



**Mittuniversitetet**  

---

MID SWEDEN UNIVERSITY

## **VHDL Design Range Sensor**

**Numbers:**

**Wen Qi**

**Ziqing Tang**

# Contents

Introduction .....	3
Module 1: Clock Generator .....	5
Module 2: Pulse Generator .....	7
Module 3: Echo Analyzer.....	9
Module 4: Constant Divider .....	12
Range Sensor Controller.....	14
Conclusion.....	16
Reference.....	17

# Introduction

VHDL (VHSIC-HDL) (Very High Speed Integrated Circuit Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems.

We used the Vivado Integrated Design Environment (IDE) for hardware system design on FPGA, selected the experiment board Basys 3(Figure 1) and SR04(Figure 2) to finish the design. First, we need to be familiar with Vivado and Basys 3. We learned it in experiment 1 through the reference document. Then, we can start the range sensor's design.

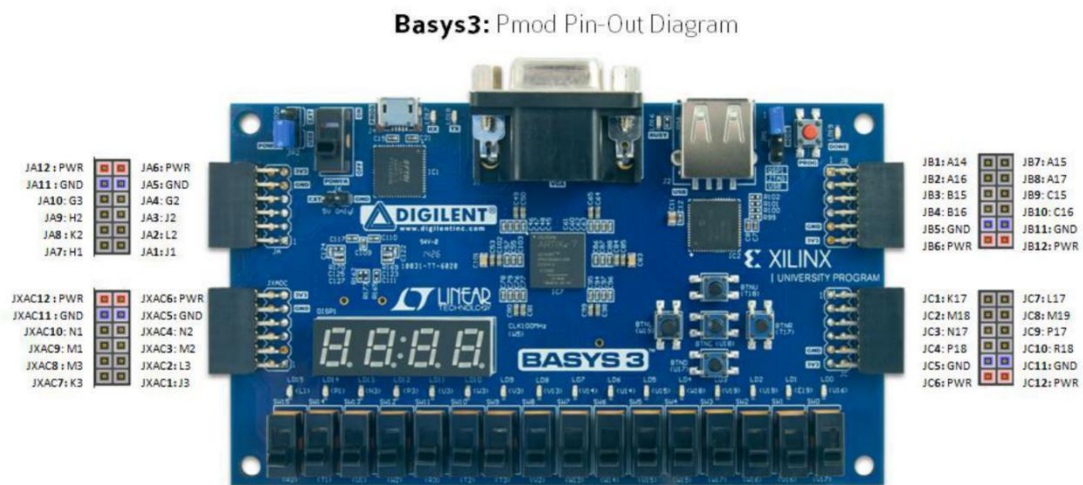


Figure 1: Basys 3 board and I/O

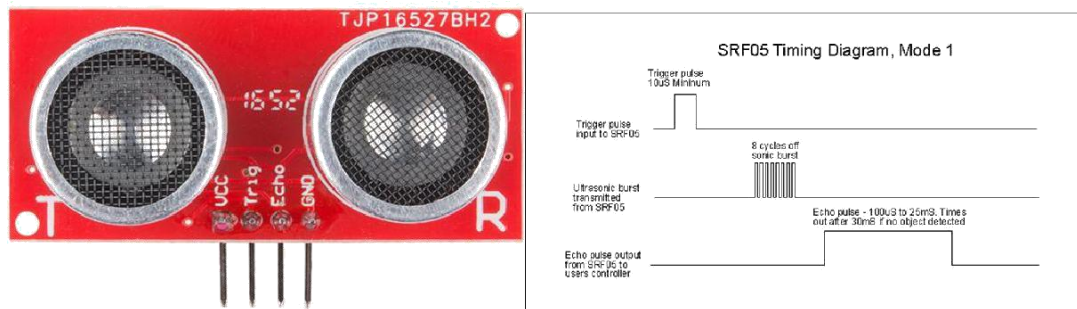


Figure 2: SR04 and Timing protocol of the Sensor Board

The design can be divided into 4 parts:

The first part is clock generator which reduce the FPGA board clock frequency from 100MHz to 1 MHz. It will be the clock for range sensor.

The second part is pluse generator. It will give trigger pulses of constant width at regular interval and in compliance with the timing protocol of the sensor board. In this part, we generate a 0.1 sec pluse period and 10µs pluse widths.

The third part is echo analyzer. This module's purpose is to measure the ToF for the transmitted ultrasound as an object reflects it back to the receiver. We use the 1MHz clock signal to count the pulse length.

The fourth part is constant divider. It will connect to the echo analyzer and convert the output time to distance.

4 modules will compose as the range sensor controller and their relationship show in Figure 3.

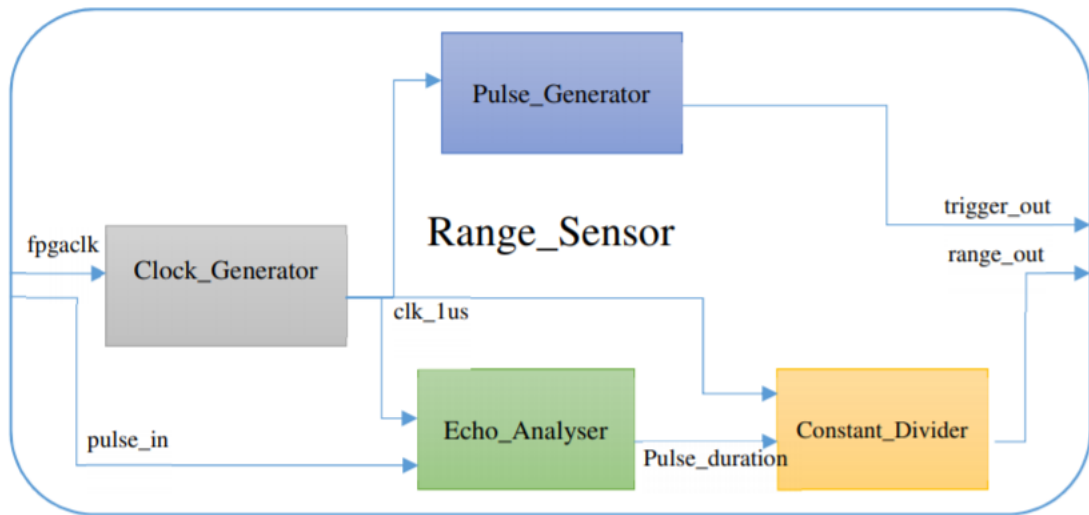
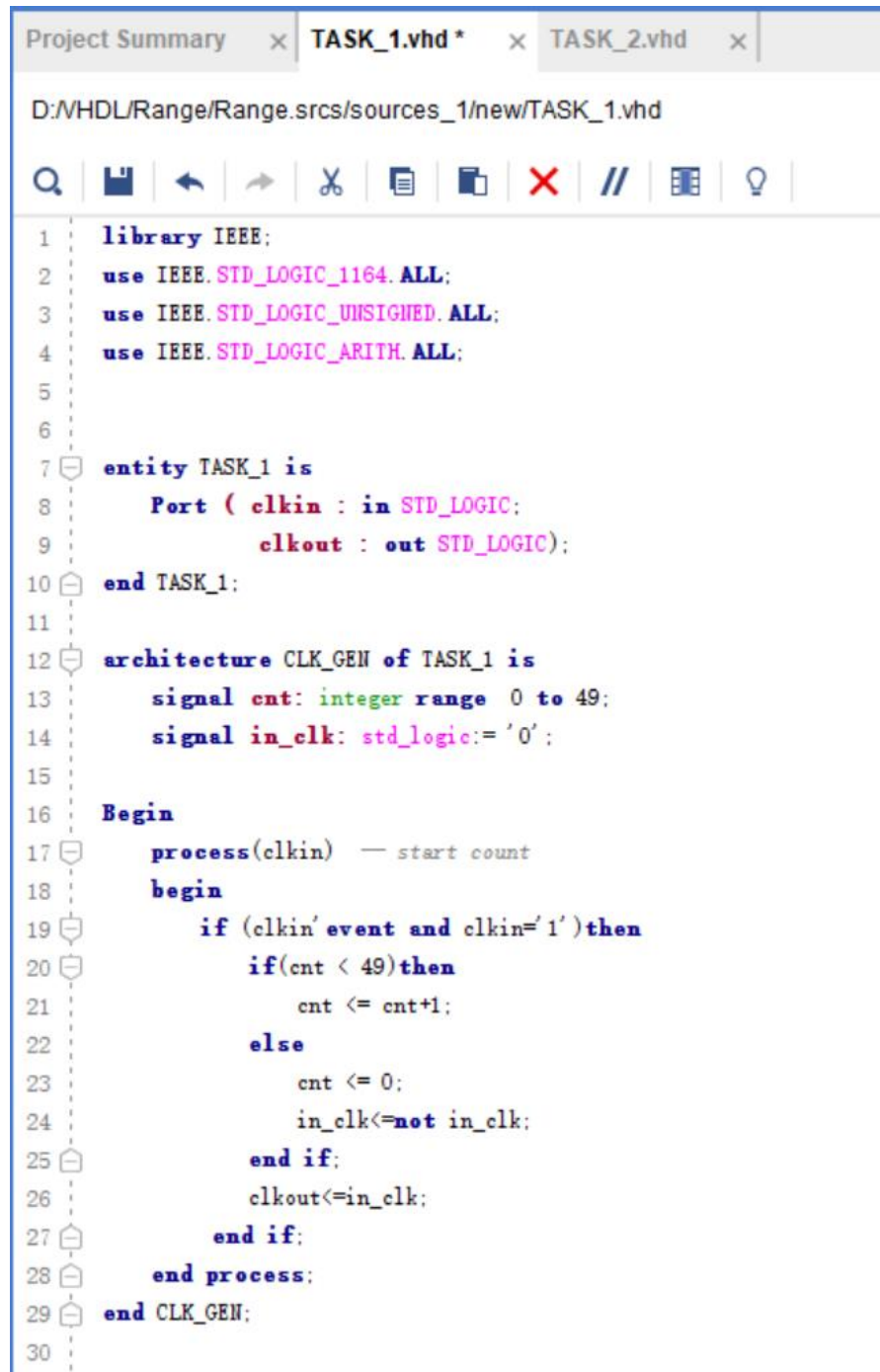


Figure 3: Range Sensor Controller

## Module 1: Clock Generator

The FPGA board clock is 100MHz, to get 1MHz clock we need a counter which count 50 pulse and then the output signal should reverse.

The code of the clock generator and simulation results are showed in Figure 4 and 5.



```
Project Summary x TASK_1.vhd * x TASK_2.vhd x
D:/VHDL/Range/Range.srcs/sources_1/new/TASK_1.vhd

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5
6
7  entity TASK_1 is
8      Port ( clkkin : in STD_LOGIC;
9            clkout  : out STD_LOGIC);
10 end TASK_1;
11
12 architecture CLK_GEN of TASK_1 is
13     signal cnt: integer range 0 to 49;
14     signal in_clk: std_logic:= '0';
15
16     Begin
17     process(clkkin) -- start count
18     begin
19         if (clkkin'event and clkkin='1') then
20             if(cnt < 49) then
21                 cnt <= cnt+1;
22             else
23                 cnt <= 0;
24                 in_clk<=not in_clk;
25             end if;
26             clkout<=in_clk;
27         end if;
28     end process;
29 end CLK_GEN;
30
```

Figure 4: Clock Generator Code

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  entity tb_clk is
6  end tb_clk;
7
8  architecture Behavioral of tb_clk is
9
10 component clk_gen is
11     Port ( clkkin : in STD_LOGIC;
12           clkout : out STD_LOGIC
13         );
14 end component;
15
16 signal clkkin : STD_LOGIC := '0';
17 signal clkout : STD_LOGIC;
18 signal cnt : STD_LOGIC;
19 begin
20 uut : clk_gen port map(clkkin=>clkkin, clkout=>clkout);
21
22 clk_generate: process (clkkin)
23     constant T_pw : time := 500 ns;
24     begin
25         if clkkin = '0' then
26             clkkin <= '1' after T_pw, '0' after 2*T_pw;
27         end if;
28     end process clk_generate;
29
30 end Behavioral;

```

Figure 4: Clock Generator Testbench Code

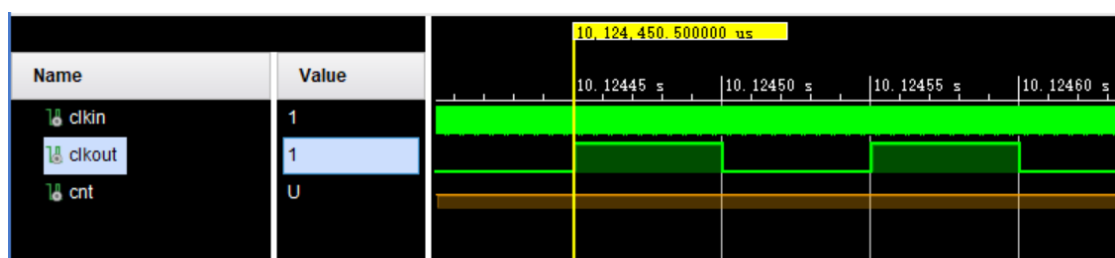


Figure 5: Clock Generator Simulation Results

## Module 2: Pulse Generator

In this module give trigger pulses of constant width at regular interval and in compliance with the timing protocol of the sensor board. The range sensor can be triggered by a 10us pulse and it can be generated with a cycle of 1us clock. We used a counter to implement the function.

The relationship between clock generator and pulse generator show in Figure 6.

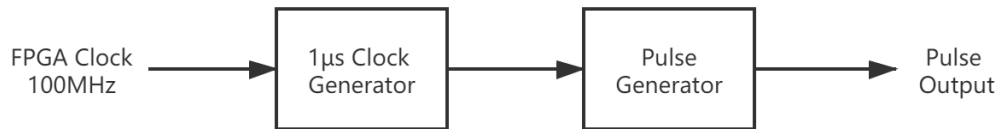


Figure 6: Clock generator and Pulse generator

The code of the pulse generator and simulation results are showed in Figure 7 and 8.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5
6  entity TASK_2 is
7      port ( clkout : in STD_LOGIC;
8            pulse_out : out STD_LOGIC);
9  end TASK_2;
10
11 architecture Pluse_Gen of TASK_2 is
12     constant cnt_bits : integer:=17;
13     constant period : integer:=100000;
14     constant pulse_length : integer:=10;
15     signal count : STD_LOGIC_VECTOR(cnt_bits-1 downto 0) :=(others=>'0'); --全部赋值为0
16 Begin
17 process(clkout)
18 begin
19     if (clkout'event and clkout='1') then
20         if count < CONV_STD_LOGIC_VECTOR(period-1, cnt_bits) then --
21             count <= count + '1';
22         else
23             count <= (others=>'0');
24         end if;
25         if count < CONV_STD_LOGIC_VECTOR(pulse_length, cnt_bits) then
26             pulse_out <= '1';
27         else
28             pulse_out <= '0';
29         end if;
30     end if;
31 end process;
32 end Pluse_Gen;
```

Figure 7: Pulse Generator Code

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  entity testbench is
6  end testbench;
7
8  architecture Behavioral of testbench is
9
10 component pulse_generator is
11     port ( clkout : in STD_LOGIC;
12           pulse_out : out STD_LOGIC);
13 end component;
14
15 signal clkout : STD_LOGIC := '0';
16 signal pulse_out : STD_LOGIC;
17
18 begin
19
20     uut: pulse_generator port map(clkout => clkout, pulse_out => pulse_out);
21
22     clock_generate: process (clkout)
23         constant T_pw : time := 500 ns;
24         begin
25             if clkout = '0' then
26                 clkout <= '1' after T_pw, '0' after 2*T_pw;
27             end if;
28         end process clock_generate;
29
30 end Behavioral;
31

```

Figure 7: Pulse Generator Testbench Code

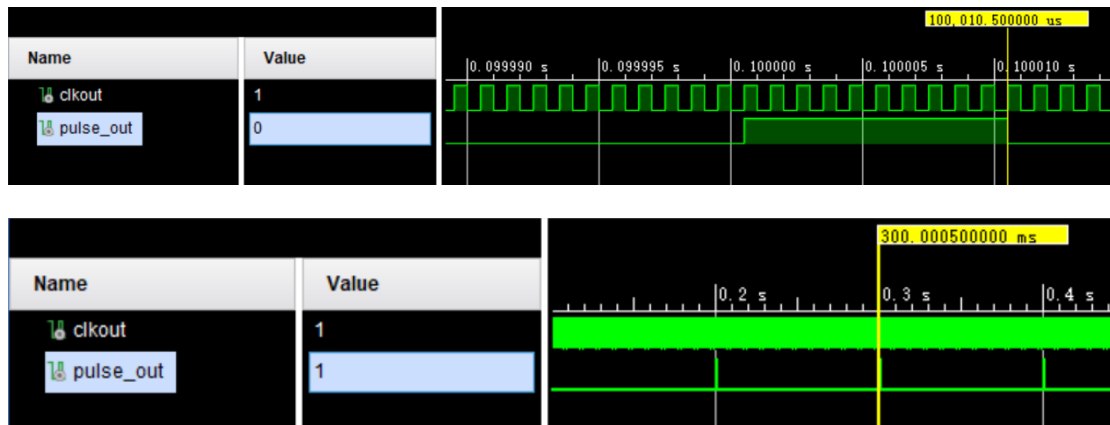


Figure 8: Pulse Generator Simulation Result

The 1us clock signal has been assigned to the J1 pin, as shown in next figure.

```

##Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {pulse}]
set_property IOSTANDARD LVCMOS33 [get_ports {pulse}]

```

Figure 9: Correspond with the J1



## Module 3: Echo Analyzer

The function of this module is to measure the ToF which equal to the ultrasonic sensor supplies pulse (echo) duration for the transmitted ultrasound as an object reflects it back to the receiver.

Therefore, we need two parts: one part measures the pulse length by using a counter connected to a 1MHz clock signal. Another output ToF is used as the number of clock cycles.

The following figure shows its structure.

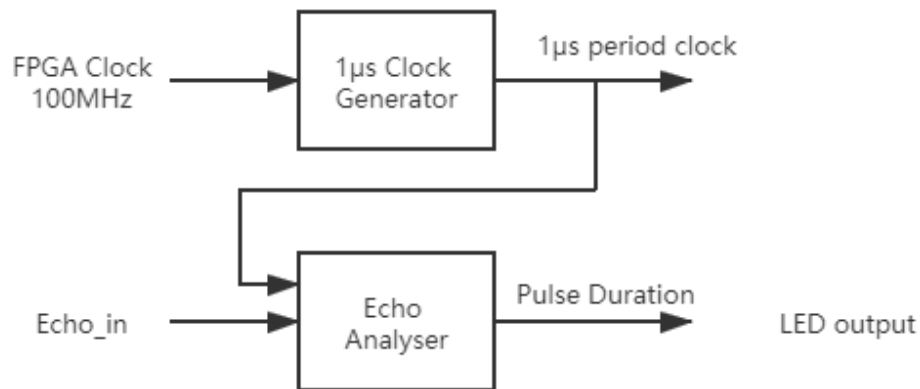


Figure 10: The structure of Echo Analyzer

The code of the echo analyzer will be showed next. The Echo\_out will connect to the constant divider.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6
7  entity TASK_3 is
8  port ( Echo_in : in std_logic;
9         clkout : in std_logic;
10        Echo_out : out std_logic_vector(15 downto 0)
11        );
12  end TASK_3;
13  architecture Echo_analyzer of TASK_3 is
14  begin
15      Process(clkout)
16      variable counter : integer := 0;
17      variable counter1 : integer := 0;
18      variable reset : integer := 0;
19      begin
20      if (clkout'event and clkout = '1') then
21          if echo_in = '1' then
```

```

22         counter := counter + 1;
23         counter1 := counter;
24     else
25         Echo_out <= std_logic_vector(to_unsigned(counter1, Echo_out'length));
26         counter := 0;
27     end if;
28 end if;
29 end process;
30 end Echo_analyzer;

```

Figure 11: Echo Analyzer Code

```

4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6  entity task3_test is
7  end task3_test;
8  architecture Behavioral of task3_test is
9  component echo_analyzer is
10     Port (Echo_in : in std_logic;
11           clkout : in std_logic;
12           Echo_out : out std_logic_vector(15 downto 0)
13           );
14  end component;
15  signal Echo_in : STD_logic := '0';
16  signal clkout : STD_logic := '0';
17  signal Echo_out : STD_logic_vector(15 downto 0);
18
19  begin
20     uut: echo_analyzer port map(Echo_in=> Echo_in, clkout=>clkout, Echo_out => Echo_out);
21
22     clock_generator: process (clkout)
23         constant T_pw : time :=500 ns;
24         begin
25             if clkout = '0' then
26                 clkout <= '1' after T_pw, '0' after 2*T_pw;
27             end if;
28         end process clock_generator;
29     echo_in_generator: process(Echo_in)
30         constant T: time :=240 us;
31         begin
32             if Echo_in = '0' then
33                 Echo_in <= '1' after T, '0' after 2*T;
34             end if;
35         end process echo_in_generator;
36 end Behavioral;

```

Figure 11: Echo Analyzer Testbench Code

The echoes of the signal generator have been assigned to the L2 pin, as shown in the next figure.

```
##Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {echo_in}]
set_property IOSTANDARD LVCMOS33 [get_ports {echo_in}]
```

Figure 12: Correspond with the L2

And the simulation result in Figure 13



Figure 13: Echo Analyzer Simulation Result

## Module 4: Constant Divider

In this part, the result measured by the echo analyzer in (us) need to be converted to the (cm). To do so, a division by a 58 factor is indeed. In particular, the performs synchronous integer division of an input number with a constant denominator 58. The resulting quote should be rounded to the closest, lower integer number.

From follow figure is the constant divider

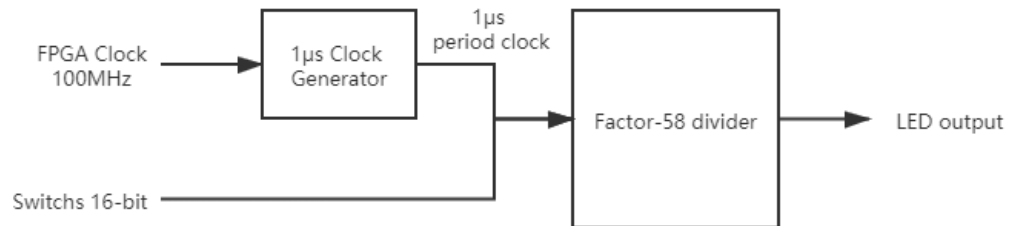


Figure 14: The structure of Constant Divider

The code of the constant divider and simulation result will be showed in Figure 15 and 16.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6
7  entity TASK_4 is
8  Port (      clkout : in STD_LOGIC;
9           Echo_out : in STD_LOGIC_VECTOR(15 downto 0);
10          Out_distance : out std_logic_vector(8 downto 0)
11        );
12 end TASK_4;
13 architecture factor_58 of TASK_4 is
14 signal input: integer:= 0;
15 signal out_distan: integer:= 0;
16 begin
17     factor_58_proc: process(clkout, Echo_out)
18     begin
19         if(clkout'event and clkout='1')then
20             input <= conv_integer(Echo_out);
21             out_distan<= input/58;
22             Out_distance <= std_logic_vector(to_unsigned(out_distan, Out_distance'length));
23
24         end if;
25     end process factor_58_proc;
26 end factor_58;
```

Figure 15: Constant Divider Code

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6  entity task4_test is
7  end task4_test;
8  architecture Behavioral of task4_test is
9  component factor_58 is
10   Port (      clkout : in STD_LOGIC;
11           Echo_out : in STD_LOGIC_VECTOR(15 downto 0);
12           Out_distance : out std_logic_vector(11 downto 0)
13         );
14 end component;
15 signal clkout : STD_LOGIC := '0';
16 signal Echo_out : STD_LOGIC_VECTOR(15 downto 0);
17 signal Out_distance : std_logic_vector(11 downto 0);
18 begin
19 uut: factor_58 port map(clkout=>clkout, Echo_out=>Echo_out, Out_distance=>Out_distance);
20 clock_factor_58 : process(clkout)
21   constant T_1 : time :=500 ns;
22   begin
23     clkout <= '1';
24   end process clock_factor_58;
25 echo_out_factor_58 : process(Echo_out)
26   begin
27     Echo_out <="0000000011110000";
28   end process echo_out_factor_58;
29 end Behavioral;

```

Figure 15: Constant Divider Testbench Code



Figure 16: Constant Divider Simulation Result

## Range Sensor Controller

Now we should to design a top module to control an SR04 range sensor. We need write a code to connect all the four previous tasks. Since the sensor has a maximum rate of 20Hz we will choose a rate of 10Hz. This means the trigger signal should have at least 100ms (100,000 $\mu$ s) between 2 consecutive pulses. We use LEDs to display measured distance as a binary number.

The code show in Figure 17.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE ieee.NUMERIC_STD.ALL;
4  USE ieee.STD_LOGIC_UNSIGNED.ALL;
5  USE ieee.STD_LOGIC_ARITH.ALL;
6  entity range_sensor is
7  Port (    clkin : in STD_LOGIC;
8           Echo_in : in STD_LOGIC;
9           Out_distance : out std_logic_vector(8 downto 0);
10          pulse_out : out std_logic);
11 end range_sensor;
12 architecture range_sensor_arc of range_sensor is
13   signal clkout: std_logic;
14   signal Echo_out: STD_LOGIC_VECTOR(15 downto 0);
15
16   -- definition of the previous tasks' modules as components of the top module
17   -- component 1: clock generator
18   component TASK_1
19   Port (    clkin : in STD_LOGIC;
20           clkout : out STD_LOGIC);
21   end component;
22
23   -- component 2: pulse generator
24   component TASK_2 is
25   port(    clkout: in std_logic;
26          pulse_out: out std_logic);
27   end component;
28
29   -- component 3: echo analyzer
30   component TASK_3 is
31   Port (    Echo_in : in STD_LOGIC;
32          clkout : in STD_LOGIC;
```

```

33   Echo_out : out STD_LOGIC_VECTOR(15 downto 0));
34   end component;
35
36   -- component 4: factor 58
37   component TASK_4 is
38   Port (      clkout : in STD_LOGIC;
39   Echo_out : in STD_LOGIC_VECTOR(15 downto 0);
40   Out_distance : out std_logic_vector(8 downto 0));
41   end component;
42
43   -- definition of the signals occurring in the implementations of the different modules
44   begin
45   unit1:TASK_1
46   port map(      clkkin => clkkin,
47   clkout => clkout);
48   unit2:TASK_2
49   port map(  clkout => clkout,
50   pulse_out => pulse_out);
51   unit3:TASK_3
52   port map(  Echo_in => Echo_in,
53   clkout => clkout,
54   Echo_out => Echo_out);
55   unit4:TASK_4
56   port map(  clkout => clkout,
57   Echo_out => Echo_out,
58   Out_distance => Out_distance);
59   end range_sensor_arc;
60

```

Figure 17: Controller Code



## Conclusion

After all parts above all, we run the program, generate bitstream and download to the board. We put a box 40cm away from the sensor. The LED display result 101000 equal to 40.

The test result show in Figure 18.

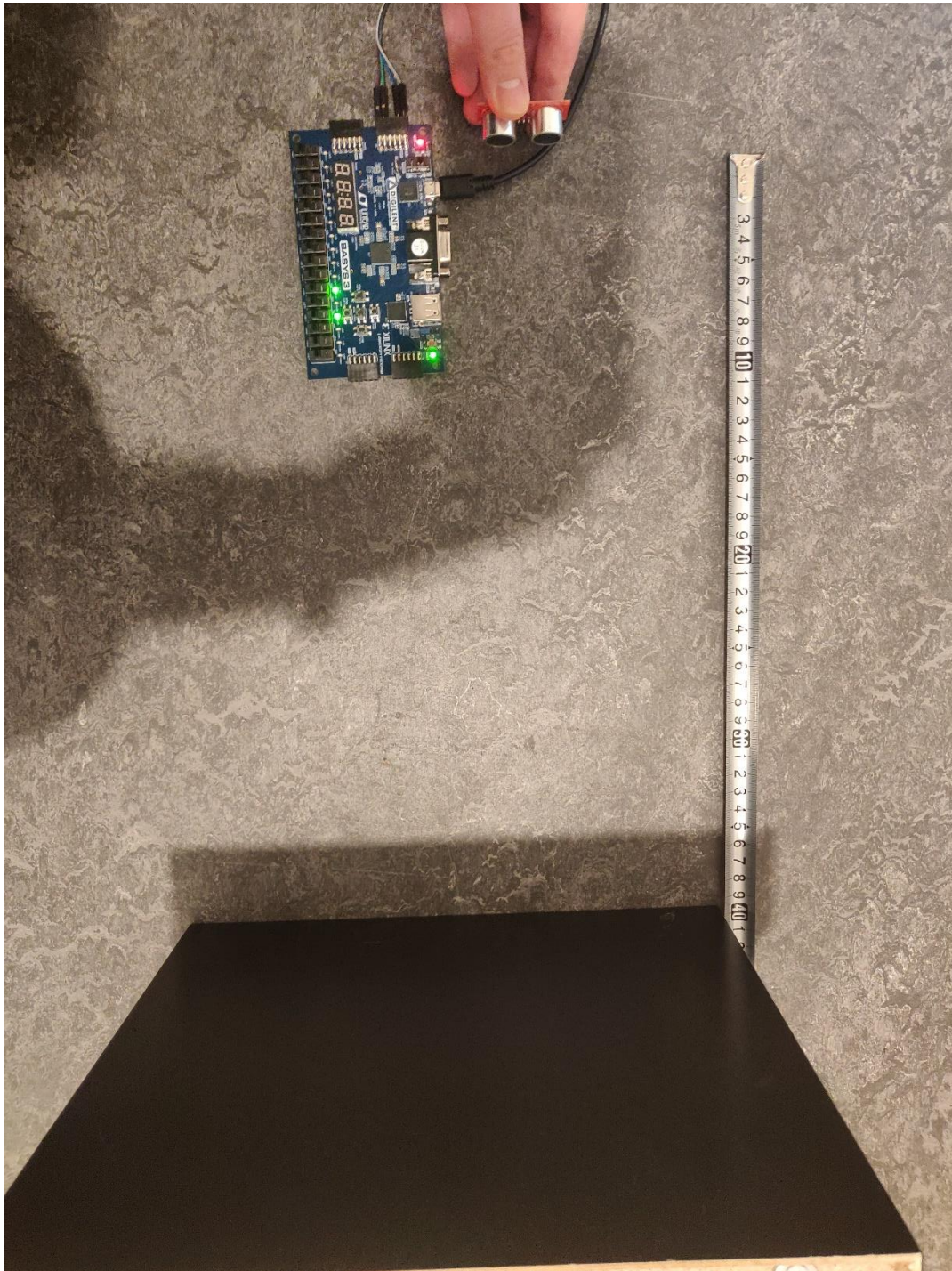


Figure 18: Test 40cm Result



## Reference

- [1] Range sensor SR04, <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>
- [2] Digilent Basys 3 board, <http://store.digilentinc.com/basys-3-artix-7-fpga-trainer-boardrecommended-for-introductory-users/>
- [3] General I/O demo for Basys 3 board, <https://reference.digilentinc.com/learn/programmable-logic/tutorials/basys-3-general-io/start>
- [4] Guide for getting started with Vivado, [https://reference.digilentinc.com/vivado/getting\\_started\\_tutorial/start](https://reference.digilentinc.com/vivado/getting_started_tutorial/start)