EE316 Digital Logic Design
Department of Electrical and Computer Engineering
University of Texas at Austin

Laboratory #2
EE 316

LABORATORY #2

Programming Logic on the Basys3 FPGA Board

PART 1 - AND Gate
PART 2 - Sprinkler Valve Controller
PART 3 - BCD to 7-segment display

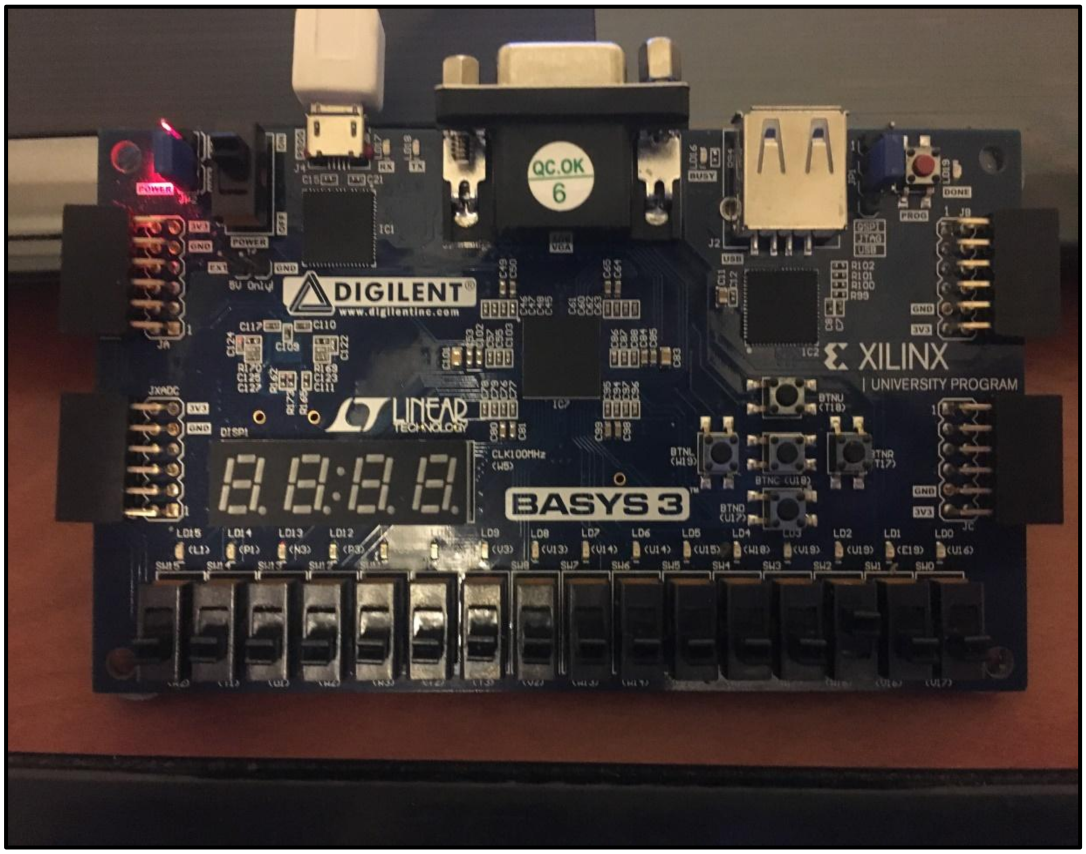


Figure 1. Basys Board

Objectives

Lab 3 contains 3 parts:

- **Part 1**: Guided Design
- **Parts 2 and 3**: Individual Design

The purpose of this lab is to get familiar with:

i. Design and Synthesis using Xilinx Vivado.
ii. Learning Basys3 Board Components and FPGA pin routing.
iii. Understanding of Configuration Files.
iv. Synthesis and Implementation of Combinational Logic Applications on FPGA.
v. Basys3 Board Programming

Equipment

- PC (Laptop)
- Digilent's Basys3 FPGA Evaluation Board

Software

- Xilinx Vivado Design Software Suite

Introduction

We will adhere to the following industry-standard flow for FPGA-based designs.

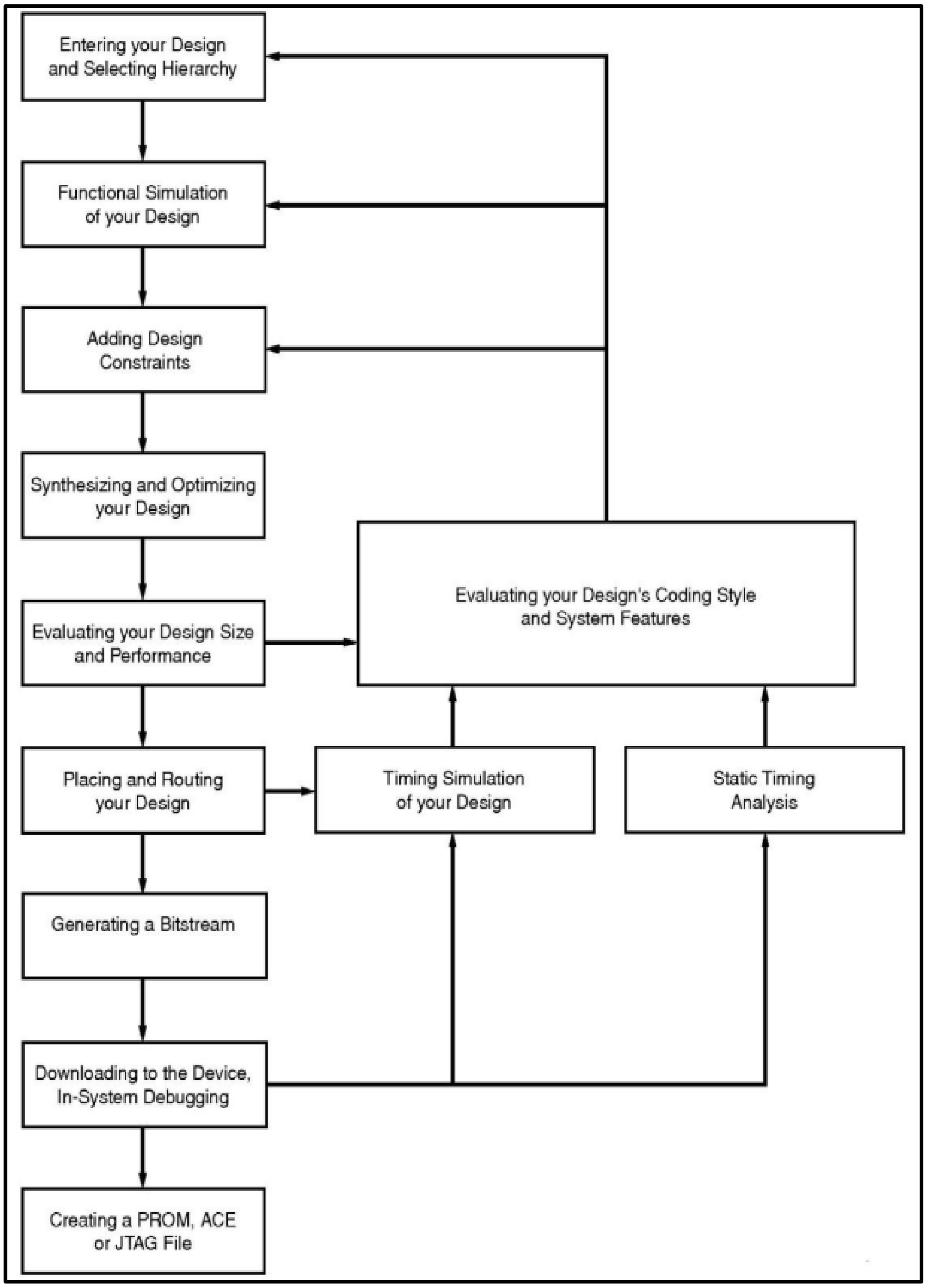
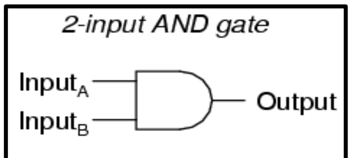

Entering your Design and Selecting Hierarchy
Functional Simulation of your Design
Adding Design Constraints
Synthesizing and Optimizing your Design
Evaluating your Design Size and Performance
Evaluating your Design's Coding Style and System Features
Placing and Routing your Design
Timing Simulation of your Design
Static Timing Analysis
Generating a Bitstream
Downloading to the Device, In-System Debugging
Creating a PROM, ACE or JTAG File

Figure 1. Major steps of the flow for implementing designs in FPGAs.

Part 1. Design, FPGA Synthesis, and Testing of a 2-input AND Gate

In this guided FPGA application development experiment, we will design and test a combinational AND gate on the Digilent's Basys3 Board.

Specification



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Figure 3. AND gate truth table + logic gate

AND gate and its associated truth table are shown in Figure 3. The goal is to realize this gate on the Digilent Basys3 Board. The switches SW0 and SW1 (ON/OFF) correspond to A and B (inputs) in the truth table and Z (output) corresponds to the LED0 (lit UP/turned OFF).

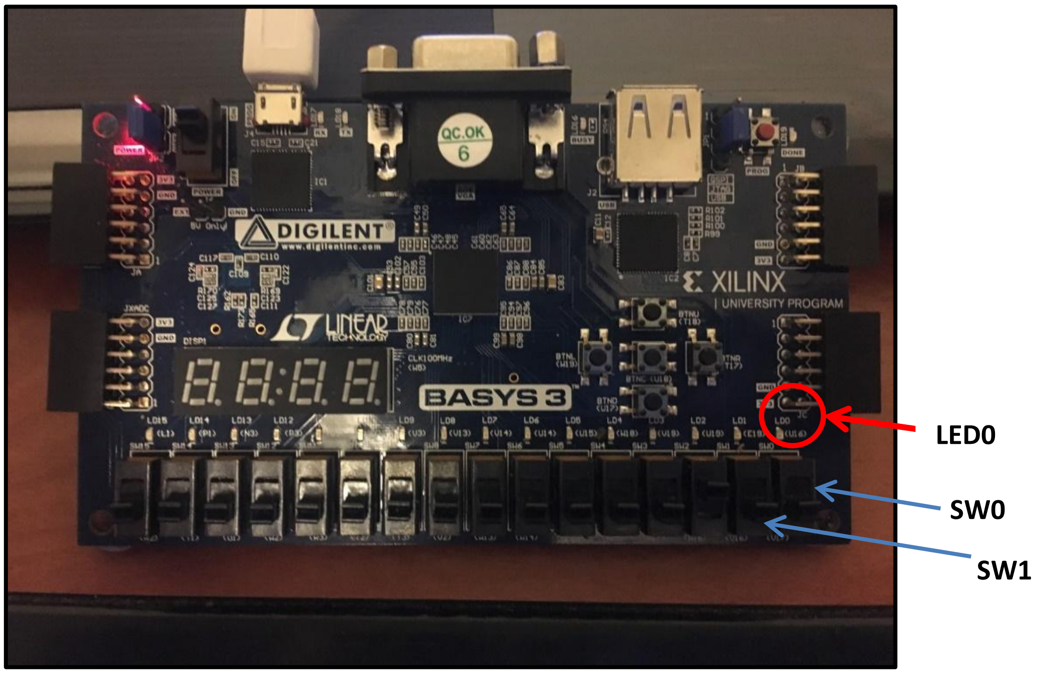


Figure 4. Basys3: Switches (SW0 and SW1) and an LED (LED0)

Procedure

A guided step-by-step procedure is provided for this part. A video tutorial on getting started with Vivado and Basys3 board, provided by Digilent is very useful and you are encouraged to watch it on this link.

Step 1 (Creating a new project) – As done in Lab 2, the first step is to create a new project in Vivado. Go to File -> New Project, to open the wizard for new project creation. The wizard will prompt you to enter Project Name (can be named as Lab_3) and location. Project Type can be selected as RTL Project (and check the box stating Do not specify sources at this time). We will be using Basys3 board in this course, and its part number is *XC7A35TCPG236-1*. Hence, this board needs to be selected in the Project menu. Click on Finish to start the new project. If required, please refer to the Lab 2 manual for a step-by step screenshots of this step.

Basys3 board belongs to the Artix-7 Family, cpg236 package, and its speed grade is -1. The information about this board is also written on a chip in the center of the board, as shown in Figure 5.

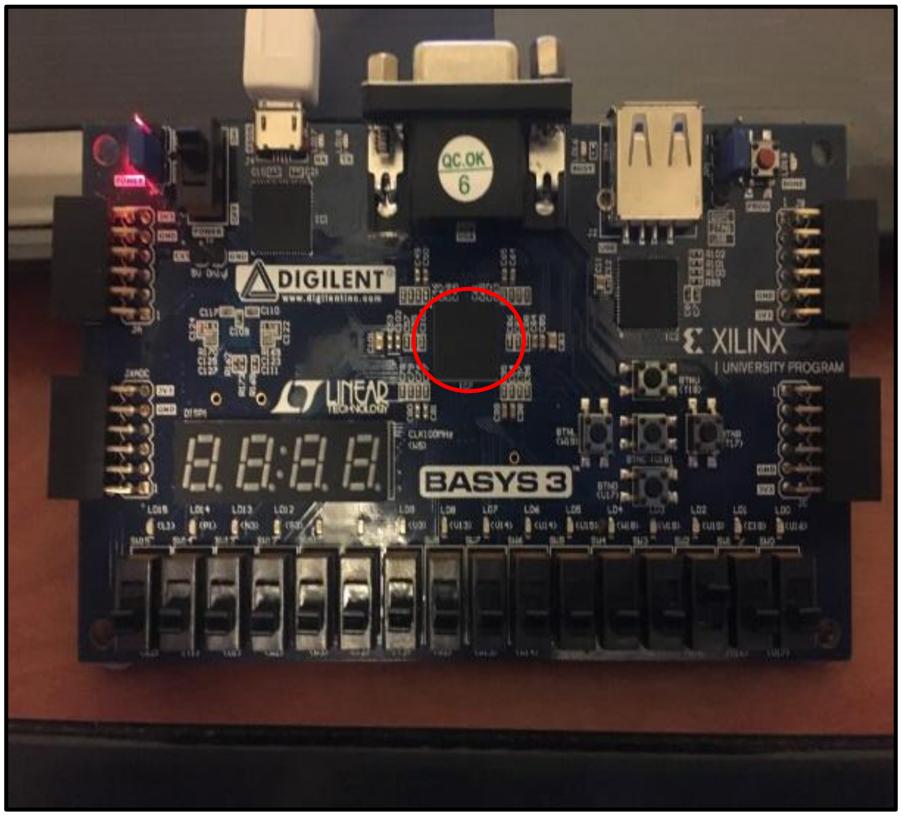


Figure 5. Basys 3 Board Information

Step 2 (Design Entry using Verilog) – As done in Lab 2, add a source file to your project and name it, for example - "myAND". Make sure Verilog is selected as the File type, as shown in Figure 6.

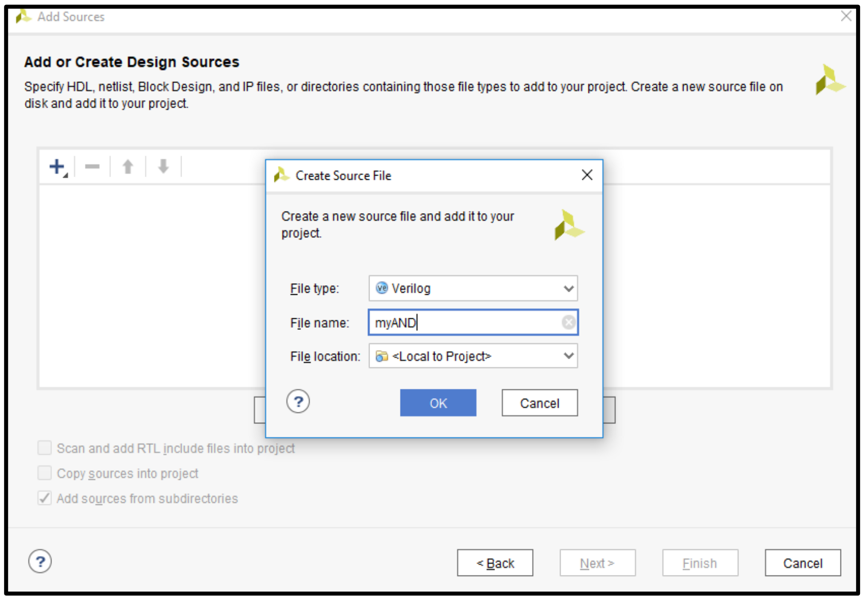
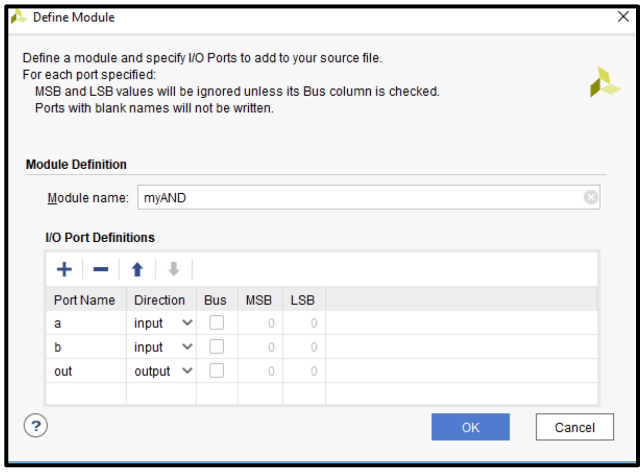

Add Sources
Add or Create Design Sources
Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.
Create Source File
Create a new source file and add it to your project.
File type: Verilog
File name: myAND
File location: <Local to Project>
OK
Cancel
Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
< Back
Next >
Finish
Cancel

Figure 6. Adding source file to the project

For the AND gate design, there are two inputs (a and b) and one output (out).


Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

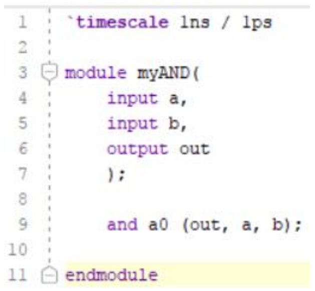
Module name: myAND

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
a	input	☐	0	0
b	input	☐	0	0
out	output	☐	0	0

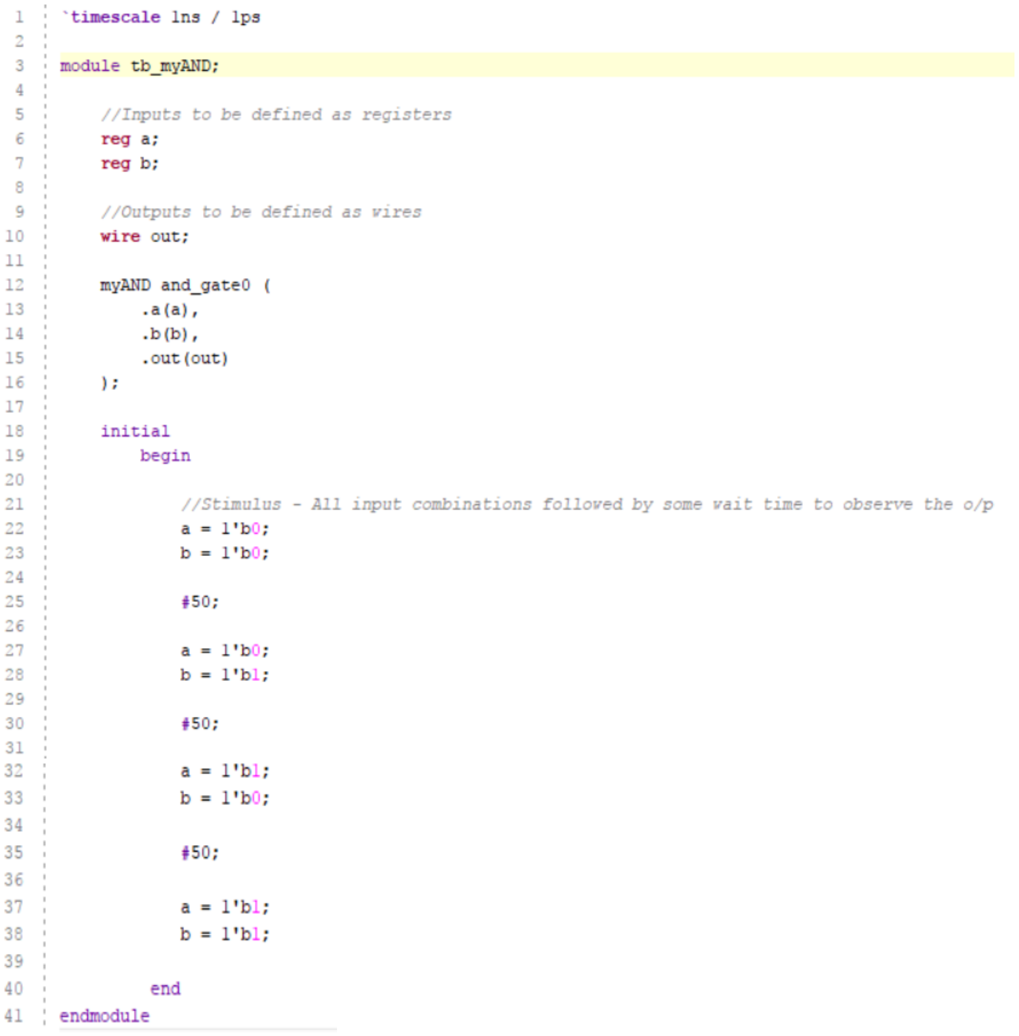
Figure 7. I/O port definition of AND gate

In the generated file *myAND.v*, the Verilog code of the module needs to be written. Since it's just a single 2-input AND gate, the module definition is simple and the code is given below.



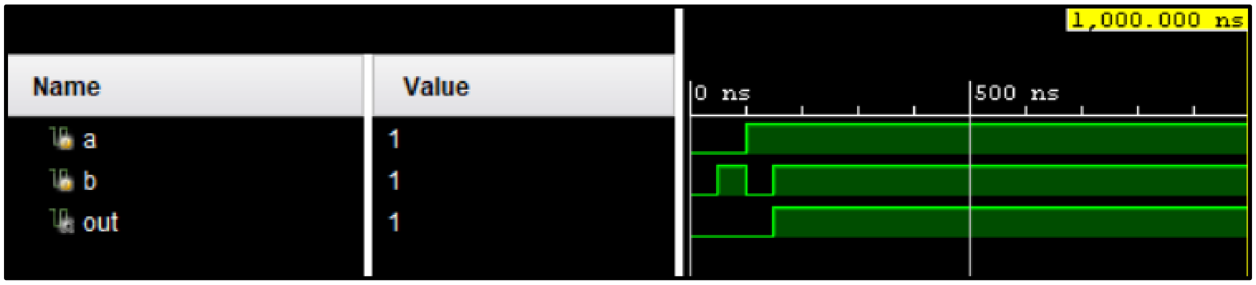
```
`timescale 1ns / 1ps

module myAND(
    input a,
    input b,
    output out
    );

    and a0 (out, a, b);

endmodule
```

For testing the code, a testbench (named, for example – tb_myAND) is required which can be created as a simulation source as done in Lab 2. All the four input combinations will be tested and the code for the testbench is provided below.



```
`timescale 1ns / 1ps

module tb_myAND;

    //Inputs to be defined as registers
    reg a;
    reg b;

    //Outputs to be defined as wires
    wire out;

    myAND and_gate0 (
        .a(a),
        .b(b),
        .out(out)
    );

    initial
        begin

            //Stimulus - All input combinations followed by some wait time to observe the o/p
            a = 1'b0;
            b = 1'b0;

            #50;

            a = 1'b0;
            b = 1'b1;

            #50;

            a = 1'b1;
            b = 1'b0;

            #50;

            a = 1'b1;
            b = 1'b1;

         end
endmodule
```

Step 3 (Circuit Simulation and Verification) – After completing the module and the testbench definitions, the next step is to run the simulation to verify the circuit behavior. Upon running the behavioral simulation as done in Lab 2 and zooming to fit the waveform, it will look like Figure 8.



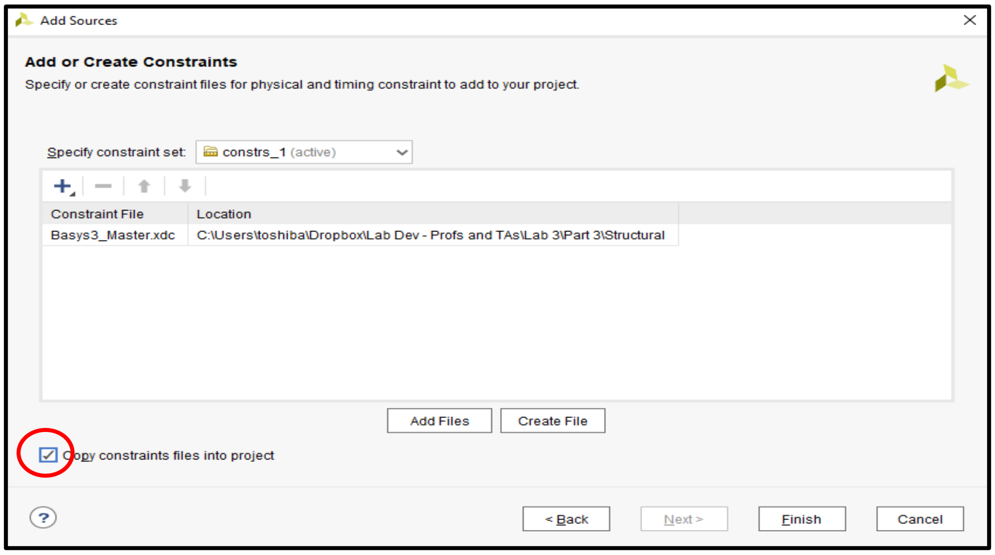
Name	Value
a	1
b	1
out	1

Figure 8. Simulation Waveform for an AND gate

***Step 4 (Adding a Constraints file)* -** Once the code is validated through simulation, the next step is to implement it on FPGA (Basys 3 board for this course).

For FPGA implementation, the first step is to map the inputs/outputs used in the Verilog code to the actual (physical) pins of the FPGA board. This is done using a *constraints file*, which needs to be added to the Vivado project.

Download the master constraint file (Basys3_Master.xdc) from the course's Canvas page under Files → Labs → Basys3 FPGA Board and save it in your project folder. This file contains the definitions of all the ports available on the Basys3 board, and the ports which will be used in the project will be uncommented as shown later. To add this file in the project, go to File → Add sources → Add or create Constraints → Next → Add Files and then select the Basys3_Master.xdc from the location where it is saved. Also, it is a good practice to create a separate copy of constraints file and not change the master file for each project since the number and type of I/Os vary. Hence, ensure that *Copy constraints files into project* option is checked as shown in Figure 9, which makes sure that Vivado creates a copy of the constraints file for that project in the project directory. Click on Finish to add the constraints file to the project.



Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

Constraint File	Location
Basys3_Master.xdc	C:\Users\toshiba\Dropbox\Lab Dev - Profs and TAs\Lab 3\Part 3\Structural

☑ Copy constraints files into project

Figure 9. Add or Create Constraints dialog box

The file will be visible under the "Constraints" tab in the Sources pane, as shown in Figure 10.

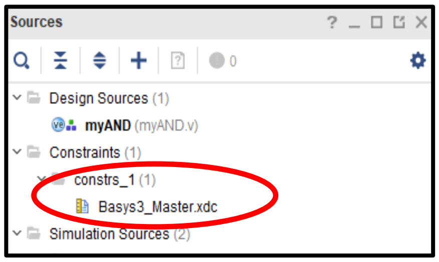
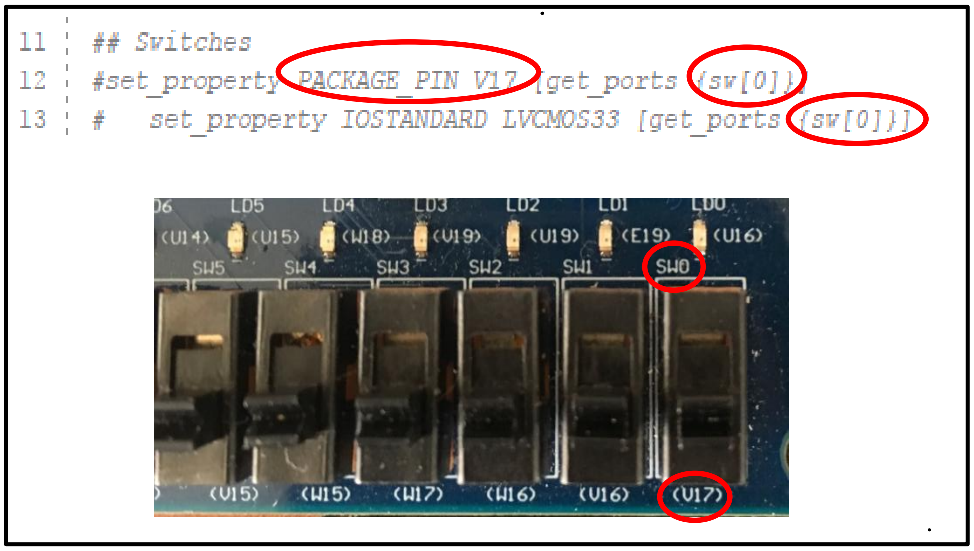


Figure 10. Constraints file in the Sources pane

Double-click on the constraints file to open it, file and familiarize yourself with its contents. Since this file is used to map the I/O of the board to the ports of the code, the file has mapping for all different I/O the Basys board has - like *Switches, LED, 7segment*, etc. For example, in the switches section, Pin V17 corresponds to sw[0]. This can be verified from the board as well, where V17 is printed under SW0, as shown in Figure 11. Essentially, it means that if there is a port sw[0] in the code, it will be mapped to V17 on the board. Hence, it is imperative to take care that the ports in the code are mapped correctly to the I/O resources on the board. Also, since the master file has definitions for all the ports, only the lines corresponding to the required ports should be uncommented and edited and the remaining ones should not be uncommented.



```
## Switches
#set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
```

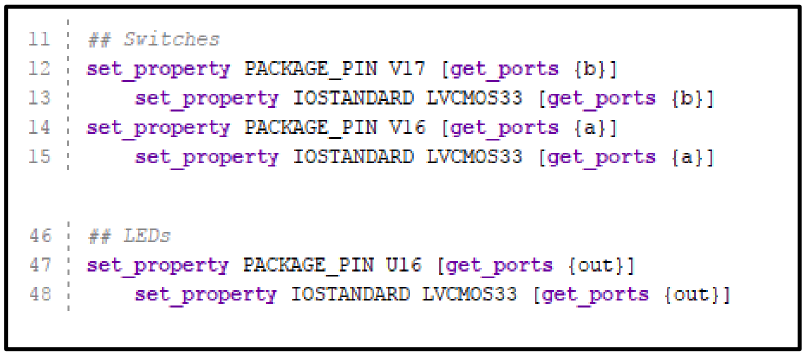
Figure 11. Pin Mapping

Now, the FPGA pins need to be mapped to the corresponding ports defined in the Verilog module.

- Input "a" is to be mapped to SW1, hence to Pin V16. To achieve that, write "*a*" instead of "*sw[1]*" in the constraints file.

- Input “b” is to be mapped SW0, hence to Pin 17. To achieve that, write “*b*” instead of “*sw[0]*” in the constraints file.
- Output “out” is to be mapped to LED0, hence to Pin U16. To achieve that, write “*out*” instead of “*LED[0]*” in the constraints file.

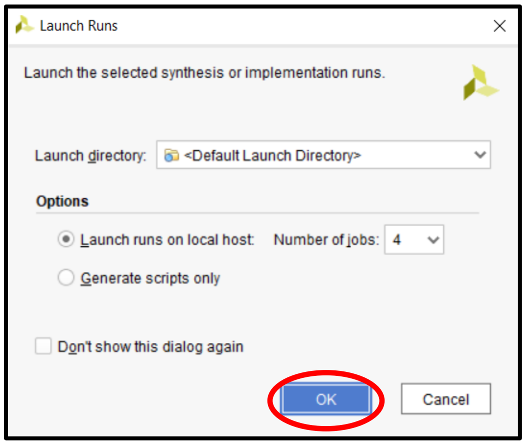
The uncommented lines in the constraints file are shown in Figure 12.



```
## Switches
set_property PACKAGE_PIN V17 [get_ports {b}]
    set_property IOSTANDARD LVCMOS33 [get_ports {b}]
set_property PACKAGE_PIN V16 [get_ports {a}]
    set_property IOSTANDARD LVCMOS33 [get_ports {a}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {out}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out}]
```

Figure 12. Constraints file

Step 5 (Implementation on FPFA board) - After completing the mapping, the next step is to run synthesis. From the left pane, click on Run Synthesis and click OK on the Launch Runs pop-up window.



Launch Runs

Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

Options

● Launch runs on local host: Number of jobs: 4

○ Generate scripts only

☐ Don't show this dialog again

Figure 13. Launch Runs window

Once synthesis is complete, the next step is to Run Implementation. Click on Run Implementation on the left pane or the pop-up window after Synthesis completes and click OK on the Launch Runs pop-up window. It will take about a minute for the Implementation run to be complete.

After Implementation is complete, Bitstream needs to be generated. Click on Generate Bitstream on the left pane or the pop-up window after Implementation completes and click OK on the Launch Runs pop-up window.

After Bitstream generation, make sure the board is connected to the PC and turned on. Click on Open Hardware Manager → Open Target →Auto-Connect. This will be required to do only once until Vivado is not closed and once the board is connected to Vivado, this step can be skipped in the future runs. Finally, to program the device, click on Program Device → Program.

At this point, the design should be mapped onto the FPGA in your board. You should be able to toggle SW0 and SW1. If both are toggled up (which signifies both bits are '1'), then LED0 should turn ON, as expected from the AND gate truth table. This state is also shown in Figure 14.

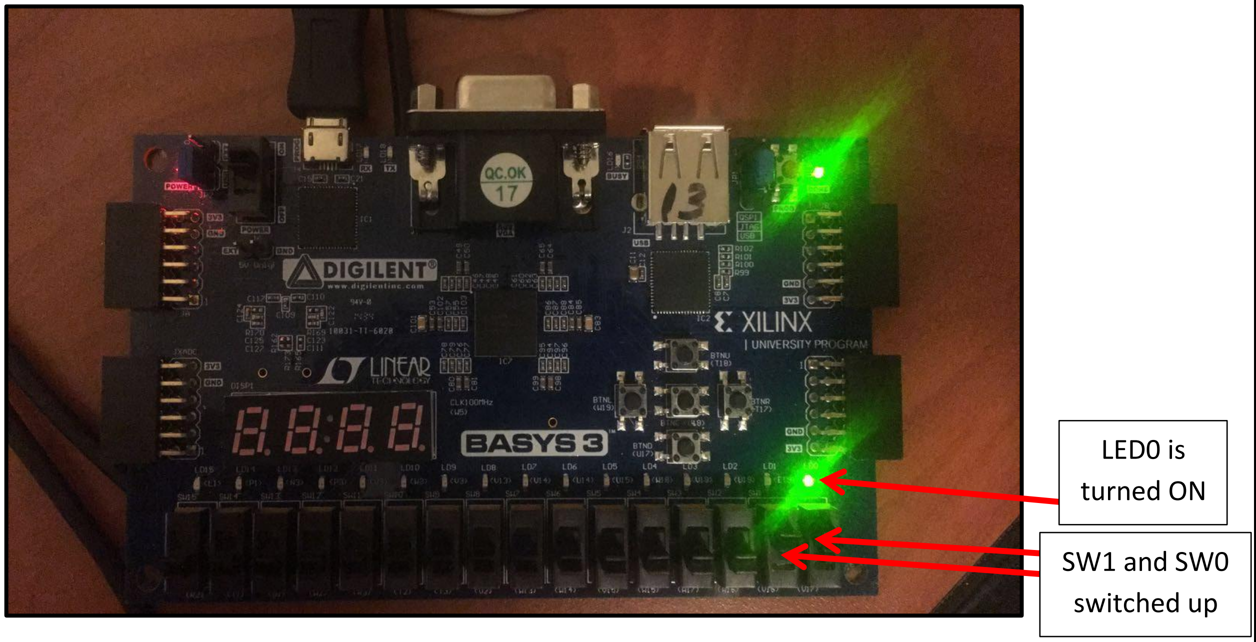


Figure 14. Switches and LED mapping the AND gate functionality

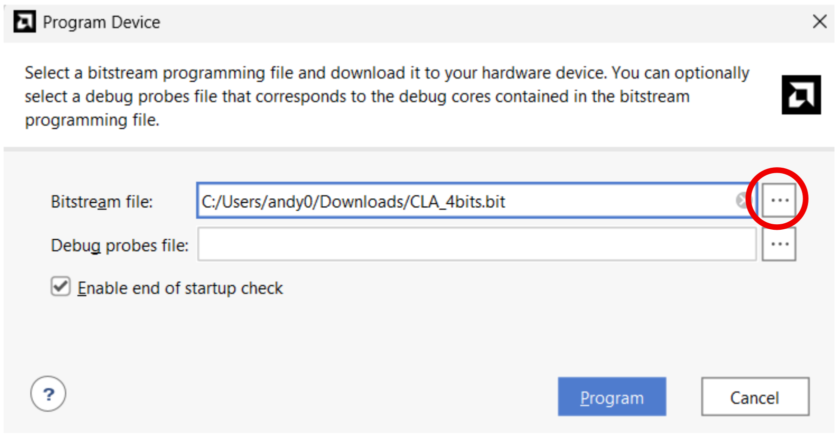
Table 1 summarizes the observations.

SW1	SW0	LED0
Down	Down	OFF
Down	Up	OFF
Up	Down	OFF
Up	Up	ON

Table 1. LED0 as function of SW1 and SW0

Tips for Programming the FPGA - Once the bitstream file has been generated, there's no need to regenerate it each time you want to program the FPGA. You can directly load the existing bitstream into the FPGA using Vivado.

Open the Hardware Manager, connect to your FPGA, and click Program Device. Then click the ... button to browse and select the corresponding .bit file. After selecting the file, click Program to load it onto the FPGA.



Program Device

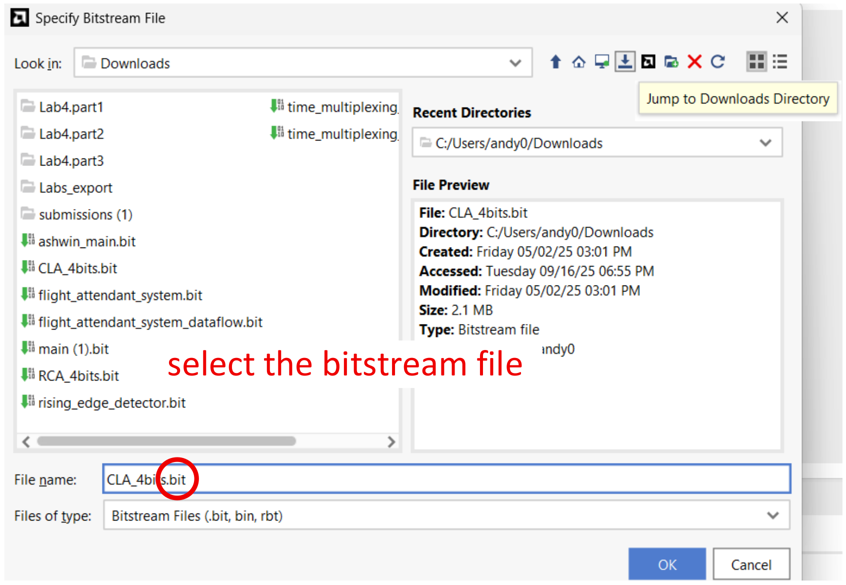
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: C:/Users/andy0/Downloads/CLA_4bits.bit

Debug probes file:

☑ Enable end of startup check

Figure 15. FPGA programming window



Specify Bitstream File

Look in: Downloads

Lab4.part1
Lab4.part2
Lab4.part3
Labs_export
submissions (1)
ashwin_main.bit
CLA_4bits.bit
flight_attendant_system.bit
flight_attendant_system_dataflow.bit
main (1).bit
RCA_4bits.bit
rising_edge_detector.bit
time_multiplexing
time_multiplexing

Recent Directories: C:/Users/andy0/Downloads

File Preview

File: CLA_4bits.bit
Directory: C:/Users/andy0/Downloads
Created: Friday 05/02/25 03:01 PM
Accessed: Tuesday 09/16/25 06:55 PM
Modified: Friday 05/02/25 03:01 PM
Size: 2.1 MB
Type: Bitstream file

File name: CLA_4bits.bit

Files of type: Bitstream Files (.bit, bin, rbt)

Figure 16. Bitstream file selection window

PART 2. Implementation of Sprinkler Controller

Specification

In this part, you are required to implement the Sprinkler Controller system, which was designed in Lab 2, on the Digilent's Basys Board.

1. Use the specifications from Lab 2 regarding the system function. The code from Lab 2 can be re-used.

2. Assume that the on-board LEDs act as sprinkler valves - map each output (d0, d1, ..., d7) to exactly one LED.

3. Control the sprinkler valves using the switches on the board - SW2 = A, SW1 = B, SW0 = C and SW7 = E.

Deliverables and Demonstration

i. Constraints file
ii. The functioning of the decoder should be shown on the Basys3 board as per the specifications.

PART 3. BCD-to-7Segment LED Display

Specification

Part A: Using a truth table, design a gate-level implementation for the BCD-to-7segment decoder, which converts a 4-bit BCD number to its corresponding 7-segment display output. The functioning of the 7-segment display is given in the Hints section below.

Part B: Implement the design of the BCD-to-7segment decoder described above using *structural (gate-level)* Verilog description and implement it on the Basys3 board (Use switches SW [3:0] to control the decimal number displayed on the right-most LED display AN1 on the board).

Hints and Care-bouts

i. A BCD (Binary Coded Decimal) number is to be displayed on the 7-segment display. The highest number that can be displayed is 9. Hence, in terms of binary numbers, 4 bits are needed to map digits 0 → 9 to their binary values "0000", "0001", "0010", "0011", …, "1001" respectively.

   0 → 0000
   1 → 0001
   .
   .
   .
   9 → 1001

   Hence, 4 different inputs (or a vector of size 4) need to be defined in the code.

ii. The 7-segment display is shown in Figure 17.

   Each segment (a, b, …, g) is a different output which needs to be controlled independently. Consider the following examples -

   - To print digit 0 on the 7-segment display, segments a, b, c, d, e, f need to be turned on.
   - To print digit 1, segments b and c need to be turned on.
   - To print digit 3, segments a, b, c, d, g need to be turned on, as shown in Figure 17.

   The total number of outputs that the decoder needs is then 7, or a vector of size 7.

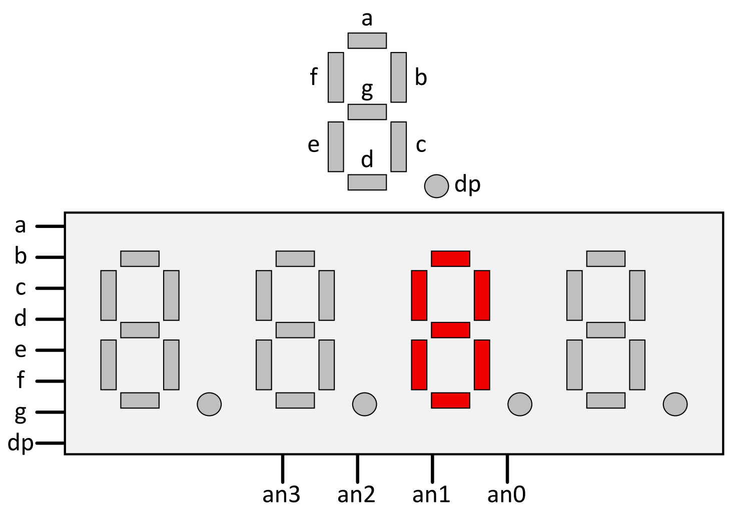


Figure 17. 7-segment Display in Basys3 Board

iii. The segments are active low, i.e., if a segment is required to be turned on, it should be assigned the value 0.

iv. The equations of each of the outputs need to be *minimized* before implementation.
   - In case an input greater than 9 (and less than 16) is selected, turn off the 7–segment display.

v. For this lab, only the second-rightmost 7 segment display, i.e., AN1 will be used. Hence, the other three 7-segment displays need to be turned off. This needs to be done in the Verilog code by following the steps below -
   - Declare 4 outputs AN0, AN1, AN2 and AN3 (each one controls one 7-segment display) in the module.
   - The 7-segment displays AN3, AN2, AN1 and AN0 are active low. Hence, to activate AN1, it should be assigned a value 0, and to deactivate the others, those should be assigned a value 1.

vi. In the constraints file, the section related to the 7-segment displays should be updated. Each segment should be mapped to its corresponding output, i.e., AN0, AN1, AN2 and AN3 in the constraints file need to be updated to match the defined outputs.

Deliverables and Demonstration:

i. Truth table mapping the outputs to the inputs, minimized equations of the outputs and gate-level design of the BCD-to-7segment decoder
ii. Structural Verilog code, testbench and the constraints file
iii. Demonstration of BCD-to-7Segment module functioning on the Basys board