Seven Segment LED Driver

Digital Design

Jacobs University Bremen

Priontu Chowdhury

MAILBOX: NB-374

Place of Execution: Teaching Lab EE, Room 56, Research 1
Date of Execution: 15/03/2021
Instructor: PhD Fangning Hu

Introduction

In this lab we learned how to control and program the seven-segment display of the FPGA board using VHDL. We accomplished this task by using the 8 switches on the FPGA board to provide a 4-bit binary input to the leftmost and the second leftmost 7-segment displays. The sum of those is to be printed on the consecutive two displays. The program translated the input into a binary output containing 7 bits for each of the displays, which were converted to “on” or “off” for each of the segments in order to provide us a meaningful output to the user.

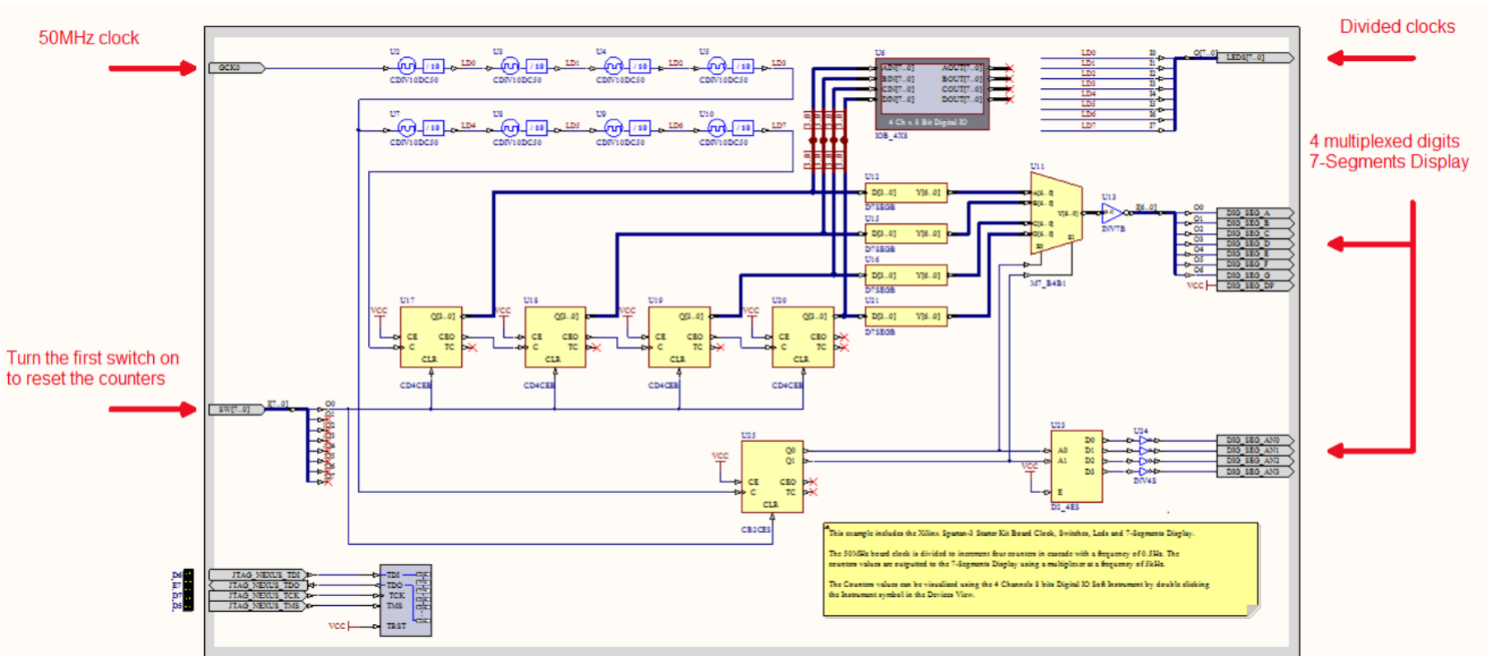
Instruments

Spartan-3 Starter Kit Board (includes switches, ports and 7-segment display).

Assignment

The leftmost digit should display the hexadecimal value representing the 4 leftmost switches. The next digit should show the hexadecimal value representing the 4 rightmost switches. The last two digits should show the arithmetic sum of the hexadecimal values in the first two digits.

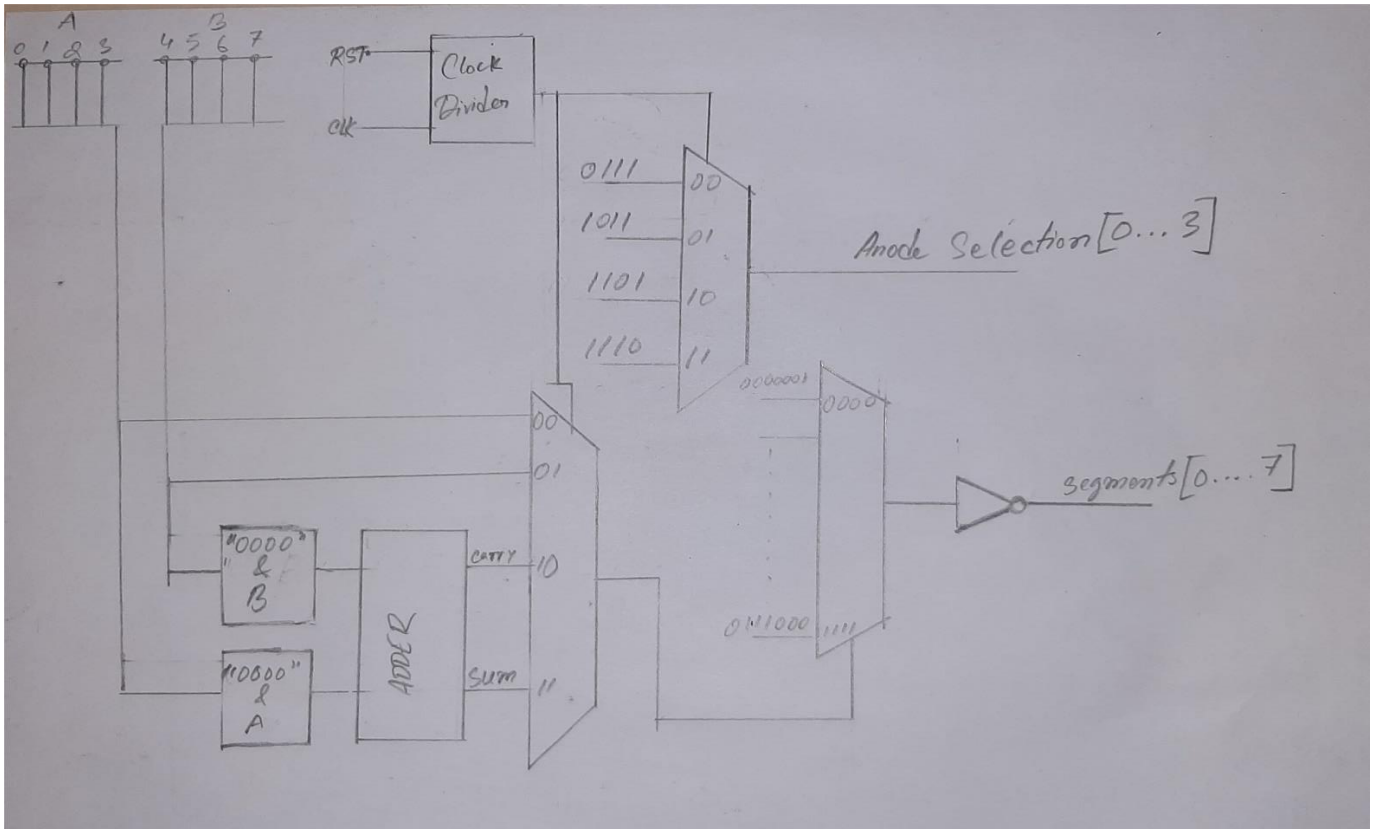
Circuit Diagram



The circuit we designed has the capability to handle addition of two 4-bit binary numbers, obtain a result for the character to be displayed, and create a bit pattern for the desired display.

Concept Diagram

A concept diagram for the logical implementation of the circuit involving MUXes, clock and an adder:



Refresh Rate

We use a 15-bit counter to slow down the clock. Therefore, we refresh the frame after every 2^{15} counts, resulting in the following refresh rate:

$$Refresh\ Rate = Frequency = \frac{50MHz}{2^{15}} = 1525.8789Hz = 0.0015259MHz$$

Design file

```
----------------------------------------------------------------------
-- Author      : Welson Sun
-- Affiliation : Brigham Young University
-- Description : Seven segment display timing control
----------------------------------------------------------------------

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

entity seg_ctrl is
  port(
    gclk     : in  std_logic;                  -- Global clock input, 50MHz
    rst      : in  std_logic;                  -- Global reset input
    switches : in std_logic_vector(7 downto 0);  -- Input switches
    segments : out std_logic_vector(6 downto 0); -- Segment control
    an_sel   : out std_logic_vector(3 downto 0)  -- Anode control
    );
end seg_ctrl;

architecture rtl of seg_ctrl is

  -- A 15 bit counter
  signal count : unsigned(14 downto 0);
  signal position : unsigned(1 downto 0);

begin

  -- Clock divider
  process (gclk, rst)
  begin
    if gclk='1' and gclk'event then
      if rst='1' then                 -- Is this a synchronous or an
        count <= (others=>'0');       -- asynchronous reset?
      else
        count <= count+1;
      end if;
              end if;
  end process;

  -- Anode decode circuitry
  position <= count(14 downto 13);

  with position select
  an_sel <=
    "1000" when "00",
    "0100" when "01",
    "0010" when "10",
    "0001" when others;
end rtl;
```

Source Code

Top Design

The following slot contains the top design code of the project that includes multiple files. All the files are connected to the top design files, which contains the execution results of all the files and puts them together to create an output.

The ports declared in this file are the 8 switches, the 4 displays, seven segments, the clock and the reset. These are first port mapped to ports of the segmentdriver.vhd file that handle the logic for these ports within that file. Other than that, the ports from segmentdriver.vhd that handle the logic for the displays are mapped to the "STD_LOGIC_VECTOR" bit-arrays Ai, Bi, Ci and Di. Now, using these arrays, we manipulate the values on the 7-segment displays. We show the converted hexadecimal value from the binary input of the left four switches on the leftmost 7-segment display using the variable Ai, and the converted hexadecimal value from the binary input of the right four switches on the second leftmost 7-segment display using the variable Bi. Then we add the 4-bit binary values of Ai and Bi and obtain a 5-bit result that we convert to 8 bits. We store the first 4 bits of this result in Ci, and the last 4 bits in Di. These results are then also shown in the display. Consequently, the converted hexadecimal value of the sum of the leftmost and second leftmost displays are shown in the consecutive two displays.

topdesign.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    13:44:24 03/26/2021
-- Design Name:
-- Module Name:    topdesign - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following library declaration if using
```

```
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity topdesign is                                    --putting everything together
    Port ( switch0 : in  STD_LOGIC;
           switch1 : in  STD_LOGIC;
           switch2 : in  STD_LOGIC;
           switch3 : in  STD_LOGIC;
           switch4 : in  STD_LOGIC;
           switch5 : in  STD_LOGIC;
           switch6 : in  STD_LOGIC;
           switch7 : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           rst : in STD_LOGIC;
           topselDispA : out  STD_LOGIC;
           topselDispB : out  STD_LOGIC;
           topselDispC : out  STD_LOGIC;
           topselDispD : out  STD_LOGIC;
           topsegA : out  STD_LOGIC;
           topsegB : out  STD_LOGIC;
           topsegC : out  STD_LOGIC;
           topsegD : out  STD_LOGIC;
           topsegE : out  STD_LOGIC;
           topsegF : out  STD_LOGIC;
           topsegG : out  STD_LOGIC);
end topdesign;

architecture Behavioral of topdesign is

component segmentdriver                                --using the driver component
           Port(
           display_A : in  STD_LOGIC_VECTOR (3 downto 0);
           display_B : in  STD_LOGIC_VECTOR (3 downto 0);
           display_C : in  STD_LOGIC_VECTOR (3 downto 0);
           display_D : in  STD_LOGIC_VECTOR (3 downto 0);
           segA : out  STD_LOGIC;
           segB : out  STD_LOGIC;
           segC : out  STD_LOGIC;
           segD : out  STD_LOGIC;
           segE : out  STD_LOGIC;
           segF : out  STD_LOGIC;
```

```
        segG : out  STD_LOGIC;
        select_Display_A : out  STD_LOGIC;
        select_Display_B : out  STD_LOGIC;
        select_Display_C : out  STD_LOGIC;
        select_Display_D : out  STD_LOGIC;
        clk : in  STD_LOGIC;
                        rst : in STD_LOGIC
         );
end component;

SIGNAL Ai : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL Bi : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL Ci : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL Di : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL digit1 : UNSIGNED (4 DOWNTO 0);
SIGNAL digit2 : UNSIGNED (4 DOWNTO 0);
SIGNAL res: STD_LOGIC_VECTOR (7 DOWNTO 0);

begin
uut2: segmentDriver PORT MAP(                    --mapping all required ports
        display_A => Ai,
        display_B => Bi,
        display_C => Ci,
        display_D => Di,
        segA => topsegA,
        segB => topsegB,
        segC => topsegC,
        segD => topsegD,
        segE => topsegE,
        segF => topsegF,
        segG => topsegG,
        select_Display_A => topselDispA,
        select_Display_B => topselDispB,
        select_Display_C => topselDispC,
        select_Display_D => topselDispD,
        clk => clk,
        rst => rst
);

Ai(0) <= switch0;
Ai(1) <= switch1;
Ai(2) <= switch2;
Ai(3) <= switch3;
Bi(0) <= switch4;
Bi(1) <= switch5;
Bi(2) <= switch6;
```

```
Bi(3) <= switch7;

digit1 <= UNSIGNED("0"&Ai);                --converting Ai to unsigned 5-bit array
digit2 <= UNSIGNED("0"&Bi);
res <= "000"&(digit1 + digit2)             --calculating sum and converting to 8-bit array

Di <= res(3 DOWNTO 0);                     --putting second half of final array in Di
Ci <= res(7 downto 4);                     --putting first half of final array in Ci

end Behavioral;
```

Segment Driver

In segmentdriver.vhd we connect together the clock divider and segment decoder components, and also connect these components to the top design component. This file acts as a layer/medium between the top design components and its components, and also solves some of the required underlying logic. In this file, ports for display, display selection, clock, reset, and segments are declared. In the architecture we use the segment decoder and clock divider components. Then we map the ports of these components to the ports of segment driver. We use these ports to slow down the clock. We also select the display to map the output on in this file.

segmentdriver.vdh

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    12:39:21 03/26/2021
-- Design Name:
-- Module Name:    segmentdriver - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity segmentdriver is
    Port ( display_A : in  STD_LOGIC_VECTOR (3 downto 0);
```

```
        display_B : in  STD_LOGIC_VECTOR (3 downto 0);
        display_C : in  STD_LOGIC_VECTOR (3 downto 0);
        display_D : in  STD_LOGIC_VECTOR (3 downto 0);
        segA : out  STD_LOGIC;
        segB : out  STD_LOGIC;
        segC : out  STD_LOGIC;
        segD : out  STD_LOGIC;
        segE : out  STD_LOGIC;
        segF : out  STD_LOGIC;
        segG : out  STD_LOGIC;
        select_Display_A : out  STD_LOGIC;
        select_Display_B : out  STD_LOGIC;
        select_Display_C : out  STD_LOGIC;
        select_Display_D : out  STD_LOGIC;
        clk : in  STD_LOGIC;
        rst : in STD_LOGIC
                        );
end segmentdriver;

architecture Behavioral of segmentdriver is

        COMPONENT segmentDecoder            --using components constructed before
        PORT(
                        Digit : in std_logic_vector(3 downto 0);
                        segment_A : out STD_LOGIC;
                        segment_B : out STD_LOGIC;
                        segment_C : out STD_LOGIC;
                        segment_D : out STD_LOGIC;
                        segment_E : out STD_LOGIC;
                        segment_F : out STD_LOGIC;
                        segment_G : out STD_LOGIC
                );
        END COMPONENT;

        COMPONENT clock_divider             --using components constructed before
        PORT(
                        clk : in STD_LOGIC;
                        enable : in STD_LOGIC;
                        reset : in STD_LOGIC;
                        data_clk : out STD_LOGIC_VECTOR (14 downto 0)
        );
        END COMPONENT;
SIGNAL temporary_data : std_logic_vector(3 downto 0);
SIGNAL clock_word : std_logic_vector (14 downto 0);
SIGNAL slow_clock : std_logic;
```

```
begin
	-- Component Instantiation
		uut: segmentDecoder PORT MAP(

				Digit => temporary_data,
				segment_A => segA,
				segment_B => segB,
				segment_C => segC,
				segment_D => segD,
				segment_E => segE,
				segment_F => segF,
				segment_G => segG
		);

		uut1: clock_divider PORT MAP(

				clk => clk,
				enable => '1',
				reset => '0',
				data_clk => clock_word
		);
slow_clock <= clock_word(14);                                   --slowing down clock

PROCESS(slow_clock)
	variable display_selection : STD_LOGIC_VECTOR(1 DOWNTO 0);
	BEGIN
	if slow_clock'event and slow_clock = '1' then

	case display_selection is
                                                                --anode selection
	when "00" => temporary_data <= display_A;

			select_Display_A <= '0';
			select_Display_B <= '1';
			select_Display_C <= '1';
			select_Display_D <= '1';

			display_selection := display_selection+'1';

	when "01" => temporary_data <= display_B;

			select_Display_A <= '1';
			select_Display_B <= '0';
			select_Display_C <= '1';
			select_Display_D <= '1';
```

```
                        display_selection := display_selection+'1';

        when "10" => temporary_data <= display_C;

                        select_Display_A <= '1';
                        select_Display_B <= '1';
                        select_Display_C <= '0';
                        select_Display_D <= '1';

                        display_selection := display_selection+'1';

        when others => temporary_data <= display_D;

                        select_Display_A <= '1';
                        select_Display_B <= '1';
                        select_Display_C <= '1';
                        select_Display_D <= '0';

                        display_selection := display_selection+'1';
        end case;
end if;

END PROCESS;

end Behavioral;
```

Segment Decoder

In segmentdecoder.vhd, we decode the segments such that we obtain output based on the input from the switches. Different combinations of the input correspond to specific 4-bit inputs, which are converted to 7-bit inputs that are to be mapped onto each of the segments of the 7-segment display.

segmentdecoder.vhd

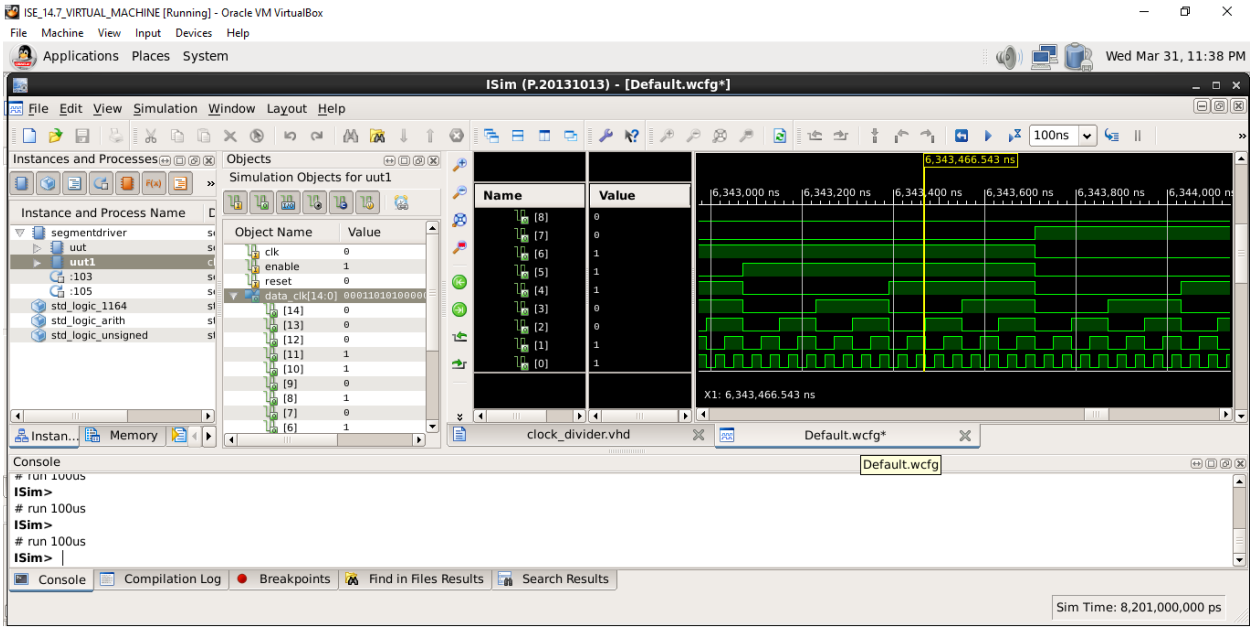
```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:58:03 03/26/2021
-- Design Name:
-- Module Name:    segmentdecoder - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity segmentdecoder is
    Port ( Digit : in  STD_LOGIC_VECTOR (3 downto 0);
           segment_A : out  STD_LOGIC;                          --segments in display
           segment_B : out  STD_LOGIC;
           segment_C : out  STD_LOGIC;
           segment_D : out  STD_LOGIC;
           segment_E : out  STD_LOGIC;
```

```
        segment_F : out  STD_LOGIC;
        segment_G : out  STD_LOGIC);
end segmentdecoder;

architecture Behavioral of segmentdecoder is

begin
process(Digit)
        variable Decode_Data: std_logic_vector(6 downto 0);

        begin
        case Digit is
                when "0000" => Decode_Data := "1111110"; --0
                when "0001" => Decode_Data := "0110000"; --1
                when "0010" => Decode_Data := "1101101"; --2
                when "0011" => Decode_Data := "1111001"; --3
                when "0100" => Decode_Data := "0110011"; --4
                when "0101" => Decode_Data := "1011011"; --5
                when "0110" => Decode_Data := "1011111"; --6
                when "0111" => Decode_Data := "1110000"; --7
                when "1000" => Decode_Data := "1111111"; --8
                when "1001" => Decode_Data := "1111011"; --9
                when "1010" => Decode_Data := "1110111"; --A
                when "1011" => Decode_Data := "0011111"; --B
                when "1100" => Decode_Data := "1001110"; --C
                when "1101" => Decode_Data := "0111101"; --D
                when "1110" => Decode_Data := "1001111"; --E
                when "1111" => Decode_Data := "1000111"; --F
                when others => Decode_Data := "0111110"; --Error 'H'
        end case;
                segment_A <= not Decode_Data(6);                        --decoding data
                segment_B <= not Decode_Data(5);    --“not” used because segments are active
                segment_C <= not Decode_Data(4);    -- low output
                segment_D <= not Decode_Data(3);
                segment_E <= not Decode_Data(2);
                segment_F <= not Decode_Data(1);
                segment_G <= not Decode_Data(0);

end process;

end Behavioral;
```

Clock Divider

The clock divider was provided to us as part of the exercise. We achieve our task by implementing a 15-bit counter. We keep the reset signal set to '0', and increment the counter at each positive cycle of the input clock, which is 50MHz. The counter goes to zero as soon as the reset signal is asserted.

The simulation is provided below:



In this picture, we see the first few bits of the counter, and the simulation for the division of the clock. As we can see, the subsequent signal is twice as slower for each iteration. This is because, as we keep adding '1' to the counter, it takes twice the time for each bit in the 15 bit-array to come back to '1' as we go up the levels in the counter. This means, for the nth level, it takes 2^n longer, so the clock is divided by 2^n. We use a 15-bit counter, so the first division takes 2^1, the second takes 2^2, and this goes on until the last division which takes 2^{15} times longer.

The result we see above will be obtained after many iterations. For our lab, however, we are told to set the first force the clock with the following settings:

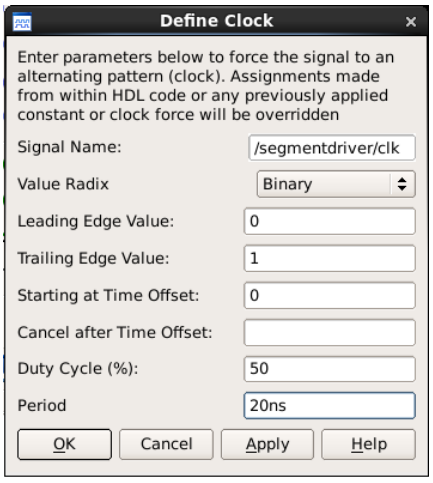


Figure: Clock Settings

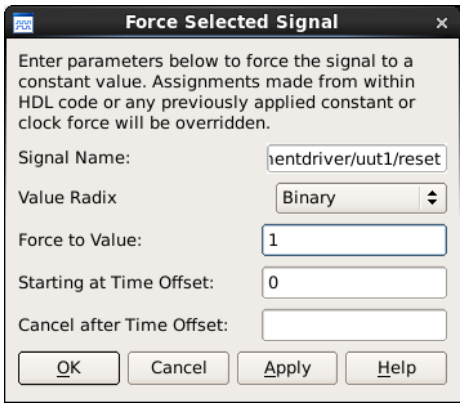
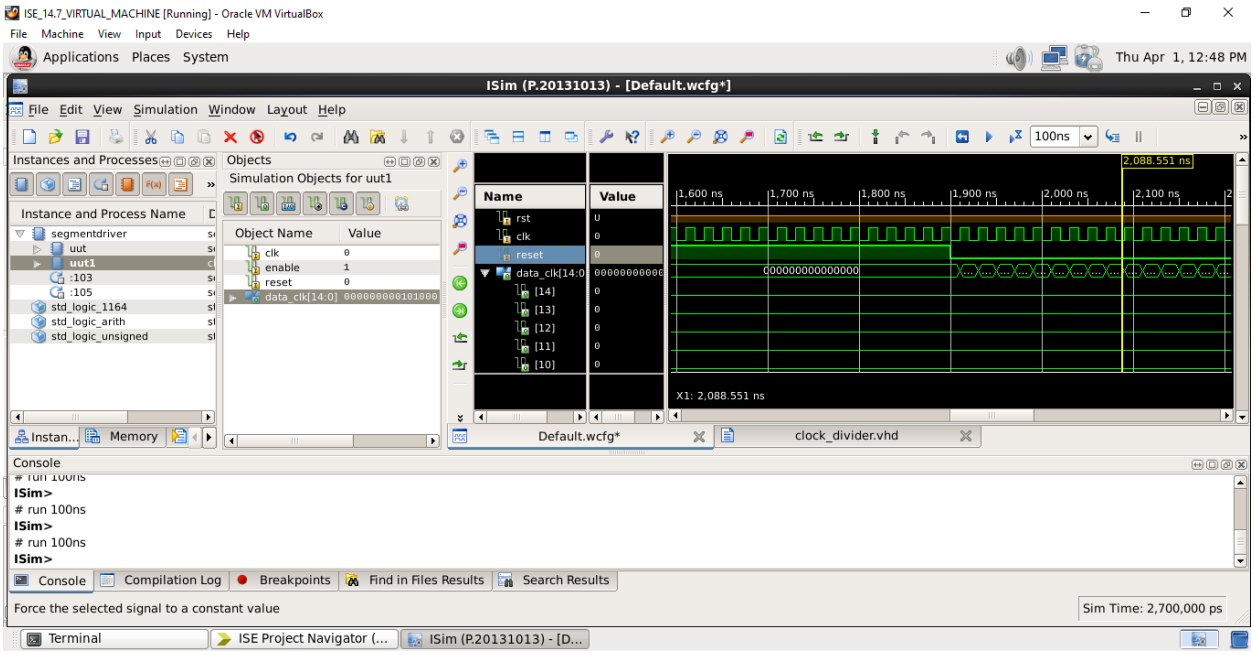


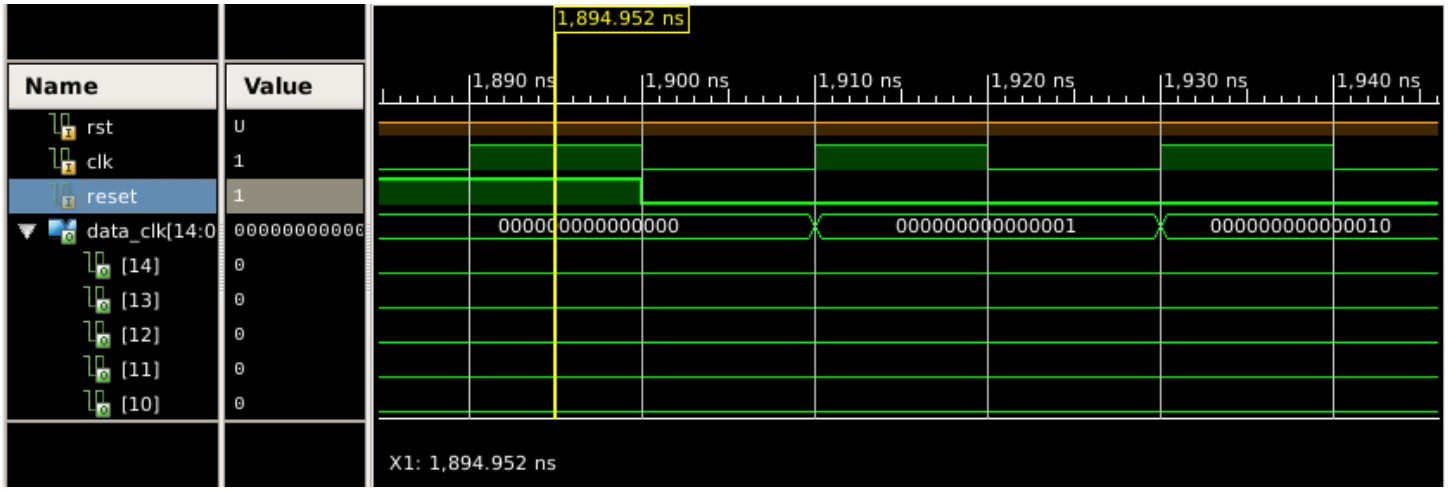
Figure: Reset Settings

Then, we run the simulation for 100ns. After that, we change the force constant for reset to ‘0’ and run the simulation again for 100ns. We see the following changes between the two settings:

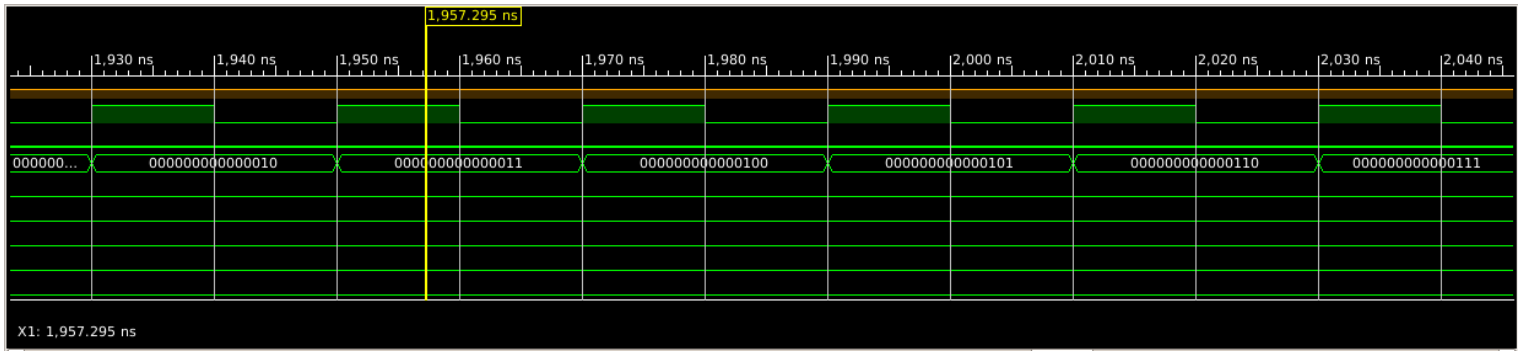


We see that data_clk, which is our counter, goes through some changes between the settings reset = ‘1’ and reset = ‘0’. The counter signal is constant for reset = ‘1’ because the clock resets on every iteration. Then, when we change the reset to reset = ‘0’, clock division starts. The system clock runs at 50MHz at all times, which is too fast for our task. Therefore, we introduce a counter to the clock and update it every time the 15th bit of our counter is ‘1’. This slows down the clock, because it takes 2^{15} times the original clock time for the 15th bit of the counter to cycle back to ‘1’. We can see the counting occurring in the following pictures:

- Reset changed from ‘1’ to ‘0’, and counting starts:



- Counting continues at Reset = ‘0’:



Now that we know how the clock is slowing down, we can answer the question posed in the manual. We are told that in the VHDL code a 2-bit slice is peeled off the counter to determine which anode should be turned on. We see this happening in the following piece of code in segmentdriver.vhd:

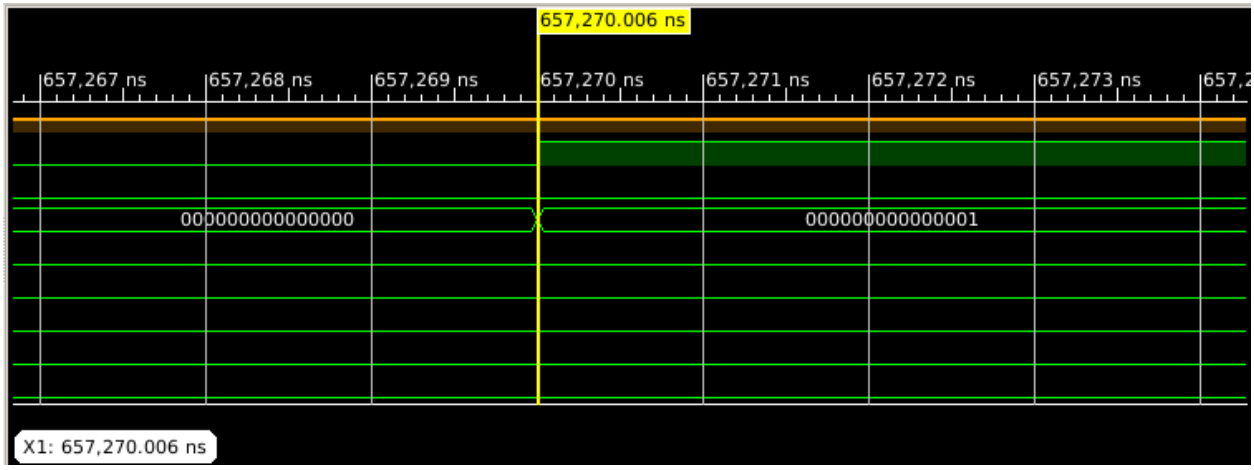
```
slow_clock <= clock_word(14);                    --slowing down clock

PROCESS(slow_clock)
 variable display_selection : STD_LOGIC_VECTOR(1 DOWNTO 0);
 BEGIN
 if slow_clock'event and slow_clock = '1' then

 case display_selection is
                                                  --anode selection
 when "00" => temporary_data <= display_A;

                select_Display_A <= '0';
                select_Display_B <= '1';
                select_Display_C <= '1';
                select_Display_D <= '1';

                display_selection := display_selection+'1';

 when "01" => temporary_data <= display_B;

                select_Display_A <= '1';
                select_Display_B <= '0';
                select_Display_C <= '1';
                select_Display_D <= '1';

                display_selection := display_selection+'1';

 when "10" => temporary_data <= display_C;
```

```
                select_Display_A <= '1';
                select_Display_B <= '1';
                select_Display_C <= '0';
                select_Display_D <= '1';

                display_selection := display_selection+'1';

 when others => temporary_data <= display_D;

                select_Display_A <= '1';
                select_Display_B <= '1';
                select_Display_C <= '1';
                select_Display_D <= '0';

                display_selection := display_selection+'1';
 end case;
end if;

END PROCESS;
```

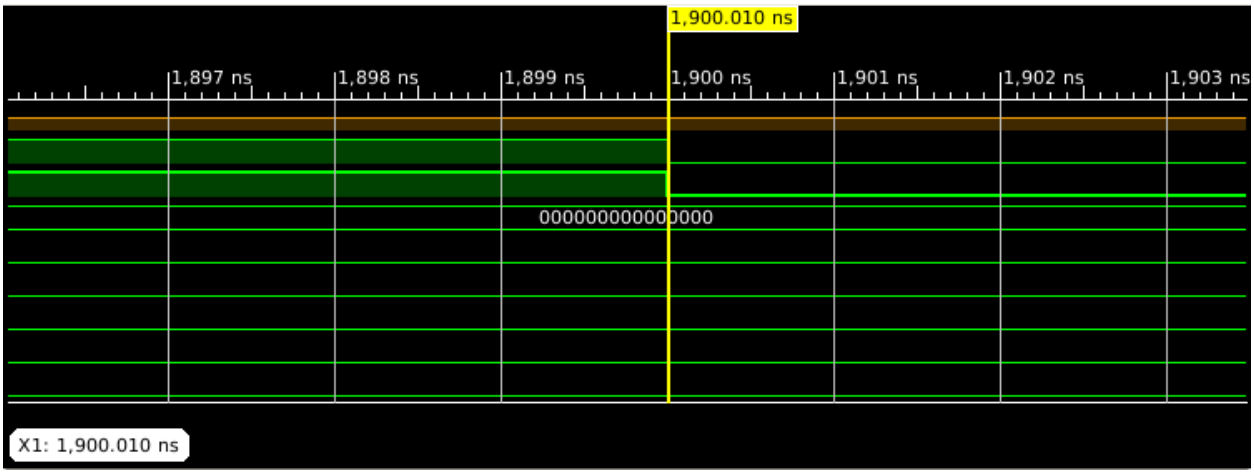
In the previous lines, the data_clk counter is passed on to clock_word, and the bit in the 14th position of the clock_word bit-array is passed on to slow clock. When this bit is ‘1’, we choose an anode and increment the selector for the display, which in turn selects the next display. This selector uses the 2-bit slice of the counter (in my code, the selector is rather a separate counter that mimics the 2-bit slice). This 2-bit slice refers to the 14th and 15th bits in the array.

From the code, we see that the cycle starts when the 2-bit slice is ‘00’, and it ends when the slice is back to ‘00’. We can find time taken using the following snippets from the simulation:

Ending point:



Starting point:



Using the simulation data, we can obtain time taken and frequency as follows:

$Time\ taken\ =\ 657{,}270.006\ ns\ -\ 1{,}900.010\ ns$ $Time\ taken\ =\ 655369.996\ ns$ $Time\ taken\ =\ 6.5537 \times 10^{-4}\ s$	$Frequency\ =\ \frac{1}{Time\ Taken}$ $Frequency = \frac{1}{6.5537 \times 10^{-4}}$ $Frequency = 1525.8556 Hz$ $Frequency\ =\ 0.0015259\ MHz$

As we can see, the value we calculated from the simulation matches the value we obtained from our theoretical calculation earlier.

The source code for the clock divider is provided below:

clock_divider.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    12:26:17 03/26/2021
-- Design Name:
-- Module Name:    clock_divider - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
```

```
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity clock_divider is
    Port ( clk : in  STD_LOGIC;            --Global clock input
           enable : in  STD_LOGIC;         --Global reset input
           reset : in  STD_LOGIC;
           data_clk : out  STD_LOGIC_VECTOR (14 downto 0));
end clock_divider;

architecture Behavioral of clock_divider is

begin
           process(clk, reset)
           variable count: STD_LOGIC_VECTOR(14 downto 0) := (others=>'0');
           begin

           if reset = '1' then
                    count := (others=>'0');
           elsif enable = '1' and clk'event and clk='1' then
                    count := count + 1;
           end if;
           data_clk <= count;
           end process;

end Behavioral;
```

Constraints File

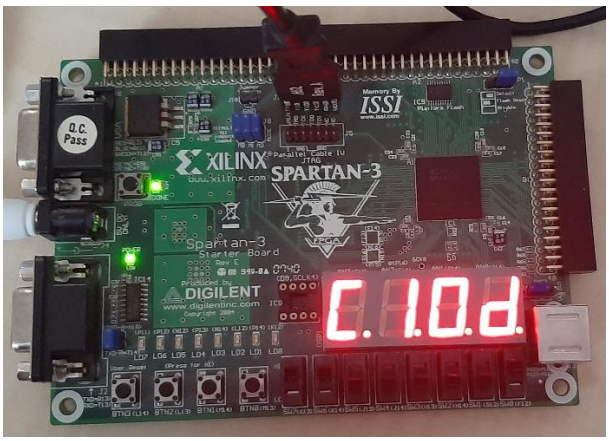
The constraints file is used to map the ports we used in our program to the associated pins on the FPGA board. In the following .ucf file, we map 8 switches used to provide input, 1 clock for counting, 1 button for reset, 7 segments for each segment in a display output, and 4 pins for anode selection.

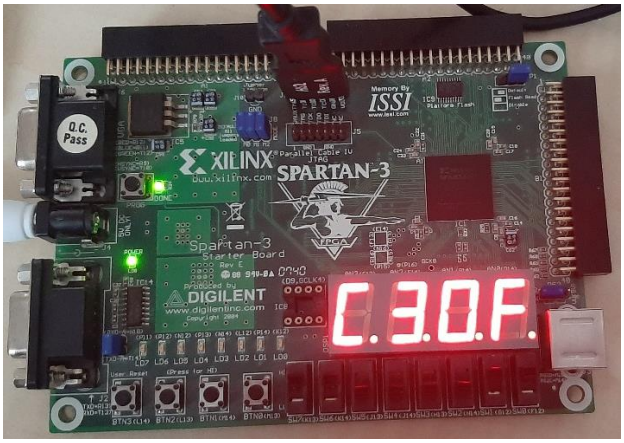
XilinxIntro.ucf

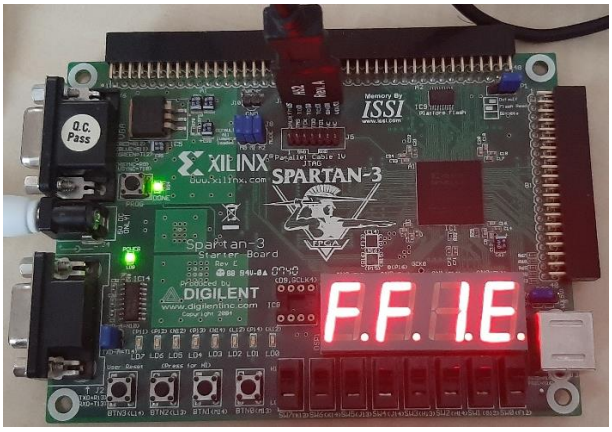
```
#switches
NET "switch0" LOC="F12" | IOSTANDARD=LVTTL;
NET "switch1" LOC="G12" | IOSTANDARD=LVTTL;
NET "switch2" LOC="H14" | IOSTANDARD=LVTTL;
NET "switch3" LOC="H13" | IOSTANDARD=LVTTL;
NET "switch4" LOC="J14" | IOSTANDARD=LVTTL;
NET "switch5" LOC="J13" | IOSTANDARD=LVTTL;
NET "switch6" LOC="K14" | IOSTANDARD=LVTTL;
NET "switch7" LOC="K13" | IOSTANDARD=LVTTL;

#clock and reset
NET "clk" LOC="T9";
NET "rst" LOC="M13" | IOSTANDARD=LVTTL;

#Anode selection
NET "topselDispA" LOC = "F14" | IOSTANDARD=LVCMOS33;
NET "topselDispB" LOC = "E13" | IOSTANDARD=LVCMOS33;
NET "topselDispC" LOC = "G14" | IOSTANDARD=LVCMOS33;
NET "topselDispD" LOC = "D14" | IOSTANDARD=LVCMOS33;

#7 segment display
NET "topsegA" LOC = "E14" | IOSTANDARD=LVCMOS33;
NET "topsegB" LOC = "G13" | IOSTANDARD=LVCMOS33;
NET "topsegC" LOC = "N15" | IOSTANDARD=LVCMOS33;
NET "topsegD" LOC = "P15" | IOSTANDARD=LVCMOS33;
NET "topsegE" LOC = "R16" | IOSTANDARD=LVCMOS33;
NET "topsegF" LOC = "F13" | IOSTANDARD=LVCMOS33;
NET "topsegG" LOC = "N16" | IOSTANDARD=LVCMOS33;
```

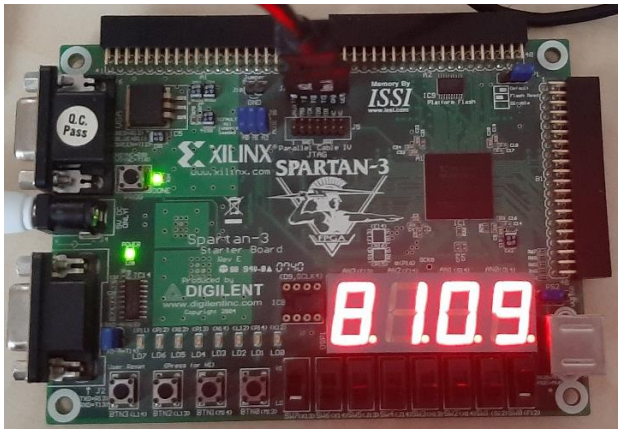
Results

Pictures of the results have been provided below:









As we can see, the two left 7-segment displays show hexadecimal values and the right two displays show the sum of the left two values.

A video implementation can be found here: https://youtu.be/lNhFvXdh0CM

Conclusion

In this lab, we learned how to use switches on the Spartan-3 FPGA board to provide input to the board. Also, we learned how to translate these inputs into hexadecimal output on the 7-segment display of the board. We learned how to use MUXes to model our circuit, and how to implement it on the FPGA board. Moreover, we learned the use of the clock divider, how variables like time and frequency become important in our implementation, and how to slow down the clock in order to obtain our desired results.

Reference

[1] http://fpga-fhu.user.jacobs-university.de/wp-content/uploads/2014/10/S3BOARD-rm.pdf

[2] http://fpga-fhu.user.jacobs-university.de/?page_id=34

[3] https://designcontent.live.altium.com/NanoBoardExampleDetail/3rd_Party_Xilinx_Spartan3

[4] https://www.youtube.com/watch?v=2Bvf1GdutFI