Department of Electrical and Computer Engineering
ENEL453: Digital System Design
Fall 2017

Lab 1: Introduction to Simulation, Synthesis and Implementation on an FPGA

Instruction Manual

1 Overview

In this lab, we will implement simple logic circuits using VHDL, and synthesize it to work on the Basys3 FPGA board. The board manual is located here: `https://reference.digilentinc.com/_media/basys3:basys3_rm.pdf`. We will begin with a combinational logic circuit, and move to sequential logic. This lab will introduce testbenches, which are used to test the designed digital systems design and will be heavily used in future labs. There are two portions of this lab which you must demonstrate to a TA.

This lab is worth 5% of your term grade and broken up as:

- Pre-lab: 1%
- In-lab simulation: 2% (any team member may be asked questions to verify understanding)
    - Combinational logic simulation – 1%
    - Sequential logic simulation – 1%
- In-lab demonstration: 3% (any team member may be asked questions to verify understanding)
    - Combinational logic demonstration -1%
    - Sequential logic demonstration -2%

2 Before You Begin

- Make sure you have completed the pre-lab exercises: you need to have the pre-lab signed off by a TA when you come into the lab.

- Read the entire lab instruction manual before coming to lab.
- **A maximum of three students may work together at any station in ENG 105/124/130.** Each group member should be able to perform the whole project.
- Back-up your project frequently. Save all intermediate materials that you generate (figures, screenshots, code snippets). Be sure to structure your code with appropriate indentations, this will greatly help you debug your code.
- **The simulation and demonstration must be checked by the TA prior to the end of the lab period.** Late demonstrations can be given at the lecture or tutorial times with the following penalty (Tuesday -1%, Thursday -2%, Friday -3%). No lab write up is required, other than the pre-lab.
- Do not share your code. Similar sources will be awarded a mark of **zero**. We may use Moss (Measure of Software Similarity developed at Stanford University), which is an automatic system for determining the similarity of programs.
- You can sign out one board per group for use for a semester, you have to fill in the Sign-Out Form and comply with its requirements. The boards are available from Mr John Shelley, room ICT 217.
- We recommend using the lab station for performing the lab rather than personal computers. If you want to work on the projects at home or on your laptop, you can download the CAD tools (free) from Xilinx: Xilinx Vivado HLx Webpack 2017.2: https://www.xilinx.com/support/download.html. These tools do not run on Mac OS and be sure to install the Webpack version.

3 Creating Projects in Vivado

Since this is an introductory lab and review from ENEL 353, the project structure for our VHDL implementation will be fairly simple. As the labs progress, you will find yourself managing many files, so keep things organized!

3.1 Open Vivado 2017.2

Open the Start menu on your computer, and locate the “Vivado 2017.2” in the list of All Programs. Open it list and navigate to Quick Start > Create New Project.

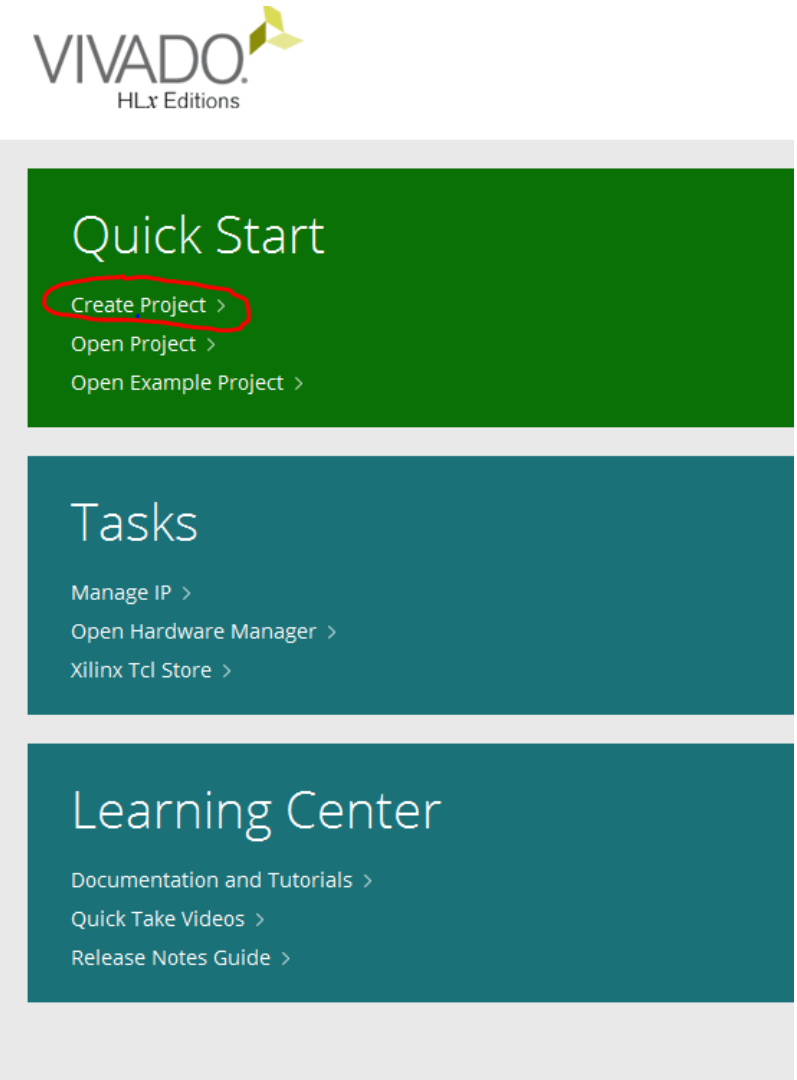


Figure 1: Select Create New Project.

3.2 Create a New Project

Launch Create New Project, choose Next in the project wizard. Follow the screenshots and the captions.

Location of your projects:
Make sure your project resides on the C drive in C:/Temp directory! This is because the latency in file transfer to the H drive prevents Xilinx tools from working. You also need to save your project to a USB stick at the end of the labs as the C drives get cleaned every week!

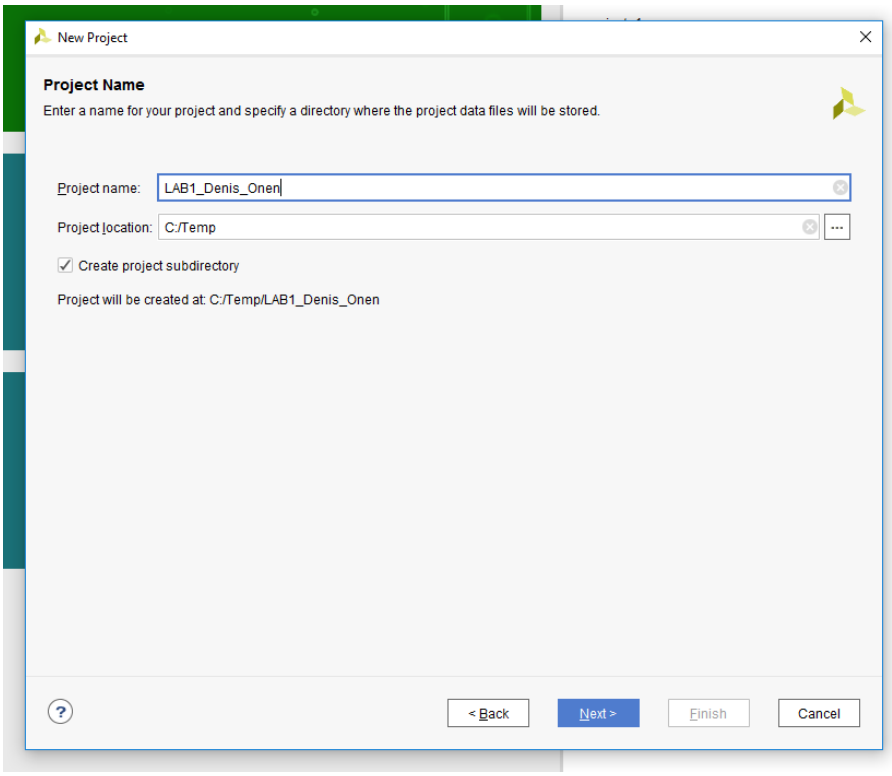
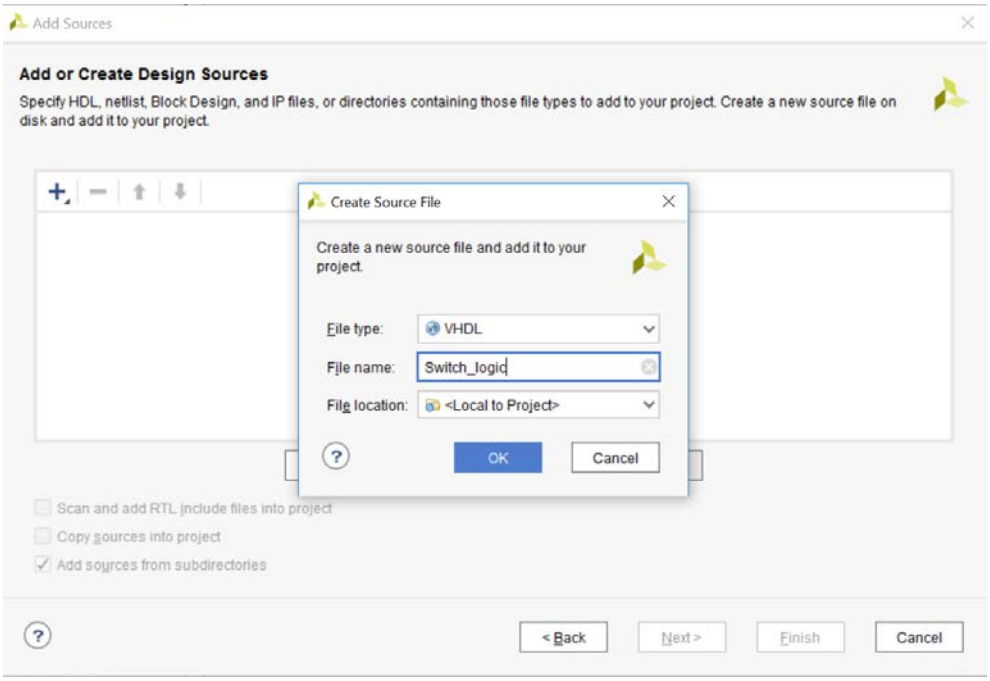


Figure 2: Fill in the name to be in the format `LAB1_LASTNAME1_LASTNAME2`. Make sure the location is not on the H drive. The Top-level source type should stay as HDL. Press "Next".

- Choose Project Type > RTL Project > Next.
- In Add Sources:
  - Select (at the bottom) Target language as VHDL. Do same for Simulator language, choose VHDL
  - Press "+" and choose Create File.
  - Select File type as VHDL,
  - Enter "Switch_logic" in File Name box and choose your location as C:/Temp/LAB1 (or leave the default "Local to project"). We will implement some logic using Slide switches at your Basys3 board. (You should be familiar with these switches from the pre-lab exercises.) For now, we do not need to add other sources, click "OK" and "Next".



- Skip “Add Existing IP” and “Add Constraints”, and come to “Default Part”.

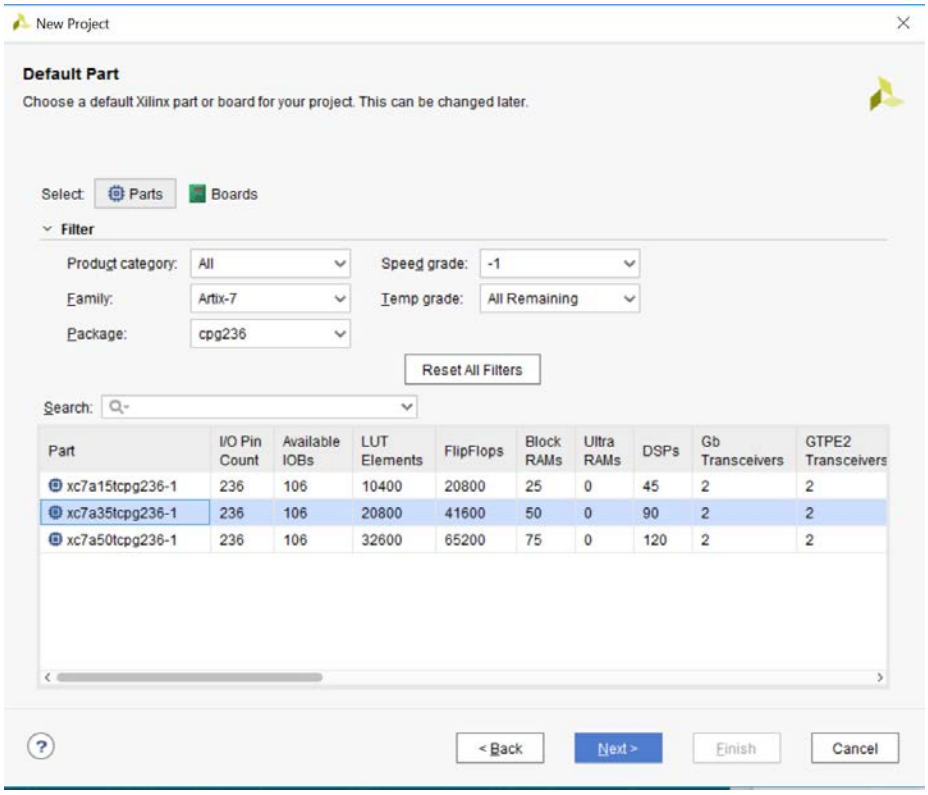


Figure 3: Select the correct device (or you will get weird errors when you try to build the design) by using the dropdowns to select “Artix-7”, “cpg236”, and “-1” as shown in the figure, to reduce the number of devices to select from. We are using Artix-7 family, and the device on the board is xc7a35tcpg236-1. The package is cpg236 (**this is important!**). Speed grade is -1. Press “Next” once you’ve selected the correct fields.

You will see the New project Summary. Press “Finish”. You will be presented with LAB1 - [C:/Temp/LAB1/LAB1.xpr] project window. Immediately, another window called “Define Module” appears.

3.3 Define Module

You should now see you “Entity name” as Switch_logic and “Architecture name” as Behavioral. Fill out the port names to match that shown in Figure 4.

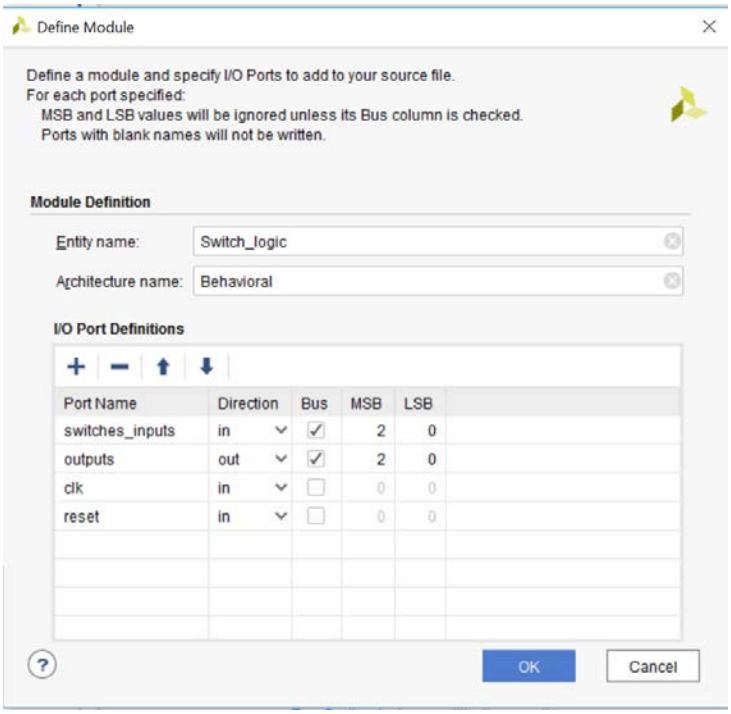


Figure 4: Define the model: we will only use 3 switches today, which is why the bus widths are set to be from 2 down to 0 (so 2, 1, 0). `Clk` is a typical notation to show clock of our design. `Reset` should be used in all designs to put them in a known state. Press "OK" when you're ready.

Clicking "OK" produces a Project Manager with Project Summary window (Figure 5).

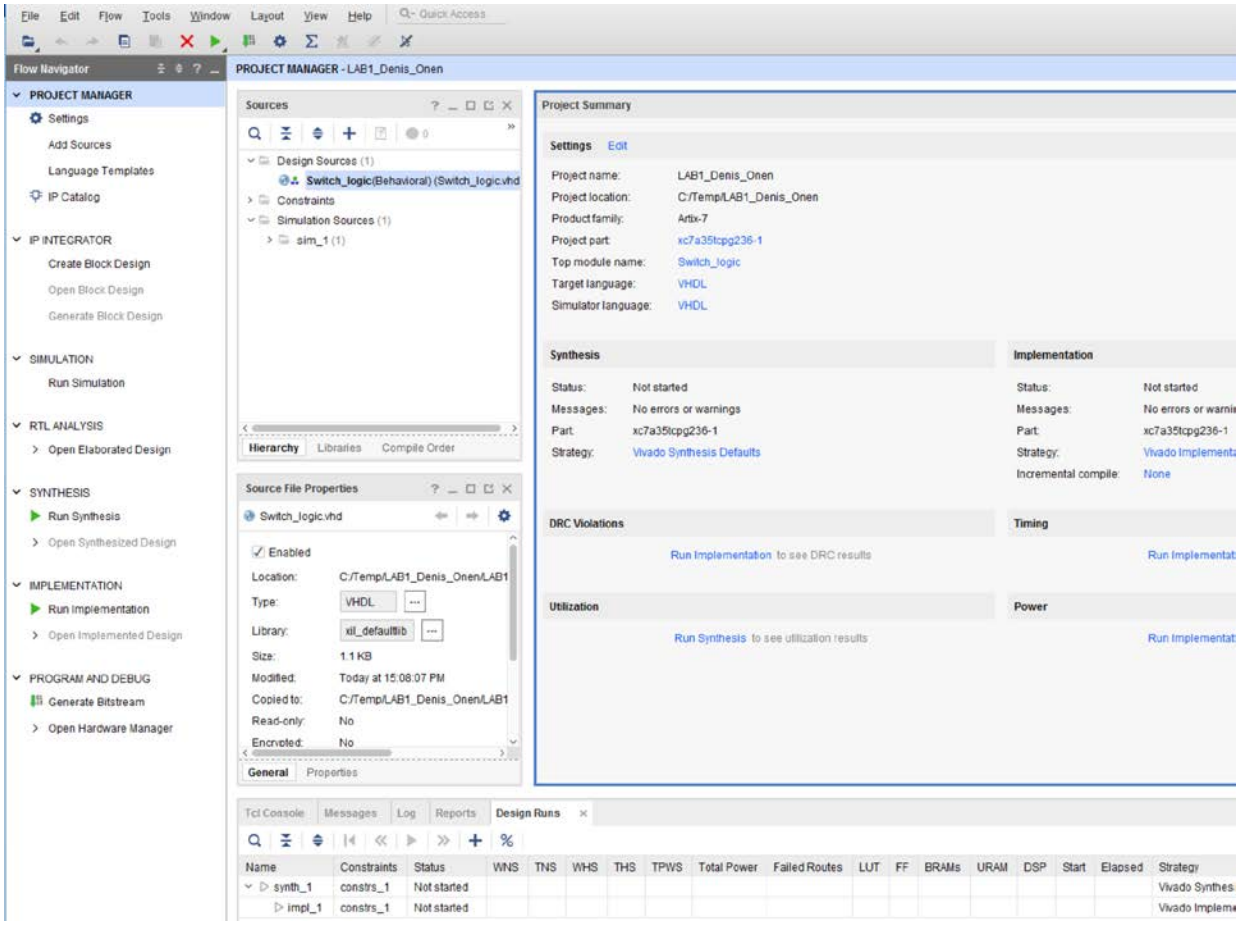


Figure 5: The Project Summary window.

In the middle window Project manager, under Sources, see “Design Sources”. Click on a file called `Switch_logic.vhd` (or whatever you chose to name it). You will see on the right the opened content of the file `Switch_logic`. You have to make changes to this file in order to add functionality.

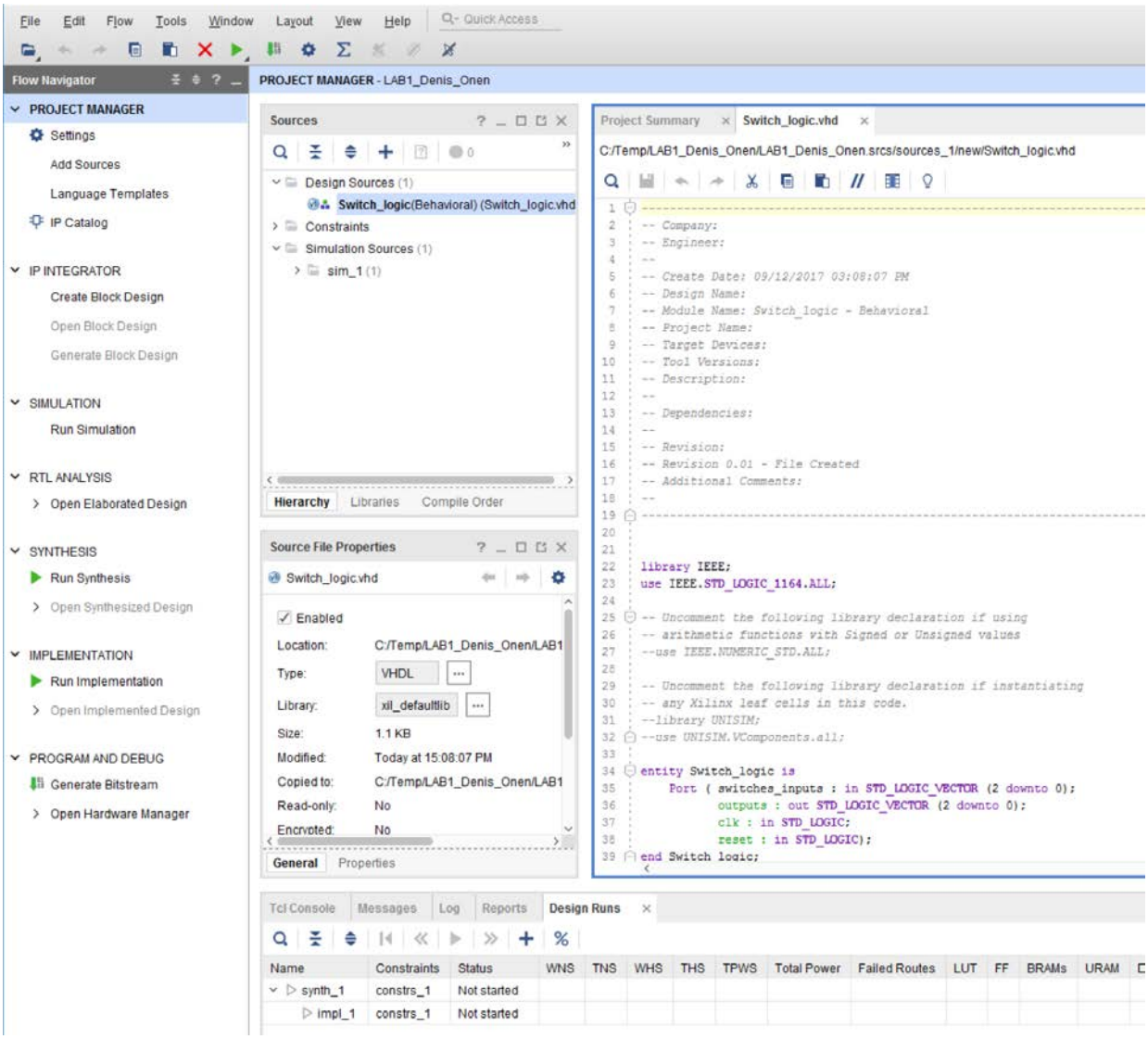


Figure 6: The source window on the right gives you details about the ports you chose to specify (e.g. switches_inputs, outputs, clk, reset), as well as file format/type and language. The ports are not set in stone and you can always change them in code.

3.4 Creating Simple Combinational Logic

The file contains two important sections. The first is `entity`. It describes the component you are designing as a “black box” - it only reveals the inputs and outputs.

```
entity Switch_logic is
    Port (switches_inputs : in STD_LOGIC_VECTOR (2 downto 0);
          outputs : out STD_LOGIC_VECTOR (2 downto 0)
         -- clk : in STD_LOGIC;
         -- reset : in STD_LOGIC
         );
end switch_logic;
```

Look at your pre-lab exercise Part 3 on creating functions W and V. Note that there is no clock in the diagram you created, since it is combinational logic. For now, you must comment out the `clk` and `reset` signals as above.

Note that to comment statements in VHDL, we use `--`, instead of the usual `//` in C or C++. Also note the lack of a semicolon at the end of the last port declaration.

The second portion of the file is `architecture`. It describes the desired functionality of the block. All internal connections are made within `architecture`. In the current file, the Architecture is empty. You will need to finish it. For now, you will be using the code given below. Replace the `architecture` block (this block begins with `architecture Behavioral of switch logic is` and ends with `end Behavioral;`) with the following code (you should be able to copy-paste it):

```
architecture Behavioral of Switch_logic is
   -- Internal signals:
   signal W_int: std_logic;
   signal W, V: std_logic;
   signal A, B, C: std_logic;

begin
   -- Combinational logic! Note that "<=" is an assignment operator
   W_int <= ((not B) and A) or ((not A) and B); -- models XOR

      -- ADD logic for signals (W and V)

   -- Assign the outputs. We only have one signal for now
   outputs(0) <= W;
   outputs(1) <= V;
   outputs(2) <= '0'; -- We will connect this later

   -- Get the inputs from the slide switches on the FPGA board
   A <= switches_inputs(0);
   B <= switches_inputs(1);
   C <= switches_inputs(2);

end Behavioral;
```

Notice the line `-- ADD signals (W and V)`. Create these two lines with the following functionality:

- W shall be connected to W_int;
- V is a logical AND between C and W_int according to the pre-lab exercise.

Pressing the top-left icon or pressing `CTRL+S` (save) will also perform a quick syntax error check. Correct any errors shown in the bottom window Messages.

3.5 Add testbench file to simulate the design

Next, we need to add a testbench. In the middle window “Sources”, right click on the .vdh file name and choose “Add Sources”. In the appeared window “Add Sources”, choose “Add or create simulation sources”. Click “Next”. Press “+” and choose “Create File”. Choose File type as VHDL, File name as tb_switch_logic and the same file location, C:/Temp/LAB1. Click “Finish”. Again, “Define Module” window will appear. Leave the file empty for now and click OK. Now your Project Manager (Sources) window should have both the tb_switch_logic.vhd and Switch_logic.vhd.

Open the tb_switch_logic.vhd file by clicking on it. You have to create an empty entity, and architecture with a declared component (which is our Unit Under Test (UUT), input and outputs. Inside the Architecture, stimulus processes are added as seen below.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_switch_logic IS
END tb_switch_logic;

ARCHITECTURE behavior OF tb_switch_logic IS

   -- Component Declaration for the UUT

   COMPONENT Switch_logic
   PORT(
        switches_inputs : IN std_logic_vector(2 downto 0);
        outputs : OUT std_logic_vector(2 downto 0) --;
            -- clk : in STD_LOGIC;
            -- reset : in STD_LOGIC
       );
   END COMPONENT;

  --Inputs
  signal switches_inputs : std_logic_vector(2 downto 0) := (others =>
     '0');

  --Outputs
  signal outputs : std_logic_vector(2 downto 0);

 BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: switch_logic PORT MAP (
         switches_inputs => switches_inputs,
         outputs => outputs
      -- clk => clk,
      -- reset => reset
       );
```

```
--Stimulus process here
  -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

      -- Set all inputs to 0
      switches_inputs(0)<='0'; --A
      switches_inputs(1)<='0'; --B
      switches_inputs(2)<='0'; --C

      wait for 50 ns;

      -- Test an input combination
      switches_inputs(0)<='1'; --A
      switches_inputs(1)<='0'; --B
      switches_inputs(2)<='0'; --C

      wait for 100 ns;
      wait; -- Keeps it from restarting
   end process;

END;
```

You can use the above code for the testbench file, and modify it later. You can right click the top module, then click “Add Sources” to add or create additional .vhd files for future lab exercises.

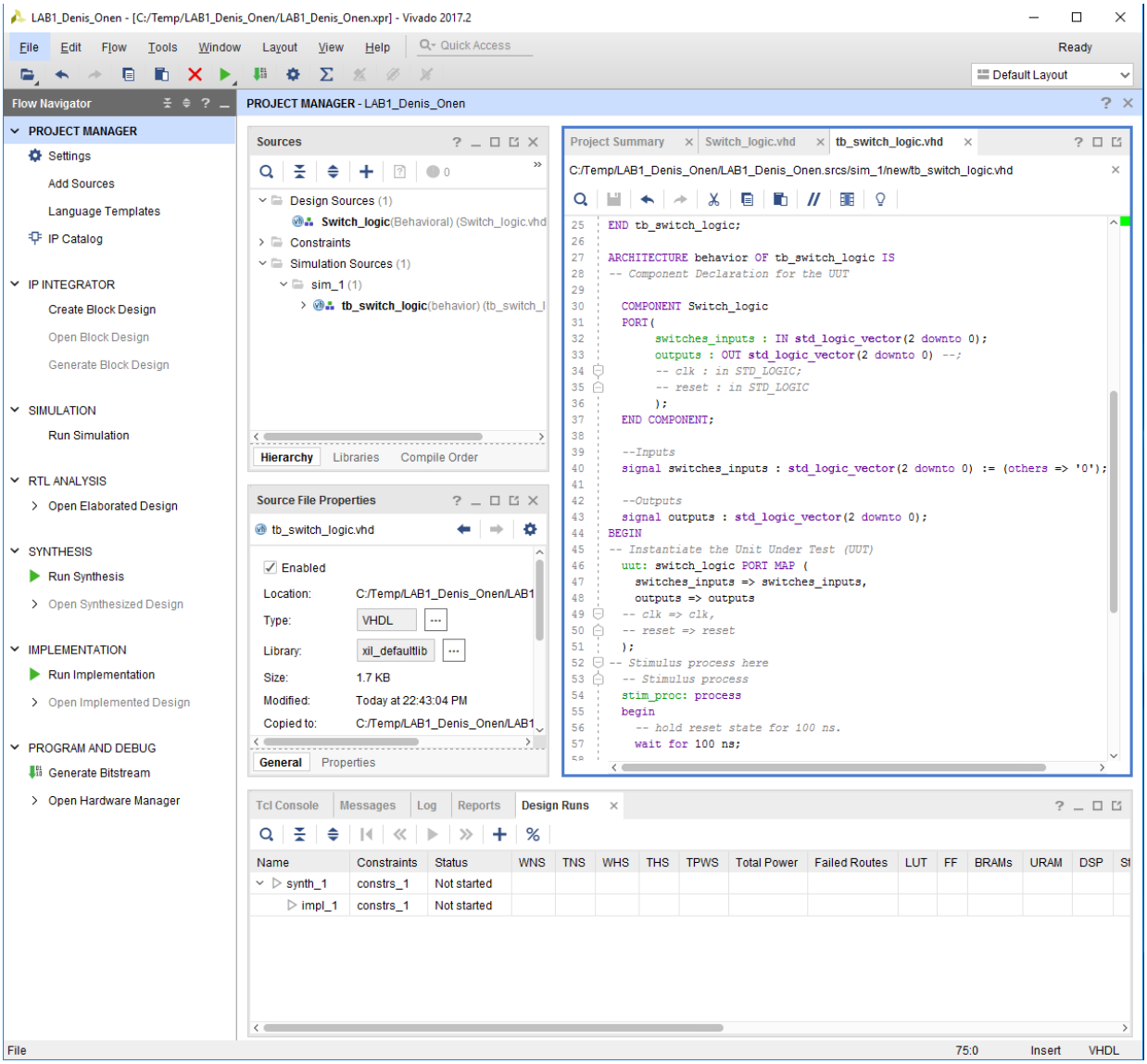


Figure 7: Adding the testbench to Simulation Sources.

To run this and make sure everything works before moving on to the experimental portion of this lab, in the top menu, choose the Flow > Run Simulation > Run Behavioral Simulation. You can do the same from the leftmost panel "Flow Navigator". Make sure to save the VHD files before running the simulation.

Errors when launching the simulator:
check for messages at Tcl Console panel at the bottom of the window, or Messages panel

In the window that opens, you will not immediately be able to see anything going on since the timeline is zoomed-in. To remedy this, press the Zoom Fit in the tool-bar for the simulation window (see Figure 8).

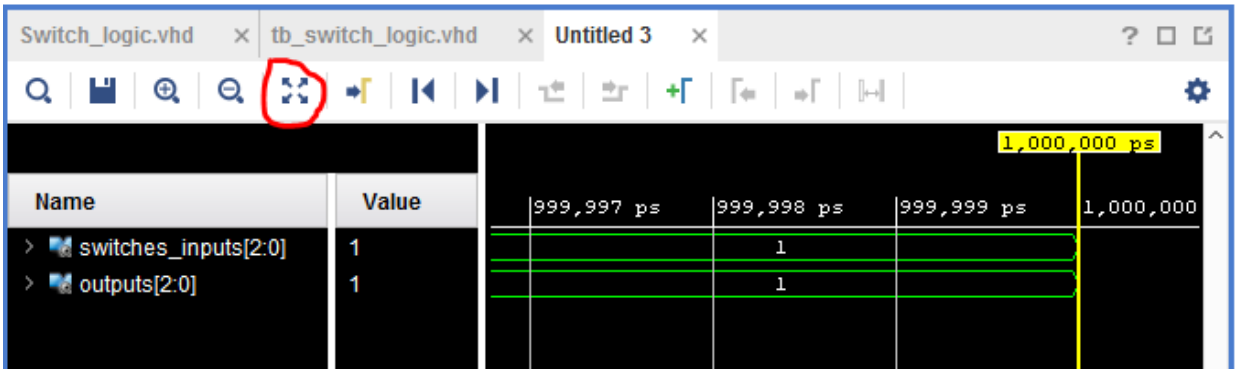


Figure 8: The Zoom Fit button is in the red circle (snapshot before button pressed).

Signals are listed on the left hand side and can be expanded in order to view individual bits. Examine the `switches_inputs` and the `outputs` to make sure this matches what you predicted (in pre-lab) for A = 1, B = 0, and C = 0. Remember, `outputs(0)` is tied to `W` and `outputs(1)` is tied to `V`. Right-click on the busses outputs[2:0] and switches_inputs[2:0] and select Radix > Binary, to see the binary value of each bus.

4 The Experiments

The experiments in this lab will consist of verifying the existing design through a complete testbench, implementing and downloading the design to the board, modifying the design, and proceeding through the testbench/implement route once again. Let's get started!

4.1 Complete the testbench

In its current state, the testbench does not test all combinations of our logic circuit. In the pre-lab exercises you should have devised a way to test all combinations in a truth-

table sort of way. If you did, consult the following code listing for syntax to implement the four processes. If you did not, you can use the method in the testbench you currently have to test all combinations manually.

```
A_process: process
   begin
     switches_inputs(0)<='0';
     --YOU DO THIS
     --wait for TIME_A ns; (pick some value, for example 20 ns)
     switches_inputs(0)<='1';
     --wait for TIME_A ns;
  end process;

  B_process: process
  begin
     switches_inputs(1)<='0';
     --YOU DO THIS
     --wait  for  TIME_B  ns;
     switches_inputs(1)<='1';
     --wait for TIME_B ns;
  end process;

  C_process: process
  begin
     switches_inputs(2)<='0';
     --YOU DO THIS
     --wait  for  TIME_C  ns;
     switches_inputs(2)<='1';
     --wait for TIME_C ns;
  end process;

     -- Stimulus process
  stim_proc: process
     begin
        -- Nothing here now :(
     wait;
  end process;
```

Adjust the view of the simulated waveforms so that it shows all possible inputs (there should be 8 different ones in total) and the corresponding outputs.

4.2 Pre-synthesis RTL design

Vivado also provides RTL schematic based on the .vhd file. In the Flow Navigator, select RTL Analysis and Open Elaborated Design. You will see a schematic diagram in the right window.

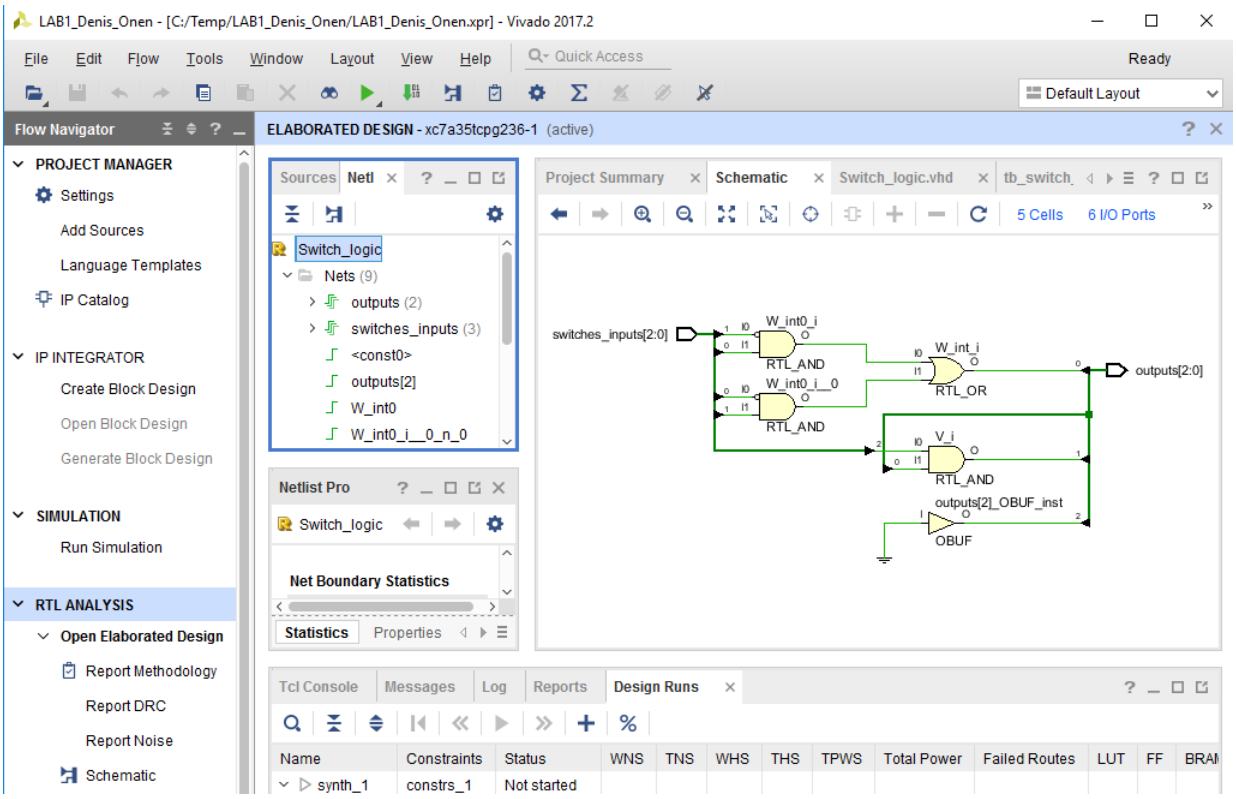


Figure 9: Synthesized schematics at gate-level

4.3 Synthesis (RTL level)

Now that your code works, we can download it to the FPGA. To do this we must perform Synthesis, Implementation and Generation of a bitstream.

In the Flow Navigator panel, under Synthesis, choose Run Synthesis. This might take a few minutes to complete. You should not have any warnings - if you get any, make sure you fix them before moving on.

When Vivado finishes synthesizing your project, you will see the Synthesis Completed window. Choose "Open Synthesized Design" and then press OK. This will show you the generated Netlist in the middle window, and the "Device" on the right, which is a routed array of blocks, representing FPGA layout. The left panel in "Synthesis" will now show the Synthesized design options, which include settings, reports, and the last one is called "Schematic". If you click on it, it will show you a look-up-table (LUT) level Schematics on the right. You shall see Schematic your Synthesized Design in the window to the right. This is your post-synthesis RTL design.

Note the bottom panel "Design Runs" – it will now show you a green checkmark at "synth_1" meaning that synthesis is complete.

4.4 Implementation

The Implementation panel is located on the left. Use the default Implementation Settings and click Run Implementation. The progress can be observed at the bottom panel "Design Run". When it is finished, the "Implementation Completed" window appears. Choose "Open Implemented design". It will update the Project Summary on the right.

4.5 Bitstream generation

In order for Vivado to know which pins on the FPGA board are used as inputs and outputs, and power supply, you need to create the Xilinx Design Constraints (XDC) file. It specifies the pin numbers and other constraints on the signals coming in and out the FPGA. It can be formed by using software or by using a pre-defined standard file. You are provided with one that Digilent generated for its Basys 3 board, called Basys3_Master_lab1.xdc. To add the .xdc file to the design, in the Sources Window right-click on constr 1 directory and choose "Add Sources" > "Add or create Constraints" > Next > + and find the Basys3 Master.xdc file (you can copy it from D2L to your C:/Temp or

to your flash drive and get it from there). Open it and you can edit it. You have to pick which pins you are going to use, and uncomment (remove # , or select a block of codes and press ctrl + / to uncomment).

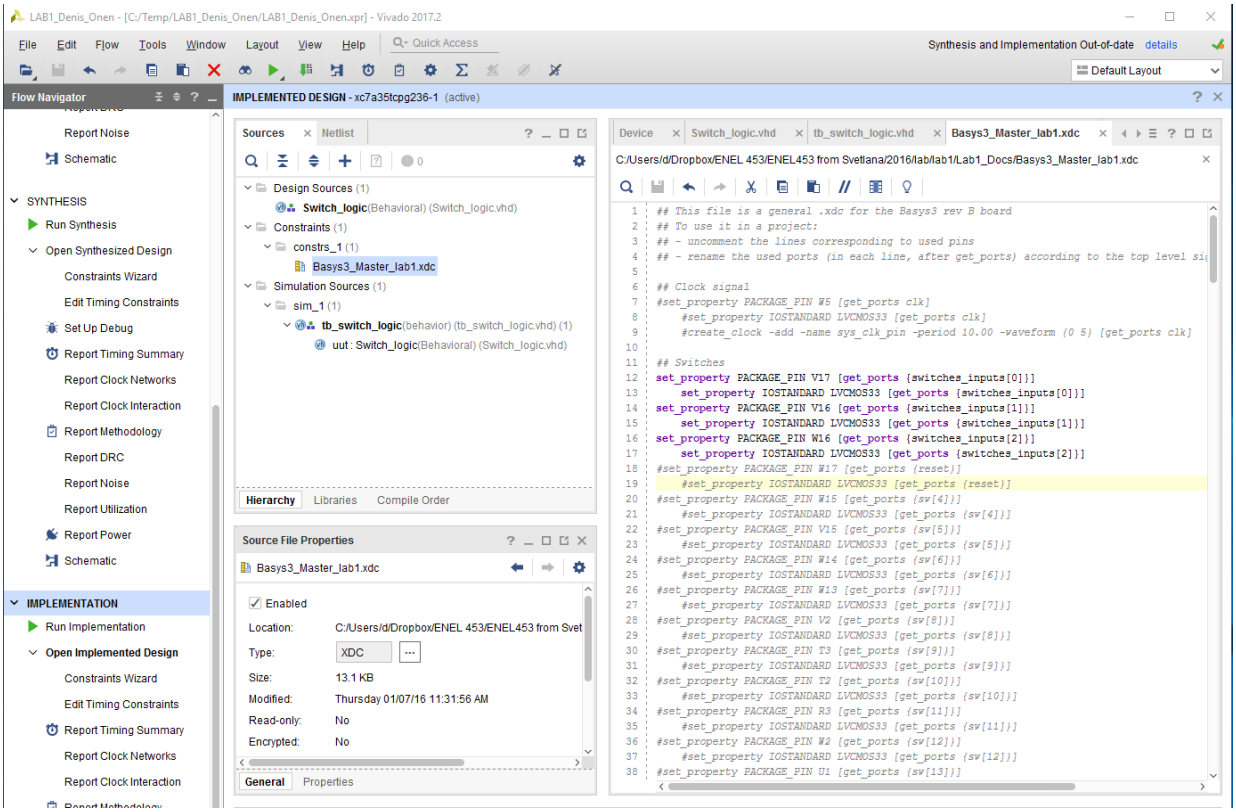


Figure 10: Editing .xdc file

Make sure the names of the inputs and outputs in your Switch_logic.vhd file match the pin names in the constraint (.xdc) file. Use the Table below to track this correspondence.

Decoder Signals	FPGA pins	Comments
switching input(0)	V17	Input A
output(0)	U16	LED(0)

Save the .xdc file after editing.

Demonstrate to a TA:
Show your constraints file to a TA to make sure you've assigned the correct pins.

Now you are ready to generate the bitstream file as following: Tools > Settings > Bitstream. Check the box beside the "-bin file" and click OK.

Next, set up Device Properties: In the left column (Flow Navigator), click "Run Synthesis" under the "Synthesis" section, click "OK" in the popup window. This will generate the ".bit" and ".bin" file. When it finishes, select "Open Synthesized Design". In the new window, select Tools > Edit Device Properties… > Configuration, choose the configuration rate to be 33. Choose the Configuration Mode to be JTAG/Boundary Scan and click "OK" in the message box.

Now go to "Program and Debug" section in Flow Navigator, click "Generate Bitstream", and Yes in the message box. When the process completes, both .bit and .bin files will be generated. Vivado will show you the "Bitstream Generation Window".

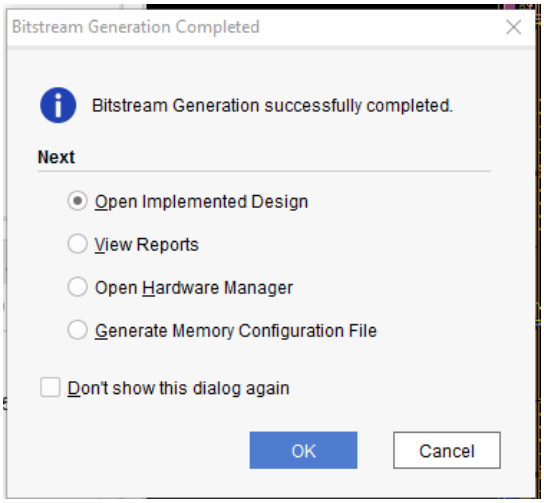


Figure 11: Bitsream generation information window

Now you are ready to download your design!

4.6 Programming the board

In our labs, we will be using programming via JTAG which is used as a programming, debugging, and probing port that communicates through the micro-USB port. The micro-USB connection also supplies power to the board.

Take your Basys3 board out of the case and make sure that the jumper JP1 is in the JTAG position. Take the provided USB cord and plug the micro-USB side into the Basys 3 and plug the USB A side into the computer.

Turn the board on - the power switch is next to the micro-USB port on the Basys3 board. A red LED should come on.

Wait until the plug and play driver finishes installing on the computer.

Open the Hardware Manager from the “Program and Debug Panel” in the Flow Navigator. Select “Open target” at the top and follow the wizard. Choose “Auto connect”.

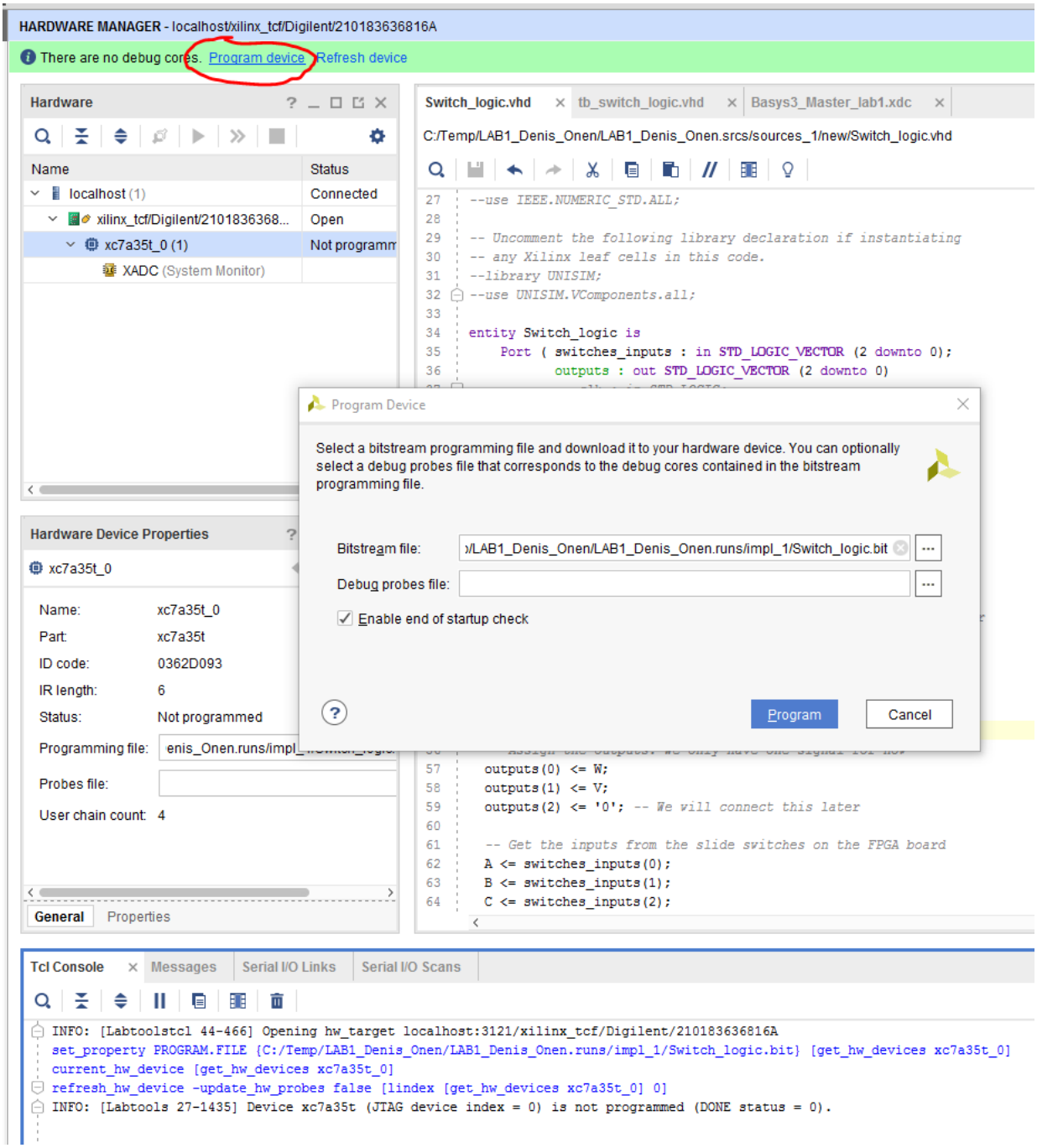


Figure 12: Hardware manager window

Click “Program device” (in the green bar) then click Program.

This will program your Basys3 through the JTAG connector. Once the programming window is closed, slide the switches that you have set as inputs around and make sure the output works as intended (it should follow your truth table).

Demonstrate to a TA:
Show that your switches turn the LED on as intended.

4.7 Modify the design: sequential logic

In this section, you will implement a **sequential logic circuit**, using an equation that you created in the pre-lab. Navigate back to the `switch_logic.vhd` file (your main one), and uncomment the `clk` and `reset` line in the entity (the ones you commented out before); remember to put the semicolon back now that your last port is different:

```
entity switch_logic is
    Port ( switches_inputs : in STD_LOGIC_VECTOR (2 downto 0);
           outputs : out STD_LOGIC_VECTOR (2 downto 0);
           clk : in STD_LOGIC;
           reset : in STD_LOGIC
          );
end switch_logic;
```

For this part, we will implement the design using an inferred simple D flip-flop (DFF). A flip-flop is a circuit with two states (hence the ”flip” and the ”flop”) that can be used to store digital data. A single flip-flop will store a single bit. A few facts about the DFF that you may find useful or interesting:

- It samples the input data using a clock **edge** as the driver.
- D is the input of a DFF, and Q is the output. Some DFFs will also have an inverted output called $\bar{Q}$. You are free to implement this inverted output if you are particularly eager outside of lab time.
- DFFs can have `set` and `reset` pins that are used to clear the output. We will only implement the `reset` - this pin will always drive the output to 0 if activated (no matter what the input is).

Consult Figures 13 and 14 for further details. Make sure you understand Figure 14.

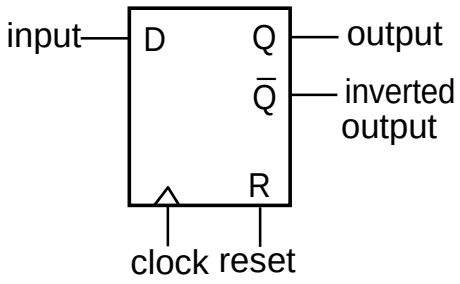


Figure 13: A block diagram usually used to describe a D Flip-Flop.

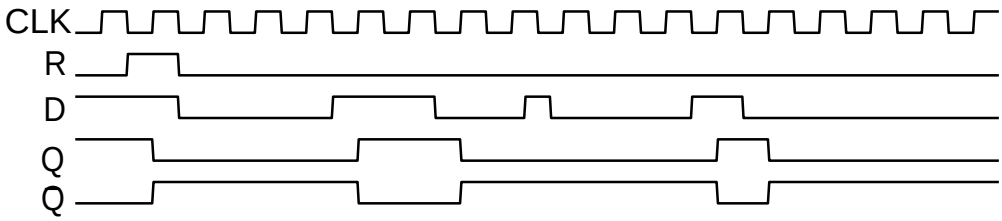


Figure 14: A timing diagram showing behaviour of a DFF. The reset signal in this case is synchronous with the clock; asynchronous reset is also quite popular. Note that the input is only latched to the output at the rising clock edge.

To begin, add a signal `V_del` to the list:

```
-- Internal signals:
  signal W_int: std_logic;
  signal W, V, V_del: std_logic;
  signal A, B, C: std_logic;
```

Signal `V_del` shall be the output of a DFF, that samples (with a delay) the **combinational logic** version of `V`. To implement a sequential process that implies a DFF, we will need to add another process called `logic_of_switches` (the name can be anything, but keep it descriptive). Keep the combinational part. Paste the following code into the `architecture` description of `switch_logic`, either immediately after the `begin` statement anywhere within the `begin` block:

```
  logic_of_switches: process(clk, reset) begin
    if (reset = '1') then
      -- Reset V_del to a known state
      V_del <= '0';
    elsif (rising_edge(clk)) then
      -- Assign logic to V_del here
      V_del <= V;
```

```
    end if;
  end process;
```

You can see that the process is sensitive to `clk` and `reset` as those are the signals contained in the sensitivity list:

```
logic_of_switches: process(clk, reset) -- <-- Sensitivity list in
   brackets
```

The process will start every time one of the signals in the sensitivity list changes. Our DFF will use an asynchronous `reset` signal, and be sensitive to the `rising edge` of the clock.

Assign the outputs, so that `outputs(2)` is `V_del`.

Because this is now a design that uses a clock, your old testbench will no longer work properly. For this, you need to generate a new testbench file (keep your previous testbench file, but you can disable it by right clicking on the file in the Vivado source window). Follow the directions as before, but this time give it a name which suggests that this is for a sequential logic test. Be sure to add the signal definitions for clk and reset in addition to uncommenting the clk and reset lines in the testbench.

The newly generated file will have the `clk_period` set to `10 ns` by default (and you will no longer get errors!):

```
-- Clock period definitions
  constant clk_period : time := 10 ns;
```

The clock will tick using a process similar to those developed for A, B and C processes:

```
  -- Clock process definitions
  clk_process :process
  begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
  end process;
```

Verify that the timing you are using for switches inputs (specified by the `wait for` statements) is valid with the selected `clk_period`. This means that **the inputs cannot be changing faster than the clock does** (or else you won't see any effect when using sequential logic) - refer to Figure 15 to see that short events that occur outside of the rising clock edge will not be captured.

Run a simulation (or two) using the new testbench until you are satisfied that your design works as intended. Make sure you check that `reset` works as well.

4.8 Further modified sequential logic design

Now in the switch logic.vhd file, move the line

```
V <= C and W_int;
```

from combinational part to the process `logic of switches` (sequential part):

```
V <= C and W_int;
V_del <= V;
```

Note that on reset=1, you have to reset both V and V del to 0.

Run the Simulation and generate the timing diagram. Check the RTL schematic.

Follow the same process as before to get your code synthesized. Once again, open the Synthesized RTL Schematic.

4.9 Implementing sequential logic

Modify your constraints file: since you now have connections to `clk` and `reset`, the old file will not be sufficient. Manually add connections for `clk` (must be `W5`) and `reset` (pick a button or a switch to use).

Demonstrate to a TA:
Show your constraints file to a TA to make sure you've assigned the correct pins.

Once verified by a TA, generate a new bit file as before. Download it to the FPGA board.

Demonstrate to a TA:
Show that your switches turn the LEDs on as intended.

You are finished!
Make sure you have demonstrated all of the relevant portions to a TA. Remember to back up your code!