Completing this assignment takes two steps:

1. Write the code, *verify* it with a *testbench* using ModelSim, and debug it.
2. Program the DE1-SoC board with your solution and *test* its functionality. Check the RTL-viewer to see if your design is implemented the way you want.

After completing *every* step, call your instructor to verify that you are done with that step.

Assigment 2: Driving the 7-segment display

In this assignment we are going to program the FPGA in such a way that you can input a 4 bit binary number using the switches on the DE1-SoC board (SW0 to SW3). The binary number must then be displayed in hexadecimal on a 7-segment display as shown in Figure 1.

Figure 1: Hexadecimal numbers 0 to F displayed on a 7-segment display.

The 7-segment displays on the DE-SoC board are common anode displays, which means that the common anode of the display is connected to the 3.3V power supply. How the 7-segment display is connected to the FPGA is shown in Figure 2.

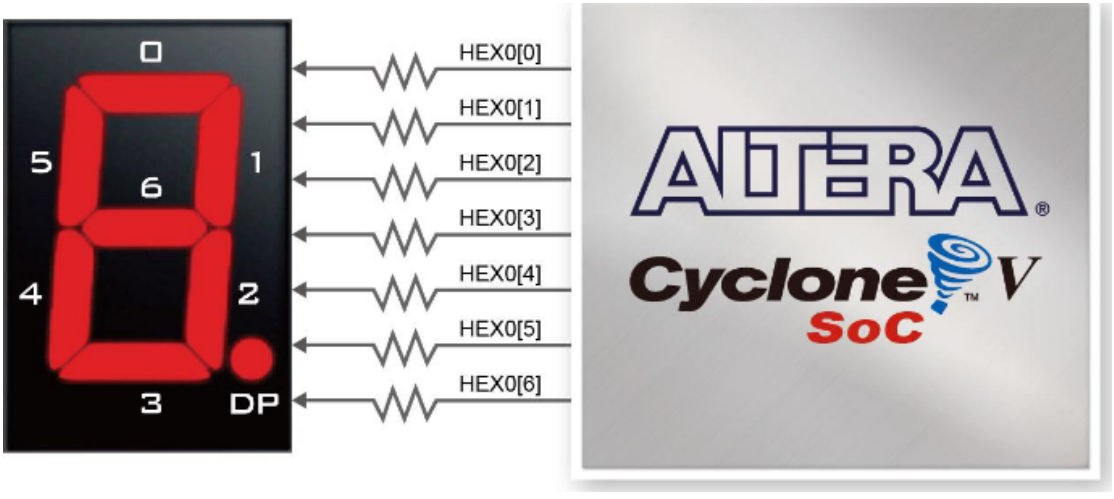

Figure 2: The connections between the common anode 7-segment display HEX0 and the FPGA on the DE1-SoC board.

This seven-segment driver shall have at-least the following inputs:

- Four-bit binary input.
- One-bit blanking input. If this input is set to 1, the display is blanked.
- Seven-bit output for the seven-segments display.

Functional testing of your design is necessary to verify your design. Following the practice of Test Driven Development (TDD)[1], we will first write a testbench for the seven-segment driver. Then we will write the VHDL code for the seven-segment driver. Finally, we will simulate the design and verify its functionality. In Chapter 3 of the Lab Work Handbook we have specified the inputs and checked the outputs manually. This was a tedious and error-prone task. Our goal is to automate this task by using a testbench. The testbench will automatically apply different inputs to the DUV (Device Under Verification)[2], check the outputs of the DUV, and report any errors. We focus on functional simulation and the purpose of the testbench is to check the functionality of the seven-segment driver. A testbench is written in VHDL, and we use ModelSim to execute the testbench. Create a new project in ModelSim and call it `assignment2`. Add a VHDL file to your project and name it `assignment2_tb.vhd`. Remember to tell the compiler that you want to use the VHDL 2008 standard. For this assignment, the testbench is already written by your instructors. Add the code found in Listing 1 to `assignment2_tb.vhd`.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- The entity of your testbench. No ports declaration in this case.
entity assignment2_tb is
end entity;

architecture testbench of assignment2_tb is
    -- The component declaration should match your entity.
    -- It is very important that the name of the component and the
    -- ports (remember direction of ports!) match your entity!
    component seven_segment_decoder is
        port (
            sw: in std_ulogic_vector(3 downto 0);
            blank: in std_ulogic;
```

[1] https://en.wikipedia.org/wiki/Test-driven_development.

[2] In this assignment the DUV is the seven-segment driver.

```
            hex0: out std_ulogic_vector(6 downto 0)
        );
    end component;
    -- Signal declaration. These signals are used to drive your
    -- inputs and store results (if required).
    signal sw_tb: std_ulogic_vector(3 downto 0);
    signal blank_tb: std_ulogic;
    signal hex0_tb: std_ulogic_vector(6 downto 0);
begin
    -- A port map is in this case nothing more than a construction to
    -- connect your entity ports with your signals.
    duv : seven_segment_decoder port map(
        sw => sw_tb, blank => blank_tb, hex0 => hex0_tb
    );

    process
        type int_array is array(0 to 15) of integer;
        constant expexted_seven_segment_code: int_array := (
            16#40#, 16#79#, 16#24#, 16#30#,
            16#19#, 16#12#, 16#02#, 16#78#,
            16#00#, 16#10#, 16#08#, 16#03#,
            16#46#, 16#21#, 16#06#, 16#0E#
        );
    begin
        report "Testing entity assignment2.";
        -- Initialize signals.
        sw_tb <= "0000";

        blank_tb <= '1';
        wait for 10 ns;
        -- Check blank.
        assert hex0_tb = "1111111"
            report "test failed for blank = 1" severity error;

        blank_tb <= '0';
        -- Loop through all possible values of switches.
        for i in 0 to 15 loop
            sw_tb <= std_ulogic_vector(to_unsigned(i, sw_tb'length));
            wait for 10 ns;

            -- Check result.
```

```
            assert hex0_tb = std_ulogic_vector(to_unsigned(
                    expexted_seven_segment_code(i), hex0_tb'length
                ))
                report "test failed for i = " & to_string(i)
                severity error;
        end loop;

        report "Test completed.";
        std.env.stop;
    end process;

end architecture;
```

Listing 1: The testbench for assignment2: `assignment2_tb.vhd`.

For more information on testbenches see Chapter 18 in your textbook[3]. In the testbench you can use whatever VHDL construct you like, but in your circuit designs you can only use synthesizable VHDL.

When you simulate the testbench, you will see that the testbench reports the following message.

```
Component instance "duv : seven_segment_decoder" is not bound.
```

This makes sense because we have not yet written the VHDL code for the seven-segment driver. Add a new file called `seven_segment_decoder.vhd` and add the **`entity`** declaration for the seven-segment driver, and it's **`architecture`** there.

As you have read in the comments it is very important that the component declaration have the same port(s) as your entity declaration. Once you have added this file you can start simulation and test your circuit. Click on "Library", then right-click on your testbench and click on "Simulate", see Figure 3.

Now you can add the signals from your testbench to your waveform, see Figure 4.

Now you can run the simulation by clicking on the "Run -All" button  .

The test results are reported in the "Transcript" window. You can stop the simulation by selecting Simulate > End Simulation in the main menu.

[3] Volnei A. Pedroni. *Circuit Design and Simulation with VHDL, Third Edition*. The MIT Press, 2020. ISBN: 978-0-262-04264-2.

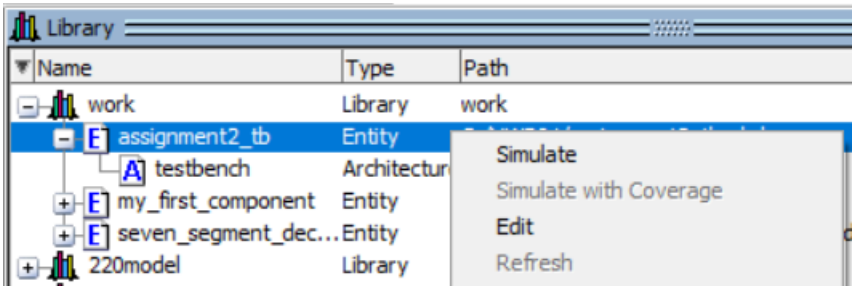

Figure 3: Starting the simulation.

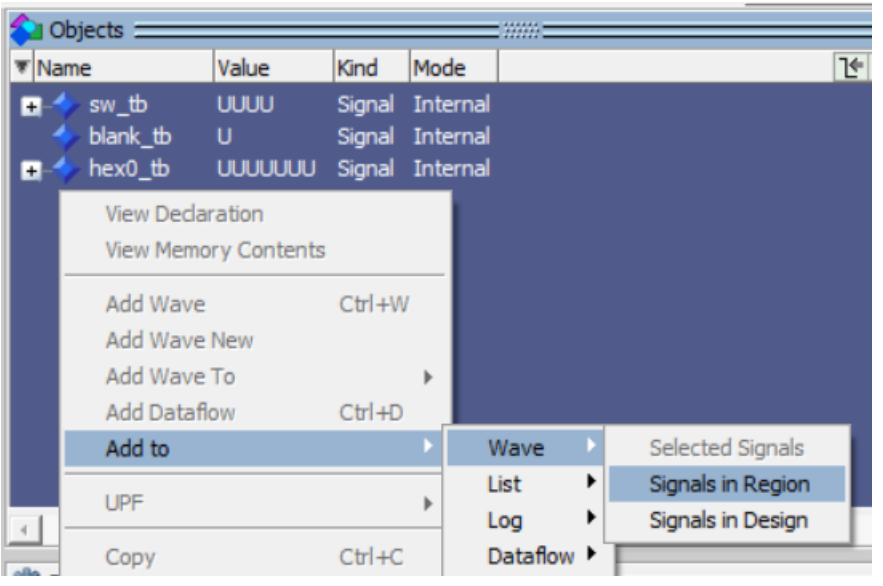

Figure 4: Adding signals to the waveform.

Debug and improve your design until the testbench reports no errors. When you are done, call your instructor to verify your design.

To test the 7-segment driver on the DE1-SoC board, you need to create top-level entity. For this assignment, the top-level entity is already written by your instructors, see Listing 2.

```
library ieee;
use ieee.std_logic_1164.all;

-- The top-level entity of your design.
-- The signal names should match the names used in the
```

```
-- pin assigments in pin_assignments.qsf.
-- After importing pin_assigments.qsf you can use the
-- Assignment Editor to find these names.
entity assignment2 is
    port (
        SW: in  std_ulogic_vector(9 downto 0);
        LEDR: out std_ulogic_vector(9 downto 0);
        HEX0: out std_ulogic_vector(6 downto 0)
    );
end entity;

architecture structural of assignment2 is
    -- The component declaration should match your entity.
    -- It is very important that the name of the component and the
    -- ports (remember direction of ports!) match your entity!
    component seven_segment_decoder is
        port (
            sw: in std_ulogic_vector(3 downto 0);
            blank: in std_ulogic;
            hex0: out std_ulogic_vector(6 downto 0)
        );
    end component;
begin
    -- Instantiate the seven_segment_decoder here.
    -- Connect SW3 downto SW0 with inputs sw;
    -- connect SW9 with input blank;
    -- connect HEX0 with hex0.
    -- INSERT YOUR CODE HERE

    -- Connect the LEDR outputs to the SW inputs
    -- INSERT YOUR CODE HERE
end architecture;
```

Listing 2: The top-level entity for assignment2: assignment2.vhd.

Create a new Quartus project as described in Chapter 4 of the Lab Work Handbook. Add the files seven_segment_decoder.vhd and assignment2.vhd to this project. Instead of using the Pin Planner to assign inputs and outputs of the top-level entity to pins on the FPGA, we will import all pin settings from the file DE1_SoC.qsf. Download this file and select the Assignments » Import Assignments... menu item and select the downloaded file.

Compile the project and download it to the DE1-SoC board. Test the functionality of your design. The binary number on the switches SW3 downto SW0 should be displayed on the LEDs and also in hexadecimal on the 7-segment display HEX0. The 7-segment display should be blanked when SW9 is in the upper position.

When you are done, call your instructor to verify your design.