



AUBURN

UNIVERSITY

SAMUEL GINN
COLLEGE OF ENGINEERING

ELEC 4200 Lab#0 Tutorial



AUBURN

UNIVERSITY

SAMUEL GINN
COLLEGE OF ENGINEERING

Objectives(1)

- In this Lab exercise, we will design and implement a 2-to-1 multiplexer (MUX), using Xilinx *Vivado* tools to create a VHDL model of the design, verify the model, and implement the model in a Field Programmable Gate Array (FPGA)
- The behavior of the multiplexer will be implemented in two different ways:
 - 1) Logic equations, which should be familiar from Digital Logic Circuits.
 - 2) Behavioral description, in which we specify the desired input/output behavior and allow the *Vivado* synthesis tool to implement the required logic.

Objectives(2)

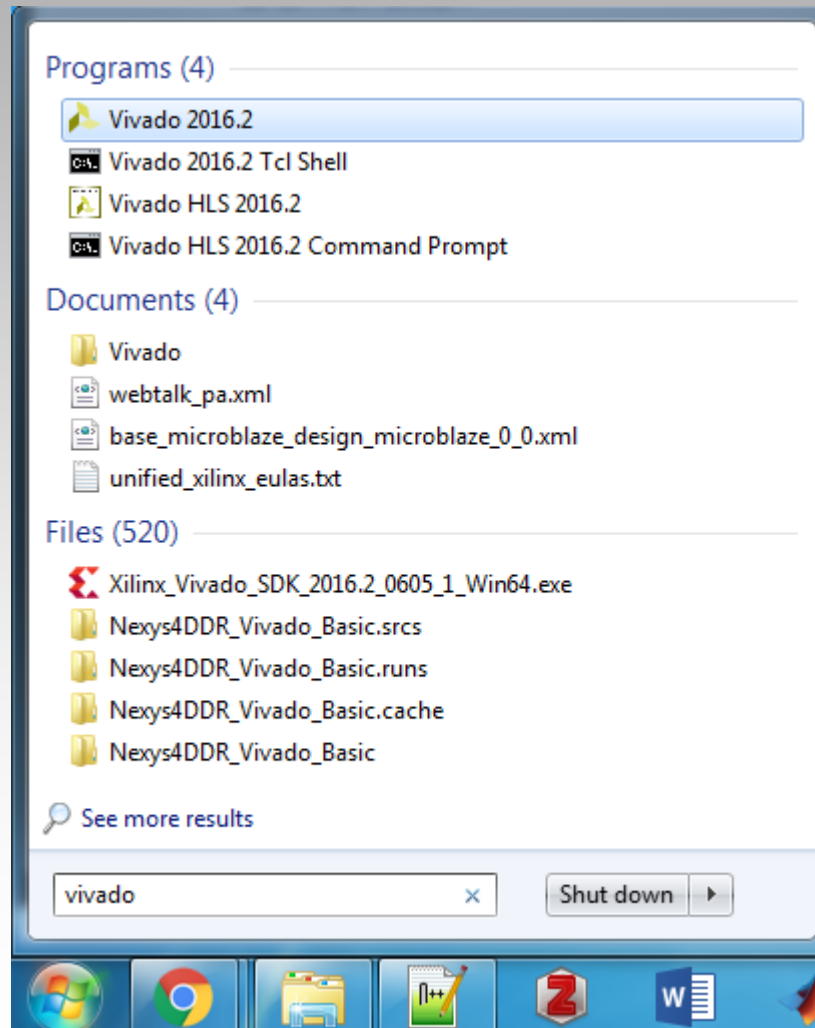
- Aldec *Active-HDL* tools will be used to simulate and verify the design, with model debugging and correction performed as needed.
- After the design is fully verified, we will use *Vivado* to synthesize the design into an Artix-7 FPGA, generating a configuration data file which will then be downloaded onto a Diligent *Nexus 4 DDR* circuit board, where the implemented design will be verified on the hardware using switches to stimulate the circuit and LEDs to observe the outputs.
- If you have no idea what any of this means, don't worry. That is why this is a tutorial!

Warning:

- You will need to actually read the instructions.
- If you attempt to just look at the pictures, you will miss steps, make mistakes, encounter errors, etc.
- When you ask for help with problems resulting from not reading the directions, your GTA will make fun of you before telling you to read the directions.
- You have been warned.

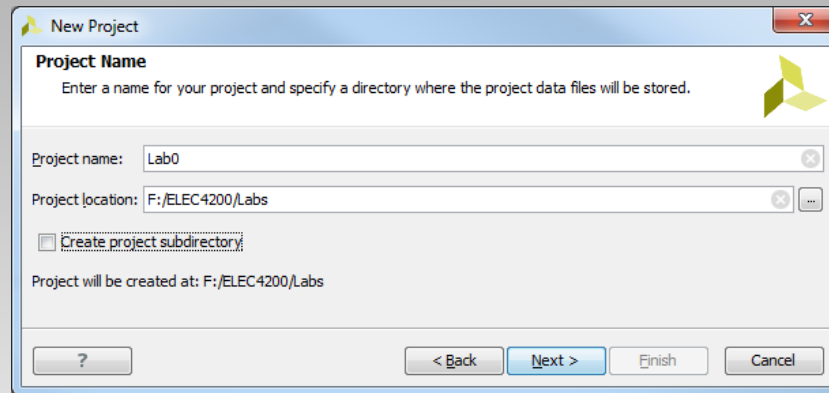
New Project (1)

- Click the start menu, and type “Vivado.” Then look for “Vivado 2016.2” (or the latest installed version) and click on it.



New Project (2)

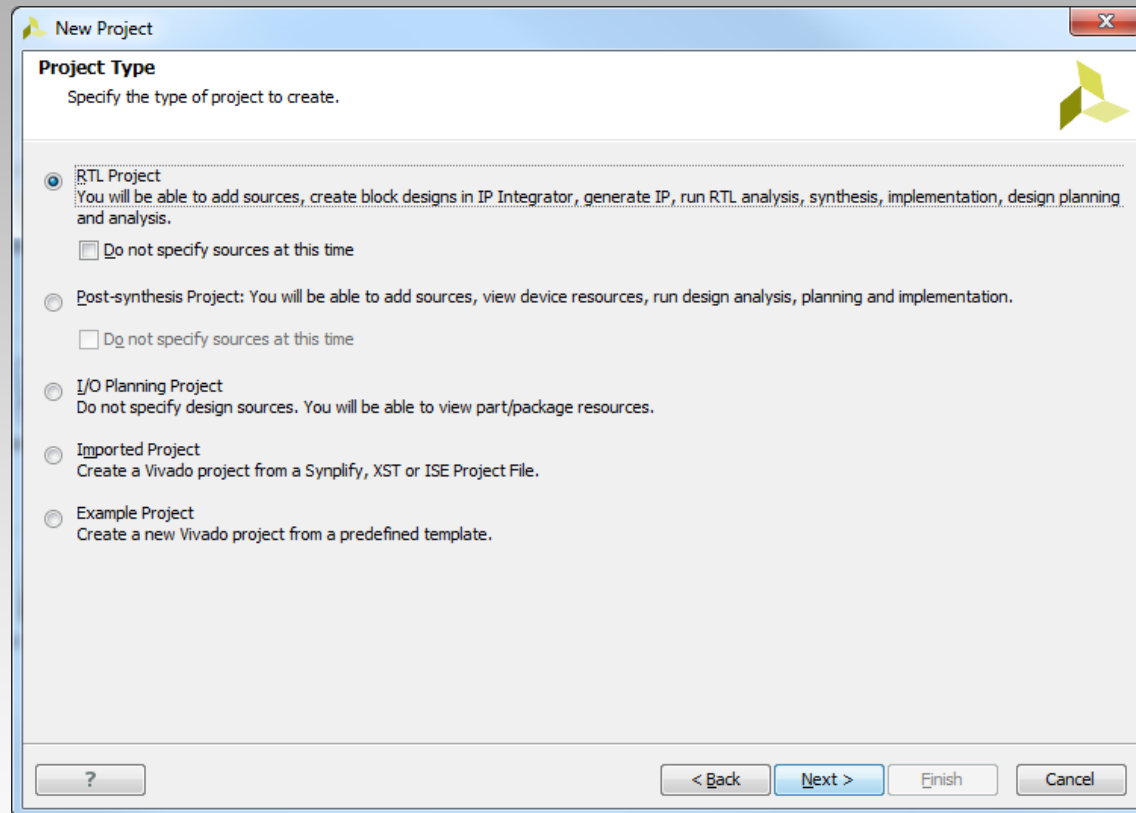
- Launch the *New Project Wizard* by clicking “Create New Project” on the *Quick Start* page, or “New Project” in the “File” dropdown menu.
- Click “Next” on the first screen to produce the *New Project* window.



- Enter the project name and location in the corresponding boxes.
- Note: It is **HIGHLY** recommended that while working in the lab your projects be located on an external flash drive or the C: drive (NOT your H: drive). If your project is located on the H: drive one of the later synthesis steps will **ALWAYS** fail! If using the C: drive you can use the “C:\TEMP\” directory.
- Give the project a descriptive name, like “LAB0” and click “Next”.

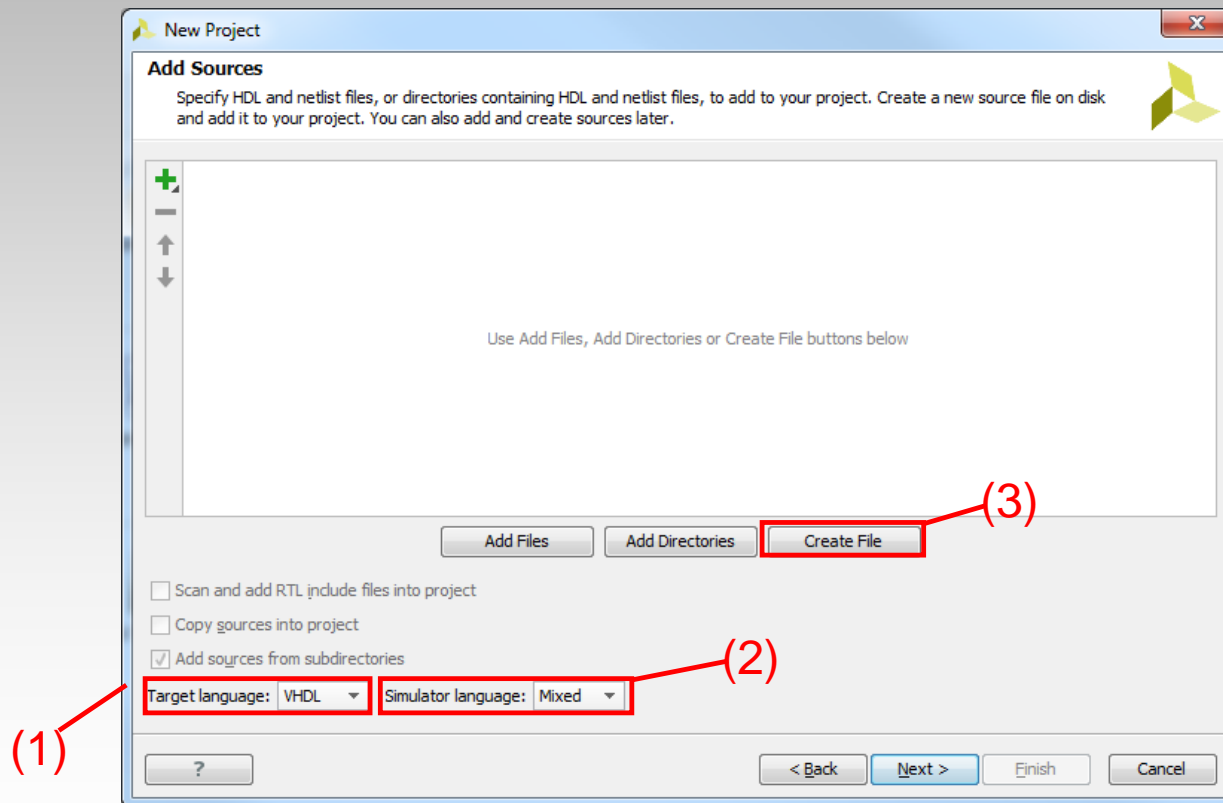
New Project (3)

- Select “RTL Project” as the project type, and click “Next”.



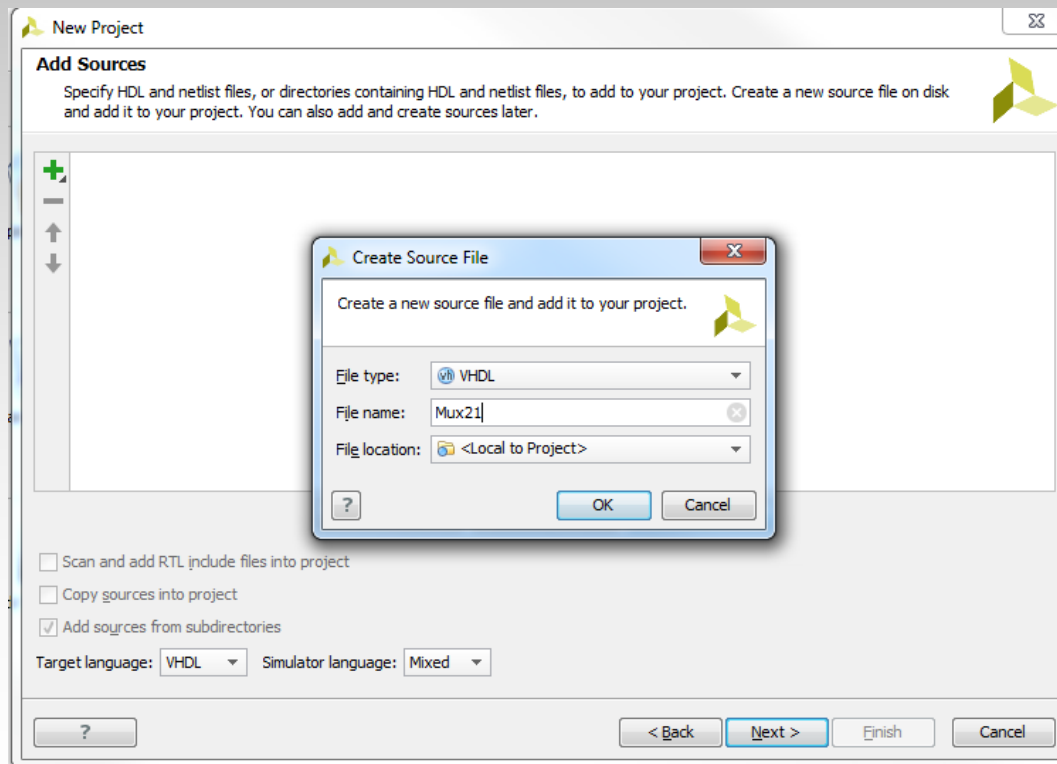
New Project (4)

- In the *Add Sources* window, set the “Target Language” to VHDL and the “Simulator language” to Mixed.
- After setting the language options, click “Create File”



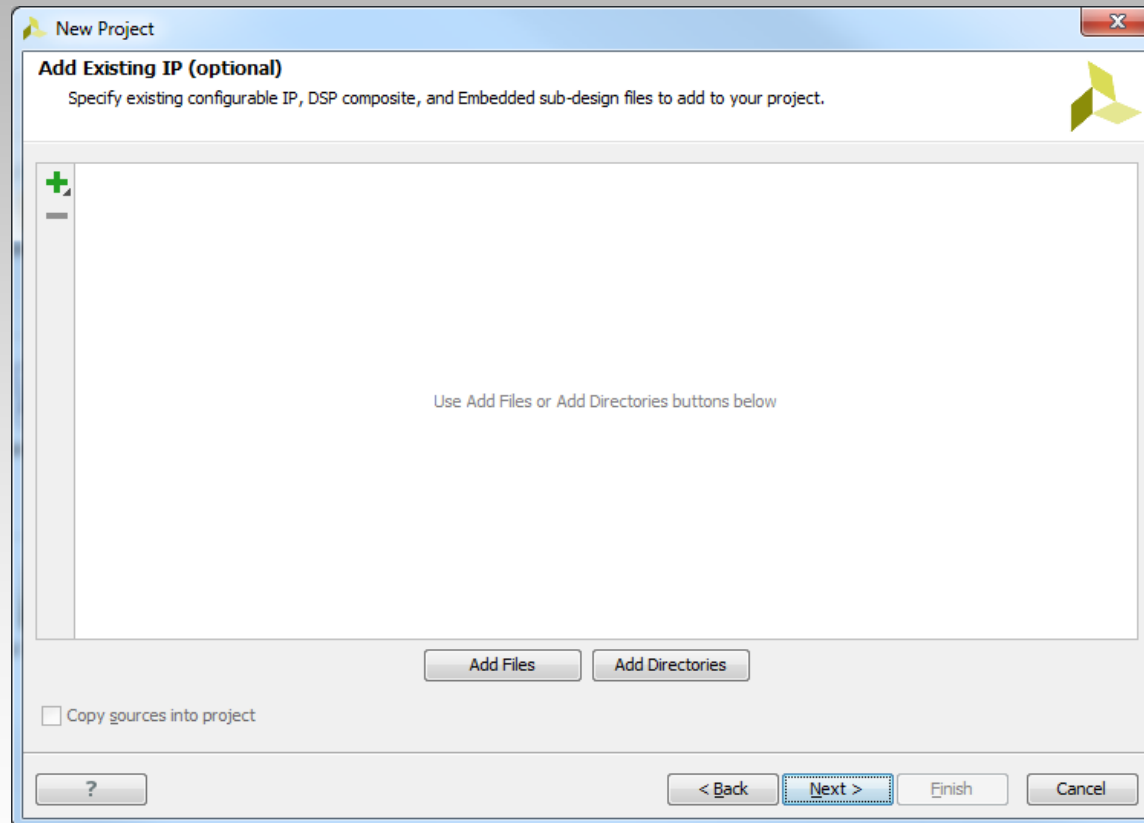
New Project (5)

- In the Create Source File dialog box, make sure the “File type” is set to VHDL and enter the name of your file.
 - Again, it is always a good idea to use a descriptive name, in this case *Mux21* because this is a 2-1 multiplexor.
- Click “OK” to close the dialog box and then “Next”.



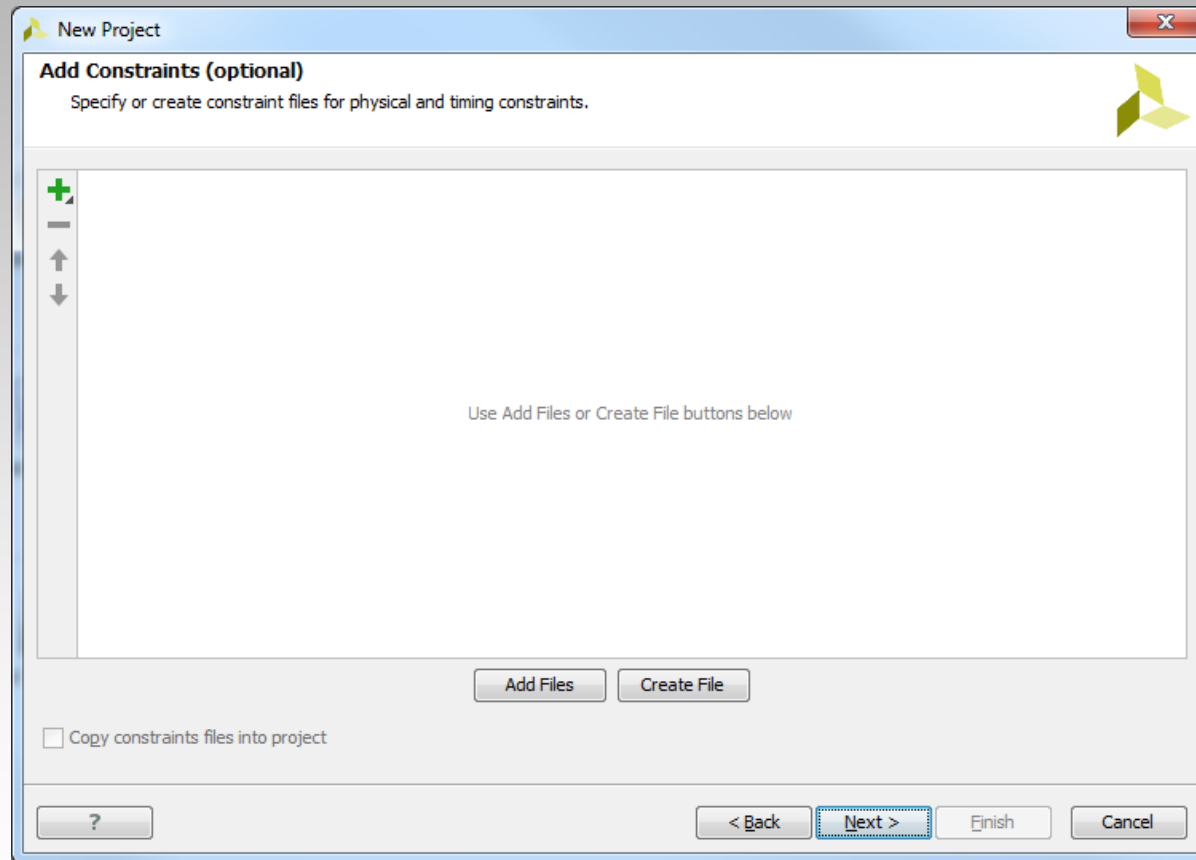
New Project (6)

- We are not using any existing IP (intellectual property components), so click “Next” again.



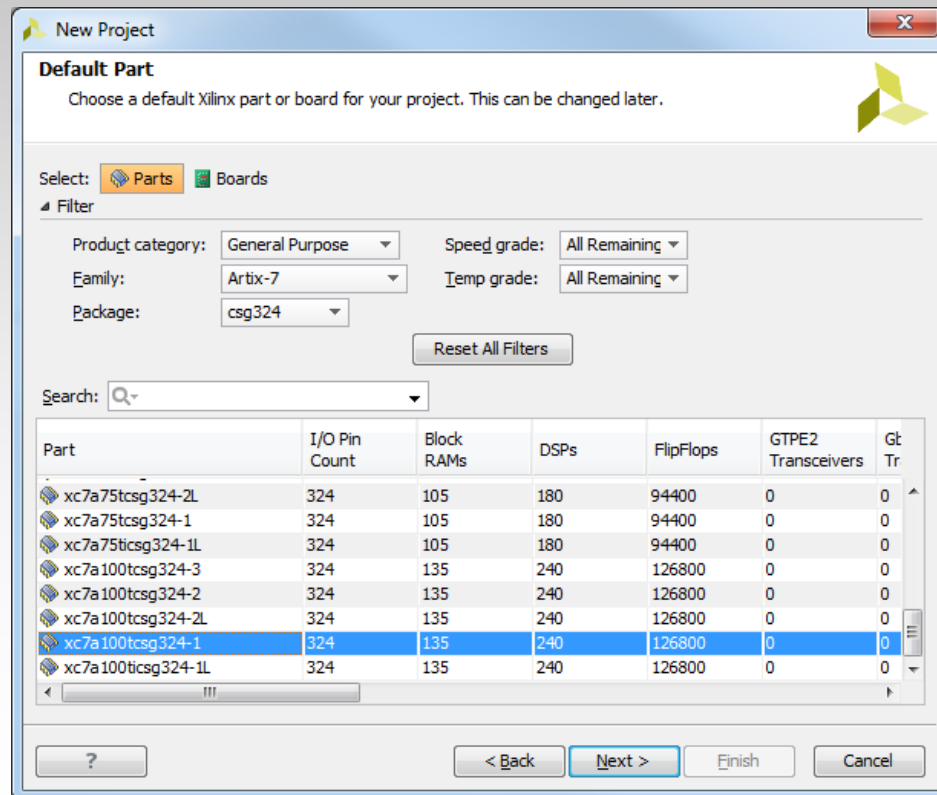
New Project (7)

- We do not have any constraints yet, but in future projects, you may import and edit existing constraints files to save time. So click “Next” for now.



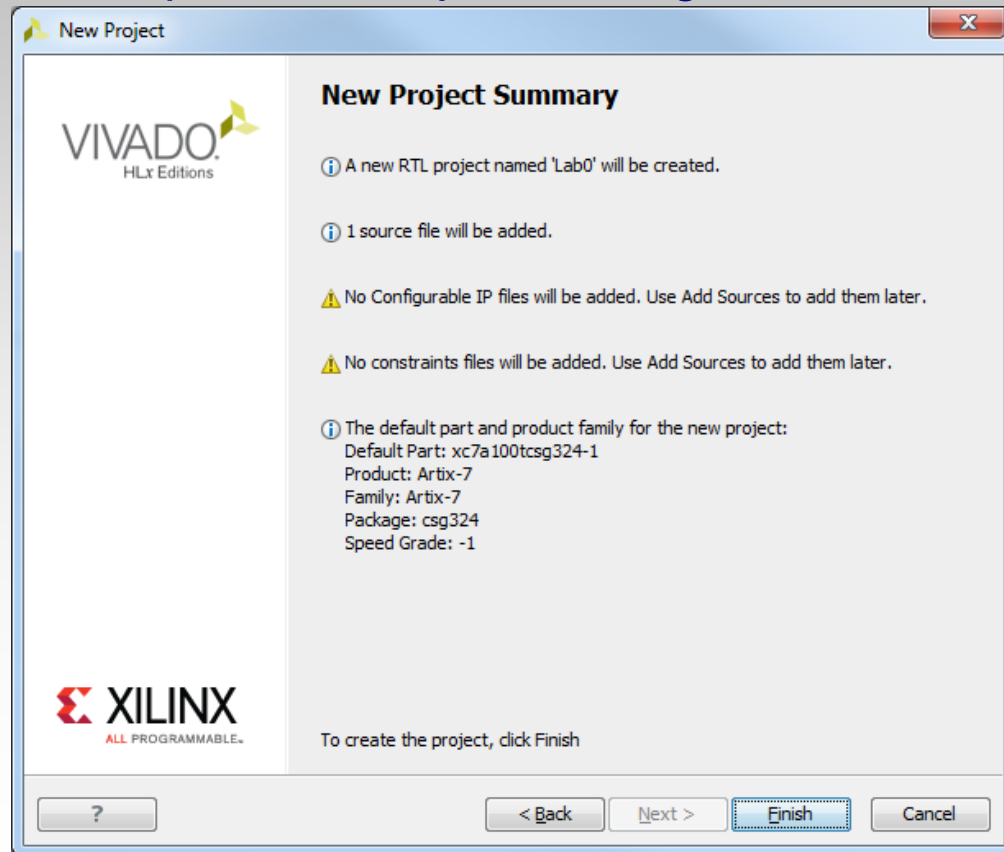
New Project (8)

- Here we need to select our device. You can use the search function, filters, or just scroll until you find our device: **XC7A100tcsg324-1**.
 - This FPGA is from the Xilinx Artix-7 family (**XC7A100T**).
 - The device is contained in a 324-pin **csg324** package.
 - The speed grade of the part is “-1”.
- Click “Next”



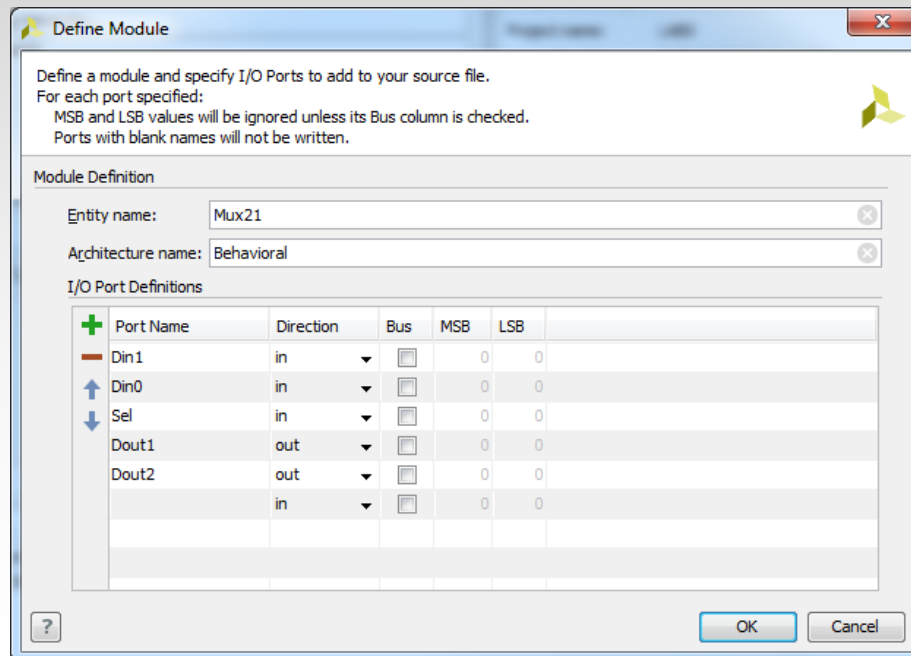
New Project (9)

- The New Project Summary window lists information selected in the previous screens.
 - If necessary, use the “Back” button to return to previous screens to make changes/corrections.
- Click “Finish” to open the Project Manager window.



Create the VHDL model

- Since we chose to create a new VHDL file, *Vivado* will automatically launch a wizard to assist in creating the entity and architecture structures that comprise a VHDL model. This can all be done by hand (and you can edit all of this later), but there is no reason not to take advantage of the wizard utility.
- Leave the Entity and Architecture names as their defaults.
- Create three inputs (Direction “in”): *Din0*, *Din1*, *Sel*
- Create two outputs (Direction “out”): *Dout1*, *Dout2*
- Click “OK”



Edit VHDL Model(1)

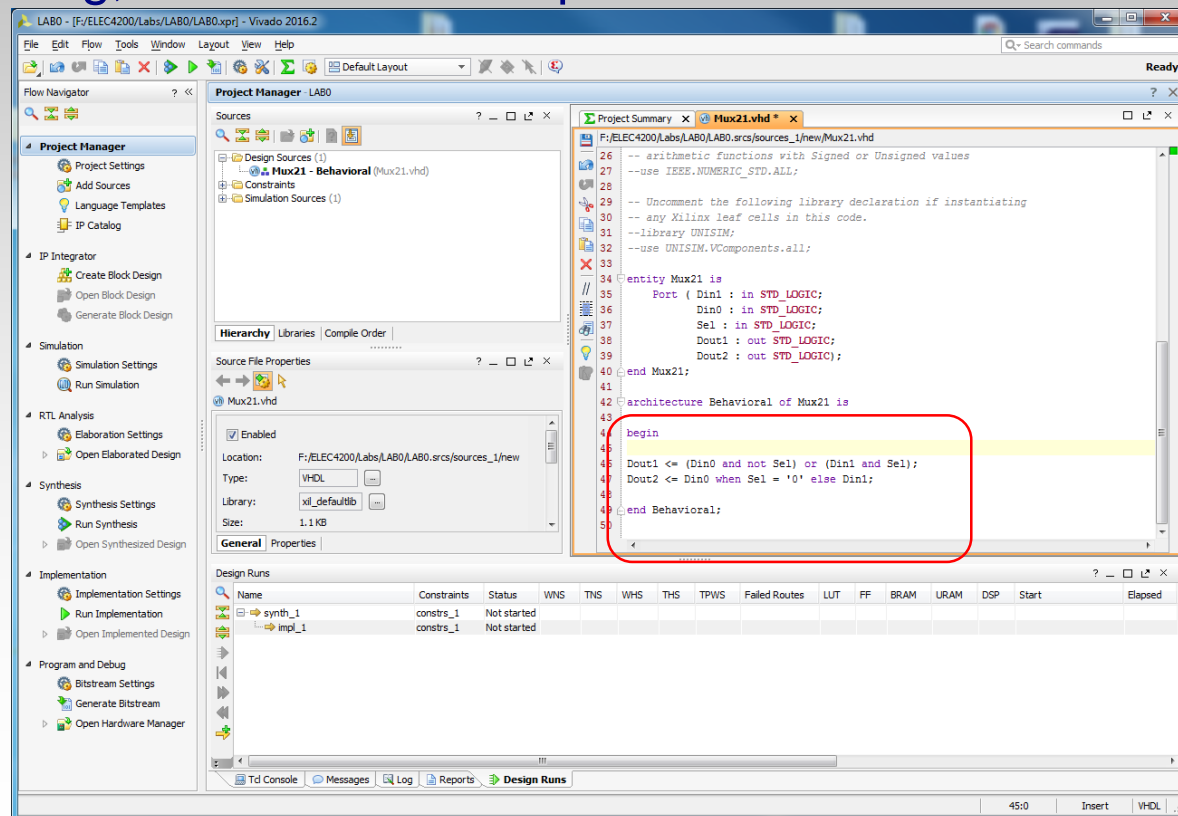
- Open your new VHDL file from the Sources window (double click on the Design Source name, *Mux21*)

The screenshot displays the Vivado 2016.2 IDE interface for a project named 'LAB0'. The 'Project Manager' window is active, showing a tree view of 'Design Sources (1)' with 'Mux21 - Behavioral (Mux21.vhd)' selected. Below it, the 'Source File Properties' window shows the file is enabled and located at 'F:/ELEC4200/Labs/LAB0/srcs/sources_1/new'. The 'Project Summary' window is also open, displaying project settings such as 'Project name: LAB0', 'Project location: F:/ELEC4200/Labs/LAB0', 'Product family: Artix-7', 'Project part: xc7a100tcsq324-1', 'Top module name: Mux21', 'Target language: VHDL', and 'Simulator language: Mixed'. The 'Design Runs' window at the bottom shows a table of synthesis and implementation runs.

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

Edit VHDL Model(2)

- Scroll down and add the following two lines of code between the Begin and End statements of the architecture section of the model.
 - $Dout1 \leftarrow (Din0 \text{ and not } Sel) \text{ or } (Din1 \text{ and } Sel);$
 - $Dout2 \leftarrow Din0 \text{ when } Sel = '0' \text{ else } Din1;$
- After saving, we can then set up the simulator.



Mux models:

1st statement:
logic equation

2nd statement:
“behavior”

Aldec Active-HDL Setup(1)

- Aldec *Active-HDL* is a commercially available modeling and simulation tool used to verify designs before synthesizing the designs into actual hardware. Xilinx *Vivado* integrates support for *Active-HDL*, as well as several other commercial simulation tools and its own simulator. Therefore, you must select *Active-HDL* from a list of simulators.
- As long as you use the same computer week to week, this setting should only have to be made this one time. However, if you ever get errors from *Vivado* about not being able to find the simulator, check this setting first.

Active-HDL Setup(2)

1. In the Project Manager pane, select *Project Settings*
2. In the Project Settings window, select *Simulation*
3. Select *Active-HDL Simulator* in the Target simulator drop-down menu.
4. Click OK to close the window.

The screenshot shows the Project Manager pane on the left with 'Project Settings' selected. The Project Settings dialog is open, showing the Simulation tab. The Target simulator is set to Active-HDL Simulator. The OK button is highlighted.

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

Active-HDL Simulation(1)

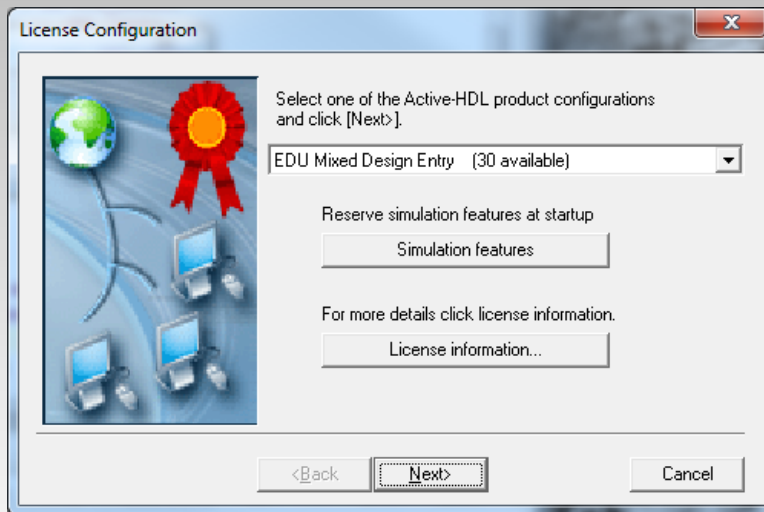
- Aldec HDL is packed full of features that help with development, simulation and debugging of FPGA designs. This tutorial describes only how to perform a basic simulation, but its is recommended that you experiment with its many capabilities, as you may find ways it can help you debug in later lab sessions.
- Click on “Run Simulation” in the Flow Navigator, and then Run “Behavioral Simulation” in the popup menu.

The screenshot displays the Aldec Active-HDL software interface. On the left, the 'Flow Navigator' pane shows the 'Simulation' section highlighted with a red box, containing 'Simulation Settings' and 'Run Simulation'. The 'Project Manager' pane shows the project 'LAB0' with 'Mux21 - Behavioral (Mux21.vhd)' selected under 'Design Sources (1)'. The 'Hierarchy' pane shows the project structure. The 'Constraint Set Properties' pane shows the default directory and other settings. The main editor window displays the VHDL code for 'Mux21.vhd', which defines a 2-to-1 multiplexer entity with two data inputs (Din0, Din1), one select input (Sel), and two data outputs (Dout1, Dout2). The behavioral architecture is implemented using conditional assignment.

```
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if you
30 -- use any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity Mux21 is
35     Port ( Din1 : in STD_LOGIC;
36           Din0 : in STD_LOGIC;
37           Sel : in STD_LOGIC;
38           Dout1 : out STD_LOGIC;
39           Dout2 : out STD_LOGIC);
40 end Mux21;
41
42 architecture Behavioral of Mux21 is
43
44 begin
45     Dout1 <= (Din0 and not Sel) or (Din1 and Sel);
46     Dout2 <= Din0 when Sel = '0' else Din1;
47
48
49 end Behavioral;
```

Active-HDL Simulation(2)

- Active-HDL will run twice
 - It will open to compile the VHDL model(s) and then close.
 - If there are no compilation errors, it will open again to simulate the model and remain open.
 - Each time you must click “Next” in the License Configuration window.



- Click OK on the “Simulation had finished” information window.
 - Note that we have not yet provided any signal inputs, so this “simulation” did not provide any useful information.
 - We will set up the desired simulation in the next steps.

Active-HDL Simulation(3)

- When the simulator is finished launching you should see the following window. If you do not, call the GTA to show you to set up the proper view windows.

The screenshot displays the Active-HDL 10.3 simulation environment. The Design Browser on the left shows a hierarchy for 'mux21 (behavioral)' with components like 'std.standard', 'std.TEXTIO', and 'ieee.std_logic_1164'. A table in the center lists signal names and their values:

Signal name	Value
Din0	U
Din1	U
Dout1	U
Dout2	U
Sel	U

A red box highlights this table, and a red arrow points to it from the text below. The console window at the bottom shows the following output:

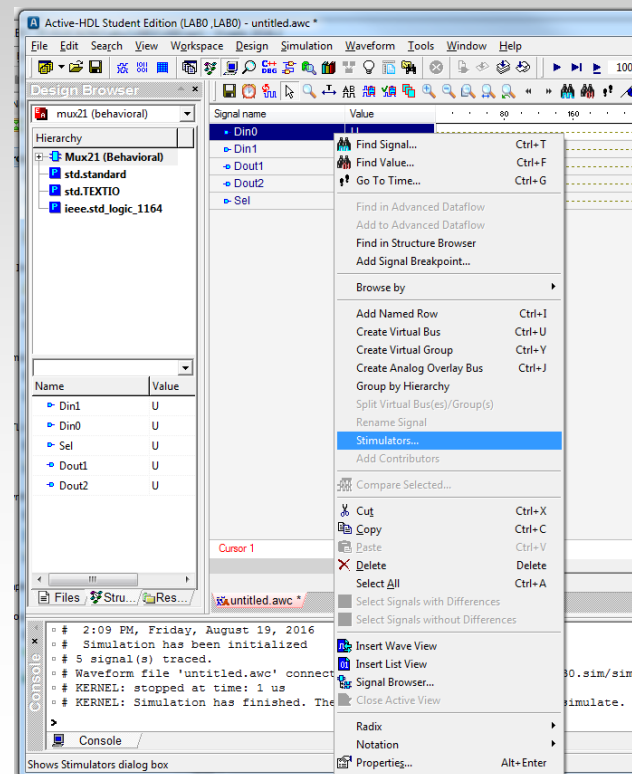
```
o # KERNEL: Kernel process initialization done.
o # Allocation: Simulator allocated 7079 kB (elbread=1280 elab2=5650 kernel=149 sdf=0)
o # KERNEL: ASDB file was created in location F:\ELEC4200\Labs\Lab0.sim\sim_1\behav\Lab0\src\wave.asdb
o # 9:50 AM, Friday, August 19, 2016
o # Simulation has been initialized
o # 5 signal(s) traced.
o # Waveform file 'untitled.awc' connected to 'F:\ELEC4200\Labs\Lab0.sim\sim_1\behav\Lab0\src\wave.asdb'.
o # KERNEL: stopped at time: 1 us
```

You should see your signal names. If there are any signals whose names are of the form XLXN, these are internal nets that can be deleted, unless you need them for debugging. Omit them from simulation screenshots captured for reports.

Active-HDL Simulation(4)

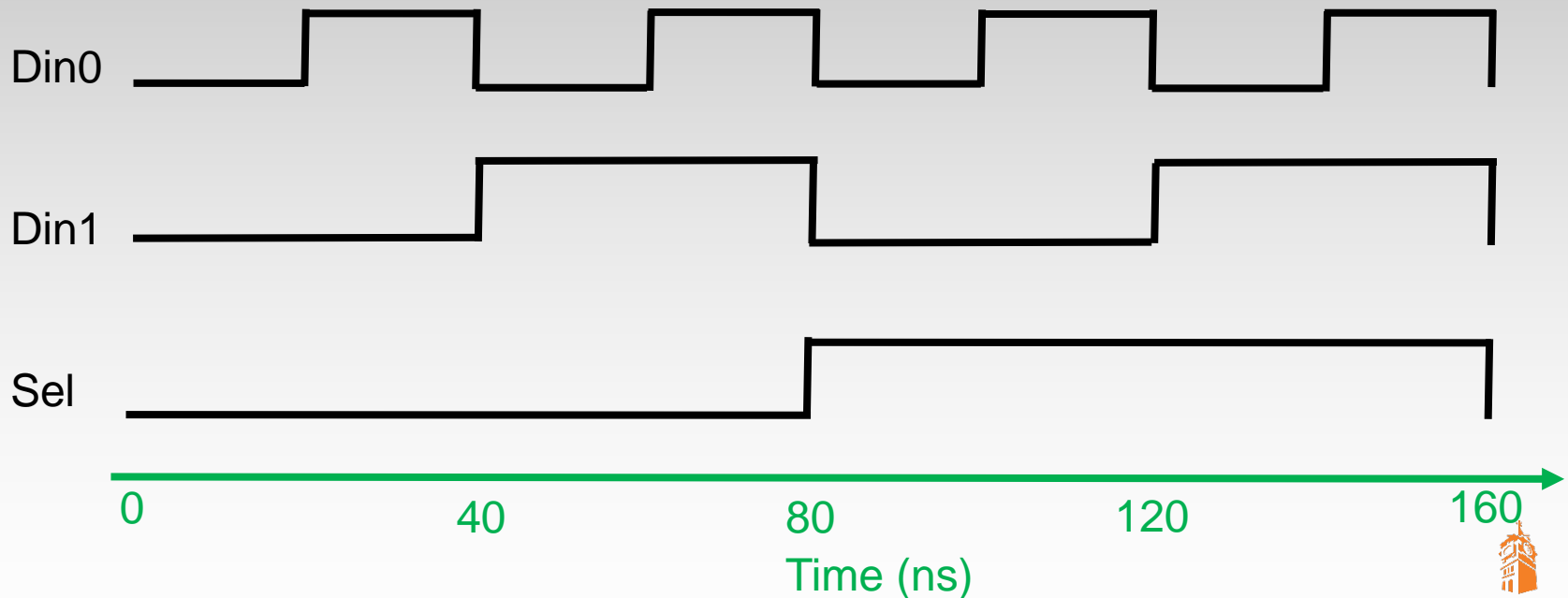
- Design verification requires that you stimulate the inputs and observe the outputs. You are to stimulate the inputs with all possible input combinations and observe each output to verify its correctness.
- This is called “exhaustive testing”. It is suitable for a simple circuit (such as the mux) but is not practical for large circuits with many inputs.

- Right click on one of your signals (ex: Din0) and select “Stimulators...” from the context menu.
- You can also shift/control click to select them all at once.



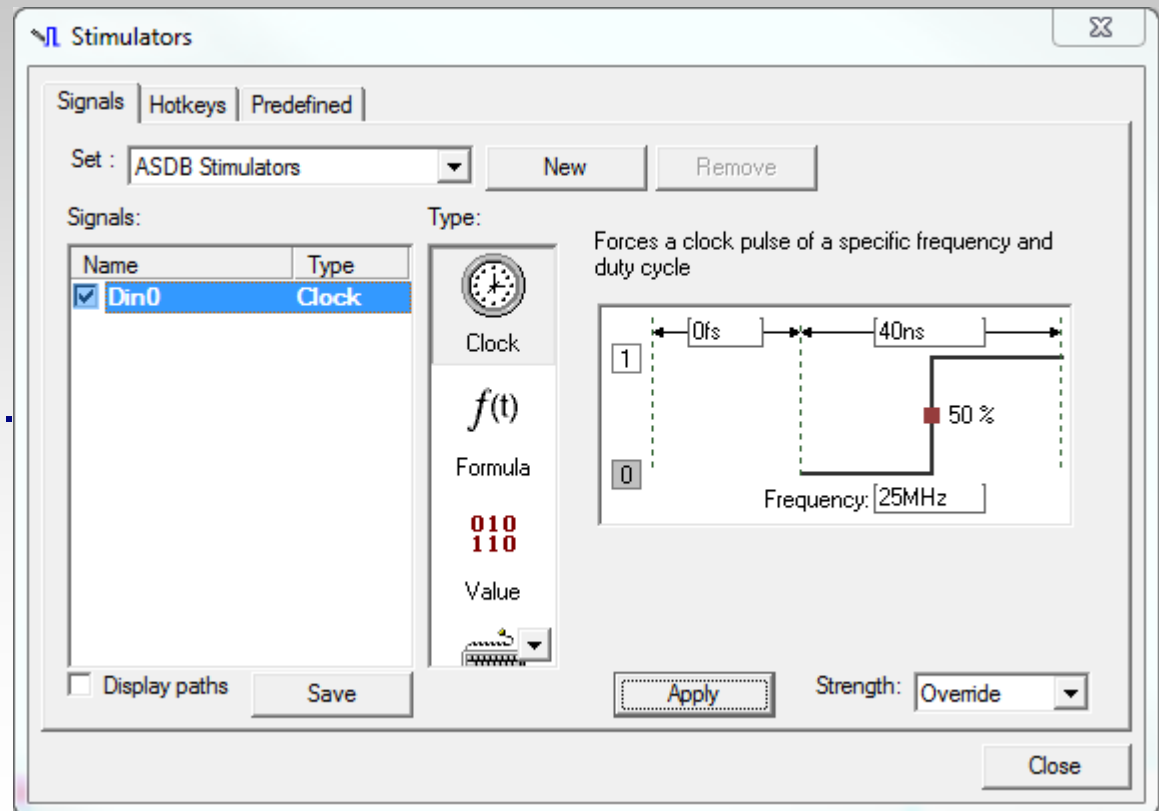
Active-HDL Simulation(5)

- For the 2-1 MUX we have 3 inputs (Din0, Din1, and Sel) where each input can be either logic high (1) or logic low (0).
- There are $2^3=8$ possible input combinations. While we could stimulate each combination individually, an easier approach would be to use *Active-HDL's* clock stimulator to generate the input timing diagrams below.



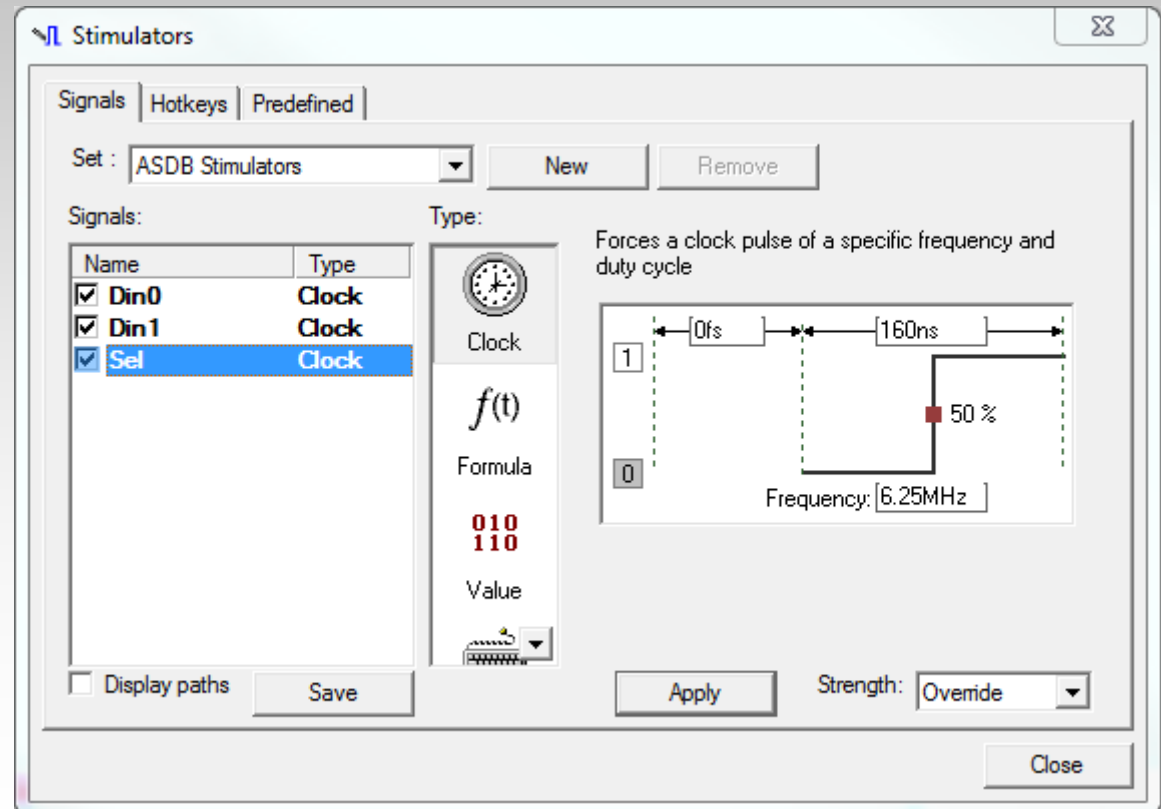
Active-HDL Simulation(6)

- Select your signal and click the “Clock” type. The right side of the Stimulators dialog box should now show the clock parameters.
- You can set the starting value (0 or 1), adjust the offset value to start the clock, adjust the period, and adjust the duty cycle
- Set the options as shown on the right to create a waveform that starts low and repeats every 40ns with a 50% duty cycle.
- Click “Apply”



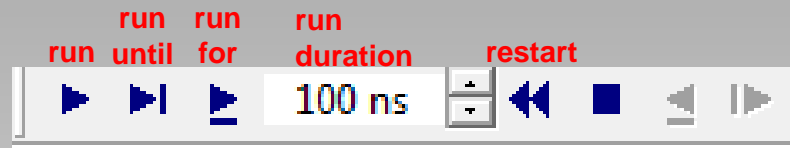
Active-HDL Simulation(6)

- Move the Stimulators dialog box out of the way so you can see the signals in the Waveform Window. Click on a signal in the Waveform Window to add it to the Stimulators dialog box.
- Repeat the clock procedure for Din1 and Sel with the following periods. Click “Apply” after each signal is added.
 - Din1: 80ns period
 - Sel: 160ns period
- Click “Close” when you are finished.

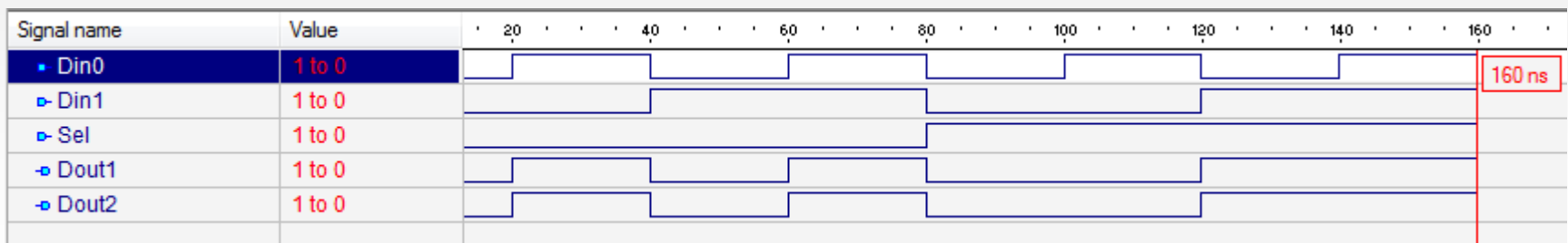


Active-HDL Simulation(7)

- Now we need to run the simulation using the following simulation controls.



- Whenever you begin a new simulation you should clear the waveforms window. Do this by clicking the “Restart Simulation” button or by typing “restart” into the Console Window.
- Run the simulation for 160ns by typing 160 ns into the “Run Duration” box and then click “Run For”. Your output should look similar to this. Use the zoom controls as needed to make the waveform fill the window.



Active-HDL Simulation(8)

- Examine your results and ensure that they accurately match the operation of a 2-1 MUX for both Dout1 and Dout2.
- If you observe any incorrect results during simulation, go back and debug your circuit. Do not proceed any further until your simulation produces the correct results.
- If time allows you may want to experiment with some of the other “types” in the Stimulators window so that you can become familiar with their operation and know how to use them for future labs.
- If your final circuit does not work and it is traced it back to an error in simulation, the GTA reserves the right to laugh at you publicly in front of your classmates.

Add Constraints (1)

- We need to define a “constraints file” that defines which signals (Din, Dout, Sel) go to which pins on the FPGA.
- Right click on the “Constraints” folder in the project manager and select “Add Sources”.

The screenshot displays the Vivado 2016.2 IDE. The Project Manager on the left shows the 'Constraints' folder selected, with a context menu open and 'Add Sources...' highlighted. The main editor shows the VHDL code for a Mux21 entity. The code is as follows:

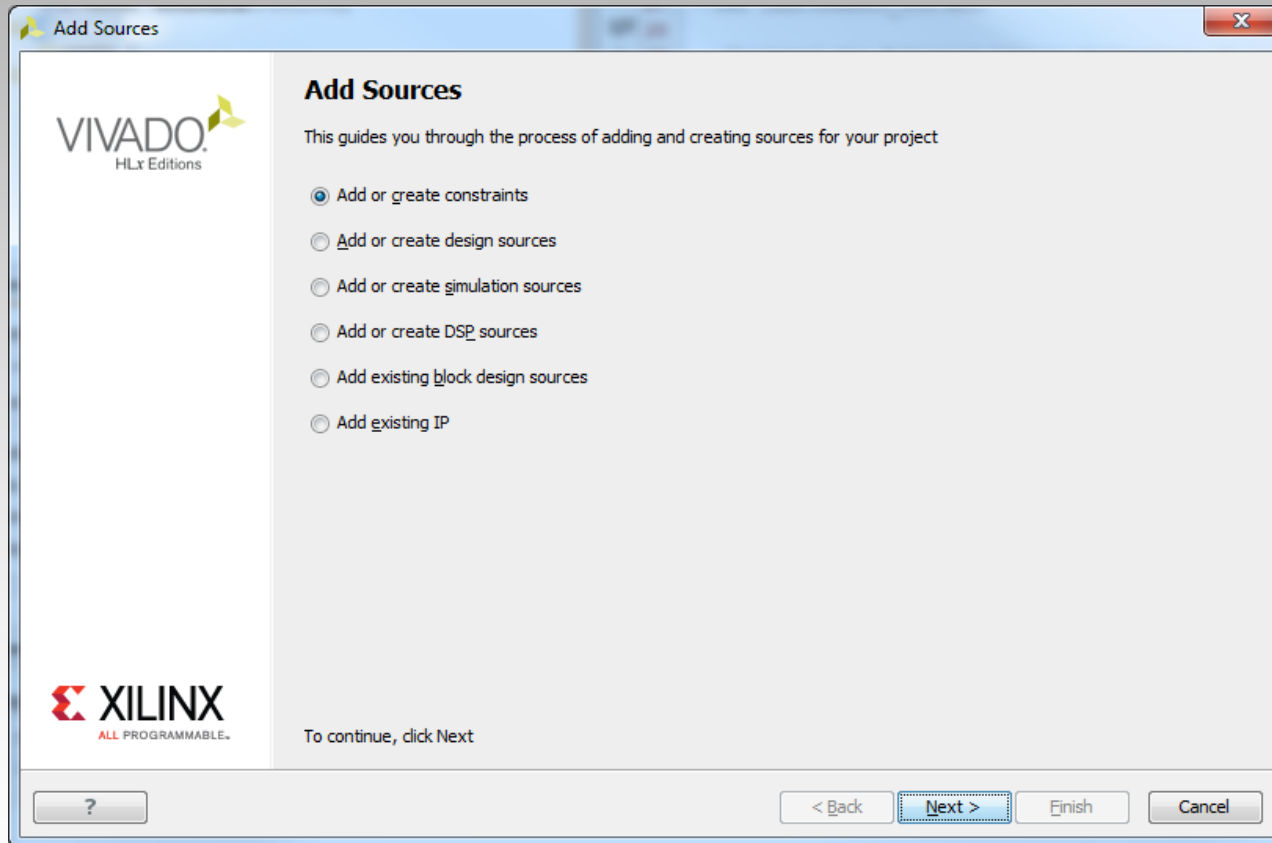
```
entity Mux21 is
    Port ( Din1 : in STD_LOGIC;
          Din0 : in STD_LOGIC;
          Sel : in STD_LOGIC;
          Dout1 : out STD_LOGIC;
          Dout2 : out STD_LOGIC);
end Mux21;

architecture Behavioral of Mux21 is
begin
    Dout1 <= (Din0 and not Sel) or (Din1 and Sel);
    Dout2 <= Din0 when Sel = '0' else Din1;
end Behavioral;
```

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

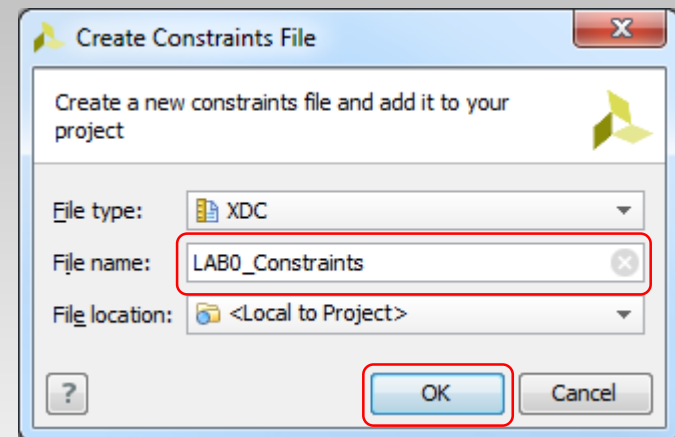
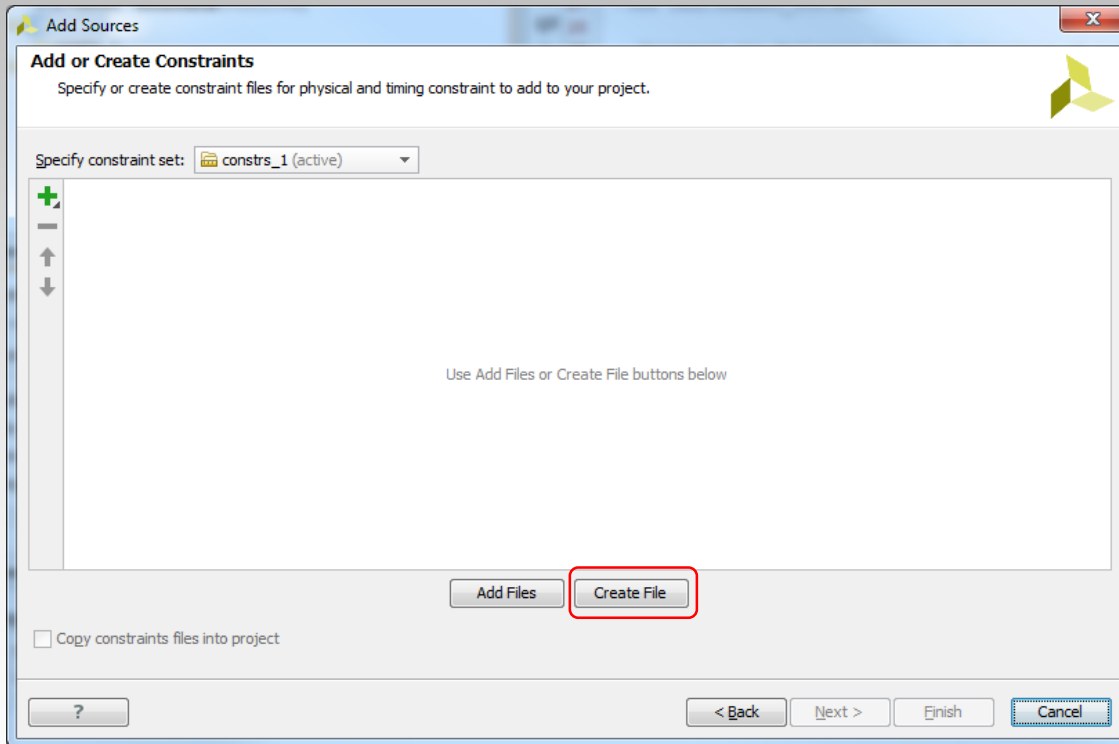
Add Constraints (2)

- This opens the “Add Sources Wizard”. Make sure “Add or create constraints” is selected and click “Next”.



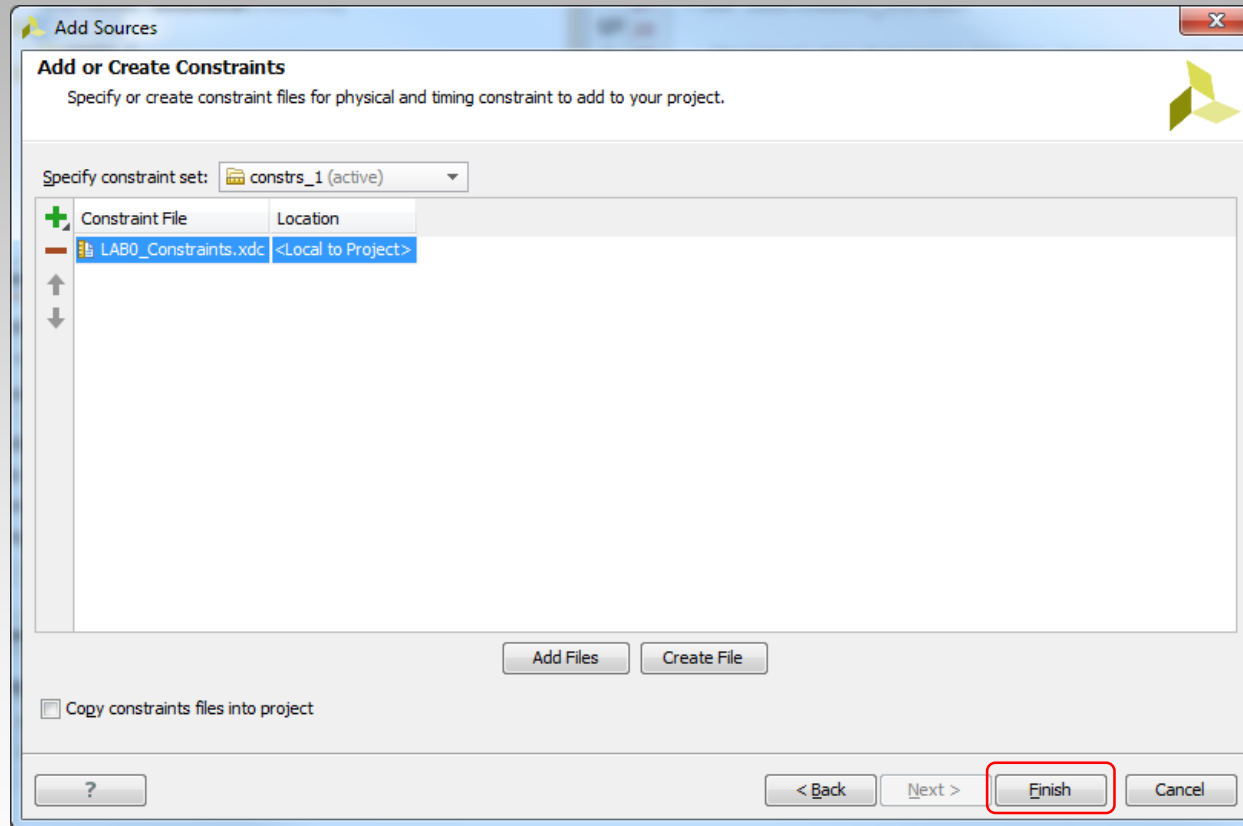
Add Constraints (3)

- Click “Create File” and in the dialog box that opens, name your constraints file. As always, use a descriptive name, like “LAB0_Constraints”. Then click “OK”.



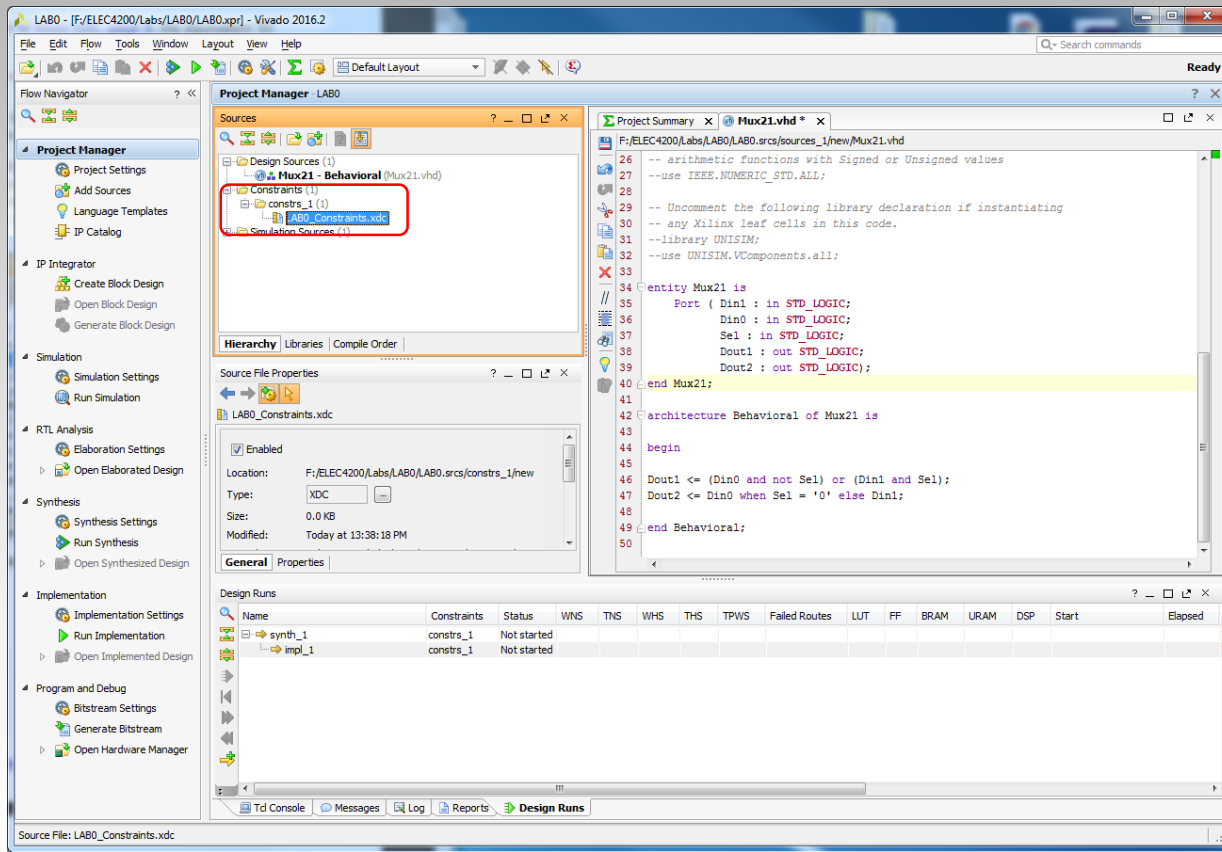
Add Constraints (4)

- Click “Finish” to add your file to the project.



Add Constraints (5)

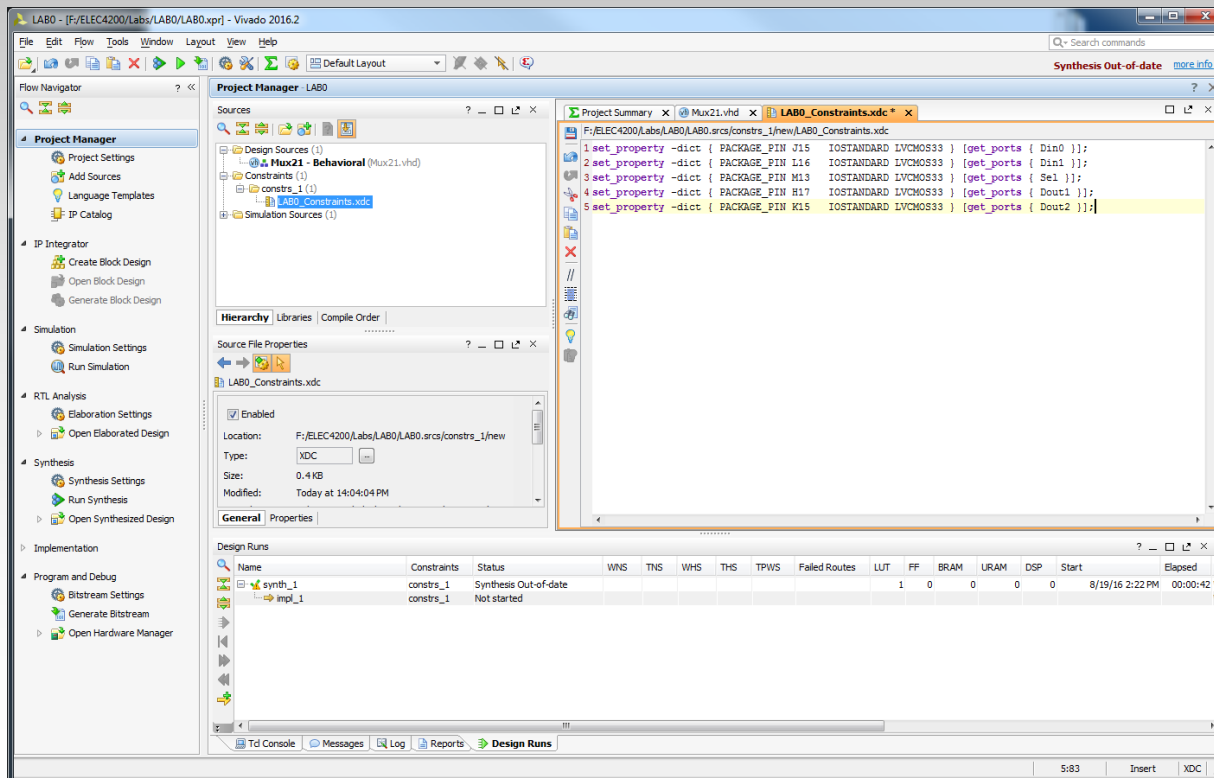
- Open your newly added file by double clicking on it from the Project Manager.



Add Constraints (6)

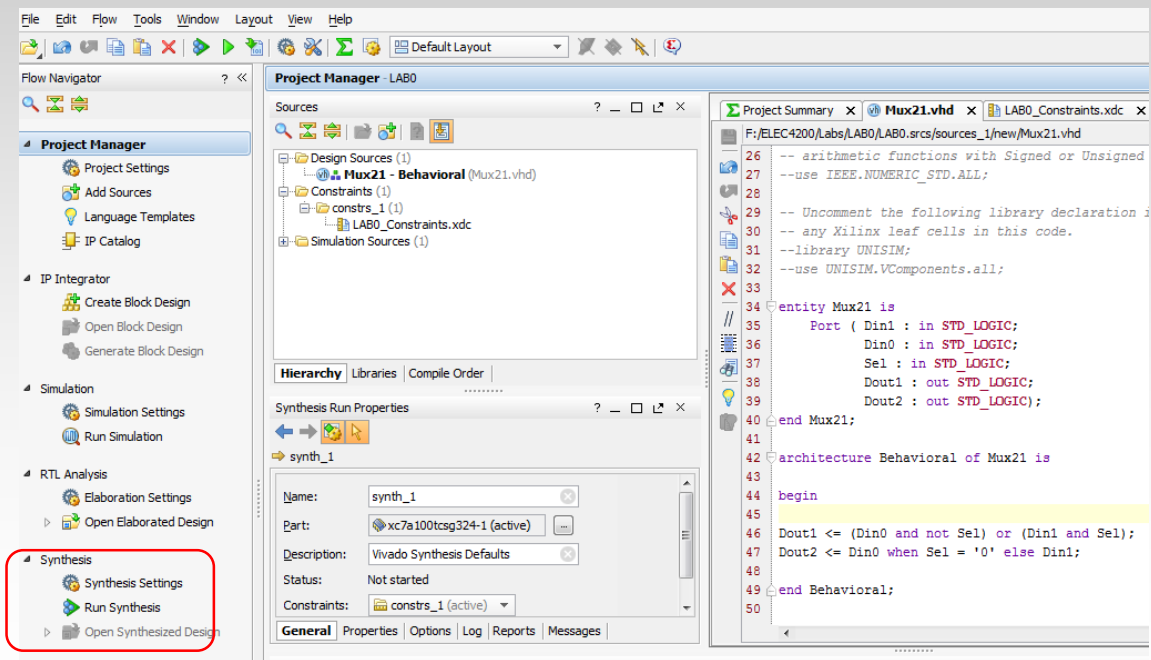
- Add the following lines in the constraints file, renaming the ports to match those in your design. Save after you are done.

```
set_property -dict { PACKAGE_PIN J15  IOSTANDARD LVCMOS33 } [get_ports { Din0 }];  
set_property -dict { PACKAGE_PIN L16  IOSTANDARD LVCMOS33 } [get_ports { Din1 }];  
set_property -dict { PACKAGE_PIN M13  IOSTANDARD LVCMOS33 } [get_ports { Sel }];  
set_property -dict { PACKAGE_PIN H17  IOSTANDARD LVCMOS33 } [get_ports { Dout1 }];  
set_property -dict { PACKAGE_PIN K15  IOSTANDARD LVCMOS33 } [get_ports { Dout2 }];
```



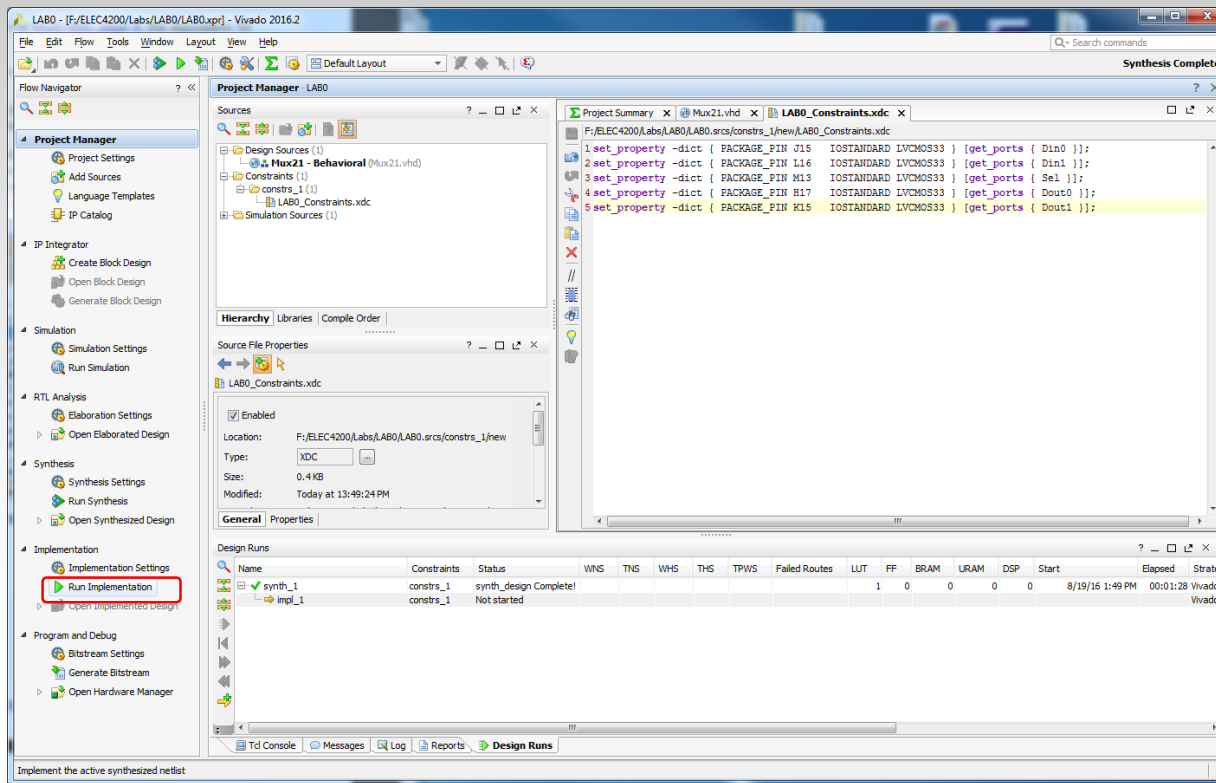
Implement the design(1)

- Now you are going to have *Vivado* “synthesize” your design into generic digital logic components.
- In the Flow Navigator, click “Run Synthesis”.
- This may take several minutes to run, so be patient.
- A dialog box will open on completion, either select “Run Implementation” or hit “Cancel”.



Implement the design (2)

- Click on “Implement Design”. This maps the synthesized design to the hardware, and routes I/O ports to the pins specified in the constraints file.
- Again, this will take several minutes to run.
- A dialog box will open on completion, either select “Generate Bitstream” or hit “Cancel”.



Generate Bitstream

- With the design now implemented, it is time to generate the bitstream, or the file that will be downloaded to the FPGA.
- Click “Generate Bitstream”
- A dialog box will open on completion, either select “Open Hardware Manager” or hit “Cancel”.

The screenshot displays the Vivado 2016.2 IDE interface. The top menu bar includes File, Edit, Flow, Tools, Window, Layout, View, and Help. The main workspace is divided into several panes:

- Flow Navigator:** Shows the project hierarchy with 'Mux21' selected.
- Netlist:** Displays the netlist for the selected object.
- Properties:** Shows properties for the selected object.
- Design Runs:** A table showing the status of various design runs.
- Implementation Complete:** A notification banner at the top right.

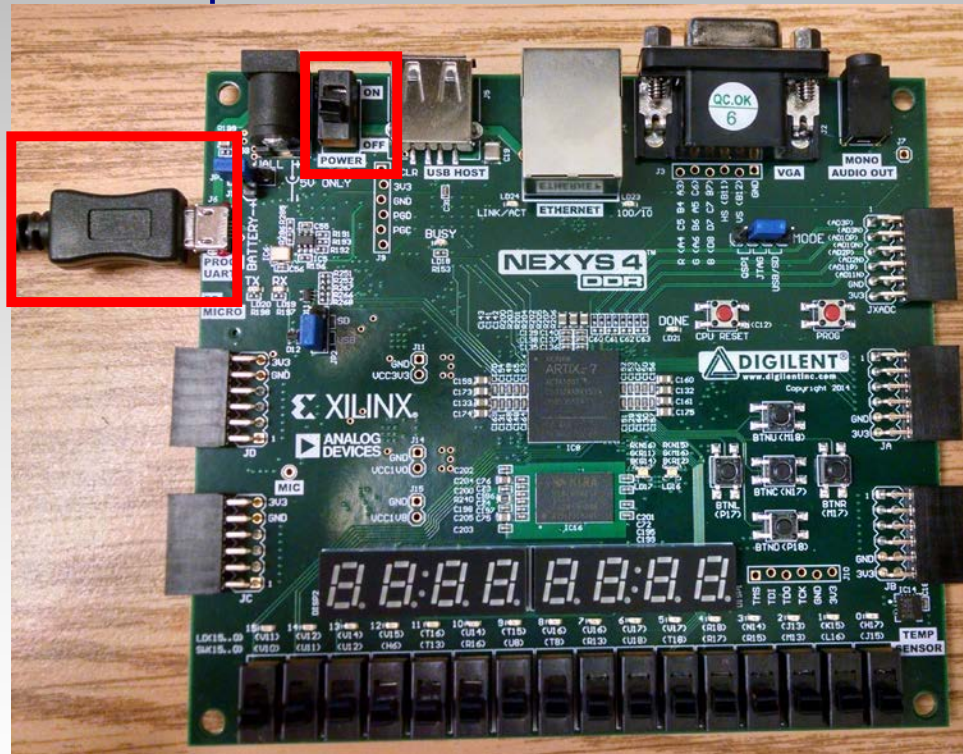
The **Design Runs** table is as follows:

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1	constrs_1	synth_design Complete!	NA	NA	NA	NA	NA		0	1	0	0	0	8/19/16 1:49 PM	00:01:28 V
impl_1	constrs_1	route_design Complete!								1	0	0	0	8/19/16 1:53 PM	00:01:36 V

The **Generate Bitstream** option in the Implementation pane is highlighted with a red box. The bottom status bar shows 'Generate a programming file after implementation'.

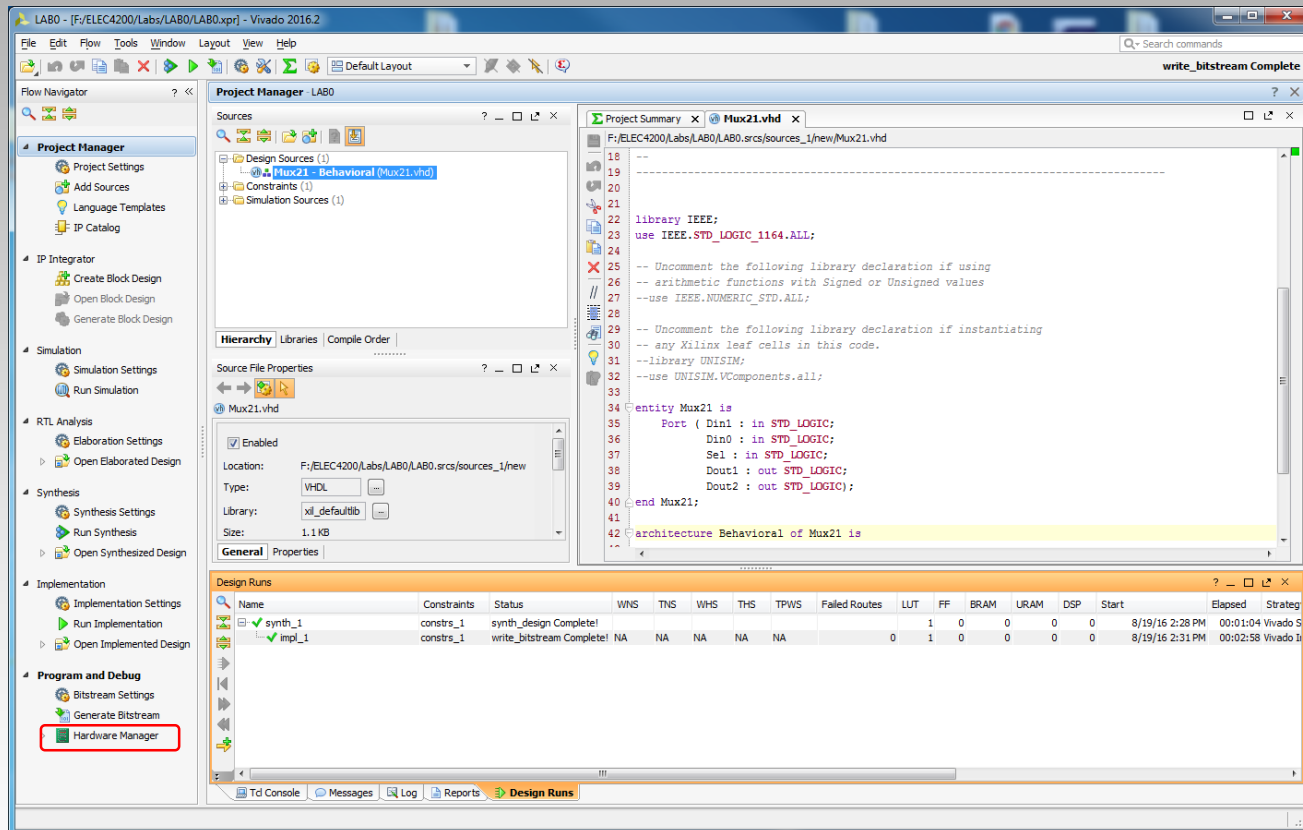
Download to Hardware(1)

- Plug in the USB cable between a PC USB port and the USB port on the FPGA board to access the JTAG programming module.
- Flip the board power switch from OFF to ON



Download to Hardware (2)

- Click on “Hardware Manager”



Download to Hardware (3)

- Click “Auto Connect” to connect to your Nexys 4 board.
- If you encounter any errors, verify that the board is plugged in and powered on.

The screenshot displays the Vivado 2016.2 IDE interface. The top menu bar includes File, Edit, Flow, Tools, Window, Layout, View, and Help. The main workspace is divided into several panes:

- Hardware Manager:** Shows "No hardware target is open. Open target" with a red box highlighting the "Auto Connect" button.
- Source File Properties:** Shows properties for the selected file "Mux21.vhd", including Location, Type (VHDL), Library (xl_defaultlib), and Size (1.1 KB).
- Tcl Console:** Displays the output of the "write_bitstream" command, including messages like "Finished scanning sources" and "open_project: Time (s): cpu = 00:00:12 ; elapsed = 00:00:34 . Memory (MB): peak = 699.844 ; gain = 109.680".

The right pane shows the VHDL code for "Mux21.vhd", which includes library declarations for IEEE and UNISIM, and the behavioral code for the Mux21 entity.

Download to Hardware(4)

- Click “Program Device” and select xc7a100t_0 from the list.

The screenshot displays the Vivado 2016.2 software interface. The main window is titled "LAB0 - [F:/ELEC4200/Labs/LAB0/LAB0.xpr] - Vivado 2016.2". The Hardware Manager window is open, showing a list of hardware components. The "xc7a100t_0 (1)" device is selected, and the "Program Device" button is highlighted with a red box. The Properties window below the Hardware Manager is empty, with the text "Select an object to see properties". The Tcl Console window at the bottom shows the following output:

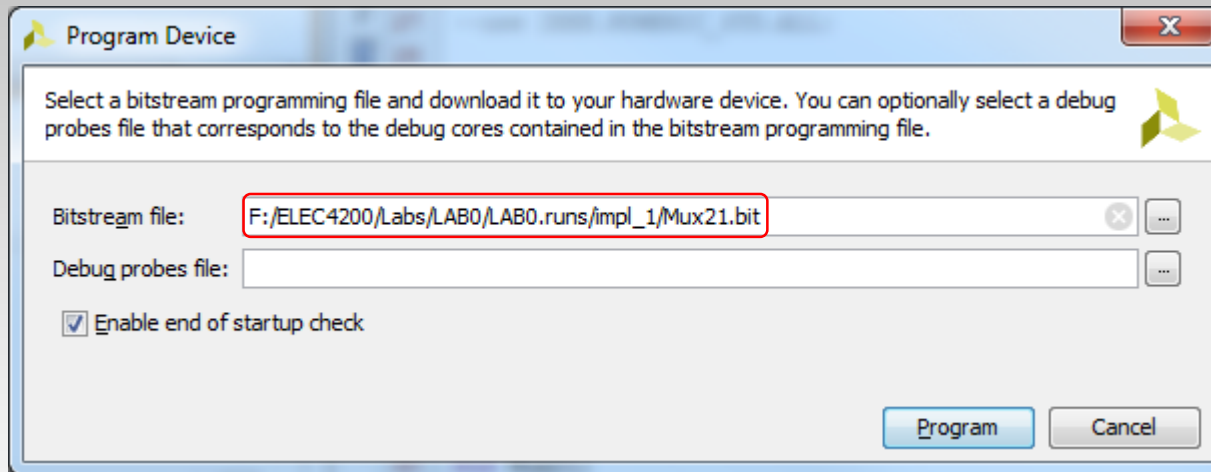
```
current_hw_device [lindex [get_hw_devices] 0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices] 0]
INFO: [Labtools 27-1434] Device xc7a100t (JTAG device index = 0) is programmed with a design that has no supported debug core(s) in it.
WARNING: [Labtools 27-3123] The debug hub core was not detected at User Scan Chain 1 or 3.
Resolution:
1. Make sure the clock connected to the debug hub (dbg_hub) core is a free running clock and is active OR
2. Manually launch hw_server with -e "set xsdb-user-bscan <C_USER_SCAN_CHAIN scan_chain_number>" to detect the debug hub at User Scan Chain of 2 or 4. To det
```

The code editor shows the following VHDL code for Mux21.vhd:

```
18 --
19 -----
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity Mux21 is
35     Port ( Din1 : in STD_LOGIC;
36           Din0 : in STD_LOGIC;
37           Sel : in STD_LOGIC;
38           Dout1 : out STD_LOGIC;
39           Dout2 : out STD_LOGIC);
40 end Mux21;
41
42 architecture Behavioral of Mux21 is
```


Download to Hardware(5)

- In the dialog box, make sure the correct bit file is selected.
- Leave the “Debug probes file” box empty for now.
- Click “Program”



Download to Hardware(6)

- Verify correct operation of the circuit using the switches and observe the output on the LEDs. If you encounter any bugs, attempt to figure out what went wrong before asking for help.
- Ensure that you apply all possible input combinations, as you did in simulation, and verify that the outputs match the simulation results.
- After verifying the circuit is correct, call the GTA over and demonstrate the circuit.

Clean up

- Turn off the board, unplug the USB cable, put them back in their box, and return the box to the shelf.
- Close Vivado, any other open programs, and save all files.
- Don't forget to log out of your machine and take any USB drives with you.