Voyager: Object Tracking Robot

EE102 Term Project Report

Saadet Büşra ÇAM | 22201857

13.05.2024

EE102-1 Digital Circuit Design Project

Youtube Link: https://youtu.be/hVvup3sD9Uc

Table of Contents

1. Introduction

Main purpose of the project was to understand how FPGA(field programmable gate arrays) works and design a sample project with it. FPGA's are widely used on real time systems because of their ability to handle concurrent tasks. In this project, I designed a robot which detects objects and tracks them via wheels.

2. Design Spesifications

Components: Car platform, wheels, 3 ultrasonic range sensors, 2 DC motor, voltage regulator, breadboard and jumper wires etc.

I designed an object following robot using abovementioned components on this project. The code of project is developed only using combinational logic design. The platform can move forward, backward, turn left, turn right etc. This object following system can be implemented on various fields such as industrial robots, medical assertive devices, cargo bots. This project can further be developed as inserting sequential circuit logic and providing the robot to make decisions on various occasions.

On this project, the system consists 3 subsystems:

1. FPGA basys3 board
2. Three ultrasonic range sensors
3. Two DC motors with voltage regulator

3. Methodology

3.1 **ULTRASONIC SENSOR:** The main component of this design was HC-SR04 ultrasonic range sensors. Each ultrasonic sensor has 4 pins: Vcc, Ground, Echo and Trigger. The sensor works basically as sending 10uS trigger output in approximately every 100ms and receiving the reflection of that sound wave from echo pin to process for distance calculation and object detection. Ultrasonic sensor has a detection range from 3mm to 4m. Via usage of FPGA boards, best measurements can be done from ultrasonic sensors with lowest delay.

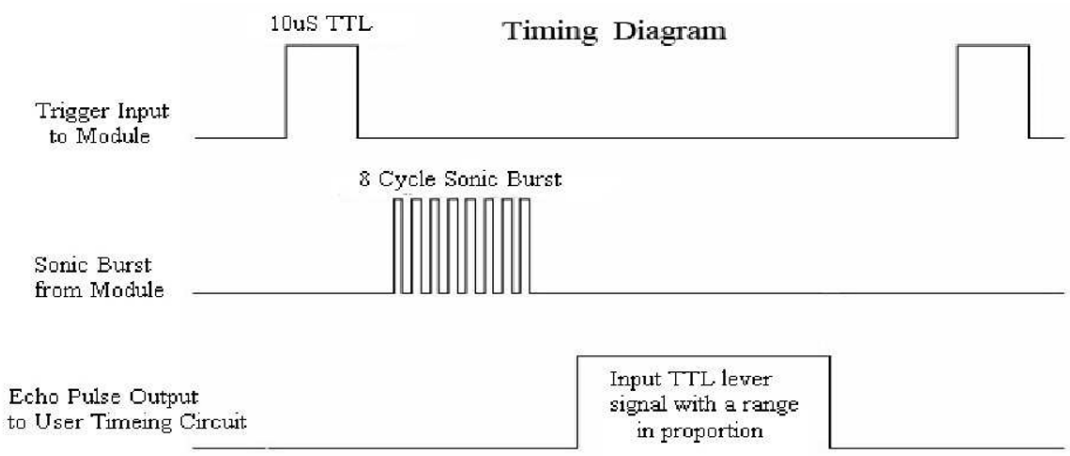


Figure 1: Ultrasonic Sensor Timing Diagram

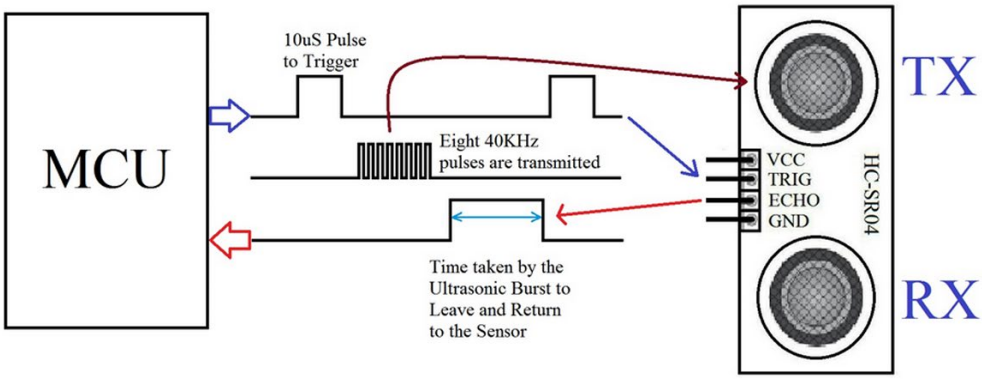


Figure 2: Ultrasonic Sensor Interface

3.2 MOTORS: In order to move the platform on desired directions, 2 DC motors used. L298N voltage regulator used to operate these motors. Each motor has 2 pins. PWM signal is sent to one of these pins while other pin is in low state. PWM signal generation is done with VHDL programmed FPGA boards. A 23-bit counter used which is working at 50MHz. PWM module generates duty cycle and period.

Characteristics:

Compact Size Motor

Specifications:

Dimensions	:	Ø 15.5 X 12.0 X 18.6 mm
Shaft Diameter	:	Ø 1.505 mm
Input Voltage	:	5.0 V DC
No Load Speed	:	12623 rpm
No Load Current	:	0.06 A
Stall Torque	:	2.09 mNm
Stall Current	:	0.64 A
Maximum Output Power	:	0.69 W
Maximum Efficiency	:	50 %
Speed at Maximum Efficiency	:	9602 rpm
Life (typical)	:	17 hr
Weight	:	10 g
Operation Temperature	:	-20 to 70 °C
Storage Temperature	:	-40 to 85 °C
Electrical Connection	:	Terminal

Figure 3: DC motor characteristics

3.3 SYSTEM OVERVIEW:

The robot moves as following the rules based on Table-1 robot movement table. Speed and turning direction are adjusted accordingly to movement. For instance, if the robot turns left, its right wheel moves fast and left wheel moves slower. On the other hand if robot keeps going forward or backward, then its wheels turn in equal speed. 8 different stages consist all different input combinations.

Obstacle	Movement
000	Backward
001	Turn Right
010	Forward
011	Turn Right
100	Turn Left
101	Backward
110	Turn Left
111	Forward

Table-1: Robot Movement

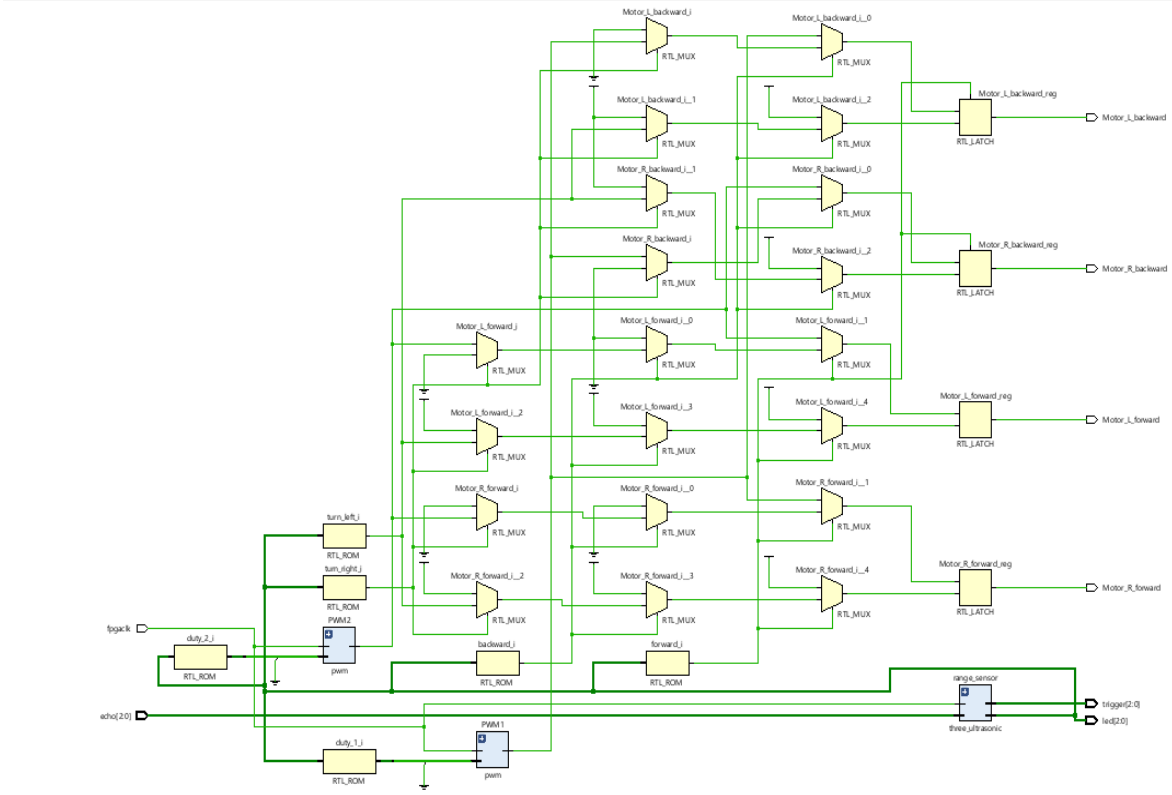


Figure-4: General RTL Schematic

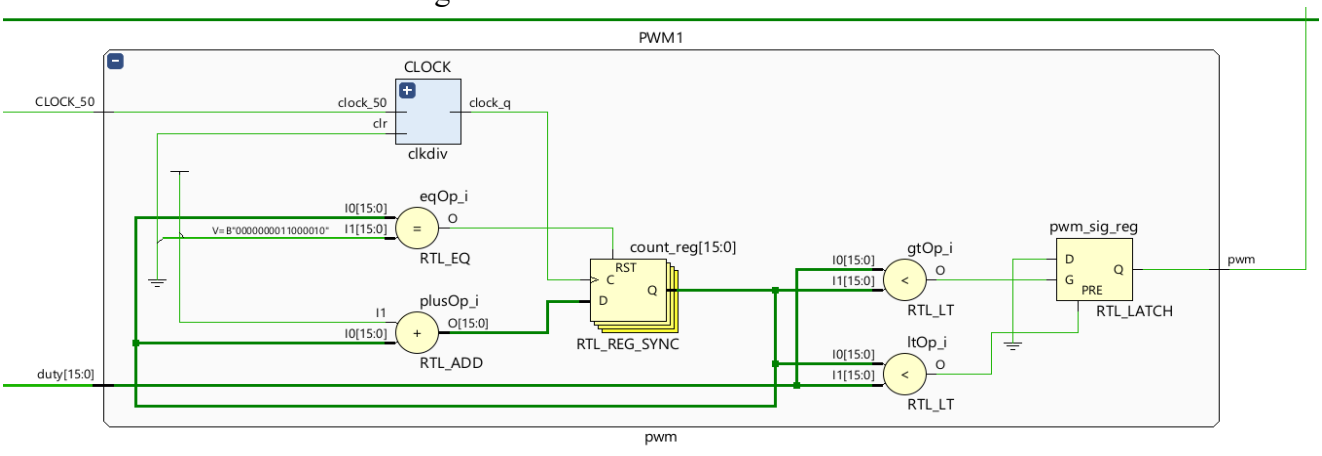


Figure-5: RTL schematic of pwm

4. Results

Since my robot works as intented, the project had been succeed. The wheels turn accordingly to the ultrasonic sensor echo pin input analyzed commands from fpga. However, due to sensitivity of ultrasonic sensors and disconnectivity on some jumper cables, in some instances wheels turn backward or stop working. This errors can be minimized via using appropriately working ultrasonic sensors or connecting wires carefully. Additionally, the use of 3 different ultrasonic sensors may caused a problem on detection of objects. What this means is that the trigger send from one sensor can be received from other sensor. Such instance may increase errors on the system.

5. Conclusion

In conclusion, on this project, I gained knowledge about project design with FPGA using ultrasonic sensors and dc motors. FPGA(field programmable gate arrays) allowed handling concurrent tasks so the use of them is advantageous. This robot is a nice sample project on understanding basic principles of basys3 board, ultrasonic sensors and dc motors. It operated successfully on simple tryings. Additionally, this project can be developed using sequential circuit logic. This robot does not make and intelligent decisions since only combinational circuit logic used. Also, the addition of more sensors to the robot may make increase its sensitivity to other objects.

6. References

BBC Bitesize. (n.d.). Revision - BBC Bitesize. Retrieved from https://www.bbc.co.uk/bitesize/guides/z6qqmsg/revision/6

Digilent. (n.d.). Basys 3 Reference Manual. Retrieved from https://digilent.com/reference/programmable-logic/basys-3/reference-manual

FPGA obstacle avoidance robot using VHDL. (n.d.). https://www.researchgate.net/publication/342014163_FPGA_Obstacle_Avoidance_Robot_Using_VHDL

HC-SR04 Ultrasonic Sensor Module User Guide

How to Implement VHDL design for a Range sensor on an FPGA. (youtube.com)

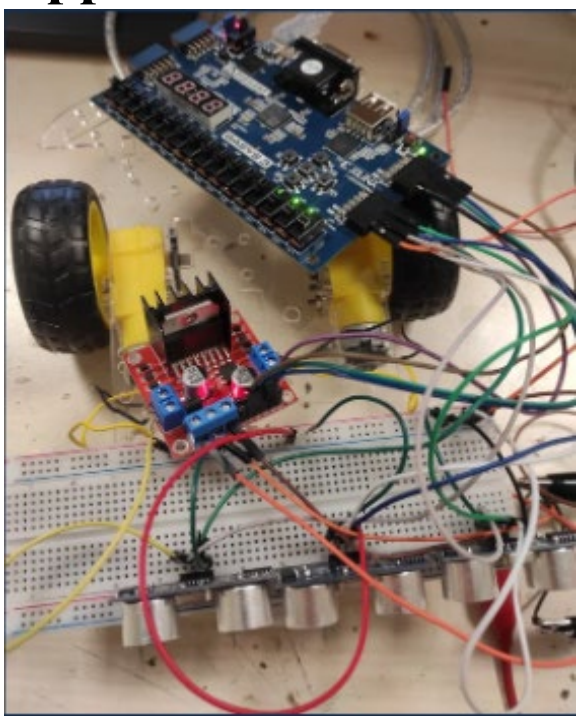
How to Implement VHDL design for Seven Segment Displays on an FPGA. (youtube.com)

https://github.com/Danishazmi29/FPGA-implementation-of-obstacle-detection-system-using-Ultrasonic-Sensor-using-VHDL..git

https://github.com/metepv/Ultrasonic-Range-Sensor.git

Instructables. (n.d.). How to Control a Stepper Motor With an FPGA. Retrieved from https://www.instructables.com/How-to-Control-a-Stepper-Motor-With-an-FPGA/

7. Appendix



MAIN.VHD

```
library ieee;
use ieee.std_logic_1164.all;
entity robot is
port( clock: in std_logic;
echo: in std_logic_vector(2 downto 0);
trigger: out std_logic_vector(2 downto 0);
led : out std_logic_vector(2 downto 0);
Motor_L_forward,
Motor_R_forward,
Motor_L_backward:out std_logic;
Motor_R_backward: out std_logic);
end entity;
architecture behaviour of robot is
```

```
component pwm is --generic(N : integer:=7);
port(
CLOCK_50: in std_logic;
duty: in std_logic_vector(15 downto 0);
pwm : out std_logic);
end component;
component three_ultrasonic is
 port(
   clock: in std_logic;
   pulse:in std_logic_vector(2 downto 0);
   triggerOut:out std_logic_vector(2 downto 0);
   ultrasonic_out:out std_logic_vector(2 downto 0));
end component;

signal ultrasonic: std_logic_vector(2 downto 0);
signal pwm_1,pwm_2 : std_logic;
signal forward,backward,turn_left,turn_right:std_logic;
signal duty_1,duty_2:std_logic_vector(15 downto 0);



begin
PWM1: pwm port map(clock,duty_1,pwm_1); -- generate pwm for the motors
PWM2: pwm port map(clock,duty_2,pwm_2); -- motion control selection of the motor
process(forward,backward,turn_left,turn_right)
begin
 if(forward = '1') then
  motor_R_forward <= pwm_1;
  motor_L_forward <= pwm_2;
  motor_L_backward <= '0';
```

```
 motor_R_backward <= '0';
elsif(backward = '1') then
 motor_R_backward <= pwm_2;
 motor_L_backward <= pwm_1;
 motor_R_forward <= '0';
 motor_L_forward <= '0';
elsif(turn_right = '1') then
 motor_L_forward <= pwm_2;
 motor_R_backward <= pwm_1;
 motor_R_forward <= '0';
 motor_L_backward <= '0';
elsif(turn_left = '1') then
 motor_R_forward <= pwm_2;
 motor_L_backward <= pwm_1;
 motor_R_backward <= '0';
 motor_L_forward <= '0';
end if;
end process;

range_sensor:three_ultrasonic port
map(clock echo,trigger,ultrasonic);

process(ultrasonic)
begin
 case (ultrasonic) is
when "000" => forward <= '0';backward <=
'1';turn_right<='0';turn_left<='0'; duty_1 <= X"00BE"; duty_2 <=
X"00C3";
when "001" => forward <= '0';backward <=
```

```
'0';turn_right<='1';turn_left<='0'; duty_1 <= X"00C3"; duty_2 <=
X"00C3";
when "010" => backward <= '0';forward <=
'1';turn_right<='0';turn_left<='0'; duty_1 <= X"0041"; duty_2 <=
X"00C3";
when "011" => turn_left <= '0';backward <= '0';forward <=
'1';turn_right<='0'; duty_1 <= X"0041"; duty_2 <= X"00C3"; --  left motor low right motor high
when "100" => forward <= '0';backward <=
'0';turn_right<='0';turn_left<='1';duty_1 <= X"00C3"; duty_2 <=
X"00C3"; --
when "101" => forward <= '0';backward <=
'1';turn_right<='0';turn_left<='0';duty_1 <= X"00BE"; duty_2 <=
X"00C3";
when "110" => turn_right <= '0';turn_left <= '1';backward
<= '0';forward <= '0'; duty_1 <= X"00C3"; duty_2 <= X"0041"; --left motor high right motor low
when "111" => backward <= '0';forward <=
'1';turn_right<='0';turn_left<='0';duty_1 <= X"00C3"; duty_2 <=
X"0041";
end case;
end process;
led(2)<= ultrasonic(2);
led(1)<= ultrasonic(1);
led(0)<= ultrasonic(0);
end architecture;
```

pwm.vhd

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

```
entity pwm is
port(
  CLOCK_50: in std_logic;
  duty: in std_logic_vector(15 downto 0);
  pwm : out std_logic);
end pwm;
architecture behaviour of pwm is

component clkdiv is
 port(
   clock_50:in std_logic;
   clr: in std_logic;
   clock_q:out std_logic);
end component;



signal count: std_logic_vector(15 downto 0);
signal clk,pwm_sig: std_logic; --signal duty:std_logic_vector(15 downto 0);
signal period :std_logic_vector(15 downto 0);
signal clr:std_logic;

begin --clk <=clock_50;

 --duty <= X"0001";
 period <= X"00C3"; --0.00256 ms period for clock - 6th bit of counter

 clr <= '0';

 process(clk,clr) --counter
```

```
begin
 if (clr='1') then
  count<=X"0000";
 elsif (clk'event and clk='1') then
  if (count=period-1) then
   count<=X"0000";
  else
   count<= count+1;
  end if;
 end if;
end process;

process(count) --signal for pwm, if duty and counter(as clock) gives right input to pwm, motors move
begin
 if (count < duty) then
  pwm_sig <='1';
 elsif(count > duty) then
  pwm_sig <='0';
 end if;
end process;

pwm <= pwm_sig;

CLOCK:clkdiv port map(clock_50,'0',clk); -- divide 50Mhz clock to clk_q6
end architecture;
```

thrultrasonic.vhd

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

```
entity three_ultrasonic is
 port(
clock: in std_logic;
   pulse:in std_logic_vector(2 downto 0);
   triggerOut:out std_logic_vector(2 downto 0);
   ultrasonic_out:out std_logic_vector(2 downto 0));
end entity;

architecture behaviour of three_ultrasonic is

component ultrasonic is
 port(
clock: in std_logic;
   pulse: in std_logic; -- echo
   triggerOut:out std_logic; -- trigger out
   obstacle:out std_logic);
end component;

begin

ultrasonic_Left:ultrasonic port
map(clock,pulse(0),triggerOut(0),ultrasonic_out(0));
ultrasonic_Middle:ultrasonic port
map(clock,pulse(1),triggerOut(1),ultrasonic_out(1));
ultrasonic_Right:ultrasonic port
map(clock,pulse(2),triggerOut(2),ultrasonic_out(2));

end architecture;
```

ultrasonic.vhd

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

entity ultrasonic is
 port(
clock: in std_logic;
   pulse: in std_logic; -- echo
   triggerOut:out std_logic; -- trigger out
   obstacle:out std_logic);
end entity;

architecture behaviour of ultrasonic is

component counter is
 generic(n : positive :=10);
 port( clk: in std_logic;
   enable : in std_logic;
   reset : in std_logic; -- active low
   counter_output: out std_logic_vector(n-1 downto 0));
end component;

component trigger_generator is
 port( clk: in std_logic;
   trigg : out std_logic);
end component;
```

```
 --signal triggerOut: std_logic; --signal distanceOut:std_logic(21 downto 0);
signal pulse_width: std_logic_vector(21 downto 0);
signal trigg:std_logic;

begin
counter_echo_pulse :
 counter generic map(22) port
map(clock,pulse,not(trigg),pulse_width);
trigger_generation :
 trigger_generator port map(clock,trigg);
obstacle_detection: process(pulse_width)
 begin
  if(pulse_width < 55000) then
   obstacle <= '1';
  else
   obstacle <= '0';
  end if;
 end process;

 triggerOut <= trigg;
 end architecture;
```

counter.vhd

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

entity counter is
 generic(n : positive :=10);
```

```
 port( clk: in std_logic;
   enable : in std_logic;
   reset : in std_logic; -- active low
   counter_output: out std_logic_vector(n-1 downto 0));
end entity;

architecture behavioural of counter is

signal count : std_logic_vector(n-1 downto 0);

begin

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

 counter_output <= count;

end architecture;
```

segmentdecoder

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

entity segmentdecoder is

Port ( digit : in STD_LOGIC_VECTOR (3 downto 0);
       segconfig : out STD_LOGIC_VECTOR (6 downto 0));
end segmentdecoder;

architecture segmentdecoder_arch of segmentdecoder is
begin
   process(digit)
--variable tmp: std_logic_vector(6 downto 0);
   begin
   case digit is
      when "0000" => segconfig <= "0000001";
      when "0001" => segconfig <= "1001111";
      when "0010" => segconfig <= "0010010";
      when "0011" => segconfig <= "0000110";
      when "0100" => segconfig <= "1001100";
      when "0101" => segconfig <= "0100100";
      when "0110" => segconfig <= "0100000";
      when "0111" => segconfig <= "0001111";
      when "1000" => segconfig <= "0000000";
      when others => segconfig <= "0000100";
--       when "1010" => segconfig <= "0001000";
--       when "1011" => segconfig <= "1100000";
--       when "1100" => segconfig <= "0110001";
--       when "1101" => segconfig <= "1000010";
```

```
--       when "1110" => segconfig <= "0110000";
--       when others => segconfig <= "0111000";
   end case;
   end process;
end segmentdecoder_arch;
```

segmentdisplay.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity main_module is
Port ( Gr1 : in STD_LOGIC_VECTOR (3 downto 0);
Gr2 : in STD_LOGIC_VECTOR (3 downto 0);
Gr3 : in STD_LOGIC_VECTOR (3 downto 0);
Gr4 : in STD_LOGIC_VECTOR (3 downto 0);
Clk_In : in STD_LOGIC;
Seven_segments : out STD_LOGIC_VECTOR (6 downto 0);
anod_sel : out STD_LOGIC_VECTOR (3 downto 0));
end main_module;
architecture Behavioral of main_module is
COMPONENT Mux
PORT ( grp1 : in STD_LOGIC_VECTOR (3 downto 0);
grp2 : in STD_LOGIC_VECTOR (3 downto 0);
grp3 : in STD_LOGIC_VECTOR (3 downto 0);
grp4 : in STD_LOGIC_VECTOR (3 downto 0);
sev_sel : in STD_LOGIC_VECTOR (1 downto 0);
mux_out : out STD_LOGIC_VECTOR (3 downto 0));
```

```
END COMPONENT;
COMPONENT Sev_Seg
PORT ( out_of_mux : in STD_LOGIC_VECTOR (3 downto 0);
seven_seg : out STD_LOGIC_VECTOR (6 downto 0));

END COMPONENT;
COMPONENT clock_divider
PORT ( clk_in : in STD_LOGIC;
Reset : in STD_LOGIC;
clk_out : out STD_LOGIC_VECTOR (1 downto 0));
END COMPONENT;
COMPONENT invert_decoder
PORT ( input : in STD_LOGIC_VECTOR (1 downto 0);
output : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;
SIGNAL temp_dig : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL temp_reset : STD_LOGIC;
SIGNAL temp_clk : STD_LOGIC_VECTOR (1 DOWNTO 0);
begin
OPP1 : Mux
PORT MAP (grp1 (3 DOWNTO 0) => Gr1 (3 DOWNTO 0),
grp2 (3 DOWNTO 0) => Gr2 (3 DOWNTO 0),
grp3 (3 DOWNTO 0) => Gr3 (3 DOWNTO 0),
grp4 (3 DOWNTO 0) => Gr4 (3 DOWNTO 0),
sev_sel (1 DOWNTO 0) => temp_clk (1 DOWNTO 0),
mux_out (3 DOWNTO 0) => temp_dig (3 DOWNTO 0));
OPP2 : Sev_Seg

PORT MAP (out_of_mux (3 DOWNTO 0) => temp_dig (3 DOWNTO 0),
```

```
seven_seg (6 DOWNTO 0) => Seven_segments (6 DOWNTO 0));
OPP3 : clock_divider
PORT MAP (clk_in => Clk_In,
Reset => temp_reset,
clk_out (1 DOWNTO 0) => temp_clk (1 DOWNTO 0));
OPP4 : invert_decoder
PORT MAP (input (1 DOWNTO 0) => temp_clk (1 DOWNTO 0),
output (3 DOWNTO 0) => anod_sel (3 DOWNTO 0));
end Behavioral;
```

clockdivider

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity clkdiv is
port(
clock_50:in std_logic;
clr: in std_logic;
clock_q:out std_logic);
end entity;
architecture behaviour of clkdiv is
signal q:std_logic_vector(23 downto 0);

begin
 --clock divider
 process(clock_50,clr)
 begin
  if clr = '1' then
   q <= X"000000"; -- hex number
```

```
 elsif clock_50'event and clock_50='1' then
  q <= q+1;
 end if;
end process;
clock_q <= q(6);
end architecture;
```

constraints

```
##3 LED out (3 ultrasonic için)
##2 motor out
##3 ultrasonic trigger out
##3 echo pulse pin in
##1 FPGA clk in

# Clock signal

set_property PACKAGE_PIN W5 [get_ports clock]
set_property IOSTANDARD LVCMOS33 [get_ports clock]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clock]
# LEDs for ultrasonic sensor
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]

##Pmod Header JC
```

```
##Sch name = JC1

set_property CLOCK_DEDICATED_ROUTE FALSE [get_ports {echo[0]_IBUF}]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_ports {echo[1]_IBUF}]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_ports {echo[2]_IBUF}]
set_property PACKAGE_PIN J1 [get_ports {echo[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {echo[0]}]
##Sch name = JC2
set_property PACKAGE_PIN L2 [get_ports {echo[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {echo[1]}]
##sch name = JC3
set_property PACKAGE_PIN J2 [get_ports {echo[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {echo[2]}]

##Pmod Header JC
##Sch name = JC4
## Using the same trigger for three sensors but have 3 different trigger module

set_property PACKAGE_PIN H1 [get_ports {trigger[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {trigger[0]}]
##Sch name = JC7
set_property PACKAGE_PIN K2 [get_ports {trigger[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {trigger[1]}]
##Sch name = JC8
set_property PACKAGE_PIN H2 [get_ports {trigger[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {trigger[2]}]


#7 segment display
```

```
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

#set_property PACKAGE_PIN V7 [get_ports dp]
#       set_property IOSTANDARD LVCMOS33 [get_ports dp]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
```

##Pmod Header JB --In(0,4) of motor driver - 4 output pins

set_property IOSTANDARD LVCMOS33 [get_ports Motor_L_backward]

set_property IOSTANDARD LVCMOS33 [get_ports Motor_L_forward]

set_property IOSTANDARD LVCMOS33 [get_ports Motor_R_backward]

set_property IOSTANDARD LVCMOS33 [get_ports Motor_R_forward]

set_property PACKAGE_PIN A14 [get_ports Motor_L_backward]

set_property PACKAGE_PIN A16 [get_ports Motor_L_forward]

set_property PACKAGE_PIN B15 [get_ports Motor_R_backward]

set_property PACKAGE_PIN B16 [get_ports Motor_R_forward]