Ada Zağyapan 22401942

EEE102-02

19 February 2025

LAB-2: Introduction To VHDL

PURPOSE

This lab's purpose is to design combinational circuits using VHDL and implement them on the Basys3 FPGA. It is aimed to gain hands-on experience with circuit debugging, testbench creation, and simulation testing using Vivado.

BACKGROUND INFORMATION

VHDL: VHDL stands for VHSIC (Very High-Speed Integrated Circuit) Hardware Description Language. It's a hardware description language used to design and simulate digital systems before actual hardware implementation. Engineers use VHDL to describe the behavior and structure of electronic systems via testing and validation in a simulated environment. This process helps in identifying and fixing electronic and logic design issues early[1].

Basys3: The Basys 3 is a development board that features the Xilinx Artix-7 FPGA. It's tailored for beginners and educational purposes. It provides a hands-on platform to learn and experiment with digital design. The board comes equipped with various peripherals, including switches, LEDs, and input/output ports. The users can design various digital circuits from basic combinational circuits to complex circuits including controllers and processors.[2]

---

[1] Hands, J. P. (1990). What is VHDL? *Computer-Aided Design*, *22*(4), 246–249. https://doi.org/10.1016/0010-4485(90)90054-G

[2] *Basys 3 - Digilent Reference*. (n.d.). Retrieved February 18, 2025, from https://digilent.com/reference/programmable-logic/basys-3/start

Vivado Design Suite: Vivado is Xilinx's software suite for FPGA design and development. It offers a range of tools for creating, simulating, and deploying digital designs onto FPGA hardware. With Vivado, designers can write VHDL code, perform simulations to verify functionality, synthesize designs into hardware-compatible formats, and program them onto FPGA boards like the Basys 3.[3]

DESIGN SPECIFICATIONS

top_module: The i_SW signal serves as the primary input, controlling which switch is turned on or off. It acts as the main source of input data for both sub_module1 and sub_module2. The final output, o_LED, determines the LED state based on the processed results from these submodules.

i_SW and o_LED are of type std_logic_vector, hence, they are multi-bit signals that enable parallel data processing. The input signal i_SW is processed separately by sub_module1 and sub_module2, producing s_output_1 and s_output_2, respectively. The output from sub_module2 is then converted to an unsigned type and incremented by 25. This conversion from std_logic_vector to unsigned allows for mathematical operations to be performed on the data.

sub_module1: This module is a component within the top_module. To define it as a submodule, sub_module1 was instantiated within the top_module, and its input and output ports were properly declared.

---

[3] *Vivado Overview*. (n.d.). Retrieved February 18, 2025, from https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html

The input signal i_input_byte is a std_logic_vector type. Since the last digit of my ID is 2, the logic gates used in this submodule include XOR, XNOR, and AND. The output signal o_output_byte computed with these gates is later used in further operations.

sub_module2: Similar to sub_module1, this module is also a component of the top_module. The input signal, i_switch_inputs, is of type std_logic_vector and undergoes processing within sub_module2. The resulting output, o_output_vector, is converted to an unsigned type, incremented by 25, and stored in s_output_3. This value is then negated with the NOT operation and XORed with s_output_1 from sub_module1. Hence, the final value of o_LED is computed.

QUESTIONS

1) In VHDL, the module's interface is defined within an entity declaration. This declaration includes a port clause, where each signal is described by its name, direction (such as in, out, or inout), and data type. With this method, the module's inputs and outputs are clearly defined.[4] For example:

```
entity ModuleName is
  port(
    input_signal  : in  std_logic; -- declares input_signal as an input
    output_signal : out std_logic  -- declares output_signal as an output
  );
end ModuleName;
```

2) To integrate one module within another, the user should use component instantiation. In the higher-level module, the user first declares the component and then includes it by

[4] Perry, D. L. . (2002). *VHDL : programming by example Douglas L. Perry*. McGraw-Hill Education.

mapping its ports to internal signals using the PORT MAP statement. The PORT MAP command connects the submodule's interface to the signals of the higher-level module.[5] For example:

```
port map(
    sub_input  => input_signal, -- connects input internal_signal to sub_input
    sub_output => output_signal    -- connects input output_signal to sub_output
  );
```

3) A constraint file is basically a bridge between the VHDL code and the physical FPGA hardware. It specifies the mapping of logical signals defined in the code to specific physical pins and parameters on the FPGA. Thus, when the design is implemented, each signal is correctly assigned to the intended hardware pin and the hardware's interface works properly.[6]
4) A testbench is a VHDL module created only to perform a simulation. It maintains an environment where stimuli intended in the code are applied to the design under test (DUT), and the responses are observed and compared against expected behavior. This is a process for verifying that the design functions correctly before it is synthesized and implemented on hardware and is important for catching errors beforehand.[7]

PART 1

Methodology: For this part, the first step was to modify a part of the given code according to my ID number. I modified sub_module1.vhd, specifically lines 14, 15, and 16. Since the last digit of

[5] Ashenden, P. J. (1996). *The Designer's Guide to VHDL (Third Edition)*.

[6] *What is a Constraints File? - Digilent Reference*. (n.d.). Retrieved February 18, 2025, from https://digilent.com/reference/programmable-logic/guides/vivado-xdc-file

[7] Harris, D. M., & Harris, S. L. (2013). Hardware Description Languages. *Digital Design and Computer Architecture*, 172–237. https://doi.org/10.1016/B978-0-12-394424-5.00004-5

my ID number is 2, I was required to use one XOR gate, one XNOR gate, and one AND gate, and I accordingly updated the code.

Results: The modified code can be seen on Figure 1.1.

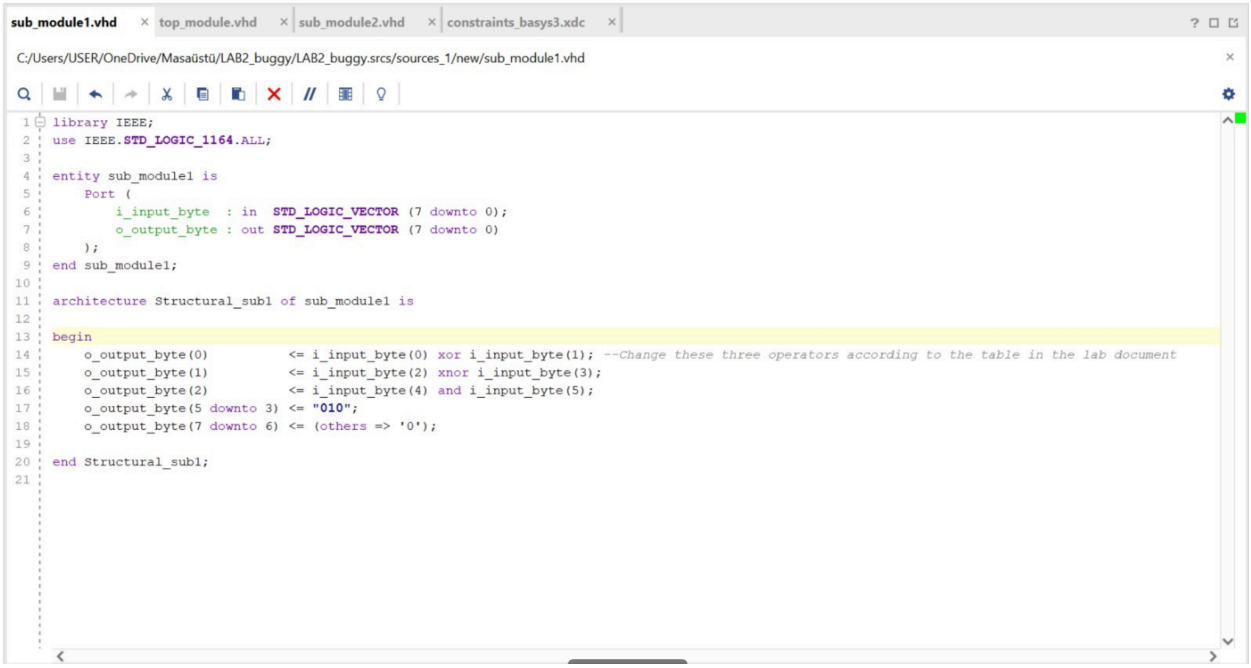

Figure 1.1: Modified version of the code according to the last digit of my ID

PART 2:

Methodology: To address the six identified bugs, the design was synthesized and implemented using the provided development environment, during which all error messages were recorded. For each error encountered, a screenshot was taken to document the issue. Each bug was then fixed by updating the corresponding sections of the code. After each correction, the circuit was re-synthesized and re-implemented to confirm that the error was resolved.

Results: The first bug addressed was changing the system device from the automatically assigned device to Basys3 (Figure 2.1). It was ensured that the code could be transferred to the right device by doing this.

Then, several synthesis errors were observed (Figure 2.2.). The second bug fixed was a syntax error due to a misspelled variable (Figure 2.3-2.4).

When run again, another synthesis error was observed (2.5). In the third bug, component instantiation is observed to use positional mapping rather than named mapping, but it supplied the signals in the wrong order. Basically, it connected i_switch_inputs to s_output_2, and o_output_vector to i_SW (Figure 2.6). This bug is corrected by using named port mapping to avoid confusion about the positional order of the ports (Figure 2.7).

After the third bug, synthesis was completed. After synthesis, various implementation errors were observed (Figure 2.8).

The fourth bug is based on the wrong naming of sub_module2. In the file sub_module2.vhd, the entity is assigned as sub_module2_the_beast (Figure 2.9). However, in the top_module.vhd file, this component was assigned as sub_module2 (Figure 2.10). This bug was fixed by changing the component name in top_module.vhd to sub_module2_the_beast (Figure 2.11).

After this bug, implementation was completed. After implementation, several bitstream errors were observed (Figure 2.12).

The fifth bug was due to a syntax error in the constraint file (Figure 2.13). This bug was fixed by putting an additional curly bracket (Figure 2.14).

The sixth bug was that the set_property statements for o_LED[0] and o_LED[7] were incorrect (Figure 2.15). Specifically, o_LED[0] should be assigned to PACKAGE_PIN U16, and o_LED[7] should be assigned to PACKAGE_PIN V14. This bug was resolved by changing the places of the pins of o_LED[0] and o_LED[7] (Figure 2.16).

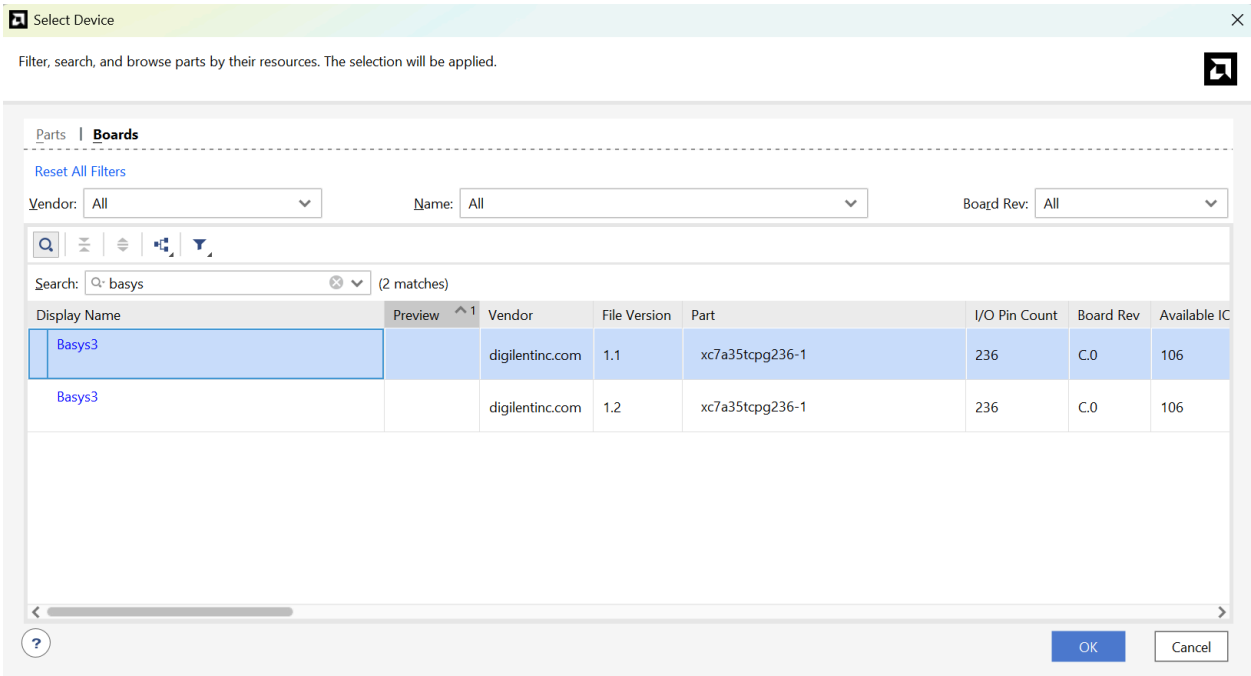

Figure 2.1: System board changed to Basys3

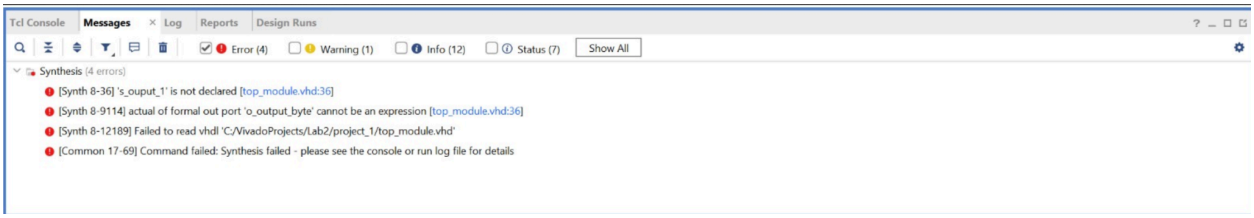

Figure 2.2: Synthesis errors

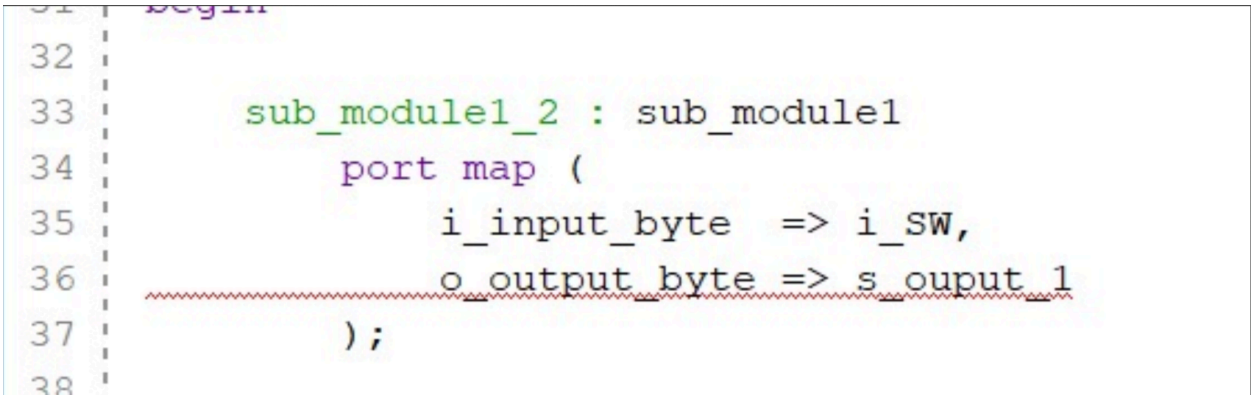

Figure 2.3: Syntax error in top_module.vhd

```
32
33      sub_module1_2 : sub_module1
34          port map (
35              i_input_byte  => i_SW,
36              o_output_byte => s_output_1
37          );
38
```

Figure 2.4: Syntax error resolved

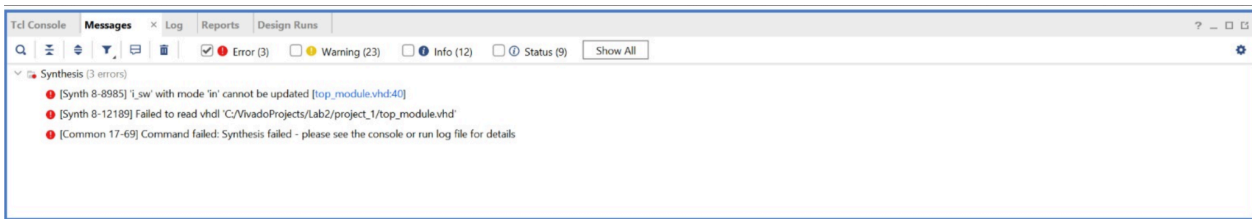

Figure 2.5: Second synthesis error

```
38
39      sub_module2_1 : sub_module2
40          port map (s_output_2, i_SW);
41      s_output_3 <= unsigned(s_output_2) + 25;
42
```

Figure 2.6: Mapping error

```
38
39      sub_module2_1 : sub_module2
40          port map (
41              i_switch_inputs => i_SW,
42              o_output_vector => s_output_2
43          );
44      s_output_3 <= unsigned(s_output_2) + 25;
45
```

Figure 2.7: Mapping error resolved

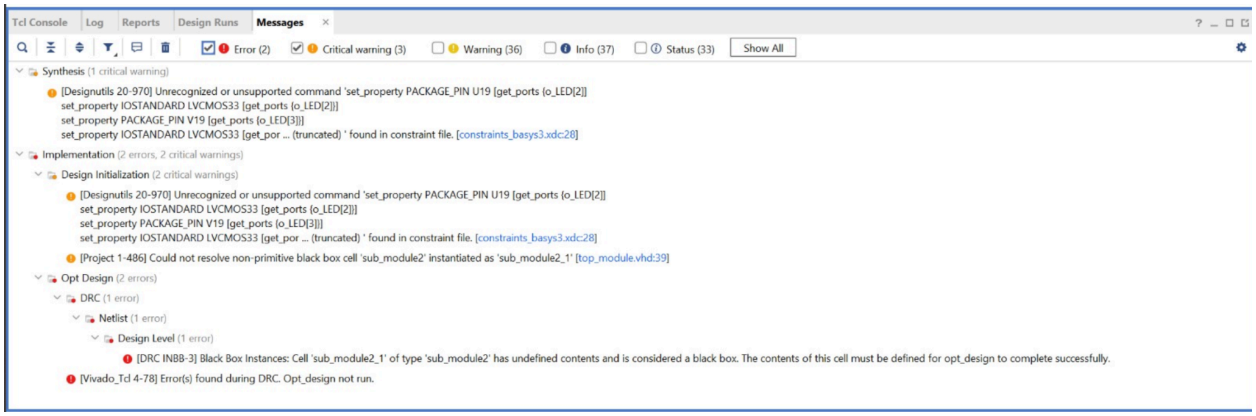

Figure 2.8: Implementation errors

```
4  entity sub_module2_the_beast is
5      Port (
6          i_switch_inputs : in  STD_LOGIC_VECTOR (7 downto 0);
7          o_output_vector : out STD_LOGIC_VECTOR (7 downto 0)
8      );
9  end sub_module2_the_beast;
```

Figure 2.9: Assignment of sub_module2_the_beast

```
21      component sub_module2 is
22          port (
23              i_switch_inputs : in  STD_LOGIC_VECTOR (7 downto 0);
24              o_output_vector : out STD_LOGIC_VECTOR (7 downto 0)
25          );
26      end component sub_module2;
27
28      signal s_output_1, s_output_2 : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
29      signal s_output_3             : unsigned(7 downto 0)         := (others => '0');
30
31  begin
32
33      sub_module1_2 : sub_module1
34          port map (
35              i_input_byte  => i_SW,
36              o_output_byte => s_output_1
37          );
38
39      sub_module2_1 : sub_module2
40          port map (
41              i_switch_inputs => i_SW,
42              o_output_vector => s_output_2
```

Figure 2.10: Naming error

```
20
21      component sub_module2_the_beast is
22          port (
23              i_switch_inputs : in  STD_LOGIC_VECTOR (7 downto 0);
24              o_output_vector : out STD_LOGIC_VECTOR (7 downto 0)
25          );
26      end component sub_module2_the_beast;
27
28      signal s_output_1, s_output_2 : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
29      signal s_output_3             : unsigned(7 downto 0)         := (others => '0');
30
31  begin
32
33      sub_module1_2 : sub_module1
34          port map (
35              i_input_byte  => i_SW,
36              o_output_byte => s_output_1
37          );
38
39      sub_module2_1 : sub_module2_the_beast
40          port map (
41              i_switch_inputs => i_SW,
42              o_output_vector => s_output_2
```

Figure 2.11: Naming error resolved

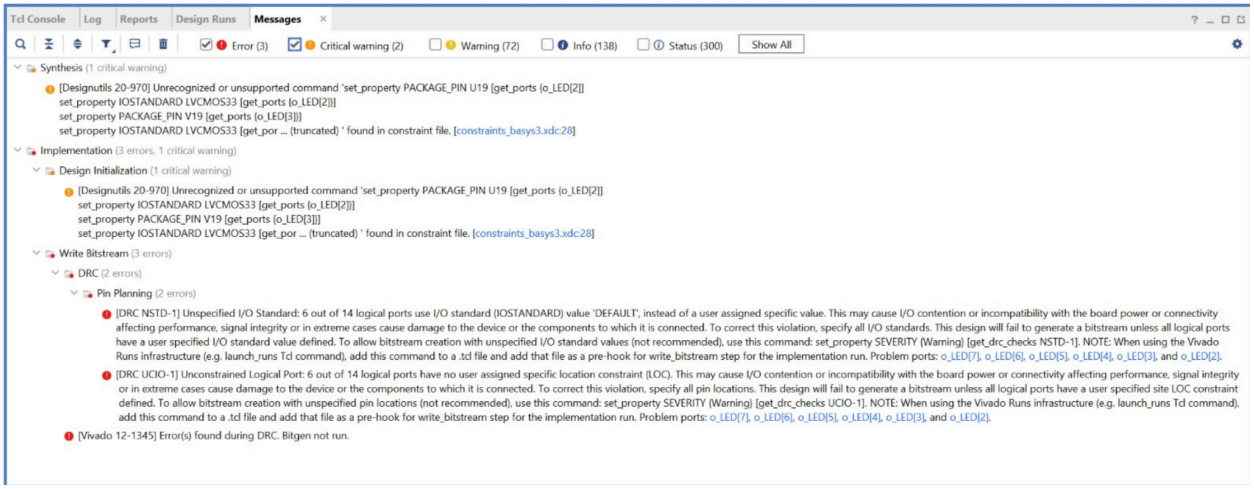

Figure 2.12: Bitstream errors

```
27      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[1]}]
28  set_property PACKAGE_PIN U19 [get_ports {o_LED[2]]
29      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[2]}]
```

Figure 2.13: Syntax error in constraint file

```
27      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[1]}]
28  set_property PACKAGE_PIN U19 [get_ports {o_LED[2]}]
29      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[2]}]
```

Figure 2.14: Syntax error resolved

```
23  # LEDs
24  set_property PACKAGE_PIN V14 [get_ports {o_LED[0]}]
25      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[0]}]
```

```
37      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[6]}]
38  set_property PACKAGE_PIN U16 [get_ports {o_LED[7]}]
39      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[7]}]
```

Figure 15: Incorrect pin assignments for o_LED[0] and o_LED[7]

```
23  # LEDs
24  set_property PACKAGE_PIN U16 [get_ports {o_LED[0]}]
25      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[0]}]
```

```
37      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[6]}]
38  set_property PACKAGE_PIN V14 [get_ports {o_LED[7]}]
39      set_property IOSTANDARD LVCMOS33 [get_ports {o_LED[7]}]
```

Figure 16: Correct pin assignments

PART 3

Methodology: Once the bitstream was successfully generated, the BASYS3 board was powered on, and "Open Target" was selected followed by "Auto Connect." After the configuration was completed, the device was programmed. This final step allowed the inputs and outputs to be observed, and the BASYS3 switches were adjusted to view the resulting output changes.

Results: After fixing the bugs, the VHDL code transfers to the FPGA as intended. The outputs of 5 different inputs can be observed in Figures 3.1 to 3.5.

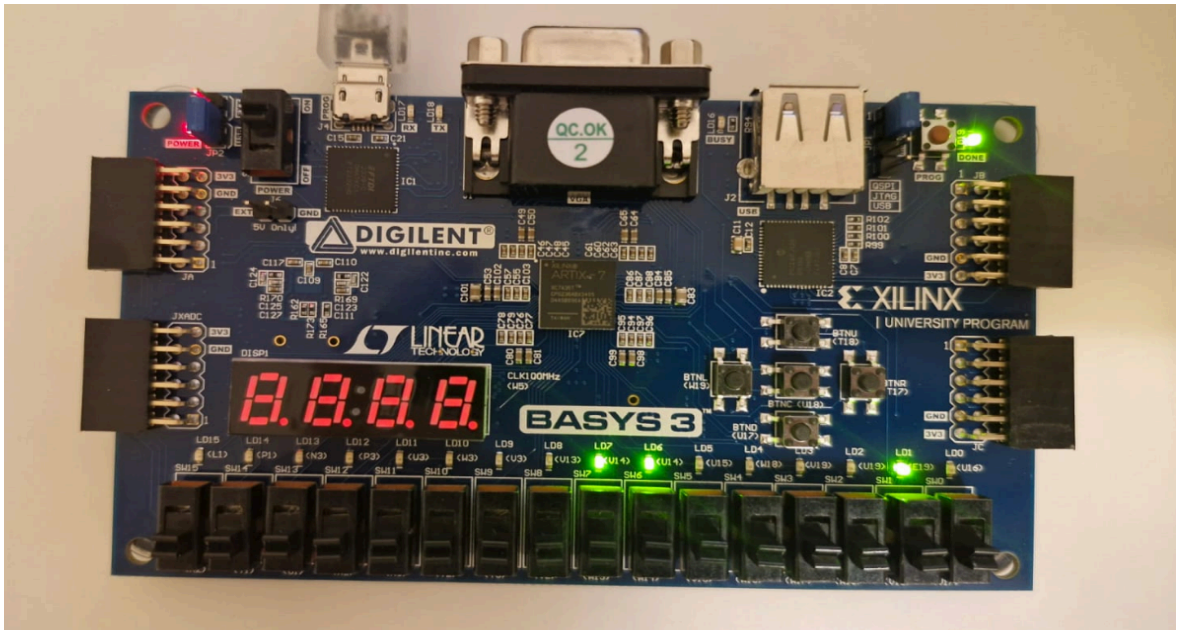

Figure 3.1: Output when all switches are 0

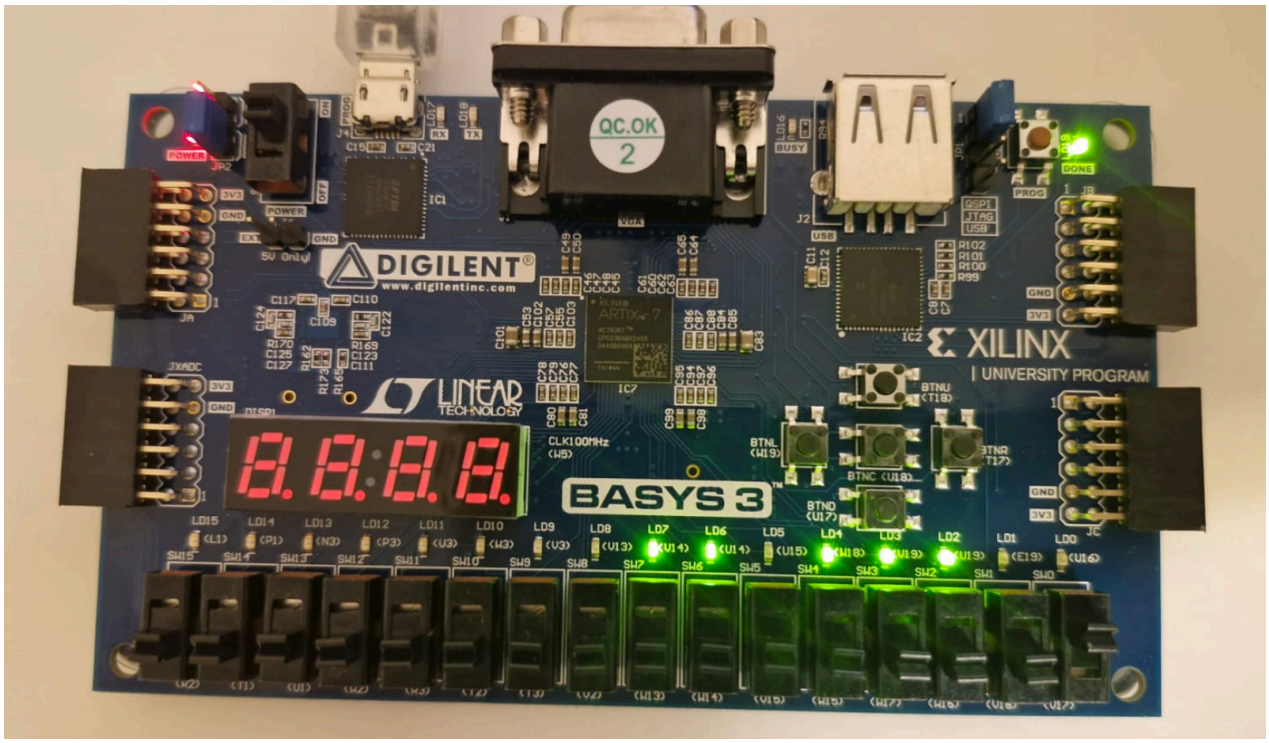

Figure 3.2: Output when SW0 is 1

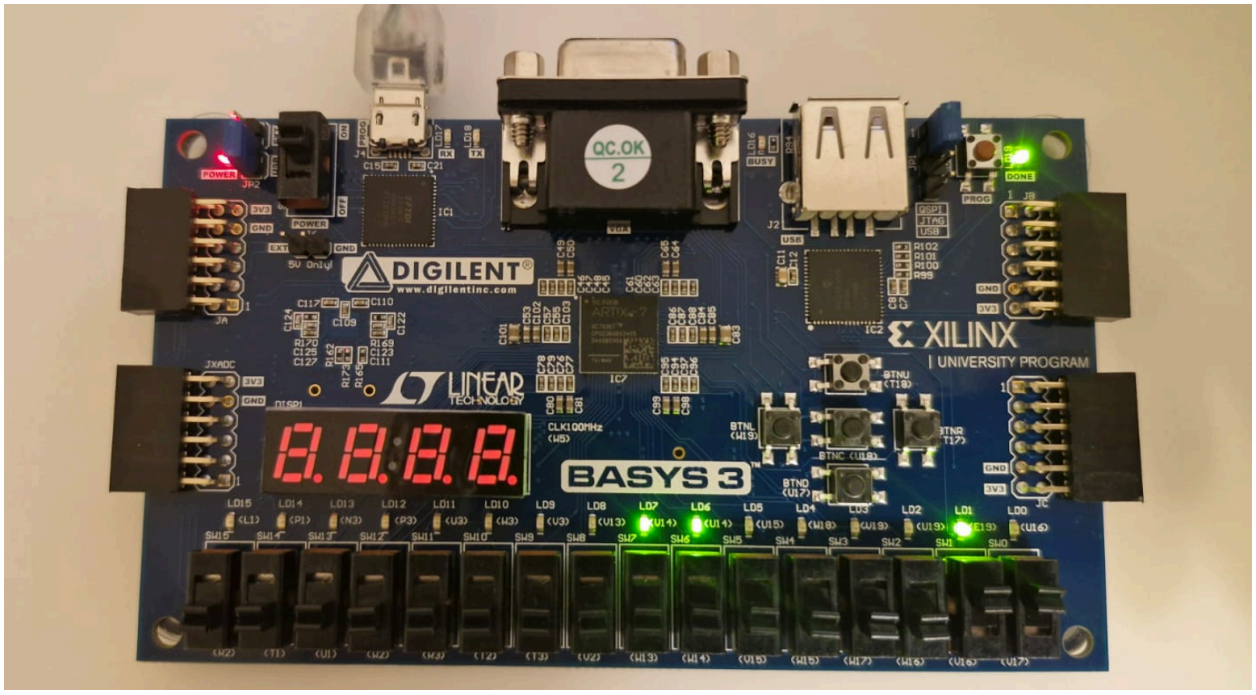

Figure 3.3: Output when SW0 and SW1 are 1

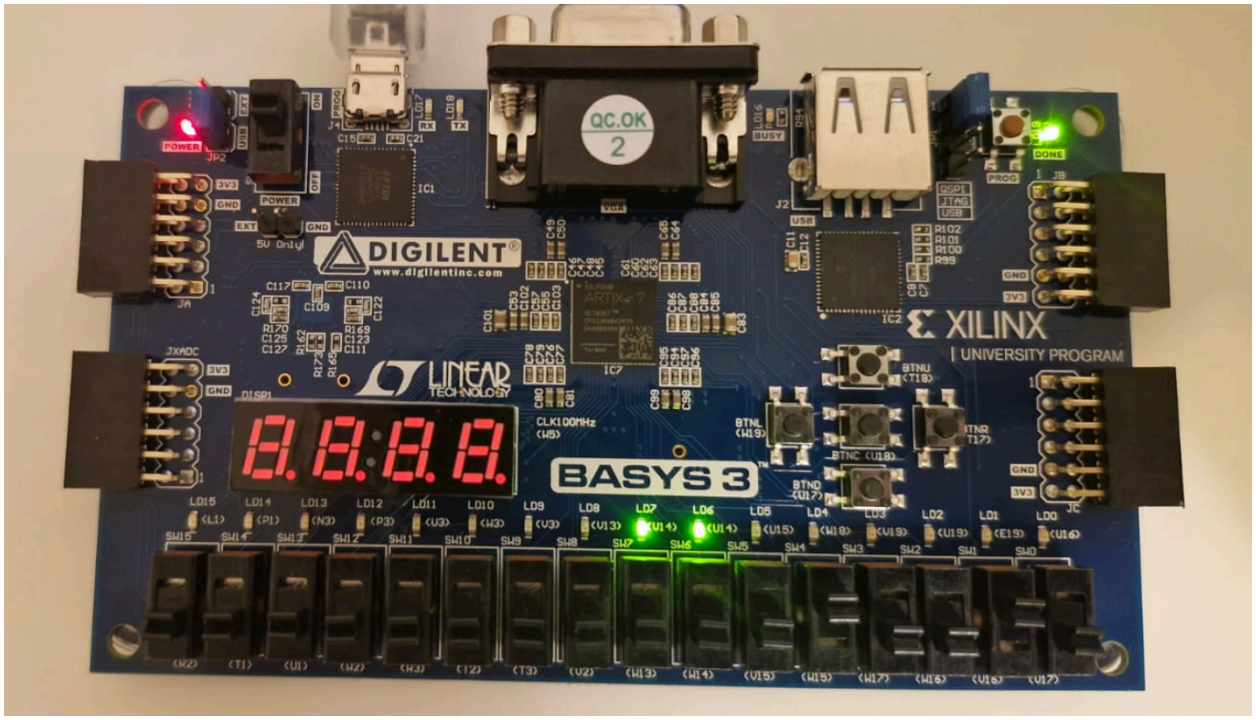

Figure 3.4: Output when SW1 and SW4 are 1

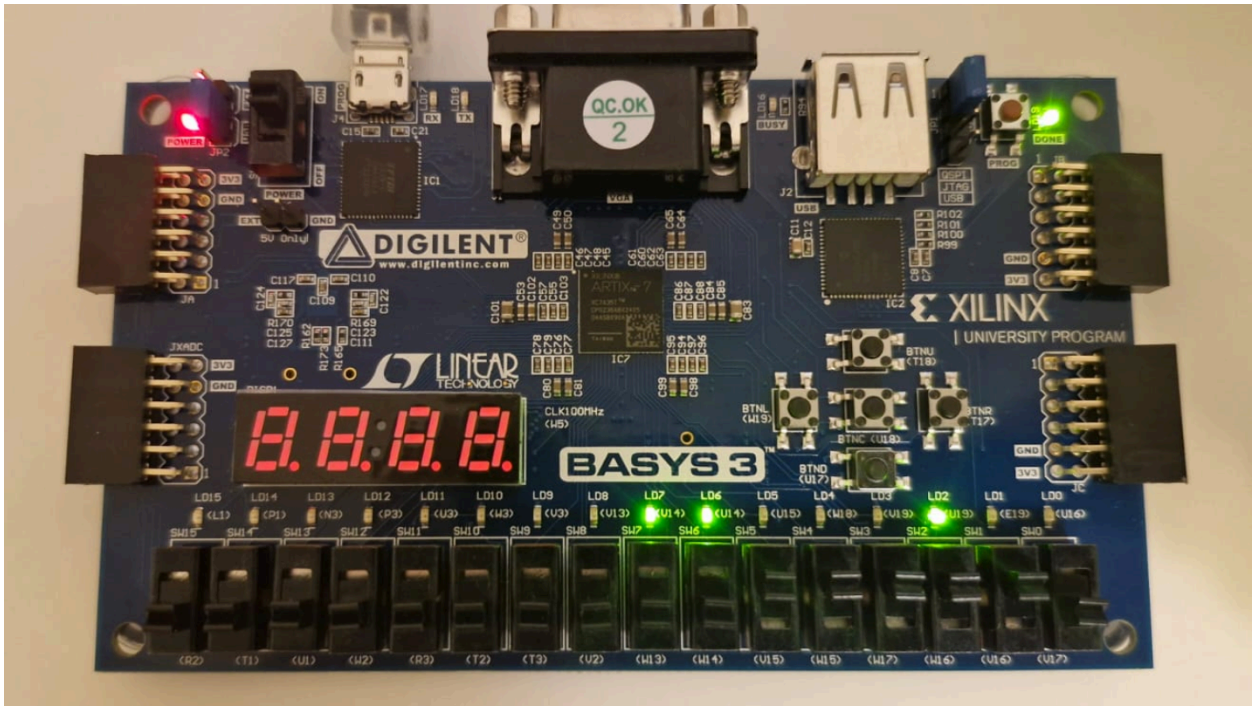

Figure 3.5: Output when SW1, SW2, SW3, SW4 and SW5 are 1

PART 4

Methodology: A test bench was created to verify whether the BASYS3 outputs align with the simulation results. A for loop that runs from 0 to 255 was used as the variable represents an 8-bit unsigned value. In binary, 8 bits can range from 00000000 (0) to 11111111 (255), covering every possible 8-bit value. The i_SW input transitioned from 00000000 to 11111111 in increments of 10 ns, resulting in a total simulation duration of 2560 ns.

Results: In the first trial, the simulation did not give the intended waveform (Figure 4.1). This error was due to the module testbench.vhd not being set as the top module. Hence, the simulation ran an incorrect module (Figure 4.2). This bug was fixed by modifying the testbench module to be the top module (Figure 4.3). Following the behavioral simulation, waveforms of the eight

input signals (switches) and eight output signals (LEDs) were observed. The outputs were shown to match those of the BASYS3 board, which confirmed the success of the code (Figures 4.4-4.9).

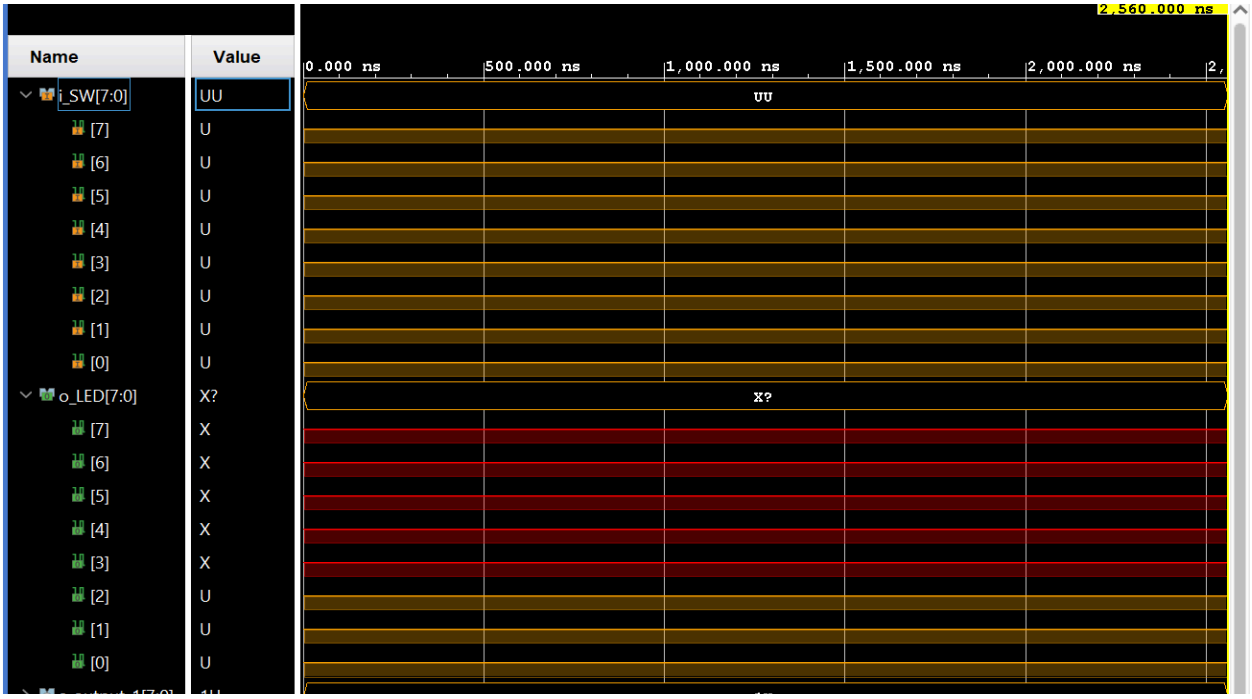

Figure 4.1: Incorrect waveform

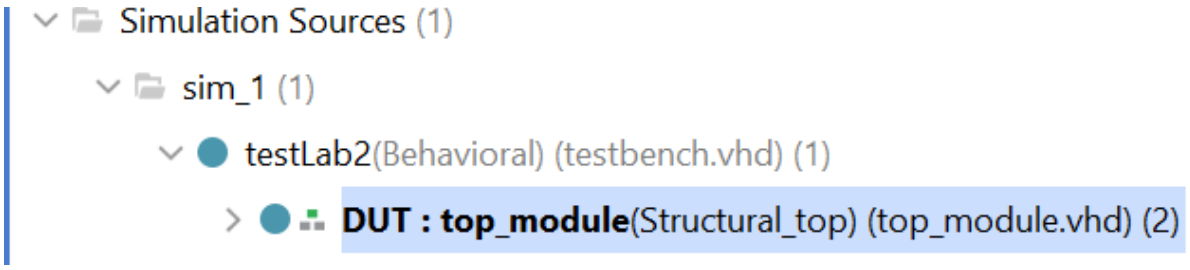

Figure 4.2: testbench.vhd not modified as the top module

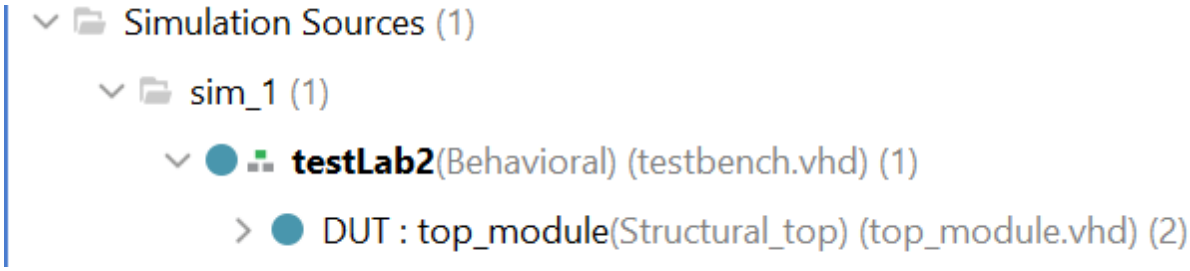

Figure 4.3: testbench.vhd set as top module

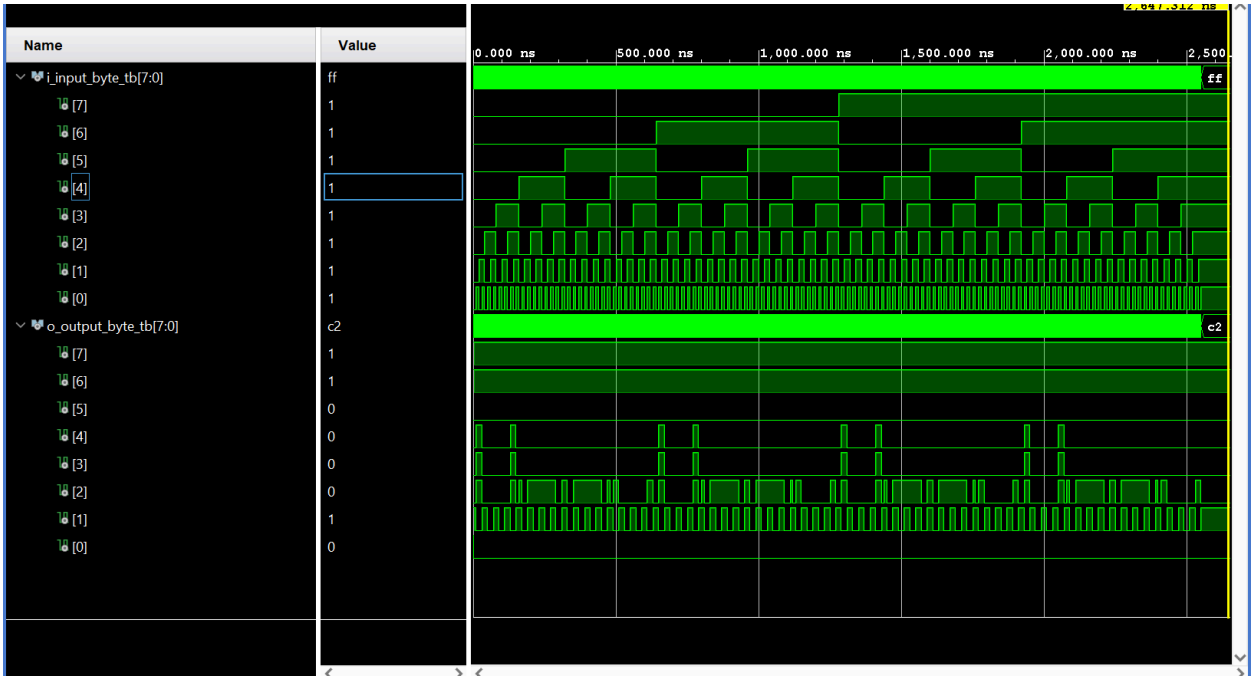

Figure 4.4: Waveform of the testbench

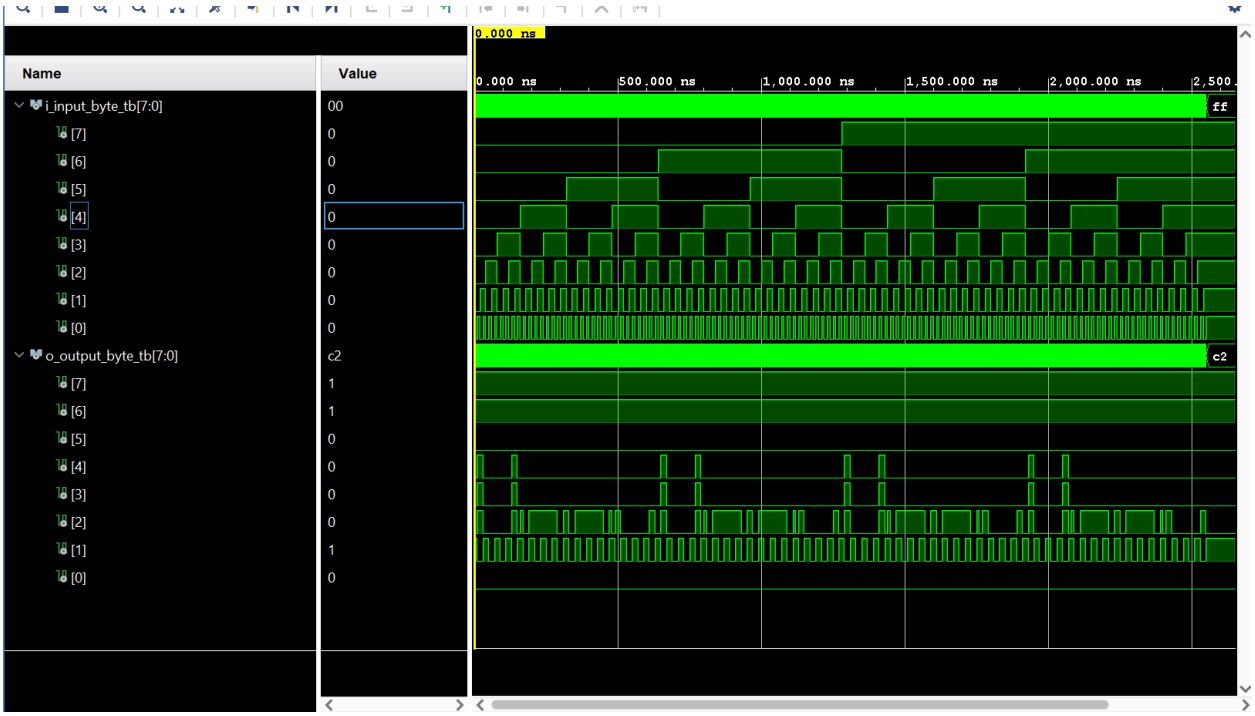

Figure 4.5: Testbench when all switches are 0 (see Figure 3.1)

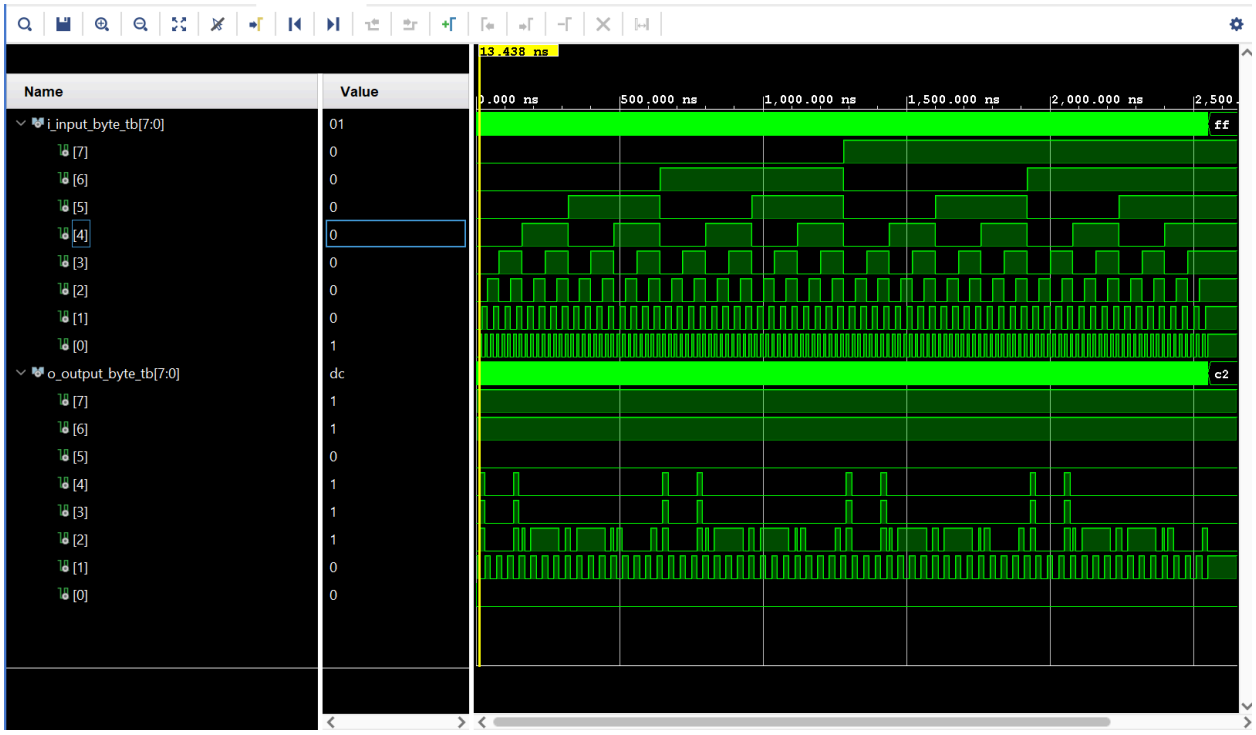

Figure 4.6: Testbench when SW0 is 1 (see Figure 3.2)

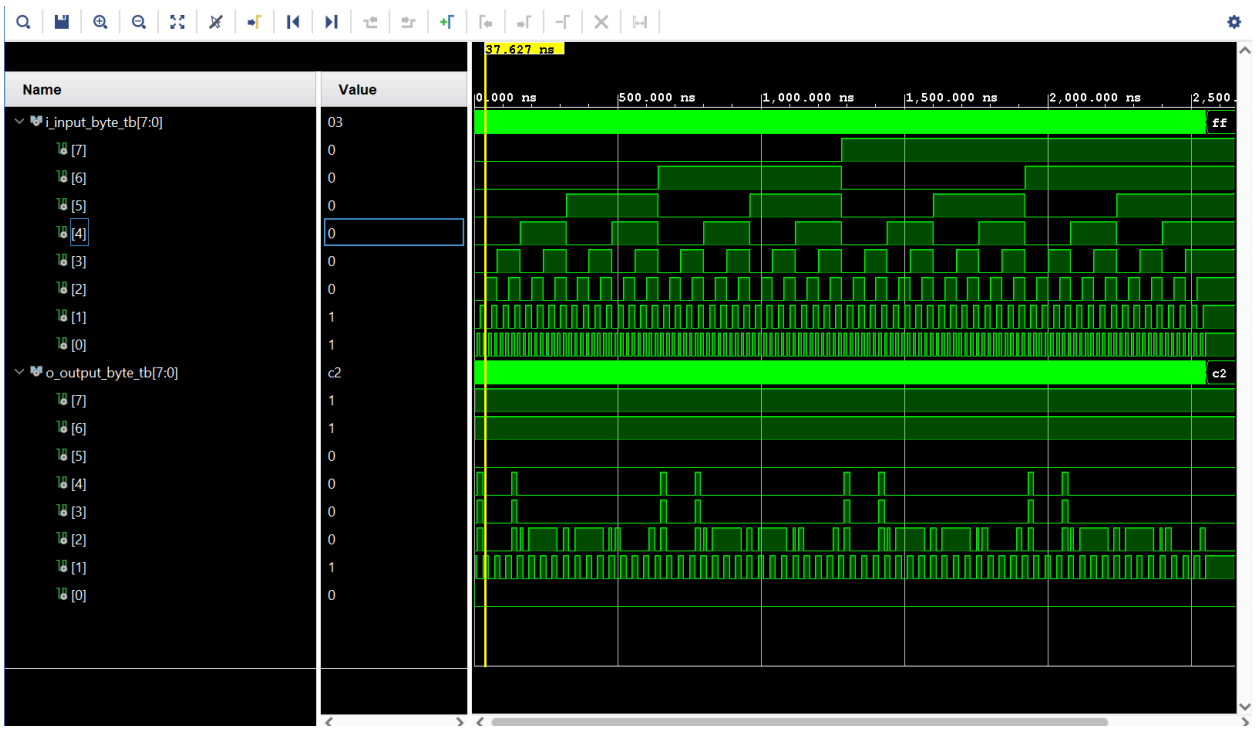

Figure 4.7: Testbench when SW0 and SW1 are 1 (see Figure 3.3)

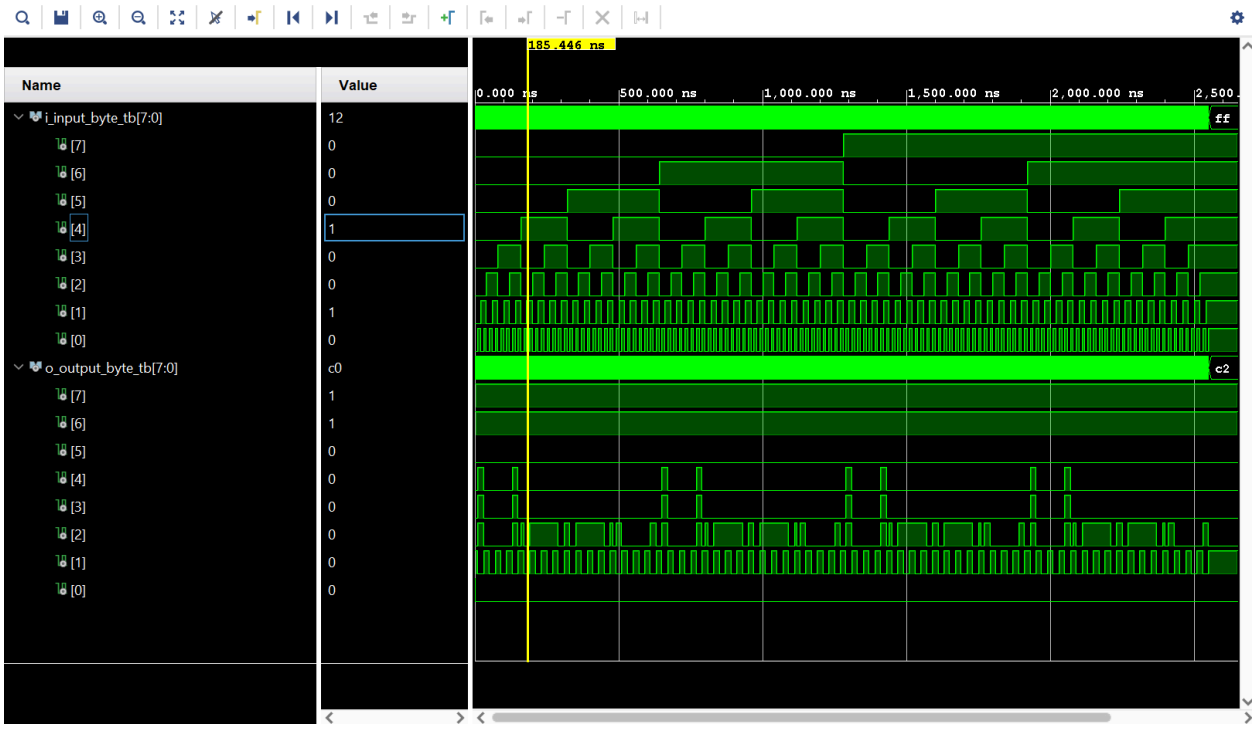

Figure 4.8: Testbench when SW1 and SW4 are 1 (see Figure 3.4)

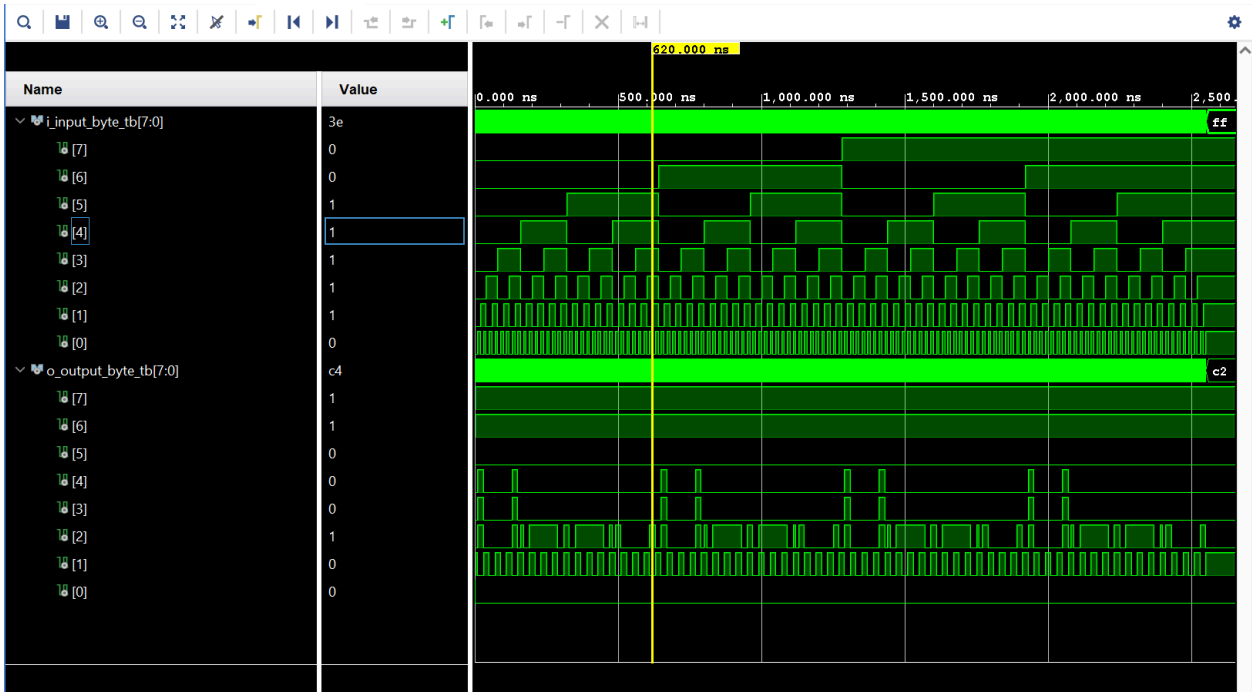

Figure 4.9: Testbench when SW1, SW2, SW3, SW4 and SW5 are 1 (see Figure 3.5)

PART 5

Methodology: Implemented, Synthesized and RTL Designs were generated. During elaboration, the code is checked for syntax errors and any unresolved references (e.g., signals, variables, components, or ports) are resolved. The elaborated design presents a high-level structural representation of the RTL code and shows a top module that instantiates sub-modules containing basic logic gates (in this case, one XOR, one XNOR and one AND gate) (Figure 5.1). This depicts that logical design aligns with the visual representation. The implemented device design represents the physical layout generated by the FPGA design tool, where logic blocks are placed in coordinate-based positions to reflect the mapped elements of the RTL design onto FPGA cells. The synthesized and implemented schematic provides a structural view of the logic and shows how 8-bit inputs are routed through input signals, various gates with differing numbers of inputs, and output signals to drive the outputs.[8] Because the design code has simple combinational logic with no timing constraints, there's not much for the implementation step to optimize. Therefore, the implemented schematic is structurally identical to the synthesized one.

Results: The elaborated, implemented and synthesized schematics can be observed in Figures 5.1-5.3.

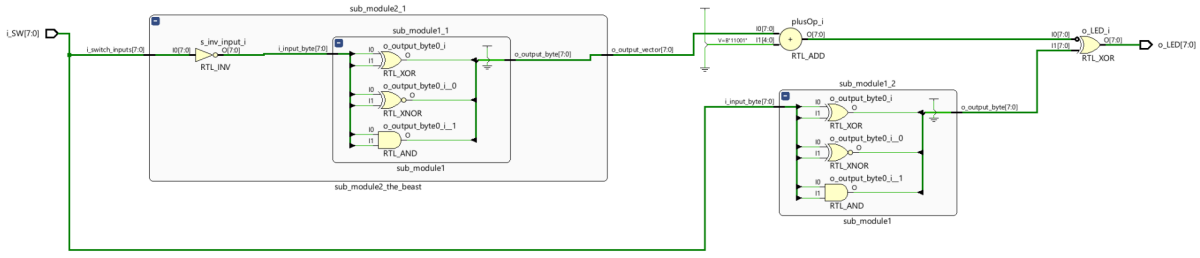

Figure 5.1: Elaborated RTL design

[8] *Using Synthesis Settings • Vivado Design Suite User Guide: Synthesis (UG901) • Reader • AMD Technical Information Portal*. (n.d.). Retrieved February 18, 2025, from https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Using-Synthesis-Settings

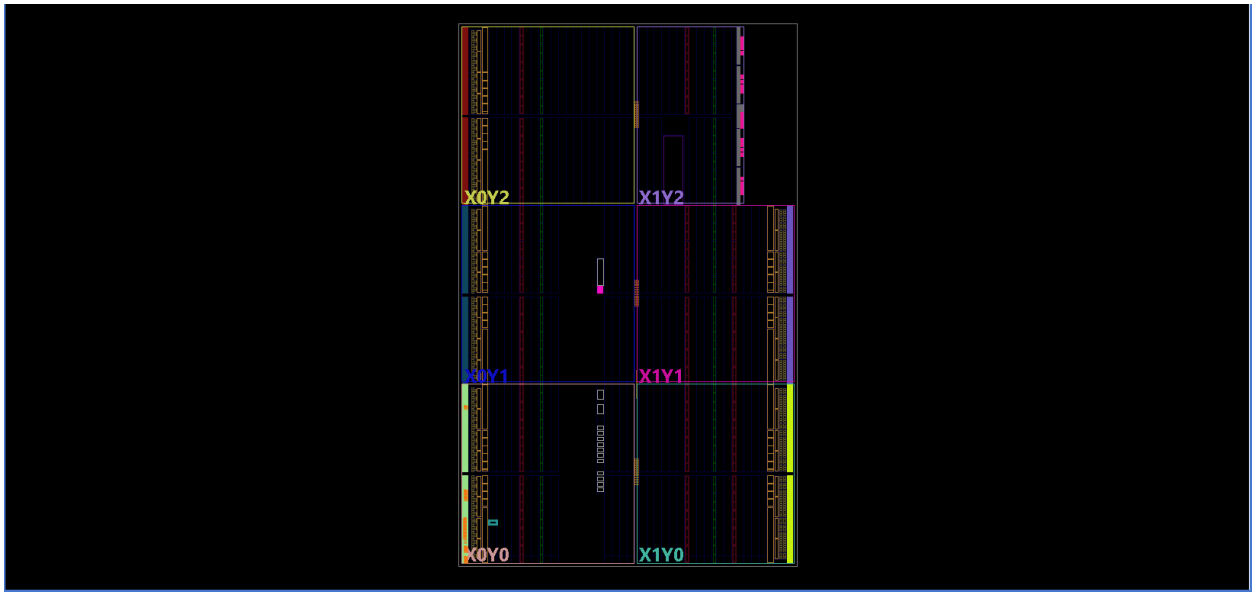

Figure 5.2: Implemented device design

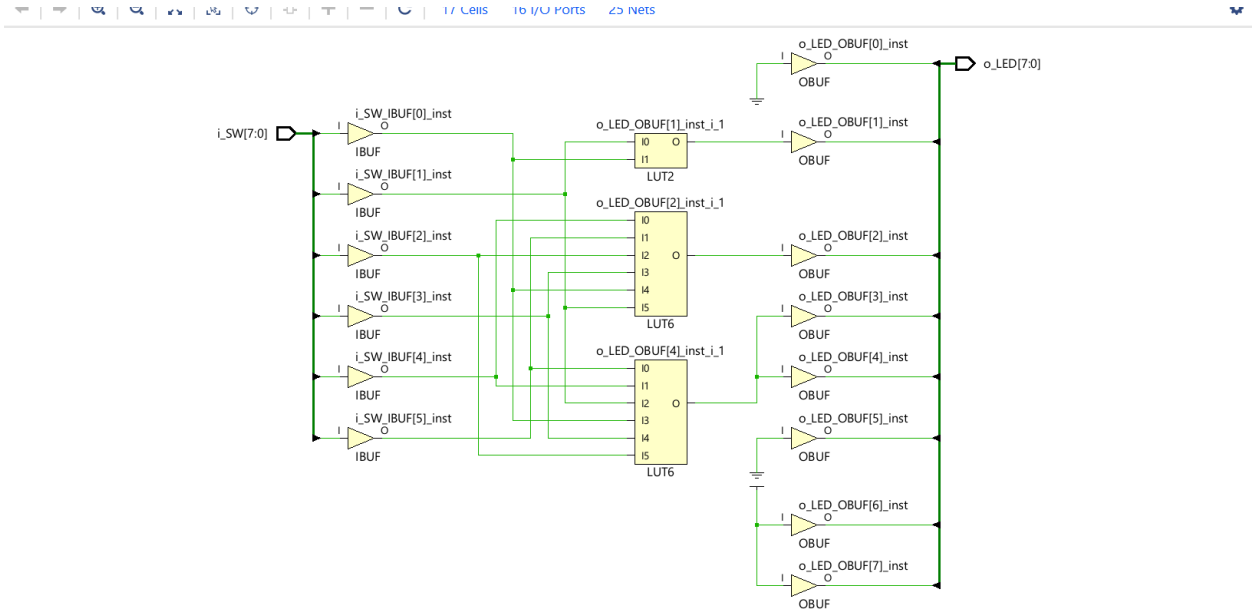

Figure 5.3: Implemented/Synthesized schematic

PART 6 - FINAL CODE BLOCKS

top_module.vhd:

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

entity top_module is
    Port (
        i_SW  : in  STD_LOGIC_VECTOR (7 downto 0);
        o_LED : out STD_LOGIC_VECTOR (7 downto 0)
    );
end top_module;

architecture Structural_top of top_module is

    component sub_module1 is
        port (
            i_input_byte  : in  STD_LOGIC_VECTOR (7 downto 0);
            o_output_byte : out STD_LOGIC_VECTOR (7 downto 0)
        );
    end component sub_module1;

    component sub_module2 is
        port (
            i_switch_inputs : in  STD_LOGIC_VECTOR (7 downto 0);
            o_output_vector : out STD_LOGIC_VECTOR (7 downto 0)
        );
    end component sub_module2;

    signal s_output_1, s_output_2 : STD_LOGIC_VECTOR(7 downto 0) :=
(others => '0');
    signal s_output_3             : unsigned(7 downto 0)         :=
(others => '0');

begin

    sub_module1_2 : sub_module1
```

```
        port map (
            i_input_byte  => i_SW,
            o_output_byte => s_output_1
        );

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

    o_LED <= (not std_logic_vector(s_output_3)) xor s_output_1;

end Structural_top;
```

sub_module1.vhd:

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

entity sub_module1 is
      Port (
            i_input_byte  : in  STD_LOGIC_VECTOR (7 downto 0);
            o_output_byte : out STD_LOGIC_VECTOR (7 downto 0)
      );
end sub_module1;

architecture Structural_sub1 of sub_module1 is

begin
      o_output_byte(0)          <= i_input_byte(0) xor
i_input_byte(1); --Change these three operators according to the table
in the lab document
      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;
```

sub_module2.vhd:

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

entity sub_module2 is
    Port (
        i_switch_inputs : in  STD_LOGIC_VECTOR (7 downto 0);
        o_output_vector : out STD_LOGIC_VECTOR (7 downto 0)
    );
end sub_module2;

architecture Structural_sub2 of sub_module2 is
    component sub_module1 is
        port (
            i_input_byte  : in  STD_LOGIC_VECTOR (7 downto 0);
            o_output_byte : out STD_LOGIC_VECTOR (7 downto 0)
        );
    end component sub_module1;

    signal s_inv_input : STD_LOGIC_VECTOR(7 downto 0) := (others =>
'0');
begin
    s_inv_input <= not i_switch_inputs;

    sub_module1_1 : sub_module1
        port map (
            i_input_byte  => s_inv_input,
            o_output_byte => o_output_vector
```

```
        );
end Structural_sub2;
```

constraints_basys3.xdc

```
set_property PACKAGE_PIN V17 [get_ports {i_SW[0]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[0]}]

set_property PACKAGE_PIN V16 [get_ports {i_SW[1]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[1]}]

set_property PACKAGE_PIN W16 [get_ports {i_SW[2]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {i_SW[2]}]

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]}]
```

testbench.vhd:

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

entity testLab2 is
end testLab2;

architecture Behavioral of testLab2 is
component top_module is
    Port (
    i_sw : in std_logic_vector (7 downto 0);
    o_led : out std_logic_vector (7 downto 0)
    );
end component;

    signal i_input_byte_tb : std_logic_vector (7 downto 0);
    signal o_output_byte_tb : std_logic_vector (7 downto 0);
```

```
begin
    DUT: top_module port map (
    i_sw => i_input_byte_tb,
    o_led => o_output_byte_tb
    );

process
    begin
    for i in 0 to 255 loop
        i_input_byte_tb <= std_logic_vector(TO_UNSIGNED(i,8));
        wait for 10 ns;
    end loop;
    wait;
end process;
end Behavioral;
```

CONCLUSION

In this lab, the fundamentals of digital design were introduced through the use of VHDL and FPGA technology. The basics of VHDL were explored by debugging and testing a combinational circuit on the BASYS3 FPGA using Vivado, and the structure of VHDL code was analyzed while errors were identified and corrected. Six errors were located, modifications to the necessary logic gates were made, and proper functionality was ensured through synthesis, implementation, and FPGA programming. Module instantiation and port mapping were examined, and constraint files were utilized to define the mapping of FPGA pins to design inputs and outputs. Additionally, a testbench was developed to simulate the digital circuit and verify all possible input combinations prior to deployment; and a comparison between the RTL schematic, synthesized design, and implemented design was performed to illustrate the evolution of the design from code to actual FPGA implementation. Finally, it was observed that the simulation

results corresponded with the outputs obtained from the BASYS3 when the switches were adjusted accordingly.

Works Cited

Ashenden, P. J. (1996). *The Designer's Guide to VHDL (Third Edition)*.

Basys 3 - Digilent Reference. (n.d.). Retrieved February 18, 2025, from https://digilent.com/reference/programmable-logic/basys-3/start

Hands, J. P. (1990). What is VHDL? *Computer-Aided Design*, *22*(4), 246–249. https://doi.org/10.1016/0010-4485(90)90054-G

Harris, D. M., & Harris, S. L. (2013). Hardware Description Languages. *Digital Design and Computer Architecture*, 172–237. https://doi.org/10.1016/B978-0-12-394424-5.00004-5

Perry, D. L. . (2002). *VHDL : programming by example Douglas L. Perry*. McGraw-Hill Education.

Vivado Overview. (n.d.). Retrieved February 18, 2025, from https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html

What is a Constraints File? - Digilent Reference. (n.d.). Retrieved February 18, 2025, from https://digilent.com/reference/programmable-logic/guides/vivado-xdc-file