Lab 5: Seven-Segment Display

Mert Örnek EEE 102-3 22302052 29.03.2024

Purpose

The purpose of this lab session is to design a seven-segment display on Basys 3 FPGA by using VHDL. Since Basys 3 can display a single digit out of four at a time, this experiment necessitates the application of the built-in clock. On these grounds, this experiment also has the purpose of learning how to use the clock built into the Basys 3 as well as how to utilize the periodic clock signal in ways that suit our needs which includes building a clock divider to obtain the desired frequency necessary for the "persistence of vision" effect.

Design Specifications

The setup I intend to construct will do the following: It will have two inputs and two outputs. The two inputs being a 16-bit binary switch vector and the build in clock of the Basys 3 operating at a predesignated 100 MHz, and the two outputs being the 4 anode(an) pins -indicating the selected digit of the display- and the 7 "seg" pins ,indicating the respective LED segments of the selected digit (Figure 1). The system will take a 16-bit switch input, interpret these as 4-bits per digit indicating a binary value between 0 and 9 and display these segments in a visually simultaneous manner.

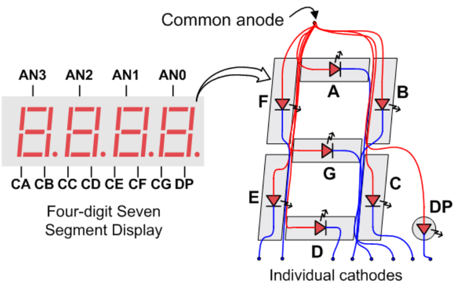

Figure 1: 7-Segment display of Basys 3 (Basys 3 Reference)

The first stage of implementing the design involves designing a setup that can take appropriate chunks of the switch input and convert this data into appropriate outputs for the an pins and the segments. For this purpose, a 4-to-1 multiplexer and a 4-to-16 decoder were utilized. Initially, sections of the 16-bit switch logic vector are assigned to four 4-bit signals. These signals represent chunks of the switch vector corresponding to a digit of a specific index. The values between 0 and 9 are converted from binary and the other values are not cared about. The decoder (led_decoder.vhd) has a 4-bit switch input consisting of one of these chunks (i_dtype) and outputs a 7-bit logic vector which states the segments (o_segdata) necessary to display the number specified by the input. The output produced by the decoder is used as the inputs of the 4-to-1 multiplexer (digit_mux.vhd). The multiplexer takes in the four 11-bit logic vectors, these being a vector consisting of the merged segment logic vector and the 4-bit anode logic vector. The multiplexer has a 2-bit switch input (i_s) which represents the index of the digit to be displayed. "00" for index 0, "01" for index 1, "10" for index 2 and "11" for index 3. This results in the multiplexer having an 11-bit output (o_data) consisting of the segment and anode vectors in that order. As a side note -to make the VHDL implementation easier- the decoder was implemented as a submodule of the module containing the multiplexer. The multiplexer module also has the 16-bit number vector (i_num) as an input and converts this value to the corresponding segment and anode data withing that module. (see Figures 2.1 and 2.2 for the visual representation)

The second stage of implementation concerns having all the digits seemingly light up simultaneously. For this purpose, the 100MHz clock built into the Basys 3 was used in conjunction with a clock divider (clock_divide.vhd) to obtain an appropriate frequency of 200Hz. This is where the clock divider comes in. The clock divider has two inputs and an output: The inputs are the default clock (i_clock) of the board and a reset button (i_res) , and the output is the modified clock (o_clock) with an adjusted frequency. The clock divider cuts down the frequency by a specified amount by using a counter connected to an if statement. The if statement uses the reset as a condition to determine whether to keep counting up to a specified value or to reset the count and restart from 0. The counter is designed to count up to a certain value by adding integer 1 each time the input clock has a rising edge. When the counter reaches a designated value, it switches the value of the output clock and starts over from 0. Through this process, the clock divides the frequency by one plus designated maximum value times two , with the two being due to the rising edge of the input clock being used as a trigger which halves the frequency. This results in a -potentially- significantly slower clock frequency being created depending on the max value of the counter. Here, it is worth noting that this method cannot be used to obtain any desired signal value, as the frequency of the output clock needs to equal the value of the input clock divided by a whole number. The obtainable values are the integer divisions of the base clock frequency, like the 200 Hz this system obtained. Therefore, we cannot use the default clock frequency of 100 MHz to obtain any arbitrary value such as "88.5 MHz", for example. For this experiment, a max value of 249 999 (250 000th rising edge) was utilized to obtain a frequency of (100x10^6 Hz) / (2 x 25 x10^4) = 200 Hz. This frequency is used in the top module in the form of a case-when if statement triggered by the rising edge of the modified clock which happens at a frequency of 200Hz. This case-when if statement determines the switch input of the 4-to-1 MUX used to select the active digit.

The testbench (top_mod_TB.vhd) of the top module does two things to simulate the outputs: It uses a 5+5=10 ns periodic loop (visible in Figure 3.1) that cycles between '1' and '0' to represent the 100MHz clock input of the Basys 3 and defines 11 different combinations of switch inputs to represent 11 out of 2^16 possible inputs.

The testbench of the clock divider (clk_TB.vhd) has two inputs and an output: The inputs are the base clock running at 100MHz and the reset button and the output is the modified clock running at 200Hz. The 100MHz clock input of Basys 3 is replicated through the same loop mentioned in the testbench of the 7-segment display and the reset button has an independent loop running at a greater period (300 ns).(Figure 3.3)

The constraints of the file are as follows:

Switches:	sw(15 downto 0):	R2, T1, U1, W2, R3, T2, T3,V2, W13, W14, V15, W15, W17, W16, V16, V17
Clock:	clk:	W5
Anodes:	an(3 downto 0):	W4, V4, U4, U2
Segments:	seg(6 downto 0):	U7, V5, U5, V8, U8, W6, W7

Methodology

This lab session will be conducted in two stages: implementation of the code in modular fashion via VHDL and observation of the properties of the code through features such as simulations. The implementation stage will be implemented in the following order: The necessary submodules will be constructed first and then combined under a single top module. These submodules include a 4-to-1 MUX, 4-to-16 DEC, a clock divider and -later on- the necessary sim codes. The MUX is expected to use the digit of the desired number to relay information about the specific number displayed on that digit and the DEC is meant to supplement the MUX by relaying information about the segments each number is meant to illuminate. Once these two are implemented, a clock divider of appropriate frequency will be constructed to create the illusion of "persistence of vision". The clock will use the FPGA board's default clock frequency to create a more desirable clock signal of a lower frequency value. Once the groundwork of the code is finished and shown to work on the Basys 3 FPGA, testbenches will be utilized to observe how the signals behave during the aforementioned processes. Findings related to these simulations will be elaborated in the "Results" section and images of the working FPGA will be provided in "Appendices".

Results

The results part of this lab session will consist of the physical implementation of the setup explained in the "Design Specifications" via Basys 3 and the simulation results of the same setup.

For the Basys 3 implementation of the project, some input combinations were entered into the switches and it was observed if the physical outputs matched the expected output or not. These implementations were as follows:

To make the outputs easier to understand, the 16-bit switch inputs will be split into 4-bit chunks representing digits from the most significant figure to the lowest.

Implementation 1: Input: 1001 , 1000 , 0000 , 0100

Output (observed binary value) : "9804" => matches the expected value (Figure 4.1)

Implementation 2: Input: 0101 , 0100 , 0111 , 0011

Output (observed binary value) : "5473" => matches the expected value (Figure 4.2)

Implementation 3: Input: 0010 , 0010 , 0011 , 0001

Output (observed binary value) : "2231" => matches the expected value (Figure 4.3)

Implementation 4: Input: 0001 , 1001 , 0101 , 0101

Output (observed binary value) : "1955" => matches the expected value (Figure 4.4)

Implementation 5: Input: 0001 , 0010 , 0011 , 0100

Output (observed binary value) : “1234” => matches the expected value (Figure 4.5)

As an extra, the frequency of the clock divider was lowered down to 2Hz for a single time in order to observe the digits being refreshed consecutively (Figure 4.6). In the image, you can see the Basys 3 while it is cycling from the “2” at index 2 to “3” at index 1. The modified frequency was increased back to its designed 200Hz once this picture was taken.

As for the simulations, two types of simulations were implemented: One to observe the behavior of the top module of the 7-Segment Display and another to see the functioning of the clock divider.

In Figures 3.1 and 3.2, we can observe the behavior of the top module under the conditions of the simulation. While we can clearly see the test inputs assigned to the switches clearly, the value of anodes and the segments are locked to “1” while the clock seems like a blur from afar due to the high frequency of the 100MHz clock input relative to the timing diagram visible in figure 3.1. The anode and segment values being locked to 1 is likely due to them being used to implement 4 numbers each test input cycle, meaning that they do not maintain a constant value during that process due to them constantly cycling. The clock -on the other hand- can be observed as cycling between ‘1’ and ‘0’ with a net period of 10 ns, 5ns per ‘1’ and 5ns per ‘0’. This value was obtained through the Basys 3’s standard frequency value of 100MHz being convertible to a period time value of 10 ns.

As for the simulation of the clock divider (Figure 3.3), there are two observable values: The default clock input of 100MHz and the modified clock output of 200 Hz. While there is also the input reset value, this value only serves the purpose of shutting the counter off and stopping the clock dividing process meaning it does not provide much to be observed. While the frequency and the close-up image of the input clock is the same as the one observed in the previous simulation (Figure 3.1), we can also observe how the previous clock has a much larger frequency than the new clock in the way that the previous one of 100MHz seems like a blur compared to the modified one of 200Hz whose cycles are visible (Figure 3.3).

Conclusion

The purpose of the experiment was to design and observe the functions of a seven-segment display and its functions. To do this, modules previously taught about in the lectures such as a 4-to-1 MUX and a 4-to-16 DEC were utilized along with a clock divider which is a relatively new concept with respect to where we are in the lectures. Considering the designed setup was implemented successfully and no major flukes were found during the physical implementations and simulations, the experiment can be considered a success. Considering the recent lectures, the clock divider could alternatively be implemented by using a number of positive-edge triggered D- flip flops to cut the frequency in half each implementation. However, such a setup would both prove too complicated for VHDL due to the number of necessary submodules to get the frequency down to an acceptable level as well as being limited to only being able to cut the frequency in half.

Appendices

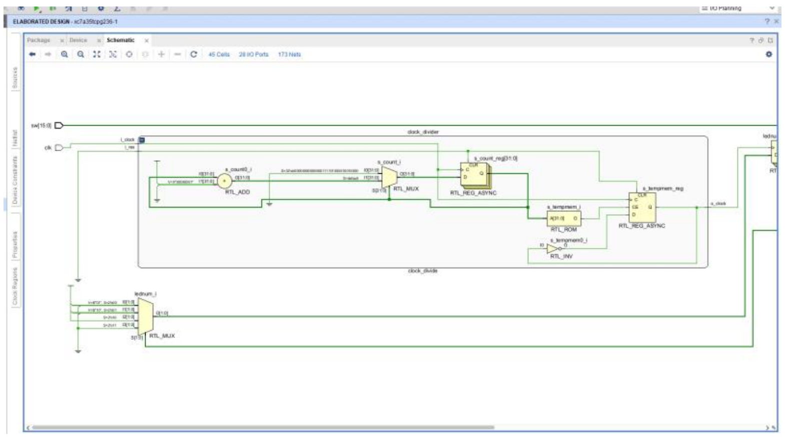

Figure 2.1: Left side of the RTL Schematic

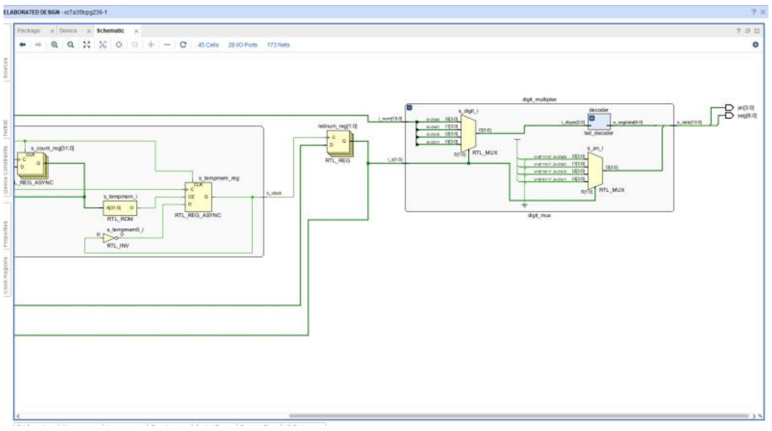

Figure 2.2: Right side of the RTL Schematic

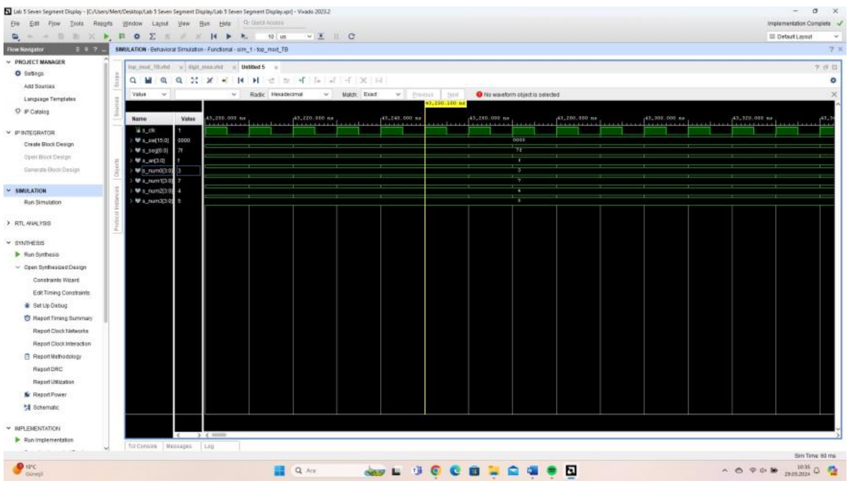

Figure 3.1: Simulated 100MHz clock

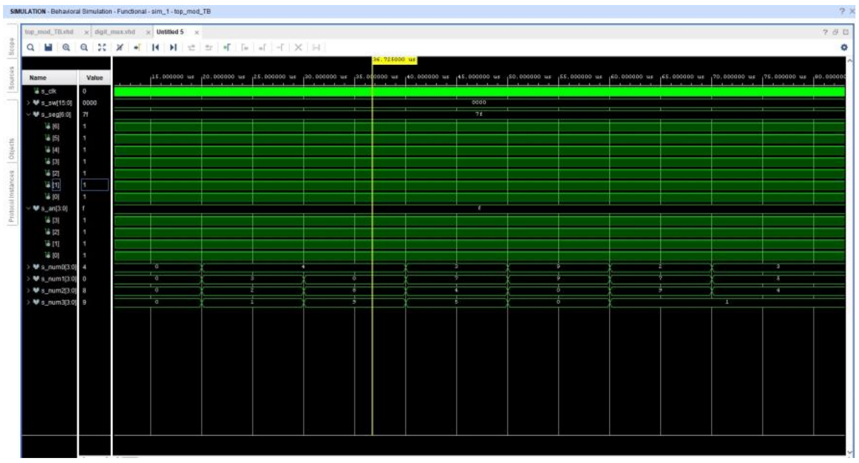

Figure 3.2: Testbench of the top module

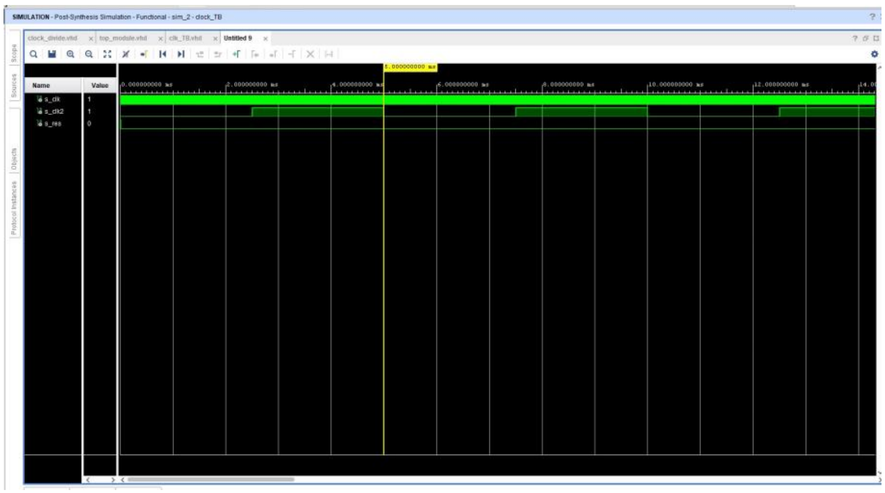
SIMULATION - Post-Synthesis Simulation - Functional - sim_2 - clock_TB
Name
Value
s_clk
s_clk2
s_res

Figure 3.3: Testbench of clock divider

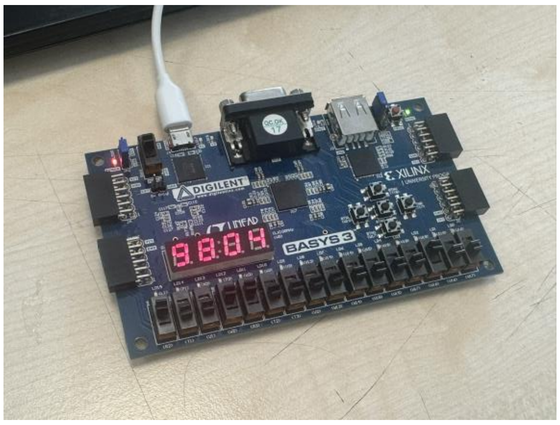

Figure 4.1: Output “9804”

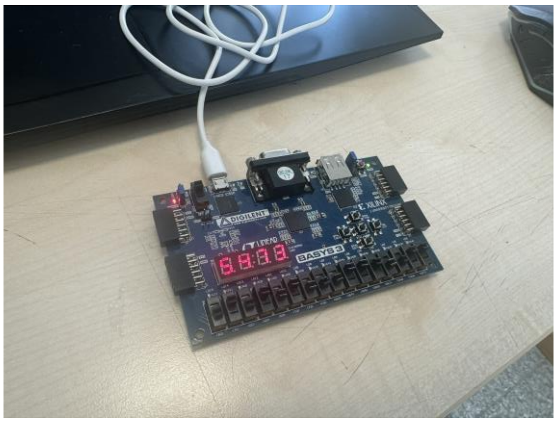

Figure 4.2: Output “5473”

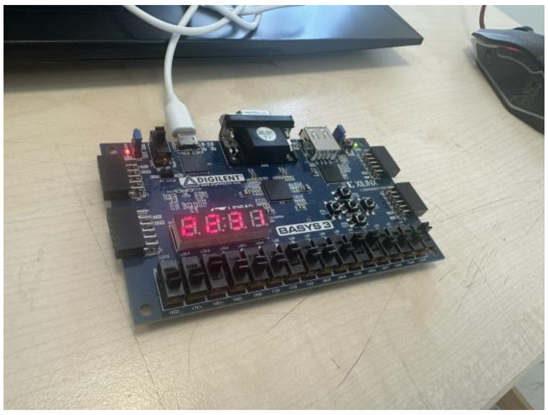

Figure 4.3: Output “2231”

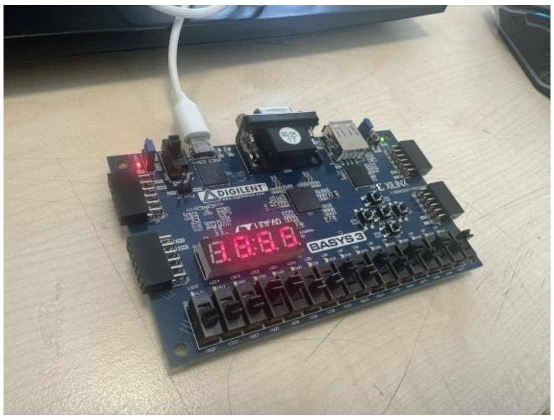

Figure 4.4: Output “1955”

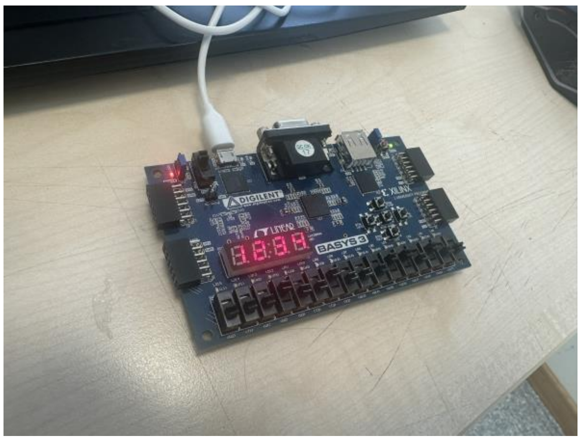

Figure 4.5: Output “1234”

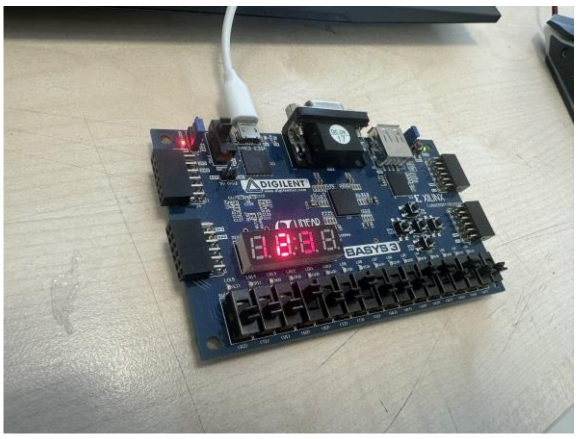

Figure 4.6: Cycling from “2” to “3”

Code of the VHDL file

top_module.vhd

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

entity top_module is
  Port (clk: in std_logic; --clock operating at 100MHz
  sw: in std_logic_vector (15 downto 0);
  seg: out std_logic_vector (6 downto 0);
```

```
   an : out std_logic_vector (3 downto 0)
    );
end top_module;

architecture Behavioral of top_module is




component clock_divide is
   Port (i_clock : in  STD_LOGIC;
      i_res  : in  STD_LOGIC;
      o_clock: out STD_LOGIC);
end component;



component digit_mux is
   Port (i_s: in std_logic_vector(1 downto 0);
   i_num : in std_logic_vector (15 downto 0);
   o_data : out std_logic_vector (10 downto 0));
end component;



signal current_number: std_logic_vector (15 downto 0);
signal clk_2: std_logic;
signal lednum: std_logic_vector (1 downto 0);
signal segan: std_logic_vector(10 downto 0);



begin


current_number <= sw;


clock_divider: clock_divide
```

```
    Port Map (i_clock => clk,
        i_res  => '0',
        o_clock => clk_2);



process(clk_2)
begin
  if rising_edge(clk_2) then
      case lednum is
         when "00" =>
           lednum <= "01";
         when "01" =>
           lednum <= "10";
         when "10" =>
           lednum <= "11";
         when "11" =>
           lednum <= "00";
         when others =>
           lednum <= "00";
         end case;
   end if;
end process;




digit_multiplex: digit_mux
    Port Map (i_s => lednum,
    i_num => current_number,
    o_data => segan);

an(3 downto 0) <= segan (10 downto 7);
seg(6 downto 0) <= segan(6 downto 0);
```

```
end Behavioral;
```

clock_divide.vhd

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

entity clock_divide is
   Port (i_clock : in  STD_LOGIC;
      i_res  : in  STD_LOGIC;
      o_clock: out STD_LOGIC);
end clock_divide;


architecture Behavioral of clock_divide is
   signal s_tempmem: STD_LOGIC;
   signal s_count : integer := 0;


begin

process (i_clock, i_res)
begin

if (i_res = '1') then
   s_count <= integer(0);
   s_tempmem <= '0';
elsif rising_edge(i_clock) then
         if (s_count = 249999) then         -- 100MHz / (250,000 * 2) = 200 Hz
            s_tempmem <= NOT(s_tempmem);
            s_count <= 0;
```

```
        else
           s_count <= s_count + integer(1);
        end if;
     end if;
  end process;

  o_clock <= s_tempmem;
end Behavioral;
```

digit_multiplex.vhd

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

entity digit_mux is
  Port (i_s: in std_logic_vector(1 downto 0);
  i_num : in std_logic_vector (15 downto 0);
  o_data : out std_logic_vector (10 downto 0));
end digit_mux;

architecture Behavioral of digit_mux is

component led_decoder is
  Port (i_dtype: in std_logic_vector(3 downto 0);
  o_segdata: out std_logic_vector(6 downto 0));
end component;


signal s_digit: std_logic_vector (3 downto 0);
signal s_segdata: std_logic_vector (6 downto 0);
signal s_an: std_logic_vector (3 downto 0);
```

```
begin

  with i_s select
    s_digit <= i_num (3 downto 0) when "00",
           i_num (7 downto 4) when "01",
           i_num (11 downto 8)  when "10",
           i_num (15 downto 12)  when "11",
           "1111" when others;

decoder: led_decoder
  Port Map (i_dtype => s_digit,
  o_segdata => s_segdata );

with i_s select
  s_an (3 downto 0) <= "1110" when "00",
     "1101" when "01",
     "1011" when "10",
     "0111" when "11",
     "1111" when others;

o_data(6 downto 0) <= s_segdata (6 downto 0);
o_data(10 downto 7) <= s_an (3 downto 0);

end Behavioral;
```

led_decoder.vhd

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

```
entity led_decoder is
   Port (i_dtype: in std_logic_vector(3 downto 0);
   o_segdata: out std_logic_vector(6 downto 0));
end led_decoder;

architecture Behavioral of led_decoder is

begin

   with i_dtype select
      o_segdata <= "1000000" when "0000",--0
                "1111001" when "0001",--1
                "0100100" when "0010",--2
                "0110000" when "0011",--3
                "0011001" when "0100",--4
                "0010010" when "0101",--5
                "0000010" when "0110",--6
                "1111000" when "0111",--7
                "0000000" when "1000",--8
                "0010000" when "1001",--9
                "1111111" when others;--all other possibilities

end Behavioral;
```

top_mod_TB.vhd

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

```
entity top_mod_TB is
end top_mod_TB;

architecture Behavioral of top_mod_TB is

  component top_module is
    Port (
      clk: in std_logic;
      sw: in std_logic_vector (15 downto 0);
      seg: out std_logic_vector (6 downto 0);
      an : out std_logic_vector (3 downto 0)
    );
  end component;

  signal s_clk: std_logic := '0';
  signal s_sw: std_logic_vector (15 downto 0) := (others => '0');
  signal s_seg: std_logic_vector (6 downto 0);
  signal s_an : std_logic_vector (3 downto 0);

  signal s_num0: std_logic_vector(3 downto 0);
  signal s_num1: std_logic_vector(3 downto 0);
  signal s_num2: std_logic_vector(3 downto 0);
  signal s_num3: std_logic_vector(3 downto 0);

begin

top_mod: top_module
 port map (clk => s_clk,
   sw => s_sw,
   seg => s_seg,
   an => s_an);

process
```

```
begin
 while now < 60000000 ns loop
   s_clk <= not s_clk;
   wait for 5 ns;
 end loop;
wait;
end process;


process
begin

-- this simulation trials 11 possible combinations out of 2^15

wait for 10000ns; --0000
        s_num3 <= "0000";
        s_num2 <= "0000";
        s_num1 <= "0000";
        s_num0 <= "0000";

    wait for 10000ns; --1234
        s_num3 <= "0001";
        s_num2 <= "0010";
        s_num1 <= "0011";
        s_num0 <= "0100";

    wait for 10000ns; --9804
        s_num3 <= "1001";
        s_num2 <= "1000";
        s_num1 <= "0000";
        s_num0 <= "0100";

    wait for 10000ns; --5473
        s_num3 <= "0101";

        s_num2 <= "0100";

        s_num1 <= "0111";

        s_num0 <= "0011";


    wait for 10000ns; --0099

        s_num3 <= "0000";

        s_num2 <= "0000";

        s_num1 <= "1001";

        s_num0 <= "1001";


    wait for 10000ns; --1972

        s_num3 <= "0001";

        s_num2 <= "1001";

        s_num1 <= "0111";

        s_num0 <= "0010";


    wait for 10000ns; --1453

        s_num3 <= "0001";

        s_num2 <= "0100";

        s_num1 <= "0101";

        s_num0 <= "0011";


    wait for 10000ns; --2231

        s_num3 <= "0010";

        s_num2 <= "0010";

        s_num1 <= "0011";

        s_num0 <= "0001";


    wait for 10000ns; --5443

        s_num3 <= "0101";

        s_num2 <= "0100";

        s_num1 <= "0100";
```

```
        s_num0 <= "0011";

    wait for 10000ns; --1955
        s_num3 <= "0001";
        s_num2 <= "1001";
        s_num1 <= "0101";
        s_num0 <= "0101";

    wait for 10000ns; --0573
        s_num3 <= "0000";
        s_num2 <= "0101";
        s_num1 <= "0111";
        s_num0 <= "0011";



        wait;
    end process;

end Behavioral;
```

clk_TB.vhd

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

entity clock_TB is
end clock_TB;

architecture Behavioral of clock_TB is
```

```
component clock_divide is
  Port (i_clock : in  STD_LOGIC;
    i_res   : in  STD_LOGIC;
    o_clock : out STD_LOGIC);
 end component;

signal s_clk  : std_logic := '0';
signal s_clk2 : std_logic := '0';
signal s_res  : std_logic := '0';

begin

gen_clk: process
begin
  while now < 60000000 ns loop
    s_clk <= not s_clk;
    wait for 5 ns;
  end loop;
  wait;
end process gen_clk;

reset_proc: process
begin
  s_res <= '1';
  wait for 300 ns;
  s_res <= '0';
  wait;
end process reset_proc;

clockdivider: clock_divide
  port map (i_clock => s_clk,
    i_res   => s_res,
    o_clock => s_clk2);


end Behavioral;
```

Citations

Bobrowicz, Sam. “Basys 3 Reference.” Basys 3 Reference - Digilent Reference, digilent.com/reference/basys3/refmanual. Accessed 29 Mar. 2024.