

## EC311 Vivado Verilog, Simulation, and FPGA Tutorial

This tutorial will give a step-by-step walk through of the Vivado design environment. This tutorial demonstrates how to design a Verilog project, test it via simulation, and programs a Xilinx NEXYS A7 FPGA board.

### Step 1: Creating a new project for a specific board

1. If you are using a **Linux machine**, in a terminal run the following commands to setup and launch Vivado:

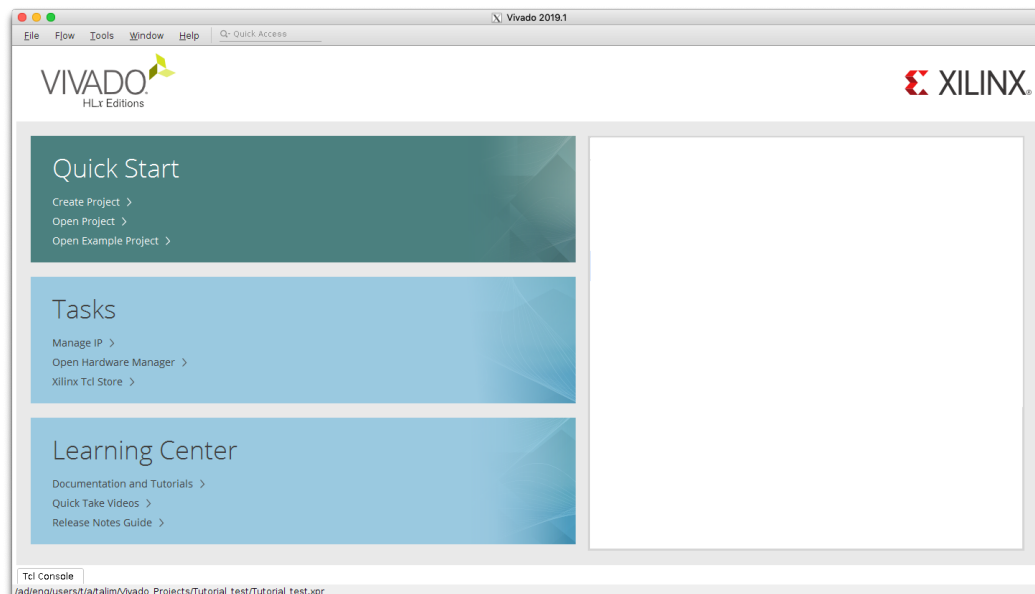
```
> source /ad/eng/opt/xilinx/Vivado/2019.1/settings64.sh
> vivado
```

You should get the following result:

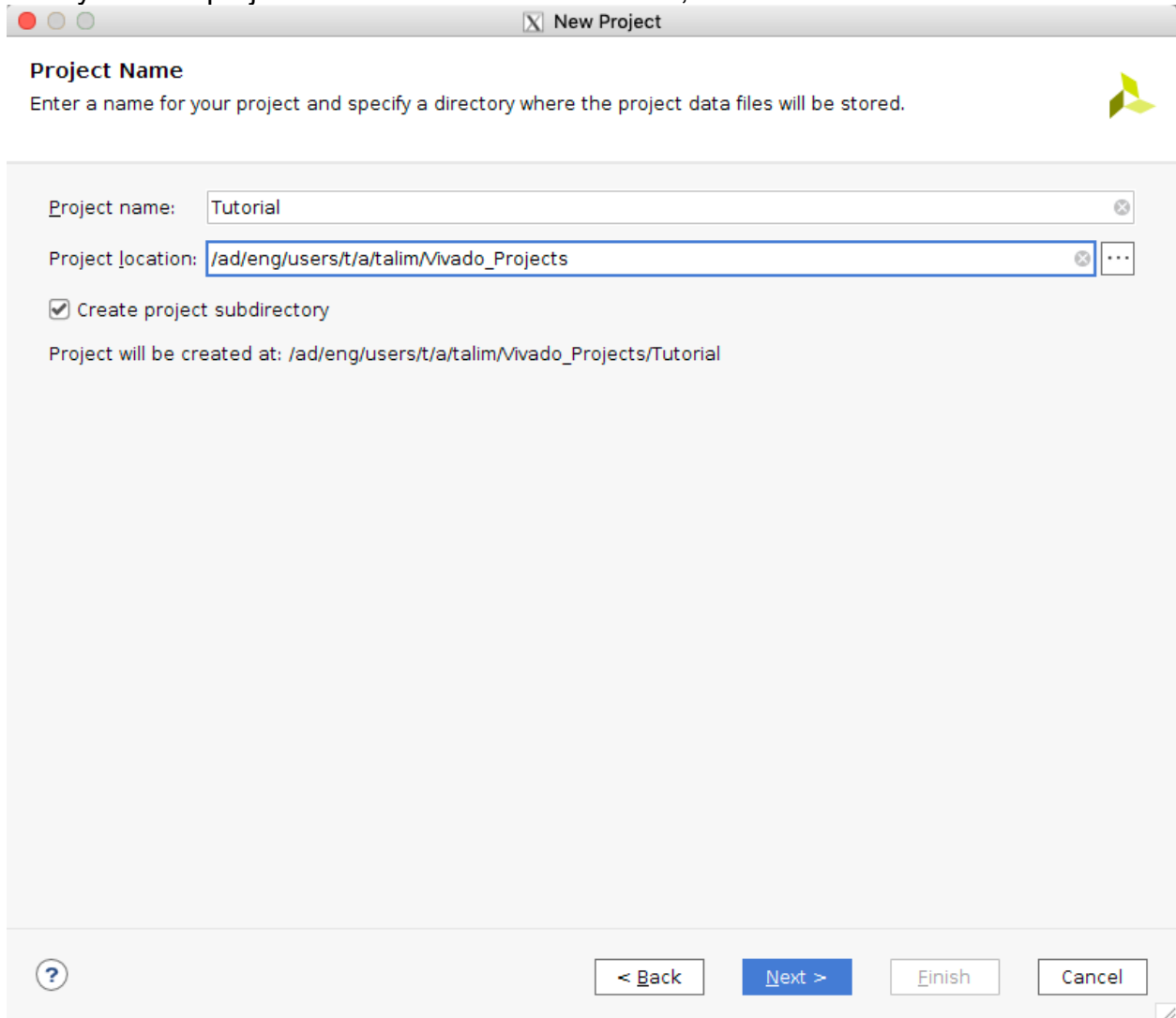
```
***** Vivado v2019.1 (64-bit)
**** SW Build 2552052 on Fri May 24 14:47:09 MDT 2019
**** IP Build 2548770 on Fri May 24 18:01:18 MDT 2019
** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
```

```
> start_gui
```

If you are using a **Windows machine**, you can launch Vivado by typing Vivado in search bar in the bottom left hand corner of the screen. Make sure you select “Vivado 2019.1”. Do not select Vivado 2019.1 Tcl Shell or Vivado HLS 2019.1 Command Prompt



2. Click *Create Project*, and then click *Next*.
3. Give your new project a name and location to save, and click *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: Tutorial

Project location: /ad/eng/users/t/a/talim/Vivado\_Projects

☒ Create project subdirectory

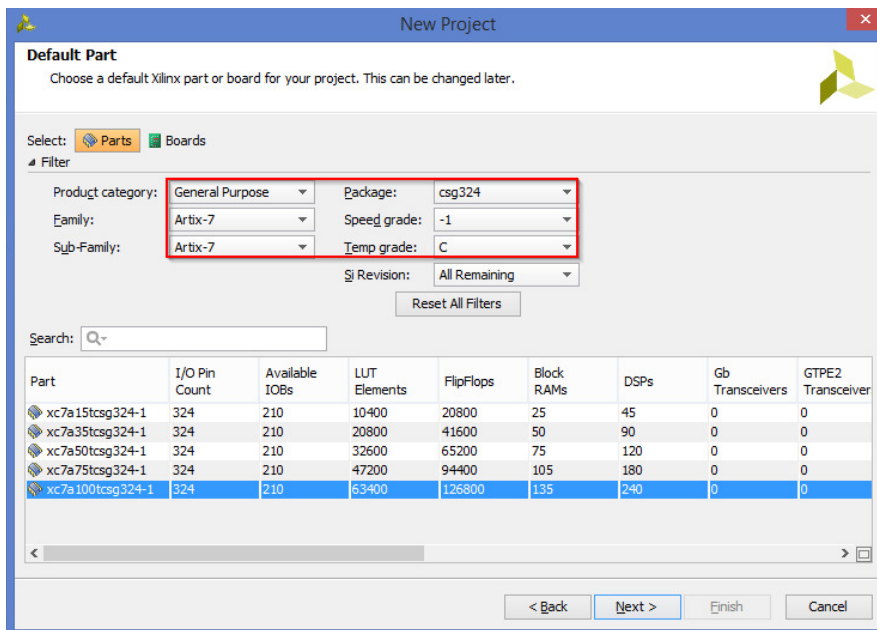
Project will be created at: /ad/eng/users/t/a/talim/Vivado\_Projects/Tutorial

? < Back Next > Finish Cancel

4. Choose *RTL Project* and select the “*Do not specify sources at this time*” check box. Click *Next*.
5. Find the specific FPGA board that we will be using in class, under the ‘Parts’ option, specify (as shown below):

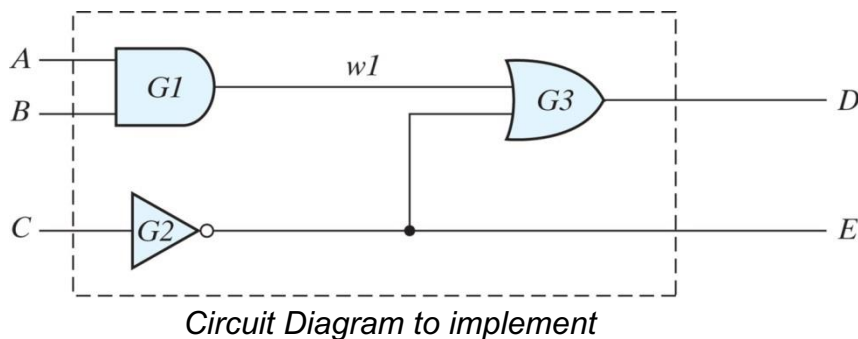
Product category: *All*  
Family: *Artix-7*  
Package: *csg324*  
Speed: *-1*  
Part: *xc7a100tcsg324-1*

Click *Next* and *Finish*.



## Step 2: Verilog gate-level implementation

We will implement the circuit diagram shown below.



### 1. Create a Verilog module:

Select *Project Manager* in the *Flow Navigator*, click on *Add Sources*. Select “Add or create design sources”, and click *Next*.

Click on “Create File”:

File type: Verilog

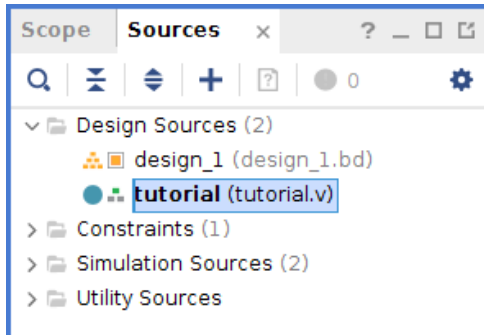
File name: tutorial (or any name you choose)

File location: <Local to Project>

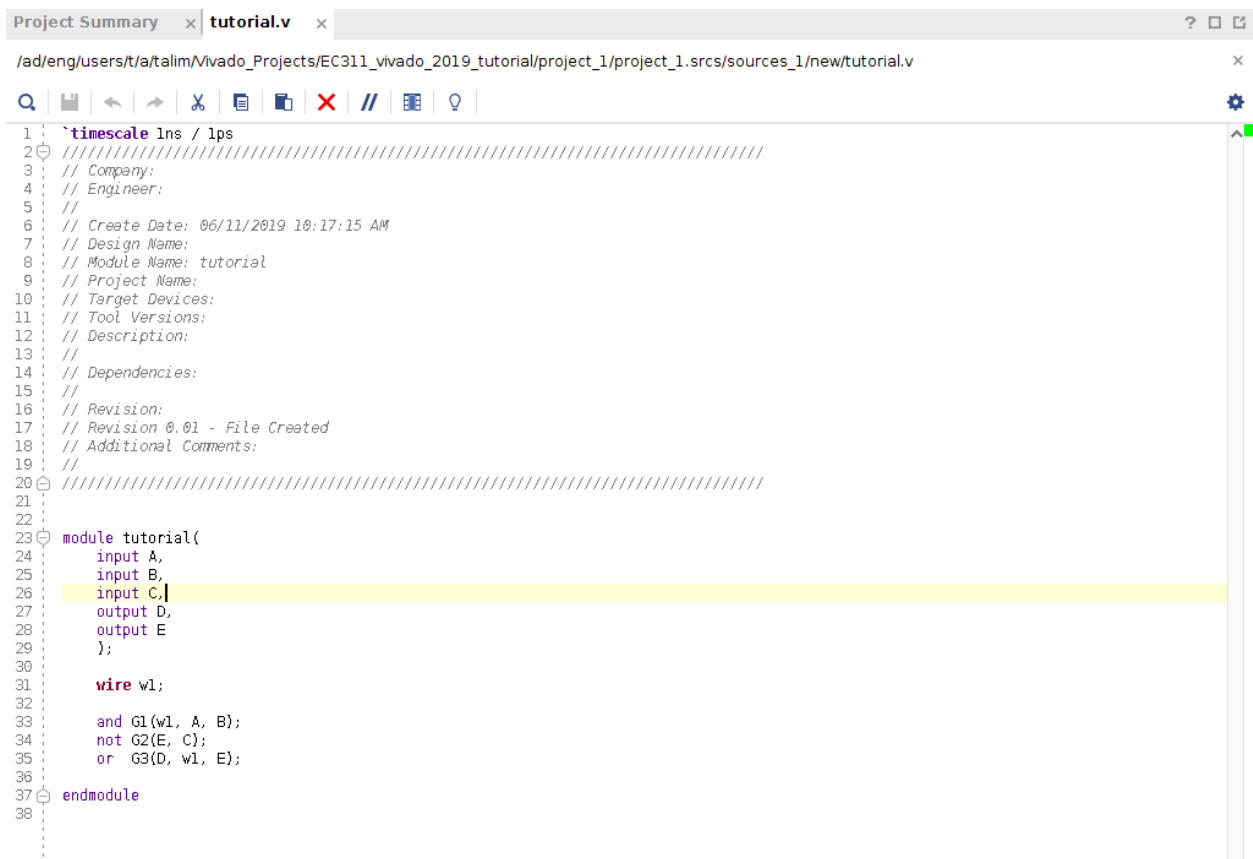
Click *OK*, then *Finish*.

In the 'Define Module' Window specify A, B, and C as inputs, and D and E as outputs and click **OK**.

Double click on the newly created file under the *Design Sources* folder:



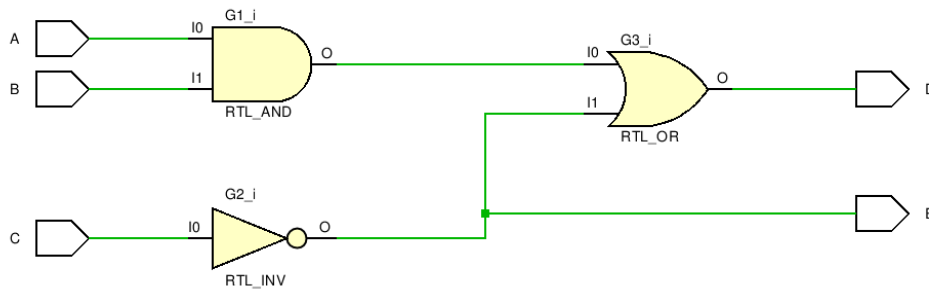
This will open the file for editing. Now edit the file to contain the following:



## 2. Perform RTL analysis:

Expand the 'Open Elaborated Design' entry under the 'RTL Analysis' tasks of the Flow Navigator pane and click on *Schematic*. Click **OK** to run the analysis.

The model (design) will be elaborated and a logic view of the design is displayed:



### 3. Add a test bench:

Under *Project Manager* in the *Flow Navigator*, click on *Add Sources*.

Select '*Add or create simulation sources*'.

Select '*Create Files*':

File type: Verilog

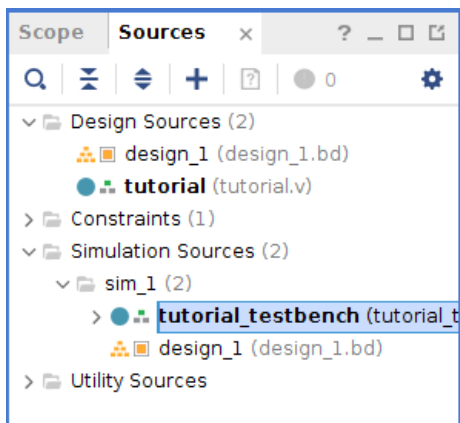
File name: tutorial\_testbench (or any name you choose)

File location: <Local to Project>

Click *OK*, then *Finish*.

Click *OK* in the '*Define Module*' Window.

- Double click on the newly created file under the *Simulation Sources* folder in the *Sources* tab:



This will open the file for editing. Now edit the file to contain the following:

```

1  `timescale 1ns / 1ps
2  // Company:
3  // Engineer:
4  //
5  // Create Date: 06/11/2019 10:22:22 AM
6  // Design Name:
7  // Module Name: tutorial_testbench
8  // Project Name:
9  // Target Devices:
10 // Tool Versions:
11 // Description:
12 //
13 // Dependencies:
14 //
15 // Revision:
16 // Revision 0.01 - File Created
17 // Additional Comments:
18 //
19 //
20 //
21
22
23 module tutorial_testbench(
24
25 );
26
27     reg A, B, C; //Inputs
28     wire D, E; //Outputs
29
30     // Instantiate the units under test
31     tutorial tut1(.A(A), .B(B), .C(C), .D(D), .E(E));
32
33     initial
34     begin
35         A = 0;
36         B = 0;
37         C = 0;
38
39         #100 A=1; B=1; C=1;
40         #100 $finish;
41     end
42
43 endmodule
44

```

## 5. Add a constraints file:

Download the default constraints file for Nexys4 DDR, *Nexys4DDR\_Master.xdc*, from the course webpage.

Under *Project Manager* in the *Flow Navigator*, click on *Add Sources*.

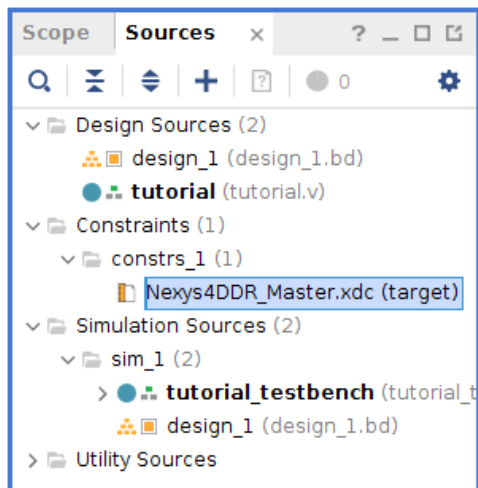
Select *'Add or create constraints'*.

Select *'Add Files'*, browse to the *Nexys4DDR\_Master.xdc* file, click *Open*, and then click *Next*.

Click OK, then *Finish*.

The XDC constraint file assigns the physical I/O locations on FPGA to the switches and LEDs located on the board. This information can be obtained either through a board's schematic or board's user guide.

Double click on the newly created file under the *Constraints* folder:



For this tutorial, we will be connecting the three inputs, A, B, and C, to switches 0, 1, and 2, respectively, and the two outputs, D and E, to LEDs 0 and 1, respectively.

In the constraints file, replace swt[0] with A, swt[1] with B, and swt[2] with C. Comment the rest of the 'swt' lines (with a '#').

In the same file, replace led[0] with D, and led[1] with E, and comment the rest of the 'LED' lines.

#### ##Switches

```
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { swt[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { swt[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { swt[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { swt[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { swt[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { swt[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { swt[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { swt[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]
```

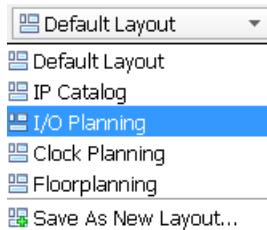
#### ## LEDs

```
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { led[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { led[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { led[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { led[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
```

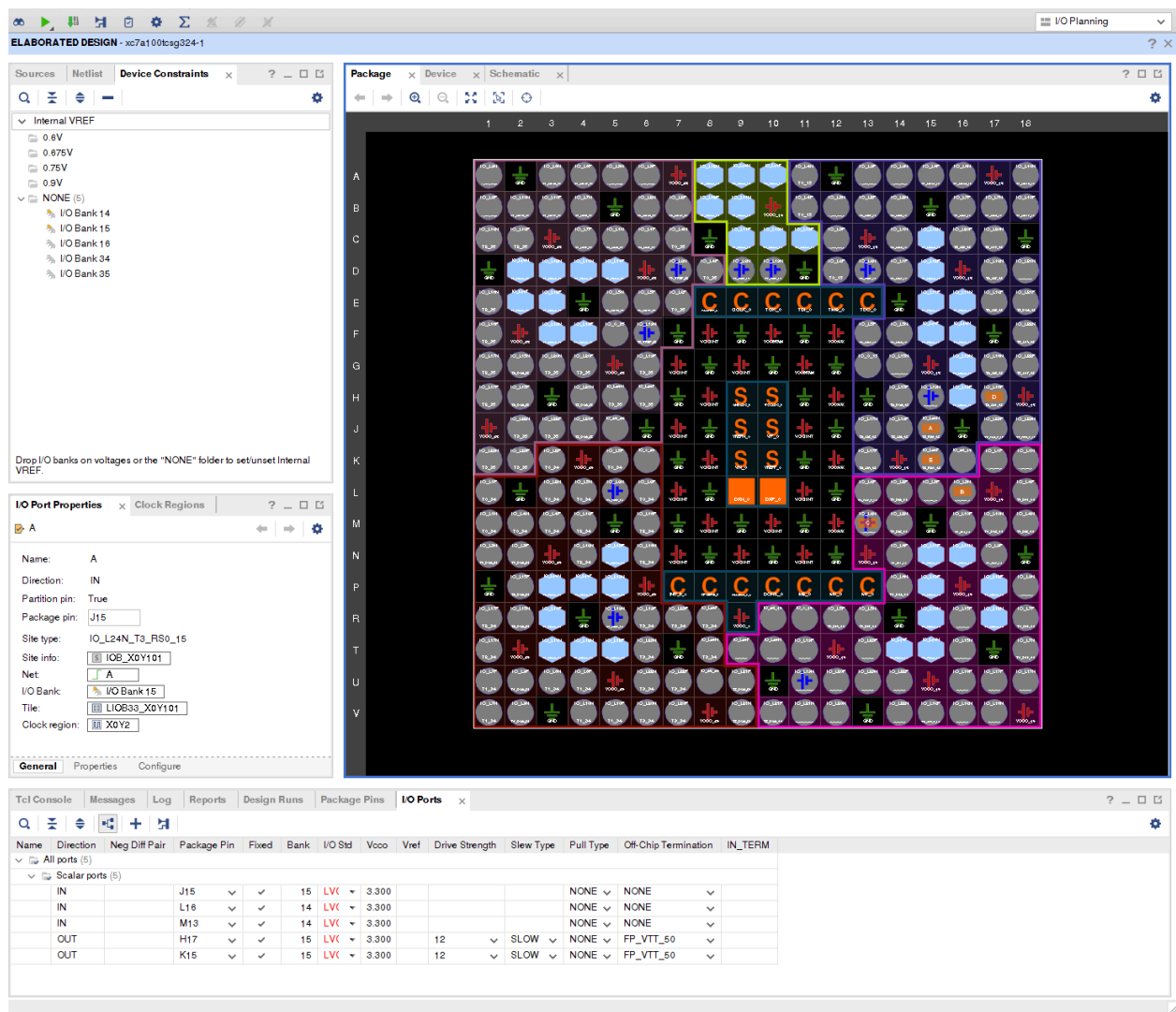
## 6. Add a constraints file – alternative method:

Expand the 'Open Elaborated Design' entry under the 'RTL Analysis' tasks of the Flow Navigator pane and click on Schematic. Click OK to run the analysis.

Once '*RTL analysis*' is performed, another standard layout called the I/O Planning is available. Click on the drop-down button at the top left corner and select the I/O Planning layout.



Notice that the Package view is displayed in the Auxiliary View area, RTL Netlist tab is selected, and I/O ports tab is displayed in the Console View area. Also notice that design ports are listed in the I/O Ports tab on the bottom.



Move the mouse cursor over the Package view, highlighting different pins. Notice the



pin site number is shown at the bottom of the Vivado GUI, along with the pin type (User IO, GND, VCCO...) and the I/O bank it belongs to.

Select the '*I/O Ports*' tab on the bottom window and change the '*Package Pin*' labels to the switches and LED labels you want to use. Also change the '*I/O Std*' column to LVCMOS33 as shown below.

See Appendix A for more information on Pin labels on your board.

Tcl Console   Messages   Log   Reports   Design Runs   Package Pins   I/O Ports x													
Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
v All ports (5)													
v Scalar ports (5)													
IN			J15	✓	15	LVCMOS33*	3.300				NONE	NONE	✓
IN			L16	✓	14	LVCMOS33*	3.300				NONE	NONE	✓
IN			M13	✓	14	LVCMOS33*	3.300				NONE	NONE	✓
OUT			H17	✓	15	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	✓
OUT			K15	✓	15	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	✓

Note that the names of the inputs and outputs for the selected port appear in the '*I/O Port Properties*' tab:

**I/O Port Properties** x Clock Regions ? \_ □ ↗

A

Name: A

Direction: IN

Partition pin: True

Package pin: J15

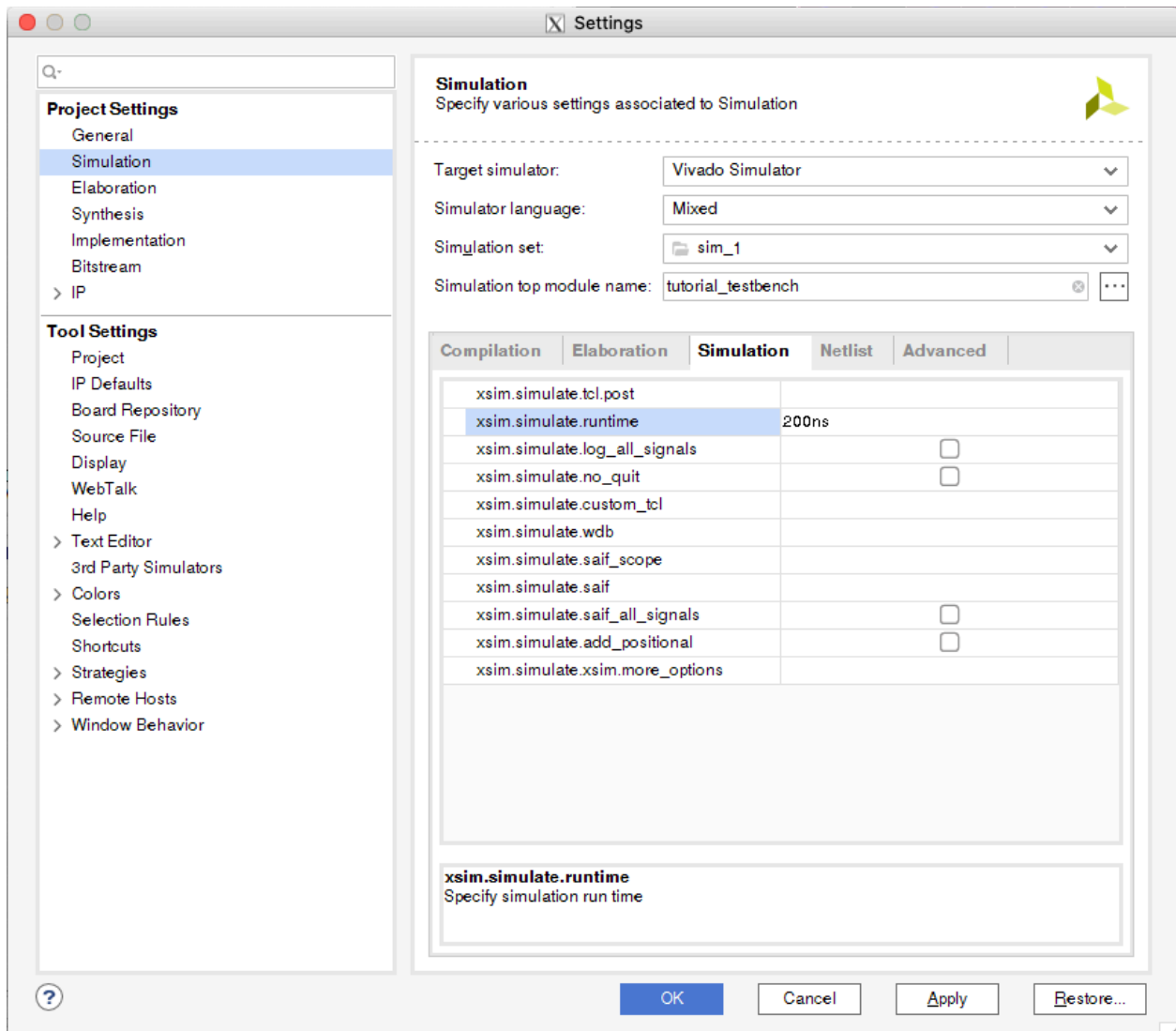
Site type: IO\_L24N\_T3\_RS0\_15

General Properties Configure

Cntrl + S to save. It will ask you if you want to overwrite the existing project. Click "OK" for all dialogue boxes.

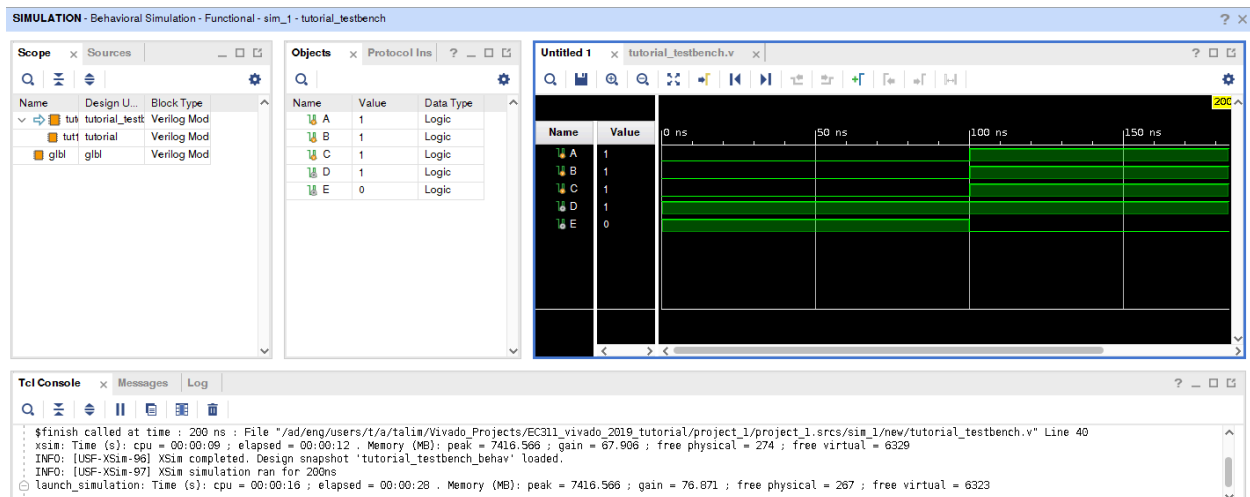
### Step 3: Behavioral Simulation

1. Select '*Settings*' under the '*Project Manager*' tasks of the '*Flow Navigator*' pane. A Project Settings form will appear showing the Simulation properties form.
2. Select the Simulation tab, and set the Simulation Run Time value to 200 ns and click OK.



- Click on *Run Simulation > Run Behavioral Simulation* under the *Flow Navigator* pane. The testbench and source files will be compiled and the XSim simulator will be run (assuming no errors). You will see a simulator output similar to the one shown below. Note that you may need to scroll and/or zoom out in order to see the full simulation.

Make sure that you understand how the waveform reflects the testbench, and that it clearly shows that the provided design functions correctly. It is a good practice to test the output for all the possible combinations of input (whenever possible).



## Step 4: Synthesis, Implementation, Generate Bitstream and Timing Simulation

1. Click on *Run Synthesis* under the *Synthesis* tasks of the *Flow Navigator* pane, and click OK.

The synthesis process will be run on the tutorial.v file (and all its hierarchical files if they exist). When the process is completed a *Synthesis Completed* dialog box with three options will be displayed.

2. Select the *Run Implementation* option and click OK. The implementation process will be run on the synthesis output files. When the process is completed an *Implementation Completed* dialog box with three options will be displayed.
3. Select *Generate Bitstream* and click OK.
4. Select *View Reports* and click OK. Select the *Project Summary* tab (you may have to change to the Default Layout view) and observe the results.

Notice that the actual resource utilization (under *Utilization* and *Table* on the bottom left) is one LUT and 5 IOs. Also, it indicates that no timing constraints were defined for this design (since the design is combinatorial).

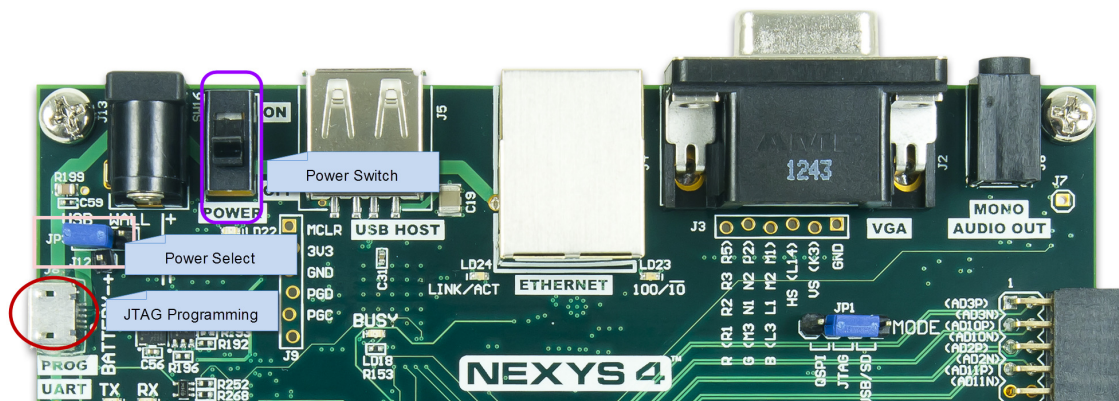
5. Select *Run Simulation > Run Post-Implementation Timing Simulation* process under the *Simulation* tasks of the *Flow Navigator* pane.

The XSim simulator will be launched using the implemented design and tutorial\_testbench as the top-level module.

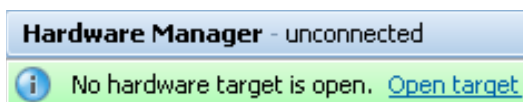
Click on the Zoom Fit button to see the waveform window from 0 to 200 ns.

## Step 5: Program the FPGA

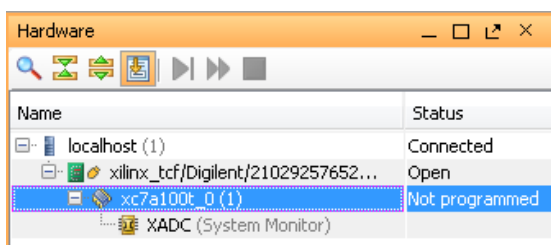
1. Make sure that the power supply source jumper is set to USB and the provided Micro-USB cable is connected between the board and the PC. Note that you do not need to connect the power jack and the board can be powered and configured via USB alone. Power ON the switch on the board.



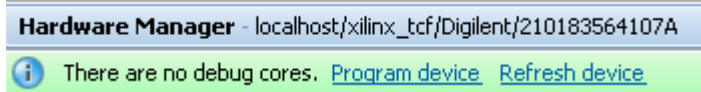
2. Select the *Open Hardware Manager* under *Program and Debug* In the *Flow Navigator* tab.
3. Click on the Open target link.



4. From the dropdown menu, click *Auto Connect*.
5. The Hardware Session status changes from Unconnected to the server name and the device is highlighted. Also notice that the Status indicates that it is not programmed.



6. Click '*Program device*' on the top green bar, then *Program*. A green LED labeled '*DONE*' should light up. You may see some other LEDs lit depending on switch positions.



7. Test the functionality of your design by toggling the three rightmost switch and inspecting the response in the two rightmost LEDs. The LED should respond according to the gate-level diagram.

## Appendix A: Digilent Nexys4 DDR Board Reference Manual

### 1. Input / output pins

The Nexys4 board includes sixteen slide switches, five push buttons, sixteen individual LEDs, and an eight-digit seven-segment display, as shown in the below diagram. The five pushbuttons, arranged in a plus-sign configuration, are “momentary” switches that normally generate a low output when they are at rest, and a high output only when they are pressed. Slide switches generate constant high or low inputs.

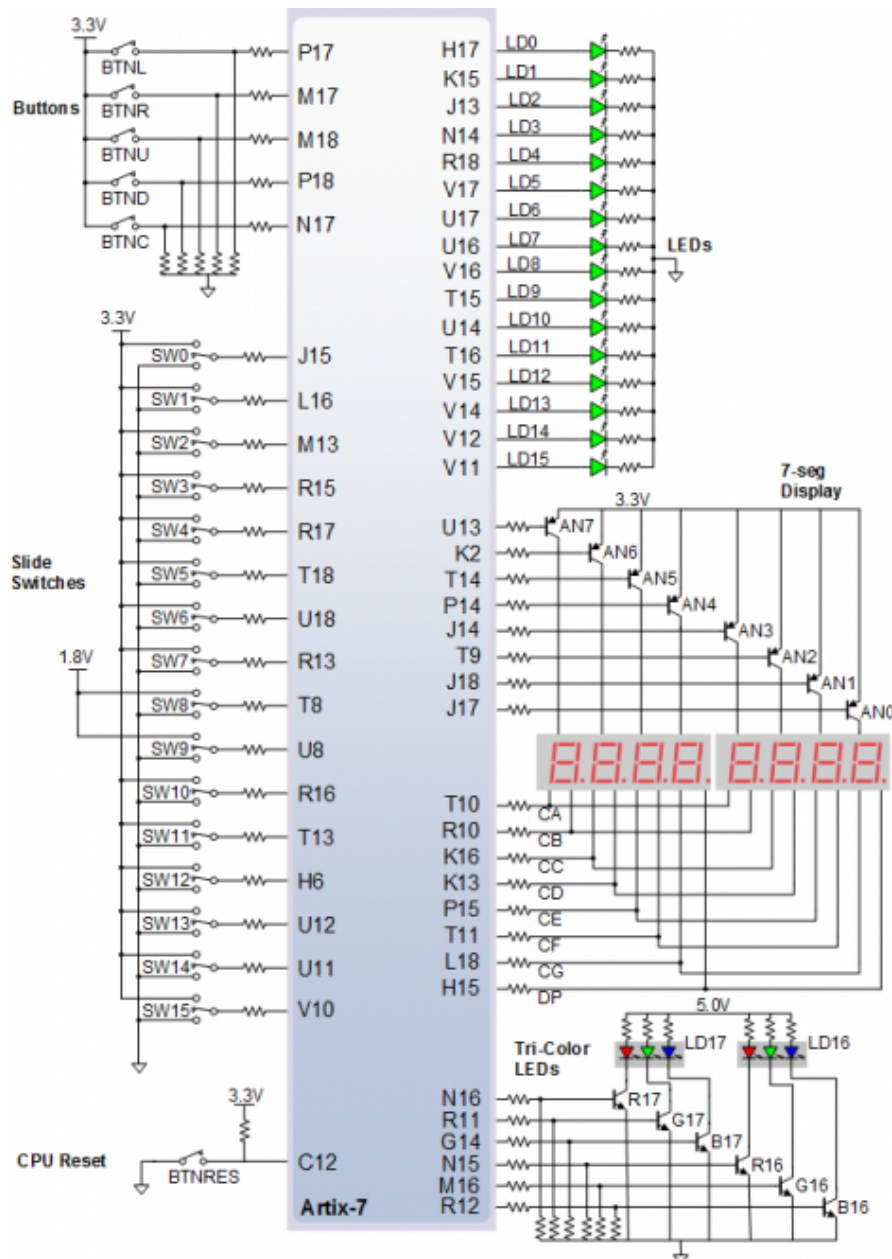


Figure 16. General Purpose I/O devices on the Nexys4 DDR.

## 2. Seven Segment Display

The Nexys4 board also includes 8 seven-segment displays.

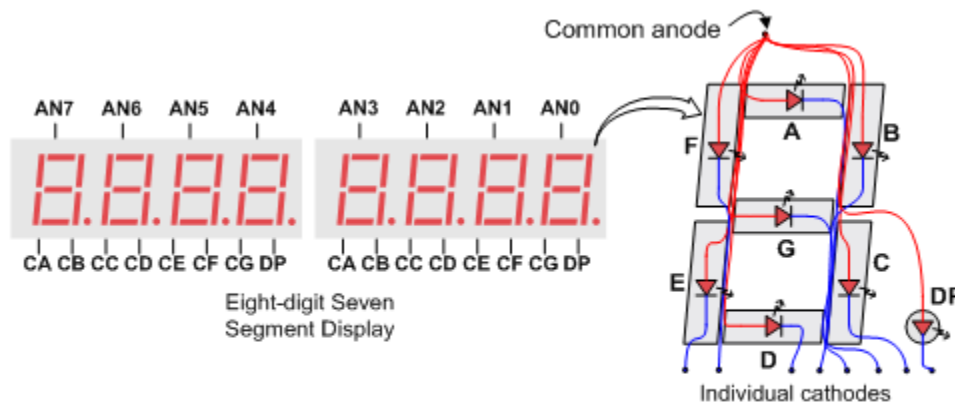


Figure 18. Common anode circuit node.

To illuminate a segment, the anode should be driven high while the cathode is driven low. However, since the Nexys4 uses transistors to drive enough current into the common anode point, the anode enables are inverted. Therefore, both the AN0..7 and the CA..G/DP signals are driven low when active.

## 3. Oscillators / Clocks

The Nexys4 board includes a single 100MHz oscillator (clock) connected to pin E3 (E3 is a MRCC input on bank 35).

Additional information can be found here:

<https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual>