Mini Project Report

on

FPGA-BASED ROBOTIC ARM MOVEMENT CONTROL

Bachelor of Technology

In

Electronics Engineering

Submitted by

GAURAV DEV (20ELB171)
MUNAZIR REZA (20ELB172)
MARIYA IFTEKHAR (20ELB173)

Under the supervision of

Dr. NAUSHAD ALAM

DEPARTMENT OF ELECTRONICS ENGINEERING
ZAKIR HUSAIN COLLEGE OF ENGINEERING & TECHNOLOGY
ALIGARH MUSLIM UNIVERSITY, ALIGARH-202002
2022-23

STUDENTS' DECLARATION

We hereby certify that this project work entitled "**FPGA-Based Robotic Arm Movement Control**", submitted to Department of Electronics Engineering of Z.H.C.E.T, AMU Aligarh, is the record of our original work carried out during our 3rd year of B.Tech. The project was completed under the guidance of Prof. Naushad Alam, Department of Electronics Engineering, Zakir Husain College of Engineering & Technology, Aligarh Muslim University, Aligarh.

GAURAV DEV	MUNAZIR REZA	MARIYA IFTEKHAR
(20ELB171)	(20ELB172)	(20ELB173)

CERTIFICATE

This is to certify that the project report entitled “FPGA-BASED ROBOTIC ARM MOVEMENT CONTROL” that is being submitted to the Department of Electronics Engineering, Zakir Husain College of Engineering and Technology, Aligarh Muslim University Aligarh, in partial fulfillment for the award of the degree of Bachelor of Technology in Electronics Engineering, during the session 2022-23, is a record of candidates’ own work carried out under my supervision.

Dr. Naushad Alam
Professor
Department of Electronics Engineering
Z.H.C.E.T, AMU, Aligarh

ABSTRACT

In this era of fast-paced technological advancements, automation has become an integral part of almost every industry, from manufacturing to healthcare. When we think of automation, we often envision robots, which have revolutionized the manufacturing sector. Among the most widely used robots in manufacturing applications is the Robotic Arm, which can perform tasks like assembly line operations, packaging and similar operations.

There are two main methods of controlling a robotic arm to perform specific tasks: using a microcontroller or designing a custom ASIC package. While a microcontroller-based approach offers programmability, it can be slow due to memory read/write operations required to execute instructions. Conversely, a custom ASIC package can deliver better and faster performance, but it is expensive for small scale production and can take longer time to build. Since an FPGA offers programmability and a quick prototype design process, it is the preferred technology for low-volume applications and can be used to implement the robotic arm controller.

In our project, we adopted the FPGA methodology and used the Xilinx Artix-7 series FPGA on the Nexys4DDR board. SG90 servomotors were used to control the robotic arm, and Verilog HDL was used to program the FPGA. We implemented our design using FPGA and simulation using the Xilinx Vivado tool. An operational two-DOF FPGA-based robotic arm controller was developed by our team. The suggested arm is designed to automatically follow a specific trajectory.

ACKNOWLEDGEMENT

We want to convey our sincere gratitude to our supervisor, Dr. Naushad Alam, for providing the motivation and necessary support to complete this project successfully. Throughout all the highs and lows of our mini-project, he has consistently been friendly, attentive, responsible, and encouraging. He dedicated a lot of time and effort to help us finish the project, offering insightful advice throughout. He has taught us a lot, and we sincerely express our gratitude to him for that.

We would also like to extend our gratitude to the Lab staff Mr. Md Salim & Mr. Sulaiman for helping us in utilizing the lab resources and providing all kind of technical support for the completion of this project.

Date: 02 May, 2023

Place: Aligarh

GAURAV DEV

MUNAZIR REZA

MARIYA IFTEKHAR

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

PWM: Pulse width Modulation

FPGA: Field Programmable Gate Array

ASIC: Application-Specific Integrated Circuit

HDL: Hardware Description Language

PMOD: Peripheral Module Interface

DC: Direct Current

CLB: Configurable Logic Block

ROM: Read Only Memory

IP: Internet Protocol

HLS: High-level synthesis

CPU: Central Processing Unit

RTL: Register Transfer Level

GUI: Graphical User Interface

LED: Light Emitting Diode

CNC: Computer Numerical Control

Chapter 1

Introduction

1.1 Problem Statement

To design an FPGA-controlled robotic arm that can perform a particular task autonomously.

1.2 Proposed Solution

The challenge lies in prototyping and implementing the movement algorithm on the FPGA board to ensure smooth arm movement. It is possible to create a robotic arm with 2 degrees of freedom using servo motors to control its movement. To put this strategy to the test, we can use the Nexys A7 board, which includes a Xilinx Artix-7 series FPGA. The algorithm can be written in Verilog HDL and simulated using the Vivado tool before being implemented on the FPGA.

1.3 Methodology

We need to produce PWM signals with a set period and adjustable duty cycle in order to drive the servo motors of the robotic arm. We used Verilog code to achieve this. The angle at which the arm was supposed to move was used to calculate the length of the PWM signal's on-time, which was then included into the Verilog code.

We used Pmod connections on the FPGA board to provide the PWM signals concurrently and separately to each servo motor. To link the servo motors to the FPGA board, a Pmod CON3 connector was used.

Chapter 2

The Basics of Servo Motors: An Overview

2.1 Background

Servo motors are DC motors with built-in gear systems and positional feedback circuits that allow shaft positions to be accurately controlled. The angular velocity and acceleration can also be controlled quite accurately.

The servo motors are designed for use in motion control applications that require high-precision positioning, quick reversing, and high performance. Servo motors are used in Drones, radar systems, automated production systems, Solar tracking systems, Camera focusing systems, CNC machines, Steer control of Autonomous vehicles, and other applications.

There are three terminals: Power, Ground, and Control, which are used to interface the servos. To drive the servo, a PWM signal with a period of 20 milliseconds and a frequency of 50 Hz is typically provided at the Control terminal. An SG90 servo motor is used in this project. A small, light, and potent servo motor is the SG90. The servo can rotate 180 degrees (90 degrees in each direction) and, despite being smaller, it nevertheless accomplishes many of the same tasks as conventional servos.

Figure 2.1 Pin diagram of servo motor

Wire Color	Name	Description
Brown	GND	Connected to ground
Red	VCC	Connected to the +5V supply
Orange	Control Signal	PWN signal input is given through the motor driver.

Table 1 Pin Description of SG90.

2.2 Servo Motor and Robotics

Servo motors are frequently referenced in the context of robotics and automation.
Whatever we want to achieve with robotics, we can always find a servo motor that fits the bill because:

- They come in a range of shapes and sizes.
- They come in different specifications of power and torque.
- Because of their reduced weight, servos don't become a hindrance in robotic arm's movement

- Servo motors are easily accessible on the market, making them simple to replace, should the need arise.
- The precision of servo motors, makes them a good choice for applications in areas which demand very tight tolerances.

2.3 Working

A servo motor is made up of a DC motor, a gear system, a position feedback sensor, and a control circuit. The DC motors operate at high speeds with low torque. The gear and shaft configuration of DC motors reduces this speed and increases the torque. The position sensor detects the position of the shaft about its fixed location and communicates the information to the control circuit. The control circuit decodes the position sensor signals, compares the present position of the motors to the desired position, and controls the direction of rotation of the DC motor appropriately. Servo motors generally demand a 4.8V to 6V DC source.

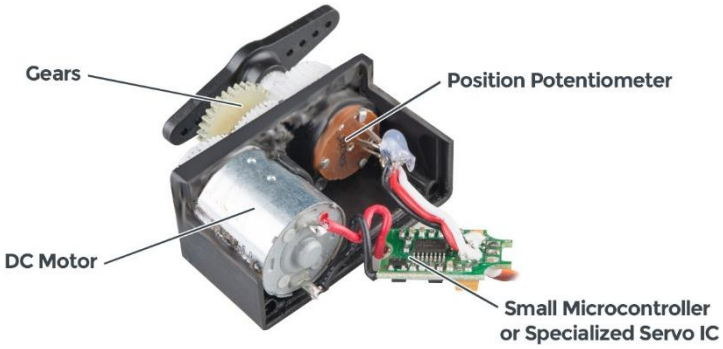


Figure 2.2 Construction of Servo motor SG90

The PWM sent to the motor determines the position of the shaft, and the rotor will rotate to the suitable position based on the duration of the pulse received via the control wire. A pulse is given to servo every 20ms, and width of the pulse determines the position of the shaft.

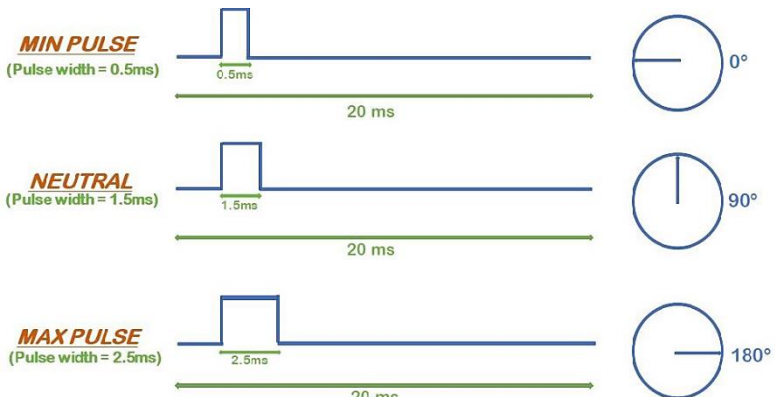


Figure 2.3 Width of pulse corresponding to positions of servo

According to the pulse width, the servo is moved to the desired position. When a servo motor has retained a position, it will try to resist any change from that position, if any mechanical force is applied to its shaft. A servo's torque rating refers to the maximum amount of force it can handle. Servos cannot maintain their position indefinitely; to instruct a servo to maintain its position, the position pulse must be repeated every 20 ms.

Chapter 3

Field-programmable gate arrays (FPGAs): An Overview

3.1 Introduction

FPGA is a semiconductor device, and as the name suggests, it consists of an array of Configurable Logic Blocks (CLBs) connected with the help of programmable interconnects. Adders, flip-flops and multiplexers make up the bulk of CLBs. The logic blocks are capable of doing both straightforward and complex computing tasks. FPGAs can be configured for a particular purpose or application.

3.2 Why FPGA?

To implement the robotic arm movement algorithm we have other options as well, like, microcontroller or ASIC. For implementation, we need the following:

- Parallel distinct outputs, to drive servos simultaneously and independently.
- Cost-effective and Faster hardware prototyping, and design validation.

Now, we will compare other options and see why we preferred to choose FPGAs.

3.2.1 FPGAs v/s Microcontroller

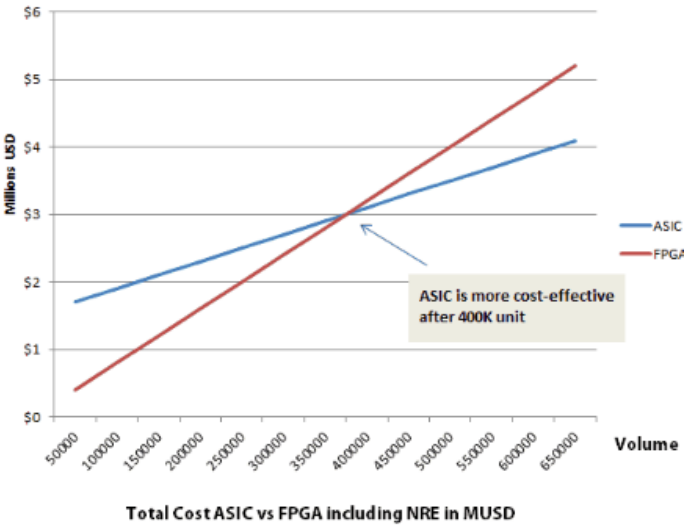
Both FPGA and Microcontroller are embedded within other devices and perform a certain application at a time. However, a Microcontroller can only support a limited number of peripherals. In contrast, in FPGA, we can program as many peripherals as the FPGA's CLBs allow. As a result, FPGAs are more customizable than microcontrollers are. Also a microcontroller, unlike FPGAs, cannot drive parallel outputs on its own. Additionally, compared to a microcontroller, the FPGA has no abstraction layers or fewer abstraction layers. These FPGA features enable faster performance compared to a micro-controller.

3.2.2 FPGA v/s ASIC

The FPGA can be modified with a new design. They can even alter a portion of the chip while leaving the remaining parts unchanged. ASIC, on the other hand, has permanent circuitry. The application-specific circuit cannot be modified once it has been taped out into silicon ASICs can contain complete analogue circuitry, whereas FPGAs are not appropriate for analogue designs..

FPGA development costs far less than ASIC development. However, high volume mass production is not a good fit for FPGAs

Due to extremely high NRE costs, ASIC costs begin higher even for low volume, although the slope is flatter.



In terms of volume manufacture, ASICs are becoming less expensive than FPGAs.

Figure 3.1: Comparison of FPGA and ASIC

3.3 Our FPGA Board

We used a Nexys-4 DDR FPGA board with an Artix-7 FPGA (part number XC7A100T-1CSG324C). This FPGA board has 15,850 CLBs with an internal clock speed of more than 450MHz, each with four 6-input LUTs and eight flip-flops. This board also features many I/O ports and switches, which can be utilized to control the movement of the proposed robotic arm.

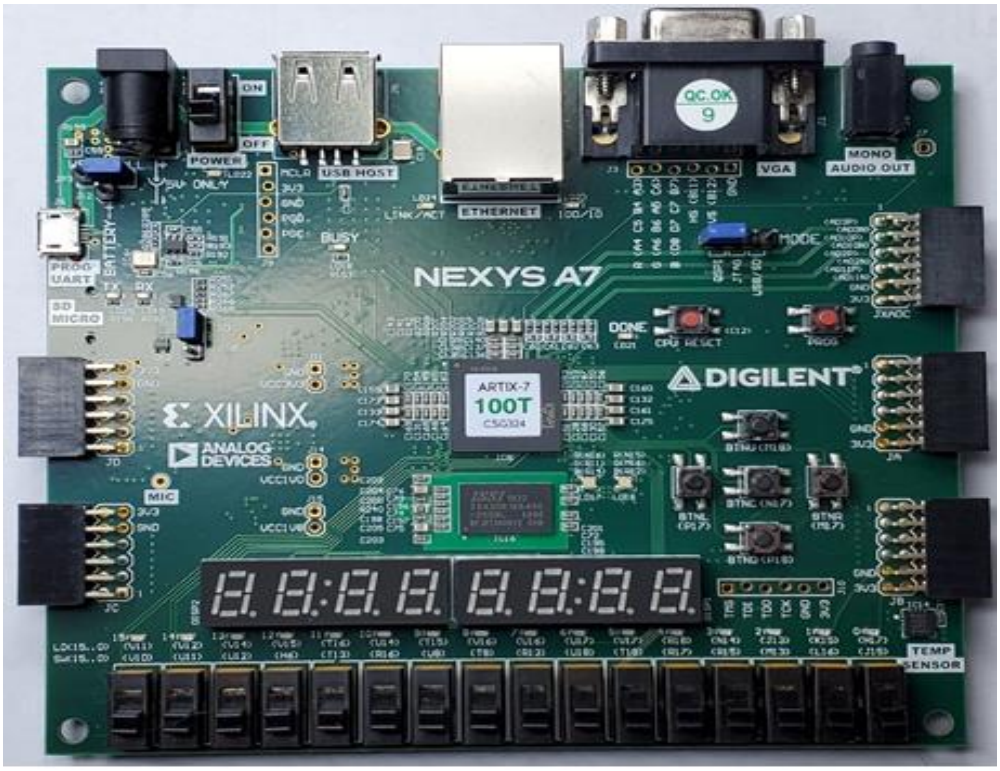


Figure 3.2 Xilinx Nexys-4 DDR Artix-7 FPGA Board

Chapter 4

Hardware Description Language and Algorithm Implementation

4.1 Hardware Description Language

The Hardware Description Language (HDL) is a computer programming language that is used to program electronics and digital circuit and devices. The design, layout, functionality can be programmed using HDL. The HDL textual description includes operators, expressions, statements, inputs, and outputs. Instead of a computer-executable file, HDL compilers produce a gate map. After that, the gate map is downloaded to the programming devices to make sure the desired circuits operates as intended. It is a great programming language for CPLDs and FPGAs since it enables us to express any digital circuit's structural, behavioral and gate-level features.

The two most popular hardware description languages are Verilog and VHDL. For our Project work, we have chosen to work with Verilog.

4.1.1 Verilog

Verilog is a hardware description language (HDL) that is used to define digital system designs such as microprocessors, ROM, RAM, flip-flops and many more HDL is used to describe any level of digital hardware. HDL designs are not dependent on technology, it is easy to construct and debug, and often this is more useful than schematics, particularly for complex circuits.

By streamlining the technique, Verilog was designed to make HDL more resilient and adaptive. In the semiconductor business, Verilog is now the most extensively used and practiced HDL.

4.1.2 Abstraction Levels of Verilog

Verilog can support a design at several degrees of abstraction, including

- Behavioral level
- Dataflow level
- Gate level

1) Behavioral level

This is the top level of abstraction in Verilog HDL. Without worrying about the specifics of the hardware implementation, a module can be implemented in terms of the desired design algorithm. The circuit functioning is best described by this level of abstraction, but it is also the most challenging to synthesize.

2) Dataflow level

The Dataflow level specifies the features of a circuit by using the flow of data between hardware registers. It is similar to the Boolean expression of a function. This method is easily structured and then implemented.

3) Gate level

In this method, Logic gates and connections between them are used to implement the module. Components are linked together by signals, resembling a schematic diagram. Any input signal of a component that changes its value activates that component. When two or more components are triggered simultaneously, they will all take action simultaneously.

4.1.3 Features of Verilog Language

- Verilog is case-sensitive.
- In Verilog, packages are not required.
- The majority of Verilog's syntax is based on the "C" programming language.
- Behavioral, Gate, and Switch level models of digital circuits can be created using Verilog.

4.2 Algorithm

The duration during which the PWM signal is in its on-state at the input, as previously mentioned, it controls the degree of rotation of a servo motor.
As a result, the complete code was divided into the following modules in response to demand:

1. Top Module
2. ROM Module
3. Saw-tooth Generator Module
4. Address Generator Module
5. PWM Generator Module

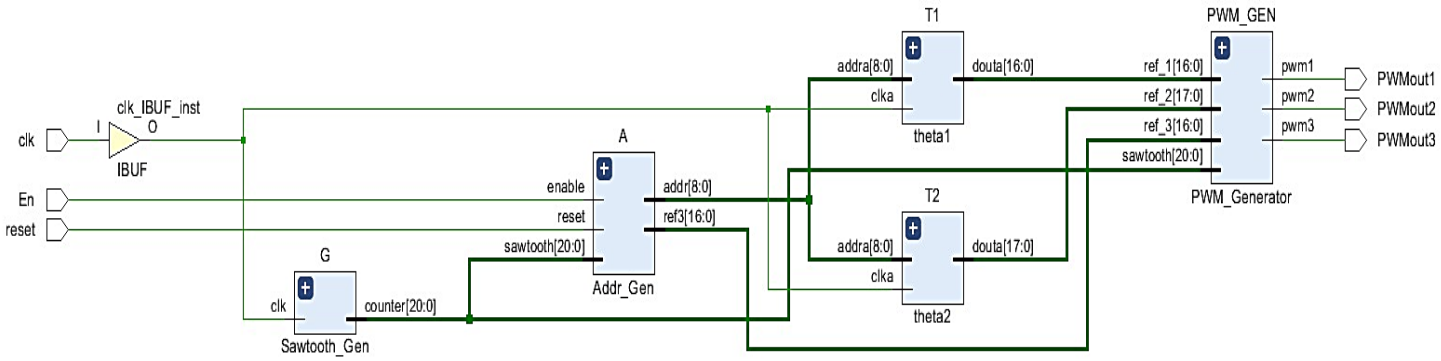


Figure 4.1 Pre-Synthesis Schematic

4.2.1 Inverse Kinematics

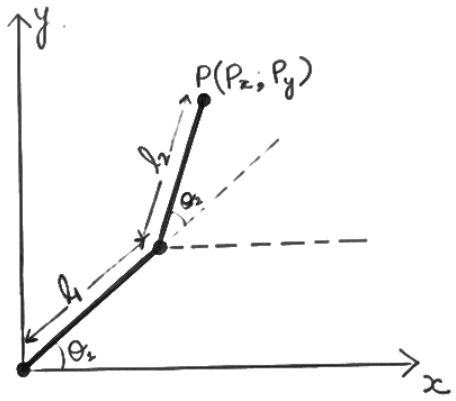


Figure 4.2: Inverse Kinematics

$$Px = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \quad \text{..... (1)}$$

$$Py = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \quad \text{..... (2)}$$

Squaring and adding equations (1) and (2) we get,

$$\cos(\theta_2) = \frac{Px^2 + Py^2 - {l_1}^2 + {l_2}^2}{2.l_1.l_2}$$

$$\sin(\theta_2) = \pm\sqrt{1 - \cos(\theta_2)^2}$$

Or we can write $\theta_2 = \tan^{-1}\left(\frac{\sin(\theta_2)}{\cos(\theta_2)}\right)$

After manipulating equations (1) and (2)

$$\sin(\theta^1) = \frac{Py(l_1 + l_2\cos(\theta^2) - Px(l_2\sin(\theta^2))}{l1^2 + {l_2}^2 + l^1.l_2\cos(\theta^2)}$$

$$\cos(\theta_1) = \frac{Px(l_1 + l_2\cos(\theta_2) - Py(l_2\sin(\theta_2)}{{l_1}^2 + {l_2}^2 + 2l_1.l_2\cos(\theta_2)}$$

$$\theta_1 = \tan^{-1}\left(\frac{\sin(\theta_1)}{\cos(\theta_1)}\right)$$

4.2.2 Trajectory Planning

To move the end effector of the robotic arm in the desired trajectory, we would need a mathematical representation of the desired path. For our test subject we have chosen the following trajectory:

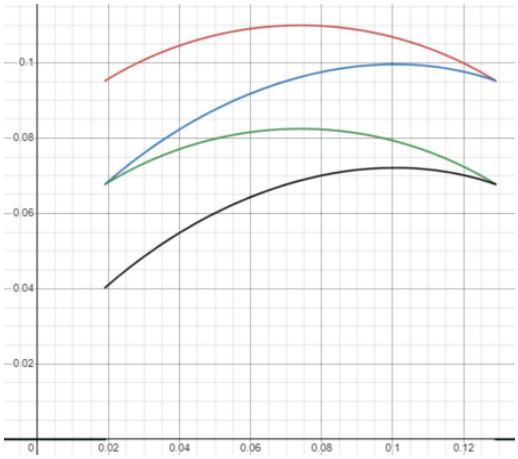


Figure 4.3: Test Pattern

The mathematical equations for the trajectory are as follows:

$$f(x) = \left\{ 0 < x - w < l : \sqrt{\left(l^2 - \left(x - w - \frac{l}{2}\right)^2\right)} \right.$$

$$g(x) = \left\{ 0 < x - w < l : \sqrt{\left(l^2 - \left(x - w - \frac{l}{2}\right)^2\right)} \right.$$

$$h(x) = \left\{ 0 < x - w < l : \sqrt{\left(l^2 - \left(x - w - \frac{l}{2}\right)^2\right)} \right.$$

$$t(x) = \left\{ 0 < x - w < l : \sqrt{\left(l^2 - \left(x - w - \frac{l}{2}\right)^2\right)} \right.$$

Figure 4.4: Mathematical Equation for Test Pattern

Now, to control the robotic arm to trace this exact trajectory, we would need both servos to be synchronized. To control servos, first, the trajectory is broken into waypoints.

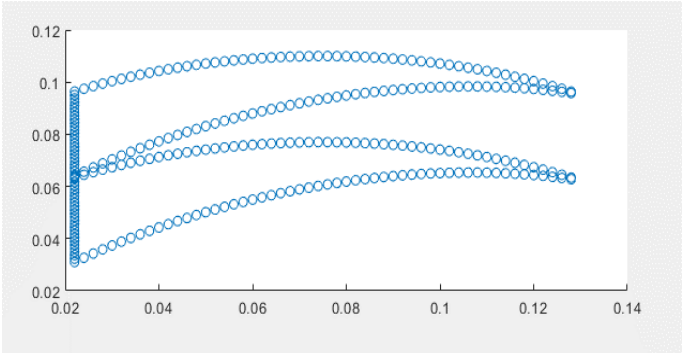


Figure 4.5: Waypoints to Trace the Test Pattern

Then using inverse kinematics the angle of the revolute joint of each link is determined. The angles obtained are then converted into binary counter values, and saved in a .coe file. This file will be used to generate the desired PWM signal corresponding to each waypoint.

To do the mathematical calculations as described above, we have used MATLAB. Crucial snippets of the codes are attached here.

```
% Trajectory Generation
L = 0.11; b=0.3;  a = L*b;  w=0.02;
for i = 1:length(x)-1
    if((x(i)-w)<L && (x(i)-w)>0 )
        y1(i) = sqrt(L^2 - (x(i)-w-(L/2)).^2);
    end
end
for i = 1:length(x)
    if((x(i)-w)<L && (x(i)-w)>0 )
        y2(i) = sqrt(L^2 - (x(i)-w-(L/2)).^2) + b*(x(i)-w)-a;
    end
end
for i = 1:length(x)
    if((x(i)-w)<L && (x(i)-w)>0 )
        y3(i) = sqrt(L^2 - ((x(i))-w-(L/2)).^2)-a;
    end
end
for i = 1:length(x)
    if((x(i)-w)<L && (x(i)-w)>0 )
        y4(i) = sqrt(L^2 - (x(i)-w-(L/2)).^2) + b*(x(i)-w)-2*a;
    end
end
```

```
% Inverse Kinematics
Len1 = 0.087; Len2 = 0.084;
 px = wp(1,:); py = wp(2,:);

 c2 = (px.*px + py.*py -Len1*Len1-Len2*Len2)./(2*Len1*Len2);
 s21 = sqrt(1-c2.*c2);
 s22 = -sqrt(1-c2.*c2);
theta2 = atan2(s21,c2);

 denom = L1*L1 +L2*L2 + 2*L1*L2*cos(theta2);
 s1 = (py.*(Len1+Len2*cos(theta2))-px.*(Len2*sin(theta2)));
 c1 = (px.*(Len1+Len2*cos(theta2))+py.*(Len2*sin(theta2)));
theta1 = atan2(s1,c1);

q1 = 90+(180/pi)*theta1; q2 = (180/pi)*theta2;

pw1 = q1/90; pw2 = q2/90;

counter1  = pw1 *1e5; counter2  = pw2 *1e5;

binary1 = string(fliplr(de2bi(round(counter1))));
binary2 = string(fliplr(de2bi(round(counter2))));

binstr1=[];
for i =1:size(binary1,2)
binstr1 = strcat(binstr1,binary1(:,i));
end
binstr2=[];
for i =1:size(binary2,2)
binstr2 = strcat(binstr2,binary2(:,i));
end
```

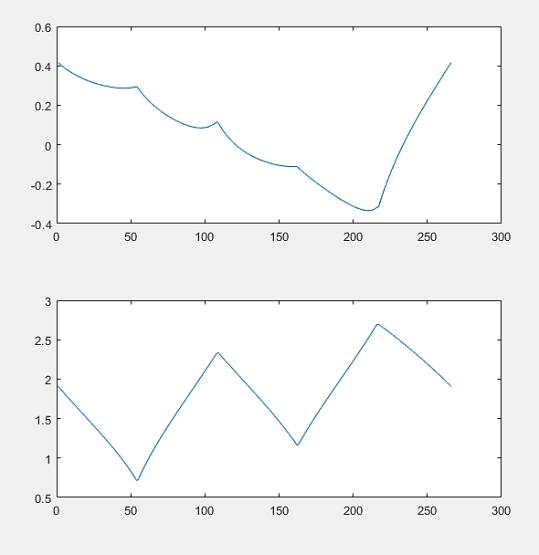


Figure 4.6: Graphical representation of Joint Angle calculated on MATLAB

4.3 Block Memory Initialization

To store angle counter values generated above, we need to initialize the Block Memory of FPGA. This is done by using the Block Memory Generator, under the IP catalog, in the Vivado tool. The .coe file is selected and the IP generator generates the necessary file for Block Memory initialization.

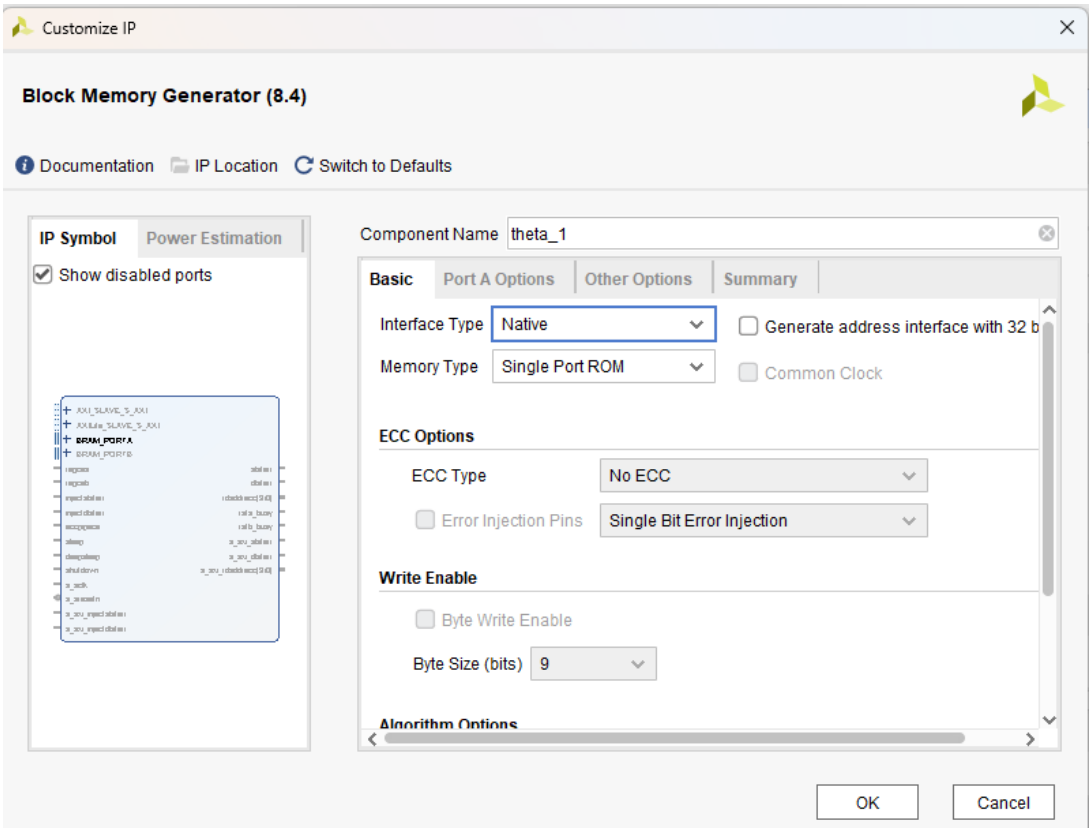


Figure 4.7: Block Memory Generator

4.4 HDL Code:

1. Top Module:

```
`timescale 1ns / 1ns

module PWM2(
    input clk,En,reset,
    output PWMout1,PWMout2,PWMout3
    );
    wire [20:0]sawtooth;
    wire [16:0]ROMout1;
    wire [17:0]ROMout2;
    wire [16:0]ref3;
    wire [8:0]addr;

Sawtooth_Gen G(clk,sawtooth);
Addr_Gen    A(En,reset,sawtooth,addr,ref3);
theta1   T1(clk,addr,ROMout1);
theta2   T2(clk,addr,ROMout2);
PWM_Generator PWM_GEN(ROMout1,ROMout2,ref3,sawtooth,PWMout1,PWMout2,PWMout3);

endmodule
```

2. Saw-tooth Generator Module:

```
module Sawtooth_Gen(
    input clk,
    output reg [20:0] counter=0
    );
always @(posedge clk)
begin

    if (counter < 'd1999999)
        counter <= counter+1;
    else begin
        counter <= 0;
    end
 end
endmodule
```

3. PWM Generator

```
module PWM_Generator(
input [16:0]ref_1,
input [17:0] ref_2,
input [16:0] ref_3,
input [20:0]sawtooth,
output pwm1,pwm2,pwm3
    );

    assign pwm1 = (sawtooth < ref_1+'d50000)?1:0;
    assign pwm2 = (sawtooth < ref_2+'d50000)?1:0;
    assign pwm3 = (sawtooth < ref_3+'d50000)?1:0;
 endmodule
```

4. Address Generator Module

```
module Addr_Gen(
   input enable,reset,
   input [20:0] sawtooth,
   output reg [8:0] addr='d485,
   output reg[16:0] ref3='d0
   );

    always @(negedge sawtooth[20])
 begin
    if(enable) begin
          if(addr=='d485)
           addr <= 0;
          else begin
             addr <= addr + 1;
             ref3 <= 0;
          end
          if(addr>='d436)
             ref3 <= 'd51440;
           else if(addr == 0 )
             ref3 <= 0;
    end

     if(reset)
    begin
    if(addr>0)
      addr <= addr-1;
      else addr<=0;

      ref3 <= 'd51440;
 end
 end
endmodule
```

5. Testbench

```
module PWM_tb;
reg clk=0;
reg En=1;
reg reset=0;
wire PWMout1;
wire PWMout2;
wire PWMout3;

PWM2 DUT(clk,En,reset,PWMout1,PWMout2,PWMout3);
always #5 clk = ~clk;
always begin
#50000000 En=0;
#30000000 En=1;
#100000000 En=0;
#150000000 reset = !reset;
end
```

6. Constraints

```
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports {clk}];

set_property IOSTANDARD LVCMOS33 [get_ports {clk}]
set_property IOSTANDARD LVCMOS33 [get_ports {PWMout1}]
set_property IOSTANDARD LVCMOS33 [get_ports {PWMout2}]
set_property IOSTANDARD LVCMOS33 [get_ports {PWMout3}]
set_property IOSTANDARD LVCMOS33 [get_ports {En}]
set_property IOSTANDARD LVCMOS33 [get_ports {reset}]

set_property PACKAGE_PIN E3 [get_ports {clk}]
set_property PACKAGE_PIN C17 [get_ports {PWMout1}]
set_property PACKAGE_PIN D18 [get_ports {PWMout2}]
set_property PACKAGE_PIN E18 [get_ports {PWMout3}]
set_property PACKAGE_PIN V10 [get_ports {En}]
set_property PACKAGE_PIN U11 [get_ports {reset}]
```

4.4.1 Simulation Output

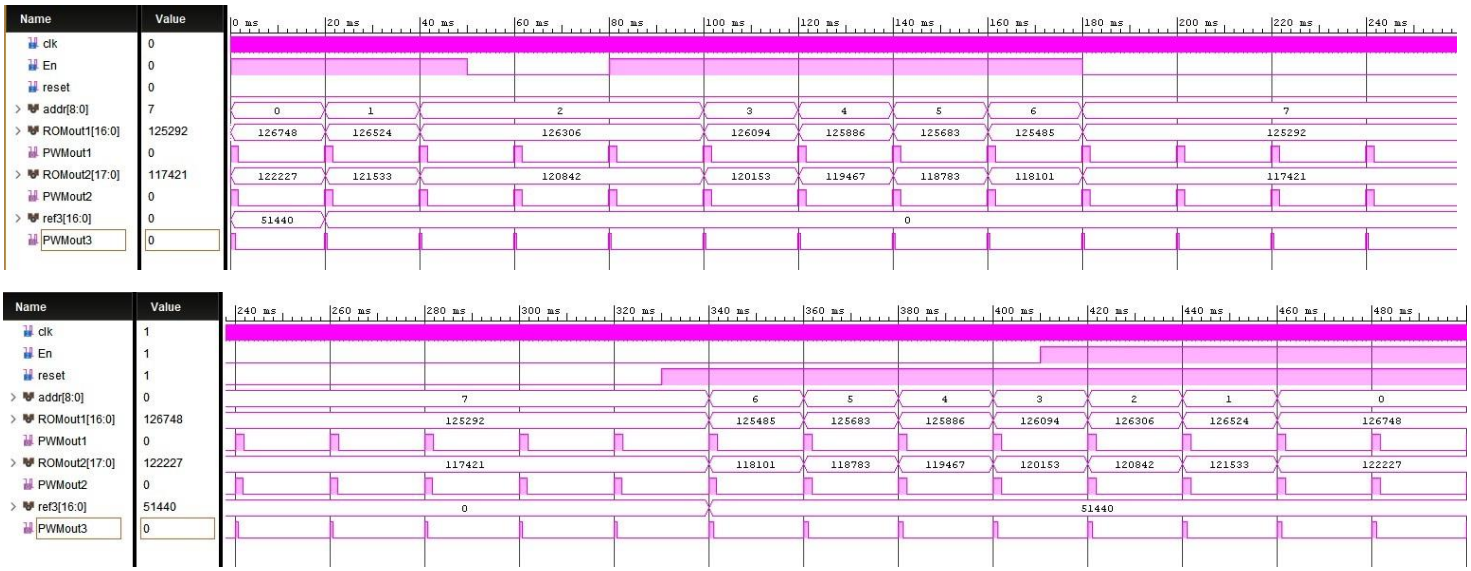


Figure 4.8: Behavioral Simulation

4.4.2 Post-Synthesis Schematic

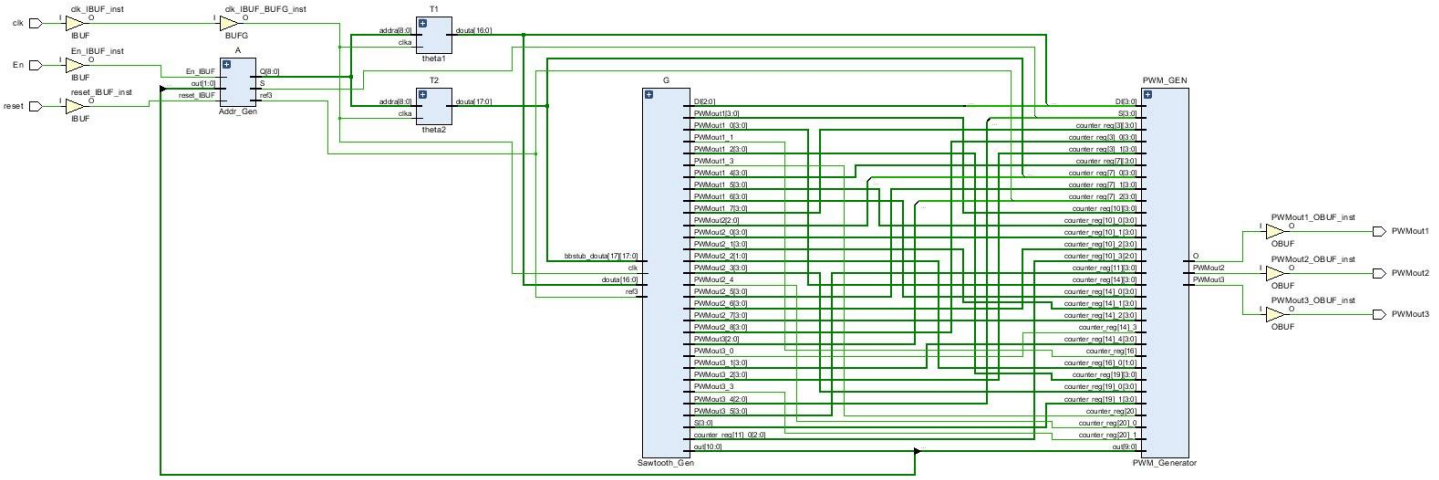


Figure 4.9 Post-Synthesis Schematic

Chapter 5

About Xilinx Vivado

5.1 Introduction

Xilinx provides the Vivado Design Suite for analysis, designing, and synthesis of hardware description language (HDL).

The Xilinx Vivado High-Level Synthesis (HLS) compiler provides a programming environment comparable to that available on both conventional and specialized CPUs for application development. Vivado HLS employs key processor compiler technologies to comprehend, analyze, and optimize C/C++ programs.

Without having to manually generate RTL, programmes written in C, C++, and SystemC can be directly targeted at Xilinx devices using the Vivado High-Level Synthesis compiler.

5.2 FPGA-Based Design Flow

The flow navigator displays the whole flow of the FPGA-based design. The flow is separated into following six phases:

- Adding Sources
- Simulation
- RTL Analysis
- Synthesis
- Implementation
- Program and Debug

1. Adding Sources

This is the first step in the design flow. All the crucial source files are created or added in this step. These files include Design source, Simulation source, and Constraints. These sources are important for the design, simulation and implementation of our desired system on FPGA.

2. Simulation

This component is used to perform the simulation after the design and simulation files have been completed. In this phase we can check and verify the functionality of our design. if the functionality is not met then we can edit our Verilog code, and run the simulation again.

3. RTL Analysis

After the successful simulation of Verilog code and verification of the design functionality. RTL analysis is performed to get the register level implementation.

4. Synthesis

After the RTL analysis, the RTL design is then forwarded for synthesis by selecting the run synthesis option. As soon as the synthesis is finished, we can plan the I/O using VIVADO's GUI in the I/O Ports section.

5. Implementation

After the completion of synthesis and I/O planning, next step is to implement the design. This is done by choosing the run implementation option under Implementation menu

6. Program and Debug

After the successful implementation, The next step is to generate Bitstream. Then We launch the Hardware Manager after generating the Bitstream. The FPGA board is then connected to the computer interfaced through a USB to micro USB adapter. Under the hardware manager, then we selected the FPGA board by clicking the target option. We choose the board and then click the Program Device option. As the device is configured, an inbuilt LED on the FPGA board generates a done signal.

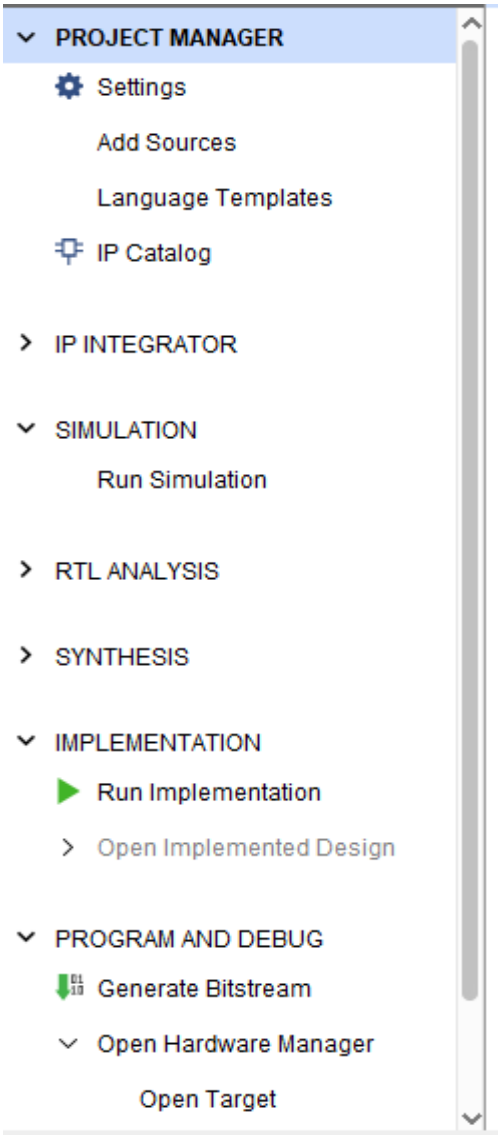


Figure 5.1: FPGA-Based Design Flow

Chapter 6

Design of Robotic Arm

6.1 Mechanical Structure

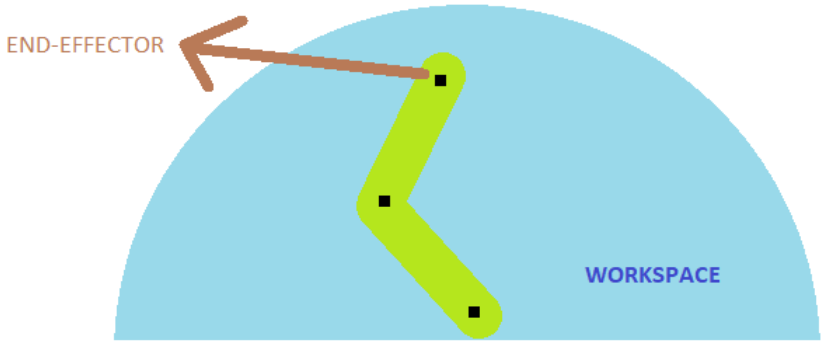


Figure 6.1 2-DOF Robotic Arm Workspace

We had to create a 2 Degree Of Freedom (DOF) Robotic Arm with an additional joint located at the end of the robotic arm, resulting in three joints in total. We chose a 2-DOF because we want that our Robotic Arm should trace the trajectory in a 2-dimensional plane.

All three joints are revolute joints with varying angles and are named as follows:

1. Base Joint - The most-bottom joint , with a range of 180°.
2. Elbow Joint – Its has a range of 180° above the base joint.
3. Gripper Joint - It has a range of 45° at the end effector.

6.2 Construction of Robotic Arm

The Robotic Arm's joints are actuated by SG90 servo motors. These servo motors are inexpensive, light, and produce a fairly exact angle of rotation. There are three servo motors, one for each joint. The Robotic Arm's linkages are comprised of light wooden ice-cream sticks of varying lengths. Because our servo motors can only deliver 1.3 Kg-cm of torque, we picked lightweight linkages.

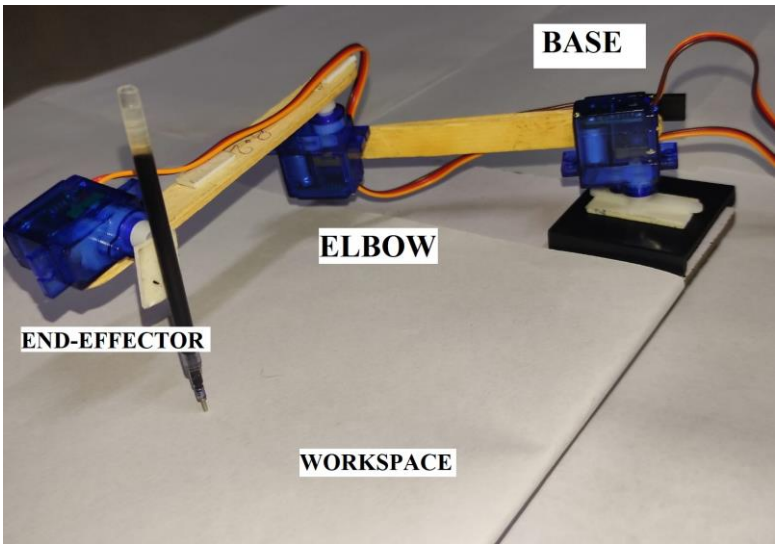


Figure 6.3 Actual Mechanical Structure

The servo motor wires were connected to the FPGA board's Pmod connections via a Pmod CON3 connector and were powered by the FPGA board itself. The connector guaranteed that our servo motors drew only necessary current from the board. However, we could use an external power supply to avoid the risk of overloading of the ports.

Figure 6.4: PMOD CON3 Connector

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

In this project, we have designed an FPGA-controlled Autonomous Robotic Arm movement. While working on the project, the main challenge was to implement the autonomous behavior of the Robotic Arm on FPGA through Verilog, and designing and building the mechanical structure.

Firstly, we wrote MATLAB code to mathematically represent our test pattern, and generate reference values corresponding to the pulse width of the PWM signal. This was then incorporated into Verilog code to initialize Block Memory, and to simulate the desired functionality, that is the servos are synchronized and move in an orderly fashion. In this, we faced some challenges in writing mathematical equations, initializing Block Memory, and synchronizing the gripper motor.

After the successful simulation, we proceeded to build up the mechanical structure of the arm. This was also a challenge for us because we had to do multiple iterations to come up with a stable design.

After building the structure, we implemented the Verilog code on the FPGA board. After implementation, we had do to some calibration of the servo motors, to get an acceptable result.

7.2 Future Scope

- This FPGA-based Robotic Arm, with some additional joints, can be programmed to do any complex task. Such as 3D printing, CNC machining, and product assembling.
- For smoother and more precise movement, analog feedback servo motors can be used.
- To improve precision and increase the range of rotation for joints, high-torque stepper motors could be used in place of low-torque servo motors.

REFERENCES

1. https://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/lab3_19/rom_vivado.html (accessed on 23 April 2023).
2. Brock J. LaMeres, "Quick Start Guide to Verilog,", 1st edi. Springer,2019.
3. *David Lancaster, "Electronics Guidebook",* Why is a servo motor used in a robotic arm? Top 5 reasons - Electronic Guidebook, Accessed on 1 April 2023
4. "Any Silicon", FPGA vs ASIC, What to Choose? - AnySilicon Accessed on 2 April 2023
5. Rohit Singh, "FPGA Vs ASIC: Differences Between Them And Which One To Use?" FPGA vs ASIC: Differences between them and which one to use? | Numato Lab Help Center, Accessed on 10 March 2023
6. Wikipedia, Xilinx Vivado - Wikipedia Accessed on 20 March 2023
7. https://www.xilinx.com Accessed on 16 March 2023
8. Urmila Meshram; Pankaj Bande; P. A. Dwaramwar; R. R. Harkare, "Robot arm controller using FPGA" Robot arm controller using FPGA | IEEE Conference Publication | IEEE Xplore Accessed on 2 April 2023
9. Java T point, Verilog Tutorial - javatpoint, Accessed on 2 April 2023
10. Introduction to High-Level Synthesis with Vivado HLS, VivadoHLS_Overview.pdf (utexas.edu), Accessed on 2 April 2023
11. "Vivado Design Suite 2014.1 Increases Productivity with Automation of UltraFast Design Methodology and OpenCL Hardware Acceleration". *SAN JOSE: Market Watch.*
12. *Maxfield, Clive.* "Free High-Level Synthesis Guide for S/W Engineers". *EE Time*