FPGA BASED ROBOTIC ARM MOVEMENT CONTROL

Minor Project Report

Submitted in partial fulfillment of the requirement for the award of the degree of

Bachelor of Technology

in

Electronics Engineering

by

ZAID AKHTAR

RISHABH VERMA

FAREHA MOHAMMADI

Supervisor

DR. NAUSHAD ALAM

DEPARTMENT OF ELECTRONICS ENGINEERING
ZAKIR HUSAIN COLLEGE OF ENGINEERING & TECHNOLOGY
ALIGARH MUSLIM UNIVERSITY ALIGARH
ALIGARH-202002 (INDIA)

APRIL, 2022

FPGA BASED ROBOTIC ARM MOVEMENT CONTROL

MINOR PROJECT REPORT

Submitted in partial fulfilment of the

requirements for the award of the degree

of

Bachelor of Technology

in

Electronics Engineering

by

ZAID AKHTAR

19ELB146

RISHABH VERMA

19ELB463

FAREHA MOHAMMADI

19ELB464

Supervisor

DR. NAUSHAD ALAM

DEPARTMENT OF ELECTRONICS ENGINEERING
ZAKIR HUSAIN COLLEGE OF ENGINEERING & TECHNOLOGY
ALIGARH MUSLIM UNIVERSITY ALIGARH
ALIGARH-202002 (INDIA)
APRIL, 2022

STUDENTS' DECLARATION

We hereby certify that the work which is being presented in this minor project report entitled **"FPGA Based Robotic Arm Movement Control"** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology and submitted in the Department of Electronics Engineering of the Zakir Husain College of Engineering & Technology, Aligarh Muslim University, Aligarh is an authentic record of our own work carried out during 3rd year of B. Tech. under the guidance of **Dr. Naushad Alam,** Associate Professor, Department of Electronics Engineering, Aligarh Muslim University, Aligarh.

Zaid Akhtar	**Rishabh Verma**	**Fareha Mohammadi**
(19ELB146)	(19ELB463)	(19ELB464)

This is to certify that the above statement made by students is correct to the best of my knowledge.

(Dr. Naushad Alam)
Project Guide

Date: 13 April 2022

ABSTRACT

In this rapidly changing world, automation is becoming an essential part of almost every industry, be it manufacturing or the healthcare field. When we talk about automation, the first thing that comes to our mind is a robot. Robotics and automation are two distinct technologies but these terms are often used interchangeably. Together, they have tremendously transformed the manufacturing space. Pertaining to the manufacturing world, the most commonly used robot is the Robotic Arm. The Robotic Arm can be used in several manufacturing applications ranging from the assembly line to fault detection and packaging.

There are two ways of controlling the robotic arm to perform a particular task. A popular method is to use a microcontroller. A microcontroller based Robotic Arm offers programmability. However, this approach is a bit slower as it has to perform memory read/write operations to execute the instructions. Another approach can be to design a custom ASIC package to implement the Robotic Arm task algorithm. Custom ASIC package offers high speed. Nevertheless, it is costly for low volume production and it takes time to get the chip manufactured. A middle path is to use and FPGA to implement the Robotic Arm controller. FPGA offers programmability and is a quick way of getting the prototype ready. It is also the favoured technology for low volume applications.

We have used FPGA approach in our project. We have done prototyping using Nexys4DDR board equipped with Xilinx Artix-7 series FPGA. In addition, we have used SG90 servo motors for actuating the Robotic Arm. Verilog HDL has been used for programming the FPGA. The simulation and FPGA implementation of the design has been carried out using the Xilinx Vivado tool. We have designed a prototype of a Robotic Arm controller with 4-DOF (Degree of Freedom). The proposed arm is designed to operate manually and automatically to perform a particular action.

****

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to our supervisor Dr.Naushad Alam for the inspiration and encouragement in the successful completion of this work. He has always been cordial, attentive, responsible, and supportive throughout all the highs and lows during our mini-project. He invested a lot of his time in our project, giving valuable input, and took great pains to see us through. We have learned a lot from them and we humbly acknowledge a lifetime of deep gratitude to him. He devoted a lot of time and patience to the reading and correction of this report.

We would also like to extend our gratitude to Syed Mohd Shahbaz Hussain, Ph.D. Scholar, to help us in utilizing the lab facility and providing every technical support for the completion of this project.

Date: 13 April 2022

Place: Aligarh (Zaid Akhtar)

(Rishabh Verma)

(Fareha Mohammadi)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Problem Statement

To create a robotic arm that can execute tasks either manually or semi-automatically using FPGA. Prototyping the algorithm of robotic arm movement on the FPGA board is also a must.

1.2 Proposed Solution

A 4-DOF robotic arm can be built, with servo motors actuating the arm's movement. We may prototype the method on the Nexys4 DDR, which has a Xilinx Artix-7 series FPGA, and write the algorithm in Verilog HDL. The Vivado HLS tool can be used to simulate the findings before they are implemented on FPGA. Onboard switches on the FPGA board can also be utilised to give commands to accomplish a certain operation.

1.3 Methodology

To drive the servo motors of the Robotic Arm, PWM signal of specific time period and specific duty cycle is needed. So, we needed to generate PWM signals of desired specifications through Verilog code. The on-time of the PWM signal was calibrated based on the angle at which the arm had to be moved, and was incorporated in the Verilog code. The PWM signals were given to each servo motors independently and parallely through Pmod connectors of the FPGA board, also a Pmod CON3 connector was used to interface between the Pmod connectors of the FPGA board and wires of servo motors. Distinct PWM signals were given to the different motors using unique combinations of on-board slide switches and pushbuttons.

Chapter 2

Servo Motor Overview

2.1 Background

A servo motor is a type of variable-speed drive that is widely used in industrial production, process automation, and building technology around the world. Although servo motors are not a specific type of motor, they are intended for use in motion control applications that demand high precision positioning, quick reversing, and high performance.

Robotics, radar systems, automated manufacturing systems, machine tools, computers, CNC machines, tracking systems, and other applications all require servo motors.

2.2 Servo motor

Servo motors are DC motors with built-in feedback circuits that allow the shaft locations to be precisely regulated. It's a rotary or linear actuator that can control angular or linear position, velocity, and acceleration precisely. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servo motors.

There are three terminals on these motors: power, ground, and control. PWM waveforms with a period of 20 ms (50 Hz) are typically used to drive the control terminal. The SG90 servo motor is used in this project. The SG90 servo motor is a small and light servo motor with a high output power. The servo can spin 180 degrees (90 degrees in each direction) and functions in the same way as regular servos but in a smaller size.

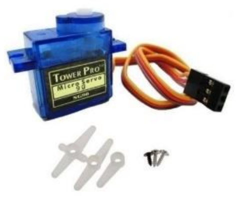
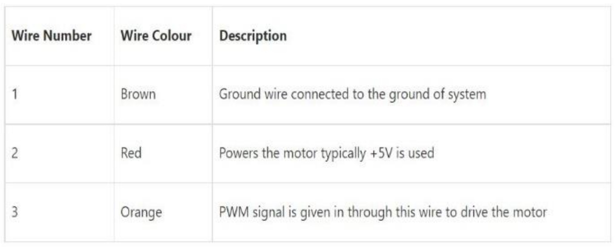


Figure 2.1 SG90 Servo motor



Wire Number	Wire Colour	Description
1	Brown	Ground wire connected to the ground of system
2	Red	Powers the motor typically +5V is used
3	Orange	PWM signal is given in through this wire to drive the motor

Table 2.1 Servo Motor Wire Configuration

2.3 Why Servo Motor?

When it comes to robotics and automation, servo motors are often mentioned. Whatever we want to do with robotics, we'll always be able to locate a servo motor that matches the bill because:

- Servo motors are available in a variety of shapes and sizes.
- Servo motors differ in terms of power and torque. For use in larger applications such as assembly lines, torque is higher in larger servos. Torque is typically reduced in lightweight applications such as packing.
- Companies must use economical and cost-effective choices to keep the cost of these robotic arms down, and servo motors fit the bill well..
- All servo motors require a voltage and PWM (pulse-width-modulation). Servo motors are much lighter than stepper motors since there is no motor control to worry about, and we only need to connect three wires instead of the six that stepper motors require.
- The lighter weight allows robotics and robot arms to move around more freely while maintaining equilibrium.
- Servo motors move very precisely, making them an excellent choice for applications with tight tolerances.
- Servo motors are widely available, with most industrial parts dealers carrying them. This is a significant benefit for companies looking for a solution that will not annoy them when searching for replacement parts.

2.4 Application in Robotics

In robotics, servo motors offer a number of advantages. They're compact, powerful, simple to programme, and precise. Most crucially, they enable near-perfect motion reproducibility. Not all actuators are created equal when it comes to becoming robot components. DC motion solutions are too rapid and low-torque, whereas alternating current engines are too massive. Steppers and servo motors, on the other hand, appear to offer the properties needed to make a robot—consistently high mechanical work output, precision positioning, and small size.

Robotics is one of the most common servo motor applications. Take a look at a simple pick-and-place robot. A "pick and place" robot is a robotic machine that picks up an object from one location and places it in another one (as the name suggests). The motors that actuate the joints in order to pick an object from position A and place it in position B are now servo motors. This is because, in order to execute this operation of picking and placing, we must plan the angular movement of each and every joint.

The robot will continue to conduct its task once this data is sent to the robot controller. The robot's individual motors will receive PWM data from the controller. This allows for exact angular control of the arm, which is impossible to achieve with a traditional DC motor.

2.5 Working

A DC motor, a gear system, a position sensor, and a control circuit make up a servo motor. The DC motors are battery-powered and run at high speeds with low torque. The DC motors' gear and shaft arrangement reduce this speed to a sufficient speed and higher torque. The position sensor detects the shaft's position relative to its fixed position and sends the data to the control circuit. The control circuit decodes the signals from the position sensor, compares the current position of the motors to the intended position, and controls the direction of rotation of the DC motor accordingly to achieve the desired position. A DC supply of 4.8V to 6V is often required for servo motors.

Servos are controlled by transmitting a variable-width electrical pulse across the control line, also known as pulse width modulation (PWM). There are three types of pulses: minimum, maximal, and repetition rate. A servo motor can only rotate 90 degrees in either direction for a total of 180 degrees. The neutral position of the motor is defined as the point at which the servo has the same amount of potential rotation in both clockwise and counter clockwise directions. The position of the shaft is determined by the PWM given to the motor, and the rotor will turn to the appropriate position based on the duration of the pulse sent via the control wire. Every 20 milliseconds (ms), the servo motor expects to see a pulse, and the length of the pulse determines how far the motor turns. A 1.5 ms pulse, for example, will cause the motor to turn 90 degrees. Any time less than 1.5ms turns the servo to 0°, and any time more than 1.5ms turns it to 180°.

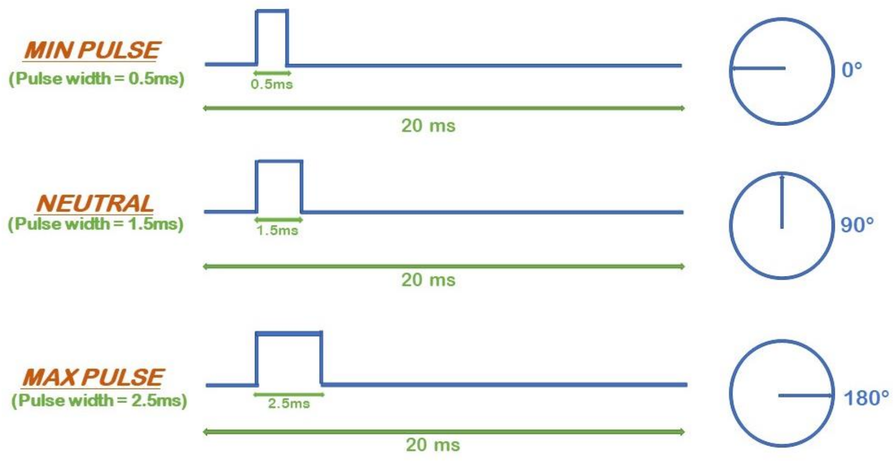


Figure 2.2 Positions of Servo Motor according to the pulse width of input PWM signal

When these servos are told to move, they will move to the specified location and stay there. If an external force is applied to a servo while it is holding a position, the servo will resist moving away from that position. The torque rating of a servo refers to the greatest amount of force it can exert. Servos, on the other hand, do not maintain their position indefinitely; the position pulse must be repeated to instruct the servo to maintain its position.

Chapter 3

Overview of FPGA

3.1 FPGA Intro

The acronym FPGA stands for Field Programmable Gate Array. It is a semiconductor device based on an array of Configurable Logic Blocks (CLBs) connected via programmable interconnects. Most CLBs consist of flip-flops, multiplexers, and adders. The logic blocks are capable of performing simple to complex computational functions. In other words, FPGAs are programmable semiconductor devices that can be programmed for a specific application/functionality.

3.2 Comparisons between the controllers and FPGAs

3.2.1 FPGAs vs Microcontroller

Both FPGA and Microcontroller perform a specific application at a time, and both are embedded within other devices. However, the basic difference is that, if a Microcontroller supports a limited number of peripherals (say, two I2C peripherals) then we cannot extract or utilize more of the same peripheral from the Microcontroller (cannot obtain a third I2C peripheral). Whereas in FPGA, we can program as many peripherals as CLBs on the FPGA permit. Thus, FPGA provides greater customizability than a Microcontroller.

Also in FPGA, instructions are executed parallel, while in Microcontrollers, instructions are executed sequentially in each core. There are also no or fewer abstraction layers on the FPGA for the code to run than a Microcontroller. These features of an FPGA provide faster performance than a Microcontroller.

3.2.2 FPGA vs ASIC

FPGA can be reconfigured with a different design. They even have the capability to reconfigure a part of the chip while keeping the remaining areas as before. Whereas, ASIC has permanent circuitry. Once the application-specific circuit is taped-out into silicon, it cannot be changed. Also, analog designs are not suited for FPGAs while ASICs can have complete analog circuitry.

The cost of development on FPGA is much lower than on ASIC but FPGAs are not suited for high-volume mass production contrary, unlike ASICs' mass production.

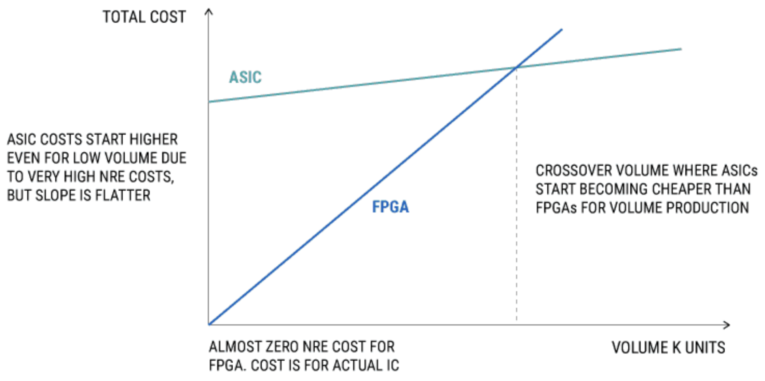


Figure 3.1 FPGA vs ASIC cost analysis

3.3 Why FPGA?

In this project, we have chosen to work in FPGA for the following reasons:-

- FPGAs are preferred for faster hardware prototyping and validating a design or concept. Many ASICs are prototyped using FPGAs themselves to make sure the design is working correctly as intended using FPGA prototyping.
- For the four servo motors to drive independently and simultaneously, parallel distinct outputs are needed from the FPGA board, for which the FPGA is capable.

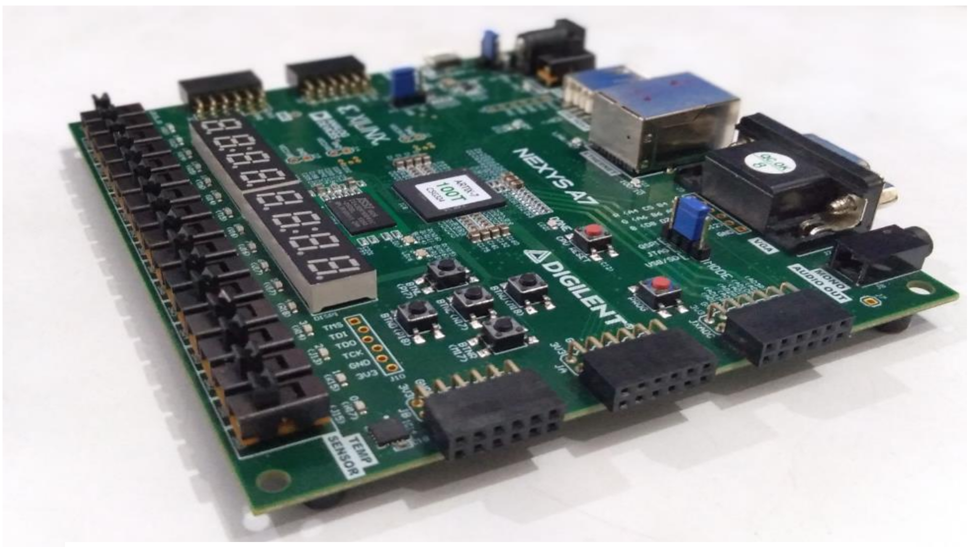


Figure 3.2 Xilinx Nexys-4 DDR Artix-7 FPGA Board

Unlike FPGAs, a microcontroller could not be able to drive parallel outputs on itself.

We have used Nexys4 DDR FPGA board that has Artix-7 (part number XC7A100T-1CSG324C) FPGA. This FPGA board has 15,850 CLBs, each with four 6-input LUTs and 8 flip-flops, internal clock speed exceeding 450MHz. Also, this board offers a greater number of I/O ports and switches which can be used to operate the Robotic Arm with a much more range of output signals

Chapter 4

HDL Implementation of Algorithm

4.1 Hardware Description Language

Moore's Law, which was published in 1970, resulted in a turning point in the world of integrated circuit technology. As a result of the development, developers have been able to create more sophisticated digital and electronic circuits. However, the lack of a superior programming language that allowed both hardware and software codesign was the issue. Development, synthesis, simulation, and debugging of complex digital circuit designs take longer. The introduction of HDLs has aided in the solution of this problem by allowing each module to be worked on by its own team.

The Hardware Description Language (HDL) is a specialized computer language for programming digital and electronic logic devices. HDL can be used to program the circuits' structure, operation, and design. Operators, expressions, statements, inputs, and outputs are all part of the HDL textual description. The HDL compilers give a gate map rather than a computer-executable file. The gate map is then downloaded to the programming device to verify the desired circuit's functionality. It is a good programming language for FPGAs and CPLDs since it helps to define any digital circuit in terms of structural, behavioral, and gate-level.

Verilog and VHDL are the most widely used hardware description languages. They're frequently used in conjunction with FPGAs, which are digital devices meant to make it easier to build customized digital circuits.

4.1.1 VHDL

VHDL (which stands for VHSIC Hardware Description Language) was developed in the early 1980s as a spin-off of a high-speed integrated circuit research project funded by the U.S. Department of Defence. Here VHSIC stands for Very High Speed Integrated Circuits. Researchers working on the VHSIC program were faced with the difficult challenge of describing circuits of tremendous scale (for their time) and managing very big circuit design problems involving several teams of engineers. With just gate-level design tools on hand, it was evident that better, more structured design methodologies and tools would be required.

4.1.1.1 Types of Modelling styles in VHDL

There are 4 types of modelling styles in VHDL:

1. Data flow modelling (Design Equations)

The Boolean expression can be used to define data flow modelling. It depicts the flow of data from input to output. It is based on the concept of concurrent execution.

2. Behavioural modelling (Explains Behaviour)

To execute statements in sequential order, behavioural modelling is utilized. It demonstrates how the system responds to the current statement. Process statements, Sequential statements, Signal assignment statements and wait statements are all examples of behavioural modelling statements.

3. Structural modelling (Connection of submodules)

The functioning and structure of a circuit are specified using structural modelling. Signal declarations, component instances, and port maps in component instances are all part of the structural modelling process.

4.1.1.2 VHDL supports the following features:

- Design methodologies and their features.
- Sequential and concurrent activities.
- Design exchange
- Standardization
- Documentation
- Readability
- Large-scale design
- A wide range of descriptive capabilities

4.1.2 Verilog

Verilog is a hardware description language used for defining digital systems including network switches, microprocessors, memory, and flip-flops. HDL can be used to describe any digital hardware at any level. HDL-described designs are technology-independent, simple to create and debug, and typically more useful than schematics, especially for large circuits.

Verilog was created to make the HDL more robust and adaptable by simplifying the procedure. Verilog is now the most widely used and practiced HDL in the semiconductor industry. HDL was created to help engineers specify desired hardware functionality and have automation tools turn that behaviour into actual hardware parts like combinational gates and sequential logic.

4.1.2.1 Verilog Abstraction Levels

Verilog supports a design at many levels of abstraction, such as

- Behavioural level
- Register-transfer level
- Gate level

1. Behavioural level

Concurrent algorithms behavioural characterize a system at the behavioural level. Every algorithm is sequential, which means it is made up of a series of instructions that are executed one after the other. The major elements are functions, tasks, and blocks. There is no respect for the design's structural execution.

2. Register-transfer level

The Register-Transfer Level specifies the features of a circuit by using operations and data transfer between registers. "Any code that is synthesizable is called RTL code," according to the contemporary definition.

3. Gate level

Within the logical level, logical links, and their time properties describe a system's characteristics. The signals are all distinct. Only definite logical values ('0, 1', 'X, Z') are allowed. Predefined logic primitives are the operations that can be used (basic gates). For logic design, gate-level modelling may not be the best option.

4.1.2.2 Features of Verilog Language

- Verilog is case-sensitive.
- Keywords in Verilog are written in lower case.
- The majority of the syntax in Verilog is derived from the "C" programming language.
- Verilog can be used to model a digital circuit at the Algorithm, RTL, Gate, and Switch level.
- In Verilog, there is no concept of a package.
- Advanced simulation features like TEXTIO, PLI, and UDPs are also supported.

4.1.3 Verilog vs VHDL

Parameters of Comparison	**Verilog**	**VHDL**
Definition	Verilog is a hardware description language used for modeling electronic systems.	VHDL is a hardware description language used to describe digital and mixed-signal systems.
Introduced	Verilog is a newer language as it was introduced in 1984.	VHDL is an older language as it was introduced in 1980.
Language	It is based on the C language.	It is based on Ada and Pascal languages.
Difficulty	Verilog is easier to learn.	VHDL is comparatively harder to learn.
Alphabets	Verilog is Case sensitive.	VHDL is case insensitive.

Table 4.1 Verilog vs VHDL

4.2 Algorithm

As previously stated, the time of the PWM signal sent at the input determines the angle of rotation of a servo motor. Our mission was to control all of the servos independently and from various angles. One servo should not have any effect on another servo.

As a result, based on the demand, the entire code was separated into the following modules:

1. Top Module
2. Angle to on-time convertor (RTL_ROM)
3. N-bit Parallel In Parallel Out (PIPO) shift register.
4. PWM Generator Module

Fig. 4.2 shows the schematic view of the proposed solution. The concept was to use an Angle to the on-time converter to choose a desired input from the input switch and then transfer the matching on time to a single on-time bus. All of the PIPO shift registers shared the on-time bus. The PIPO shift register was used before each PWM generator to update the on-time of the associated PWM generator only when a CLK trigger pulse was present at the input. When we apply the trigger pulse to a specific PIPO shift, the on-time of the associated PWM generator is updated to the value on the on-time bus at the time. Using this method, we can give any angle to any servo without changing the angles of other servos. Also, an enable signal is used just for each servo for the safety purpose to avoid any conflict. In addition, for the sake of safety, an enable signal is used for each servo.

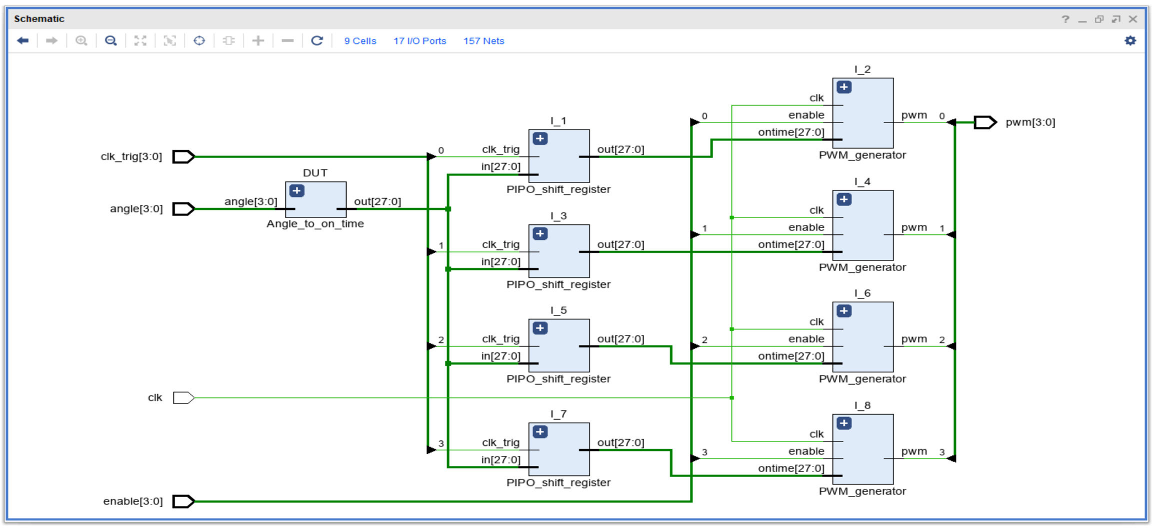


Fig: 4.1 Schematic View of Proposed Model from Vivado HLS

4.3 Verilog Code for PWM generation

- **Top Module:**

```
`timescale 1ns / 1ps

module Top_Module(
input clk,                //Clock signal
input [3:0] enable,          //Enable signal
input [3:0] angle,           //ON/OFF FPGA switch to select angle
input [3:0] clk_trig,        //Trigger switch on FPGA for PIPO
output [3:0] pwm             //PWM output signal
    );

    wire [27:0] t[3:0];       //Output from PIPO
    wire [27:0] ontime_BUS;   //Ontime_bus

    //Module instantiations

    Angle_to_on_time DUT(angle,ontime_BUS);

    PIPO_shift_register I_1( clk_trig[0],ontime_BUS,t[0]);
    PWM_generator  I_2(clk,enable[0],t[0],pwm[0]);

    PIPO_shift_register I_3( clk_trig[1],ontime_BUS,t[1]);
    PWM_generator  I_4(clk,enable[1],t[1],pwm[1]);

    PIPO_shift_register I_5( clk_trig[2],ontime_BUS,t[2]);
    PWM_generator  I_6(clk,enable[2],t[2],pwm[2]);

    PIPO_shift_register I_7( clk_trig[3],ontime_BUS,t[3]);
    PWM_generator  I_8(clk,enable[3],t[3],pwm[3]);

endmodule
```

- **Angle to On-time Convertor:**

```
`timescale 1ns / 1ps

module Angle_to_on_time(
  input [3:0] angle,          //ON/OFF FPGA switch to select angle
  output reg [27:0] out       //Ontime_bus
    );
parameter x=100000;// factor for 1 ms

    always@(angle)
        begin
        case(angle)

        /*Assigning set  of on-time/Angle using different ON/OFF
          Switches for a particular action to be performed*/
        4'b0001: out= 1.0*x; // 45  degree
        4'b0010: out= 1.5*x; // 90  degree
        4'b0100: out= 2.0*x; // 135 degree
        4'b1000: out= 2.5*x; // 180 degree
        default: out= 0;     // default on-time is zero
        endcase

    end
endmodule
```

- **PIPO Shift Register**

```
`timescale 1ns / 1ps

module PIPO_shift_register(
    input clk_trig,         //Trigger switch on FPGA for PIPO
    input [27:0] in,        //Input to PIPO: On-time bus output
    output reg [27:0] out   //Output from PIPO: PWM generator input
    );

    always @(posedge clk_trig)
    out<=in;    //Input is transferred to output on positive clk edge

endmodule
```

- **PWM_Generator**

```
`timescale 1ns / 1ps

module PWM_generator(
    input  clk,         //Clock signal
    input enable,       //Input enable-ON/OFF switch on FPGA
    input [27:0] ontime,//Input on-time: Output From PIPO
    output reg pwm      //PWM output signal
    );

    reg [27:0] n;       //On-time count variable
    reg [27:0] period;  //Period of pulse
    parameter x=100000; // factor for 1 ms

    //Setting initial conditions
    initial
    begin
    n<=0;          //Count=0
    pwm <=1'b1;    //Output signal is Logic High
    period <= 20*x;//20 ms
    end

    //Main PWM code
        always@(posedge clk)
        begin
            if(enable)
            begin
                if(ontime!=0)
                begin
                      n<=n+1;
                      if( n == ontime )
                                pwm <= 1'b0 ;
                      else if (  n == period)
                          begin
                                pwm <= 1'b1 ;
                                n<=0;
                          end
                end
                else
                pwm<=1'b0;

            end

            else
            pwm<=1'b0;

        end
```

```
endmodule
```

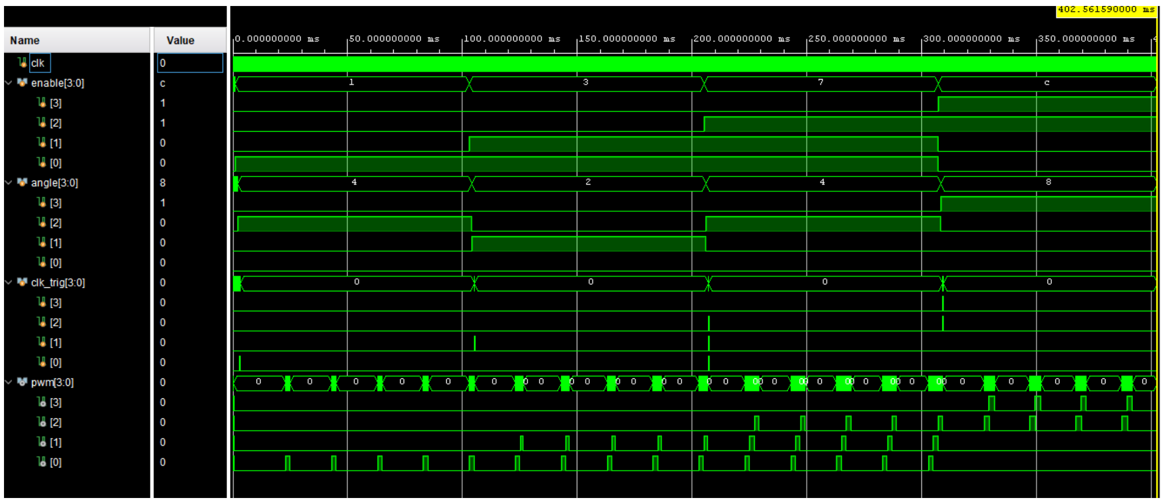
- **Constraints File:**

```
#For using internal clock of FPGA = 100 MHz as input clock signal
set_property -dict{PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports {clk}];

#For using trigger switches of FPGA for generating trigger pulses and
disabling internal clock for the given input clk_trig
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_trig_IBUF[0]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_trig_IBUF[1]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_trig_IBUF[2]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_trig_IBUF[3]]


#Defining all the I/O ports aa Standard LVCMOS33

set_property IOSTANDARD LVCMOS33 [get_ports clk]

set_property IOSTANDARD LVCMOS33 [get_ports {angle[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {angle[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {angle[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {angle[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {clk_trig[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk_trig[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk_trig[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk_trig[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {enable[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {enable[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {enable[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {enable[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {pwm[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {pwm[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {pwm[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {pwm[0]}]

#Configuring ON/OFF switches to assign Angle
set_property PACKAGE_PIN R15 [get_ports {angle[3]}]
set_property PACKAGE_PIN M13 [get_ports {angle[2]}]
set_property PACKAGE_PIN L16 [get_ports {angle[1]}]
set_property PACKAGE_PIN J15 [get_ports {angle[0]}]

#Configuring trigger switches for generating trigger pulse
set_property PACKAGE_PIN P17 [get_ports {clk_trig[3]}]
set_property PACKAGE_PIN P18 [get_ports {clk_trig[2]}]
set_property PACKAGE_PIN M17 [get_ports {clk_trig[1]}]
set_property PACKAGE_PIN M18 [get_ports {clk_trig[0]}]

#Configuring ON/OFF switches for enable
set_property PACKAGE_PIN V10 [get_ports {enable[3]}]
set_property PACKAGE_PIN U11 [get_ports {enable[2]}]
set_property PACKAGE_PIN U12 [get_ports {enable[1]}]
set_property PACKAGE_PIN H6  [get_ports {enable[0]}]

#Configuring Output ports to get PWM signal
set_property PACKAGE_PIN C17 [get_ports {pwm[3]}]
set_property PACKAGE_PIN D18 [get_ports {pwm[2]}]
set_property PACKAGE_PIN E18 [get_ports {pwm[1]}]
set_property PACKAGE_PIN G17 [get_ports {pwm[0]}]

#Configuring clk with internal clock pin E3
set_property PACKAGE_PIN E3  [get_ports clk]
```

- **Test Bench**

```
`timescale 1ns / 1ps

module Top_Module_tb();
reg clk;
reg [3:0] enable;
reg [3:0] angle;
reg [3:0] clk_trig;
wire [3:0]pwm;

Top_Module DUT(clk,enable,angle,clk_trig, pwm);

initial
   begin
   clk=0;enable=0;clk_trig=0;angle=0;
   end

initial

begin
//Enable signal of PWM Gen;    Data on Angle Bus;              Giving ontime to PWM Gen by generating trigger pulse;
#1000000   enable=4'b0001;    #1000000 angle= 4'b0100;  #1000000 clk_trig[0]  =1;            #100000 clk_trig[0]  =0;
#100000000 enable=4'b0011;  #1000000 angle= 4'b0010;  #1000000 clk_trig[1]  =1;            #100000 clk_trig[1]  =0;
#100000000 enable=4'b0111;  #1000000 angle= 4'b0100;  #1000000 clk_trig[2:0]=3'b111;  #100000 clk_trig[2:0]=3'b000;
#100000000 enable=4'b1100;  #1000000 angle= 4'b1000;  #1000000 clk_trig[3:2]=2'b11;    #100000 clk_trig[3:2]=2'b00;
end

always
#5 clk=~clk;            //200 MHz clock frequency same as internal clock of FPGA

initial
#500000000 $finish;

endmodule
```

4.3.1 Simulation Results



Name	Value
clk	0
enable[3:0]	c
[3]	1
[2]	1
[1]	0
[0]	0
angle[3:0]	8
[3]	1
[2]	0
[1]	0
[0]	0
clk_trig[3:0]	0
[3]	0
[2]	0
[1]	0
[0]	0
pwm[3:0]	0
[3]	0
[2]	0
[1]	0
[0]	0

Fig. 4.2 Simulation results of test bench from Vivado HLS tool

4.3.2 Schematic View After Synthesis

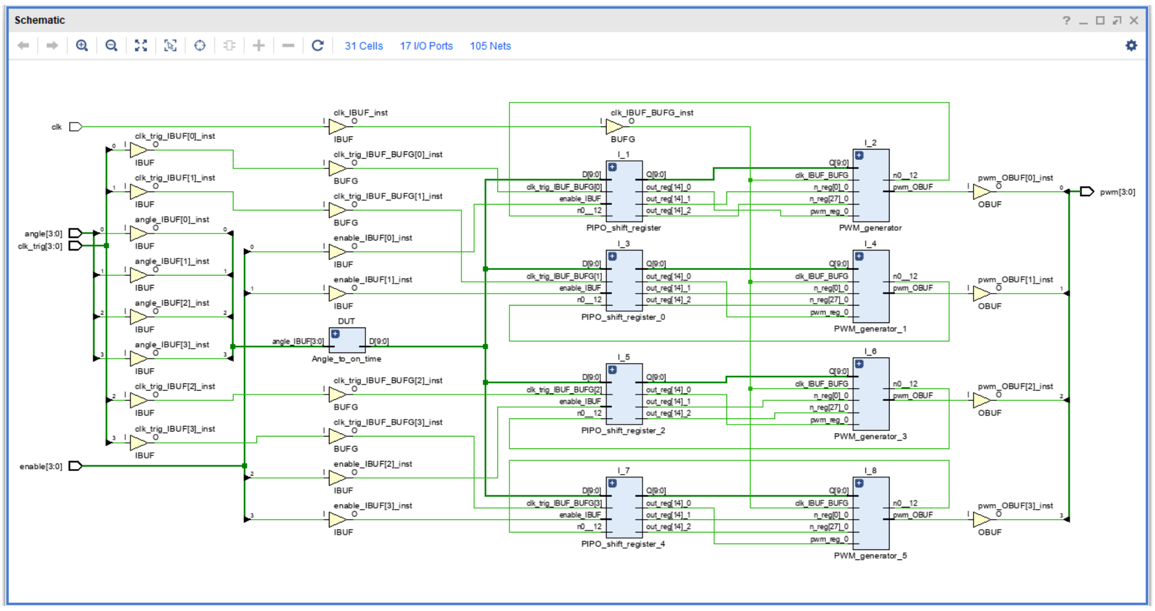


Fig. 4.3 Schematic view after Synthesis and I/O planning

Chapter 5

Vivado Tool Flow

5.1 Introduction

The Vivado Design Suite is a collection of Xilinx tools for designing, programming, synthesis, and analysis of hardware description language (HDL). It replaces Xilinx ISE and adds new functionality for system-on-a-chip development and synthesis. The Xilinx Vivado High-Level Synthesis (HLS) compiler provides a programming environment that is similar to that found on both regular and specialised processors for application development. For the interpretation, analysis, and optimization of C/C++ programs, Vivado HLS shares key technology with processor compilers. The main distinction is in the application's execution target.

Vivado is an integrated design environment (IDE) featuring system-to-IC level capabilities built on a shared scalable data model and a common debug environment that was released in April 2012. Vivado offers ESL design tools for synthesising and testing C-based algorithmic IP; standards-based packaging of both algorithmic and RTL IP for reuse; standards-based IP stitching and systems integration of all types of system building blocks; and block and system verification. The Vivado Webpack Edition is a free version of the design environment that provides designers with a limited version of the design environment.

Vivado HLS enables a software engineer to optimise code for throughput, power, and latency without having to deal with the performance bottleneck of a single memory area and restricted computing resources by targeting an FPGA as the execution fabric. This enables the incorporation of computationally costly software algorithms into genuine goods rather than merely functionality demos. The same categories apply to application code targeted at the Vivado HLS compiler as they do to any other processor compiler. All programs are analysed in terms of operations, conditional statements, loops, and functions using Vivado HLS.

5.2 FPGA Based Design Flow

The complete flow of the FPGA-based design can be seen in the flow navigator. The whole flow is divided into 6 sections such as

1. Adding Sources
2. Simulation
3. RTL Analysis
4. Synthesis
5. Implementation
6. Program and Debug

Explaining each step one by one:

1. Adding Sources

In this section we have to add all the sources required such as the Design source, Simulation source, and constraints. Those are required for the Design, Simulation, and Implementation on FPGA.

2. Simulation

We the design and simulation files are made ready, this section is used to run the simulation. The important thing to remember in this is to keep the test bench as top module before running the simulation.

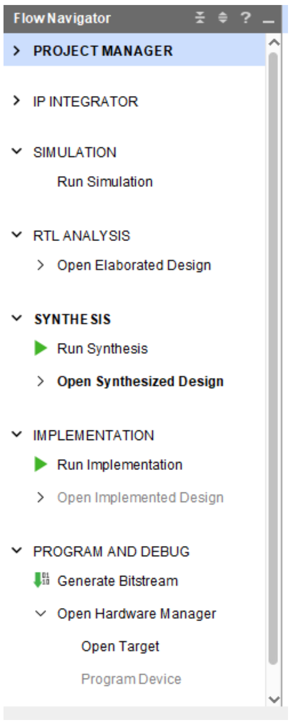


Figure 5.1 Flow Navigator

3. RTL Analysis

After the simulation is done and the design functionality is verified, this section is used to perform RTL analysis. After the analysis is done, we can see the RTL view of our design/code.

4. Synthesis

This is the next step after the RTL analysis, the RTL design is then synthesized after clicking on the run synthesis option. As soon as the synthesis is complete, we can do the I/O planning using the GUI of the VIVADO under the I/O Ports section.

5. **Implementation**

After synthesis and I/O planning, the design is then implemented by clicking the run implementation option.

6. **Program and Debug**

This is the next step after Implementation, in this the first step is to do Bitstream generation. After the Bitstream is generated, we opened the Hardware manager. After this, we connect the FPGA to the Computer using a USB to micro-USB cable. Now, under hardware manager, we clicked the target option and select the FPGA board. As the board is selected, we click on the Program Device option. As the device is programmed, a done signal is generated on the FPGA board, by turning ON an onboard LED.

5.3 Analysis

5.3.1 Utilization Report

The utilization report is generated after Synthesis from Vivado. This report gives a closer view of the resources used, like the number of CLBs engaged, BRAMs, and FIFOs utilised when our Verilog code is synthesised onto the FPGA.

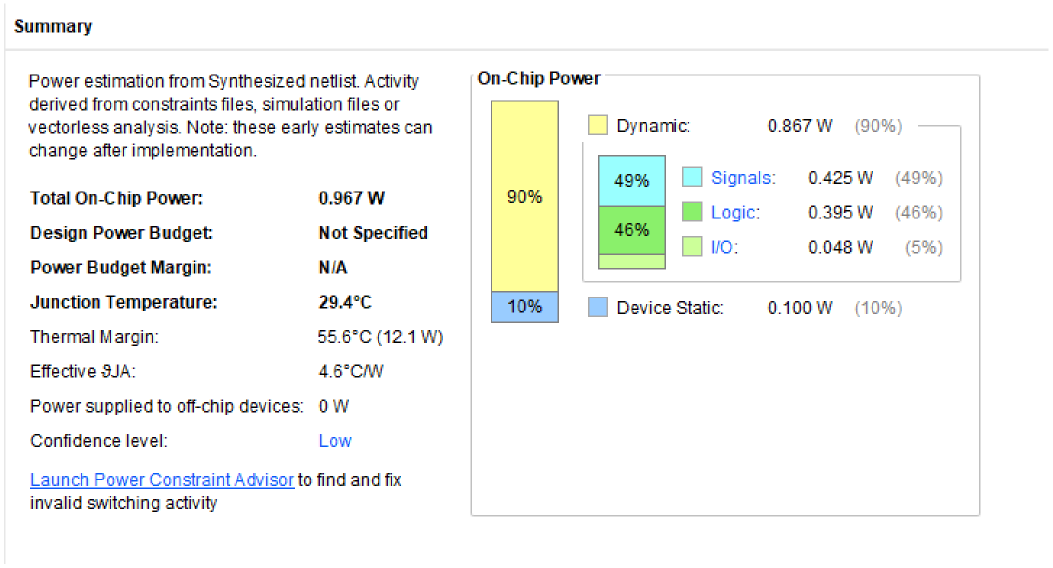
```
Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
-----------------------------------------------------------------------------------------
----------------------
| Tool Version : Vivado v.2020.2 (win64) Build 3064766 Wed Nov 18 09:12:45 MST 2020
| Date         : Wed Apr 13 15:41:33 2022
| Host         : RoboArmFPGA running 64-bit major release  (build 9200)
| Command      : report_utilization -file Top_Module_utilization_synth.rpt -pb
Top_Module_utilization_synth.pb
| Design       : Top_Module
| Device       : 7k70tfbv676-1
| Design State : Synthesized
-----------------------------------------------------------------------------------------
----------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------
```

```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |  101 |     0 |     41000 |  0.25 |
|   LUT as Logic          |  101 |     0 |     41000 |  0.25 |
|   LUT as Memory         |    0 |     0 |     13400 |  0.00 |
| Slice Registers         |  156 |     0 |     82000 |  0.19 |
|   Register as Flip Flop |  156 |     0 |     82000 |  0.19 |
|   Register as Latch     |    0 |     0 |     82000 |  0.00 |
| F7 Muxes                |    0 |     0 |     20500 |  0.00 |
| F8 Muxes                |    0 |     0 |     10250 |  0.00 |
+-------------------------+------+-------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full implementation, is
typically lower. Run opt_design after synthesis, if not already completed, for a more
realistic count.


1.1 Summary of Registers by Type
--------------------------------


+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
| 0     |          Yes |           - |            - |
| 0     |          Yes |           - |          Set |
| 0     |          Yes |           - |        Reset |
| 0     |          Yes |         Set |            - |
| 156   |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+


2. Memory
---------


+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| Block RAM Tile |    0 |     0 |       135 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |       135 |  0.00 |
|   RAMB18       |    0 |     0 |       270 |  0.00 |
+----------------+------+-------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can
accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block
RAM Tile, that tile can still accommodate a RAMB18E1


3. DSP
------


+-----------+------+-------+-----------+-------+
| Site Type | Used | Fixed | Available | Util% |
+-----------+------+-------+-----------+-------+
| DSPs      |    0 |     0 |       240 |  0.00 |
+-----------+------+-------+-----------+-------+


4. IO and GT Specific
---------------------


+-----------------------------+------+-------+-----------+-------+
|          Site Type          | Used | Fixed | Available | Util% |
+-----------------------------+------+-------+-----------+-------+
| Bonded IOB                  |   17 |     0 |       300 |  5.67 |
| Bonded IPADs                |    0 |     0 |        26 |  0.00 |
| Bonded OPADs                |    0 |     0 |        16 |  0.00 |
| PHY_CONTROL                 |    0 |     0 |         6 |  0.00 |
| PHASER_REF                  |    0 |     0 |         6 |  0.00 |
| OUT_FIFO                    |    0 |     0 |        24 |  0.00 |
| IN_FIFO                     |    0 |     0 |        24 |  0.00 |
| IDELAYCTRL                  |    0 |     0 |         6 |  0.00 |
| IBUFDS                      |    0 |     0 |       288 |  0.00 |
| GTXE2_COMMON                |    0 |     0 |         2 |  0.00 |
| GTXE2_CHANNEL               |    0 |     0 |         8 |  0.00 |
| PHASER_OUT/PHASER_OUT_PHY   |    0 |     0 |        24 |  0.00 |
| PHASER_IN/PHASER_IN_PHY     |    0 |     0 |        24 |  0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY |    0 |     0 |       300 |  0.00 |
| ODELAYE2/ODELAYE2_FINEDELAY |    0 |     0 |       100 |  0.00 |
| IBUFDS_GTE2                 |    0 |     0 |         4 |  0.00 |
| ILOGIC                      |    0 |     0 |       300 |  0.00 |
| OLOGIC                      |    0 |     0 |       300 |  0.00 |
+-----------------------------+------+-------+-----------+-------+


5. Clocking
-----------

+------------+------+-------+-----------+-------+
|  Site Type | Used | Fixed | Available | Util% |
+------------+------+-------+-----------+-------+
| BUFGCTRL   |    5 |     0 |        32 | 15.63 |
| BUFIO      |    0 |     0 |        24 |  0.00 |
| MMCME2_ADV |    0 |     0 |         6 |  0.00 |
| PLLE2_ADV  |    0 |     0 |         6 |  0.00 |
| BUFMRCE    |    0 |     0 |        12 |  0.00 |
| BUFHCE     |    0 |     0 |        96 |  0.00 |
| BUFR       |    0 |     0 |        24 |  0.00 |
+------------+------+-------+-----------+-------+


6. Specific Feature
-------------------

+-------------+------+-------+-----------+-------+
|  Site Type  | Used | Fixed | Available | Util% |
+-------------+------+-------+-----------+-------+
| BSCANE2     |    0 |     0 |         4 |  0.00 |
| CAPTUREE2   |    0 |     0 |         1 |  0.00 |
| DNA_PORT    |    0 |     0 |         1 |  0.00 |
| EFUSE_USR   |    0 |     0 |         1 |  0.00 |
| FRAME_ECCE2 |    0 |     0 |         1 |  0.00 |
| ICAPE2      |    0 |     0 |         2 |  0.00 |
| PCIE_2_1    |    0 |     0 |         1 |  0.00 |
| STARTUPE2   |    0 |     0 |         1 |  0.00 |
| XADC        |    0 |     0 |         1 |  0.00 |
+-------------+------+-------+-----------+-------+


7. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| FDRE     |  156 |        Flop & Latch |
| CARRY4   |   40 |          CarryLogic |
| LUT6     |   28 |                 LUT |
| LUT4     |   26 |                 LUT |
| LUT3     |   20 |                 LUT |
| IBUF     |   13 |                  IO |
| LUT5     |   12 |                 LUT |
| LUT2     |   12 |                 LUT |
| LUT1     |   12 |                 LUT |
| BUFG     |    5 |               Clock |
| OBUF     |    4 |                  IO |
+----------+------+---------------------+


8. Black Boxes
```

```
--------------

+----------+------+
| Ref Name | Used |
+----------+------+


9. Instantiated Netlists
------------------------

+----------+------+
| Ref Name | Used |
+----------+------+
```

5.3.2 Power Report

The power report is also generated after the Synthesis in Vivado. This report presents a clear picture of how much our Verilog design is power efficient. Besides providing total power consumption, it also shows dynamic and static power consumption, which could be very helpful for power optimisation in our design.


Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.967 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**29.4°C**
Thermal Margin:	55.6°C (12.1 W)
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	0.867 W	(90%)
Signals:	0.425 W	(49%)
Logic:	0.395 W	(46%)
I/O:	0.048 W	(5%)
Device Static:	0.100 W	(10%)

Figure 5.2 Summary of power estimation from synthesized netlist

Chapter 6

Robotic Arm Design

6.1 Mechanical Design

We needed to design a 3 Degree Of Freedom (DOF) Robotic Arm with one joint at the end effector, making a total of four joints. We chose 3-DOF as we wanted to have our Robotic Arm a spherical workspace. As for picking and placing tasks, a wider workspace region is needed, so that the end effector could reach possibly every coordinate in the spherical region.

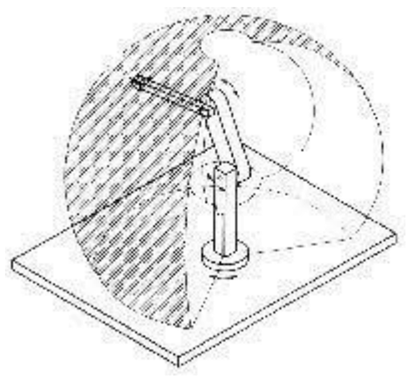


Figure 6.1 DOF Robotic Arm Workspace

All four joints are rotary joints with different range of angles, and are named as :-

1. *Base Joint* - The one at the base, its range of angle is 180°.
2. *Elbow Joint* - Above the base joint, its range of angle is 90°.
3. *Wrist Joint* - Between the Elbow and Gripper, its range of angle is 90°.
4. *Gripper Joint* - At the end effector, its range of angle is 30°.

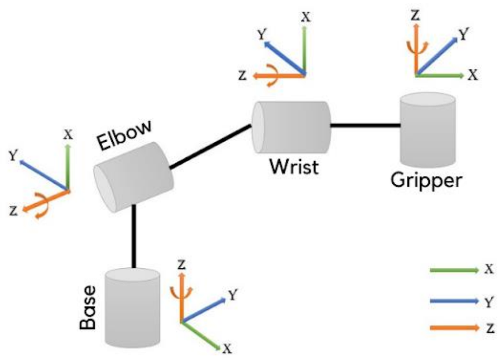


Figure 6.2 Configuration of Joints

6.2 Construction of Robotic Arm

For actuating the joints of the Robotic Arm, SG90 servo motors are used. These servo motors are low cost, low weight and provide pretty precise angle of rotation. Four servo motors are used, one at each joint.

The links of the Robotic Arm are made up of light wooden ice cream sticks, each of appropriate length. These light weighted links are chosen because our servo motors can provide a torque of only upto 1.3 Kg-cm.

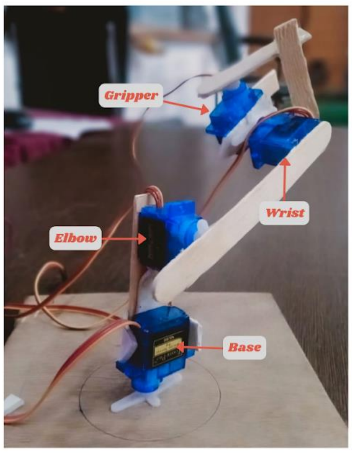


Figure 6.3 Actual Mechanical Design

The wires of the servo motors were interfaced to the Pmod connectors of the FPGA board through a Pmod CON3 connector and were powered externally by four 1.5V cells in series (equivalent to 6V). This connector ensured that our servo motors extracted required current from the external power supply. Thus, keeping the FPGA board safe from high load of the motors.

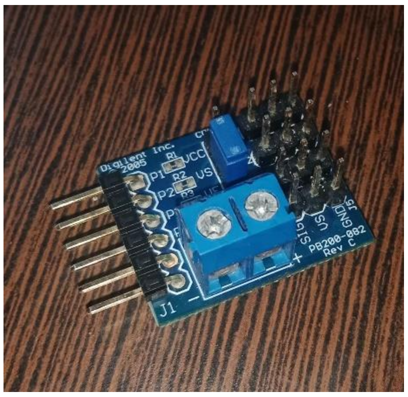


Figure 6.4 Pmod CON3 connector

Chapter 7

Conclusions and Future Scope

7.1 Conclusions

In this project, we have designed a Robotic Arm and controlled it using FPGA. During this project, our major challenges were to design a mechanical structure of the Robotic Arm and implementation of behaviour of the Robotic Arm on FPGA through Verilog.

Firstly, we wrote Verilog code to simulate PWM signals with desired specifications to drive servo motors parallelly and with independent angles. We then proceeded to implement the written Verilog code on the FPGA board. In this, the use of the internal clock of FPGA was a challenge for us.

As soon as we got our behavioural design implemented on the FPGA, we started working on designing the mechanical part of the Robotic Arm. The placement and attachment of servos at the joints of or 4-DOF Robotic Arm was a bit challenging. We had to place the servo motors, and decide the amount of rotation, and direction of rotation considering the mechanical restrictions of the joints of the arm.

7.2 Scope for Future Research

- This FPGA based Robotic Arm can be incorporated with real time object detection implemented on the FPGA itself. The Robotic Arm then could determine the object's position and could calculate the angle of rotation of the servo motors by inverse kinematics to pick the object.
- The low torque servo motors could be replaced by high torque stepper motors to have a much better precision and larger range of rotation for joints. Also, through stepper motors one can have much better control on the speed of rotation of joints.

References

[1]. Moaied Abdelmoniem Rahamta Allah, Ala Eldien Abd Alaziz Mohammed Taha, Tarig Sanhoury, Abo Elgasim Mohammed, Samer Alwaly Ibrahim Idris, Dr. Sami Hassan Omer Salih - "CONTROL OF ROBOTIC ARM OVER FPGA", Sudan University of Science and Technology.

[2]. Jack Erdozain, Nicholas Klugman, Mark Vrablic - " FPGA Robot Arm Assistant"

[3]. https://ieeexplore.ieee.org/document/5164161

[4]. http://www.csjournals.com/IJEE/PDF%202-1/42.pdf

[5]. https://www.ijser.org/researchpaper/FPGA-based-control-system-for-Robotic-Applications.pdf

[6]. https://www.hackster.io/adam-taylor/fpga-soc-controlled-robot-arm-bbdd29

[7]. https://www.jameco.com/Jameco/workshop/Howitworks/how-servo-motors-work.html

[8]. https://electronicguidebook.com/why-is-a-servo-motor-used-in-a-robotic-arm-top-5-reasons/

[9]. https://www.electrical4u.com/servo-motor-applications-in-robotics-solar-tracking-system-etc/

[10]. https://anysilicon.com/fpga-vs-asic-choose/

[11]. https://www.mclpcb.com/blog/fpga-vs-microcontroller

[12]. https://numato.com/blog/differences-between-fpga-and-asics/

[13]. https://www.mepits.com/tutorial/143/vlsi/hardware-description-language

[14]. https://www.allaboutcircuits.com/technical-articles/what-is-a-hardware-description-language-hdl/ https://learn.digilentinc.com/Documents/241

[15]. https://www.doulos.com/knowhow/vhdl/a-brief-history-of-vhdl/

[16]. http://www.uco.es/~ff1mumuj/h_intro.htm

[17]. https://www.javatpoint.com/verilog

[18]. https://www.javatpoint.com/vhdl

[19]. https://askanydifference.com/difference-between-verilog-and-vhdl/

[20]. https://en.wikipedia.org/wiki/Xilinx_Vivado

[21]. http://users.ece.utexas.edu/~gerstl/ee382v_f14/soc/vivado_hls/VivadoHLS_Overview.pdf