Lab 2: Introduction to VHDL

Mert Örnek EEE 102-3 22302052 23.02.2024

Purpose

The purpose of this lab is for the students to get a grasp of the basic concepts related to VHDL. These include learning the language's syntax rules, useful features the program offers, how the code is relayed to the FPGA board as well as the general purpose the features in Vivado serve.

Methodology

This lab session consisted of five stages, the third and fourth of which required a check from the TA. These tasks were as follows:

-For the first task, I changed the types of three individual logic gates in submodule 1 (lines 14, 15 and 16) based on my university number. My number ends with a "2", therefore they will be "xor", "xnor" and "and" respectively.

- For the second task, I inspected the VHDL code found in the file provided in Moodle and located six individual errors. These errors and their solutions are detailed in the "results" section of this report.

-For the third task, I connected my Basys 3 FPGA to Vivado and embedded the code I just debugged into it. The debugged code -if debugged correctly- would light up the LEDs found on the board in a certain manner. This step required a check from the TA. Further information on how the board functioned can be found in the "results" section.

-For the fourth task, I wrote a testbench code to simulate how the VHDL code I just debugged would act with certain input combinations. More about the testbench can be found in "results".

-For the fifth and final task, I used the "RTL schematic" functions that come pre-loaded with Vivado to visualize how the file I debugged in the second task would function. As requested by the lab instructions, I took pictures of all three and commented on them in the "results" section.

Results

Task 1: The first task was straightforward: I downloaded the VHDL file from the course Moodle, opened it, opened "sub_module1_1" from the design sources tab and replaced the logic function in line 14, 15 and 16 to "xor", "xnor" and "and" consecutively. There isn't much to explain about this task. However, an image of the replaced lines of code can be found in the "Appendices" section regardless (Figure 1).

Task 2: For the second task, I was asked to find 6 individual bugs, show how I identified them, what their fix was as well as the theoretical background for the bug itself. The images related to these bug fixes can be found in the "appendices" section of this report. They are as follows:

Bug 1: Incorrect Specified Part

This bug was the first one I identified because its warning message popped up the moment Vivado loaded up the “LAB2_buggy.xpr” file. The error displayed that -for this project- a different type of FPGA board had been set instead of the default one I set from the settings. I solved this bug by opening “settings” from the left-hand side “project manager” menu and filtering out Vivado’s compatible parts list(Figure 2.2) until I found the variant this project was meant to run.

Bug 2: Output Syntax Error

This next bug was identified when the codes of the individual modules were opened for further inspection. Some lines of code under the “top_module” were underlined in red(Figure 3.1) , indicating some form of irregularity. There was also an error message that popped up under messages after attempting a synthesis: “ ‘s_ouput_1’ is not declared”(Figure 3.2). Two lines were marked, the top one of which (line 36) was identified as the second bug. In this line, in which the assignment operator was linking “o_output_byte” to “s_output_1”, there was a syntax error in the form of “s_output_1” being written as “s_ouput_1”. This error was corrected by correcting the spelling(Figure 3.3). Outside of this case, inputs and outputs for a module can be pre-loaded by adding them in the final stage of creating a new design source from the top left-hand “Add Sources” option. When this is done, inputs and outputs are defined under “entity”. They can also be manually written within that portion by assigning them a name and specifying what they are(ex. Logic vector or std_logic). Other inputs and outputs can be defined under “architecture” if the file contains other inputs and/or outputs necessary for the program. Inputs and outputs for modules can be defined in the “architecture” section of the top module and in the “entity” sections of each submodule. When defining these, a naming convention is generally used with “i_” being put at the beginning of inputs and “o_” for outputs. Signals are also differentiated by “s_”.

Bug 3: Port Map Sequence Error

Alongside bug 2, there was another line (line 40) marked by Vivado, this one underlining the port map section of “sub_module2_1”(Figure 3.1). This error was also reported in the messages section, with the error message of “ ‘i_sw’ with mode ‘in’ cannot be updated”(Figure 3.3). This had to do with the sequence of the two ports it included. In the port map, the input ports are supposed to be defined before the output ports. However, as can be seen in the provided image, port map was defined with the “i_SW” input defined after “s_output_2”. This issue was fixed by replacing the sequence of the ports (Figure 3.4). Port maps are used for defining individual inputs, outputs and signals used within a module/submodule and/or to connect the ones used within the module to architectural ones by using the “=>” operator. In port maps, inputs come before outputs and -when using the “=>” operator- inputs/outputs/signals used within a module are put on the left and the architecture value they are connected to on the right. This also allows for the usage of submodules within a module of higher level as the port map allows for the relation and connection of different variables found within individual modules.

Bug 4: Board Constraint Syntax Error

After the first three bugs were resolved, the program was synthesized with no issues. However, once I attempted to implement the program, it failed and new errors popped up. One of these errors was linked to line 28 of the constraints file(Figure 4.1). Once I inspected it closer, I noticed that one of the parenthesis necessary for the syntax of “o_LED[2]” was missing. I resolved this bug by adding the parenthesis so that that line of code constraining pin U19 looked like the others(Figure 4.2). Constraint files are necessary to connect the program to a physical logic device, such as our FPGA. The constraint file maps the names of inputs/outputs the program uses to specific pins found on the physical device. It also sets constraints on how the program will be enacted by the device, such as the I/O constraints, operating voltage as well as timing and placement constraints. To map specific inputs/outputs to pins, a constraint file must be added to the project which specifies the program input/outputs, targeted pins as well as all the necessary constraints. This file -alongside the rest of the project- is transferred to the programmable FPGA via bitstream which allows the logic circuit to relate the necessary signals to the assigned pins which allows the programmable FPGA to physically function.

Bug 5: Disconnected Sub Module

Once I fixed the fourth bug, I cleared errors and reattempted to implement the project. This time, I received a warning that the program “could not resolve non-primitive black box cell ‘sub_module2’”(Figure 5.1). Upon inspection, I noticed that -on the top left hand side sources menu- “sub_module2_1” was not connected to a higher module. This was due to the way sub module 2 was defined in “sub_module2.vhd”, in which it was referred to as “sub_module2_the_beast”. The issue was caused by this definition, as the “sub_module2_1” defined in the top module could not get the data it needed from the module above it in the hierarchy. I resolved this bug by replacing the name of sub module 2 simply as “sub_module2”(Figure 5.2). After this correction, the program implementation was completed without failure.

Bug 6: Incorrect I/O Ports

This bug was the hardest one to notice as it was not warned through conventional error messages. It was instead discovered after running the testbench and comparing it to Basys 3’s LED output. For instance, when an input vector of “00000010” was specified, the testbench would specify an output of “11011100”(Figure 8) whereas the Basys 3 FPGA would return an output of “01011101”(Figure 6.1). Upon re-inspecting the constraints file of the project, I discovered that LED[0] and LED [7] had incorrect specified pin codes(Figure 6.2). Normally, LED[0] on Basys 3 has a pin code of U16 and LED[7] is linked to V14 pin. However, it was specified the other way round. This bug was fixed by switching the specified pins of both constraints(Figure 6.3). After this was done, the outputs of the Basys matched(Figure 7.3) the ones from the testbench.

Task 3: The third task required that I connect my Basys 3 FPGA to Vivado and test run the program I just debugged. This step also required a check from the TA. I specified 6 different inputs on the Basys 3 FPGA. These inputs and their corresponding outputs are as follows: input “00000000”, output “11000010”(Figure 7.1). input “00000001”, output “11011100”(Figure 7.2). input “00000010”, output "11011100”(Figure 7.3). input “00000011”, output "11000010”(Figure 7.4). input “00000100”, output "11000000”(Figure 7.5). input “00000101”, output "11000010”(Figure 7.6).

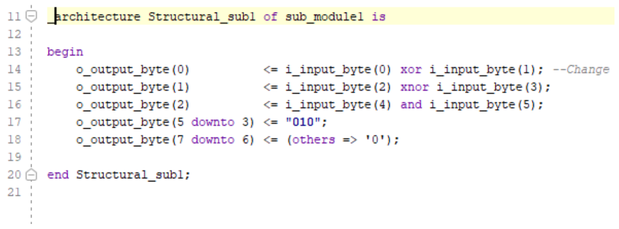
Task 4: The fourth task required that I write a testbench code and compare it to the results I obtained from task 3. I wrote the testbench code by first adding another simulation source to the project. I wrote the testbench code that can be found at the end of the appendices section of this report. The code increases the “i_SW” input vector’s values every 10 ns by using a for loop configuration. This loop adds 1 to the previous input vector value by converting it to unsigned and reverting it back to a vector once it’s done. Conversion to unsigned was necessary as -in VHDL- logic vectors cannot perform arithmetic calculations. The for loop repeats 256 times, as there are 2^8 possible inputs for the 8-bit “i_SW” logic vector. To fit in all 256 configurations, I needed to readjust the simulation runtime from a pre-loaded 1000ns to 2560ns. With this setup, I ran the behavioral simulation and it displayed all 256 configurations(Figure 8). I checked the input/output values obtained from Basys 3 FPGA in task 3(Figure 7.1 to 7.6) and didn’t obtain any inconsistencies in its final and reported form. Testbenches are useful for digitally testing the logic program’s corresponding inputs/outputs before transferring it to a physical device. It allows us to see every single possible signal combination and how the circuit reacts to every possible one. This saves a lot of time on the programmer’s part as for larger circuits it is near impossible to physically test every single combination and it allows for quicker correction and/or adjustments if an issue should arise.

Task 5: For the fifth and final task, I was to use the “schematic” option under “RTL Analysis”, “Synthesis” and “Implementation” bars and compare the three schematics obtained from each. Upon first glance, schematics generated by “Synthesis”(Figure 9.1) and “Implementation”(Figure 9.2) were identical in appearance outside of the implemented schematic having more options bars under the schematic, such as “Timing”, “Power”, ”DRC”, “Package Pins” and “I/O Ports”. Compared to these two, it is easier to understand what is going on in the “RTL Analysis” schematic(Figure 9.3) because it shows the program’s implemented logic gates in more detail. It initially shows the general layout of the project with the option of being able to expand submodules(Figure 9.4) and see their internal workings too. These three schematic functions -in general- show us how the physical logic circuit implementation of our code would be. It also shows where each input or signal goes in, what operation it goes through, what output comes out as well as where defined submodules go. Seeing the physical manifestation of the project makes it easier to envision what operations the program is performing as a whole. This schematic also helps us draw certain conclusions regarding the simulation as the simulation’s job is to show how the code reacts to certain stimuli and the schematic shows how that result was reached. For instance, the “Synthesis” and “Implementation”(Figure 9.1 and 9.2) schematics show that LEDs 5,6 and 7 are independent of the rest of the logic function. This confirms what the simulation(Figure 8) shows, which is the phenomenon of LED 5, 6 and 7 having constant “0”, “1” and “1” values respectively under every possible input combination. As for the “RTL Analysis”s schematics, they explain the values LEDs 1,2,3 and 4 took on during the simulation via logic gate symbols.

Conclusion

This experiment was done with the intention of introducing the student to VHDL and Vivado along with their basic features. This includes learning basic levels of VHDL to fix the bugs and write a testbench, learning Vivado functionalities such as schematics and simulations as well as learning how to physically transfer VHDL code from Vivado to a Basys 3 device. This experiment likely doesn’t have much in the way of errors due to there being a minimal number of possible inconsistencies due to the lack of necessity for any quantitative measurement. If there were to be errors related to the results obtained, it would likely be due to factors related to digital equipment such as the computer this lab was conducted on as well as issues relating to the software and hardware of the Basys 3 FPGA itself. The latter possibility is present due to me having bought the device second hand which means it might be showing signs of age as I don’t know how long it has been around. On the whole, this experiment was successful as I met all the aforementioned objectives of this lab session. One thing that I struggled with was the lack of available material for this experiment. While we were provided with four tutorial sheets explaining certain skills and information related to the experiment, they were lackluster as they barely mentioned anything about coding in VHDL, the properties of the language as well as how all of the necessary functions of Vivado were to be used. This meant that I had to find other means of obtaining and mastering this information. While I managed to pull through in the end, the process ended up being more challenging and time-consuming than it had to be. One possible solution for this might be expanding on some of the original tutorial sheets as well as preparing a new and comprehensive guide on how to code in VHDL.

Appendices

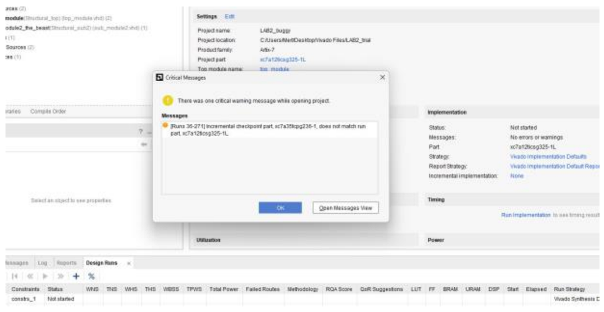


```
architecture Structural_sub1 of sub_module1 is

begin
    o_output_byte(0)          <= i_input_byte(0) xor i_input_byte(1); --Change
    o_output_byte(1)          <= i_input_byte(2) xnor i_input_byte(3);
    o_output_byte(2)          <= i_input_byte(4) and i_input_byte(5);
    o_output_byte(5 downto 3) <= "010";
    o_output_byte(7 downto 6) <= (others => '0');

end Structural_sub1;
```

Figure 1: Replaced logic gates



Critical Messages

There was one critical warning message while opening project.

Messages

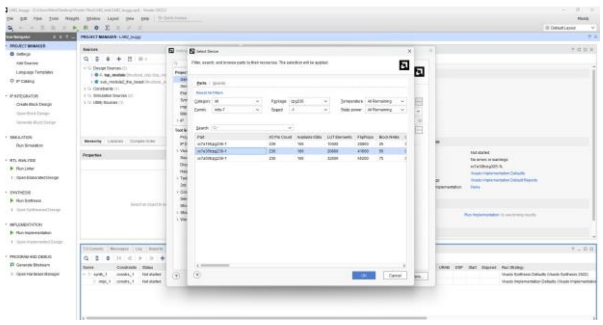
[Runs 36-271] Incremental checkpoint part, xc7a35ticpg236-1, does not match run part, xc7a12ticsg325-1L.

Settings	
Project name:	LAB2_buggy
Project location:	C:/Users/Mert/Desktop/Vivado Files/LAB2_trial
Product family:	Artix-7
Project part:	xc7a12ticsg325-1L
Top module name:	top_module

Implementation	
Status:	Not started
Messages:	No errors or warnings
Part:	xc7a12ticsg325-1L
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Repo
Incremental implementation:	None

Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy
constrs_1	Not started																			Vivado Synthesis D

Figure 2.1: FPGA Error Message



Select Device

Filter, search, and browse parts by their resources. The selection will be applied.

Category: All
Family: Artix-7
Package: cpg236
Speed: -1
Temperature: All Remaining
Static power: All Remaining

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs
xc7a15ticpg236-1	236	106	10400	20800	25
xc7a35ticpg236-1	236	106	20800	41600	50
xc7a50ticpg236-1	236	106	32600	65200	75

Figure 2.2: Filtering compatible device list

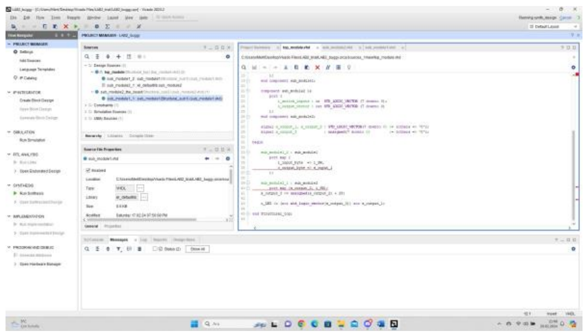
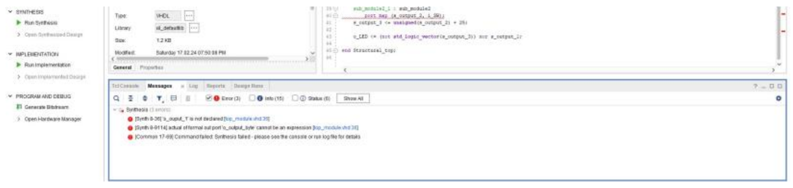


Figure 3.1: Underlined code



Type:	VHDL
Library:	xil_defaultlib
Size:	1.2 KB
Modified:	Saturday 17.02.24 07:50:08 PM

```
    sub_module2_1 : sub_module2
        port map (s_output_2, i_SW);
    s_output_3 <= unsigned(s_output_2) + 25;

    o_LED <= (not std_logic_vector(s_output_3)) xor s_output_2;

end Structural_top;
```

Error (3) Info (15) Status (6)

```
Synthesis (3 errors)
  [Synth 8-36] 'o_ouput_1' is not declared [top_module.vhd:36]
  [Synth 8-9114] actual of formal out port 'o_output_byte' cannot be an expression [top_module.vhd:36]
  [Common 17-69] Command failed: Synthesis failed - please see the console or run log file for details
```

Figure 3.2: Error messages post synthesis

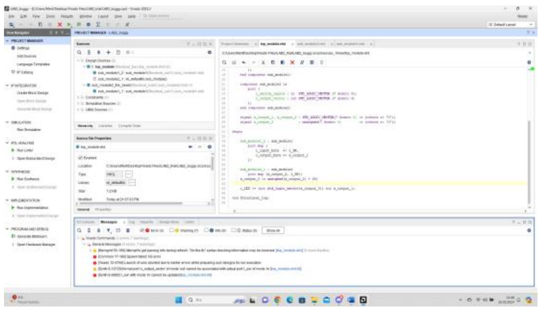


Figure 3.3: Corrected Syntax/ Error Messages

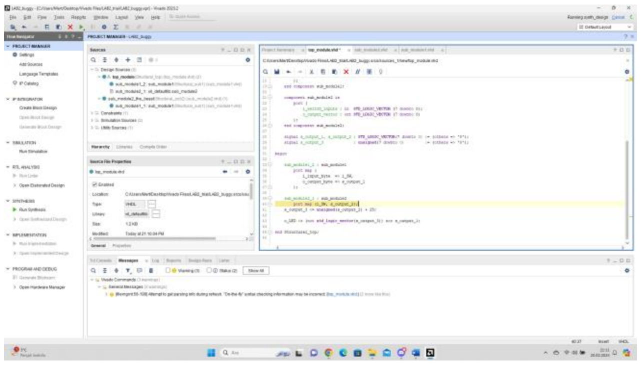


Figure 3.4: Fixed port map

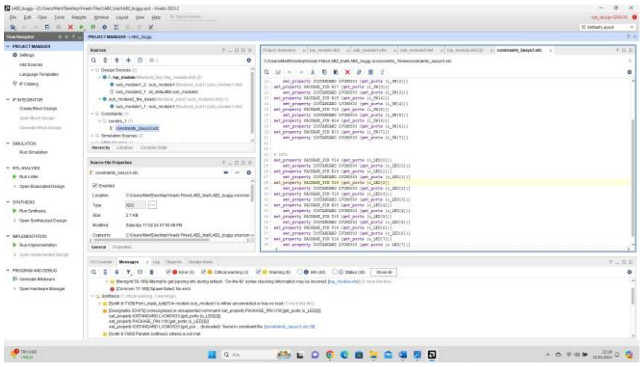


Figure 4.1: Constraints file syntax error

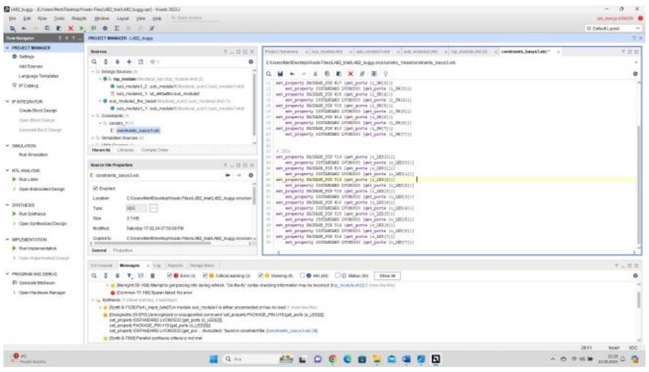


Figure 4.2: Fixed constraints file

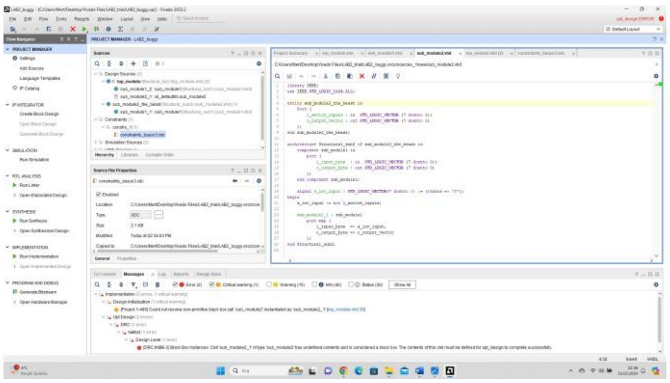


Figure 5.1: Black box error and sub module 2

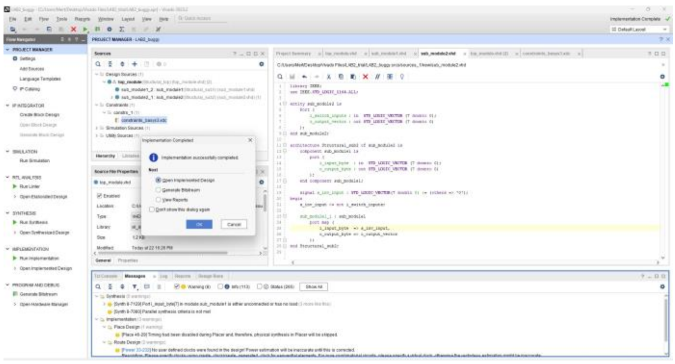


Figure 5.2: Implemented design and changed module names

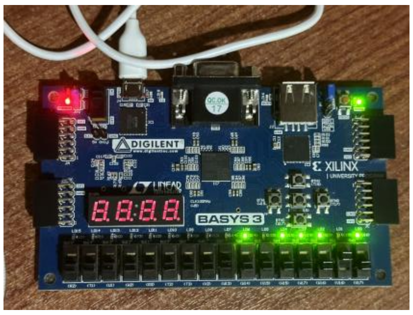
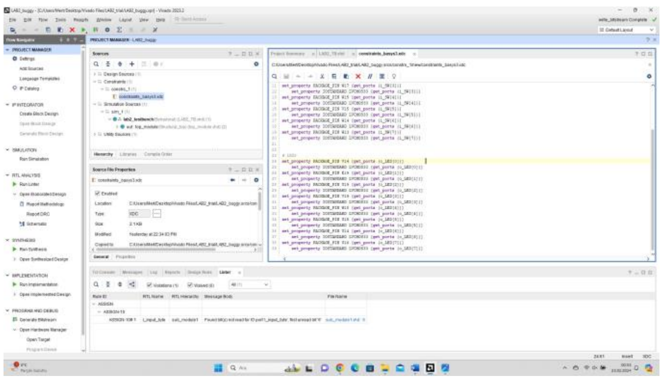


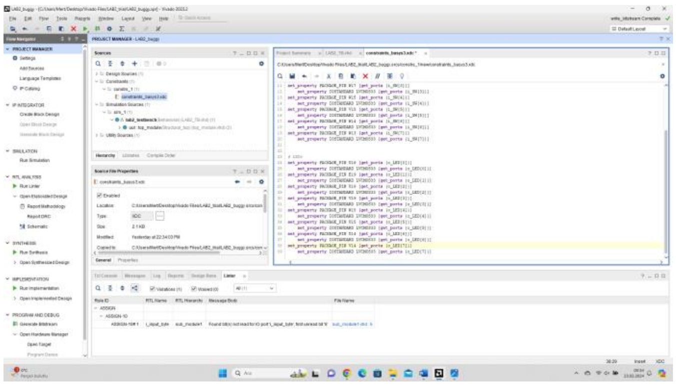
Figure 6.1: Basys with incorrect constraints



```
set_property PACKAGE_PIN W17 [get_ports {i_SW[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[3]}]
set_property PACKAGE_PIN W15 [get_ports {i_SW[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[4]}]
set_property PACKAGE_PIN V15 [get_ports {i_SW[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[5]}]
set_property PACKAGE_PIN W14 [get_ports {i_SW[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[6]}]
set_property PACKAGE_PIN W13 [get_ports {i_SW[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[7]}]


# LEDs
set_property PACKAGE_PIN V14 [get_ports {o_LED[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[0]}]
set_property PACKAGE_PIN E19 [get_ports {o_LED[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[1]}]
set_property PACKAGE_PIN U19 [get_ports {o_LED[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[2]}]
set_property PACKAGE_PIN V19 [get_ports {o_LED[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[3]}]
set_property PACKAGE_PIN W18 [get_ports {o_LED[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[4]}]
set_property PACKAGE_PIN U15 [get_ports {o_LED[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[5]}]
set_property PACKAGE_PIN U14 [get_ports {o_LED[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[6]}]
set_property PACKAGE_PIN U16 [get_ports {o_LED[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[7]}]
```

Figure 6.2: Incorrect pin specifications



```
set_property PACKAGE_PIN W17 [get_ports {i_SW[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[3]}]
set_property PACKAGE_PIN W15 [get_ports {i_SW[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[4]}]
set_property PACKAGE_PIN V15 [get_ports {i_SW[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[5]}]
set_property PACKAGE_PIN W14 [get_ports {i_SW[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[6]}]
set_property PACKAGE_PIN W13 [get_ports {i_SW[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[7]}]


# LEDs
set_property PACKAGE_PIN U16 [get_ports {o_LED[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[0]}]
set_property PACKAGE_PIN E19 [get_ports {o_LED[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[1]}]
set_property PACKAGE_PIN U19 [get_ports {o_LED[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[2]}]
set_property PACKAGE_PIN V19 [get_ports {o_LED[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[3]}]
set_property PACKAGE_PIN W18 [get_ports {o_LED[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[4]}]
set_property PACKAGE_PIN U15 [get_ports {o_LED[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[5]}]
set_property PACKAGE_PIN U14 [get_ports {o_LED[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[6]}]
set_property PACKAGE_PIN V14 [get_ports {o_LED[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[7]}]
```

Figure 6.3: Fixed pin specifications

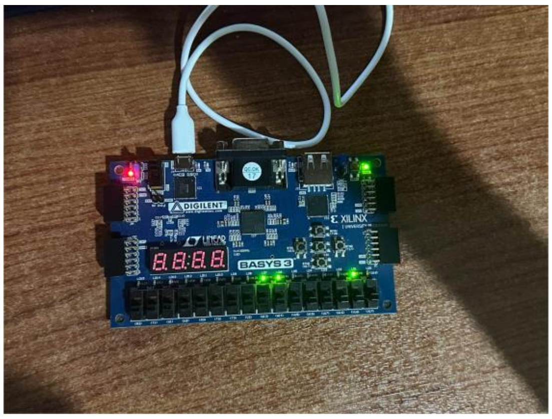


Figure 7.1: First input/output

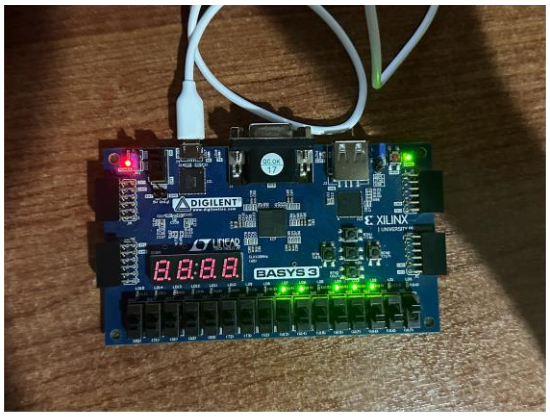


Figure 7.2: Second input/output

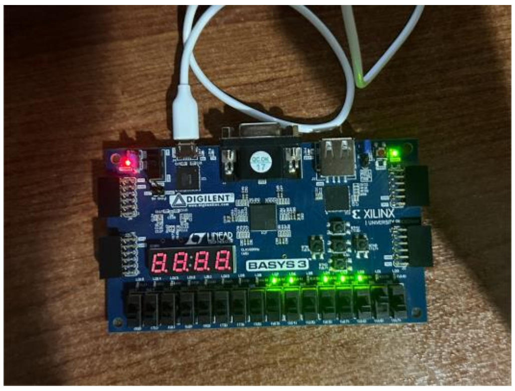


Figure 7.3: Third input/output

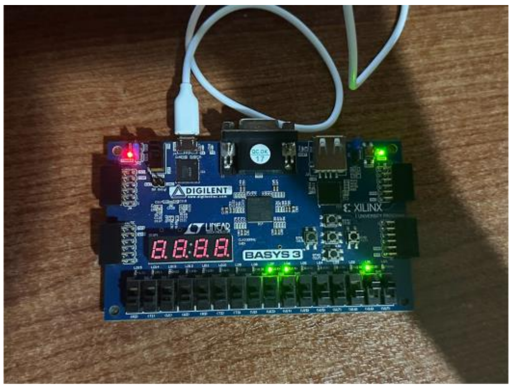


Figure 7.4: Fourth input/output

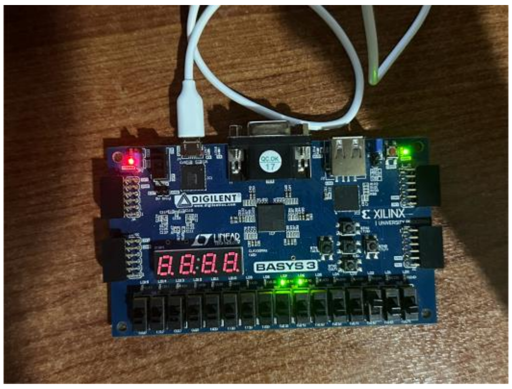


Figure 7.5: Fifth input/output

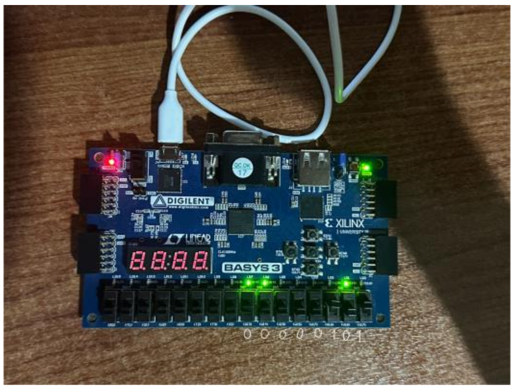


Figure 7.6: Sixth input/output

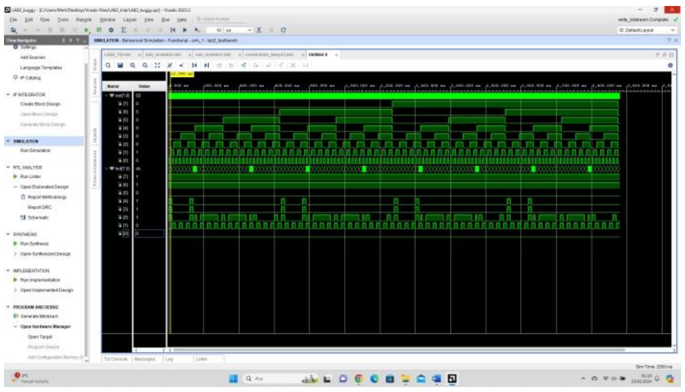


Figure 8: Simulated testbench with 10 ns intervals

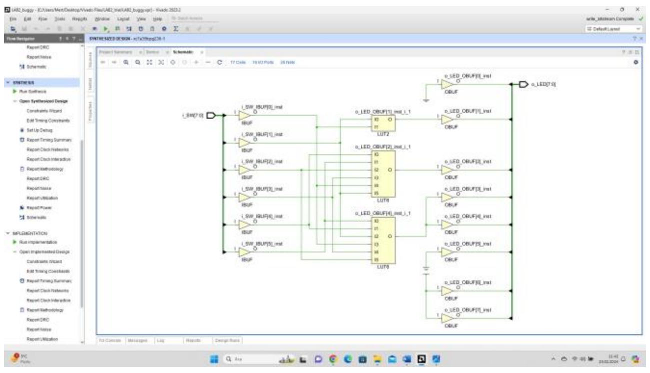


Figure 9.1: Synthesized Design

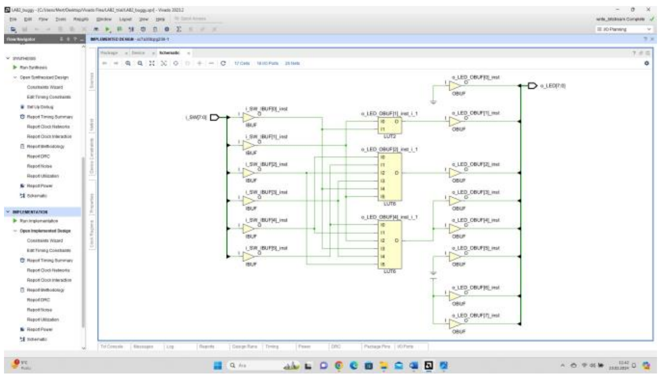


Figure 9.2: Implemented Design

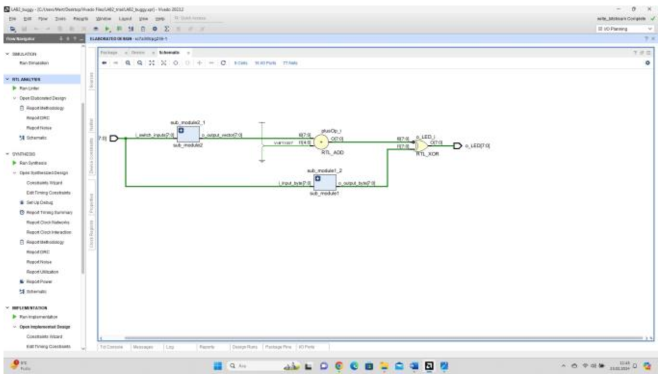


Figure 9.3: Elaborated Design(RTL Analysis)

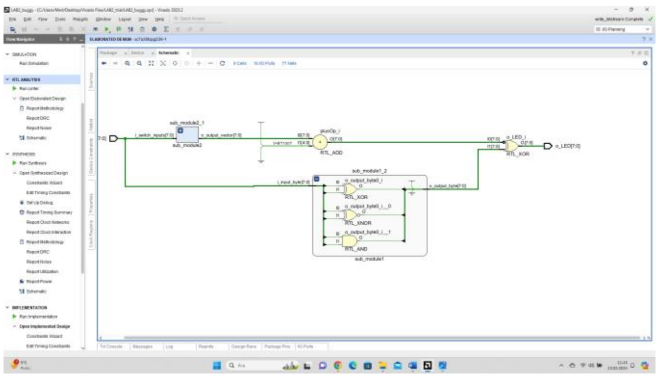


Figure 9.4: Elaborated Design with submodule1_2 expanded

Replaced Code:

top_module.vhd :

```
        o_output_byte => s_output_1              (line 36)
    port map (i_SW, s_output_2);                 (line 40)
```

constraints_basys3.xdc :

```
set_property PACKAGE_PIN U16 [get_ports {o_LED[0]}]        (line 24)
set_property PACKAGE_PIN U19 [get_ports {o_LED[2]}]        (line 28)
set_property PACKAGE_PIN V14 [get_ports {o_LED[7]}]        (line 38)
```

sub_module1.vhd :

```
        o_output_byte(0)      <= i_input_byte(0) xor i_input_byte(1);       (line 14)
        o_output_byte(1)      <= i_input_byte(2) xnor i_input_byte(3);      (line 15)
        o_output_byte(2)        <= i_input_byte(4) and i_input_byte(5);          (line 16)
```

sub_module2.vhd :

```
entity sub_module2 is                    (line4)
end sub_module2;                         (line 9)
architecture Structural_sub2 of sub_module2 is        (line 11)
```

VHDL Testbench Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity lab2_testbench is

end lab2_testbench;

architecture Behavioral of lab2_testbench is

signal sw : std_logic_vector(7 downto 0) := (others => '0');
signal led: std_logic_vector(7 downto 0);

begin
   uut:entity work.top_module
   port map(
   i_SW => sw,
   o_LED => led);

   stim_proc: process
   begin
      for i in 0 to 255 loop
         sw <= std_logic_vector(unsigned(sw)+1);
         wait for 10 ns;
```

```
        end loop;

        wait;

    end process stim_proc;


end Behavioral;
```