University of Montenegro
Faculty of Electrical Engineering

Course: Automated Design of Electrical Circuits and Systems

Two Digit Seven Segment Counter

Authors
Marko Marković
Stefan Šćepanović
Nina Blagojević
Milena Božović

Mentor
Prof. dr Radovan Stojanović

November 19, 2019

Contents

1 Summary

This is a laboratory project, made at the Faculty of Electrical Engineering, University of Montenegro, with mentoring of prof. dr Radovan Stojanovic. The problem is presented in detail with our proposed solution. The following is included: high level design and description of the solution, with hardware and software structure. One of the goals of this assignment is to demonstrate how to control a FPGA DE2-70 board, on which we did a simulation. This was done using Quartus 9.1 Altera software.

2 Problem Description

The task was to create a two digit seven segment counter, which is able to count at a rate of 0.5Hz, 1Hz, 2Hz or 4Hz, using a FPGA DE-2 70 board.

3 Solution

Our approach to creating a solution was to split the problem in smaller parts and then solve each one separately. The next step was to do computer simulations to test if everything would work as expected, this would make it easier to localise any errors that might come up.

The software that we used was ltera Quartus 9.1, which is approved and tested for all FPGA and CPDL based systems. The possibility to make block diagrams out of vhdl files and block diagrams, and to use them in different block diagrams suited our approach quite well too.

3.1 High level design

The first step was to analyse what the outputs of the circuit should be, what inputs to control the circuit we needed, and what the circuit should do with the inputs to get the desired outputs.

3.1.1 Input and Output

Determining the outputs was easy enough, as we had to control two seven segment displays, all we needed were 14 outputs, one for each segment.

As for the inputs, in the problem it is stated that we need to be able to choose between four counting rates, this we could do with two on/off switches, as that would give us a total of four input combinations: ”00”, ”01”, ”10” and ”11”, where the 0 represents the ”off” and the 1 the ”on” state of the switch. For resetting the counter to zero we used an additional switch, which when set to ”on” would keep return the counter to zero and prevent it from counting until set back to ”off”. The final input that we needed was a clock signal.

3.1.2 The circuit

The most obvious part of the circuit, given the problem, is a counter which increases its binary output values by one each time the clock signal, that is fed to it, changes from zero to one. Also this counter needs to reset to zero when it reaches the number 99. For a basic counter, first we needed to make a D-flip flop. As we had to control two seven segment displays, we decided to modify a basic counter so that it counted in BCD code and not in binary.

For the counter to count at the desired frequencies, we needed a circuit that modifies the input clock of the board, which is 50MHz, so it has the desired frequency. As the highest needed frequency was 4Hz, and the others were 2, 4 and 8 times lower respectively, we divided the problem into three parts: make a circuit that turns a 50MHz clock into a 4hz one; make a circuit that divides it's input clock by two and use three of those to make the other desired clock signals; make a circuit that selects which of these signals is sent into the counter.

And finally we needed a circuit that turns a binary number, which is between zero and nine (as we used a BCD counter) into appropriate outputs for the display.

3.2 D-FlipFlop

A D-flipflop is a basic digital circuit that changes its output value to its input value every time the clock signal changes from zero to one, or from one to zero, depending on the realisation. It also has a reset input to change the output to zero independently of its input.

As this was a really simple circuit, we made it in vhdl. The code is given below.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mydff IS
   PORT (d, clk, rst: IN STD_LOGIC;
      q: OUT STD_LOGIC);
END mydff;
ARCHITECTURE behavior OF mydff IS
BEGIN
   PROCESS (clk, rst)
   BEGIN
      IF (rst='1') THEN
         q <= '0';
      ELSIF (clk'EVENT AND clk='1') THEN
         q <= d;
      END IF;
   END PROCESS;
END behavior;
```

3.3 BCD to Seven Segment

This one was very simple to implement using VHDL, for each 4 bit input sequence, representing binary numbers from zero to nine, a seven bit output sequence is assigned that sets the number on the display to the appropriate number. The code is given below.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY bcd_7seg IS
   PORT(
      d 3, d2, d1, d0 : IN STD_LOGIC;
      a, b, c, d, e, f, g : OUT STD_LOGIC);
END bcd_7seg;
ARCHITECTURE seven_segment OF bcd_7seg IS
   SIGNAL input : STD_LOGIC_VECTOR (3 downto 0);
   SIGNAL output : STD_LOGIC_VECTOR (6 downto 0);
BEGIN
   input <= d3 & d2 & d1 & d0;
   WITH input SELECT
      output <= "0000001" WHEN "0000",--display 0
      "1001111" WHEN "0001",--display 1
      "0010010" WHEN "0010",--display 2
      "0000110" WHEN "0011",--display 3
      "1001100" WHEN "0100",--display 4
      "0100100" WHEN "0101",--display 5
      "1100000" WHEN "0110",--display 6
      "0001111" WHEN "0111",--display 7
      "0000000" WHEN "1000",--display 8
      "0001100" WHEN "1001",--display 9
      "1111111" WHEN others;
   a <= output(6);
   b <= output(5);
   c <= output(4);
   d <= output(3);
   e <= output(2);
   f <= output(1);
   g <= output(0);
End seven_segment;
```

3.4 50MHz to 4hz CLK

The idea behind the VHDL code for this realisation was quite simple. As the 50MHz clk changes from zero to one 12 500 000 times in one period of the desired 4Hz clk, we can just set the output signal to zero, then count 6 250 000 rising edges of the input signal and then set the output signal to one, after another 6 250 000 rising edges we set the signal again to zero, and repeat. If the reset signal is set to one, then we set the output to zero. The code is given below:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity clk4Hz is
   generic(ulaz : integer := 50000000);
   Port (
      clk_in : in STD_LOGIC;
      reset : in STD_LOGIC;
      clk_out: out STD_LOGIC
   );
end clk4Hz;
architecture ktoHz of clk4Hz is
   signal temporal: STD_LOGIC;
   signal counter : integer range 0 to ulaz/2/4-1 := 0;
begin
   frequency_divider: process (reset, clk_in) begin
      if (reset = '1') then
         temporal <= '0';
         counter <= 0;
      elsif rising_edge(clk_in) then
         if (counter = ulaz/2/4-1) then
            temporal <= NOT(temporal);
            counter <= 0;
         else
            counter <= counter + 1;
         end if;
      end if;
   end process;
   clk_out <= temporal;
end ktoHz;
```

3.5 Frequency divider

To divide the frequency of the input clock by two, the circuit just needs to invert its output every time the input clock changes from zero to one. The easiest way to do this is to connect the inverse output of a d-flipflop to its input, and connect the input clock to the clk pin of the d-flipflop. We made this circuit in a block diagram form, as can be seen in 3.1.

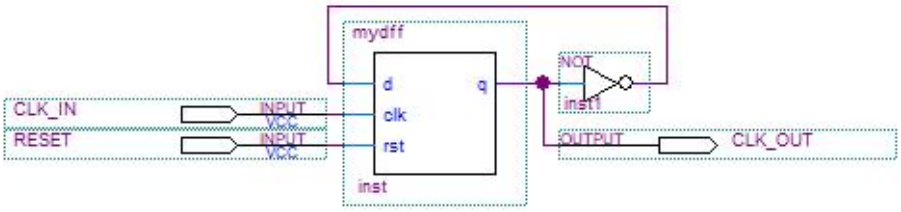

Figure 3.1: Frequency divider

3.6 Clock Selector Circuit

Depending on the state of the input pins the circuit needs to send to the output ether the input clock or the clock form one of the three subsequent frequency dividers. The easiest way to achieve this is to use four AND gate circuits with three inputs each. To each one of the AND gates a separate clock signal is connected. The output of an AND gate will be zero if any of the other inputs is zero, but if all the other inputs are ones, the output will be equal to the signal.

So if we want the output of the first AND gate to be equal to the signal with the lowest frequency when both inputs are zero, and zero when at least one of them is equal to one, we need to connect the inverse of both input signal to the AND gate. If the second AND gate should be equal to the the signal that is connected to it when one input signal is zero and the other one, the latter needs to be connected directly and the first one needs to be first inverted, similarity it is done for the other AND gates.

The output of all the AND gates are then connected to an OR gate, as only one of the AND gates can have an output equal to a signal, and the others will be zero, the output of the OR gate will be equal to the appro-

priate signal, and it will be the output of this circuit.

This circuit was made in a block diagram, as shown in figure 3.2.

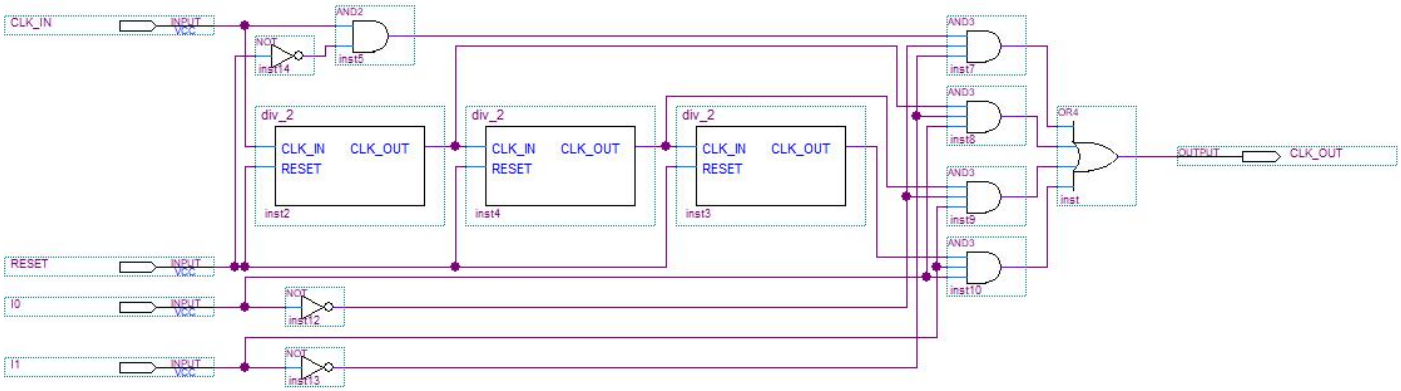

Figure 3.2: Clock Selector Circuit

3.7 Counter Circuit

If we connect the inverted output of a D-flipflop to its input and if its output was zero, when the first rising edge of a clock signal, that is connected to the clk pin of the D-flipflop, gets to the flipflop the output changes from zero to one, when the second rising edge arrives the output changes from one to zero. If we connect the inverted output of the D-flipflop to the clk of another D-flip flop, whose inverted output is connected to its input as well, when the second rising edge arrives at the first flip flop its output will change from zero to one, at the fourth rising edge its output will change from one to zero.

If we connect a total of four flipflops in this way, we can make a counter that counts to 15 in binary, where the bits are the outputs of each of the flipflops. As the circuit needs to count only up to one, when it arrives at ten it needs to reset back to all zeroes. This can be done if we connect the output of the second and fourth, and the inverted output of the first and third flipflop to an AND gate, and then we connect its output to the reset inputs of all the D-flipflops.

To be able to count the tens we need a second just as the previous one,

with the exception that to the first D-flopflop of the second circuit we connect the output of the AND gate of the first one. This way every time when the first circuit reaches ten, the second is incremented by one.

Again this circuit was made in the form of a block diagram which is given in figure 3.3

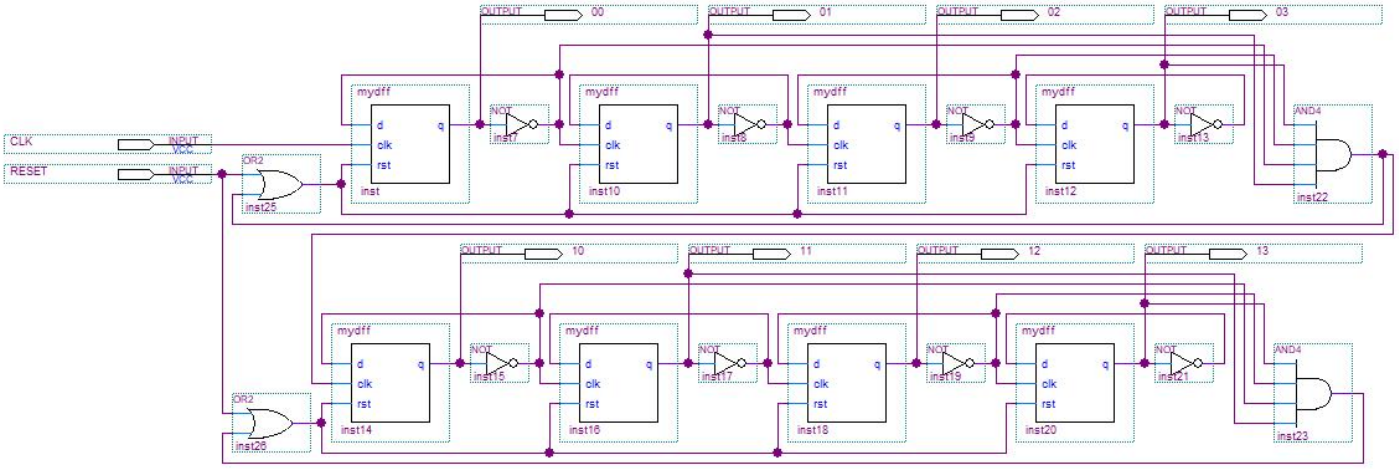

Figure 3.3: Counter Circuit

3.8 Complete circuit

Finally we just need to put together the pieces in a block diagram and we get our desired circuit, as given in figure 3.4.

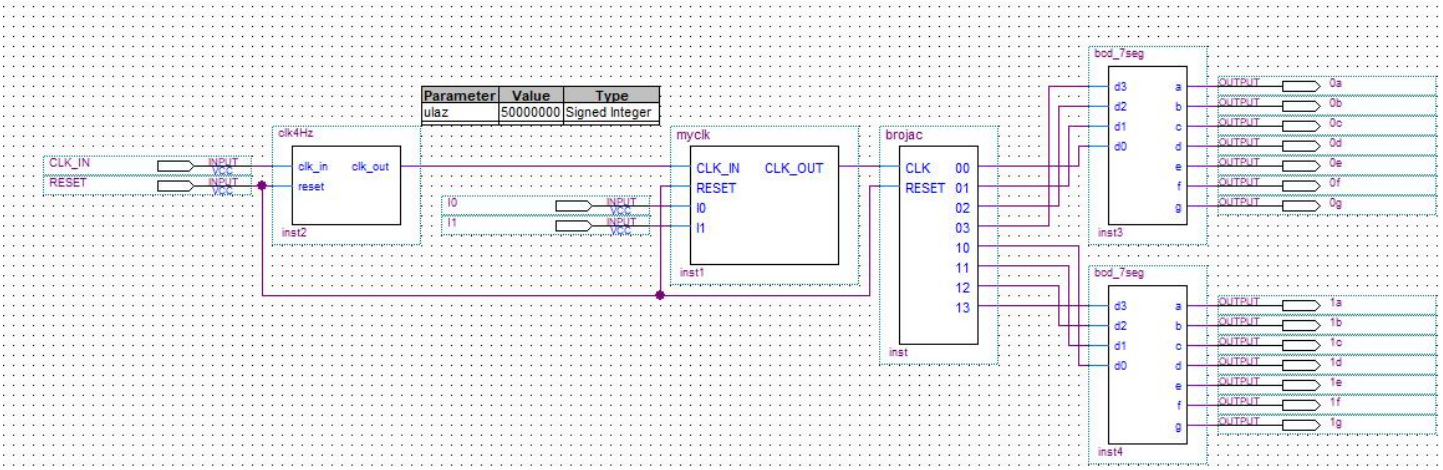

Figure 3.4: Complete circuit

4 Computer Simulations

For the computer simulations we used different combinations of input signals, and analysed the behaviour of the output signals. All simulations were successful and the results were as expected. For the simulation of the 50 MHz to 4 Hz circuit, instead of and 50 MHz input signal we used a 20 Hz 100 Hz and 1 kHz, as the simulation for the 50 MHz would have been too resource intensive. There was no reason to believe that the results would be different, as was confirmed after testing it on the board. As all the graphs from the simulations are pretty self explanatory, they will be given without further commentary.

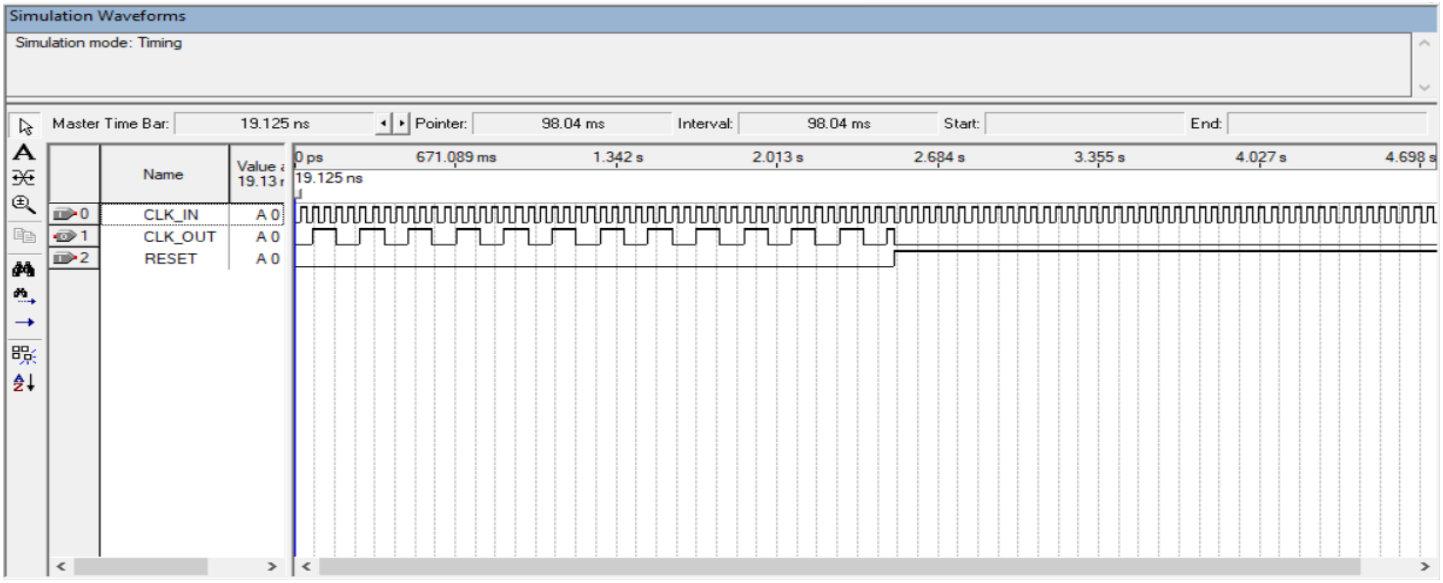

Figure 4.1: 20 Hz to 4 Hz

Figure 4.2: 100 Hz to 4 Hz

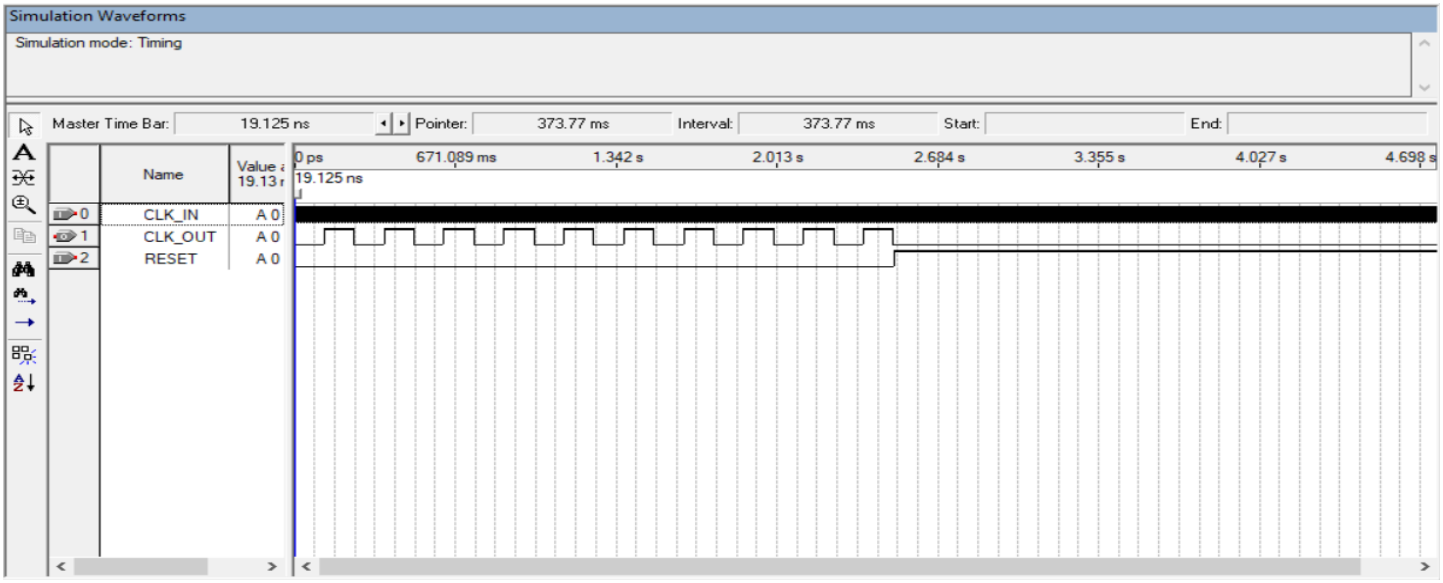

Figure 4.3: 1 kHz to 4 Hz

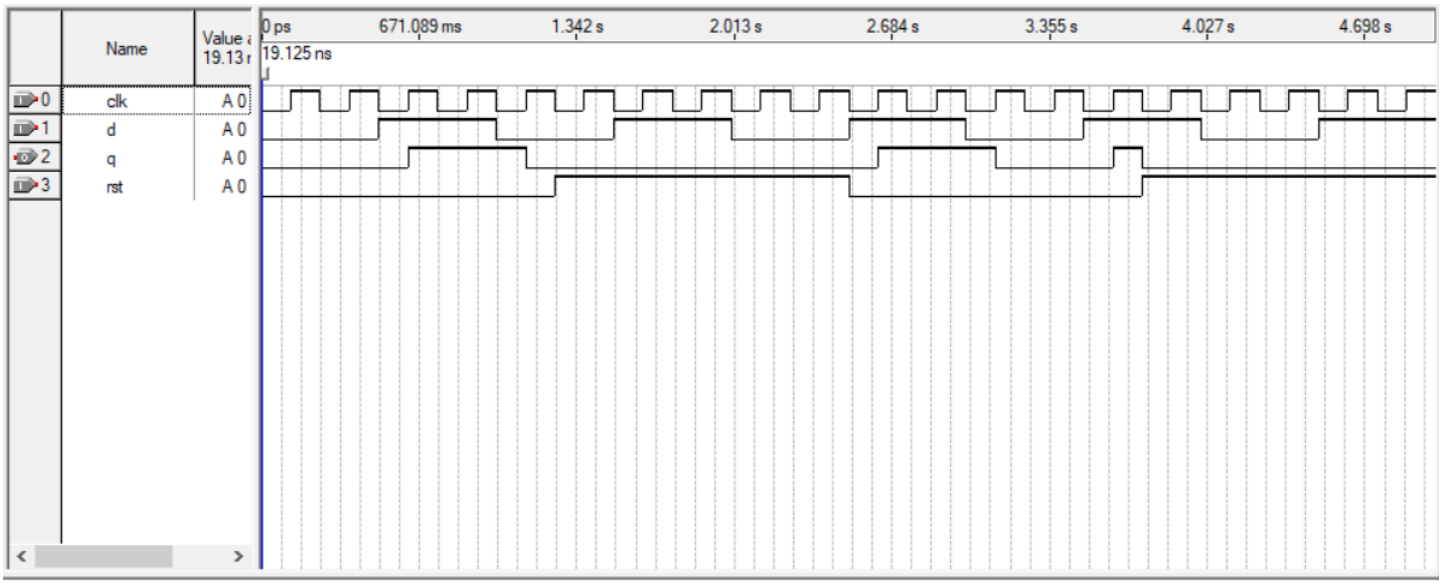

Figure 4.4: D-fliflop

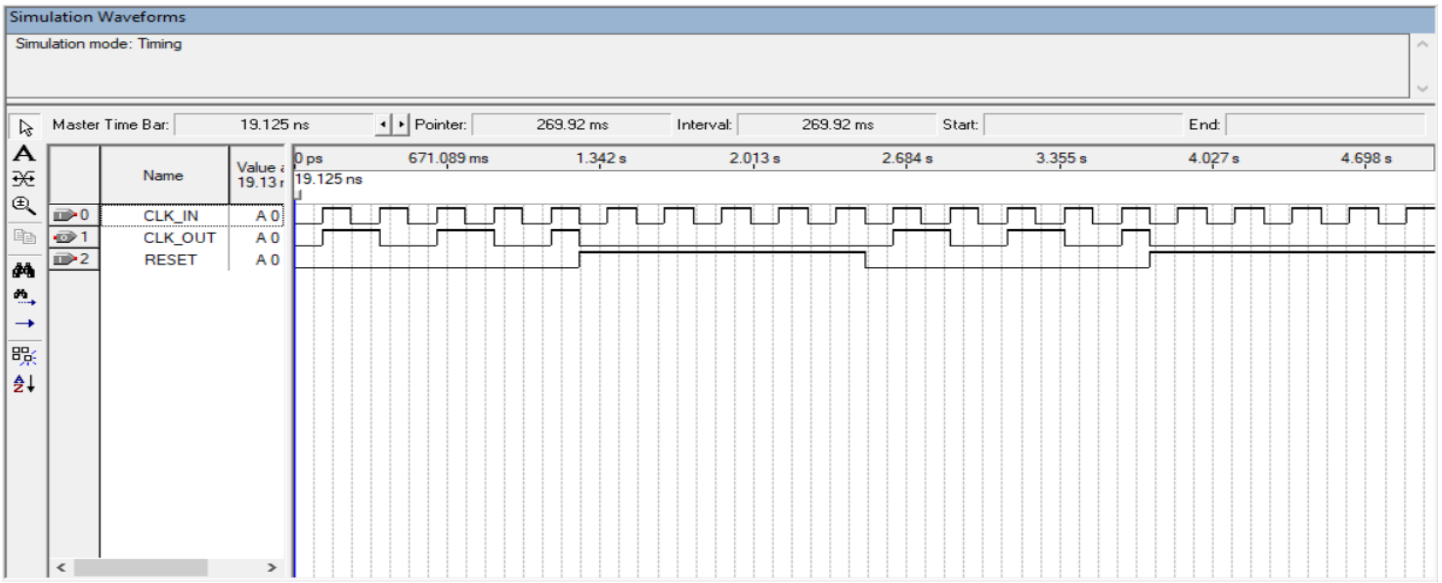

Figure 4.5: Frequency divider circuit

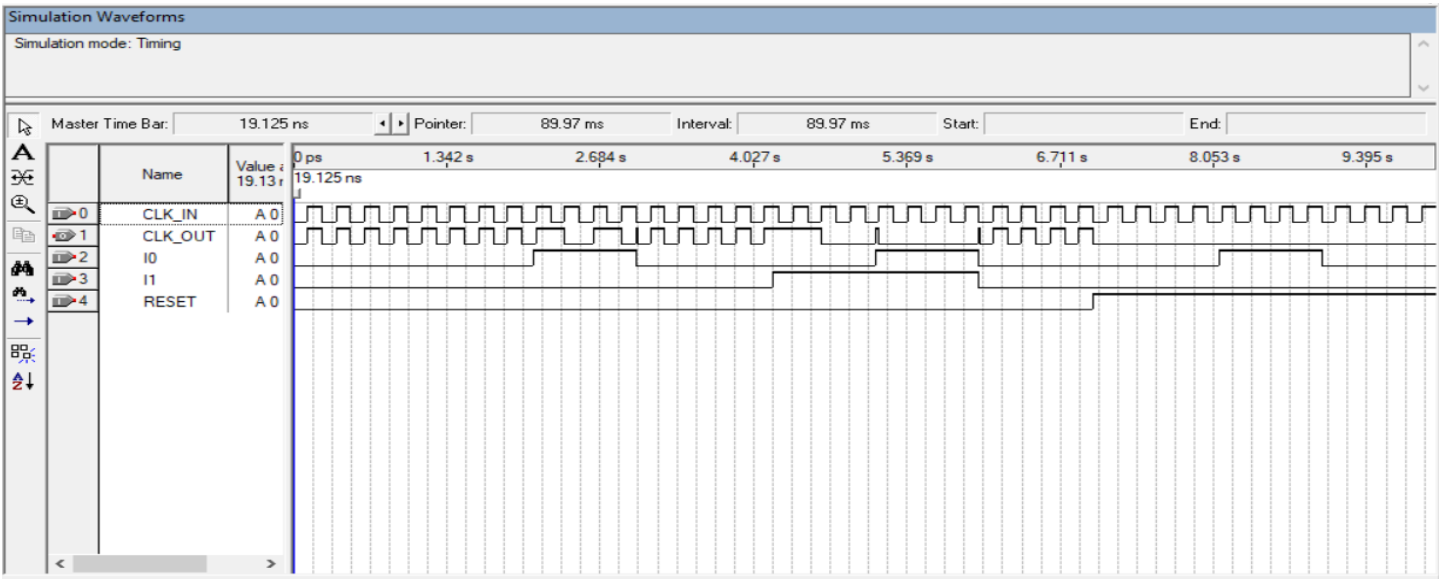

Figure 4.6: Clock selector circuit

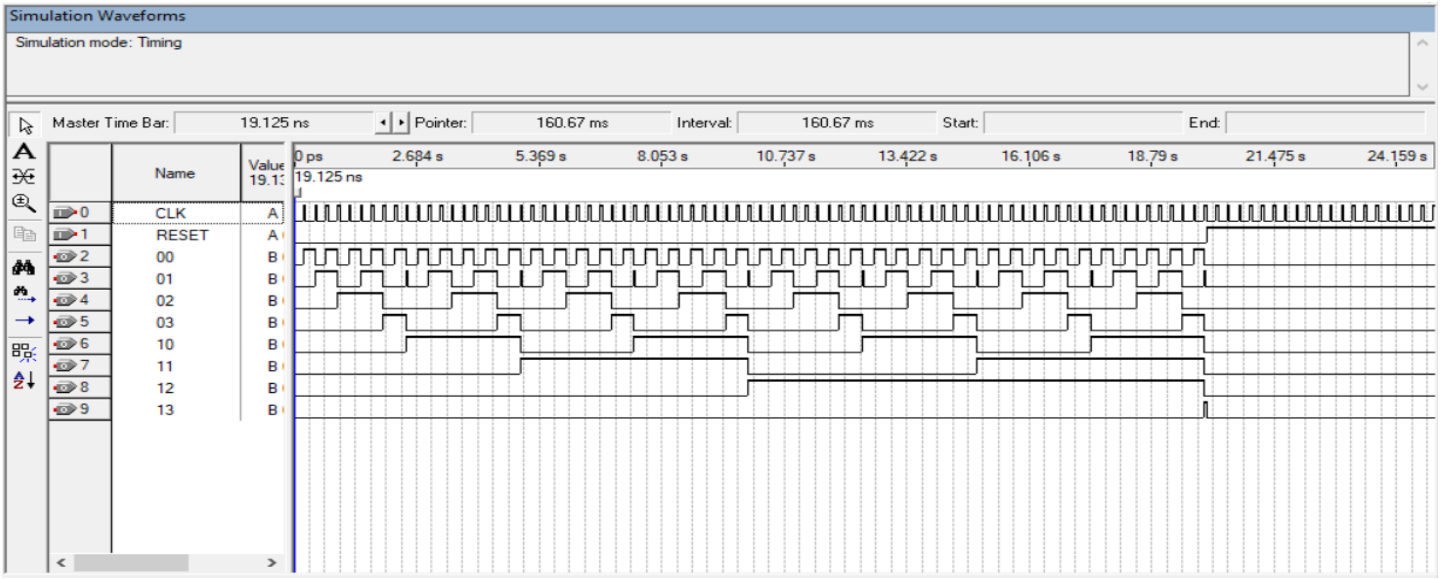

Figure 4.7: Counter circuit

5 Video Link

https://youtu.be/HJvbyuUAXIk

References

[1] Radovan D. Stojanović, *AUTOMATIZOVANO PROJEKTOVANJE DIGITALNIH SISTEMA (VHDL i FPGA)*

[2] *DE2-70 User manual version 1.08*

[3] *Vojislav Bego, Mjerenja u elektrotehnici, Tehnička knjiga, Zagreb, 1971*