FPGA Obstacle Avoidance Robot Using VHDL

Article · June 2020

1 author:

Gavin Anjitha
Ceymoss Technology

SHEFFIELD HALLAM UNIVERSITY

ENGINEERING PROGRAM

OBSTACLE AVOIDANCE ROBOT USING FPGA

MINI PROJECT REPORT

Submitted By

PALLATTARAGE GAVIN ANJITHA

M.ENG. (HONS) IN ELECTRICAL AND ELECTRONIC ENGINEERING

OCTOBER 2016

Table of Contents

Introduction 1
System Overview 1
Design and Implementation 3
Sensors 3
Motors 4
FPGA Development Board – Processing 6
Conclusion and Critique 7
References 8

Introduction

FPGAs (Field Programmable Gate Arrays) are widely used in real time tasks in recent years. The main advantage of FPGA based real time systems is its ability to handle concurrent tasks. The hardware description languages allow programmers to think in different ways. The ability to run concurrent tasks limits only by the hardware resources. This concurrency is very helpful in mobile robots. A robot with number of sensors and numbers of motors could be controlled concurrently with use of a single FPGA chip.

So that, this is a small attempt to implement a mobile robot which avoids obstacles with use of range sensors and wheeled motors. The mobile robot platform can move forward, backward, turn right, left. With use of the three range sensors the robot avoids obstacles. The algorithm is developed using combinational logic. This obstacle avoidance system can be implemented in medical assertive devices, industrial robots and outdoor / indoor navigation robots. While a microprocessor could be used for real time systems, it lacks the ability to parallel data processing in time critical applications such as an obstacle avoidance system.

System Overview

The obstacle avoidance mobile robot consists of three sub systems.

1. Three ultrasonic sensors obstacle detection range of 120^0
2. Mini FPGA board – EP2C5 Mini FPGA Development Board
3. Two DC motors with a H – Bridge motor driver

The figure 1 shows the simple overview of the system.

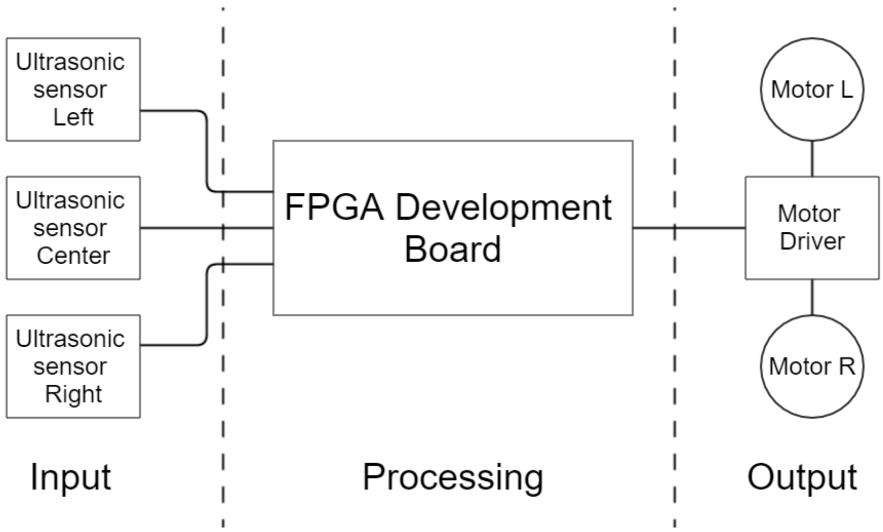


Figure 1 – System Overview

The parts assembly of the mobile robot is shown in the figure 2. The three ultrasonic range sensors are mounted in order to cover a range of 120^0. The parts list is shown in table 1.

Table 1 – List of Components

Component	Specifications
Ultrasonic Sensor	HC-SR04
FPGA Development Board	EP2C5T144C8N – EP2C Mini Board
Motor Driver	L298N H- Bridge
Motors	12V DC 300 rpm Geared
Voltage Regulator	12V to 5V
Programmer	USB Blaster I for FPGA

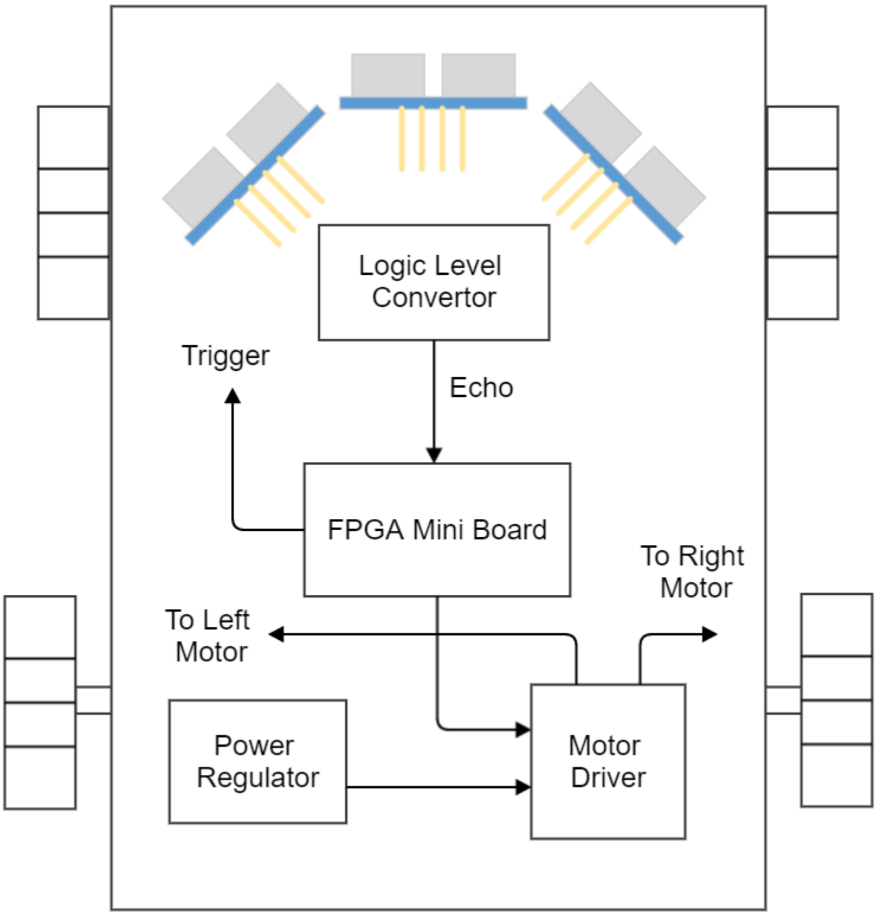


Figure 2 – Assembly of parts from above

Design and Implementation

Sensors

The input to the system is an array of three low cost HC-SR04 ultrasonic sensors. Each sensor has 4 pins, namely Vcc, Trigger, Echo and Ground. When a trigger signal of 10uS is sent to the ultrasonic sensor the sensor itself produces a set of eight burst signals through the transmitter. The receiver receives the reflected back signal and output a pulse proportional to the distance measured. A detailed image is shown in figure 3. The ultrasonic range sensors have a detection range of 4m with an accuracy of 3mm. The best of the ultrasonic range sensors could be gained through a FPGA chip. A large amount of sensor readings could be gained concurrently without any delay with use of an FPGA.

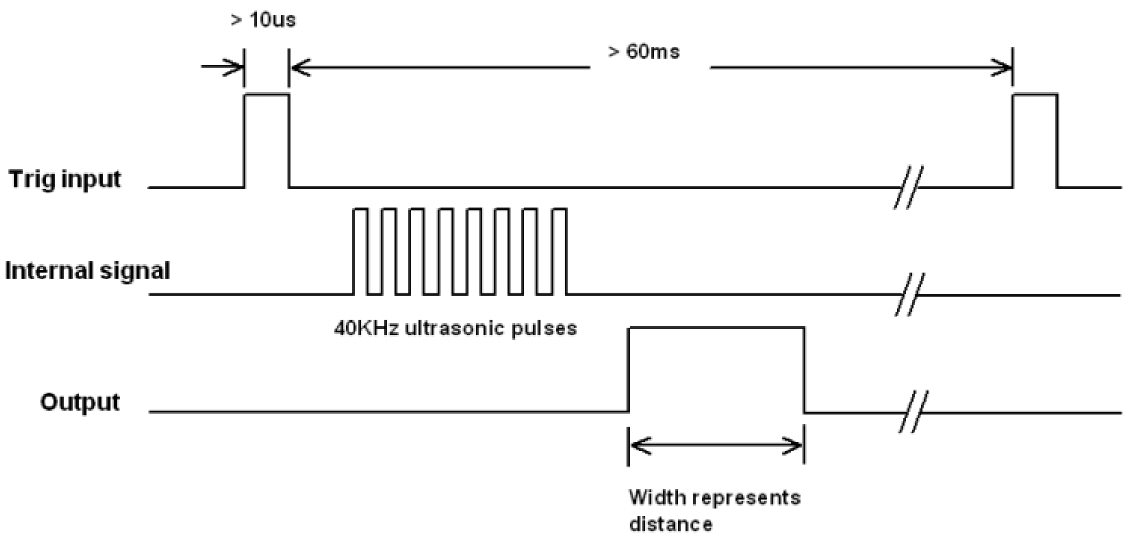


Figure 3 – Timing Diagram for HC – SR04 Ultrasonic Sensor

The ultrasonic sensor works at 5V and the FPGA board works at 3.3V. A trigger signal of 3.3V is enough for the ultrasonic sensor while the echo signal output of 5V could damage the FPGA board. So a voltage divider is used to level shift the echo signal.

With respect to FPGA development board programming, two components are needed to operate the HC-SR04 ultrasonic sensor module namely a trigger generator and a counter. The trigger generator generates a 10uS pulse at every 100ms. Then the ultrasonic sensor output an echo pulse proportional to the distance. This pulse is fed into a counter as the enable input. The counter outputs a number proportional to the distance. At each trigger the counter resets its value to ‘0’. A faster response could be gain if the time between triggers could be minimized. Its provided in the datasheet the minimum time between pulses should be at least 60ms.

The figure 4 shows the how a single ultrasonic sensor component is set upped in the FPGA program.

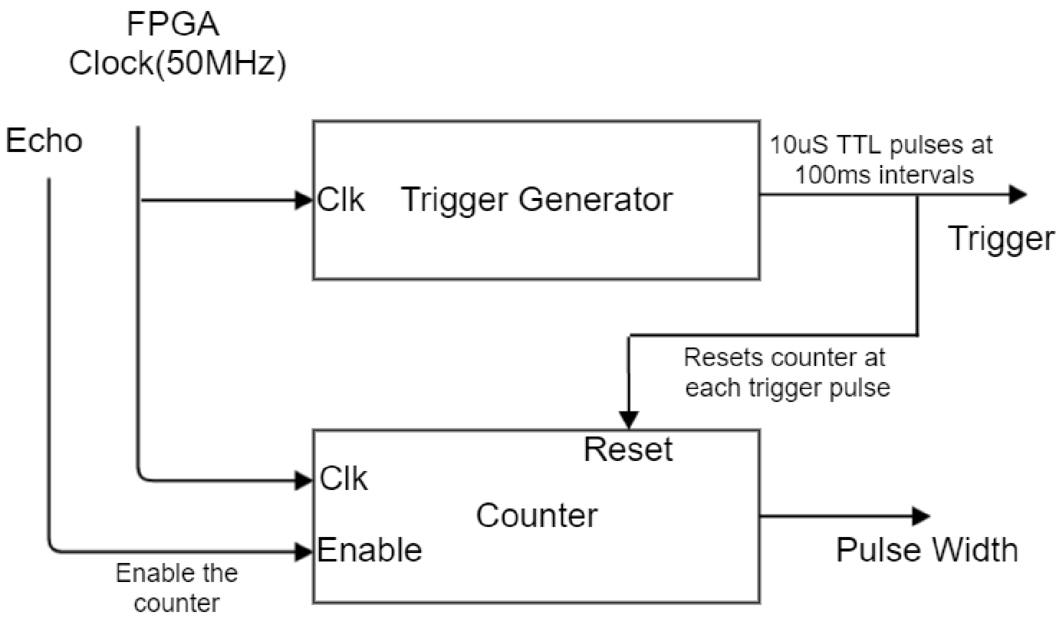


Figure 4 – Block diagram for Ultrasonic VHDL entity

Motors

To move the mobile platform forward, backward, turn left right two 12V 300rpm geared motors are used. In order to operate the motors L298N motor driver was used. When a PWM pulse is supplied to the motor driver it controls the speed of the motors accordingly. There are two pins for each motor for direction control. The PWM signal should be fed into one pin while the other pin is LOW.

The PWM signal generation is done by the FPGA development board. For this two VHDL entities are needed. A clock divider entity and a PWM generation entity. The clock division is done by a counter. A counter which runs at 50MHz is used. The 6^{th} bit of the counter is output as a clock to the PWM generation entity which has a period of 0.00256ms. Then the PWM entity sets the period and the duty cycle. The period is predefined to get a 2kHz wave.

At first the signal was generated by simulation. The figure 5 show the generated PWM signal in simulation. Then the 2kHz signal was generated at the laboratory. The generated PWM is shown in figure 6. Through the oscilloscope, the signal was observed. Then using a LED, the PWM signal was checked. Finally, the motor driver was set upped and the motors were run.

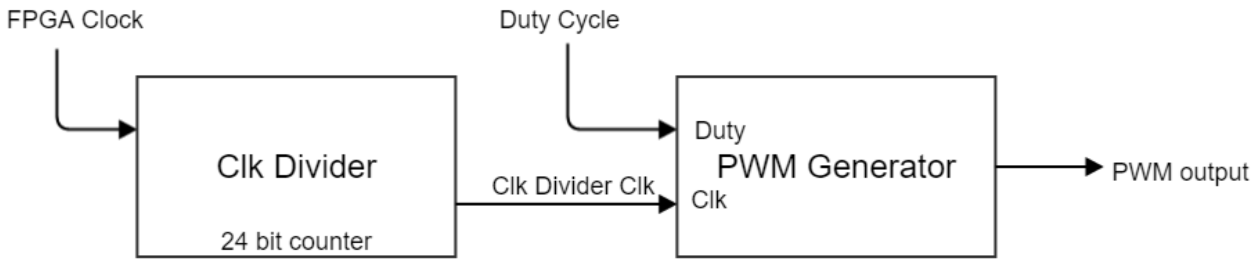


Figure 5 – PWM entity block diagram

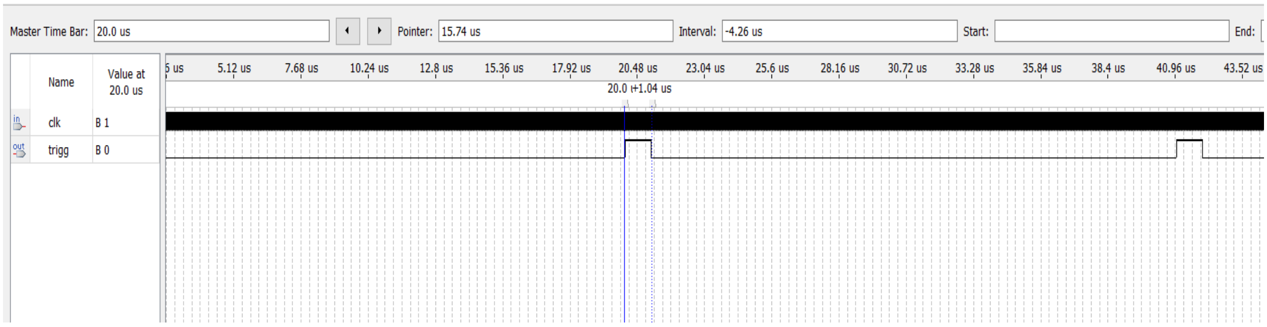


Figure 6 – PWM signal simulation waveform

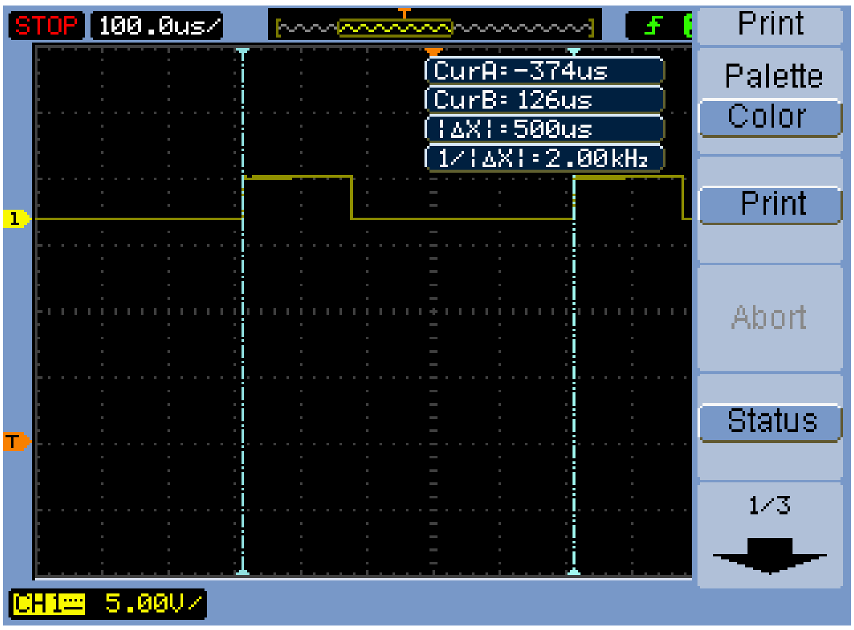


Figure 7 – PWM signal oscilloscope waveform

FPGA Development Board – Processing

The FPGA development board used in this mini project is the Altera EP2C5T144C8N – EP2C5 Mini Board. It is a development board without integrated peripherals. It is operated from a 50MHz crystal oscillator. There are 89 I/O pins. The FPGA development board is responsible for generating the PWM signal, taking readings from the ultrasonic sensor and movement of the motors. The VHDL programs were written from Altera Quartus II software.

According to the VHDL program the FPGA chip takes input from the ultrasonic sensor and detect obstacles. Then according to a rule base the movement of the motors are controlled in order to get away from the obstacle and move forward. The distance to keep away is defined by the designer. This distance is set as 20cm. This value is not taken from an experimental procedure. So that, it might not be the best value for the mobile robot to operate efficiently. As there is an array of three ultrasonic sensors, three inputs are fed into the top level entity. Then according to the rule base shown in table 2, the motors are operated. The PWM changes dynamically in order to get the desired movement. For example, to move backward in right direction a low PWM value is set for the right motor wheel and a high PWM value is set for the left motor.

Table 2 – Mobile robot Rule base

Obstacle detected	**Motor movement**
000	Forward
001	Turn Left
010	Forward
011	Reverse Right: PWM change Left motor – High, Right Motor - Low
100	Turn Right
101	Forward
110	Reverse Left: PWM change Left motor – Low, Right Motor - High
111	Reverse Right: PWM change Left motor – High, Right Motor - Low

This mobile robot is operated according to combination logic from a rue base table. So that, no intelligent decisions are made.

Conclusion and Critique

As real time processing tasks are trending nowadays, the idea for developing an obstacle avoidance mobile robot using a FPGA chip seemed to be a good idea. FPGA chips are better at handling concurrent tasks in real time activities. But harder at programming than a microprocessor. Even though this project handle only 3 input sensors and two motors, the FPGA chip can handle a lot more sensors and motors at the same time without any delay. The mobile robot in this mini project takes 3 inputs from the ultrasonic sensors and according to a rule base the motors are controlled. The mobile robot operated successfully at simple obstacles.

The placement of the ultrasonic sensor array was not experimental. So that, the robot did not operate successful at complex obstacles like corners. The reason was in corners the sensors were not operating efficiently and correctly. The figure 8 shows how an ultrasonic sensor operates at a 45^0 angle as well as in different conditions. The sensor cannot receive correct signal at corners. As this is a combinational logic the robot cannot take intelligent decisions as well.

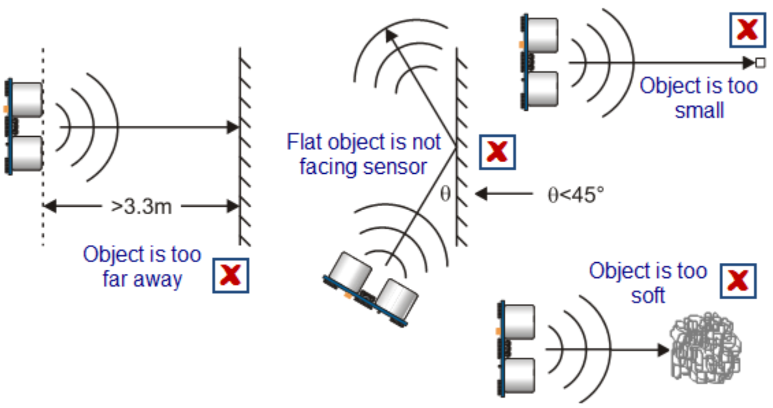


Figure 8 – Ultrasonic sensor operation at different conditions

The same obstacle avoidance robot can be developed to an efficiently working robot. There are two things need to be done. The tasks such as the ultrasonic distance reading and motor control can be made responsible for the FPGA chip and then the algorithm can be programmed in a SoC. Then the algorithms can be made more intelligent and efficient. While the sensors and actuators are handled by the FPGA itself without any delay. The other requirement is to introduce more sensors into the mobile robot. So that, more resourceful decisions could be taken.

References

[1]*An FPGA-based Portable Real-time Obstacle Detection and Notification System*, 1st ed. School of Engineering and Technology Central Michigan University Mount Pleasant, Michigan, USA, 2016.

[2]"Arduino Modules - L298N Dual H-Bridge Motor Controller", *Instructables.com*, 2016. [Online]. Available: http://www.instructables.com/id/Arduino-Modules-L298N-Dual-H-Bridge-Motor-Controll/. [Accessed: 02- Oct- 2016].

[3]S. C, "HC-SR04 Ultrasonic Sensor", *Arduinobasics.blogspot.com*, 2012. [Online]. Available: http://arduinobasics.blogspot.com/2012/11/arduinobasics-hc-sr04-ultrasonic-sensor.html. [Accessed: 01- Oct- 2016].

[4]"Designing a Simple Robot with NI LabVIEW - National Instruments", *Ni.com*, 2016. [Online]. Available: http://www.ni.com/white-paper/10460/en/. [Accessed: 31- Sep- 2016].

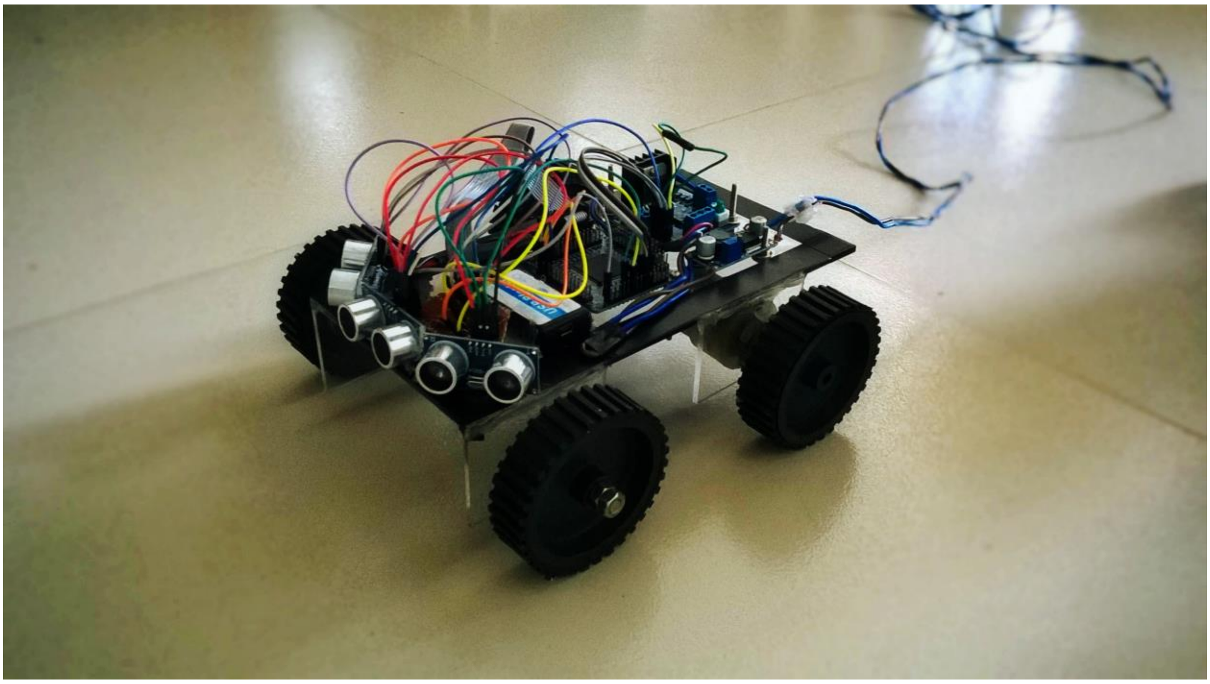
[5]"Getting started with the EP2C5 Cyclone II Mini Board", *Leonheller.com*, 2016. [Online]. Available: http://www.leonheller.com/FPGA/FPGA.html. [Accessed: 20- Sep- 2016].

[6]"Implementation of Obstacle-Avoidance Control for an Autonomous Omni-Directional Mobile Robot Based on Extension Theory", 2012. .

[7]*How to Implement VHDL design for a Range sensor on an FPGA.*. https://www.youtube.com/watch?v=PJkiDAKVTFg: Mittuniversitetet, 2014.

[8]*Lesson 82 - Pulse-width modulation PWM*. https://www.youtube.com/watch?v=EGNdoWRfHS0: LBEbooks, 2012.

Appendix



robot.vhd

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

entity robot is
	port(
			fpgaclk: 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,
			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(
                fpgaclk: 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(fpgaclk,X"00BE",pwm_1); -- generate pwm for the
motors
--PWM2:pwm port map(fpgaclk,X"00C3",pwm_2);
--
PWM1:pwm port map(fpgaclk,duty_1,pwm_1); -- generate pwm for the
motors
PWM2:pwm port map(fpgaclk,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(fpgaclk,echo,trigger,ultrasonic);

process(ultrasonic)
begin
     case (ultrasonic) is
```

```
            when "000" => forward <= '1';backward <=
'0';turn_right<='0';turn_left<='0'; duty_1 <= X"00BE"; duty_2 <=
X"00C3";
            when "001" => forward <= '0';backward <=
'0';turn_right<='0';turn_left<='1'; duty_1 <= X"00C3"; duty_2 <=
X"00C3";
--          when "001" => forward <= '1';backward <=
'0';turn_right<='0';turn_left<='0'; duty_1 <= X"00C3"; duty_2 <=
X"0041";
            when "010" => backward <= '1';forward <=
'0';turn_right<='0';turn_left<='0'; duty_1 <= X"0041"; duty_2 <=
X"00C3";
            when "011" => turn_left <= '0';backward <= '1';forward <=
'0';turn_right<='0'; duty_1 <= X"00C3"; duty_2 <= X"0041";
--          when "011" => turn_left <= '1';backward <= '0';forward <=
'0';turn_right<='0'; duty_1 <= X"00C3"; duty_2 <= X"00C3";
            when "100" => forward <= '0';backward <=
'0';turn_right<='1';turn_left<='0';duty_1 <= X"00C3"; duty_2 <=
X"00C3";
--          when "100" => forward <= '1';backward <=
'0';turn_right<='0';turn_left<='0';duty_1 <= X"0041"; duty_2 <=
X"00C3";
            when "101" => forward <= '1';backward <=
'0';turn_right<='0';turn_left<='0';duty_1 <= X"00BE"; duty_2 <=
X"00C3";
            when "110" => turn_right <= '0';turn_left <= '0';backward
<= '1';forward <= '0'; duty_1 <= X"0041"; duty_2 <= X"00C3";
--          when "110" => turn_right <= '1';turn_left <= '0';backward
<= '1';forward <= '0'; duty_1 <= X"00C3"; duty_2 <= X"00C3";
            when "111" => backward <= '1';forward <=
'0';turn_right<='0';turn_left<='0';duty_1 <= X"00C3"; duty_2 <=
X"0041";
      end case;
end process;

led(2)<= not(ultrasonic(2));
led(1)<= not(ultrasonic(1));
led(0)<= not(ultrasonic(0));

end architecture;
```

clkdiv.vhd

```
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;
```

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;
```

ultrasonic.vhd

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

entity ultrasonic is
	port(
			fpgaclk: 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(fpgaclk,pulse,not(trigg),pulse_width);
trigger_generation :
	trigger_generator port map(fpgaclk,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;
```

three_ultrasonic.vhd

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

entity three_ultrasonic is
     port(
                fpgaclk: 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(
                fpgaclk: 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(fpgaclk,pulse(0),triggerOut(0),ultrasonic_out(0));
ultrasonic_Middle:ultrasonic port
map(fpgaclk,pulse(1),triggerOut(1),ultrasonic_out(1));
ultrasonic_Right:ultrasonic port
map(fpgaclk,pulse(2),triggerOut(2),ultrasonic_out(2));

end architecture;
```

trigger_generator.vhd

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

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

architecture behaviour of trigger_generator 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;

signal resetCounter:std_logic;
signal outputCounter: std_logic_vector(23 downto 0);

begin

trigger_gen:counter generic map(24)
                        port
map(clk,'1',resetCounter,outputCounter);

     process(clk)

     constant ms100:std_logic_vector(23 downto
0):="010011000100101101000000";--20ns/100ms
--   constant ms100And20us: std_logic_vector(23 downto
0):="010011000100111100100110";
     constant ms100And20us: std_logic_vector(23 downto
0):="010011000100110100110011";--20ns/(100ms+20us)

     begin
          if(outputCounter > ms100 and outputCounter <
ms100And20us) then
               trigg <= '1';
          else
               trigg <='0';
          end if;

          if(outputCounter = ms100and20us or
outputCounter="XXXXXXXXXXXXXXXXXXXXXXXX") then
               resetCounter <= '0';
          else
               resetCounter <= '1';
          end if;
     end process;

end architecture;
```

pwm.vhd

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

entity pwm is
--generic(N : integer:=7);
port(
          CLOCK_50: in std_logic;
          duty: in std_logic_vector(15 downto 0);
          pwm : out std_logic);
          --GPIO_0:out std_logic_vector(2 downto 0);
          --LEDR: out std_logic_vector(15 downto 0);
          --SW: in std_logic_vector(0 downto 0));
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";

     clr <= '0';

     process(clk,clr)
     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)
     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;
```