NATIONAL UNIVERSITY OF SINGAPORE

Department of Electrical and Computer Engineering

EE2026: DIGITAL DESIGN - LAB 1

Getting Started with Vivado 2016.2, Basys 3 Development Board, and Verilog HDL

AY 2017/2018, Semester 2

Lab Assistants

Christopher M. Shin (Monday and Tuesday B.Eng Sessions)
Gao Jieyi (Wednesday and Friday B.Eng Sessions)

IMPORTANT NOTES TO REMEMBER THROUGHOUT THE SEMESTER

- Please use the **D:\MyWork** folder for your work in the Digital Electronics Lab [E4-03-07]. You are encouraged to **delete** all folders within the **D:\MyWork** folder before starting your work. You are further encouraged to **copy** your work folder to your personal flash drive after the session is over.
- Please **delete** your work folder from the laboratory's computers after your session is over. You are responsible to **safeguard** your confidential programs. For assessable programs, you will be penalised if two programs with similarities beyond empirical evidence is detected. Both the source(s) and recipient(s) of plagiarised programs are equally penalised.
- For a more pleasant lab experience, you are **encouraged to bring your own laptop** as auxiliary for internet resources.
- You are free to use your own laptop to work on the lab and project contents in the Digital Electronics Lab.
- Do not work with your work folder and files on your thumb drive in Vivado, as this may significantly reduce your productive time. Work folders and files should preferably be on solid state disks or hard disks for faster accesses and processing.

OVERVIEW

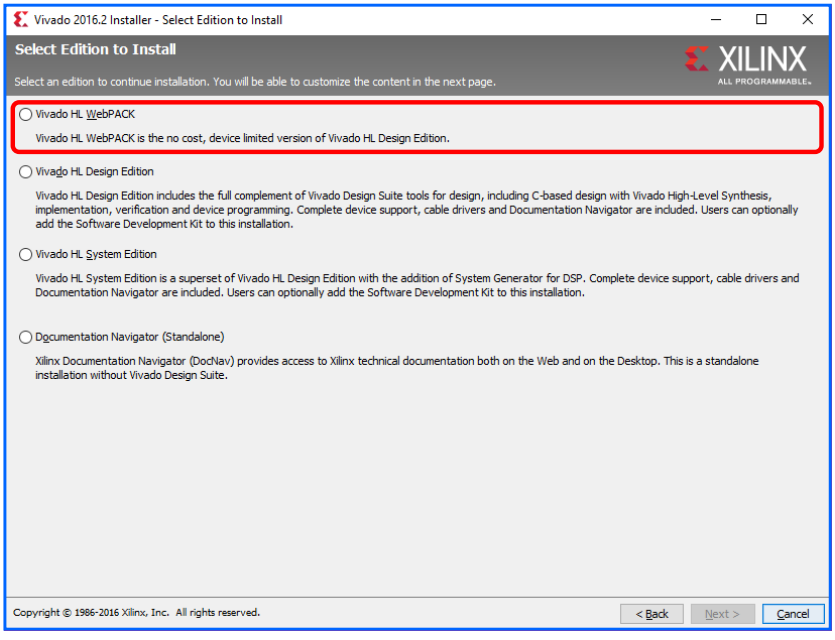
Using a simple Boolean design problem, an introductory approach to the Vivado software used in EE2026 will be covered. Quick instructions on downloading and installing the Vivado software on your personal computer are provided. The Vivado software is provided by Xilinx, which is an industry leader of FPGA. It is a comprehensive integrated development environment (IDE) for FPGA design flow.

In this lab:

- An introduction to basic Verilog HDL (Hardware Description Language) is provided.
- The overall process flow to of designing, synthesising, simulating and implementing a program is covered.
- Programming Digilent's Basys 3 development board, which features an FPGA from Xilinx's state-of-the-art Artix-7 family, is illustrated.
- Your familiarity with the IDE and Basys 3 development board is verified through the implementation of a multiplexer circuit.

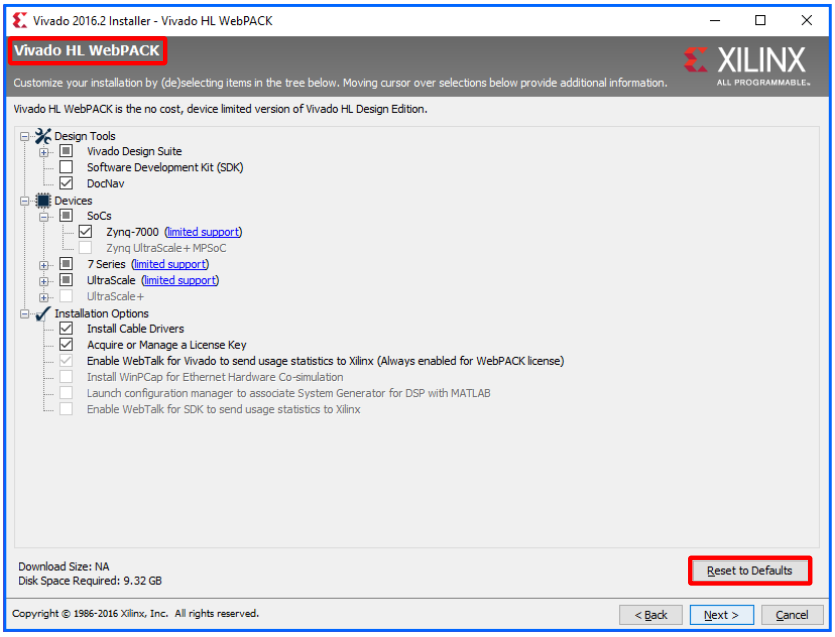
1.0 VIVADO DOWNLOAD AND INSTALLATION

The Vivado 2016.2 software is already installed on the computers in the Digital Electronics Lab, and are ready for immediate usage. It is also required that you install such software on your own personal computer. Some quick guidelines on installing the required software for EE2026 on your personal computer is provided in this section.

Software Name	Vivado Design Suite - HLx Editions - 2016.2 **Warning: Do not use** other versions of the software. Only the Vivado 2016.2 Windows version has been tested. Computer compatibility issues will occur with other versions of the software, and assessment of your project may not be possible. This will lead to loss of project marks if your project cannot be assessed.
Software Weblink	https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html
OS Requirement	Windows 7 and above. Mac version is not supported
OS Architecture	64 bit versions. 32 bit versions are not supported
Quick Download and Installation Guide	Select the installer **[Vivado HLx 2016.2]** that best suits your preferences. All downloads require registration with Xilinx, and you will be prompted for registration during the download: Vivado HLx 2016.2: Windows Web Installer (EXE - 50.41 MB) MD5 SUM Value: 31edee9382d432aca96a9fc80befdd40 ◀ Installer download takes multiple seconds. Registration is required for installation. Download time during installation may be short, but loss of internet connection may result in installation failure. Use this if you want to minimise the total time from download to installation, as compared to downloading a full installer. Vivado HLx 2016.2: Linux Web Installer (BIN - 80.67 MB) MD5 SUM Value: d5607db40f368915052e885a9470d752 ◀ Linux version has not been tested in EE2026. Do not use. Vivado HLx 2016.2: All OS Installer Single-File Download (TAR/GZIP - 11.17 GB) MD5 SUM Value: 0e41f991e5d89410ad5ed6d30407f379 ◀ Installer download takes time, or can be obtained from alternate sources. Registration is not required for installation. There is no download during installation and it takes a few minutes to install, while being the most reliable method for installation After obtaining the installer file, click on the setup executable of the software. Along the installation process, you may wish to take note of the following: ▶ Select the **Vivado HL WebPACK** option when given the choice of installation edition. 

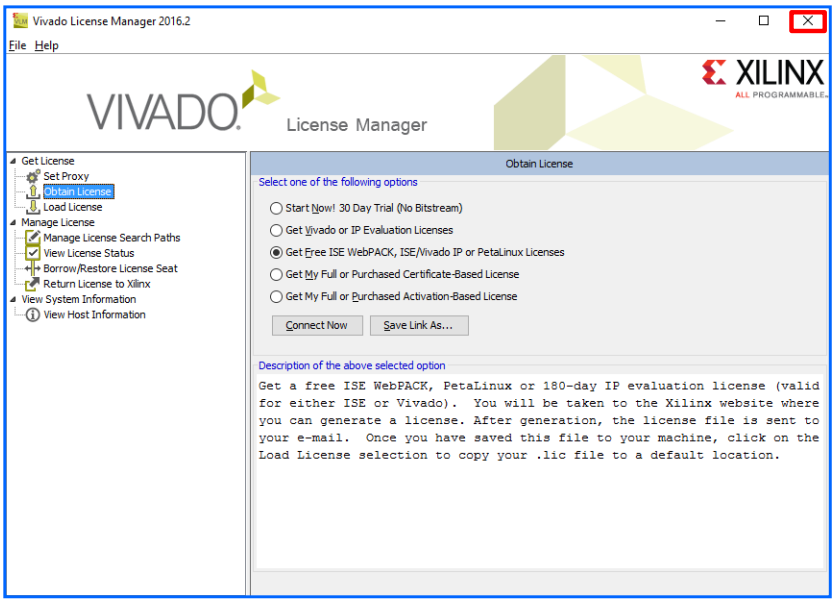
Quick Download and Installation Guide

▶ Select the default options during installation. There is no need to change the installation options.



▶ Your firewall software may prompt you to allow the installation of certain devices. The verified publishers should be allowed access.

▶ After the installation is complete, a license window will appear. You can safely close the window without selecting any license.



Please restart your computer before using the Vivado 2016.2 software. You may wish to uninstall the **Xilinx Information Centre** from the control panel as it is not needed. Not uninstalling it will cause pop-up messages to appear whenever updates for Xilinx are available. You are now ready to use the Vivado 2016.2 IDE for EE2026 purposes.

2.0 DESCRIPTION OF THE SIMPLE BOOLEAN DESIGN TASK

The following task is required to be implemented on the Basys 3 development board:

- When switch **A** turns on, only **LED1** lights up.
- When switch **B** turns on, only **LED2** lights up.
- When both switches **A** and **B** turn on, **LED1**, **LED2**, and **LED3** light up.

UNDERSTANDING | TASK 1

Complete the truth table for the design task:

INPUT		OUTPUT			MINTERM
A	B	LED1	LED2	LED3	
0	0				$\bar{A}\bar{B}$
0	1				$\bar{A}B$
1	0				$A\bar{B}$
1	1				AB

2.1 Deriving an SOP Boolean Equation for the Design Task

Given any truth table with any number of input variables, the sum-of-products (SOP) or product-of-sums (POS) canonical form may be used to write out a Boolean equation for each output variable. Let us use the SOP form as it leads to a shorter equation, albeit not necessarily the simplest.

A minterm, which is a product involving all input variables, that is true for each row of the table has been provided for the truth table. Computing a minterm does not require any consideration of the output. The canonical SOP Boolean equation for an output variable can then be worked out, by summing up each minterm for which the output is TRUE. For example, the canonical SOP form for **LED1** is written by summing up each minterm for which the output **LED1** is TRUE, as indicated below:

$$\mathbf{LED1} = A\bar{B} + AB$$

UNDERSTANDING | TASK 2

Work out the canonical SOP Boolean equations for **LED2** and **LED3** based on your truth table from **UNDERSTANDING | TASK 1**

$$\mathbf{LED2} =$$

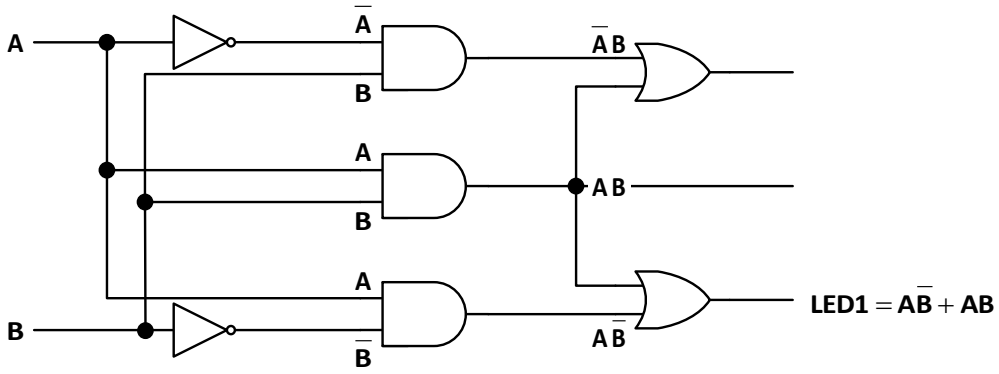
$$\mathbf{LED3} =$$

2.3 Illustrating Logic Expressions by Using a Schematic of Gates

The Boolean equations for **LED1**, **LED2**, and **LED3**, as obtained from section 2.1, can be implement by using:

- 2 NOT gates (NOT gates are also called inverters)
- 3 AND gates
- 2 OR gates

The gate-level schematic is as depicted below:



2.4 Verilog Hardware Description and FPGA Implementation

Xilinx's Vivado software is an integrated design environment that has numerous amount of advanced features used in the industry, and among which we will be introducing the following:

- Writing and editing HDL codes for digital system designs.
- Simulation of the design's behaviour.
- Synthesis of the codes, in order to convert the design from textual description into logic gates.
- Implementation of the design to map and route the logic to a target FPGA.
- Optimising the synthesis, implementation, and bitstream generation according to the user's strategies. The default optimisation strategies shall be used in EE2026, as changing them is beyond the scope of introductory digital designs.
- Programming an FPGA with the optimised bitstream.

The remaining part of this lab manual will now show the steps required to go from the design task, to the FPGA implementation. The Basys 3 development board shall be used, and it is expected that students train themselves to be able to easily go through all these steps before the EE2026 lab 2 starts.

3.0 INTRODUCTORY STEP-BY-STEP GUIDE TO XILINX'S VIVADO 2016.2 SOFTWARE

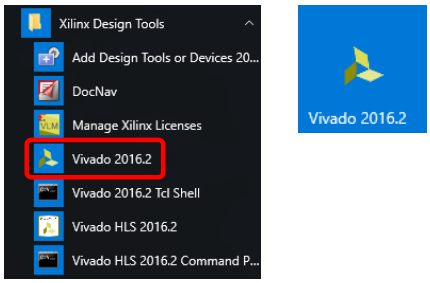
During your lab session, your EE2026 lab assistant may provide you helpful hints on the usage of the Vivado 2016.2 software, beyond the most basic things that are described in this section.

3.1 Creating a New Verilog Project in Vivado

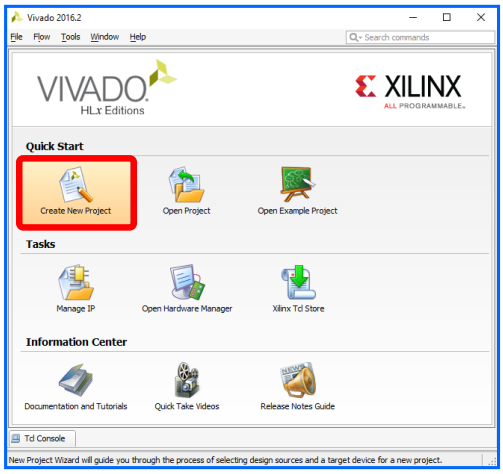
1. Open the executable: **Vivado 2016.2**

   You will need to wait multiple seconds before the program opens. Avoid clicking on the icon multiple times, as errors while using the program may occur if multiple copies of Vivado 2016.2 are opened at the same time.

   If you have used the default installation, and depending on your operating system, the icon may look similar to what is shown to the right.

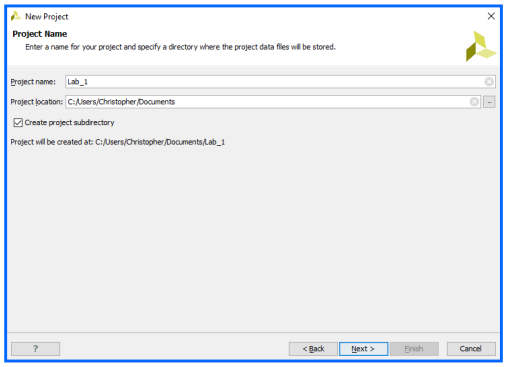


2. **Create New Project** and continue until the next step.



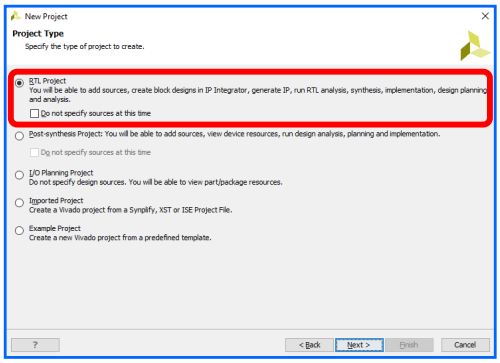
3. Enter a **Project name** and continue.

   Ensure that the complete **Project location** name for your project folder does not have any spaces, and that your **Project name** does not start with a number. Failure to follow these rules will prevent successful designs in Vivado



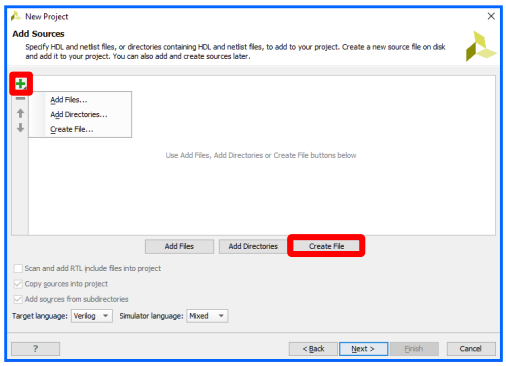
4. Select **RTL Project**.

   Uncheck "**Do not specify sources at this time**".



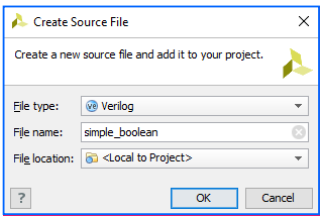
5. Click on **Create File**.

   Target language: **Verilog**
   Simulator language: **Mixed**



6. Enter a **File name** for the source file.

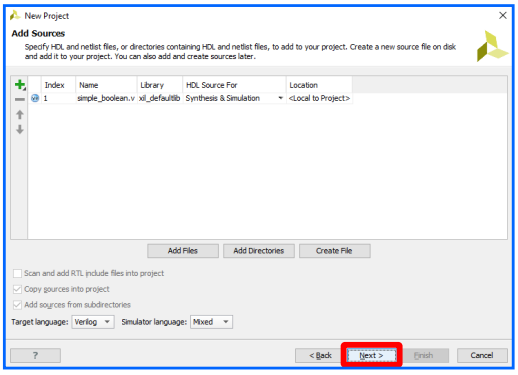
   Ensure that the **File name** does not have any spaces and does not start with a number.



7. In the **Add Sources** window, click on **Next** after the file has been added. This leads to the **Add Existing IP (optional)** window.

   In the **Add Existing IP (optional)** window, just click on **Next**. This leads to the **Add Constraints (optional)** window.

   In the **Add Constraints (optional)** window, just click on **Next**. This leads to the **Default Part** window.

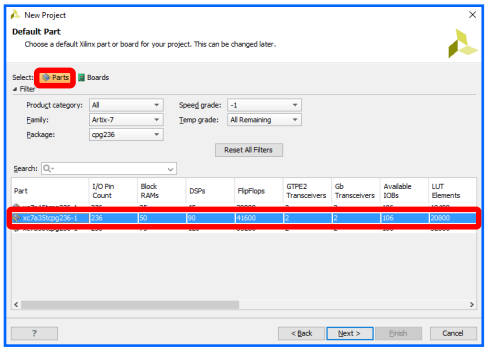


8. Select **Parts**.

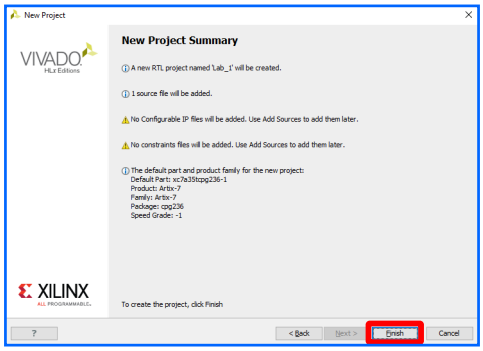
   The FPGA chip in use needs to be specified. The Basys 3 development board contains the **xc7a35t** chip from the Artix-7 family. Use the filter to fasten the search:

   Family: **Artix-7**
   Package: **cpg236**
   Speed grade: **-1**

   Select **xc7a35tcpg236-1** and click on **Next**.



9. Click **Finish**.

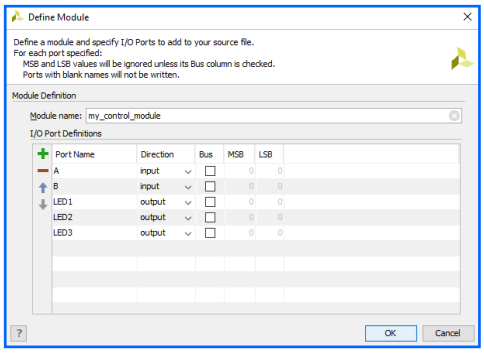


10. A module, that resides within the source file that had been created in the previous steps, need to be created. Change the **Module name** to "**my_control_module**".

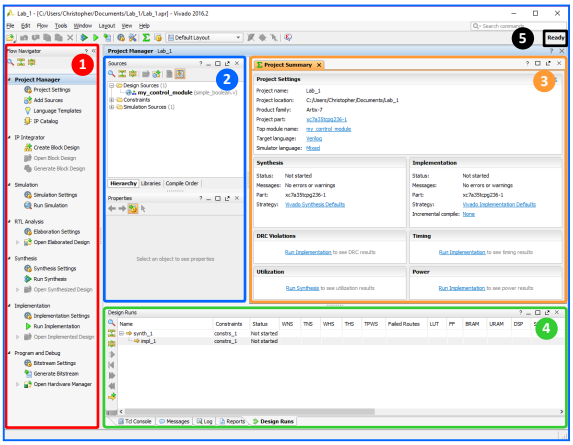
    What do you notice in the sources file hierarchy? (Refer to the next step, focusing on the **Data Windows Area**)

    Fill in the remaining cells with:

    Port Name: **A**, Direction: **input**
    Port Name: **B**, Direction: **input**
    Port Name: **LED1**, Direction: **output**
    Port Name: **LED2**, Direction: **output**
    Port Name: **LED3**, Direction: **output**



11. Upon creating a new project, the Vivado IDE viewing environment is shown. The main parts of the graphical user interface are:



1. The **Flow Navigator** provides quick access to tools from design entry to bitstream generation.
2. The **Data Windows Area** displays design sources and data. In the screenshot, the **Sources** window is currently selected.
3. The **Workspace** displays the text editor, schematic, project summary, and other report.
4. The **Results Window** displays messages and log files during simulation, synthesis, implementation, and so on.
5. The **Project Status Bar** displays the current status of the active design.

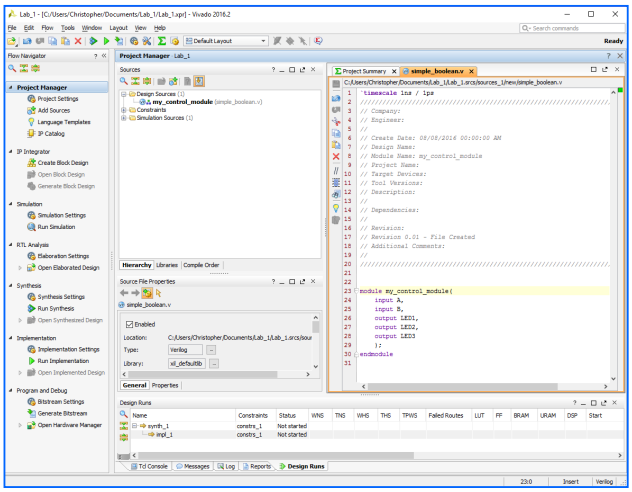
3.2 Using Vivado Text Editor to Write Verilog HDL Code

1. Double click on the design source, for example: **my_control_module**, to open and display the file in the **Workspace** for editing.

   In the text editor, it can be seen that a Verilog module begins with the module name, and the inputs and outputs to the design.

   Words that are highlighted in **purple** are reserved words that have specific functions, and should not be used as labels.

   Currently, the module is empty in terms of the types of outputs to produce when given certain inputs.



2. Some codes for the behaviour of the program need to be inserted, between **module** and **endmodule**. Using the Boolean expressions obtained in section **2.1**, the operators are translated to Verilog representations, as tabulated below:

Operators		Verilog Representation
OR	$A + B$	|
AND	AB	&
NOT	$\overline{A}$	~
XOR	$A \oplus B$	^

In the screenshot, the **assign** statement causes the left hand side of the expression to be updated *every* time there is a change on the right hand side of the expression. It is therefore called a *continuous assignment* statement, describing combinational logic whereby the output on the left is a function of current inputs on the right.

For the image shown on the right, it means that the statements on line 31 till line 33 execute concurrently. This is in contrast to sequential execution of statements in a computer programming language such as C.

```
23 module my_control_module(
24     input A,
25     input B,
26     output LED1,
27     output LED2,
28     output LED3
29     );
30
31     assign LED1 = (A & ~B) | (A & B);
32     // Delete this comment and write the Verilog code for LED2
33     // Delete this comment and write the Verilog code for LED3
34
35 endmodule
```

Code Templates

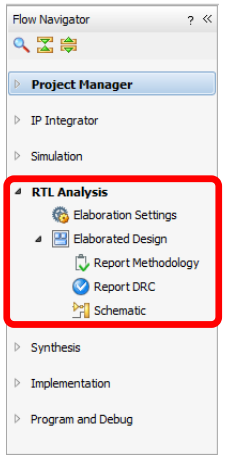
UNDERSTANDING | TASK 3

Complete the code for line 32 and line 33 based on your Boolean expressions from the section **2.1**

3. Save your current file by clicking on **File → Save File**, or by pressing **Ctrl+S**.

   Each time a file is saved, a syntax check is carried out. If there are any syntax errors in your code, they will be indicated in the **Messages** tab of the **Results Window**. Indications also appear in the **Sources** window of the **Data Windows Area**.

   After saving, perform the following:
   - Expand **RTL Analysis** in the **Flow Navigator Panel**
   - Expand **Elaborated Design**
   - Click on **Schematic**
   - If an **Elaborate Design** window pops up, click **OK**



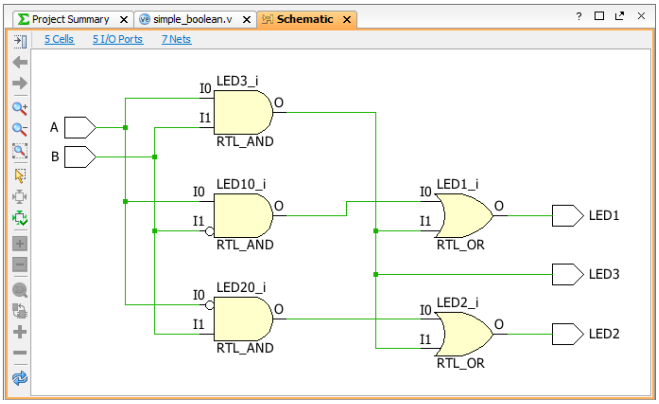
4. The schematics window will appear, showing the RTL schematic of the design.

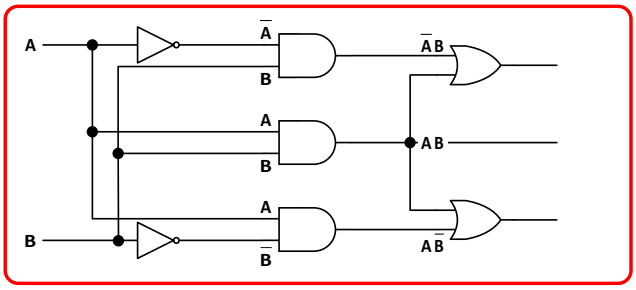
   Register Transfer Level (RTL) schematic is a pre-optimised design in terms of generic symbols, such as AND gates, OR gates.

   In other words, the RTL schematic allows one to view a schematic representation of the design, with symbols such as logic gates, adders, multipliers, counters providing a functional view of the design.

UNDERSTANDING | TASK 4

What similarities and differences do you notice between the RTL schematic and the schematic obtained from the previous section, both shown to the right. How do they compare to the actual schematic obtained on your computer screen?





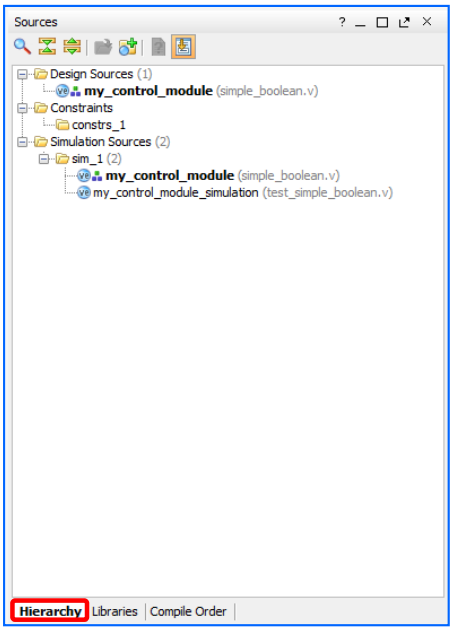
3.3 Testbench and Behavioural Simulation

1. After writing the codes, there is a need to test them to check their behaviours. Inputs are applied to a module, and the outputs are checked to verify whether the module operates as intended. This ensures that a system is tested before it is physically built.

   A testbench is an HDL module that is used to test another module, and which is called the Device Under Test (DUT) or Unit Under Test (UUT). The testbench contains statements to apply inputs to the DUT, and in our case, it is the **my_control_module.**

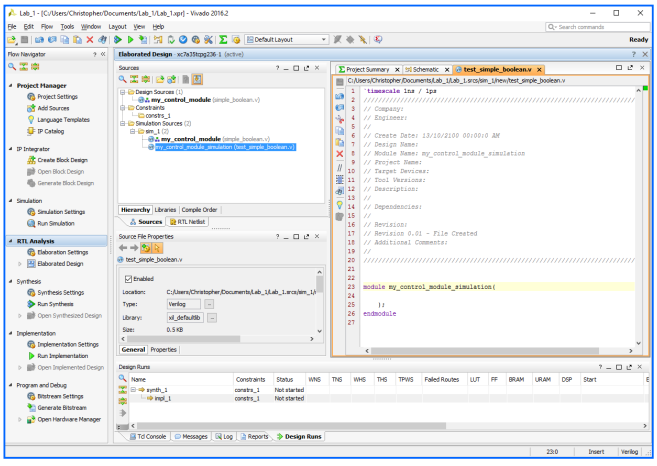
   As a start, perform the following:
   - Expand **Project Manager** in the **Flow Navigator Panel**
   - Click **Add Sources**
   - Select **Add or create simulation sources,** then **Next**
   - Similar to section **3.1**, create a Verilog source file, and give it a file name, such as **test_simple_boolean**
   - In the **Define Module** window, you may change the name to **my_control_module_simulation** if you wish
   - Do not input any **I/O Port Definition**, and click on **OK**
   - The **Sources** window will appear as depicted on the right

   You may click on the **Libraries** and **Compile Order** tab to see how the **Sources** window look like. This will help you in the future if ever the **Hierarchy** tab is not selected by default. Furthermore, understand the relationship between the different file names and module names that have been used here.



2. Double-click on the **my_control_module_simulation** from the **Sources** window to open the **Workspace** text editor for the **test_simple_boolean.v** file.

   The timescale directive at the top of the **test_simple_boolean.v** file specifies the time unit used in the file. **'timescale** 1ns / ps specifies a time unit in the file to be 1 ns, and the simulation to have a precision of 1 ps. Time unit delays are indicated by the symbol **#**



3. Add the codes as shown in the image on the right.

   An **initial** block is used to execute statements in its body once, starting from t=0. For this example, the following happens:

   - At time t = 0 ns, A = 0 and B = 0
   - At time t = 10 ns, A = 0 and B = 1
   - At time t = 20 ns, A = 1 and B = 0
   - At time t = 30 ns, A = 1 and B = 1

   An **initial** statement should only be used in testbenches for simulation, not in modules intended to be synthesised into actual hardware.

   With reference to the screenshot, all signals on the left of assignments must be declared as **reg**. Note that **reg** does not necessarily imply that the signal is a register.

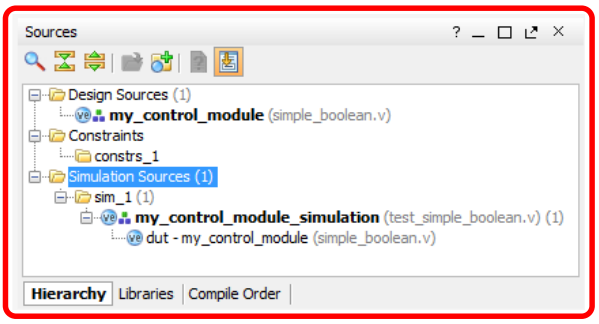
   The **LED1**, **LED2** and **LED3** outputs have been declared as **wire** here. **wire** is used when an output is expected to be used as input to another module, or in an assignment statement. In our case, since the outputs are not used by other modules, the simulation will succeed even without the **wire** declarations for **LED1**, **LED2** and **LED3**.

```
23 module my_control_module_simulation(
24
25     );
26
27     // Inputs
28     reg A;
29     reg B;
30
31     // Outputs
32     wire LED1;
33     wire LED2;
34     wire LED3;
35
36     // Instantiate Device under Test (DUT)
37     my_control_module dut(A, B, LED1, LED2, LED3);
38
39     //Stimuli
40     initial begin
41         A = 0; B = 0; #10;
42         A = 0; B = 1; #10;
43         A = 1; B = 0; #10;
44         A = 1; B = 1; #10;
45     end
46
47 endmodule
48
```

Code Templates

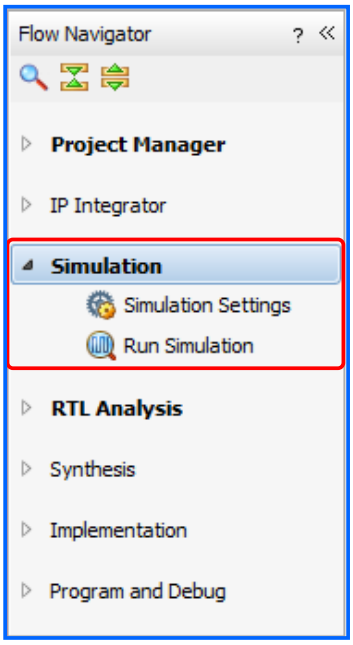
4. After saving the file, note how the **Sources** window has changed. Testbench **my_control_module_simulation** *instantiates* or *calls* **my_control_module**.

   This causes **my_control_module** to be a leaf node in the hierarchy for **my_control_module_simulation**.



5. Perform the following:
   - Expand **Simulation** in the **Flow Navigator Panel**
   - Click on **Run Simulation**
   - Select **Run Behavioral Simulation**

   Several seconds later, the vivado simulator will provide a waveform window.

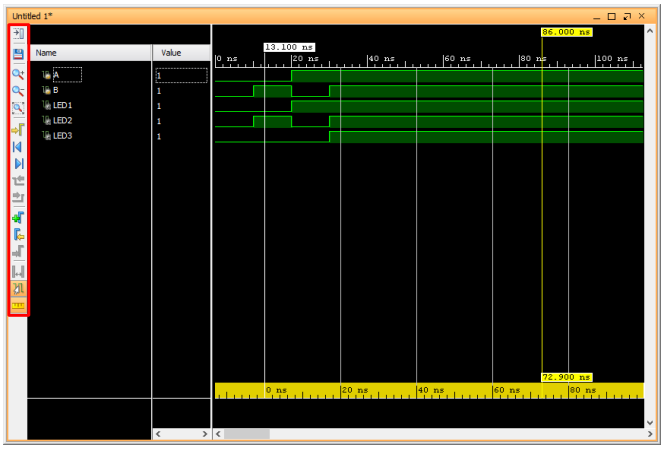


6. A noticeable waveform pattern may not be seen by default, as the time resolution used in the simulation is very small as compared to the amount of time the simulation is ran.

   Hence, with the simulation windows being the active window and from the menu, select **View → Zoom Fit**, or press **Ctrl+0**

   Look at the simulation results closely. How do the waveforms show that your design is indeed working as desired?

   Consider trying out the various options provided in the simulation window before going back to the **Workspace**.

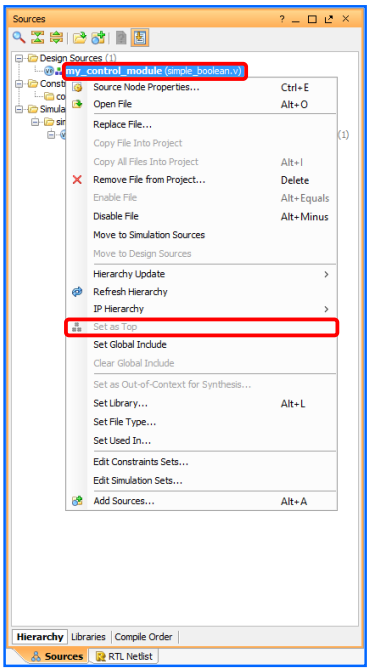


3.4 Synthesis

1. Logic synthesis transforms HDL code into an optimised set of logic gates to reduce the amount of hardware and efficiently perform the intended function. Different optimisation strategies are available, but this will not be the focus of this course.

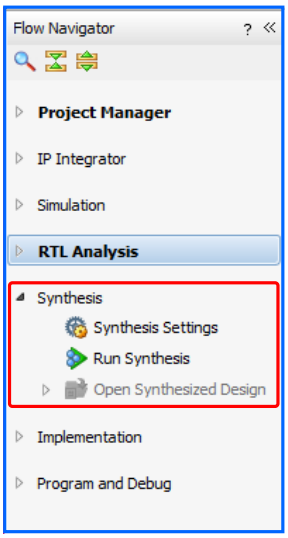
   Right-click on the **simple_boolean.v** file in the **Sources** window, and select **Set as Top**. This option is disabled if the file is already the top module, and in such a case, proceed directly to the next step.

   In general, when there are multiple design and simulation modules, the "Set as Top" option selects the design and simulation modules to be considered when performing the different stages of the project flow. Simply right-click on the module that you wish to process, and select the "Set as Top" option.



2. Perform the following:
   - Expand **Synthesis** in the **Flow Navigator Panel**
   - Click on **Run Synthesis**

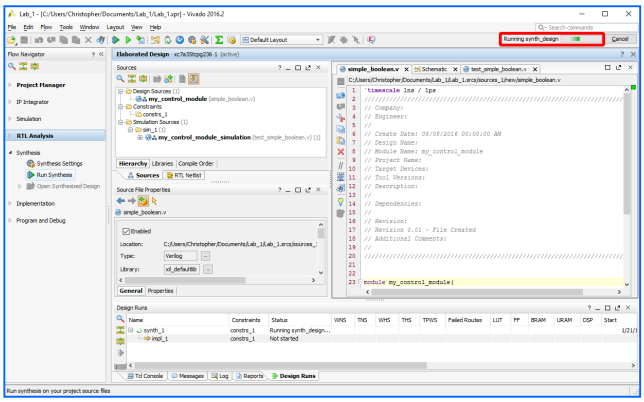
   Synthesis is usually one the most time-consuming part of the FPGA design flow. However, for this very simple example, it should take only multiple seconds.



3. While Vivado performs synthesis, the **Project Status Bar** at the top right provides an indication of the ongoing progress.

   Upon completion of the synthesis, the progress will change from **Running synth_design** to **Synthesis Complete**. A log will appear in the **Log** tab of the **Results Window**, while warnings and errors are displayed in the **Messages** tab.

   If a **Synthesis Completed: Synthesis successfully completed** windows appears, you may close it.



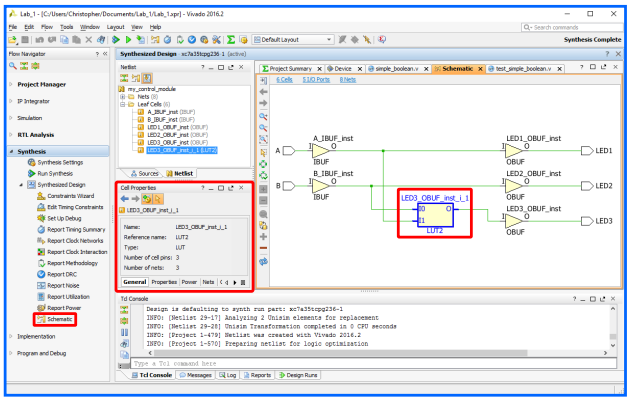
4. Perform the following:
   - Expand **Synthesis** in the **Flow Navigator Panel**
   - Expand **Synthesized Design** under **Synthesis**
   - Click on **Schematic**

   The schematic of the synthesised design will be generated and this synthesised circuit is an optimised version of the RTL schematic that was obtained in section **3.2**

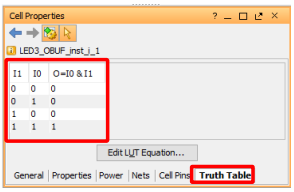
   The IBUF or OBUF symbols represent buffers. A buffer behaves like an inverter without its bubble, whereby its output is equivalent to its input.

   Click on the Look-up Table (LUT) that defines how the output **LED3** behaves. The **Cell Properties** window will appear for that specific LUT.

   Note that Field-Programmable Gate Arrays (FPGAs) consists of large amount of LUTs that can be configured to behave as multi-input gates.



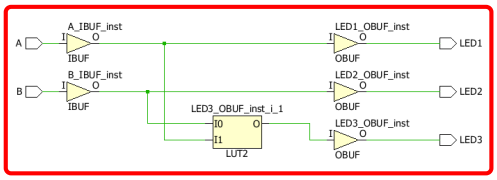
5. In the **Cell Properties** window for the LUT of **LED3**, open the **Truth Table** tab. Notice how for this simple example, this LUT is behaving as a simple AND gate.

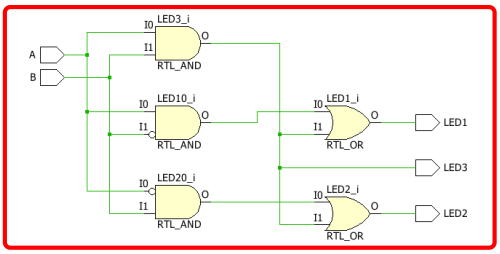


6. Compare the optimised and non-optimised schematics.

UNDERSTANDING | TASK 5

How is this optimised circuit equivalent to the SOP equations of section **2.1**?



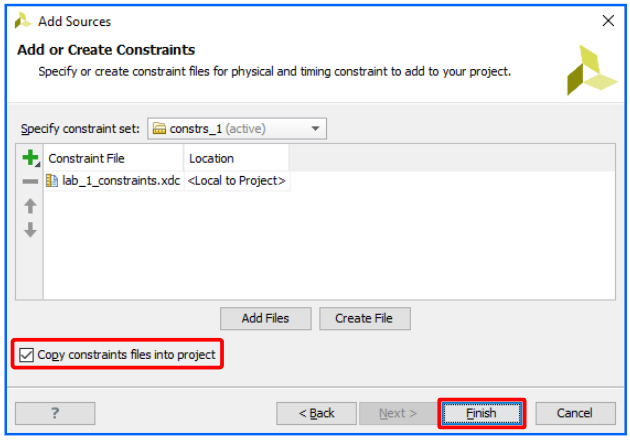


3.5 Design Constraints

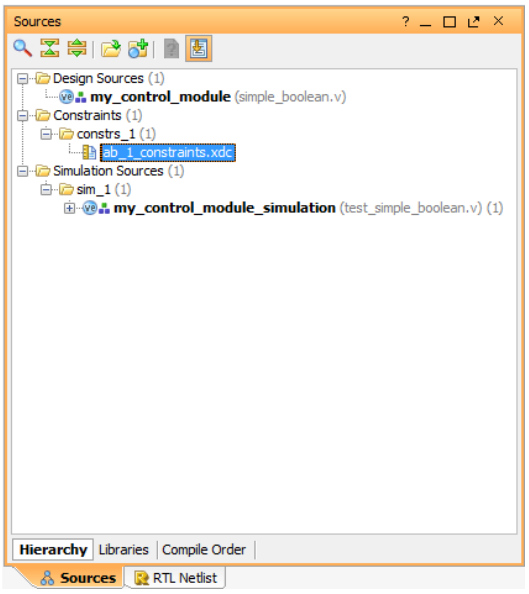
1. Design constraints, such as timing and physical I/O pin mapping, must be defined before doing an implementation. Given that our example is a combinational circuit, timing constraints will not be specified as there are no critical sequential elements in the design.

   Such design constraints may be set any time after writing the HDL code described in section **3.2**, including before the synthesis phase discussed in section **3.4**

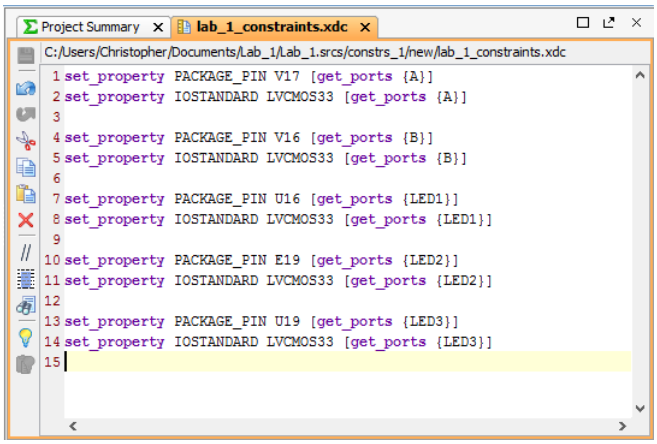
   Proceed with the following sequence:
   - Expand **Project Manager** in the **Flow Navigator Panel**
   - Click **Add Sources**
   - Select **Add or create constraints** and click **Next**
   - Click on **Create File** and give the XDC file a file name, such as **lab_1_constraints**. The XDC format stands for Xilinx Design Constraints here
   - When adding or creating a source file, it is recommended to select the option to copy the file into the project. This ensures that the all the files you are working with is located in one work folder, and not on multiple places on your storage device
   - Finally, click on **Finish**

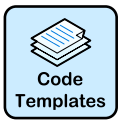


2. Open the **lab_1_constraints.xdc** file from the **Sources** window.

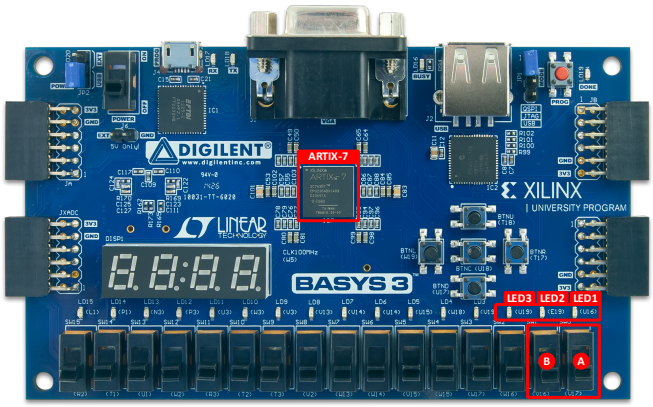


3. Enter the constraints as shown in the screenshot. The statements tell the Artix-7 FPGA how to link the signals in the Verilog top level design source module, to physical I/O pins on the FPGA.





4. The Artix-7 is located around the middle of the Basys 3 development board, and the switches and LEDs being used are located in the bottom right corner.

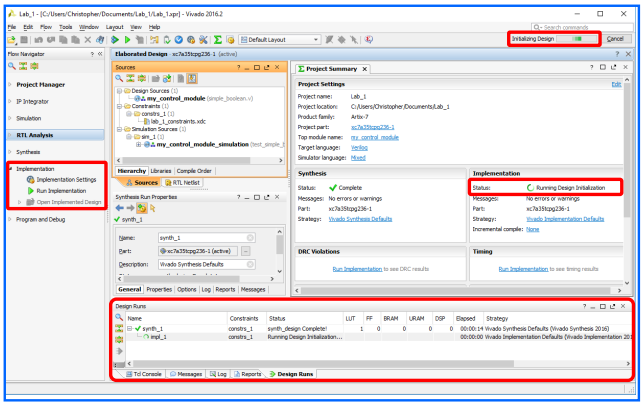


3.6 Implementation and Bitstream Generation

1. Do the following:
   - Expand **Implementation** in the **Flow Navigator Panel**
   - Click **Run Implementation**. You may be prompted and required to perform synthesis again if it is out-of-date due to changes in the HDL code

   Similar to the synthesis phase, the status of the implementation phase can be monitored at multiple places, including in the **Project Summary** window, the top-right corner of the **Project Status Bar**, or the **Design Runs** tab. Monitoring might be useful as implementation is another time-consuming process.

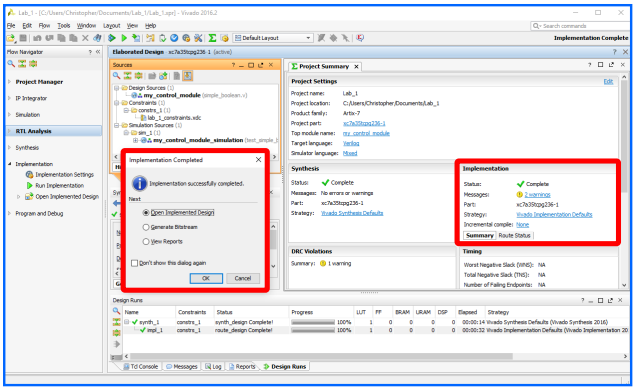
   If there are any warnings or errors, they are displayed in the **Messages** tab.



2. After successful completion of the implementation process, the **Implementation Status** will change to **Complete**.

   Notice that there are warnings but these can be ignored. However, critical warning and errors cannot be ignored, and will lead to failed implementation.

   If an **Implementation Completed** window pops up, you can choose **Generate Bitstream** and click on **OK**, or else you can click on **Cancel**.

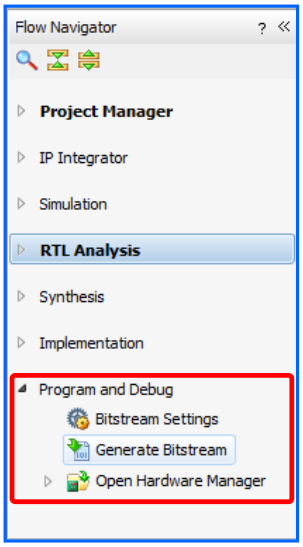


3. After the implementation phase, there is a need to generate a file that can be uploaded to the FPGA. Such a file is called a bitstream file, and it consists of binary values 0's and 1's that tells the FPGA how to behave.

   The bitstream file is obtained by doing:
   - Expand **Program and Debug** in the **Flow Navigator Panel**
   - Click on **Generate Bitstream**

   After waiting for several moments, the bitstream generation will be completed. Monitoring of the bitstream generation is done in the same manner as for the synthesis and implementation phase.

   A successful bitstream generation is the last step required before uploading the program to the Artix-7 FPGA.



3.7 Using the Basys 3 Development Board and Program Upload

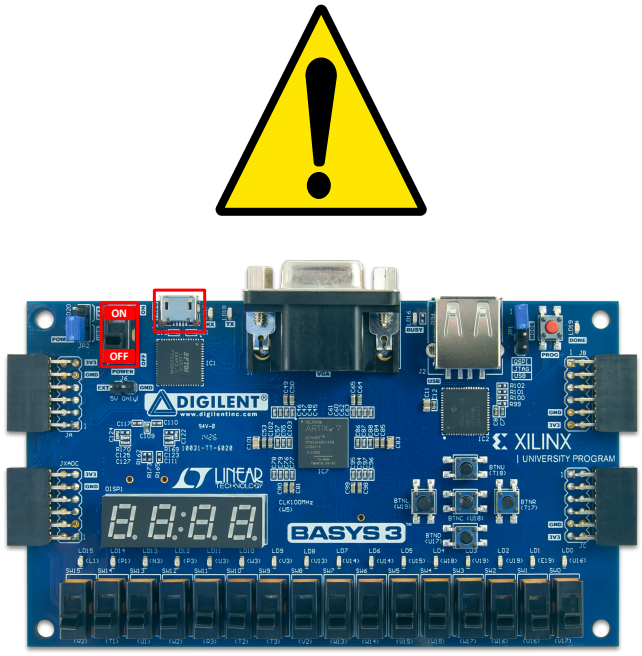
1. Before using your Basys 3 development board, and to prevent potential damage to it, take note of the following recommendations and warnings:

   ⚠ The Basys 3 development board is powered OFF by placing SW16 in the OFF position before connection to/removal from the USB port of the computer.

   ⚠ Do not force in the micro-USB cable upside down to the Basys 3 development board, as this will damage your micro-USB port and device. Carefully connect to the micro-USB cable in the correct orientation.

   ⚠ The chips on the board are electrostatic sensitive. Avoid touching them. Handle the board by the edges to prevent damage.

   ⚠ Make sure the board is not in contact with any metal components, whether above or below. Do not place any liquid sources near the FPGA board.

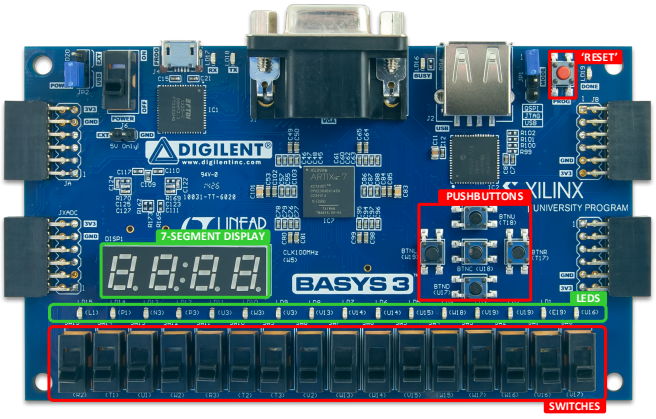


2. After connection of the Basys 3 development board to the computer, turn on the power by setting SW16 in the ON position.

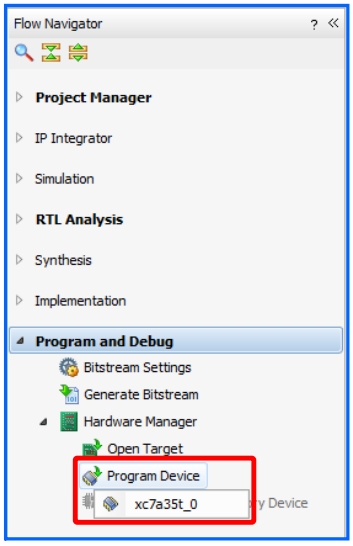
   Each Basys 3 development board has a unique identifier, and if this is the first time you are connecting your device to a specific computer, it will take a few moments for it to be detected and installed for that specific computer. Internet connection and windows device update is recommended to be enabled if your device cannot be installed.

   On faster computers and optimised computer system configurations, the device detection and installation takes a few seconds. In the lab, the desktop computers sometimes take a few minutes for the detection and installation.

   You may test your Basys 3 development board afterwards. Whenever powered up, the Basys 3 on-board flash device loads a pre-configured demo program. Observe the 7-segment display, use the pushbuttons and switches, or try the 'reset' button.

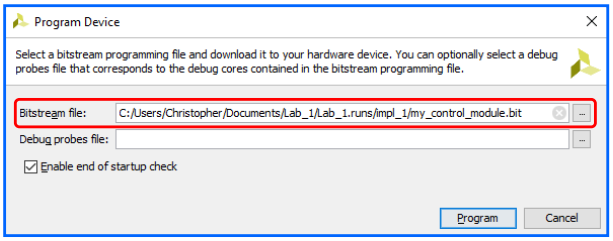


3. After successful connection and installation of your Basys 3 development board to the computer, and verification through the default pre-configured demo program, it is now time to upload your developed program, by doing the following:
   - Expand **Program and Debug** in the **Flow Navigator Panel**
   - Expand **Open Hardware Manager**
   - Click **Open Target**
   - Select **Auto Connect.** In case connection fails, consider pressing the 'reset' button, or turn your device OFF for a few seconds and ON again, while ensuring that it is detected and installed on your computer. Then try **Auto Connect** again.
   - If successful, the **Program Device** will be enabled, and you will be able to select **xc7a35t_0**



4. Clicking on **xc7a35t_0** leads to the window shown on the right. By default, if the bitstream was successfully generated, the path name in the **Bitstream file** is automatically provided.

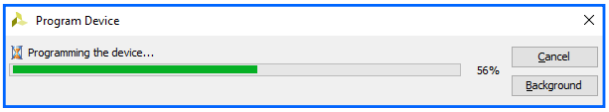
   Upload the .bit program to the FPGA by clicking on **Program**.



5. Your program will then be uploaded within a few seconds.

UNDERSTANDING | TASK 6

Verify the functionality of the design by using the input devices you have assigned to **A**, **B**, and observing the output devices assigned to **LED1**, **LED2** and **LED3**. Check what happens if the 'reset' pushbutton on the Basys 3 development board is pressed.



4.0 CLOSING NOTES FOR LAB 1

Congratulations on successfully completing your possibly first introductory FPGA design flow ^_^ . The step-by-step instructions provided throughout section 3.0 should have been relatively straightforward for you to complete. They included:

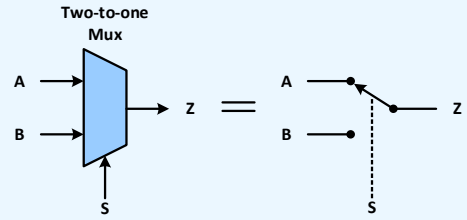
Two post-lab practice tasks are provided below for your own self-assessment before the upcoming lab 2.

FINAL UNDERSTANDING | PRACTICE TASK 1 FOR LAB 1

Create a new Vivado project from scratch. Do not reuse your existing project or design. The same design as described in section 2.0 needs to be implemented, with the following exception: There is an additional switch C, and if this switch C is in the OFF state, it forces all the three LEDs to be in the OFF state. If the switch C is in the ON state, the design behaves exactly as described in section 2.0. Simulate your design, and implement it on the Basys 3 development board.

FINAL UNDERSTANDING | PRACTICE TASK 2 FOR LAB 1

A multiplexer (MUX) is a combinational circuit that connects one of its input signals to the output, based on the control signal. A simple 1-bit two-to-one mux, with inputs **A**, **B**, control signal **S**, and output **Z**, is illustrated as a functional block diagram, together with its simplified truth table, in ***Figure 1.1.***



S	Z
0	A
1	B

Figure 1.1: *Functional block diagram and truth table of a 1-bit two-to-one multiplexer*

A quick way to implement the Verilog code for a 1-bit two-to-one multiplexer is using the conditional syntax:

condition ? expression1 : expression2;

Notice in the schematic, how the code is automatically recognised as a MUX.

Verilog code for 1-bit two-to-one mux, using the dataflow method

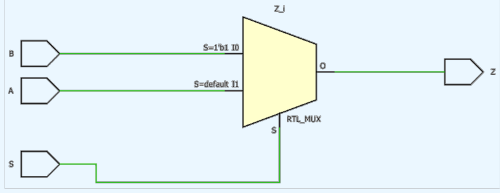
```
module my_2_to_1_mux (input A, B, S, output Z);

    // assign B to Z if S = 1 or assign A to Z if S = 0;
    assign Z = S ? B : A;

endmodule
```

Code Templates

RTL schematic for the 1-bit two-to-one mux



Simulate the multiplexer design to verify the Verilog code, and then create a constraint file code to connect inputs **A**, **B**, and **S** to switches, and output **Z** to an LED, on the Basys 3 development board. Upload your bitstream to the Basys 3 development board and verify that the behaviour of the program follows the truth table of the 1-bit two-to-one mux.