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

**Xilinx Vivado**

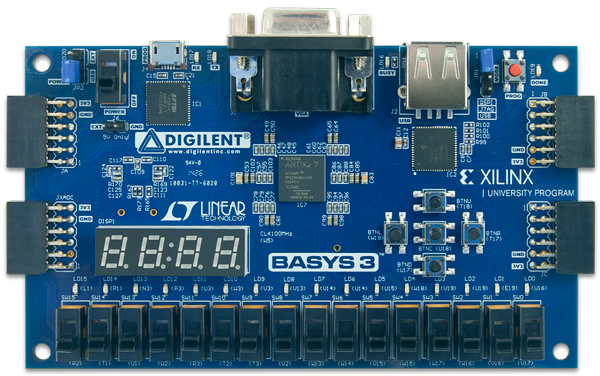
* + - * Specify FPGA Package (Xilinx Artix-7, etc)
* Assign signals to pins
* Implement design
* Generate reports
* Produce bitstream for FPGA (bit file or bin file)
* Program the FPGA

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

**USB cable**

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

**Digilent BASYS3 FPGA Board**



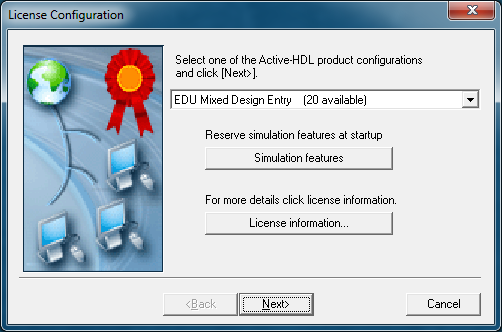
This tutorial will guide you through:

* 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.

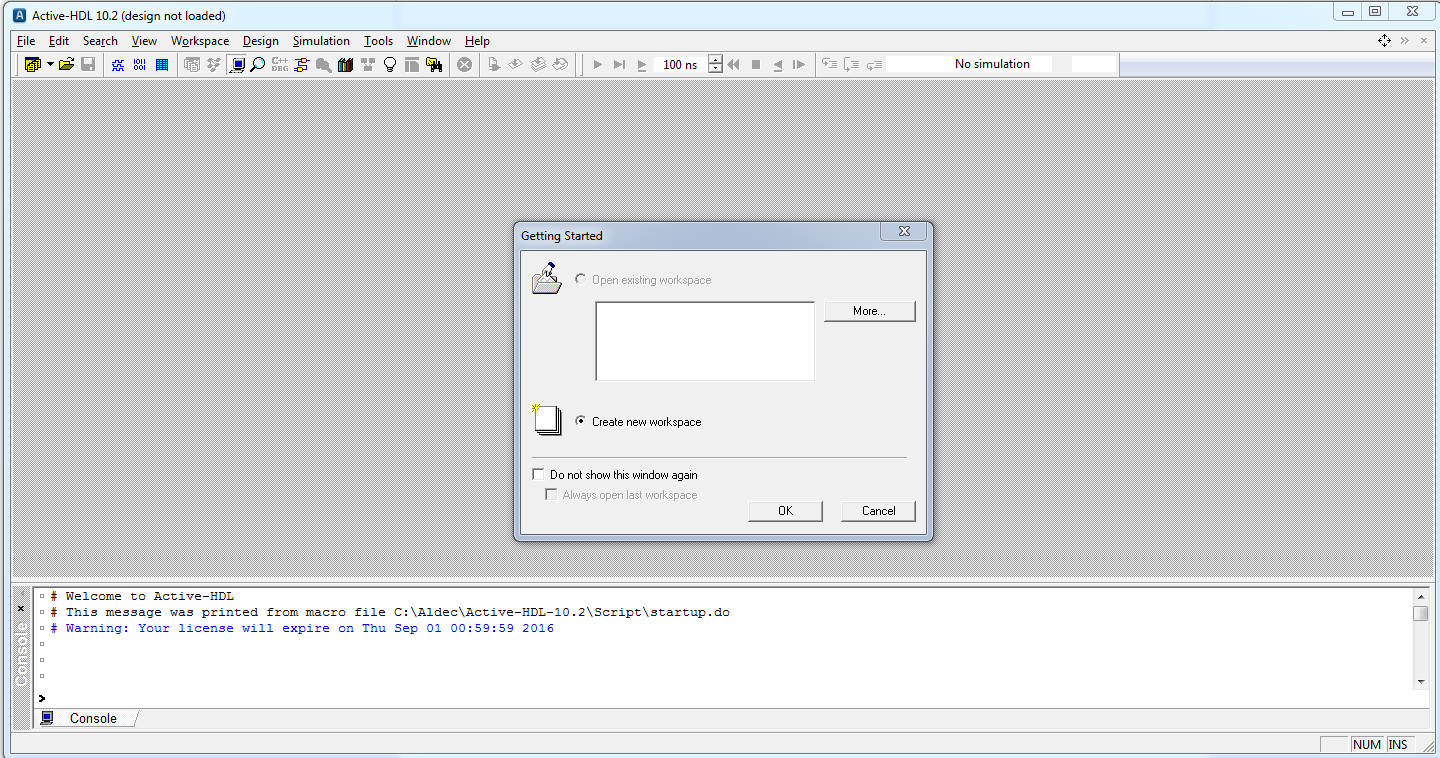
Warning: Aldec and Xilinx can be particular about file names and folder names. Stick to using letters, numbers, and underscores! Avoid filenames such as \Lab 5\Lab#5.aws

**1. Creating a Project with Aldec Active-HDL**

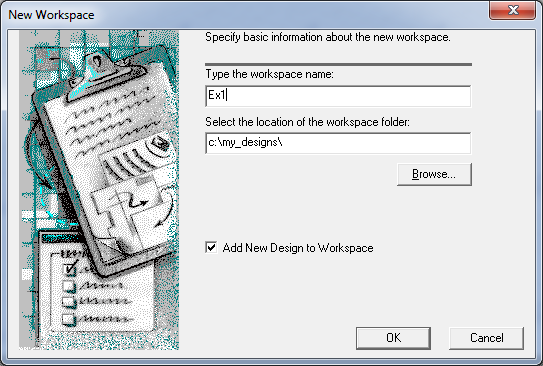
* Launch **Aldec Active-HDL**
* The screen below should appear. Select **Next**.



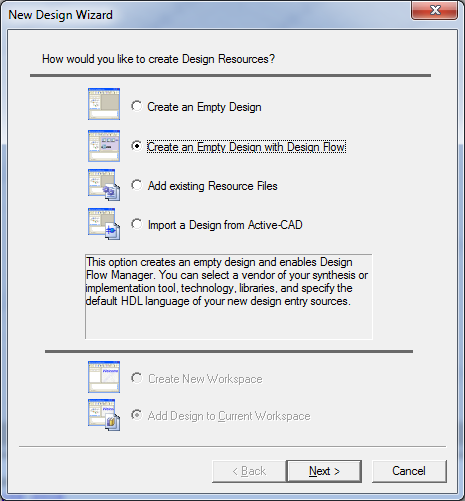
* Select **Create new workspace** as shown below and then select **OK**.



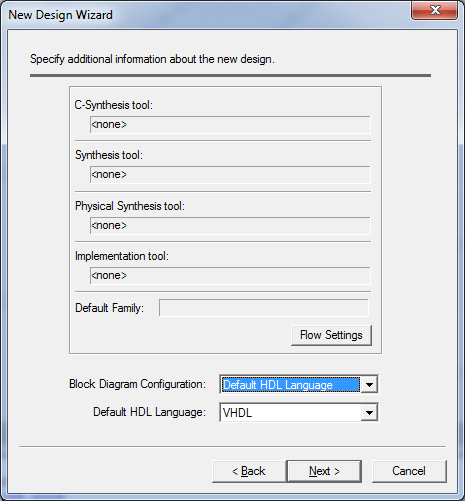
* Enter a name for the workspace (**Ex1** was entered below), change the location of the workspace folder (or use the default as below), and select **OK**. Note that the name for the project workspace, VHDL entity (to be entered later), and architecture (to be entered later) should be the same. ***Use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



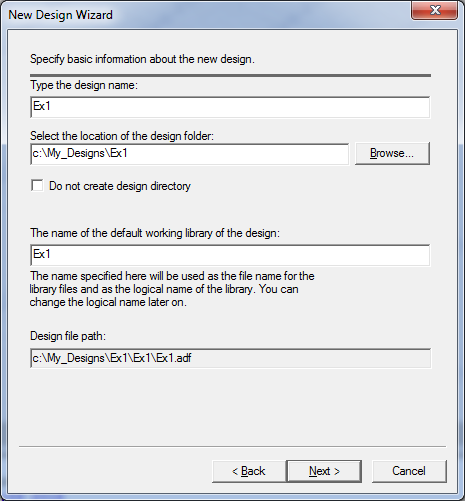
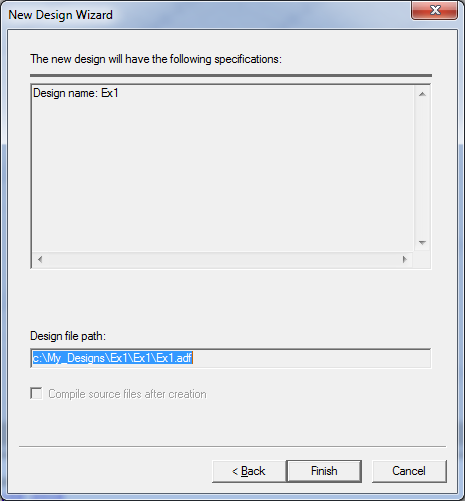
* Select **Create an Empty Design with Design Flow** as shown below and select **Next**.



* 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**.

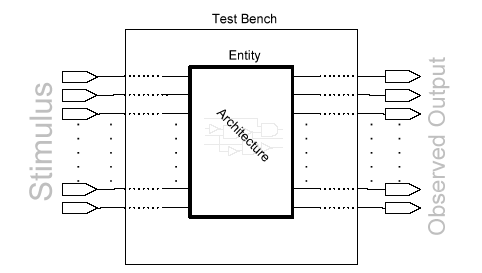


* Enter the design name. Note that it should match the workspace name used earlier (Ex1 for this example). Enter the name and select **Next**. Select **Finish** on the final screen shown below.

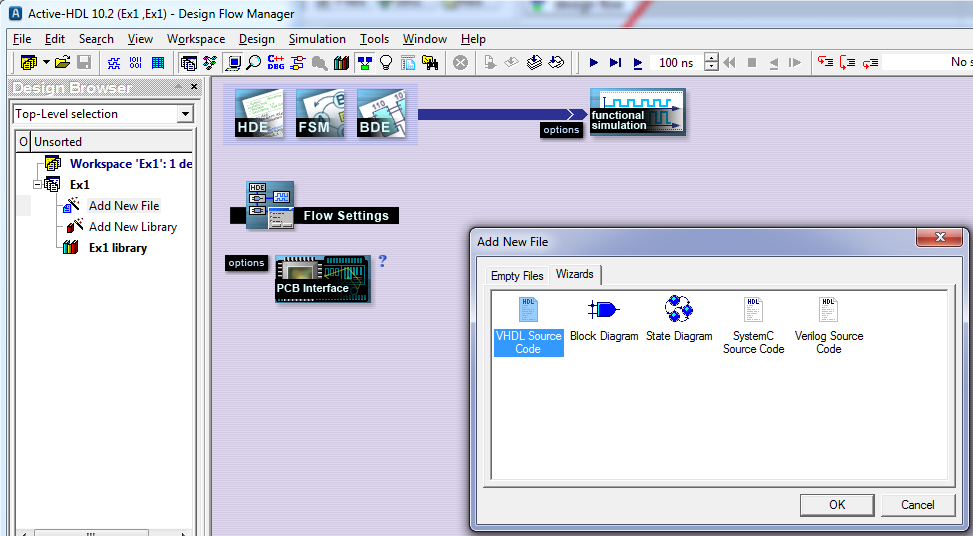
 

**2. Adding VHDL code to your design**

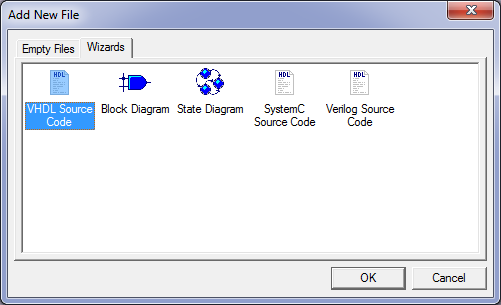
* Before adding VHDL code to the design, recall that VHDL designs typically have three parts:
  + **Entity** – basically defines the inputs and outputs to a black box
  + **Architecture** – defines the function of the black box
  + **Testbench** - defines stimulus signals as inputs to the design and 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.

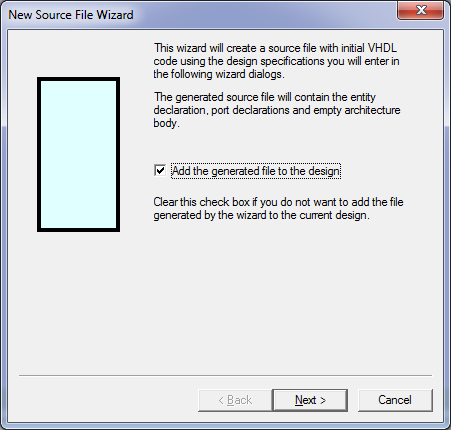
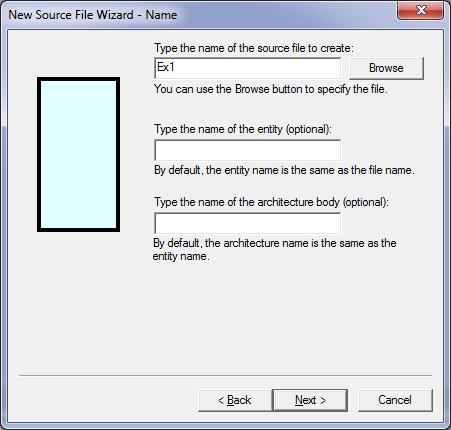


* After selecting the **Wizards** tab, select **VHDL Source Code** and then select **OK**.

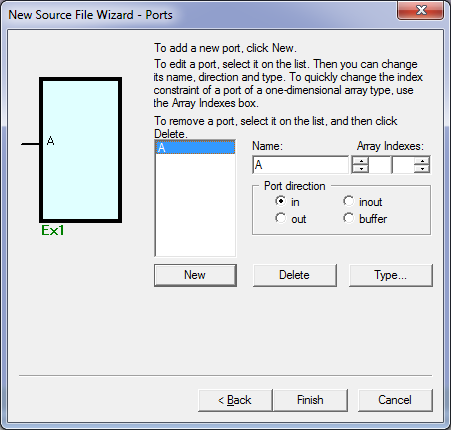


The ***VHDL Source File Wizard*** is used to specify the inputs and outputs that will be used in the entity and architecture sections of the VHDL file to be created (Ex1.vhd). If the function to be implemented is F(A,B,C,D) = Σ(0,1,4,5,7,8,10,14) = A’C’ + A’BD + ACD’ + B’C’D’, then we will need 4 inputs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

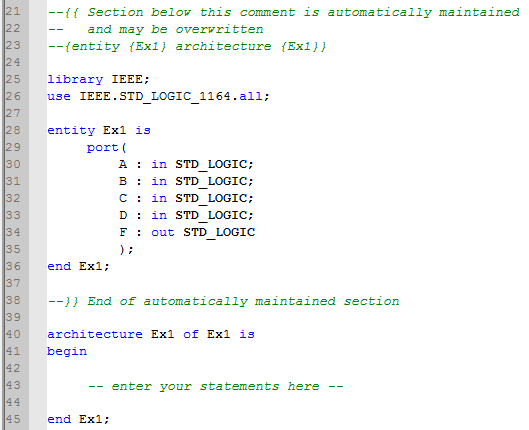
 

* Select **New** in the window below on the left to add a new port. Name it **A** with direction **in**.
* Also add input ports B, C, and D and output port F as shown below and then select **Finish**.

**The diagram should clearly illustrate your inputs and outputs.**

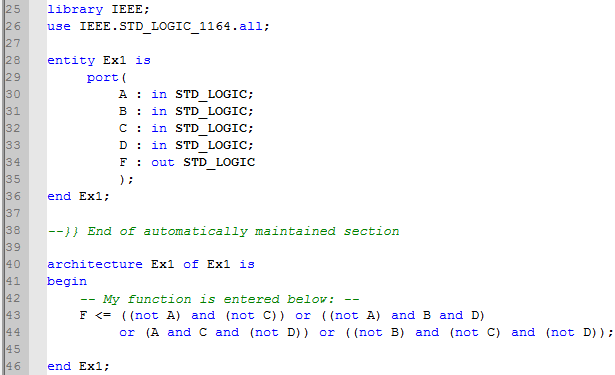
* Note that a VHDL file (Ex1.vhd) has now been generated with a complete entity section and the shell of the architecture section. Only the architecture description (Boolean equation for F in this example) needs to be entered.



**Enter architecture description here**

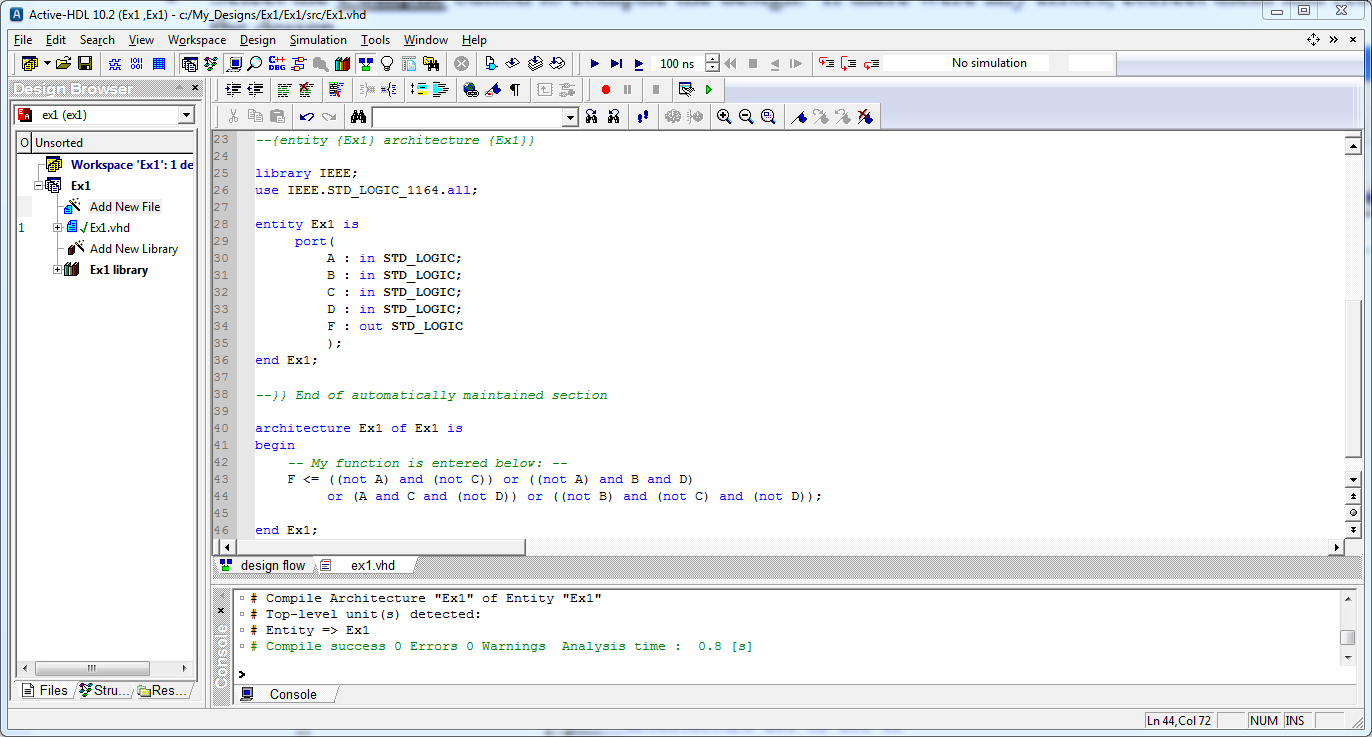
**Note that the entity portion of the VHDL files was completed by the wizard.**

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



**Completed architecture description**

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



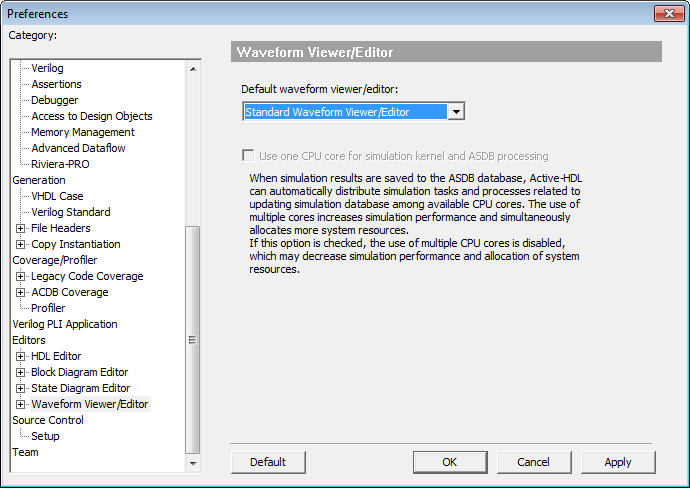
**Successfully compiled**

**Compile button**

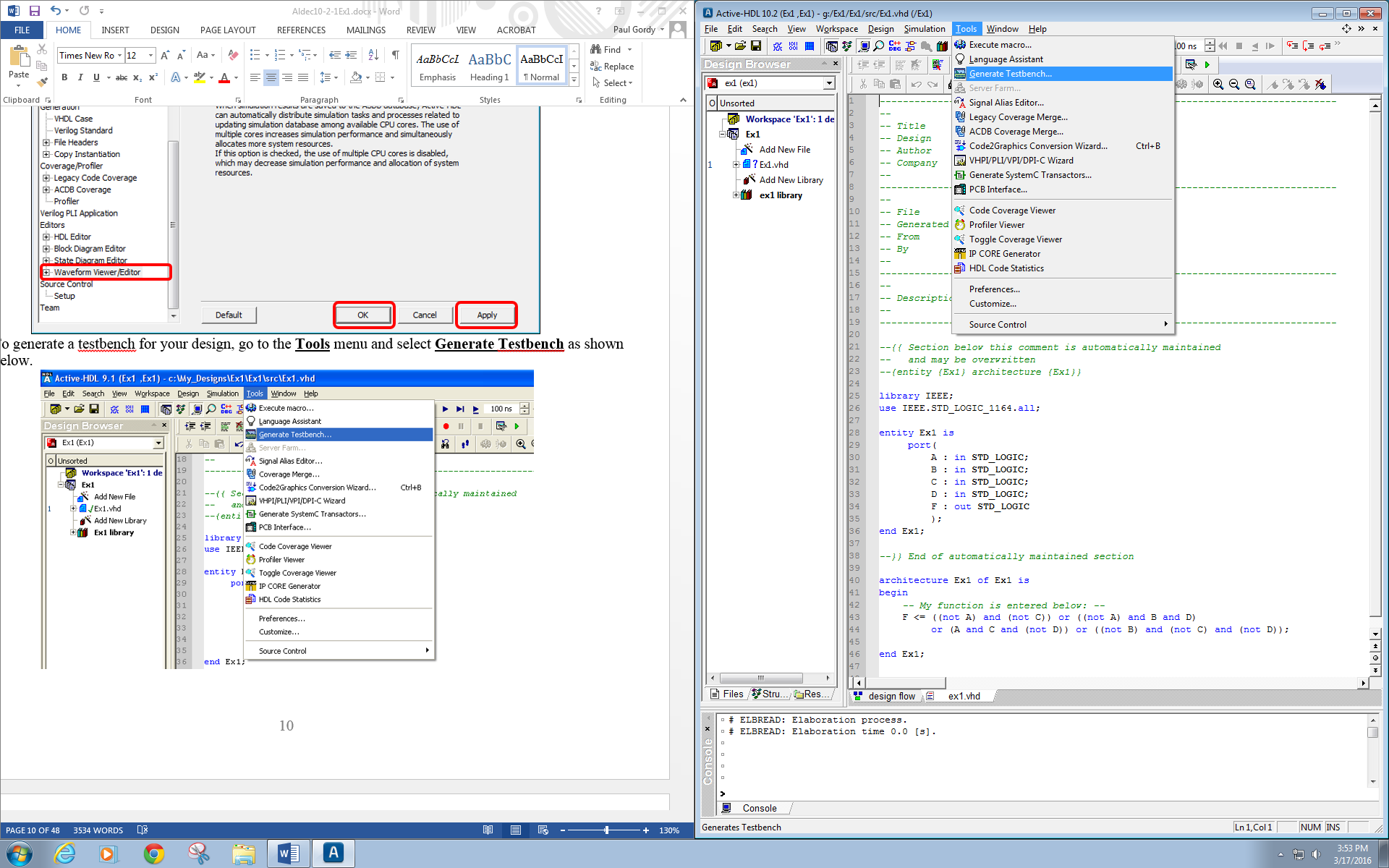
**3. Simulating VHDL Code**

* 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.

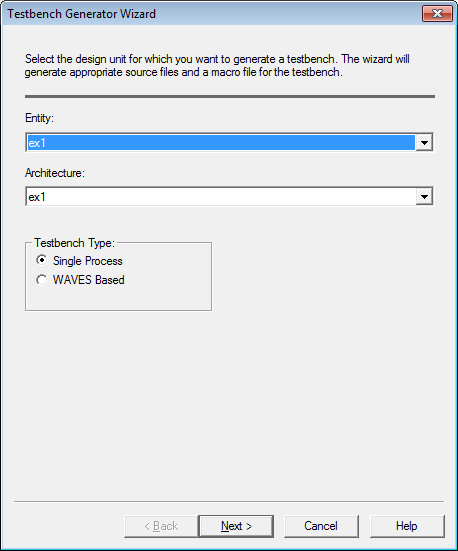
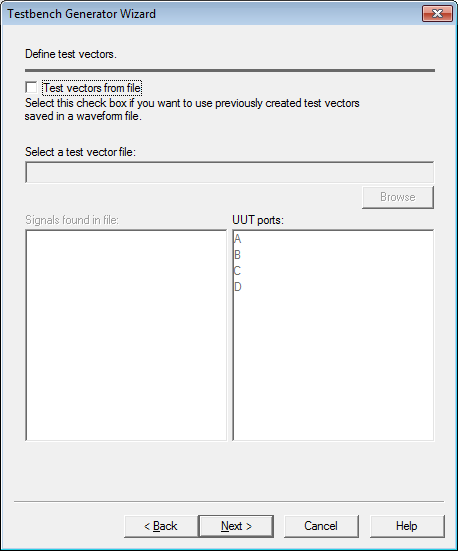
Select **Tools – Preferences** and the Preferences window below should appear as shown below. Select **Waveform Viewer/Editor** and change the default waveform viewer/editor to **Standard Waveform Viewer/Editor** and then select **Apply** and then **OK**.



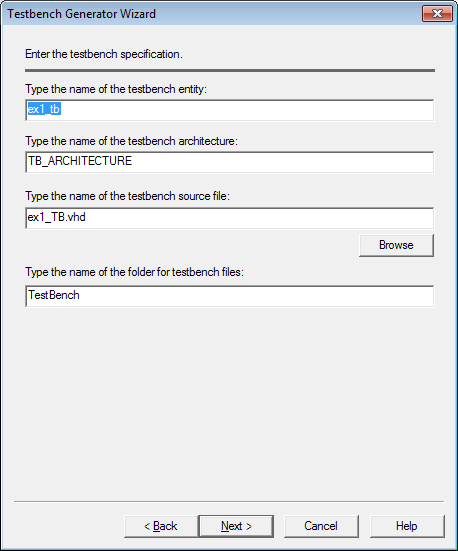
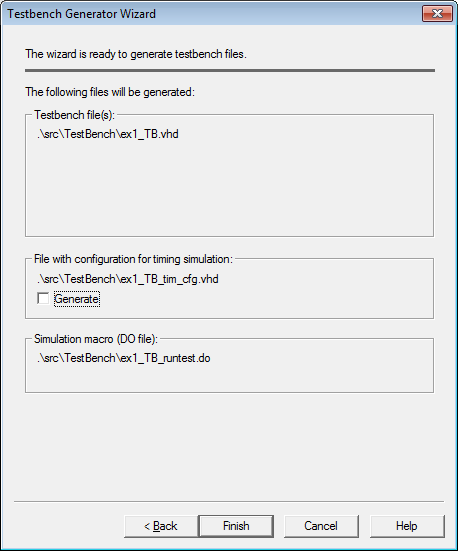
* To generate a testbench for your design, go to the **Tools** menu and select **Generate Testbench** as shown below.



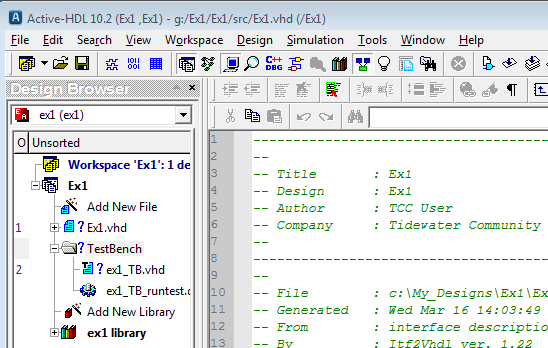
* Select the entity name and the architecture name in the window shown below on the left and then select **Next**. Also select **Next** in the window shown below on the right.

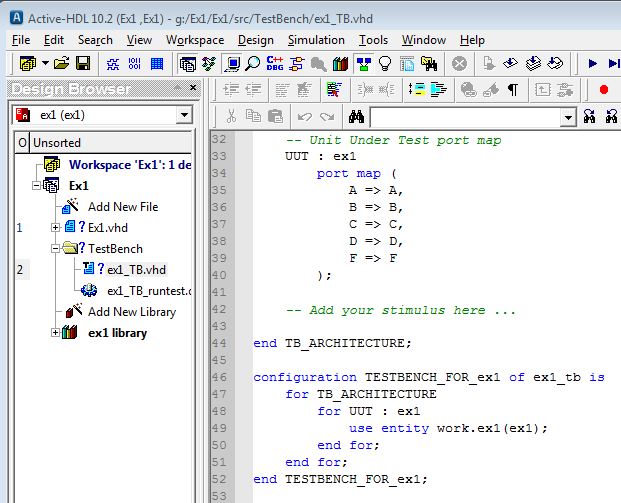
* Select **Next** in the window below on the left and **Finish** in the window below on the right.

* 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.



* 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*** ...”



**Scroll down to here.**

* 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.

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

0

A

B

C

D

* 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.

0

ns

160

ns

A

B

C

D

150

ns

140

ns

130

ns

120

ns

110

ns

100 ns

90

ns

80

ns

70

ns

60

ns

50

ns

40

ns

30

ns

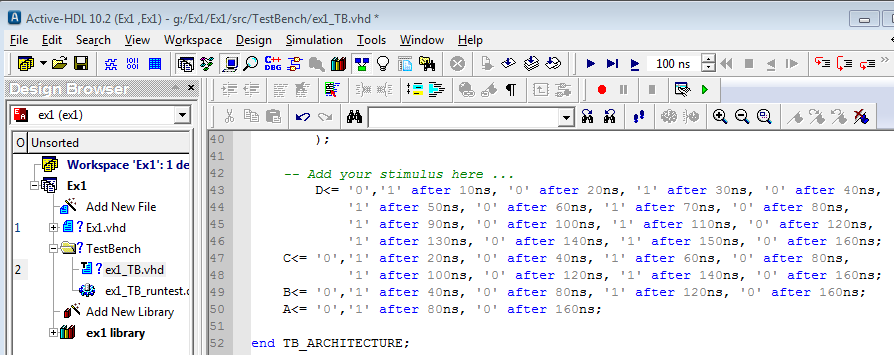
20

ns

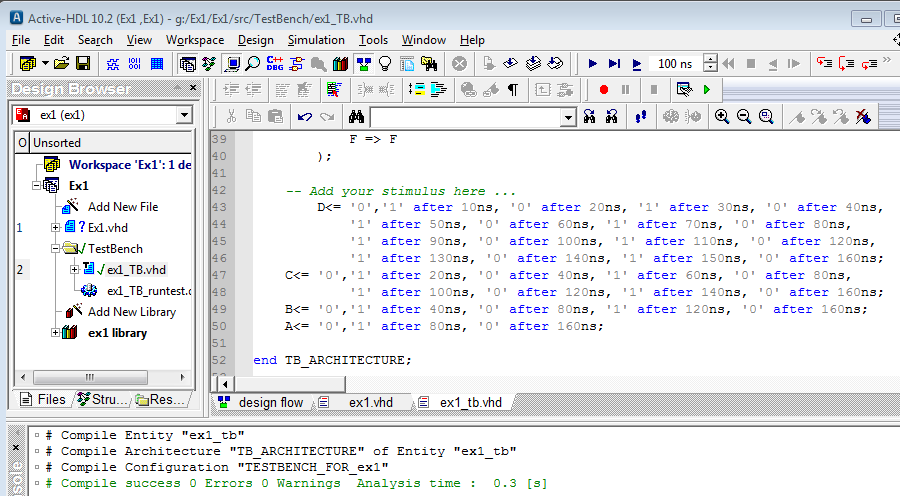
10

ns

* 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*** ...”



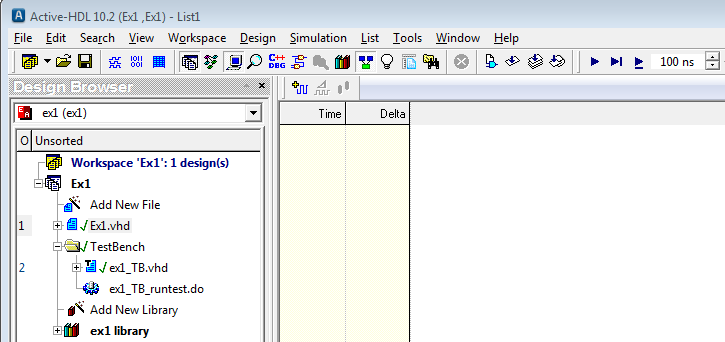
* **Compile** your design again. Correct any errors that occur in the VHDL code.



**Compile**

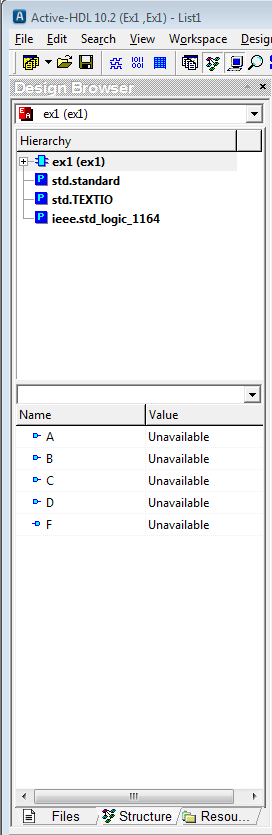
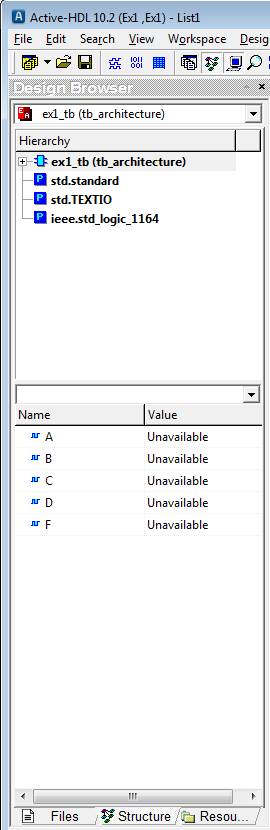
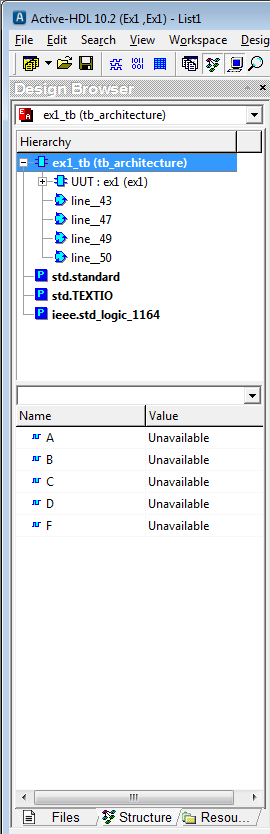
**Successfully compiled**

* 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.



**New List**

* 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.

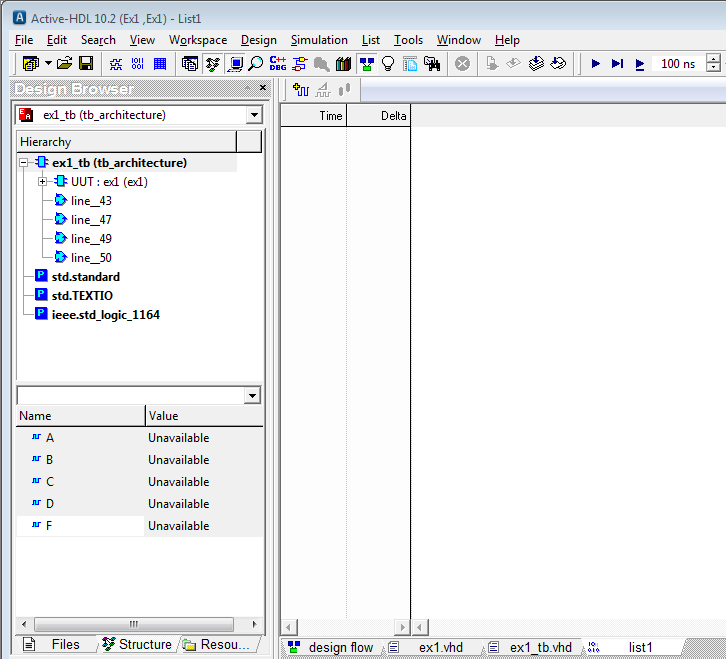
  

**Expand**

**Select arrow and change to ex1\_tb(tb\_architecture)tructure Tab**

**Structure Tab**

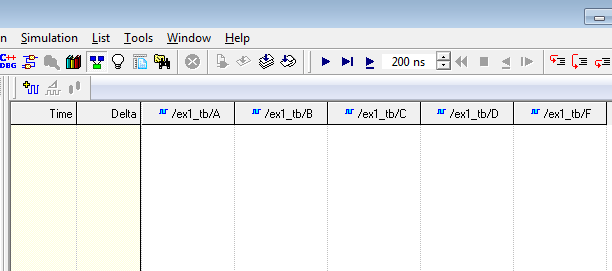
* Click on any signal in the browser (A, B, C, D, or F) and press Ctrl + A to select all of the signals. Next drag the signals to any location in the right window (List File).



**The name of this list file (truth table) is list1.**

**Select all signals and drag to 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**.

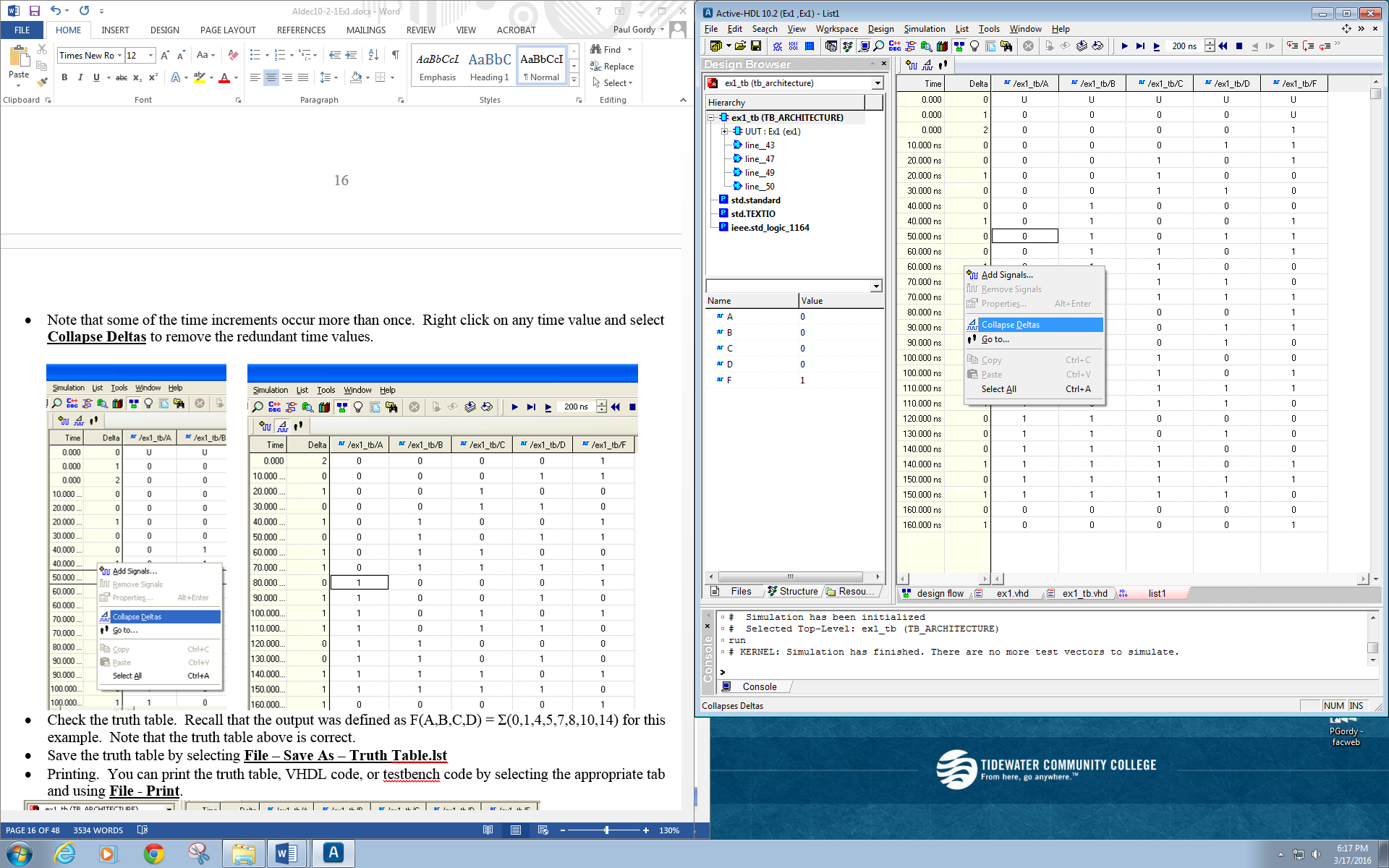
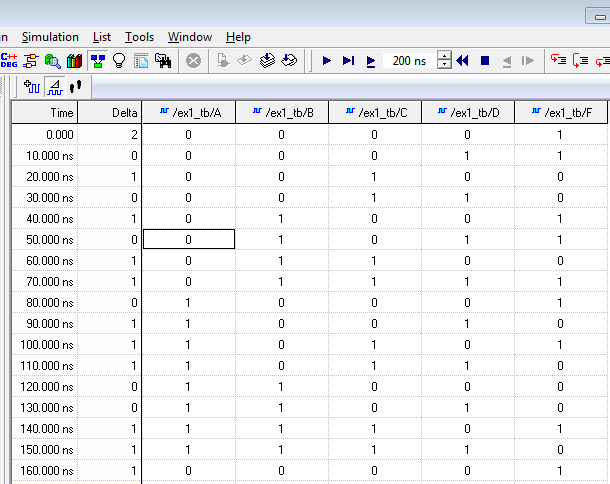


**Rearrange signals if necessary**

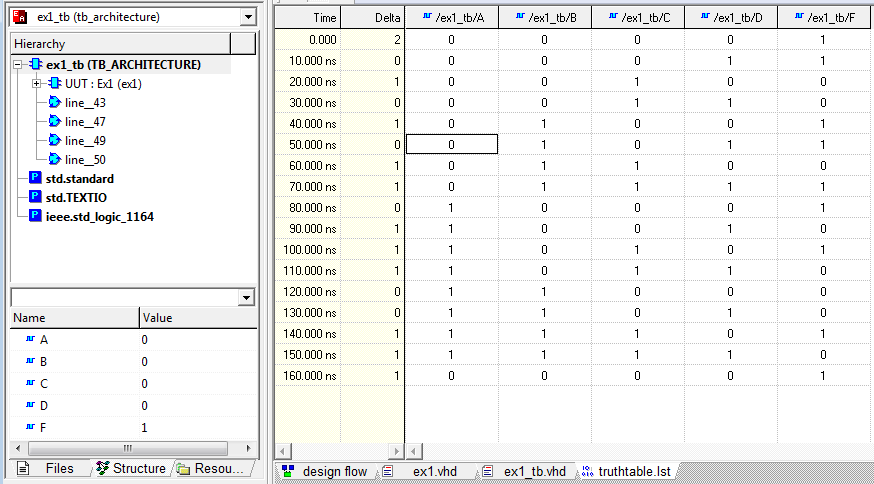
**RUN**

**Change time to 200 ns**

* 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.

* 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**.



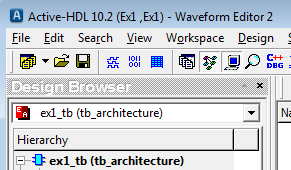
**Select the desired file to view or print.**

**Generating Waveforms**

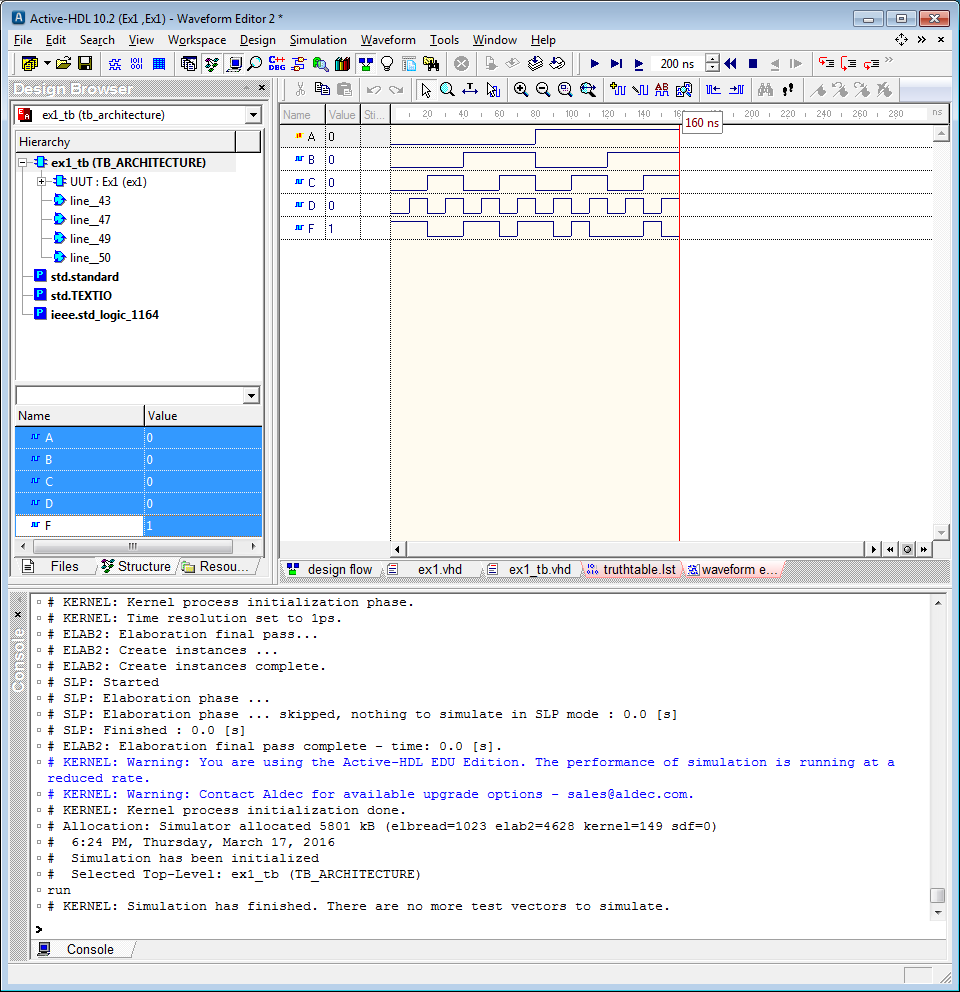
The results of the simulation can also be displayed using waveforms.

If you have already run a simulation to generate a truth table (list), select **Simulation – End Simulation** from the main menu.

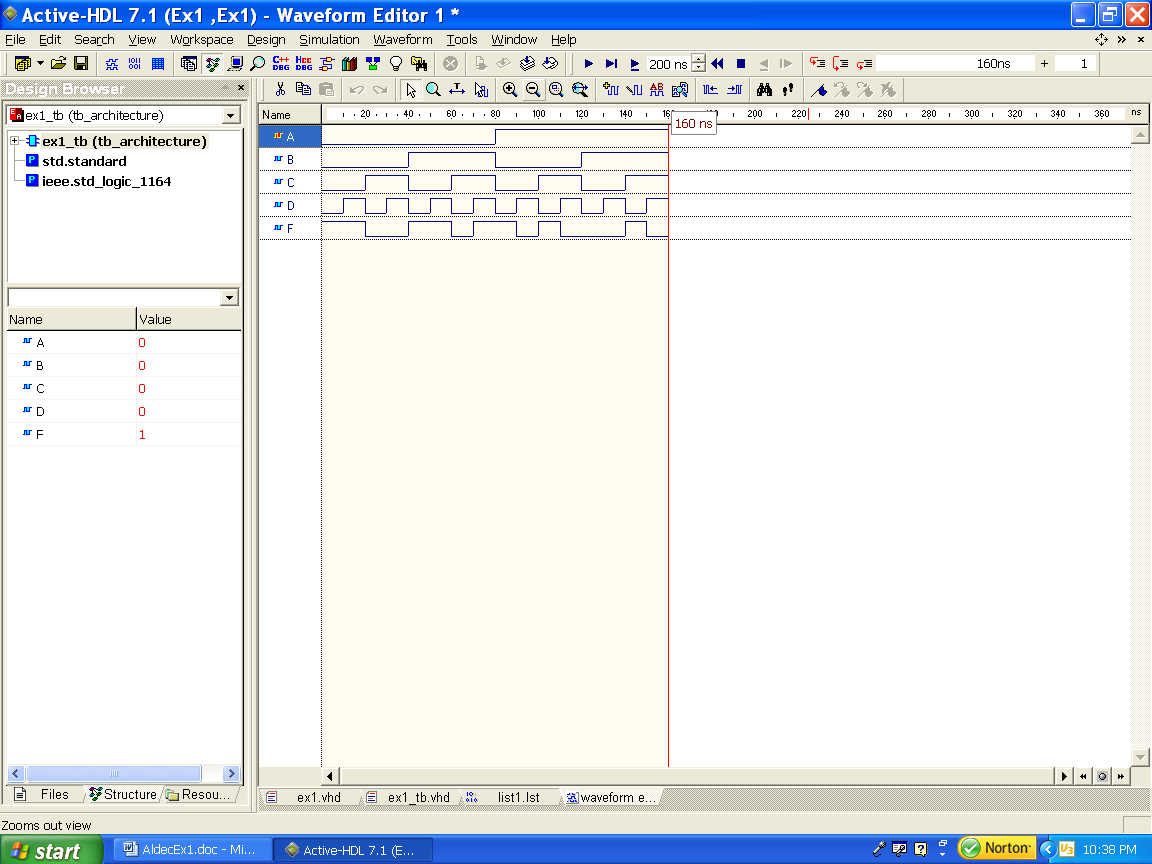
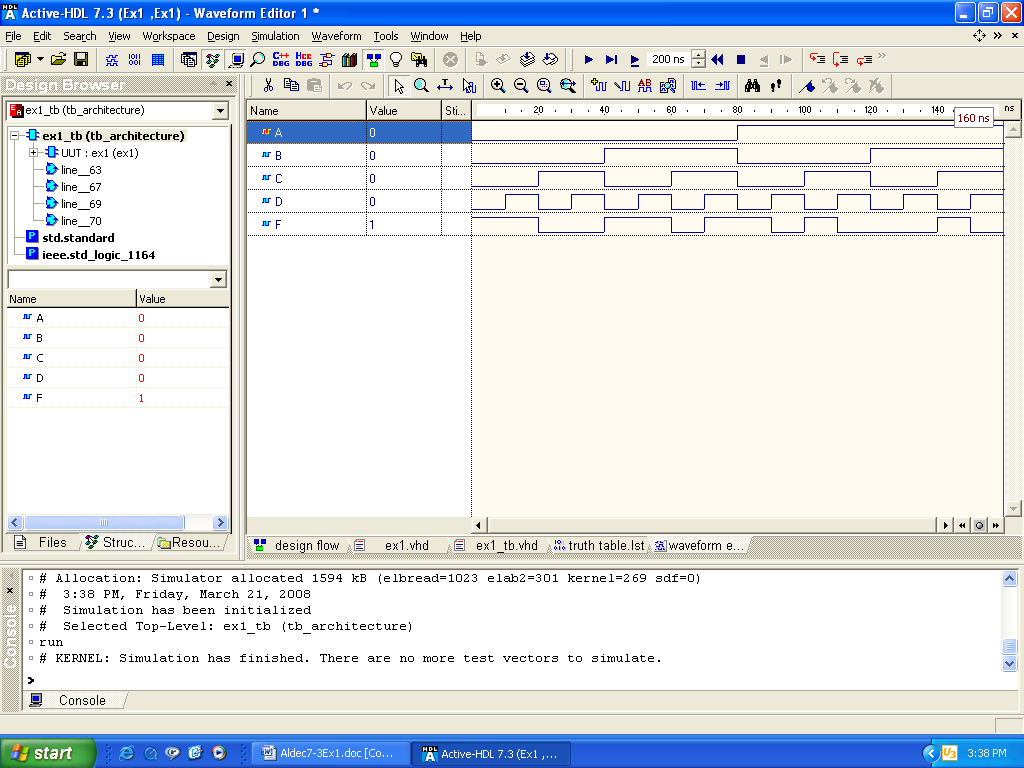
* Select the **New Waveform** button as shown below.

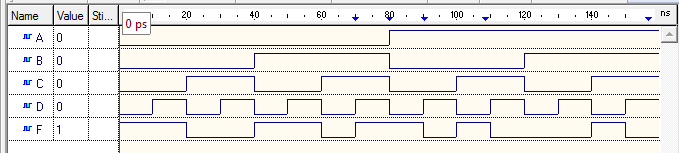


* Drag the available signals (inputs and outputs) into the empty window on the right.



**Drag each waveform to the window on the right**

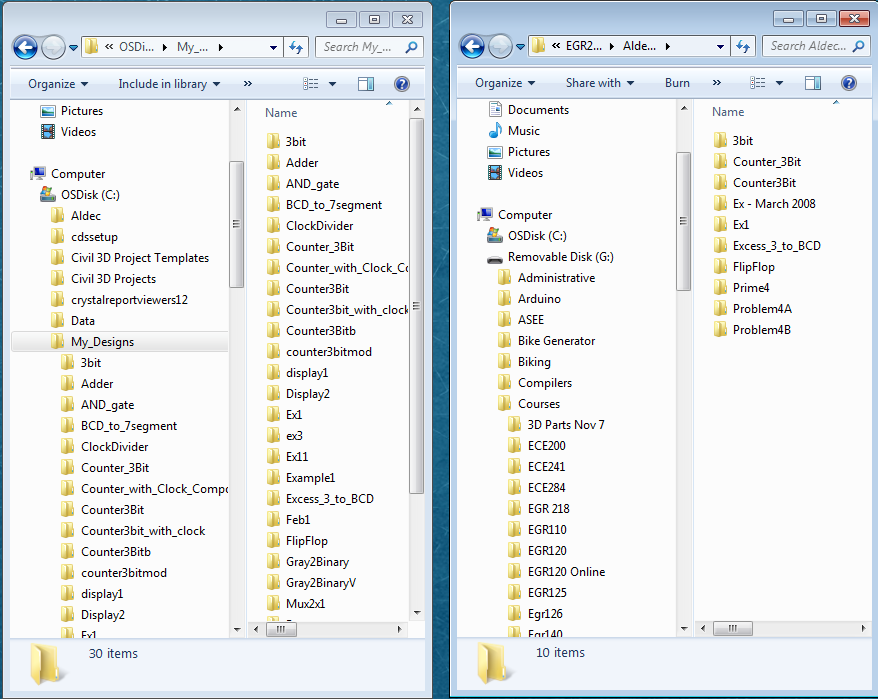
* If necessary, rearrange the waveforms (by dragging them) into the desired order.
* Select the **RUN** button to run the simulation. Use the symbols (Zoom In, Zoom Out, Zoom To Fit) to adjust the size of the waveform.



* 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**

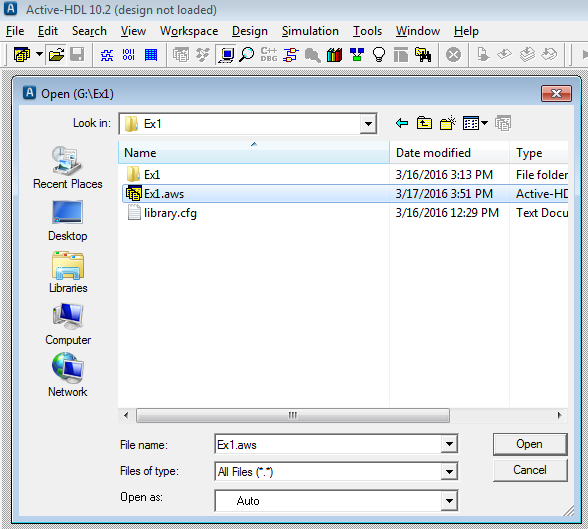
* Aldec Active-HDL will save your files by default in the C:\My\_Designs folder.
* It is recommended that you leave this as the default setting.
* When you have finished your work in lab, copy the folder for your design onto a personal memory storage device as illustrated below.
* If you return to lab 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.**

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

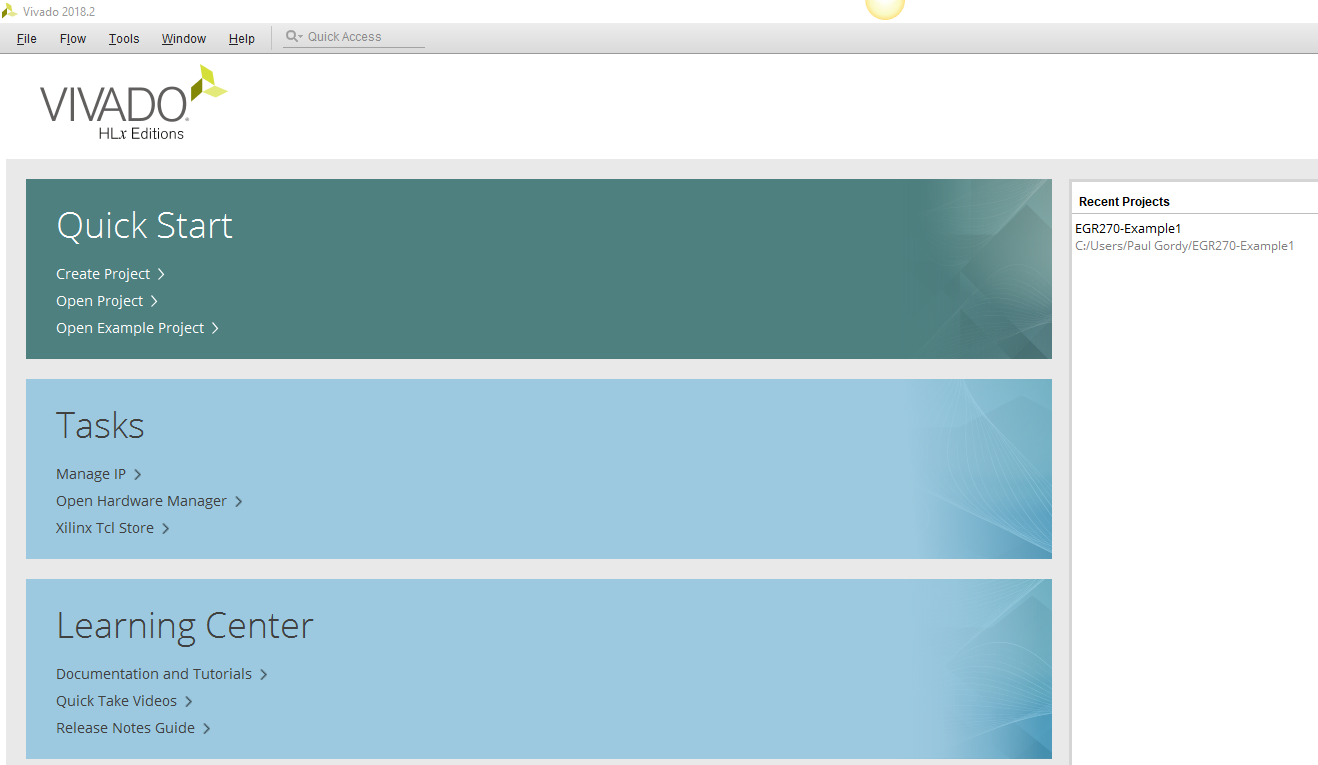
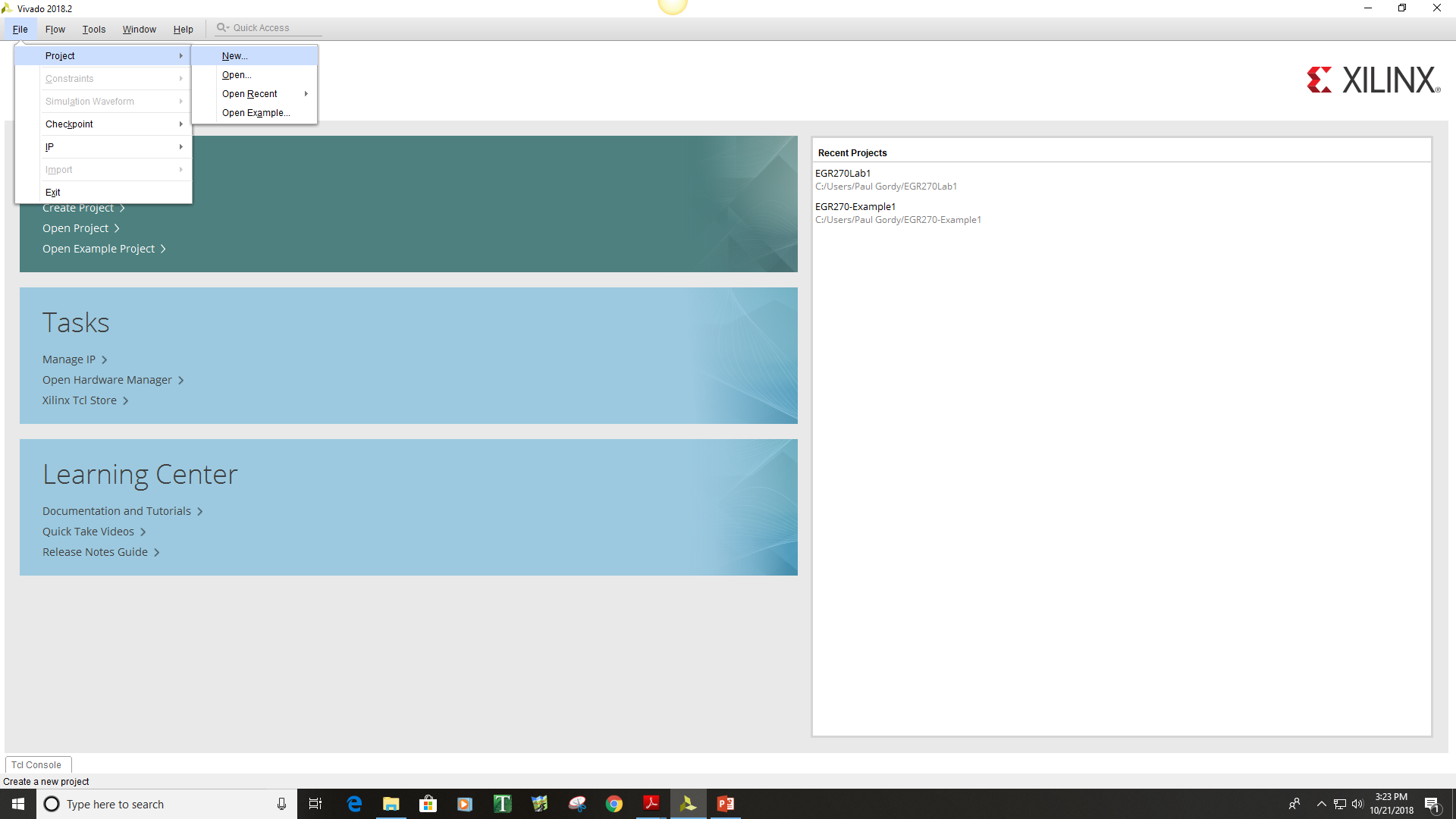
* Launch **Aldec Active-HDL**.
* Cancel the Getting Started window.
* Select **File – Open** and locate your workspace file (**aws** extension) as shown below.



**Xilinx Vivado**

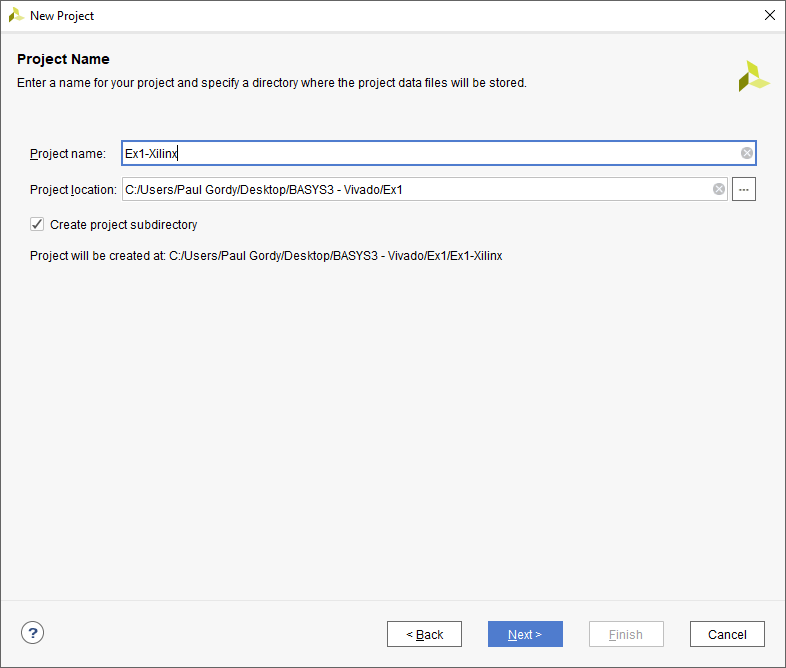
Xilinx Vivado will be used to implement a VHDL design into an FPGA. The design just created using Aldec Active-HDL is specified in the file **Ex1.vhd**.

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)



1. **Project Name**

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

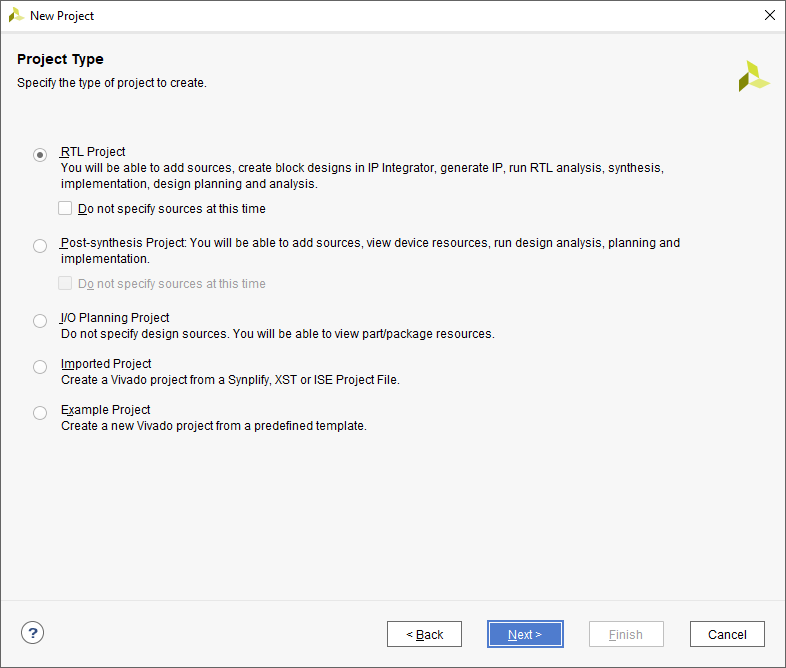


Recommendations:

* 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).

1. **Project Type**

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

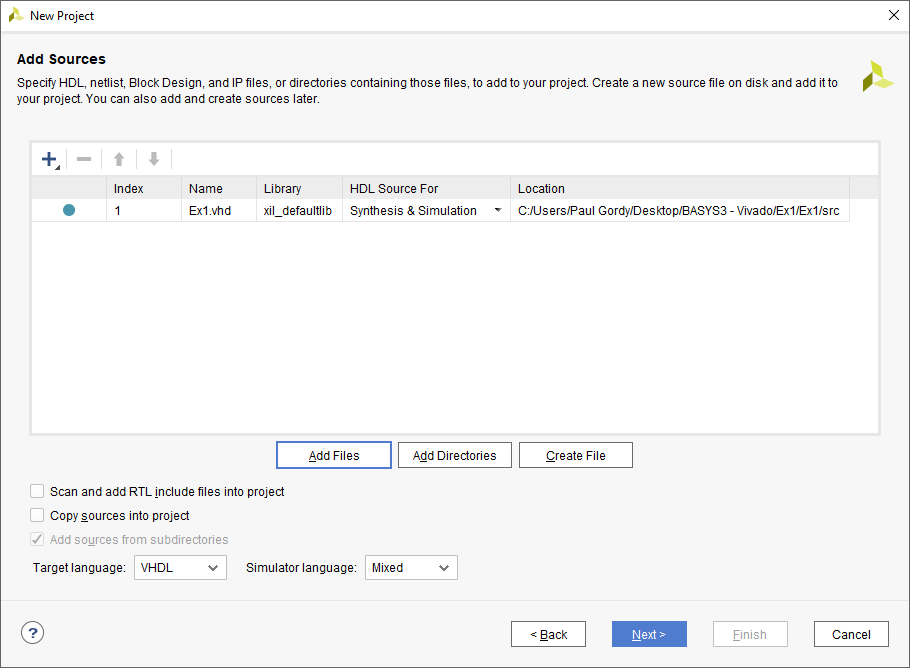


**Note**: ***RTL*** stands for ***Register Transfer Level*** or ***Register Transfer Logic***) and refers to describing a design in terms of signals transferred between registers (flip-flops) using HDL. It can include signal descriptions for both synchronous and combinational circuits.

An RTL description may be converted into a gate-level design during the implementation phase.

1. **Add Sources**

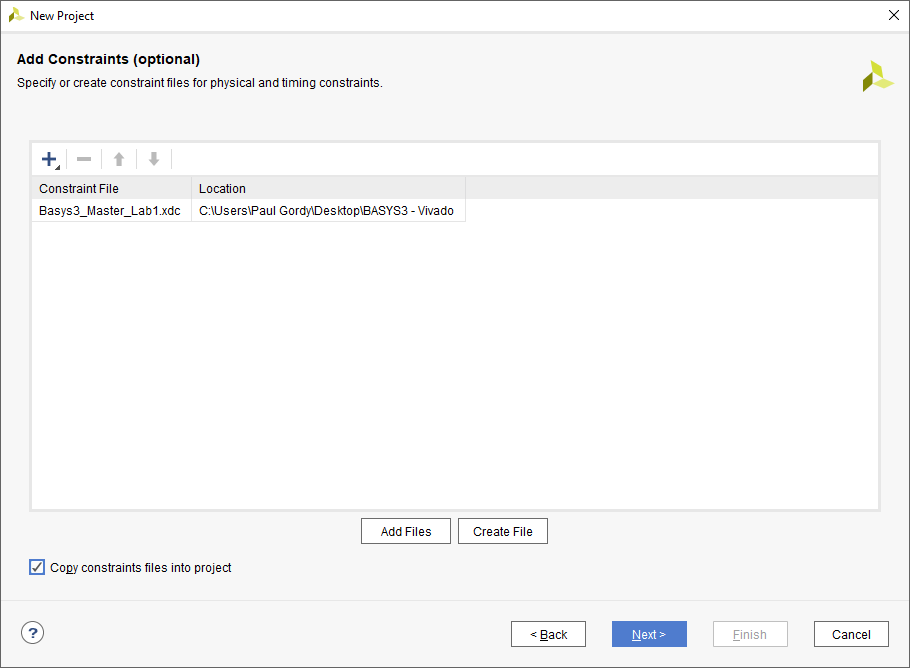
* **Add Files** – Select Add Files and then browse to find the vhd file created using Aldec Active HDL. Ex1.vhd was added in this example.
* **Target language**: VHDL
* Select **Next**



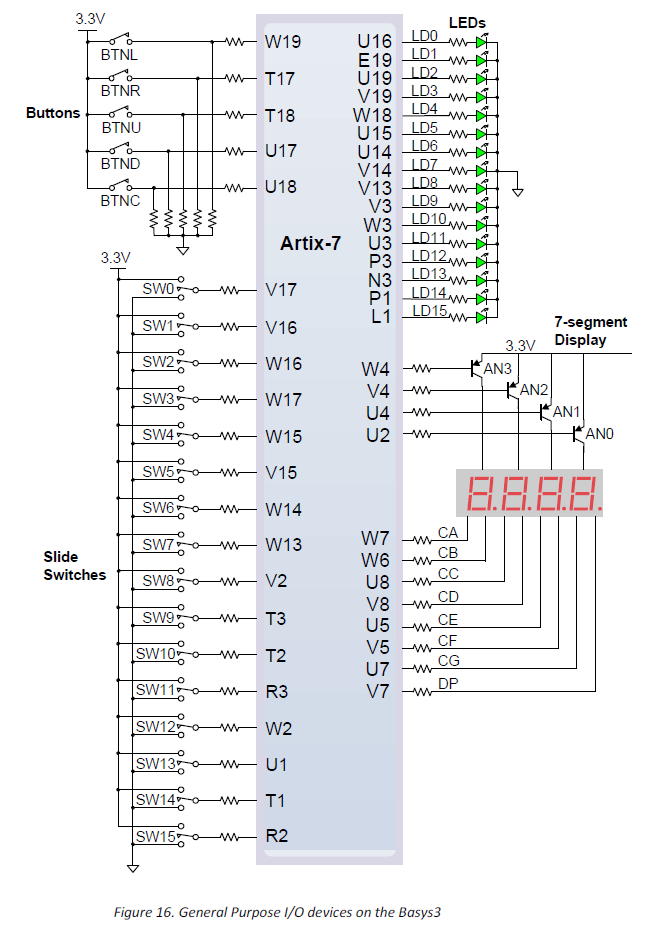
1. **Add Constraints**

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.

* **Add Files** – 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.
* **Check the box** – Copy constraints files into project
* Select **Next**



**Creating a Constraint File**



**BASYS3 Pinout**

100MHz clock on pin W5

* 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’**

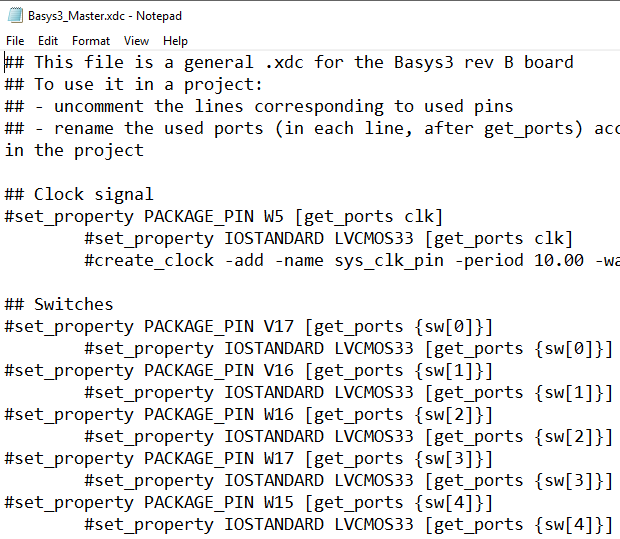
so we need 4 inputs switches (A,B,C,D) and one output LED (F).

* 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.

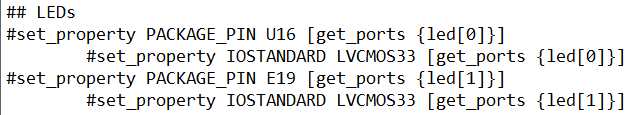
|  |  |  |
| --- | --- | --- |
| **Input/**  **Output** | **BASYS3**  **Name** | **BASYS3**  **Pin** |
| **A** | **SW3** | **W17** |
| **B** | **SW2** | **W16** |
| **C** | **SW1** | **V16** |
| **D** | **SW0** | **V17** |
| **F** | **LED0** | **U16** |

**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).



**Original Constraint File – Available on course website**

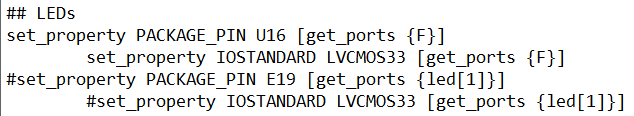
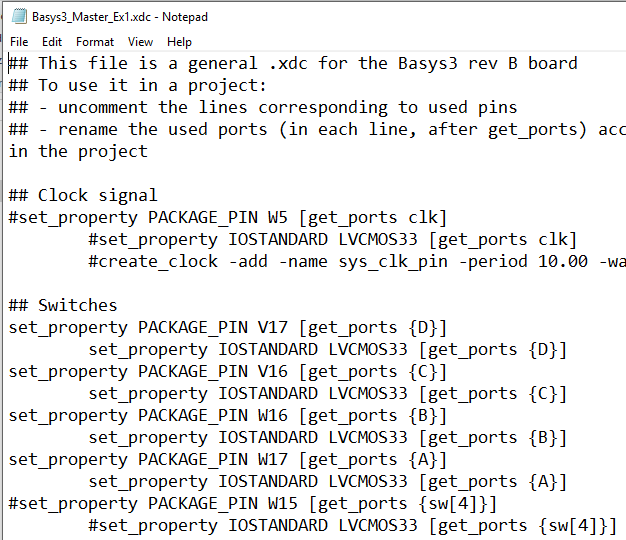


***Uncomment***

***Rename***

***Uncomment***

***Rename***



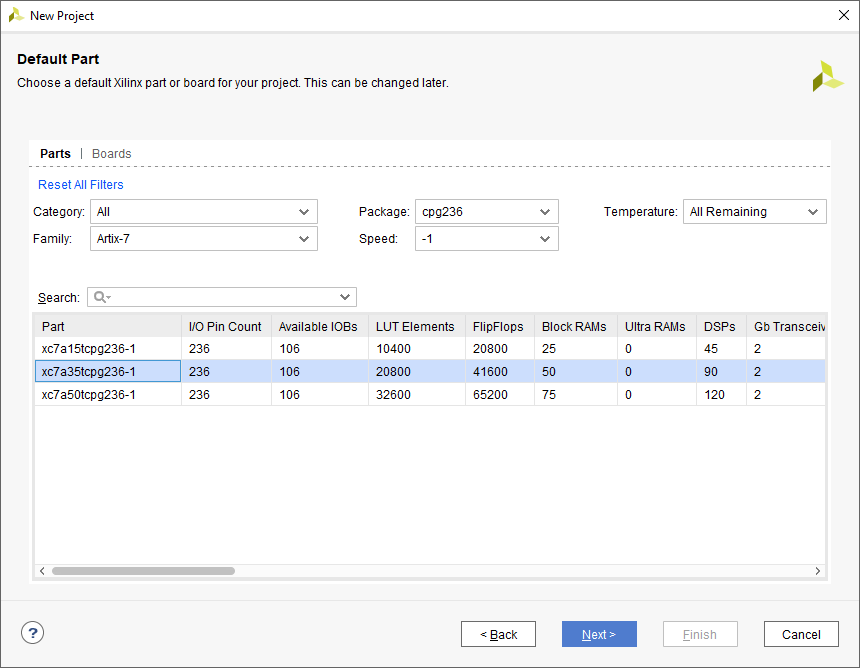
**Modified Constraint File**

**Note that signal A is assigned to pin W17.**

**See table on previous slide.**

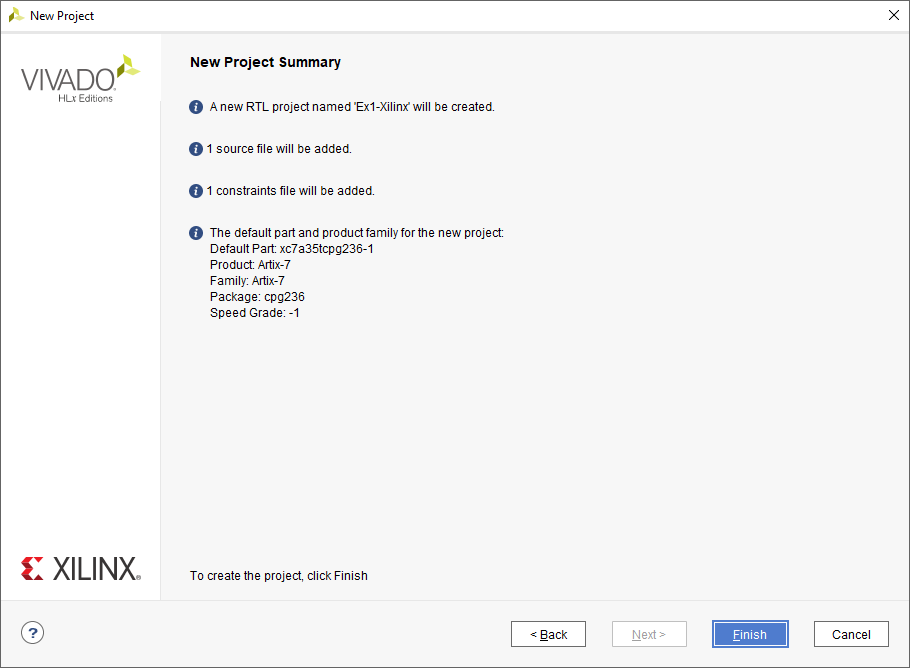
1. **Default Part**

* 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***

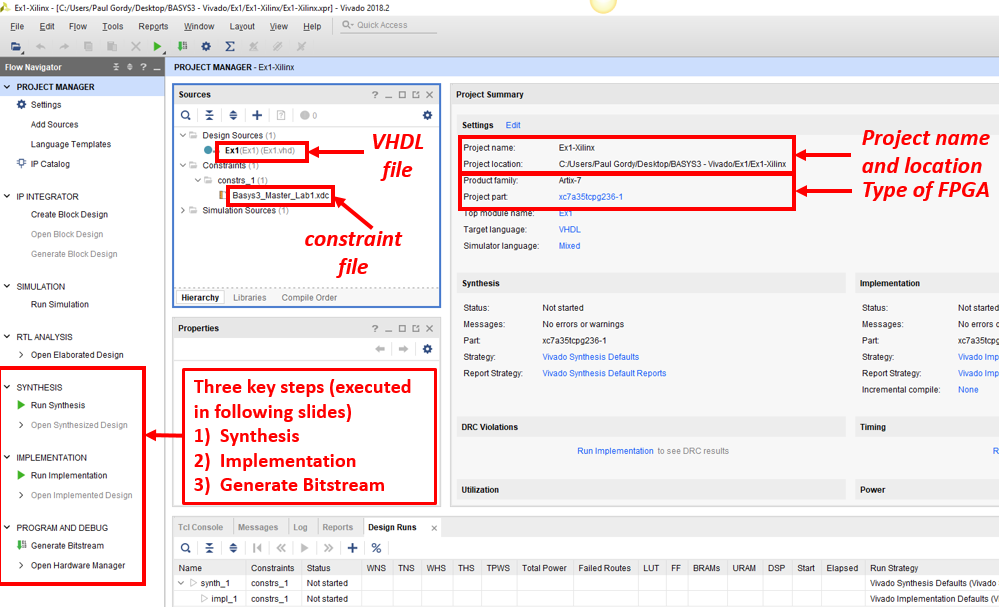


1. **New Project Summary**

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



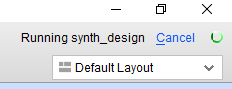
**The main screen in Vivado now appears** – a few key features are highlighted



**1) Synthesis**

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

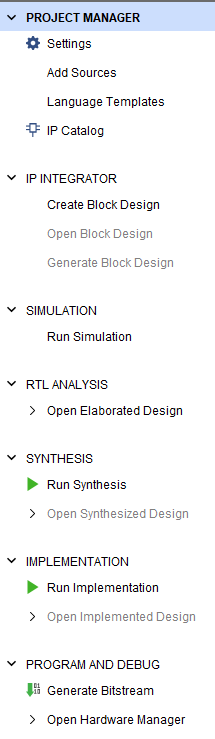
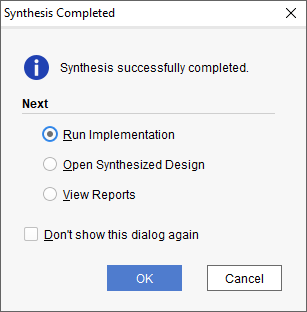
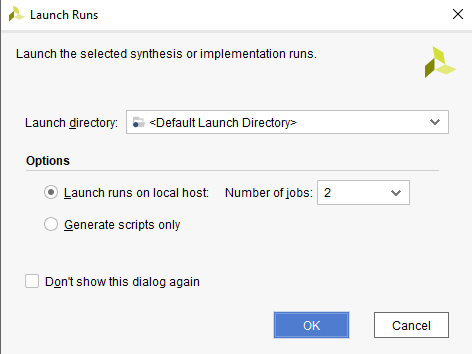
(This may take a few minutes. Is it still running? Check the upper right corner of the main screen.)



***Still running!***



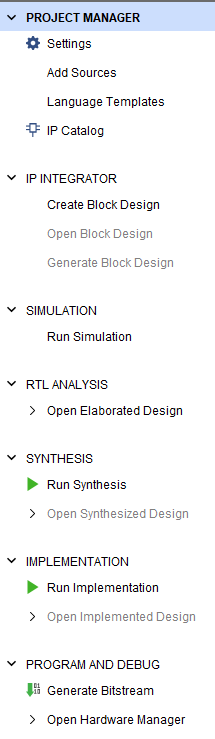
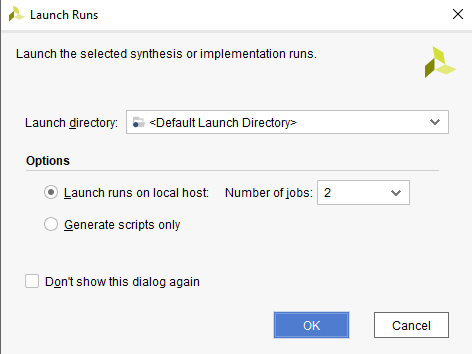
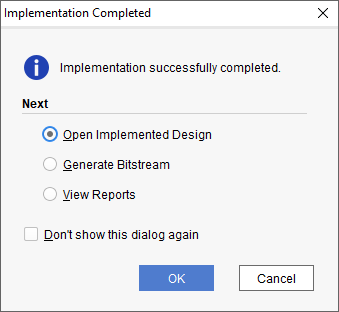
***Complete!***



**2) Implementation**

* (This may run automatically after synthesis)
* Under **Implementation**, select ***Run Implementation***
* **Launch Runs** - select ***OK***
* **Implementation 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.)



**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):

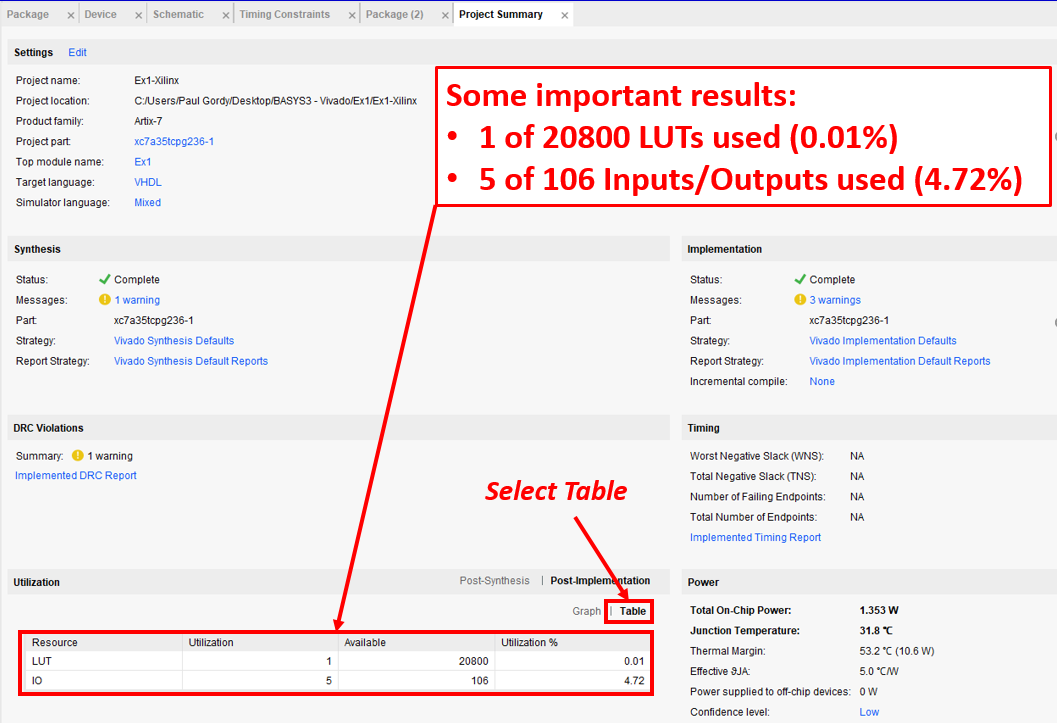
* ***Project Summary*** – Shows project and FPGA information as well as utilization (number of LUTs and number of I/Os used)
* ***Schematic (for implemented design)*** – Shows used LUTs, input buffers, output buffers, etc.
* ***Package View*** – Shows the bottom view of the FPGA showing which of the 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).

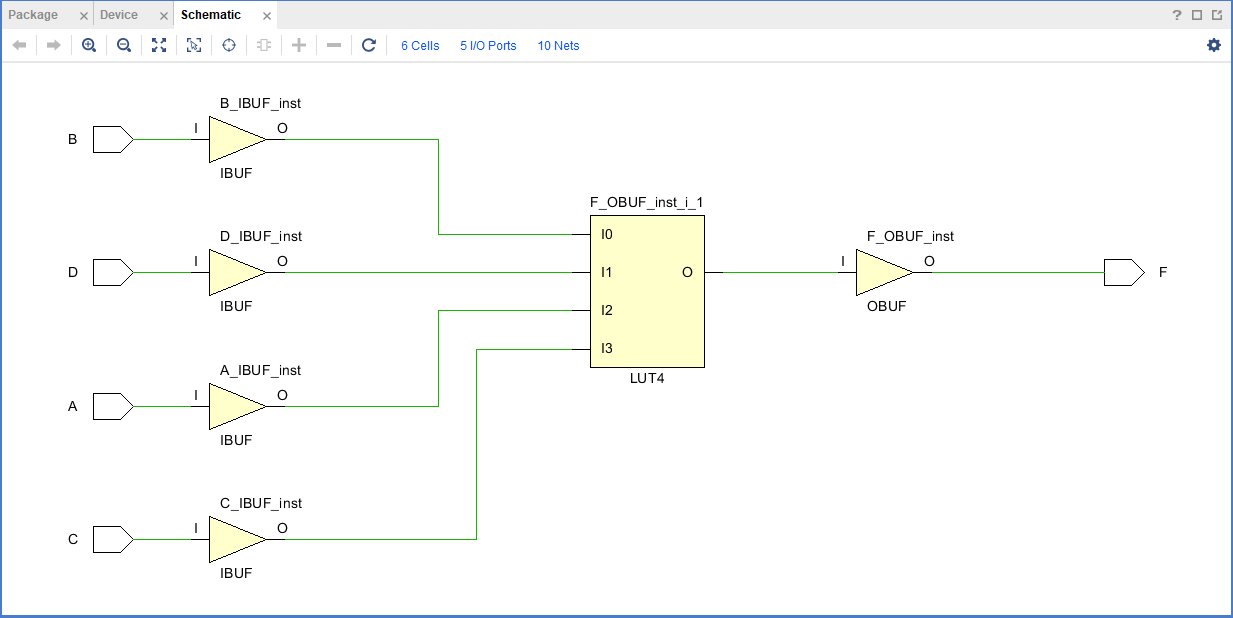
**Project Summary**

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



**Schematic (from implementation)**

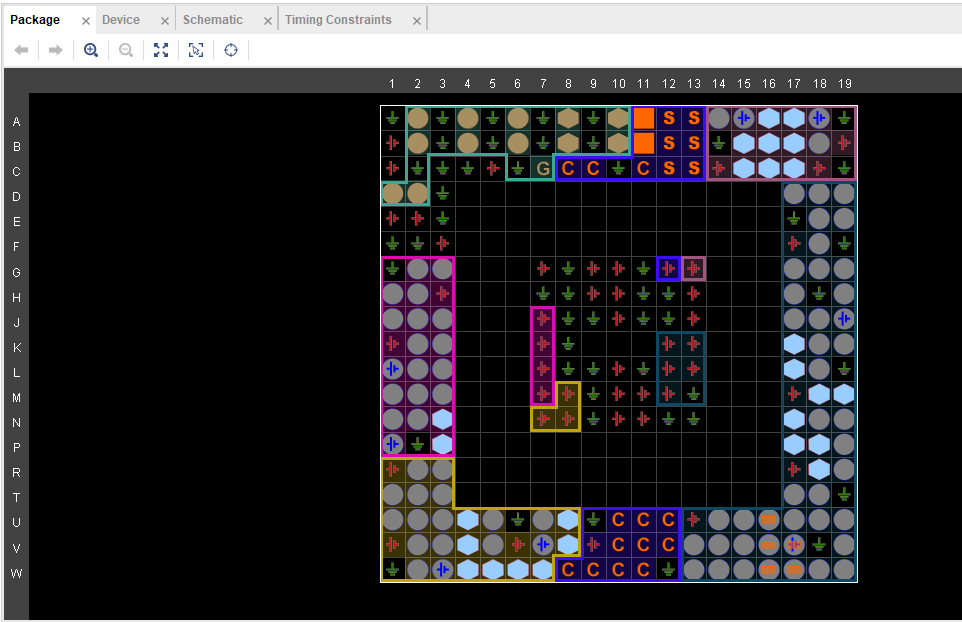
* Shows used LUTs, input buffers, output buffers, etc. LUT4 means that the LUT has 4 inputs.
* Under **Open Implemented Design** – select ***Schematic***

****

**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).

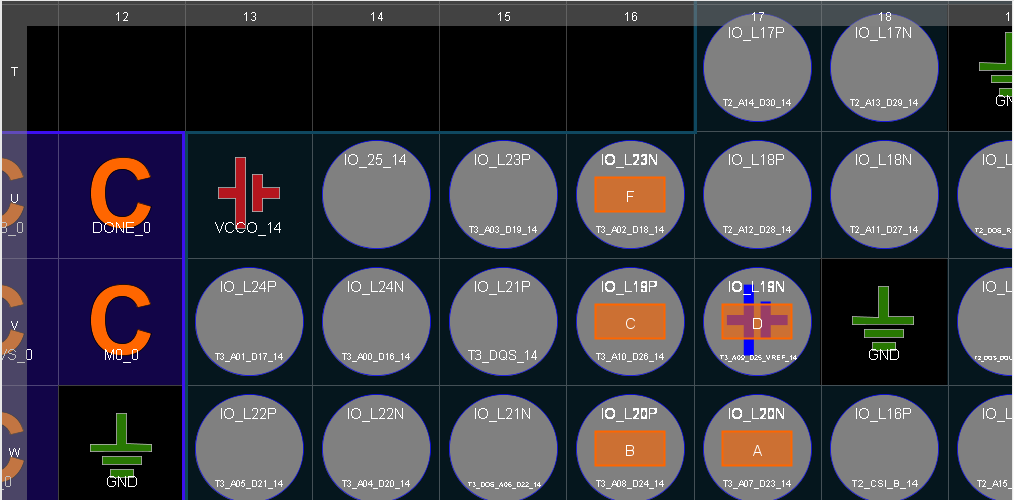
|  |  |
| --- | --- |
| **Input/**  **Output** | **BASYS3 Pin** |
| **A** | **W17** |
| **B** | **W16** |
| **C** | **V16** |
| **D** | **V17** |
| **F** | **U16** |



**Zoom in on the area with the assigned pins. See table.**

**Package View – (continued)**

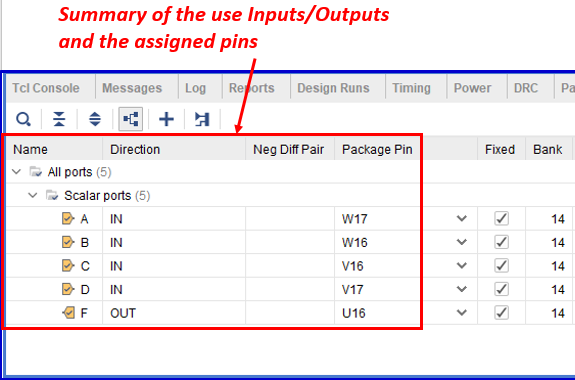
* Note that you can see the signal names A,B,C,D,F after zooming in.



**Output F is on pin U16**

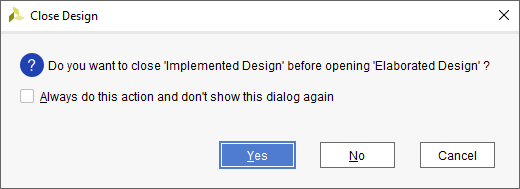
**I/O Ports**

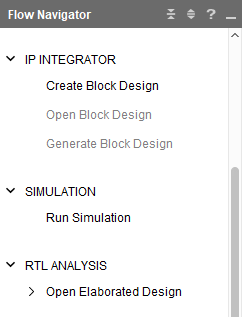
* Shows the used inputs/outputs and the assigned pins
* Select **Window – I/O Ports**
* Expand **scalar ports** to see the signals used

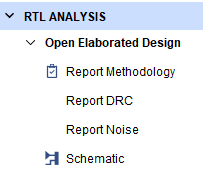


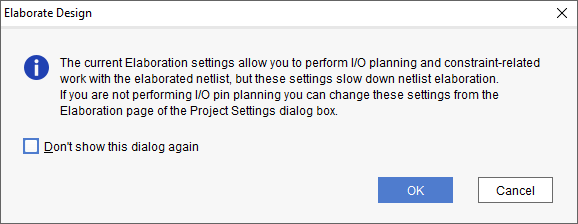
**Schematic (from RTL Analysis)**

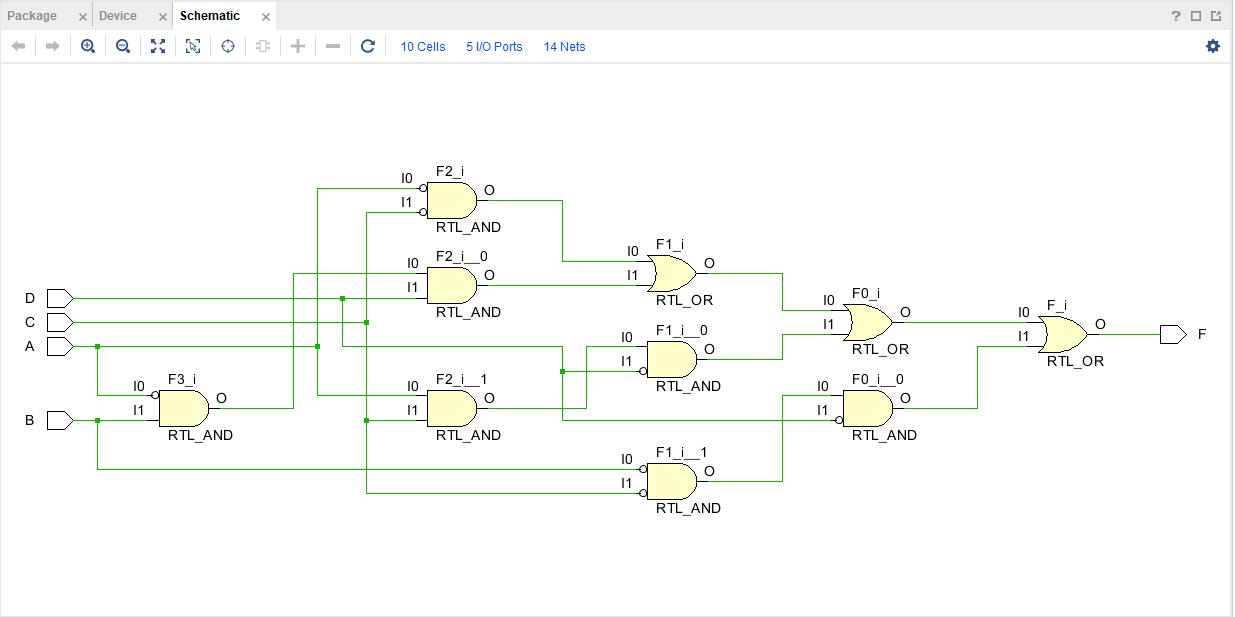
* Under **RTL Analysis**, select ***Open Elaborated Design***
* (Select ***Yes*** and ***OK***if the two windows below appear)
* Select ***Schematic***(schematic is shown on the next slide)











**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):

1. 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.
2. 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.
3. 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.



**Jumper JP1**

**Jumper JP2**

**Power switch**

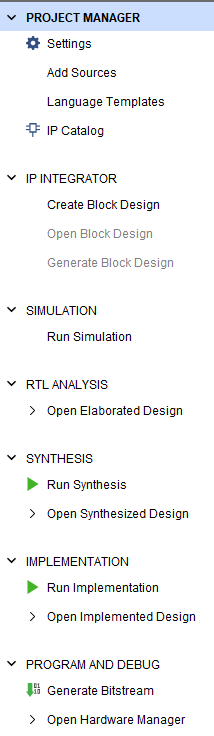
**USB2 port**

**Done LED**

* ***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**

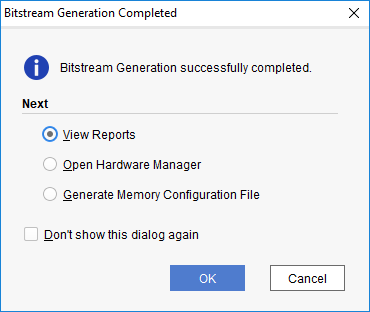
* Check to be sure that Jumper JP1 has been moved to the middle position (JTAG).
* Under **Program and Debug** (scroll down in the Flow Navigator), select ***Generate Bitstream***



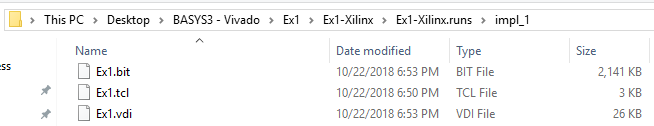
* If the **Launch Runs** window appears, select ***OK***.
* This may take a while to run. Check the status in the upper right corner of the screen.



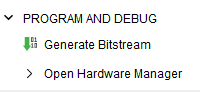
* **Bitstream Generation Completed** (window will appear when completed) - select ***OK***.



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



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

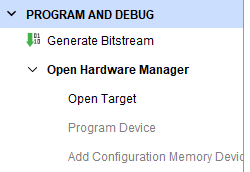


* After opening the **Hardware Manager**, you may see the window below with the message: ***No hardware target is open.***



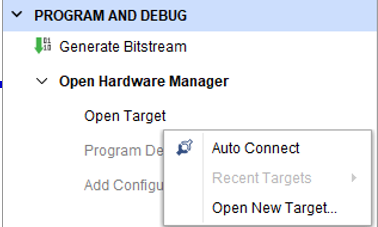
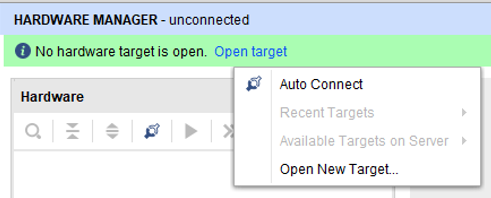
* Under **Program and Debug** (or in the **Hardware Manager** window), select ***Open Target***

****

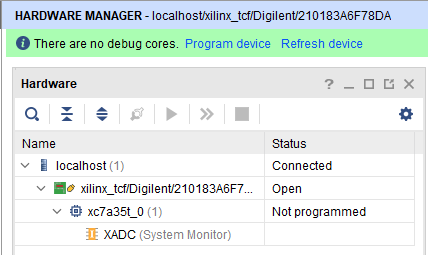


**Generate the bitstream** – **Method A (continued)**

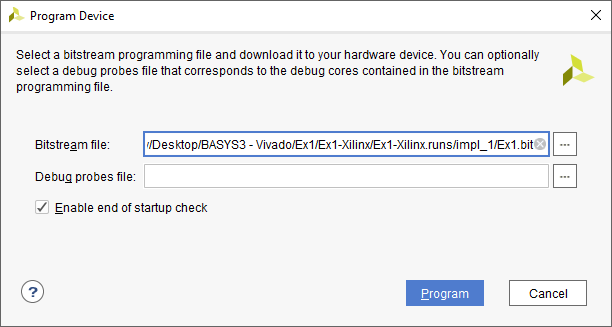
* Select ***Auto Connect*** (use either window below)



* Select ***Program Device***

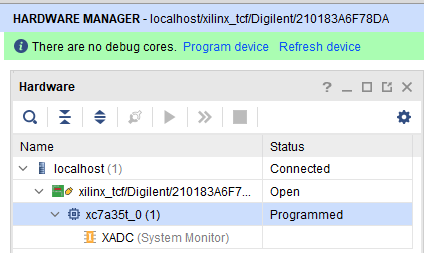


* Select ***Program*** (note that the file Ex1.bit was automatically filled in for the Bitstream file)



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

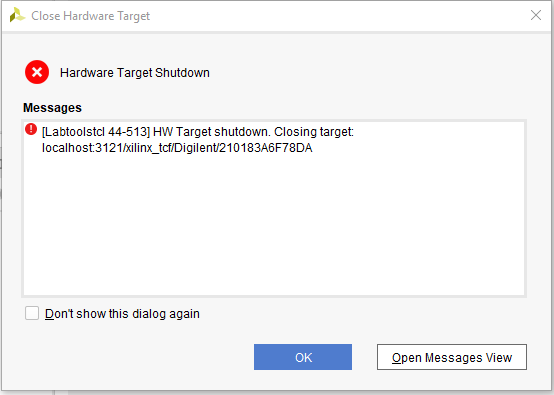
* The **Hardware Manager** should now indicate that the FPGA has been ***Programmed.***



* 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)

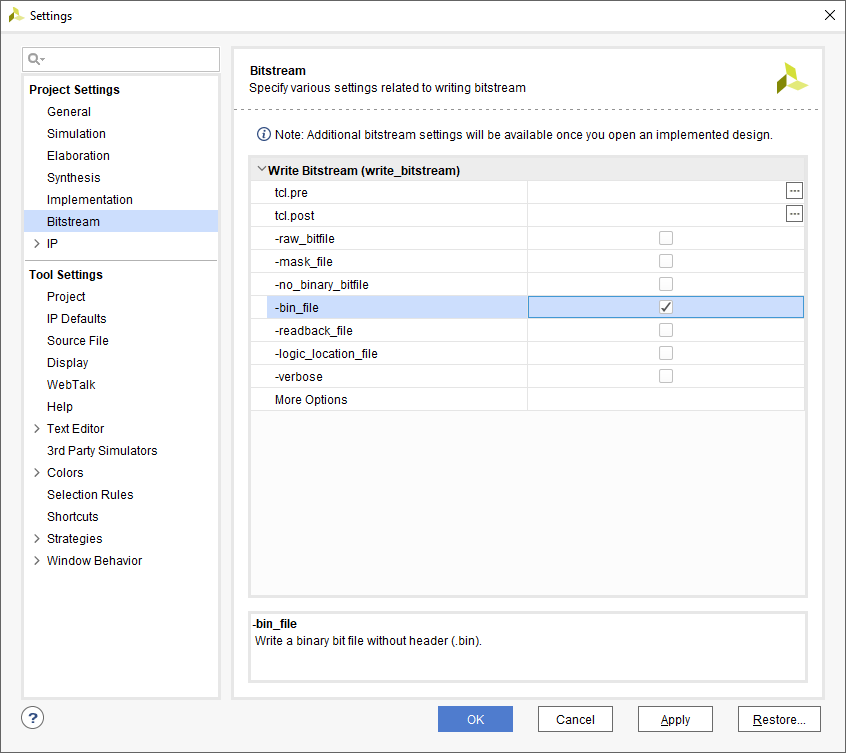


**Generate the bitstream – Method B: Program the FPGA using serial flash memory**

* ***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***



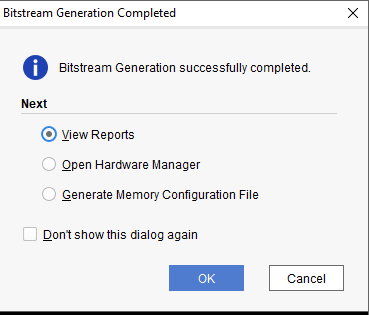
* ***Check the box*** next to ***-bin\_file*** and then select ***OK***



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

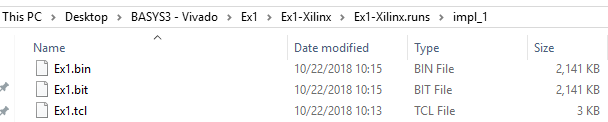


* If the **Bitstream Generation Completed** window appears, select **OK**.



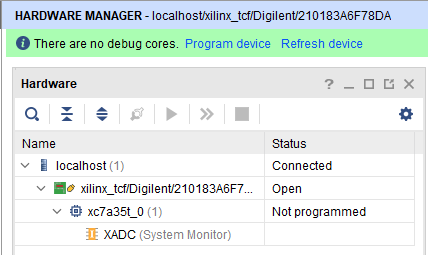
* + Note that a **bin file** has been created. This image is just shown for reference, but you will need to know the location of this file later. It should be in the ***/YourProjectFolder/YourFileName.runs/impl\_1/ folder***.

(in this example in the ***/Ex1-Xilinx/Ex-Xilinx.runs/impl\_1/*** folder)

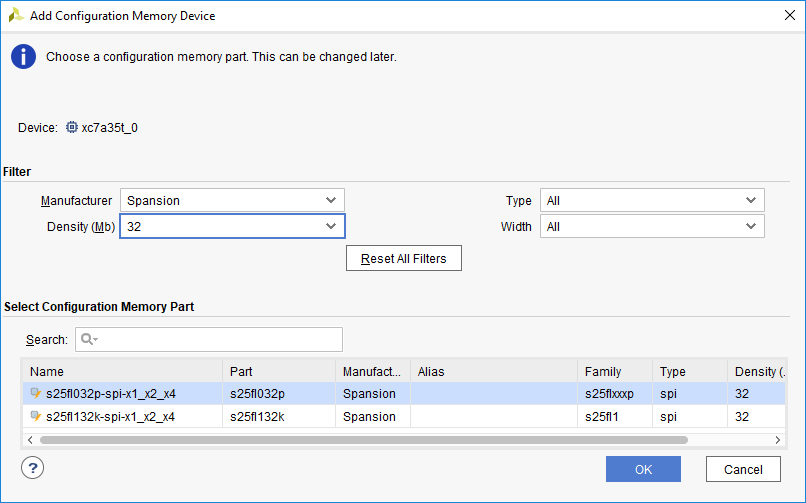
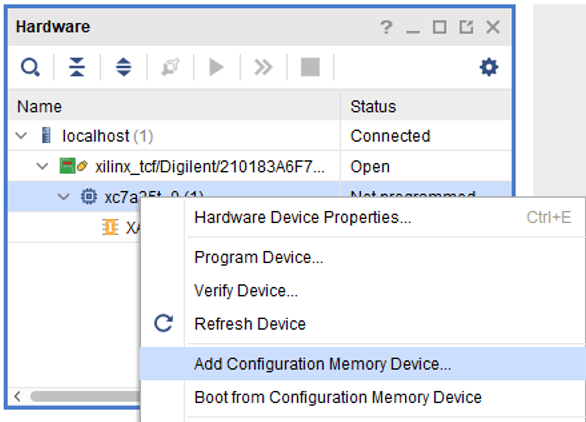


As shown earlier under Method A:

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

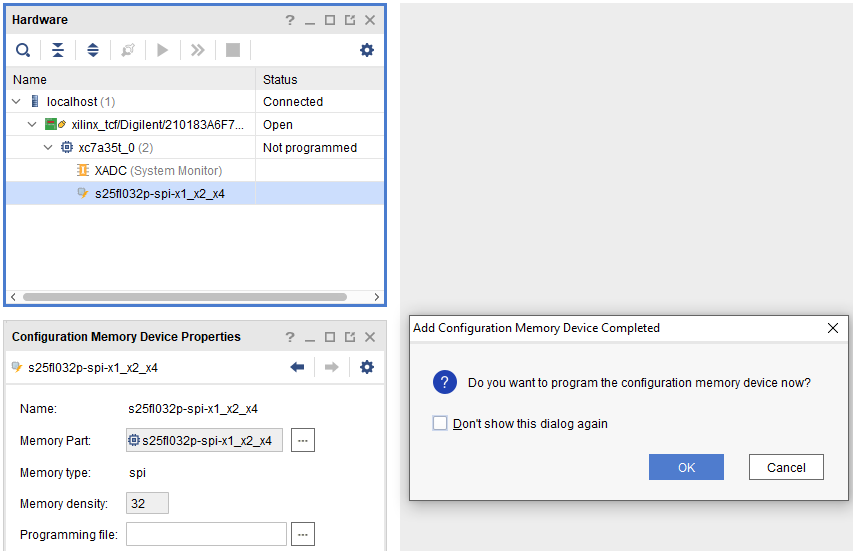


* ***Right- click on the FPGA*** (xc7a35t\_0) and select ***Add Configuration Memory Device***
* When the **Add Configuration Memory Device** window opens:
  + **Manufacturer** – Select ***Spansion***
  + **Density** – Select ***32MB***
  + **Name** – Select ***s25fl032p-spi-x1\_x2\_x4***
  + Select ***OK***.



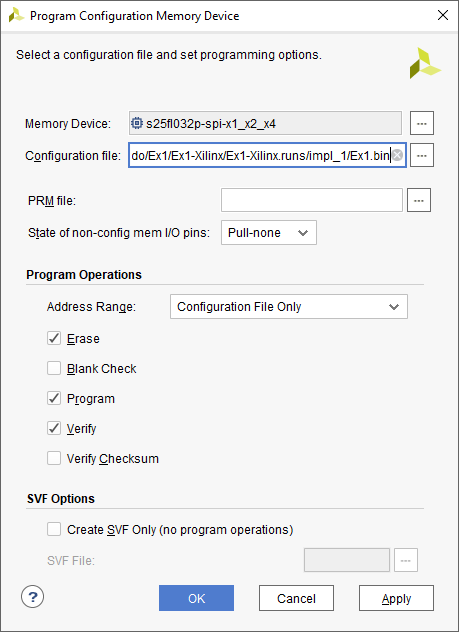
(The 32MB serial flash memory on the BASYS3 is made by Spansion)

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



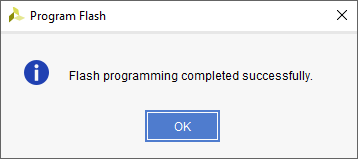
**Program Configuration Memory Device** window:

* ***Select …*** to find the bin file
* Select the **bin file**. The file and path should appear
* ***…/Ex1-Xilinx/Ex-Xilinx.runs/impl\_1/Ex1.bin*** for this example
* Select ***OK***



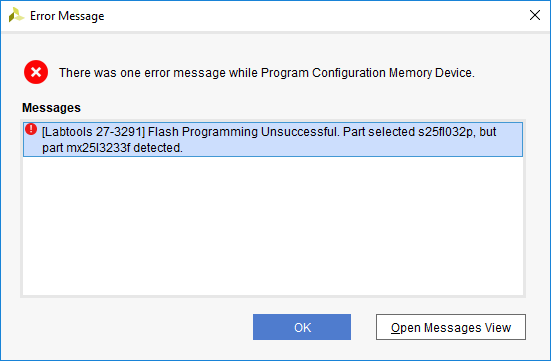
**Select … to find the bin file**

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

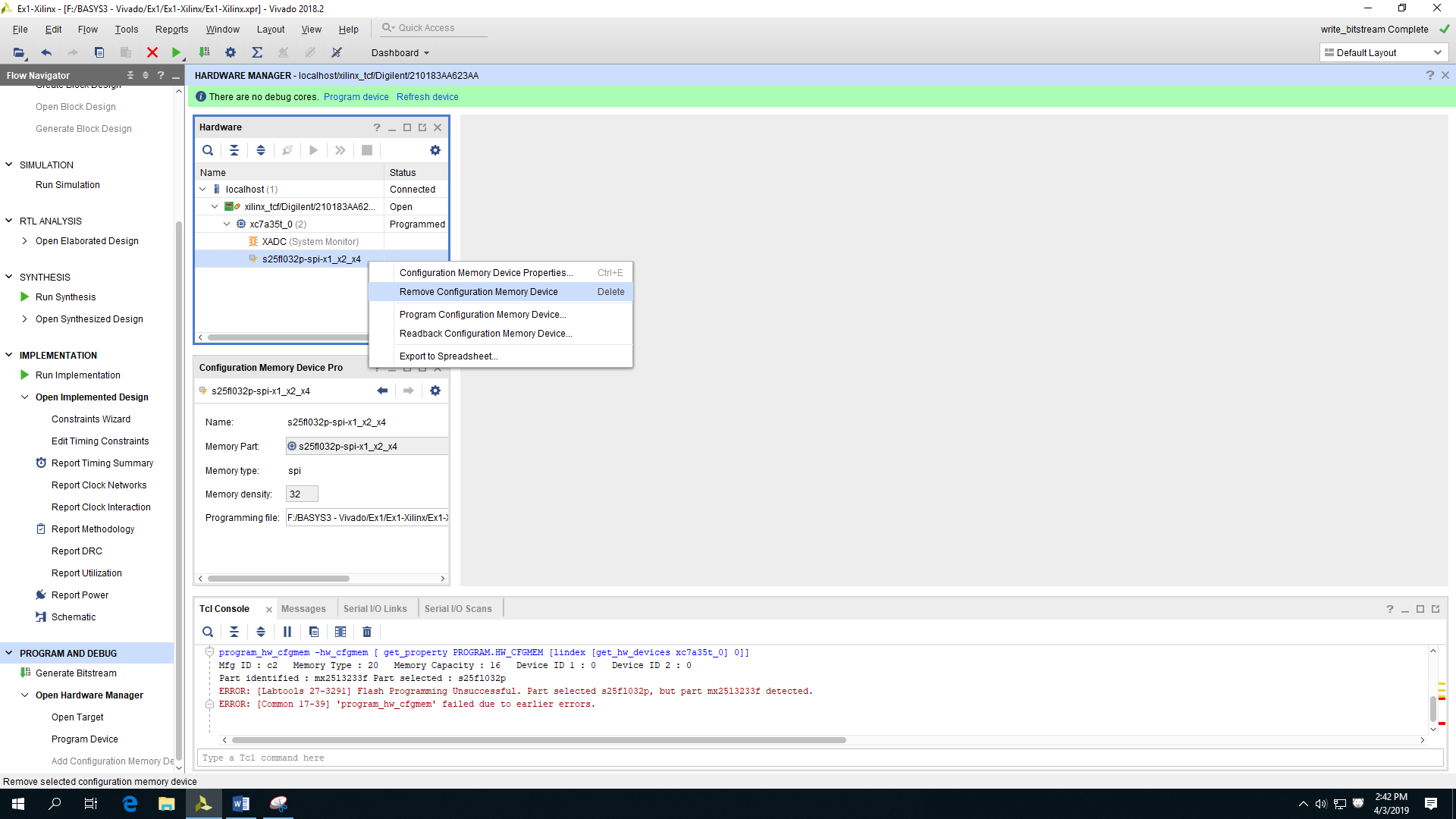


You might get an error message like the one below about the memory chip selected because apparently some of our BASYS3 boards use the Spansion flash memory chip and some use the Macronix flash memory chip.

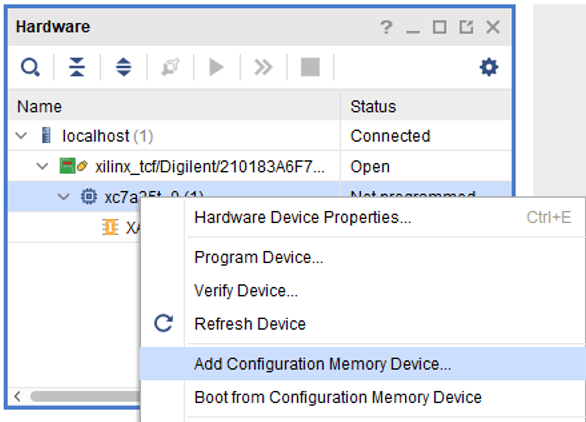
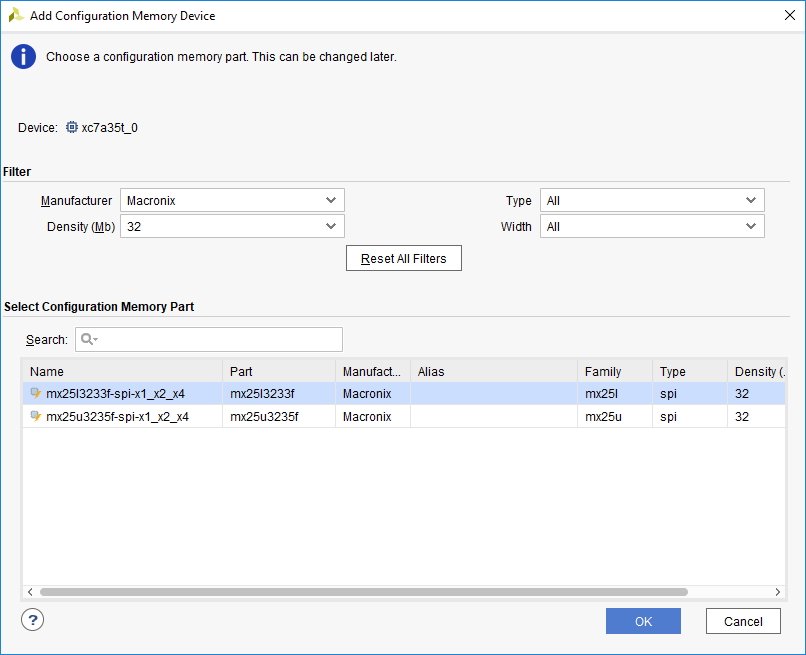
* If this message occurs, select ***OK***. (If not skip ahead to **In order to use the design loaded into flash memory:** )



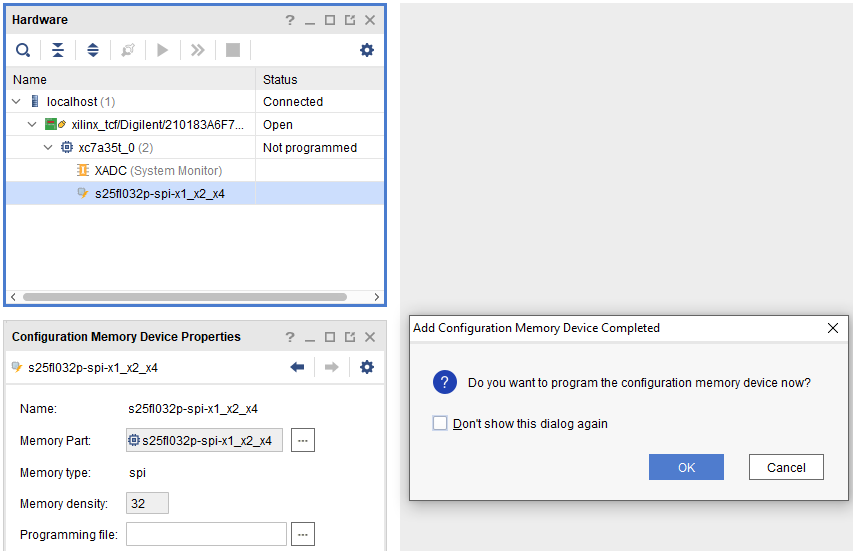
* Right-click on the Spansion memory device (s25fl032p-spi-x1\_x2\_x4) and select ***Remove Configuration Memory Device***



* ***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***.

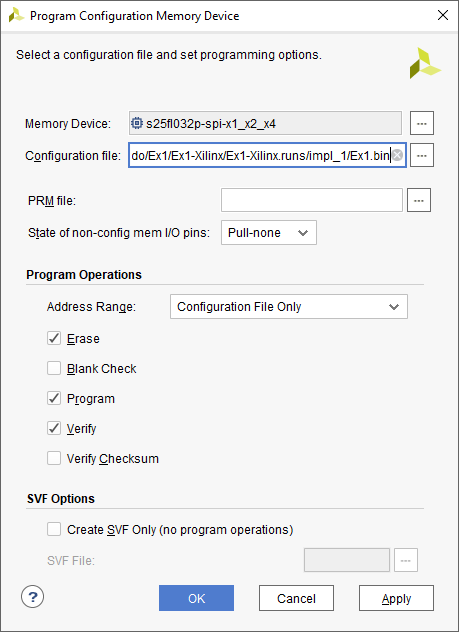


* Do you want to program the configuration memory device now? Select ***OK***



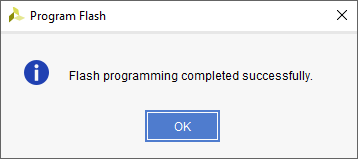
**Program Configuration Memory Device** window:

* ***Select …*** to find the bin file
* Select the **bin file**. The file and path should appear
* ***…/Ex1-Xilinx/Ex-Xilinx.runs/impl\_1/Ex1.bin*** for this example
* Select ***OK***



**Select … to find the bin file**

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



**In order to use the design loaded into flash memory:**

* **Power**: Power off the Basys3 board
* **JP1 Jumper**: Move the jumper from JTAG to QSPI
* **Power**: Power back on the BASYS3 board
* **Wait for LED**: 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.
* **Testing**: 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.

Try it again:

* **Press Reset or turn the BASYS3 off and on again**
* **Wait for LED**: 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.
* **Testing**: 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.