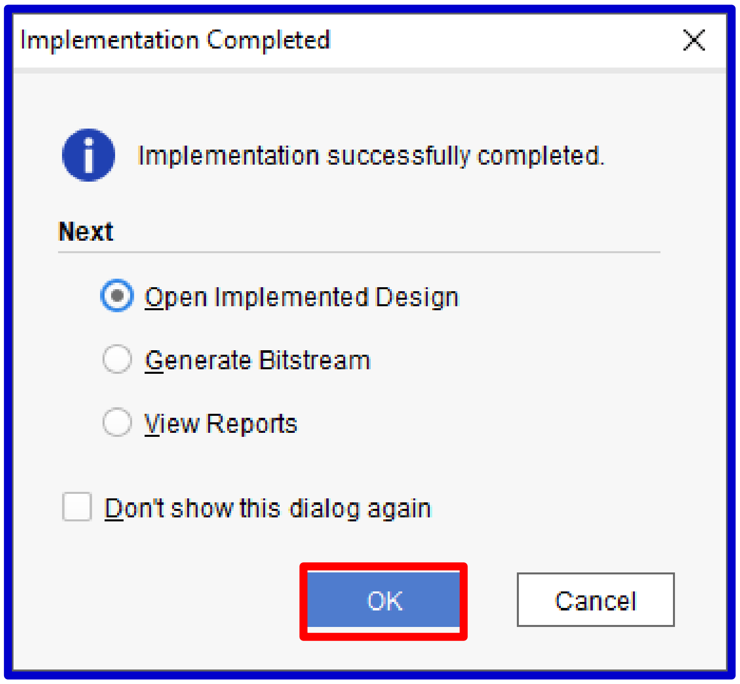
Launch directory: <Default Launch Directory>

Options

[✓ Launch runs on local host:] Number of jobs: 2

Generate scripts only

☐ Don't show this dialog again


Implementation Completed

Implementation successfully completed.

Next

[✓ Open Implemented Design]

Generate Bitstream

View Reports

☐ Don't show this dialog again

Documentation from the Implemented Design

Before proceeding to step 3 where we will generate a bitstream to program the FPGA, there is some useful documentation available when we open the implemented design.
Some items include (and will be printed as part of your lab report):

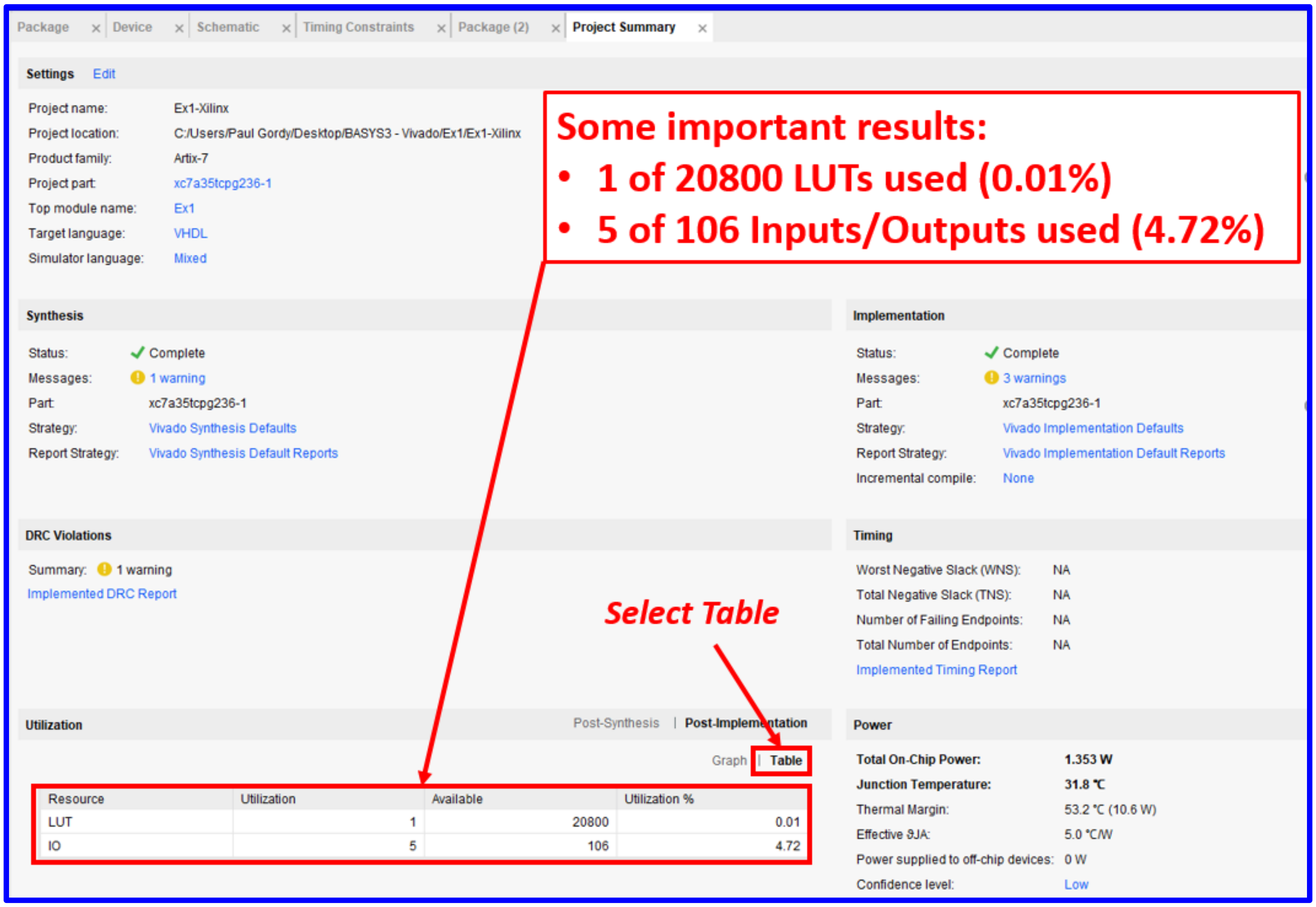
- ***Project Summary*** – Shows project and FPGA information as well as utilization (number of LUTs and number of I/Os used)
- ***Schematic (for implemented design)*** – Shows used LUTs, input buffers, output buffers, etc.
- ***Package View*** – Shows the bottom view of the FPGA showing which of the 106 pins have been used.
- ***IO Ports*** – Shows the used inputs/outputs and the assigned pins

Additionally, if we run an RTL Simulation, we can get an additional gate-level schematic:

- ***Schematic (for RTL simulation)*** – Shows schematic using logic gates. Note that Aldec Active-HDL will automatically minimize logic expressions, so it is possible that the schematic may be different from what you expected (but equivalent).

Project Summary

- Select **Window – Project Summary**
- Under **Utilization** – Select ***Table***



Some important results:
- 1 of 20800 LUTs used (0.01%)
- 5 of 106 Inputs/Outputs used (4.72%)

Settings

Project name:	Ex1-Xilinx
Project location:	C:/Users/Paul Gordy/Desktop/BASYS3 - Vivado/Ex1/Ex1-Xilinx
Product family:	Artix-7
Project part:	xc7a35tcpg236-1
Top module name:	Ex1
Target language:	VHDL
Simulator language:	Mixed

Synthesis

Status:	✓ Complete
Messages:	1 warning
Part:	xc7a35tcpg236-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports

Implementation

Status:	✓ Complete
Messages:	3 warnings
Part:	xc7a35tcpg236-1
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental compile:	None

DRC Violations

Summary: 1 warning
Implemented DRC Report

Timing

Worst Negative Slack (WNS):	NA
Total Negative Slack (TNS):	NA
Number of Failing Endpoints:	NA
Total Number of Endpoints:	NA

Implemented Timing Report

Utilization

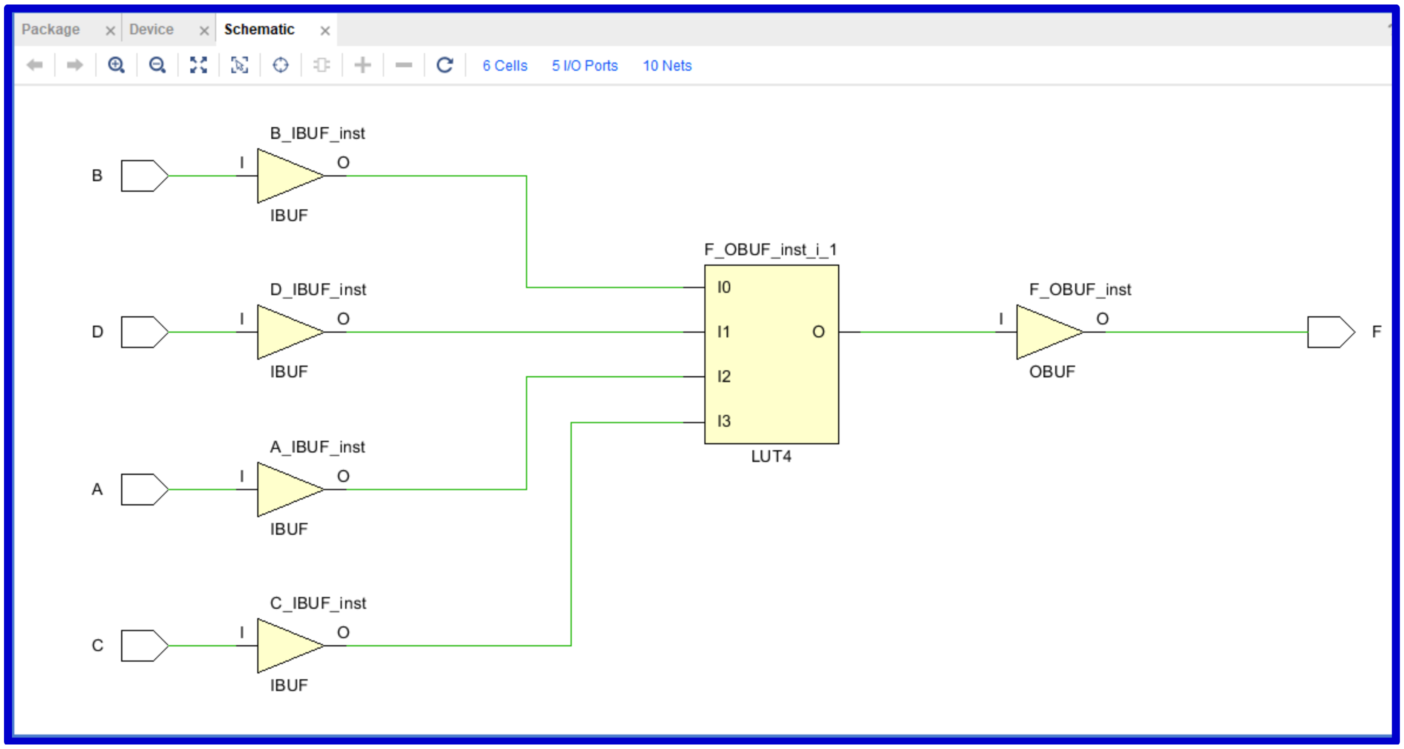
Resource	Utilization	Available	Utilization %
LUT	1	20800	0.01
IO	5	106	4.72

Power

Total On-Chip Power:	**1.353 W**
Junction Temperature:	**31.8 ℃**
Thermal Margin:	53.2 ℃ (10.6 W)
Effective ϑJA:	5.0 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Schematic (from implementation)

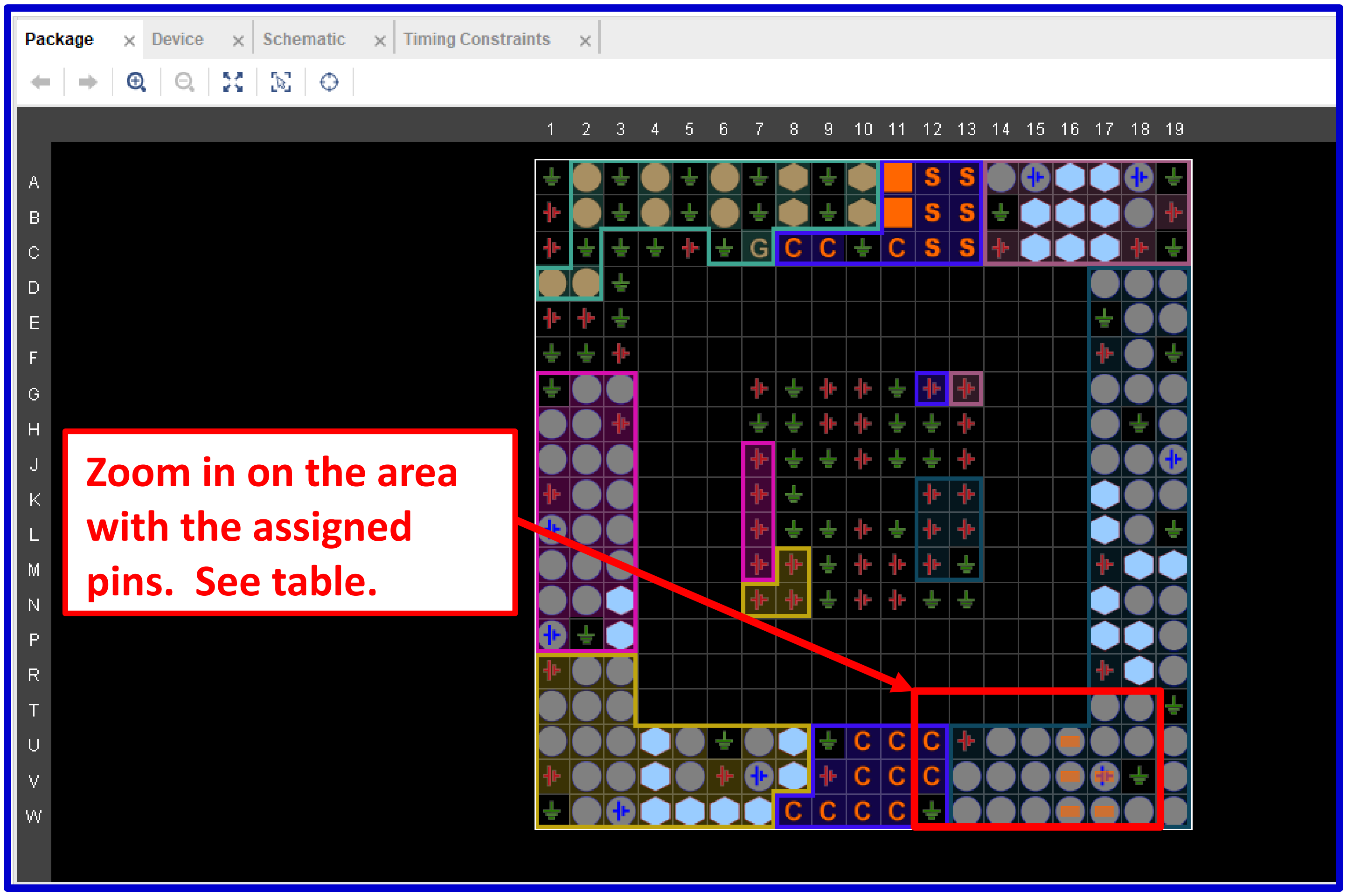
- Shows used LUTs, input buffers, output buffers, etc. LUT4 means that the LUT has 4 inputs.
- Under **Open Implemented Design** – select ***Schematic***



Package View

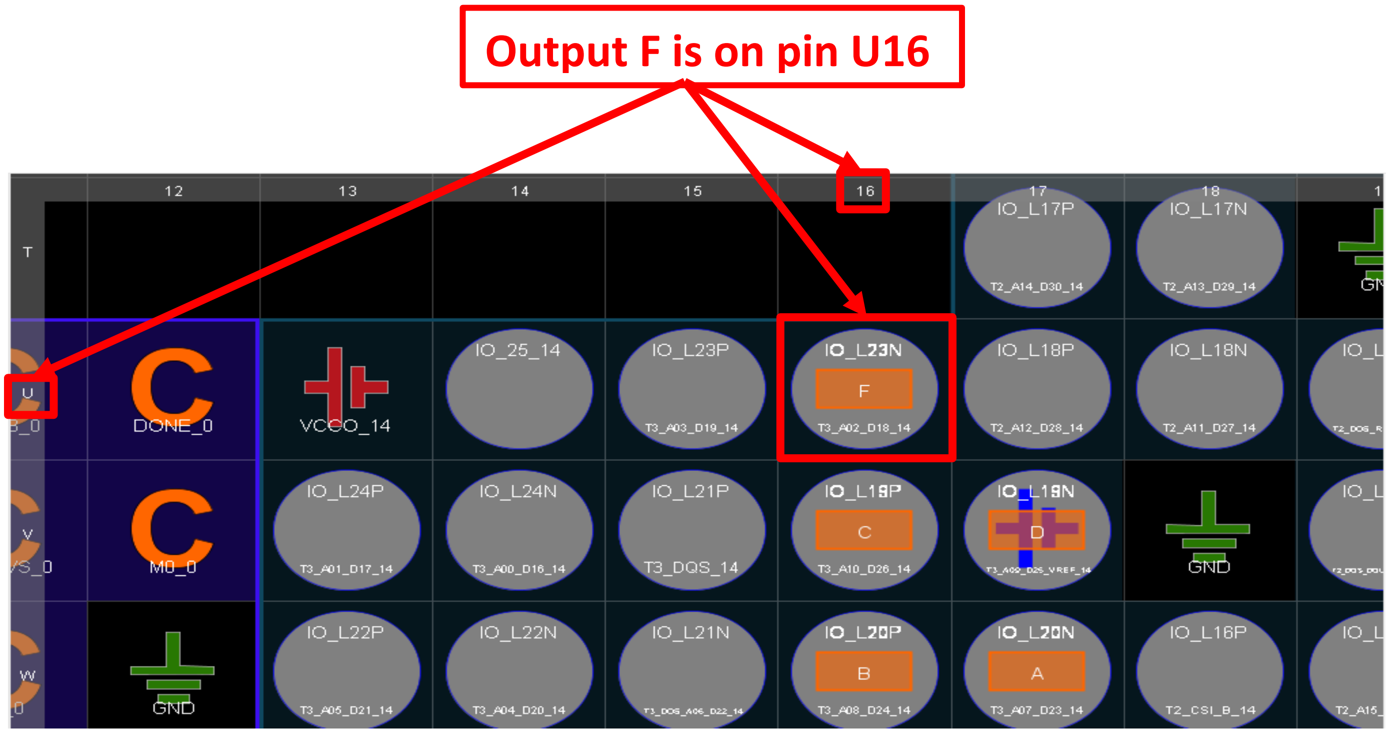
- Shows the bottom view of the FPGA where you can see which of the 106 pins have been used. Note that the pins are arranged in columns 1-19 and rows A-W.
- Under **Layout** – select ***I/O Planning***
- This example used the following pins. ***Zoom in on these pins*** (see next page).

Input/ Output	**BASYS3 Pin**
A	**W17**
B	**W16**
C	**V16**
D	**V17**
F	**U16**


Package
Device
Schematic
Timing Constraints
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
A B C D E F G H J K L M N P R T U V W
Zoom in on the area with the assigned pins. See table.

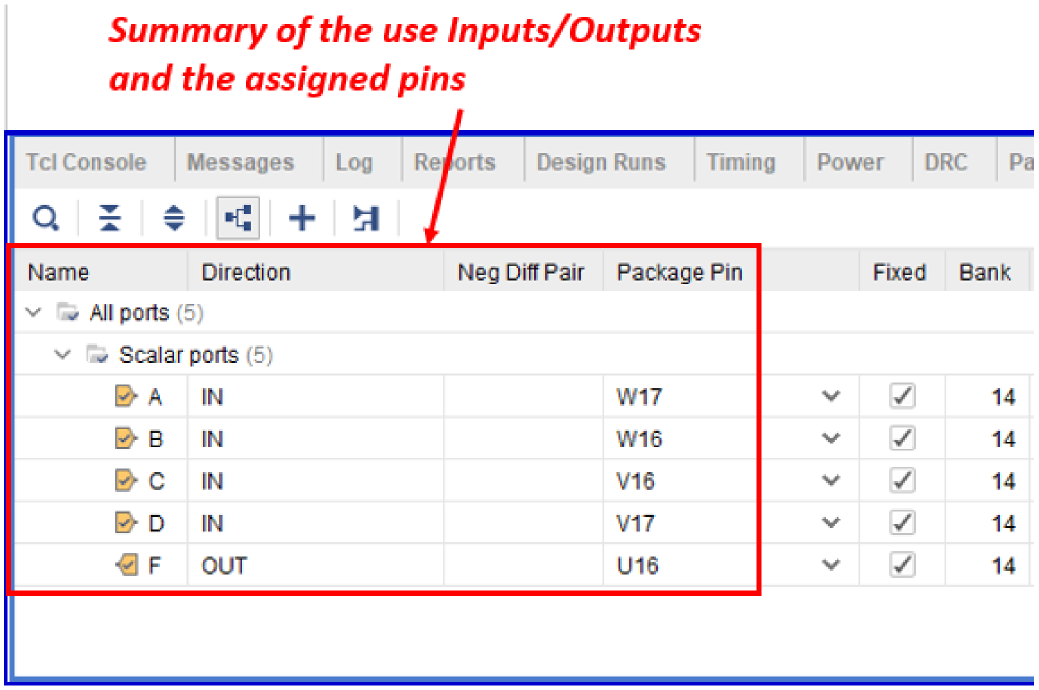
Package View – (continued)

- Note that you can see the signal names A,B,C,D,F after zooming in.



I/O Ports

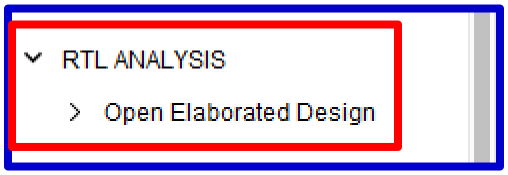
- Shows the used inputs/outputs and the assigned pins
- Select **Window – I/O Ports**
- Expand **scalar ports** to see the signals used

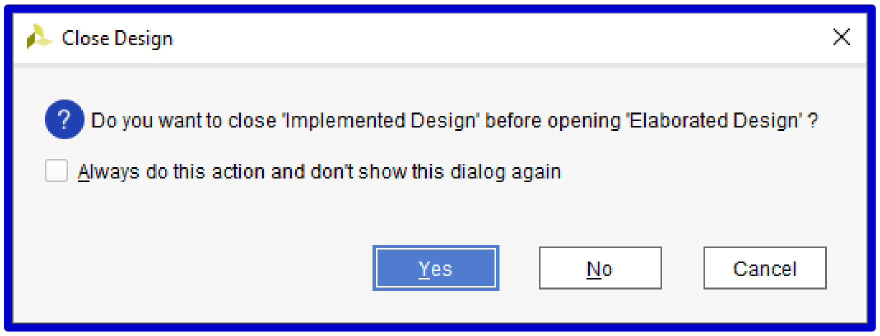


Name	Direction	Neg Diff Pair	Package Pin		Fixed	Bank
All ports (5)						
Scalar ports (5)						
A	IN		W17		✓	14
B	IN		W16		✓	14
C	IN		V16		✓	14
D	IN		V17		✓	14
F	OUT		U16		✓	14

<u>Schematic (from RTL Analysis)</u>

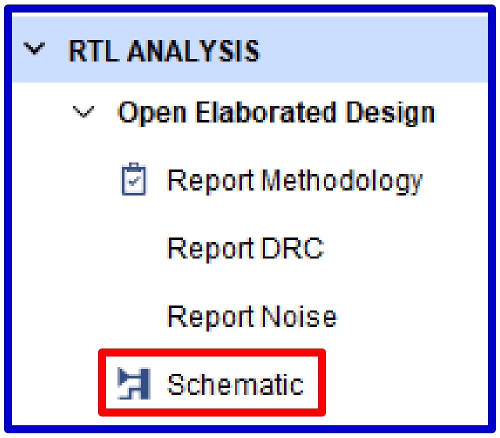
- Under **<u>RTL Analysis</u>**, select ***Open Elaborated Design***
- (Select ***Yes*** and ***OK*** if the two windows below appear)
- Select ***Schematic*** (schematic is shown on the next slide)

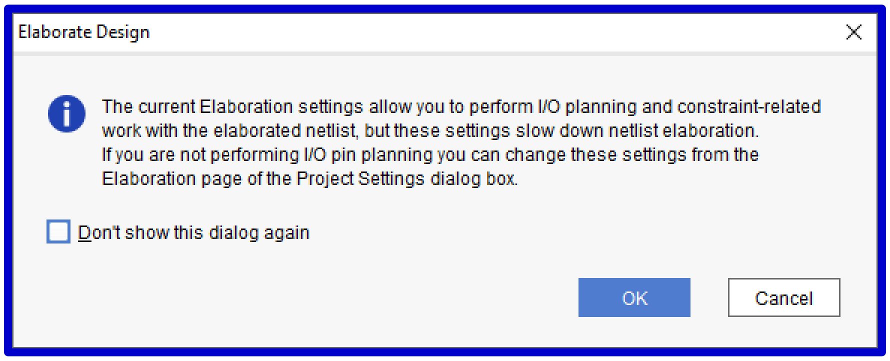



Close Design

Do you want to close 'Implemented Design' before opening 'Elaborated Design' ?

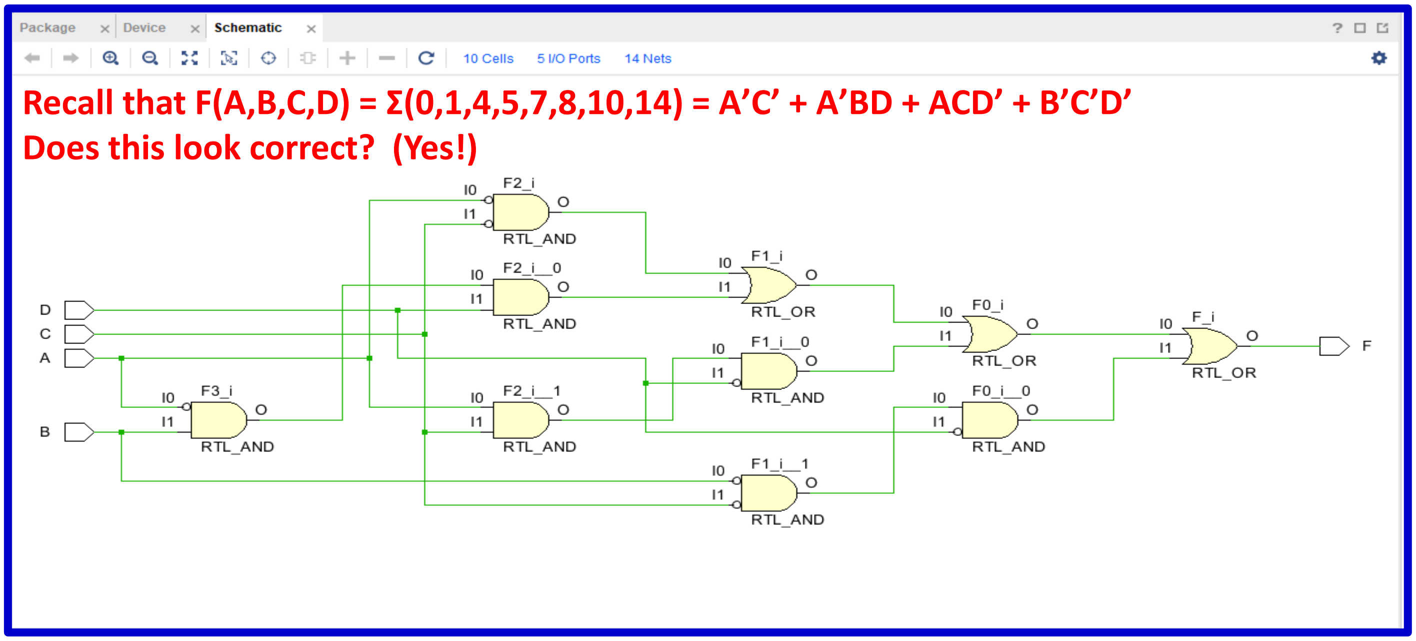
Always do this action and don't show this dialog again




Elaborate Design

The current Elaboration settings allow you to perform I/O planning and constraint-related work with the elaborated netlist, but these settings slow down netlist elaboration.
If you are not performing I/O pin planning you can change these settings from the Elaboration page of the Project Settings dialog box.

Don't show this dialog again



Recall that F(A,B,C,D) = Σ(0,1,4,5,7,8,10,14) = A'C' + A'BD + ACD' + B'C'D'

Does this look correct? (Yes!)

3) Generate the bitstream (to program the FPGA)

Recall that there are three ways to program the FPGA (we will use the first two methods):

A) Program the FPGA directly

- Jumper JP1 must be moved to the middle position (JTAG).
- A *bit file* will be download into the FPGA using the USB-JTAG input.
- The FPGA is SRAM-based (volatile memory), so the design will be lost as soon as the BASYS3 is powered down.

B) Program the FPGA using non-volatile serial (SPI) flash memory on the BASYS3 board

- Jumper JP1 must be originally in the middle position (JTAG).
- A *binary file* will be downloaded into the FPGA using the USB-JTAG input.
- The BASYS3 board must be powered down.
- Jumper JP1 must then be moved to the top middle position (QSPI).
- Now every time the BASYS3 board is powered up or the reset button is pushed, the FPGA will be reprogrammed from the onboard flash memory.

C) Program the FPGA using a USB memory device attached to the USB HID port

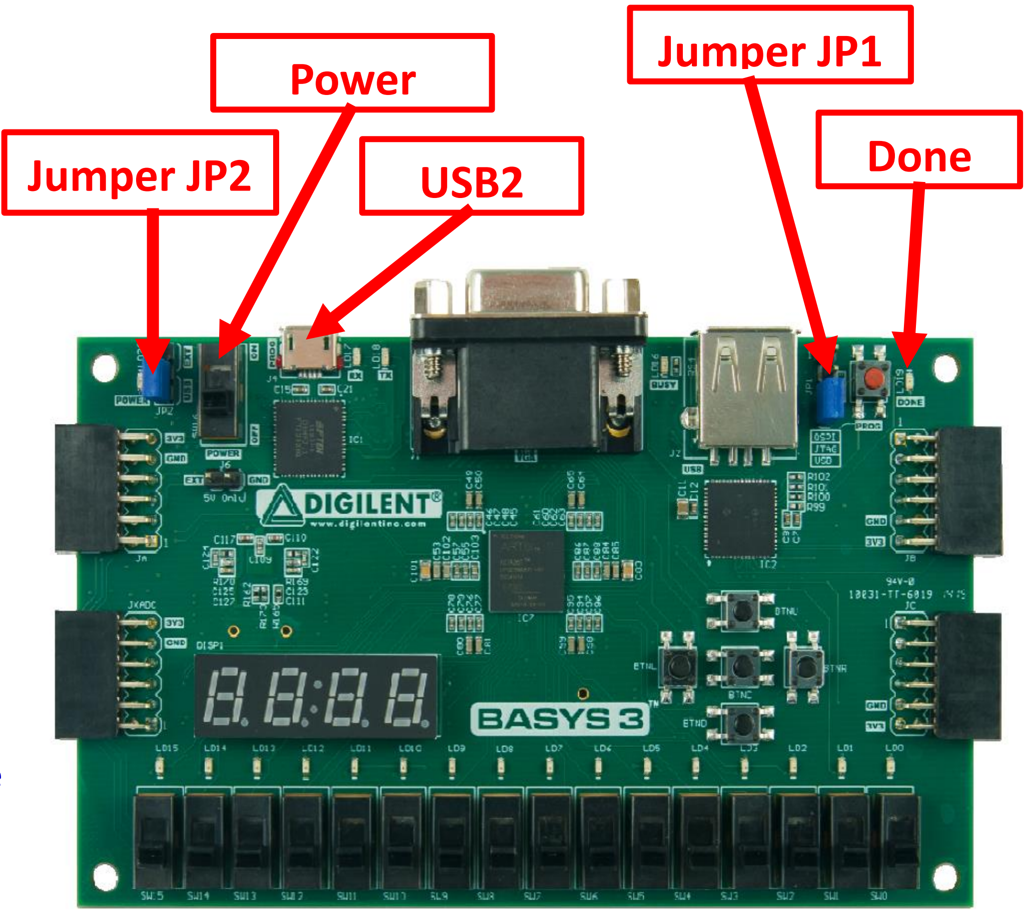
- Jumper JP1 must be moved to the bottom position (USB). We will not use this method.

Methods A and B will be covered in the following pages.

Programming the FPGA

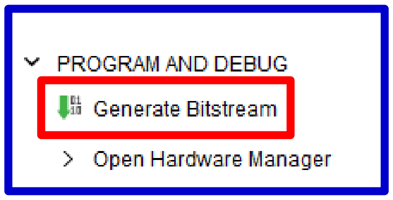
Before we generate the bitstream to program the FPGA, we need to connect it to the computer.

- ***USB Cable***: Connect the BASYS3 board to the computer using a USB cable.
- ***Jumper JP2***: Move Jumper JP2 to the **USB** position (not Ext) as we will power the board via the USB instead of using an external power source.
- ***Power***: Turn on the power switch.
- ***Jumper JP1***: Move Jumper JP1 to the middle (JTAG) position.
- ***Done LED***: Turns on once the design has been downloaded into the BASYS3 board

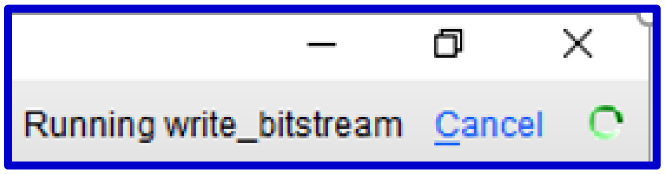


Generate the bitstream – **Method A: Program the FPGA directly**

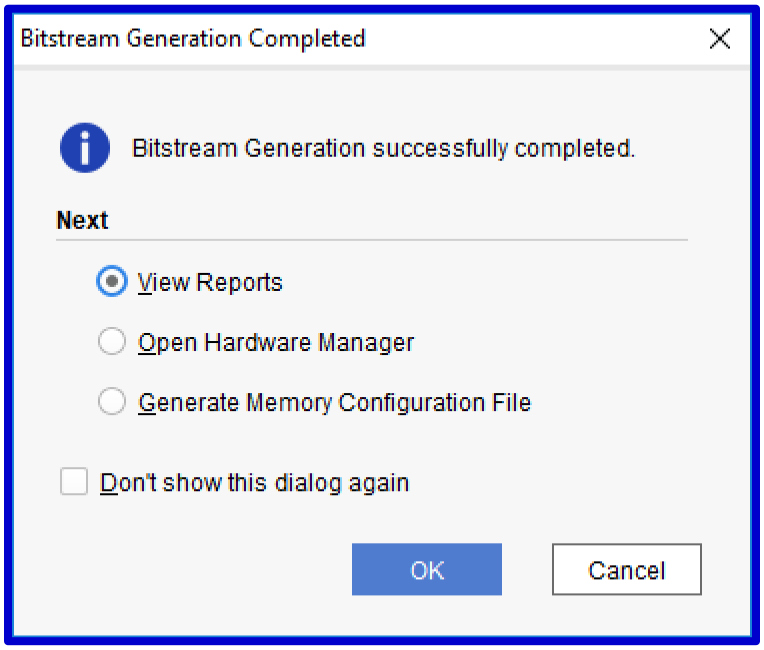
- Check to be sure that Jumper JP1 has been moved to the middle position (JTAG).
- Under **Program and Debug** (scroll down in the Flow Navigator), select ***Generate Bitstream***



- If the **Launch Runs** window appears, select ***OK***.
- This may take a while to run. Check the status in the upper right corner of the screen.



- **Bitstream Generation Completed** (window will appear when completed) - select ***OK***.



Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

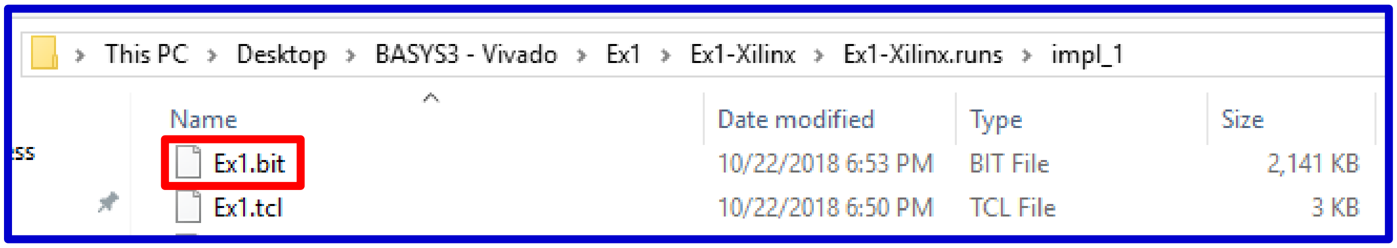
[✓ View Reports]

Open Hardware Manager

Generate Memory Configuration File

Don't show this dialog again

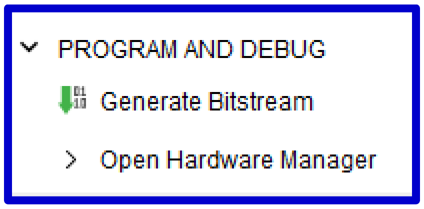
- When it finishes it makes a bit file that will be downloaded into the FPGA (this image is just shown for reference)

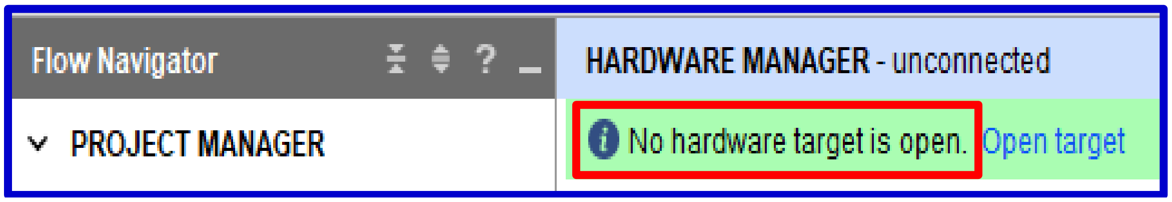


This PC › Desktop › BASYS3 - Vivado › Ex1 › Ex1-Xilinx › Ex1-Xilinx.runs › impl_1

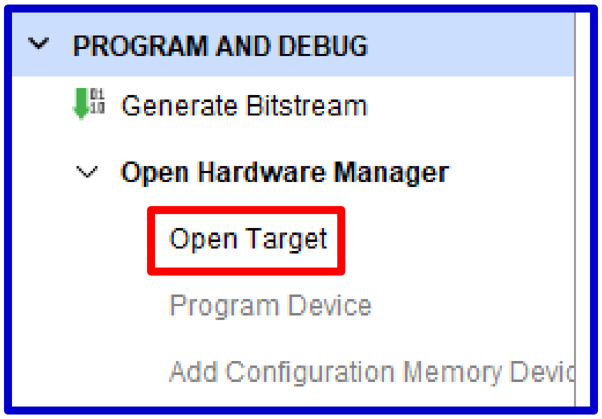
Name	Date modified	Type	Size
Ex1.bit	10/22/2018 6:53 PM	BIT File	2,141 KB
Ex1.tcl	10/22/2018 6:50 PM	TCL File	3 KB

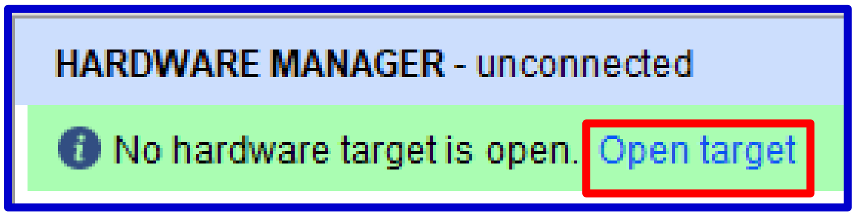
- Under **Program and Debug**, select ***Open Hardware Manager***



- After opening the **Hardware Manager**, you may see the window below with the message: ***No hardware target is open.***
- 

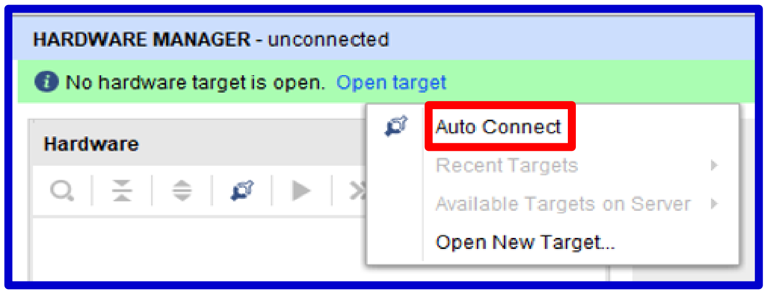
- Under **Program and Debug** (or in the **Hardware Manager** window), select ***Open Target***

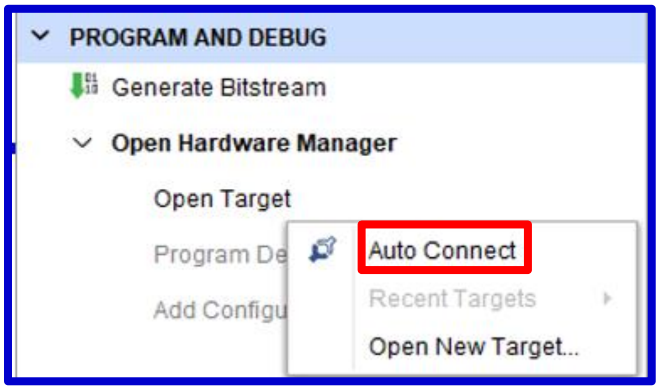




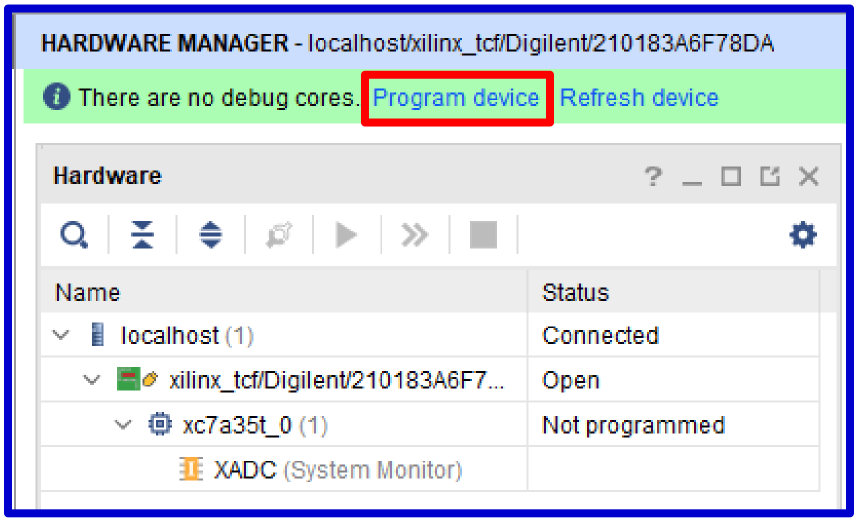
Generate the bitstream – Method A (continued)

- Select ***Auto Connect*** (use either window below)


HARDWARE MANAGER - unconnected
No hardware target is open. Open target
Hardware
Auto Connect
Recent Targets
Available Targets on Server
Open New Target...

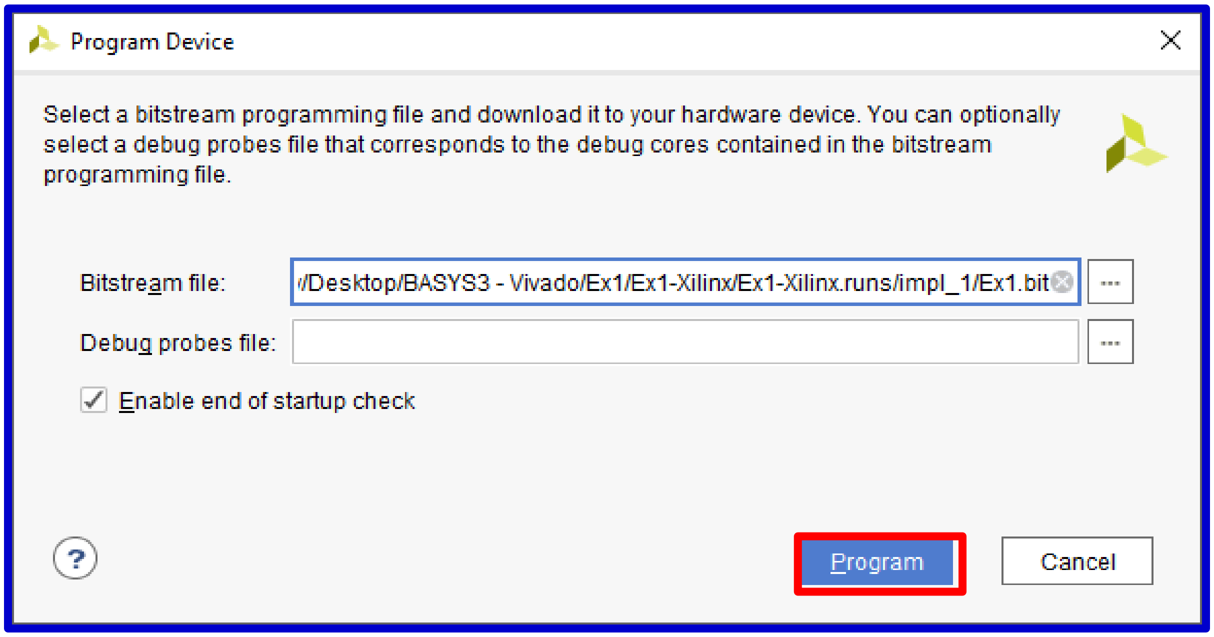


- Select ***Program Device***


HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/210183A6F78DA
There are no debug cores. Program device Refresh device
Hardware

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/210183A6F7...	Open
xc7a35t_0 (1)	Not programmed
XADC (System Monitor)	

- Select ***Program*** (note that the file Ex1.bit was automatically filled in for the Bitstream file)



Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

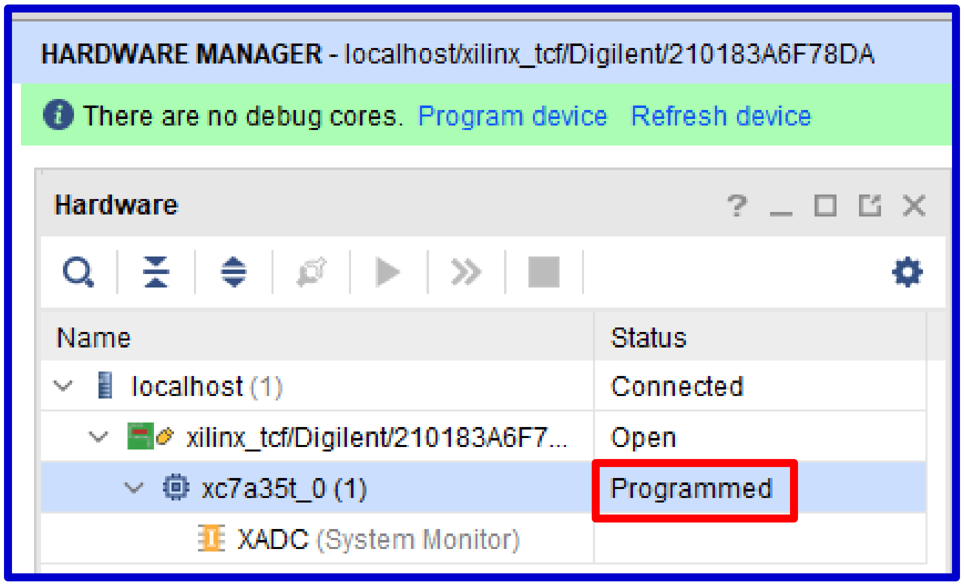
Bitstream file: /Desktop/BASYS3 - Vivado/Ex1/Ex1-Xilinx/Ex1-Xilinx.runs/impl_1/Ex1.bit

Debug probes file:

✓ Enable end of startup check

Generate the bitstream – **Method A (continued)**

- The **Hardware Manager** should now indicate that the FPGA has been ***Programmed.***


HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/210183A6F78DA

There are no debug cores. Program device Refresh device

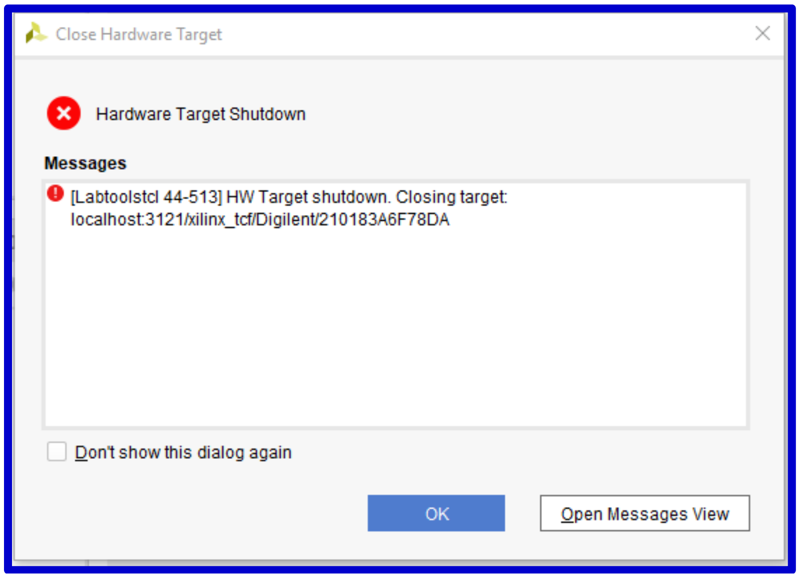
Hardware

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/210183A6F7...	Open
xc7a35t_0 (1)	Programmed
XADC (System Monitor)	

- Test the FPGA (move slide switches SW3 – SW0 corresponding to inputs A-D and view the output F on LED0. Recall that F(A,B,C,D) = Σ(0,1,4,5,7,8,10,14) for this example.

Recall that the FPGA is SRAM-based (volatile memory), so ***the design will be lost*** when you turn off the BASYS3 board or press the Reset button on the BASYS3 board.

- ***Press the Reset button*** on the FPGA board. Test the output for several input combinations and verify that it no longer produces the output.
- **Turn Off the BASYS3 board**. You will see that the Hardware Manager noticed the loss of connection to the FPGA. Turn the FPGA board back on and once again select ***Program Device*** to reprogram the FPGA. Verify that the output on LED0 is correct again. (If you clicked OK on the window below, you may need to Open the target again)


Close Hardware Target

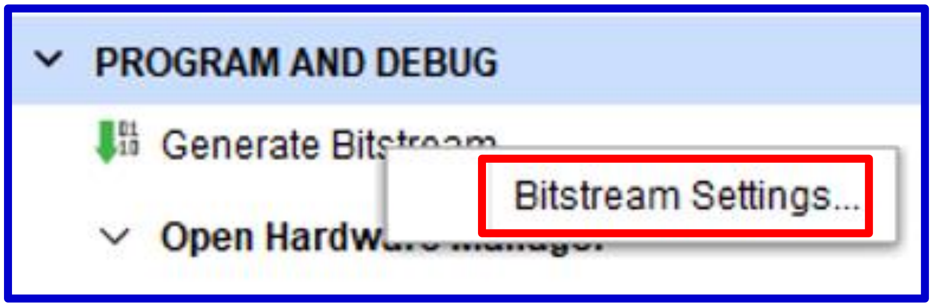
Hardware Target Shutdown

Messages

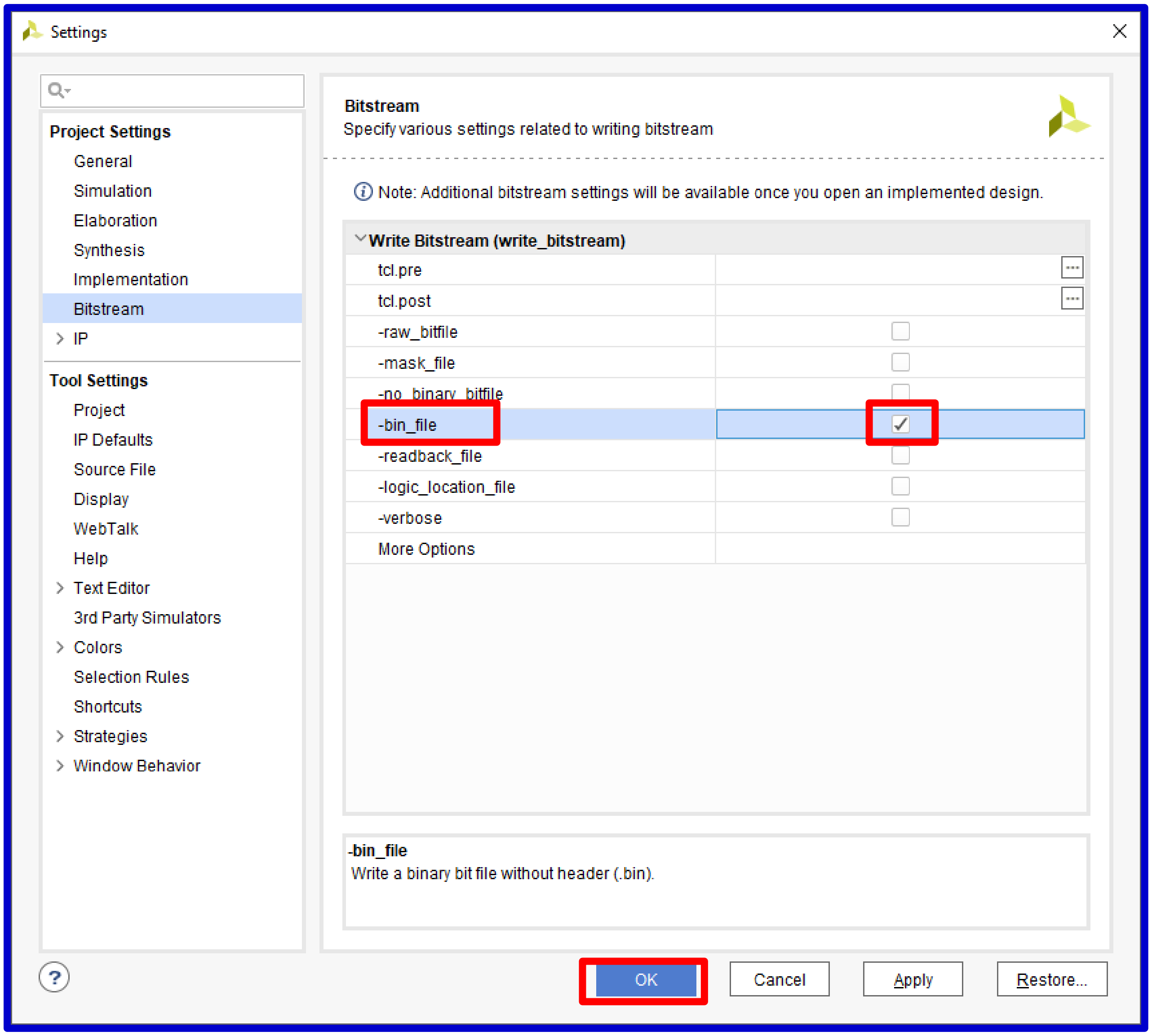
[Labtoolstcl 44-513] HW Target shutdown. Closing target: localhost:3121/xilinx_tcf/Digilent/210183A6F78DA

Generate the bitstream – **Method B: Program the FPGA using serial flash memory**

- ***Jumper JP1***: Move it originally be in the middle position (JTAG). (It will be moved to a different position shortly.)
- This method requires creating a bin file rather than a bit file as follows:
- Right-click on ***Generate Bitstream*** and select ***Bitstream Settings***



- ***Check the box*** next to ***-bin_file*** and then select ***OK***



Settings

Bitstream
Specify various settings related to writing bitstream

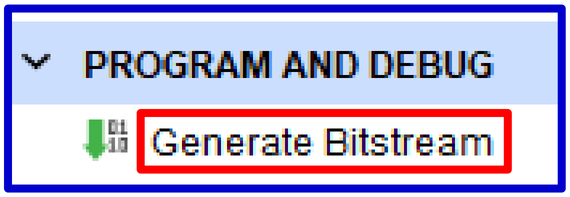
Note: Additional bitstream settings will be available once you open an implemented design.

Write Bitstream (write_bitstream)

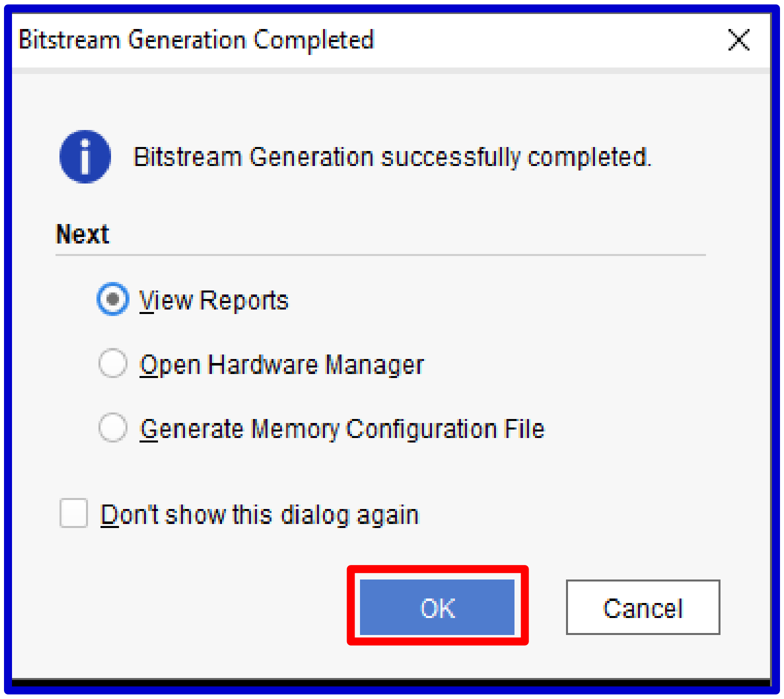
tcl.pre	
tcl.post	
-raw_bitfile	☐
-mask_file	☐
-no_binary_bitfile	☐
-bin_file	✓
-readback_file	☐
-logic_location_file	☐
-verbose	☐
More Options	

-bin_file
Write a binary bit file without header (.bin).

- Close the **Hardware Manager** if currently open.
- Under **Program and Debug**, select ***Generate Bitstream***



- If the **Bitstream Generation Completed** window appears, select **OK**.


Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

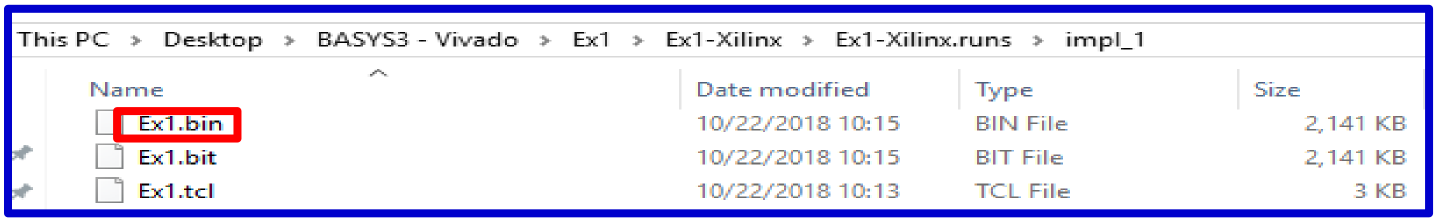
[✓ View Reports]

Open Hardware Manager

Generate Memory Configuration File

Don't show this dialog again

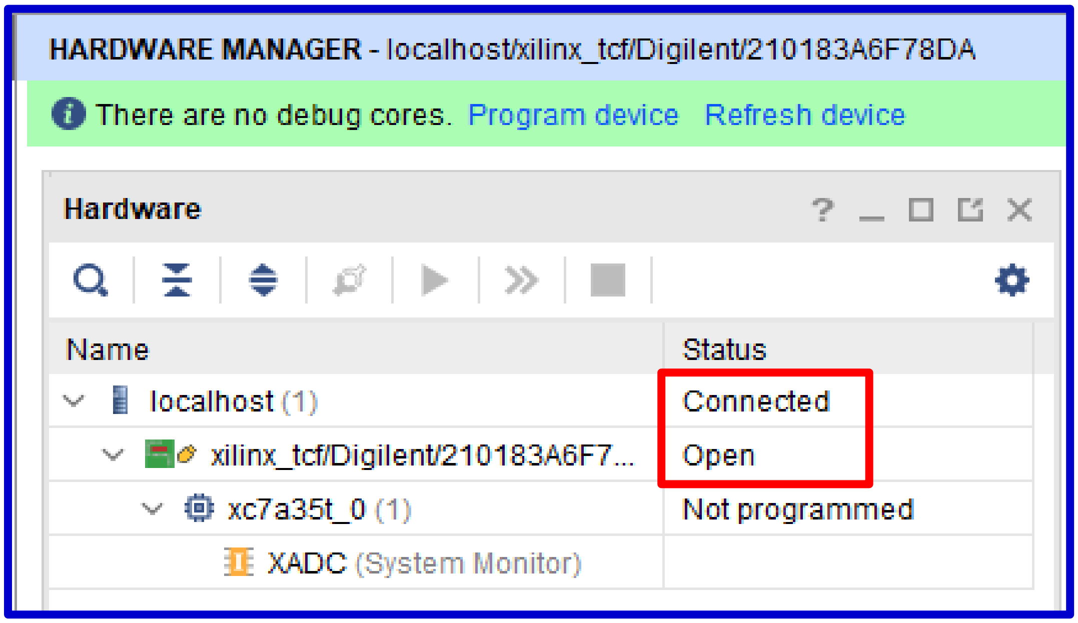
- Note that a **bin file** has been created. This image is just shown for reference, but you will need to know the location of this file later. It should be in the ***/YourProjectFolder/YourFileName.runs/impl_1/ folder***.
(in this example in the ***/Ex1-Xilinx/Ex-Xilinx.runs/impl_1/*** folder)


This PC > Desktop > BASYS3 - Vivado > Ex1 > Ex1-Xilinx > Ex1-Xilinx.runs > impl_1

Name	Date modified	Type	Size
Ex1.bin	10/22/2018 10:15	BIN File	2,141 KB
Ex1.bit	10/22/2018 10:15	BIT File	2,141 KB
Ex1.tcl	10/22/2018 10:13	TCL File	3 KB

As shown earlier under Method A:

- Under **<u>Program and Debug</u>**, select ***Open Hardware Manager***
- Under **<u>Program and Debug</u>** (or in the **<u>Hardware Manager</u>** window), select ***Open Target***
- Select ***Auto Connect***
- You should now see that the hardware is connected



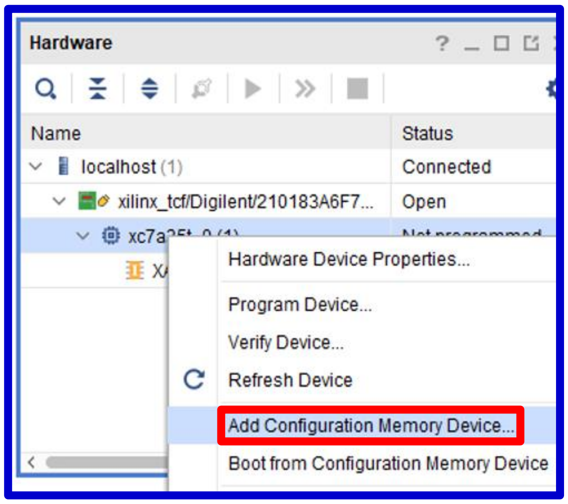
HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/210183A6F78DA

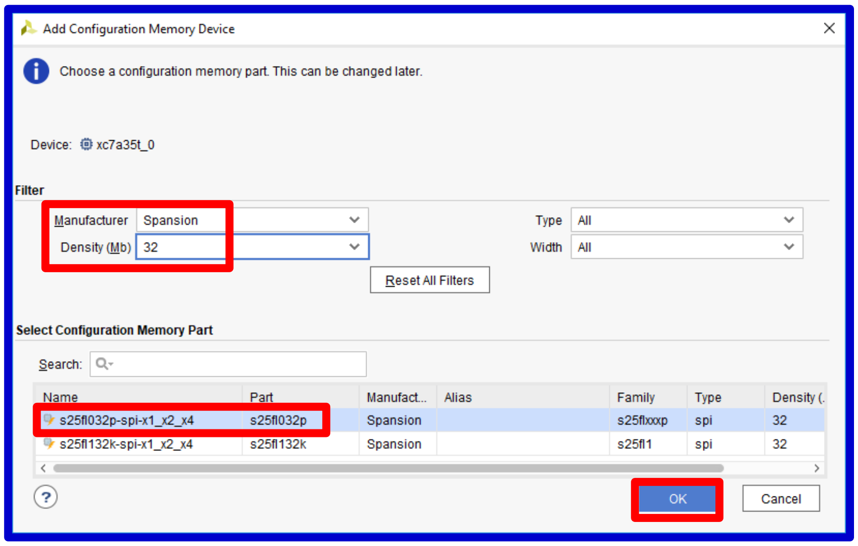
There are no debug cores. Program device Refresh device

Hardware

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/210183A6F7...	Open
xc7a35t_0 (1)	Not programmed
XADC (System Monitor)	

- ***Right- click on the FPGA*** (xc7a35t_0) and select ***Add Configuration Memory Device***
- When the **<u>Add Configuration Memory Device</u>** window opens:
    - **<u>Manufacturer</u>** – Select ***Spansion***
    - **<u>Density</u>** – Select ***32MB***
    - **<u>Name</u>** – Select ***s25fl032p-spi-x1_x2_x4***
    - Select ***OK***.


Hardware
Name
Status
localhost (1)
Connected
xilinx_tcf/Digilent/210183A6F7...
Open
Hardware Device Properties...
Program Device...
Verify Device...
Refresh Device
Add Configuration Memory Device...
Boot from Configuration Memory Device



Add Configuration Memory Device

Choose a configuration memory part. This can be changed later.

Device: xc7a35t_0

Filter

Manufacturer: Spansion
Density (Mb): 32
Type: All
Width: All

Reset All Filters

Select Configuration Memory Part

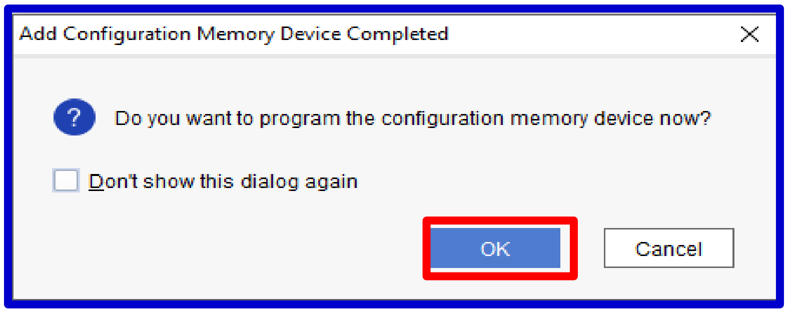
Search:

Name	Part	Manufact...	Alias	Family	Type	Density (.
s25fl032p-spi-x1_x2_x4	s25fl032p	Spansion		s25flxxxp	spi	32
s25fl132k-spi-x1_x2_x4	s25fl132k	Spansion		s25fl1	spi	32

OK Cancel

(The 32MB serial flash memory on the BASYS3 is made by Spansion)

- Do you want to program the configuration memory device now? Select ***OK***

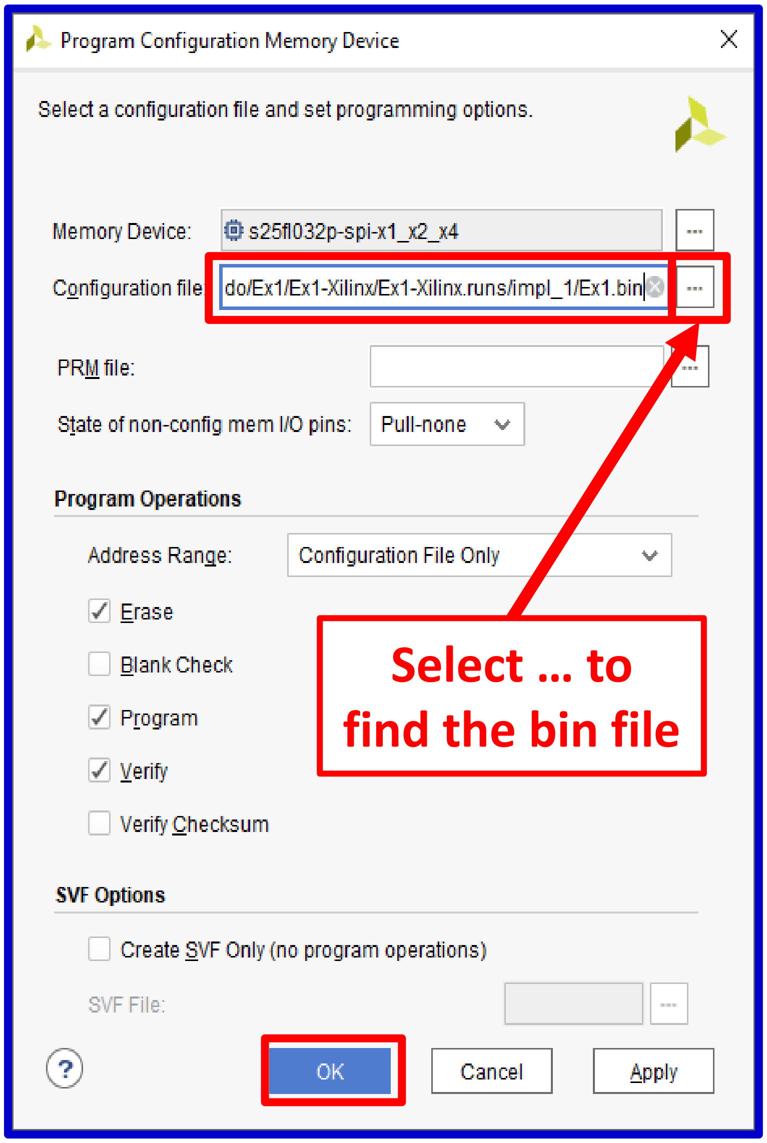

Add Configuration Memory Device Completed

Do you want to program the configuration memory device now?

☐ Don't show this dialog again

<u>Program Configuration Memory Device</u> window:

- ***Select …*** to find the bin file
- Select the **<u>bin file</u>**. The file and path should appear
- ***…/Ex1-Xilinx/Ex-Xilinx.runs/impl_1/Ex1.bin*** for this example
- Select ***OK***


Program Configuration Memory Device

Select a configuration file and set programming options.

Memory Device: s25fl032p-spi-x1_x2_x4

Configuration file: do/Ex1/Ex1-Xilinx/Ex1-Xilinx.runs/impl_1/Ex1.bin

PRM file:

State of non-config mem I/O pins: Pull-none

Program Operations

Address Range: Configuration File Only

☑ Erase

☐ Blank Check

☑ Program

☑ Verify

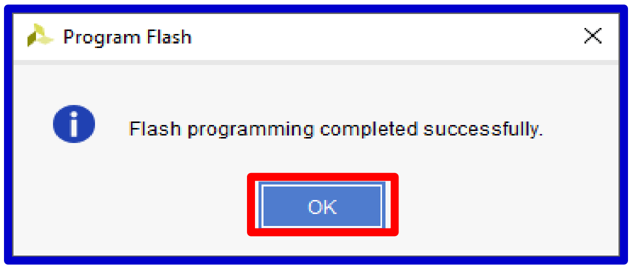
☐ Verify Checksum

SVF Options

☐ Create SVF Only (no program operations)

SVF File:

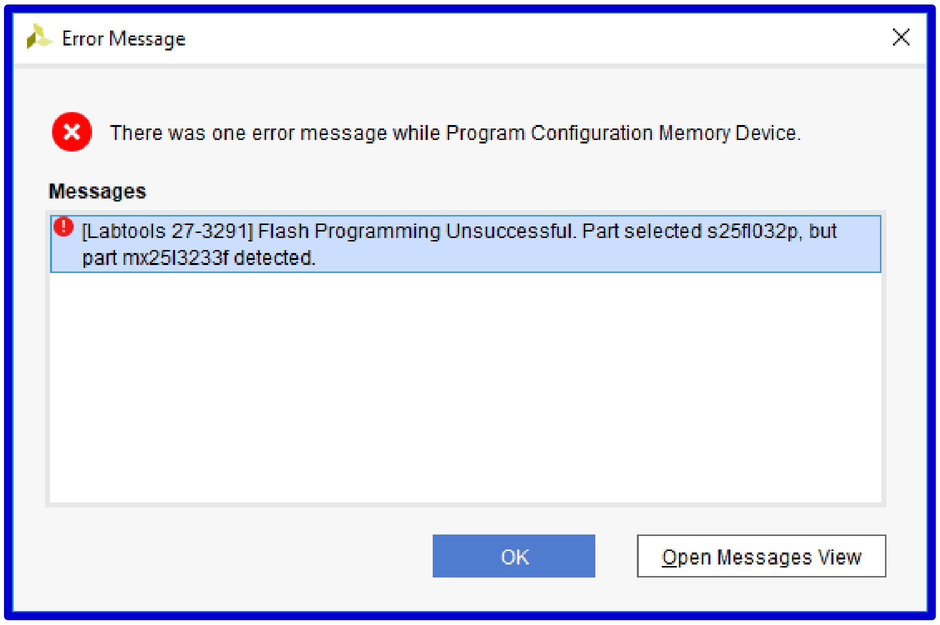
- It may take a few minutes to program the flash memory and then the **<u>Program Flash</u>** window will appear
- Select ***OK***


Program Flash

Flash programming completed successfully.

You might get an error message like the one below about the memory chip selected because apparently some of our BASYS3 boards use the Spansion flash memory chip and some use the Macronix flash memory chip.

- If this message occurs, select ***OK***. (If not skip ahead to **In order to use the design loaded into flash memory:**)



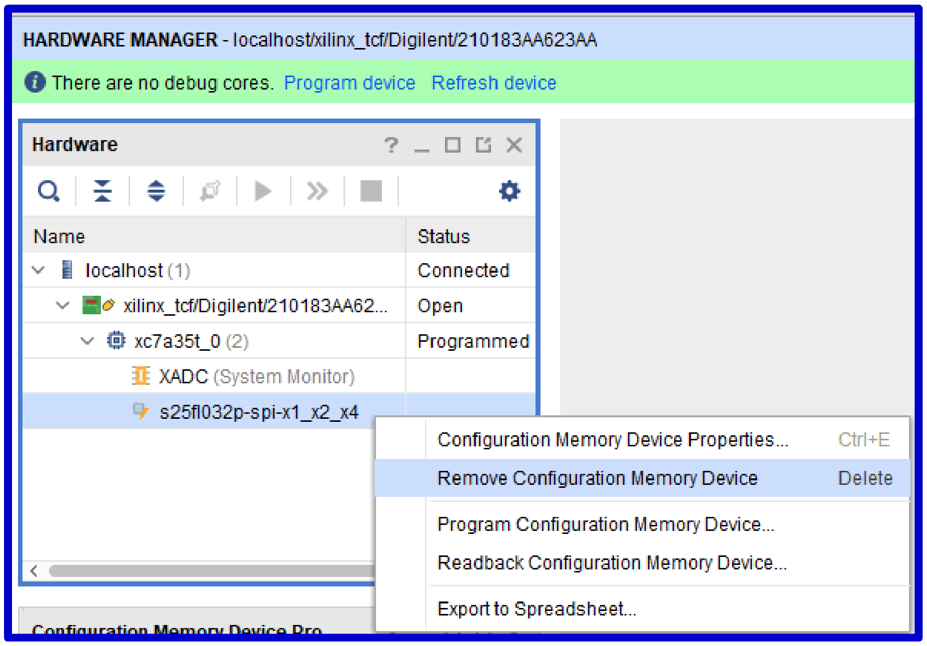
Error Message

There was one error message while Program Configuration Memory Device.

Messages

[Labtools 27-3291] Flash Programming Unsuccessful. Part selected s25fl032p, but part mx25l3233f detected.

- Right-click on the Spansion memory device (s25fl032p-spi-x1_x2_x4) and select ***Remove Configuration Memory Device***



HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/210183AA623AA

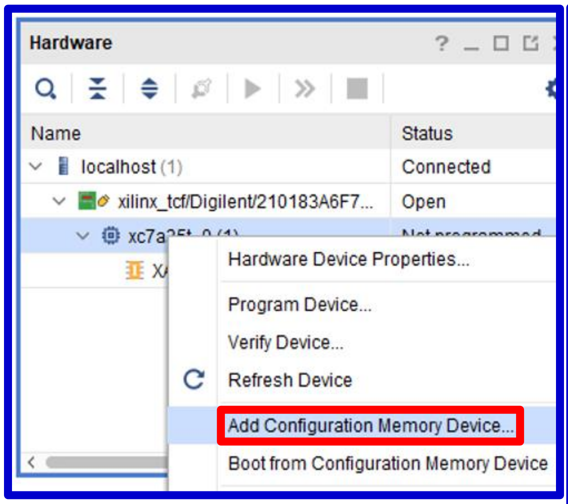
There are no debug cores. Program device Refresh device

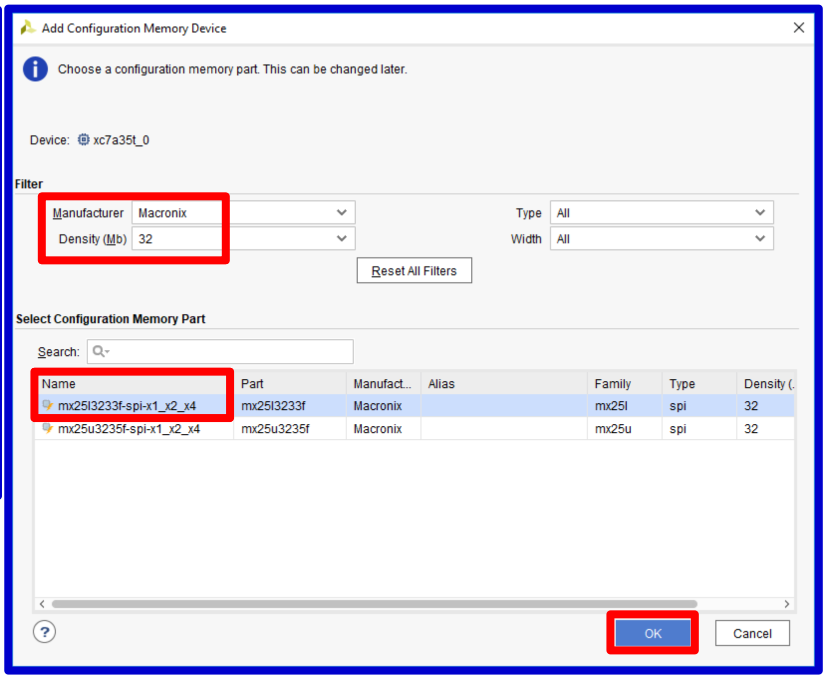
Hardware

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/210183AA62...	Open
xc7a35t_0 (2)	Programmed
XADC (System Monitor)	
s25fl032p-spi-x1_x2_x4	

Configuration Memory Device Properties... Ctrl+E
Remove Configuration Memory Device Delete
Program Configuration Memory Device...
Readback Configuration Memory Device...
Export to Spreadsheet...
Configuration Memory Device Pro

- ***Right- click on the FPGA*** (xc7a35t_0) and select ***Add Configuration Memory Device***
- When the **Add Configuration Memory Device** window opens:
  - **Manufacturer** – Select ***Macronix***
  - **Density** – Select ***32MB***
  - **Name** – Select ***mx2513233f-spi-x1_x2_x4***
  - Select ***OK***.


Hardware
Name
Status
localhost (1)
Connected
xilinx_tcf/Digilent/210183A6F7...
Open
Hardware Device Properties...
Program Device...
Verify Device...
Refresh Device
Add Configuration Memory Device...
Boot from Configuration Memory Device


Add Configuration Memory Device

Choose a configuration memory part. This can be changed later.

Device: xc7a35t_0

Filter

Manufacturer: Macronix
Density (Mb): 32
Type: All
Width: All

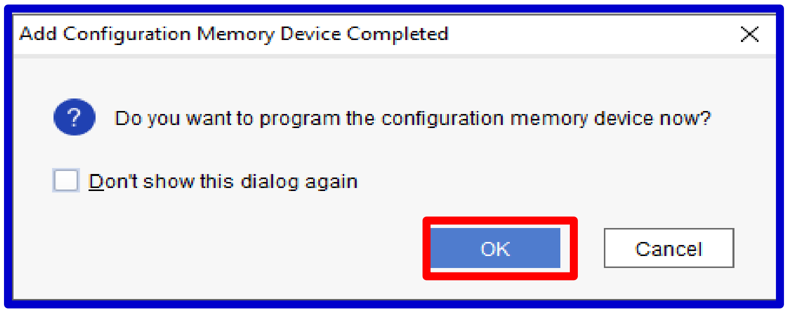
Reset All Filters

Select Configuration Memory Part

Search:

Name	Part	Manufact...	Alias	Family	Type	Density (...
mx25l3233f-spi-x1_x2_x4	mx25l3233f	Macronix		mx25l	spi	32
mx25u3235f-spi-x1_x2_x4	mx25u3235f	Macronix		mx25u	spi	32

- Do you want to program the configuration memory device now? Select ***OK***

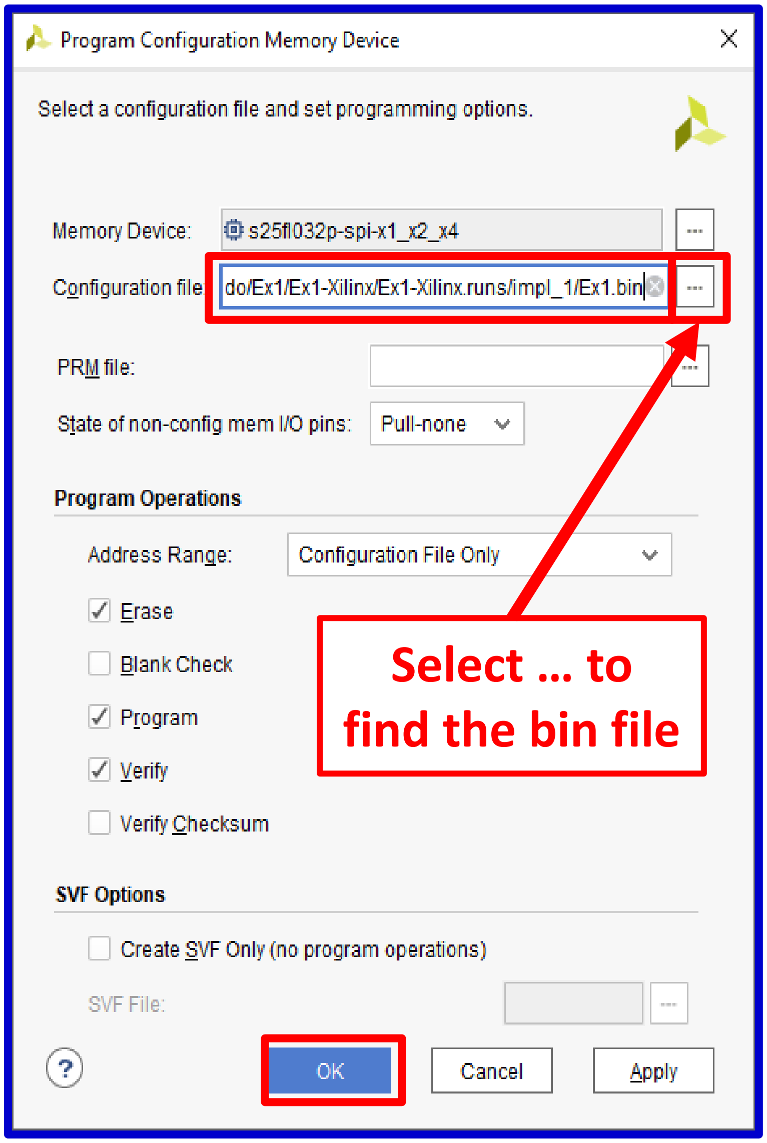

Add Configuration Memory Device Completed

Do you want to program the configuration memory device now?

Don't show this dialog again

<u>Program Configuration Memory Device</u> window:

- ***Select …*** to find the bin file
- Select the **<u>bin file</u>**. The file and path should appear
- ***…/Ex1-Xilinx/Ex-Xilinx.runs/impl_1/Ex1.bin*** for this example
- Select ***OK***



Program Configuration Memory Device

Select a configuration file and set programming options.

Memory Device: s25fl032p-spi-x1_x2_x4

Configuration file: do/Ex1/Ex1-Xilinx/Ex1-Xilinx.runs/impl_1/Ex1.bin

PRM file:

State of non-config mem I/O pins: Pull-none

Program Operations

Address Range: Configuration File Only

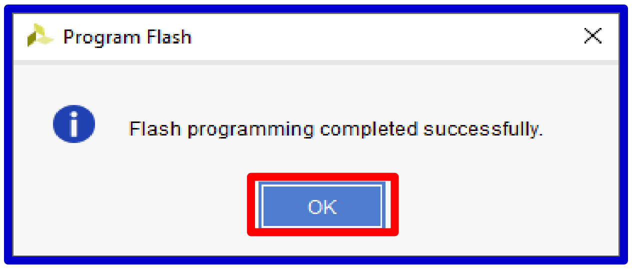
- [✓] Erase
- [ ] Blank Check
- [✓] Program
- [✓] Verify
- [ ] Verify Checksum

SVF Options

- [ ] Create SVF Only (no program operations)

SVF File:

- It may take a few minutes to program the flash memory and then the **<u>Program Flash</u>** window will appear
- Select ***OK***



Program Flash

Flash programming completed successfully.

<u>In order to use the design loaded into flash memory:</u>

- **<u>Power</u>**: Power off the Basys3 board
- **<u>JP1 Jumper</u>**: Move the jumper from JTAG to QSPI
- **<u>Power</u>**: Power back on the BASYS3 board
- **<u>Wait for LED</u>**: After a few seconds the Done LED will turn on indicating that your decoder design has been automatically loaded into the FPGA from the serial flash.
- **<u>Testing</u>**: Test the FPGA (move slide switches SW3 – SW0 corresponding to inputs A-D and view the output F on LED0. Recall that $F(A,B,C,D) = \Sigma(0,1,4,5,7,8,10,14)$ for this example.

Try it again:

- **<u>Press Reset or turn the BASYS3 off and on again</u>**
- **<u>Wait for LED</u>**: After a few seconds the ***Done LED*** will turn on indicating that your decoder design has been automatically loaded into the FPGA from the serial flash.
- **<u>Testing</u>**: Test the FPGA (move slide switches SW3 – SW0 corresponding to inputs A-D and view the output F on LED0. Recall that $\mathbf{F(A,B,C,D) = \Sigma(0,1,4,5,7,8,10,14)}$ for this example.