GENERATING PWM TYPE SIGNAL WITH VARIABLE DUTYCYCLE ON FPGA

Submitted in partial fulfillment of the requirements for the

Degree of

Bachelors in Engineering

(Electronics)

By

Name	Exam no.
1.) MONISHA GANDHI	**806009**
2.) KOMAL RAINA	**806011**

DEPARTMENT OF ELECTRICAL ENGINEERING

FACULTY OF TECHNOLOGY AND ENGINEERING

THE MAHARAJA SAYAJIRAO UNIVERSITY OF BARODA

GUJARAT, INDIA

MAY 2024

DEPARTMENT OF ELECTRICAL ENGINEERING
FACULTY OF TECHNOLOGY AND ENGINEERIN
THE MAHARAJA SAYAJIRAO UNIVERSITY OF BARODA
GUJARAT, INDIA
MAY 2024

CERTIFICATE

This is to certify that the project report entitled **"GENERATING PWM TYPE SIGNAL WITH VARIABLE DUTY CYCLE ON FPGA" is submitted by Miss Monisha Gandhi and Miss Komal Raina** towards the partial fulfillment of the requirements for the degree of **Bachelors in Engineering (Electronics).** It is the record of work carried out by them under my supervision and guidance. In our opinion, the submitted work has reached a level for being accepted for examination. The results embodied in this project work, to the best of my knowledge, have not been submitted to any other university or institution for award of any degree or diploma.

GUIDE
Mrs. Ami Sen
Assistant Professor
Electrical Engineering Department
Faculty of Technology & Engineering
The Maharaja Sayajirao University of Baroda

HEAD OF DEPARTMENT
Dr. Sorum Kotia
Associate Professor
Electrical Engineering Department
Faculty of Technology & Engineering
The Maharaja Sayajirao University of Baroda

ACKNOWLEDGEMENT

With immense pleasure, we would like to present a project on

"***GENERATING PWM TYPE SIGNAL WITH VARIABLE DUTY CYCLE ON FPGA."***

First of all, we would like to express our deep sense of gratitude to our supervisor Mrs. Ami Sen for her invaluable encouragement, suggestions and meticulous supervision in each and every phase of our work. We would also like to thank Dr. Pramod Modi and Miss. Shyama Gandhi for their throughout support and valuable insights. We would also like to extend our gratitude to our parents and to all those who have guided, encouraged, and supported us on this journey.

Monisha Gandhi

Komal Raina

ABSTRACT

Pulse Width Modulation (PWM) is a widely used technique in digital control systems for achieving variable power output or signal modulation by controlling the duty cycle of a square wave. This paper presents a VHDL implementation of PWM generation on a Field-Programmable Gate Array (FPGA) platform, focusing on duty cycle control.

The proposed design leverages VHDL (VHSIC Hardware Description Language) for its hardware description, facilitating efficient synthesis and implementation on FPGA devices. The PWM generator is designed to produce a square wave with variable duty cycle, allowing precise control over the output signal's characteristics.

Key components of the design include a counter to generate the PWM period, a comparator to compare the counter value with a user-defined duty cycle, and logic for generating the PWM output signal based on the comparator's output. The implementation offers flexibility in duty cycle adjustment, enabling the adaptation of the PWM signal for various applications such as motor control, power regulation, and communication systems.

The VHDL code is synthesized and verified using FPGA synthesis tools, ensuring compatibility with a range of FPGA platforms. Performance metrics including maximum achievable frequency, resource utilization, and power consumption are evaluated to demonstrate the efficiency of the design.

Overall, the proposed VHDL implementation provides a versatile and efficient solution for PWM generation via duty cycle control on FPGA platforms, offering potential benefits in terms of flexibility, reconfigurability, and real-time performance in digital control applications.

Contents

List Of Figures

List Of Tables

CHAPTER 1 INTRODUCTION

1.1 Motivation

As semiconductors increase in density, growing trend toward moving complete systems from the board level to the chip level such as FPGA. In the very early days, modulating currents was made mainly by inserting variable resistor in the path. Obviously, this led to power losses, heavy and extensive 'Controllers' and poor performance. Heat dissipated implies large more expensive components (power semiconductor and heat sinks). How can we reduce this wasted power? One clever method is a technique called pulse width modulation.

Today, thanks to electronic development, there are different means for controlling current with out much power loss and bulky components. One of these methods is the usage of pulse width modulation (PWM) in power electronics to control high energy with maximal efficiency and power saving. Using pulse width modulation (PWM) in power electronics control system is not new, there are different approaches for developing pulse width modulation. Many digital circuits can generate PWM signals, but what is interesting is, to generate pulse width modulation using Hardware Description Language (VHDL) and implementing it in FPGA. FPGA implementation of PWM is selected because FPGA can process information faster, controller architecture can be optimized for space or speed, available in radiation tolerant package, implementation in VHDL allows the targeting of a variety of commercially available device, FPGA allows for implementation of parallel processing.

A field programmable Gate Array (FPGA) is a large Programmable logic device (PLD) that can be used to implement large logic equations as well as arbitrary combinational and sequential logic circuits. Thus, an FPGA can also be used to implement the control- logic design. FPGA are programmable semiconductor devices that are based around a matrix of configurable logic block (CLBs) connected via programmable interconnects as opposed to ASICs where the device is custom built for a particular design. FPGA can be programmed to the desired application or functionality requirements. VHDL is a language that is used to describe the behavior of digital circuit designs. It is VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

and now used extensively by industry and academia for the purpose of simulating and synthesizing digital circuit design. Its designs can be simulated and translated into a form suitable for hardware implementation. VHDL is rapidly being embraced as the universal communication medium of design FPGA vendors, and ASIC vendors thorough the industry is standardizing on VHDL as input and output from their tools.

In this thesis, VHDL modeling is used to generate the PWM signal and XILINX VIVADO Design Software is used for the design of PWM. And also XILINX FPGA ARTIX-7 family is used for the process, the VHDL language is exhaustively used for PWM generation and the language's capability is explored. The PWM developed in this way can be integrated into complete system where PWM is required,as in applications requiring motor and motion control, robotics etc. Therefore, most industries and others involve PWM in their control system benefited from this thesis.

1.2 Objectives of the thesis

Getting Field Programmable Gate Array based PWM controller is the choice of every controller designer, because of the design flexibility, design complexity, the gate capacity, reconfigure ability, design time, speed etc. and this paper describes the development of PWM in Xilinx FPGA. Many digital and Transistor Logic circuit (such as microprocessor ,microcontroller etc.) can develop PWM, but what is interesting is to design the PWM using the latest Programmable device so as to use the features of FPGA.

1.2.1 General objective

The general objective of this thesis is to Design PWM using XILINX VIVADO software.

1.2.2 Specific objective

Specifically thesis is very important in the area of PWM generation systems

To Design PWM using VHDL modeling

To study the architectural features of XILINX FPGAs

To explore these features using HDL (VHDL)

Synthesizing and simulating the design using Xilinx VIVADO simulator.

Implementing the design in FPGA and realizing the design

Controlling the Speed of DC Motor For Actual Hardware Implementation via Generated PWM type Signal

1.3 Thesis Organization

The Design of Pulse Width Modulation in Xilinx Field Programmable Gate Array has a great role in the control system where the pulse width modulation is used. This thesis contains the total of six chapters.

Chapter one is general introduction of a thesis, which highlights the basic idea and gives an insight or a direction to the body of the thesis.

Chapter two focuses on the introduction of pulse width modulation systems, the advantages and the application of it.

Chapter three tells about Programmable Logic Devices and VHDL. This gives an overview of how and where programmable logic devices are used. It gives a brief history of the programmable logic devices and goes on to describe the different ways of designing with PLDs. And also discuses the architecture of the Xilinx FPGA. In addition to this, it explores the VHDL Modeling.

Chapter four deals about the design of PWM in Xilinx FPGA, and describes the functional description of the design PWM and the FPGA design flow is explored. This chapter also describes the VHDL modeling of the design PWM in Xilinx Integrated System Environment.

Chapter five develops the Synthesis and simulation results of the VHDL design of PWM in Xilinx Field Programmable Gate Array and discuses the result of simulation and synthesis. It describes conclusion on what is done in this project and suggest the recommendations for the future work.

CHAPTER 2 WORKING AND FUNCTIONAL DESCRIPTION

2.1 PULSE WIDTH MODULATION

Pulse width modulation (PWM) is a technique to provide a logic “1” and logic “0” for a Controlled period of time. It is a signal source involves the modulation of its duty cycle to control The amount of power sent to a load .The following sections describe the design of Pulse Width Modulation (PWM) on a Xilinx FPGA using very high speed integrated circuit hardware Description language (VHDL). The PWM generates pulses on its output. The pulses are made in Such a way that the average value of highs and lows is proportional to the PWM input. By Filtering the pulses, we obtain an analog value proportional to the PWM input. A PWM input can Be of any width. Most common values are 8-bits and 16-bits. The PWM developed can be used in many diverse and complex applications like robotics, motor and motion control.

2.2 PWM as Digital Technology

There are two approaches for implementing control systems using digital technology The First approach is based on software which implies a memory-processor interaction. The memory holds the application program while the processor fetches, decodes, and executes the program Instructions. Programmable Logic Controllers (PLCs), microcontrollers, microprocessors, Digital Signal Processors (DSPs), and general purpose computers are tools for software implementation On the other hand, the second approach is based on hardware. Early hardware implementation is achieved by magnetic relays extensively used in old industry automation systems. It then became achievable by means of digital logic gates and Medium Scale Integration (MSI) components. When the system size and complexity increases, Application Specific Integrated Circuits (ASICs)are utilized . The ASIC must be fabricated on a manufacturing line, a process that takes several months, before it can be used or even tested.

FPGAs are configurable ICs and used to implement logic functions. Early generations of FPGAs were most often used as glue logic which is the logic needed to connect the

major components of a system. They were often used in prototypes because they could be programmed and inserted into a board in a few minutes, but they did not always make it into the final product. Today's high-end FPGAs can hold several millions gates and have some significant advantages over ASICs. They ensure ease of design, lower development costs, more product revenue, and the opportunity to speed products to market. At the same time they are superior to software-based controllers as they are more compact, power efficient, while adding high speed capabilities. The target FPGA device used in this paper is Basys-3 manufactured recently by Xilinx .Digital controllers usually encompass Input/output (I/O) modules to communicate with users. Embedded systems typically consist of both application-specific hardware and a general purpose microprocessor or microcontroller. Many of the functions performed by the system can be implemented either on dedicated hardware (for example, in an FPGA, in an ASIC, or in a special-purpose function block added to the Microcontroller itself) or can be implemented by the microprocessor in software.

The decision to implement a function in hardware or software depends on trade-offs between the hardware/software implementations like cost, speed, power consumption, design time, size (silicon area or program size), risk and others. Partitioning functions efficiently between hardware and software can be key to timely design of high performance, low cost digital systems. The speed of a DC motor is approximately proportional to the supply voltage, so reducing the supply voltage by half will reduce the speed by approximately one-half. The speed of the motor can therefore be controlled by varying the average supply voltage. The supply voltage could be changed using a variable supply voltage source, but this technique is inefficient since voltage is controlled in these cases through a voltage drop across a transistor. Since all current must go through the transistor and P=VI (the drop across the transistor times the current), a significant amount of power is lost at the transistor. A better way to control the motor is to switch the motor's supply on and off very quickly. If the on time is equal to the off time, the average voltage seen by the motor will equal half the supply voltage and the motor will run at half the maximum speed. As the on time increases compared with the off time, the average speed of the motor will increase. The user should not notice the motor turning on and off, because it is done very quickly. A pulse-width modulated (PWM)

signal is a constant period square wave with a varying duty cycle (on-time compared to off-time).In other words, the frequency of a PWM signal is constant but the time the signal remains high varies as shown in Figure 2.1 The duty cycle (percent on time) is given by τ/T.

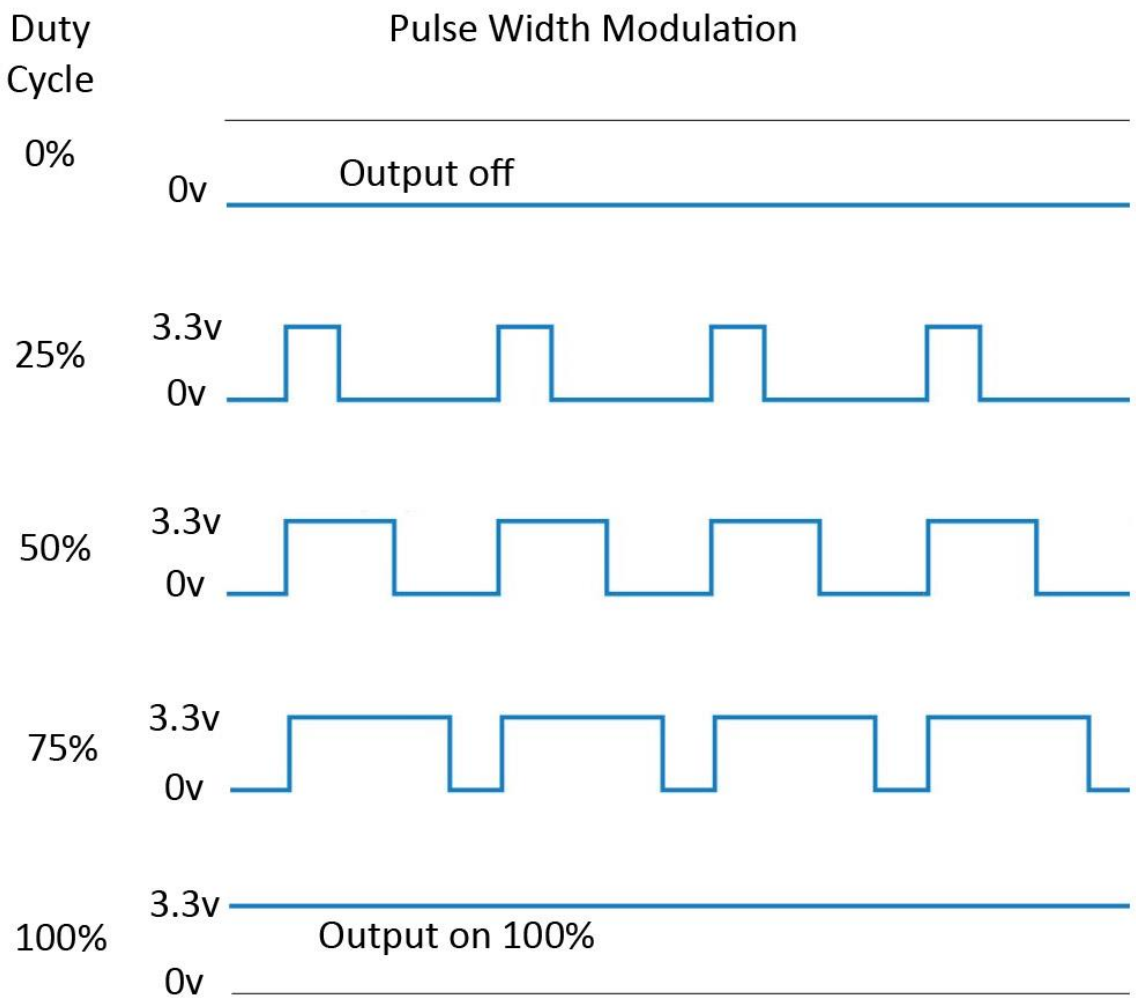

Figure 2. 1 Different duty cycles

2.2.1 Components Used

Software Required - VIVADO Software Version 2023.1

Hardware Required

ARTIX -7 Basys 3 FPGA Board

Digital Oscilloscope

Set of Connecting wires

Armature DC Motor

Driver

Power supply

For hardware implementation of PWM we will control DC motor as mentioned earlier, so we are using MOSFET here as a switching device. The Detail of components are given in the table below:

Table 1 Details of components

Driver	ULN2003APG
Resistor	10Kohm
DC External Supply for BASYS 3 FPGA board	5V

DC motor is a machine that converts electrical energy of direct current into mechanical energy. In a DC motor, the input electrical energy is direct current which is converted into mechanical rotation It is based on electromagnetic induction, where a conductor carrying current (normally a coil of wire) placed in a magnetic field experiences force to rotate. This rotation is used to perform mechanical work. DC Motor Speed The formula for DC Motor speed is given as:

$$N = KEb/\Phi$$

where, N is speed in RPM K is constant proportionality which is equal to 60A/ZP

2.2.2 Working and Functional Description

A PWM circuit works by making a pulsating DC square wave with a variable on to-off ratio. The average on time may be varied from 0 to 100 percent. The widths of the pulses are proportional to the input signal. When the signal is small, a series of narrow pulses is generated. When the signal is large, a series of wide pulses is generated. In many of the applications, the single bit digital output is subject to a low-pass filter those results in an analog output level. The output level is the analog equivalent of the digital PWM's duty-cycle.

PWM can be used to reduce the total amount of power delivered to a load without losses normally incurred when a power source is limited by resistive means. This is because the average power delivered is proportional to the modulation duty cycle. High frequency PWM Power control systems are realizable with semiconductor switches. The discrete on/off states of the modulation are used to control the state of the switch(es) which correspondingly control the voltage across or current through the load. The major advantage of this system is the switches are either off and not conducting any current, or on and have (ideally) no voltage drop across them. The product of the current and the voltage at any given time defines the power dissipated by the switch, thus (ideally) no power is dissipated by the switch .Realistically, semiconductor switches such as MOSFETs or BJTs are non ideal switches, but high efficiency controllers can still built.

Functional Block Diagram of PWM:

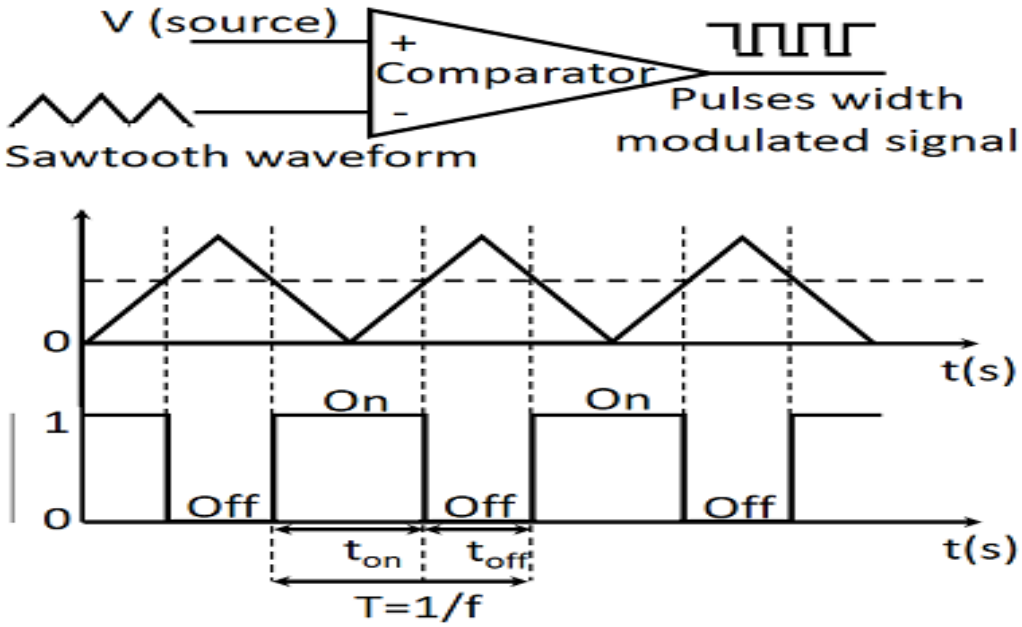

Figure 2. 2 Functional Description of PWM

The Comparator Block is shown below:

It is observed that the 1st input is duty cycle of 8 bit and 2nd input is sawtooth waveform generated by counter.

And 2nd input applied to the comparator and its output is our PWM signal.

A simple method is used where a sawtooth waveform(Carrier signal) and Variable Duty Cycle is required. A comparator compares between two values in order to generate pulse width modulation signal.

Reset Logic: When the counter reaches its maximum value (end of the PWM period), reset it back to zero to start the next period.To control the duty cycle of the PWM, value will be used to set the high duration of the PWM signal.

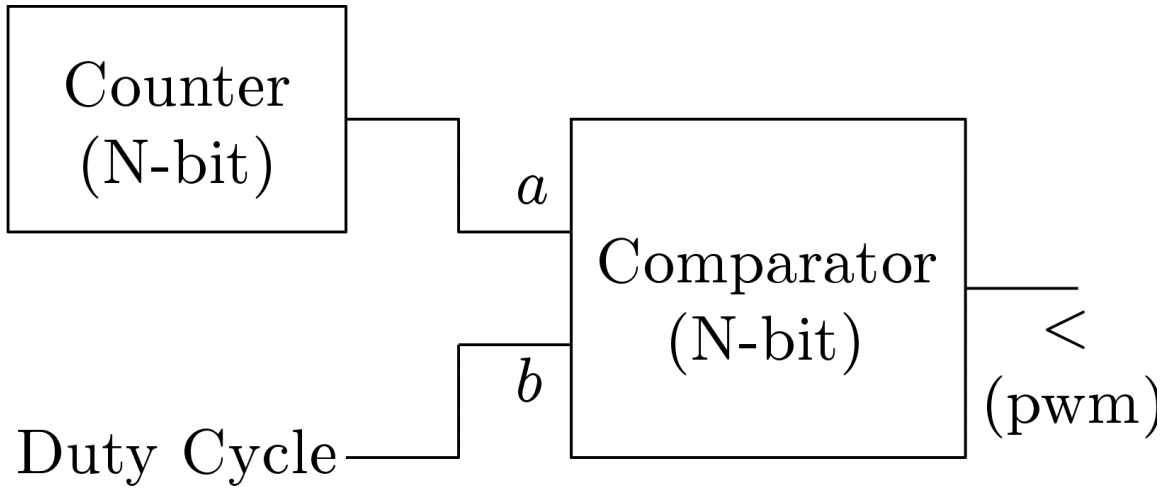

Figure 2. 3 Comparison block of PWM

Comparator compares between duty cycle and sawtooth waveform,

if sawtooth waveform< duty cycle , the output register = 1 then PWM = 1;

If sawtooth waveform> duty cycle , the output register = 0 then PWM = 0;

CHAPTER 3 IMPLEMENTATION DETAILS

3.1 VHDL - VHSIC Hardware Description Language

Very High Speed Integrated Circuits (VHSIC) program

VHDL is designed to fill a number of needs in the design process. Firstly, it allows description of the structure of a design that is how it is decomposed into sub designs, and how those sub-designs are interconnected. Secondly, it allows the specification of the function of designs using familiar programming language forms. Thirdly, as a result, it allows a design to be simulated before being manufactured, so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping.

3.1.1 Design Units

One concept unique to VHDL (when compared to software programming languages and to Verilog HDL) is the concept of a "design unit". Design units (which may also be referred to as "library units") are segments of VHDL code that can be compiled separately and stored in a library. You have been introduced to two design units already: the entity and the architecture. There are actually five types of design units in VHDL: entities, architectures, packages, package bodies, and configurations.

Entities

A VHDL entity is a statement (identified by the entity keyword) that defines the external specification of a circuit or sub-circuit. The minimum VHDL design description must include at least one entity and one corresponding architecture. When you write an entity declaration, you must provide a unique name for that entity and a port list defining the input and output ports of the circuit. Each port in the port list must be given a name, direction (or "mode", in VHDL jargon) and a type. Optionally, you may also include a special type of parameter list (called a generic list) that allows you to pass additional information into an entity.

Architectures

A VHDL architecture declaration is a statement (beginning with the architecture keyword) that describes the underlying function and/or structure of a circuit. Each architecture in your design must be associated (or bound) by name with one entity in the design. VHDL allows you to create more than one alternate architecture for each entity. This feature is particularly useful for simulation and for project team environments in which the design of the system interfaces (expressed as entities) is done by a different engineer than the lower-level architectural description of each component circuit. An architecture declaration consists of zero or more declarations (of items such as intermediate signals, components that will be referenced in the architecture, local functions and procedures, and constants) followed by a begin statement, a series of concurrent statements, and an end statement.

Packages and Package Bodies

A VHDL package declaration is identified by the package keyword, and is used to collect commonly-used declarations for use globally among different design units. You can think of a package as a common storage area, one used to store such things as type declarations, constants, and global subprograms. Items defined within a package can be made visible to any other design unit in the complete VHDL design, and they can be compiled into libraries for later re-use. A 33 package can consist of two basic parts: a package declaration and an optional package body. Package declarations can contain the following types of statements:

type and subtype declarations

constant declarations

VHDL supports many possible styles of design description. These styles differ primarily in how closely they relate to the underlying hardware. The different styles of VHDL refer to the differing levels of abstraction possible using the language -- behaviour, dataflow, and structure.

In this work, I have used Behaviour modeling.

Behaviour

The highest level of abstraction supported in VHDL is called the behavior level of abstraction. When creating a behavioral description of a circuit, you will describe your circuit in terms of its operation over time. The concept of time is the critical distinction between behavioral descriptions of circuits and lower-level descriptions (specifically descriptions created at the dataflow level of abstraction). In a behavioral description, the concept of time may be expressed precisely, with actual delays between related events (such as the propagation delays within gates and on wires), or it may simply be an ordering of operations that are expressed sequentially (such as in a functional description of a flip-flop). When you are writing VHDL for input to synthesis tools, you may use behavioral statements to imply that there are registers in your circuit. It is unlikely, however, that your synthesis tool will be capable of creating precisely the same behavior in actual circuitry as you have defined in the language.

3.2 Software description

To design the PWM in Field programmable gate array first the functional description of the design modeled in very high speed integrated circuit HDL using the behavioral abstraction leveland this VHDL code is synthesized and simulated using Xilinx Synthesis and simulation tool. After successfully synthesized and simulated the design it can be downloaded to the targeting device (FPGA). These chapter discuses about how the functional description generates and this functional description modeled in VHDL to generate the pulse width modulation. In the process Xilinx ISE and FPGA design flow is used. This chapter discuses about the design of pulse width modulation in Xilinx FPGA.

3.2.1 Xilinx VIVADO Design Suite

VIVADO is the design software for AMD adaptive SoCs and FPGAs. It includes: Design Entry, Synthesis, Place and Route, Verification/Simulation tools.

Design entry is the first step in design flow. During design entry, source files are created based on design objectives. We can create the top-level design file using a Hardware Description Language (HDL), such as VHDL.

After synthesis, we run design implementation, which converts the logical design into a physical file format that can be downloaded to the selected target device.

Implementation processes vary depending on targeting a Field Programmable Gate Array (FPGA).

We can verify the functionality of the design at several points in the design flow. we can use simulator software to verify the functionality and timing of the design or a portion of the design.

After generating a programming file, we can configure the device. During configuration, we generate configuration files and download the programming files from a host computer to a Xilinx device.

3.3 Flow Chart

To create a flowchart algorithm for the VHDL code, let's break down the process into steps and represent them in a structured flowchart format:

Start: Begin the algorithm.

Initialize Variables: Set initial values for variables count, duty cycle, and i.

Process Clock Signal: Check for a rising edge of the clock signal.

If a rising edge is detected:

Increment the count variable.

Check if count is equal to duty_cycle

If true, set pwm_output to '1'.

Check if count is equal to 500.

If true, set pwm_output to '0' and reset count to 0.

If false, continue without changing pwm_output.

Check for conditions to increment or decrement duty_cycle.

If the increment signal is active (increment = '1') and

duty_cycle is less than 450, increment duty_cycle.

the decrement signal is active (decrement = '1') and

duty_cycle is greater than 50, decrement duty_cycle.

End Process: End the algorithm.

The flow chart:

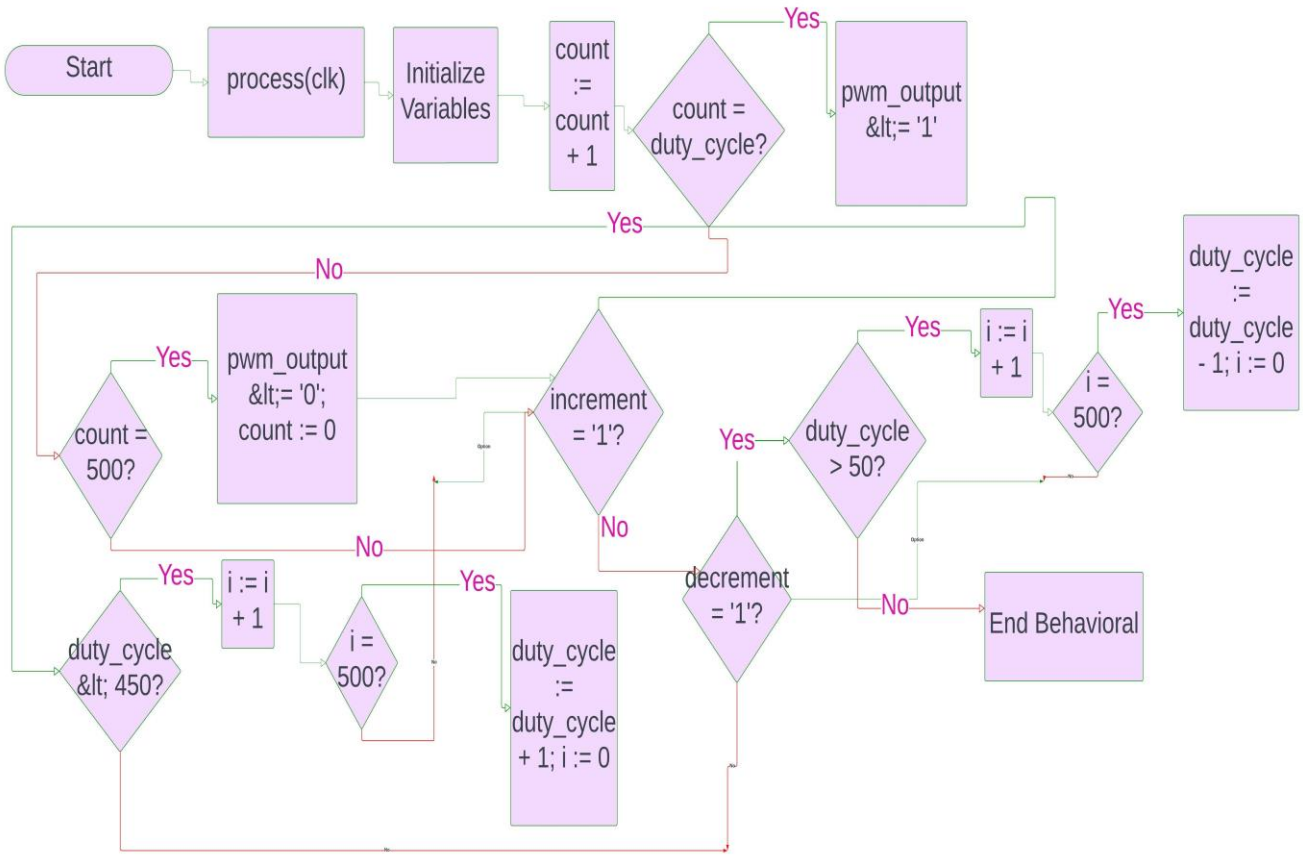

Figure 3. 1 Flow graph of VIVADO software implementation

In this flow graph, we can change the switching frequency by changing the value of count which is here 500.

3.4 Interfacing details

A PWM signal can be used to switch the motor on and off. PWM is almost noise free gives high efficiency and flexibility and quick response also it has low cost features.

3.4.1 Speed control of DC Motor using PWM

The PWM signal Is applied to the transistor pair, which acts as a switch. Whenever the PWM signal is high, the switch is closed and the entire supply voltage is applied across the motor terminals. When the PWM signal is low, the switch is open and the supply voltage across the motor is 0 volts.

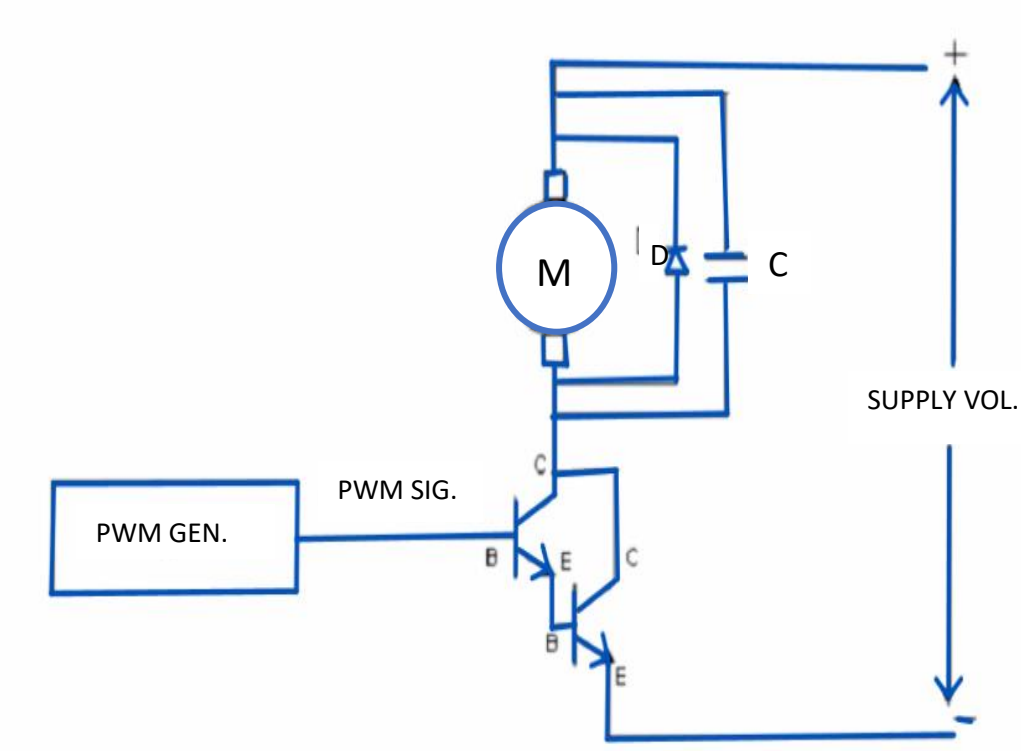

Figure 3. 2 Interfacing with DC Motor

If we apply a PWM signal with a 50 % duty cycle then the average voltage across the motor is 50%. It does not take much work to show that the average voltage across the motor is given by:

V motor, average = V supply*duty cycle.

Therefore, motor speed = (motor speed when driven by V supply) * duty cycle. A PWM signal can be generated in a number of ways. One using software implemented by the microcontroller and the other using hardware implemented in an FPGA. A software implementation may also be lower risk than a hardware implementation because it may be changed at the last minute by changing the program in memory (changing the program, though, also increases the risk of undetected software bugs). However, the

capability of a microcontroller to perform these tasks may be limited. The microprocessor may have to perform several tasks in addition to implementing the PWM and thus may not be able to implement the PWM fast enough or with enough accuracy. implementing the PWM in hardware frees up the microprocessor for these other tasks and ensures that the task is performed quickly and accurately. The best choice – hardware or software – depends significantly on the application.

CHAPTER 4 - SIMULATION AND TESTING

4.1 Testbench

At this point, the sample circuit is complete and ready to be processed by synthesis tools. Before processing the design, however, you should take the time to verify that it actually does what it is intended to do. You should run a simulation. Simulating a circuit such as this one requires that you provide more than just the design description itself. To verify the proper operation of the circuit over time in response to input stimulus, you will need to write a test bench. The easiest way to understand the concept of a test bench is to think of it as a virtual tester circuit. This tester circuit, which you will describe in VHDL, applies stimulus to your design description and (optionally) verifies that the simulated circuit does what it is intended to do. To apply stimulus to your design, your test bench will probably be written using one or more sequential processes, andit will use a series of signal assignments and wait statements to describe the actual stimulus. You will probably use VHDL's looping features to simplify the description of repetitive stimulus (such as the system clock), and you may also use VHDL's file and record features to applystimulus in the form of test vectors.

In addition VHDL has got the following properties

VHDL is not case sensitive.

The semicolon is used to indicate termination of a statement.

Two dashes ('—') are used to indicate the start of a comment.

Identifiers must begin with a letter, subsequent characters must be alphanumeric or '_' (underscore).

TEST BENCH SIMULATION :

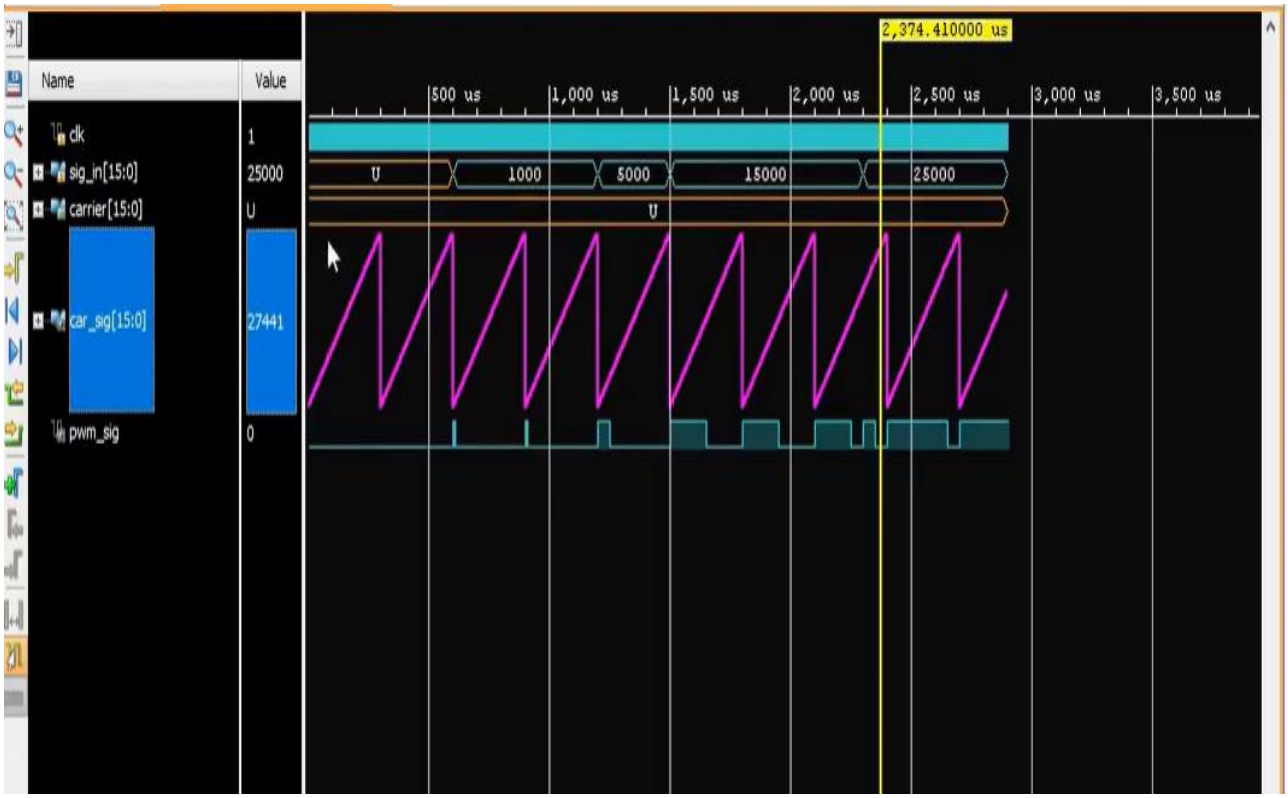

Figure 4. 1 Testbench

4.1.2 Obstacles

Test bench runs completely However for constrains file and VHDL file we faced issues that I have mentioned below: Counter had been taken as SIGNAL so during simulation individuals need to change its clock force value in VIVADO by themselves , so for solution we found by using the instruction namely: WAIT FOR (mentioned time) and we got our PWM pulse varying by itself… BUT by using this instruction our carrier signal started to varying instead of our input signal and how to make constrain for that we do not know!! We searched a lot and finally we analyze that we need to take counter as a variable instead of signal. For variable duty cycle we need to apply push button or switches for increment or decrement the value and in program it cannot be done automatically we need to change our input.

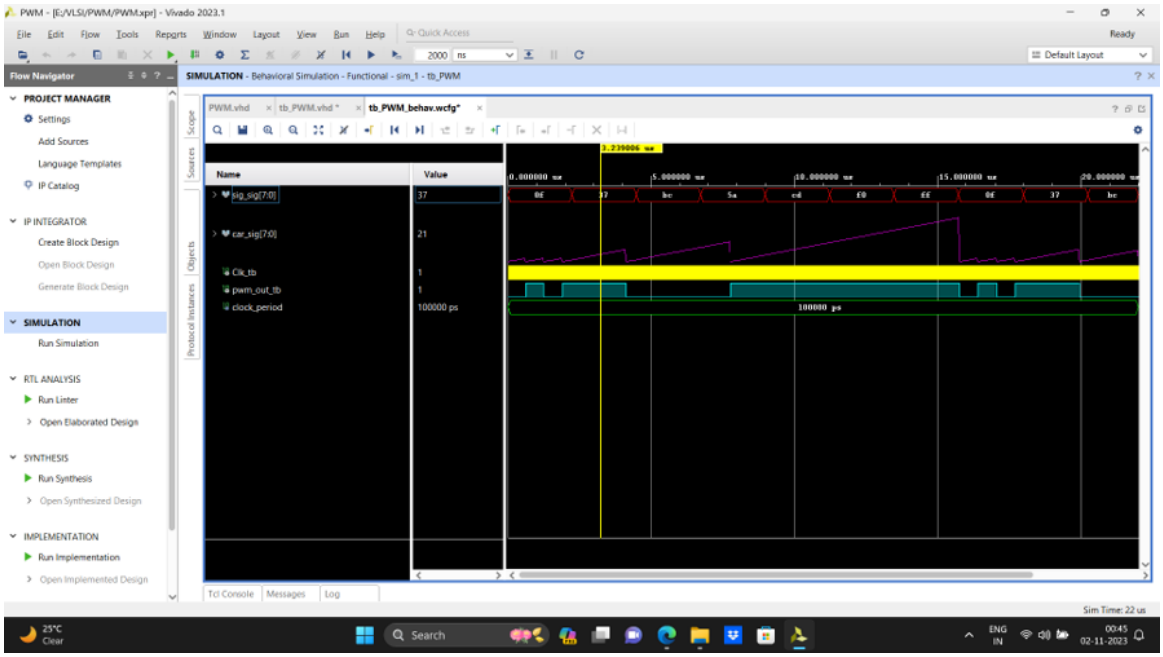

Figure 4. 2 carrier changing

4.2 Procedure on VIVADO to simulate

Creating a VIVADO project and generating a bitstream involves several steps. Here's a general outline:

Step:1 Launch VIVADO: Open VIVADO Design Suite.

Step:2 Create a New Project: Click on "Create Project" from the welcome page or navigate to File > Project > New. Follow the wizard to specify project name, location, and select the target FPGA device. You might also choose a project directory and specify whether you want to add existing source files.

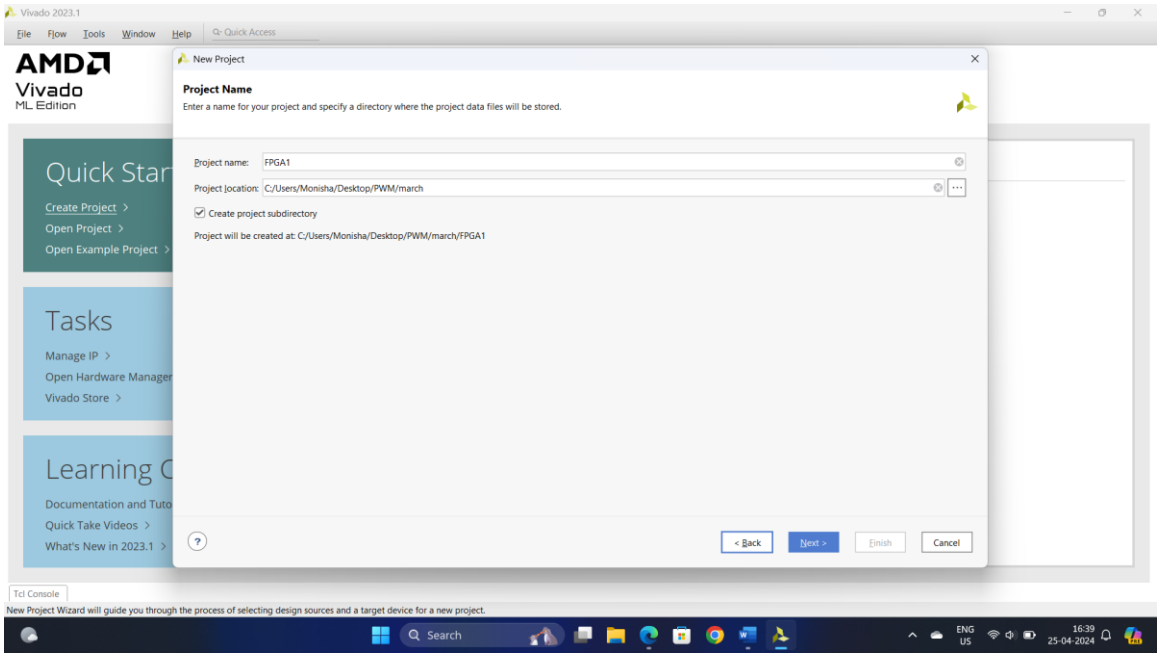

Figure 4. 3 Step 1 of creating project on VIVADO

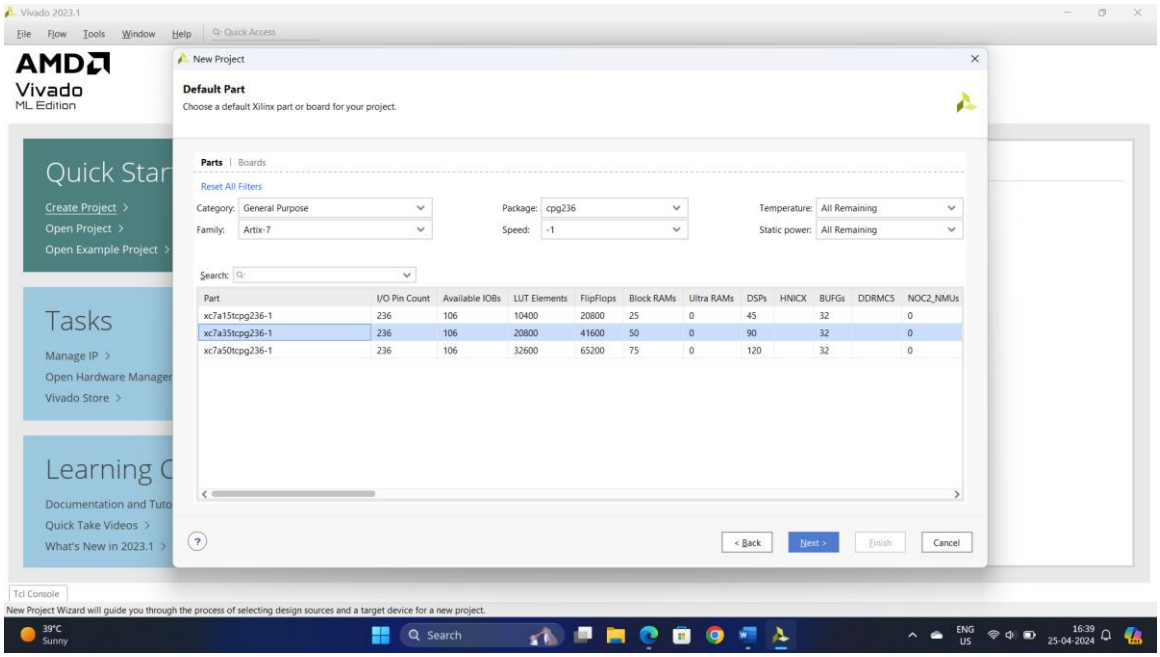

Figure 4. 4 adding details regarding FPGA board

Step:3 Add Design Sources: Once the project is created, add your design sources like Verilog, VHDL, or block diagrams. Navigate to Flow Navigator > Project Manager > Add Sources.

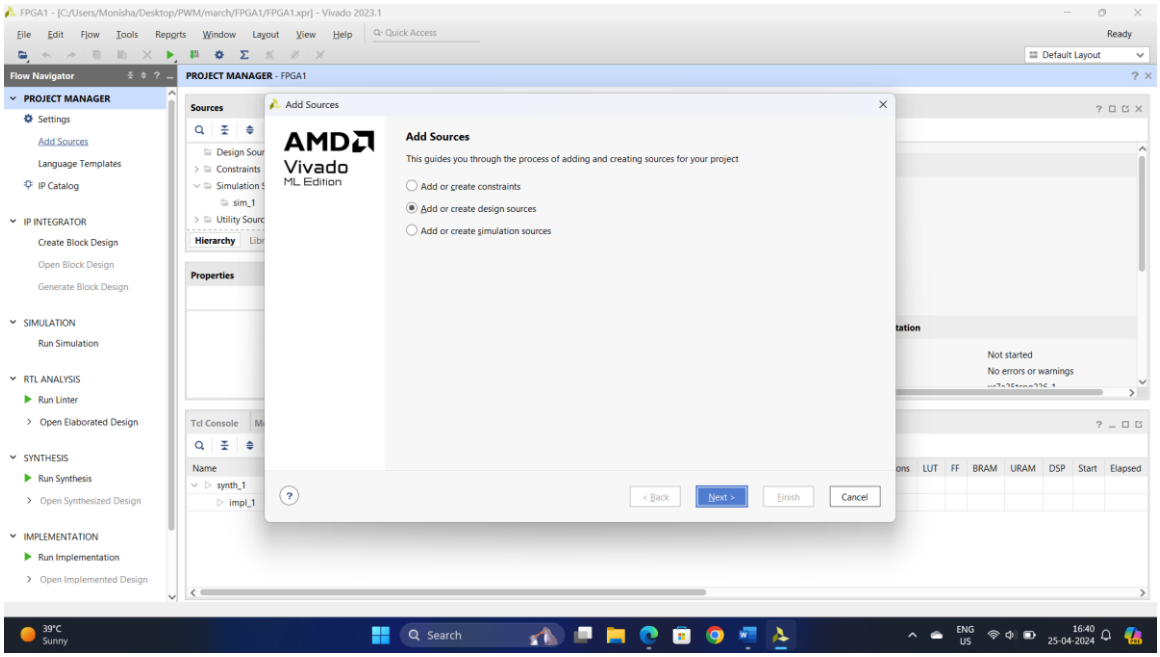

Figure 4. 5 Adding Design Source

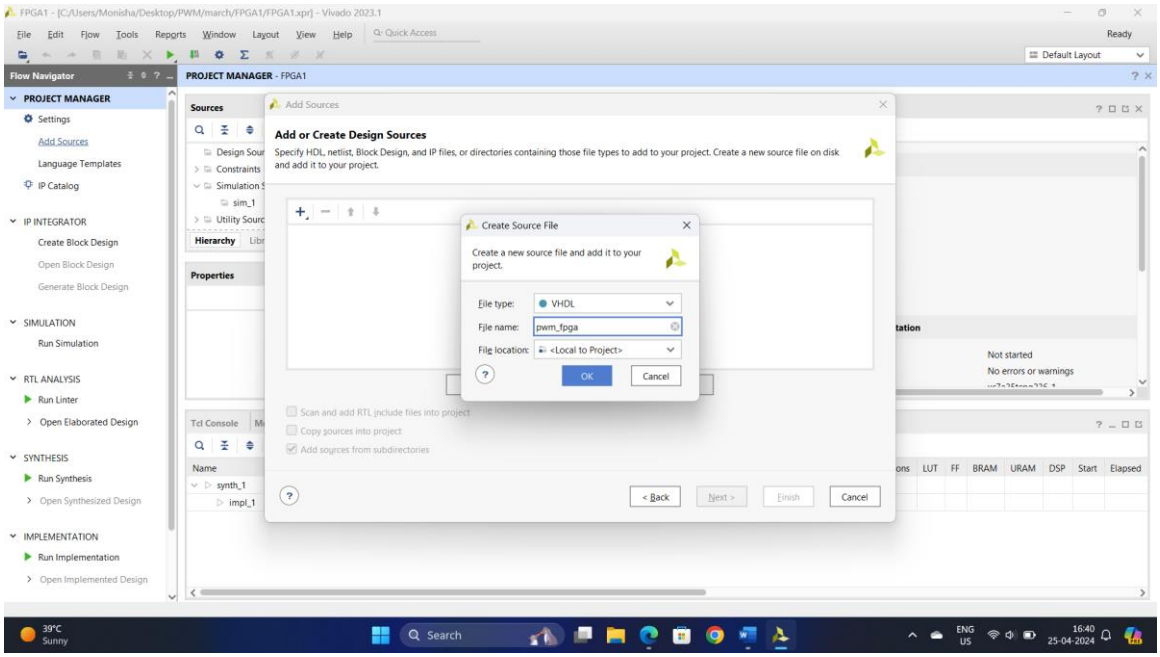

Figure 4. 6 Selecting Target Language - VHDL

Step:4 Synthesize Design: After adding design sources and constraints, synthesize your design to check for syntax errors and analyze its structure. Navigate to Flow Navigator > Run Synthesis.

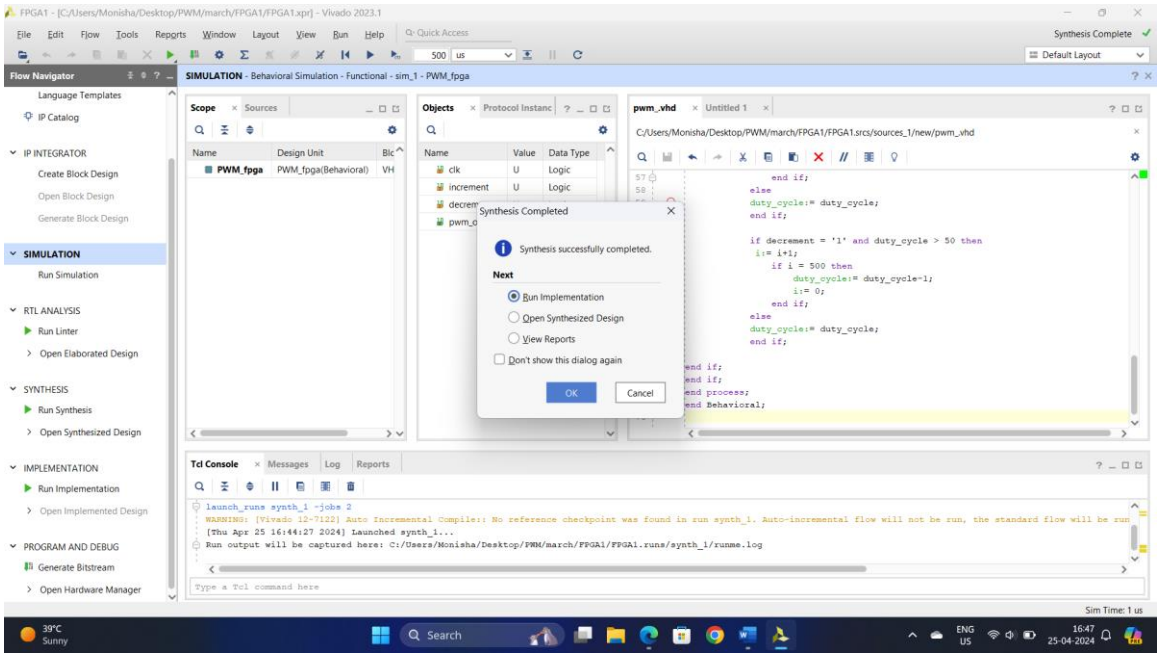

Figure 4. 7 Run Synthesis

Step:6 Add Constraints: Define constraints for your design, such as pin assignments, timing constraints, and I/O standards. Navigate to Flow Navigator > Project Manager > Add Constraints.

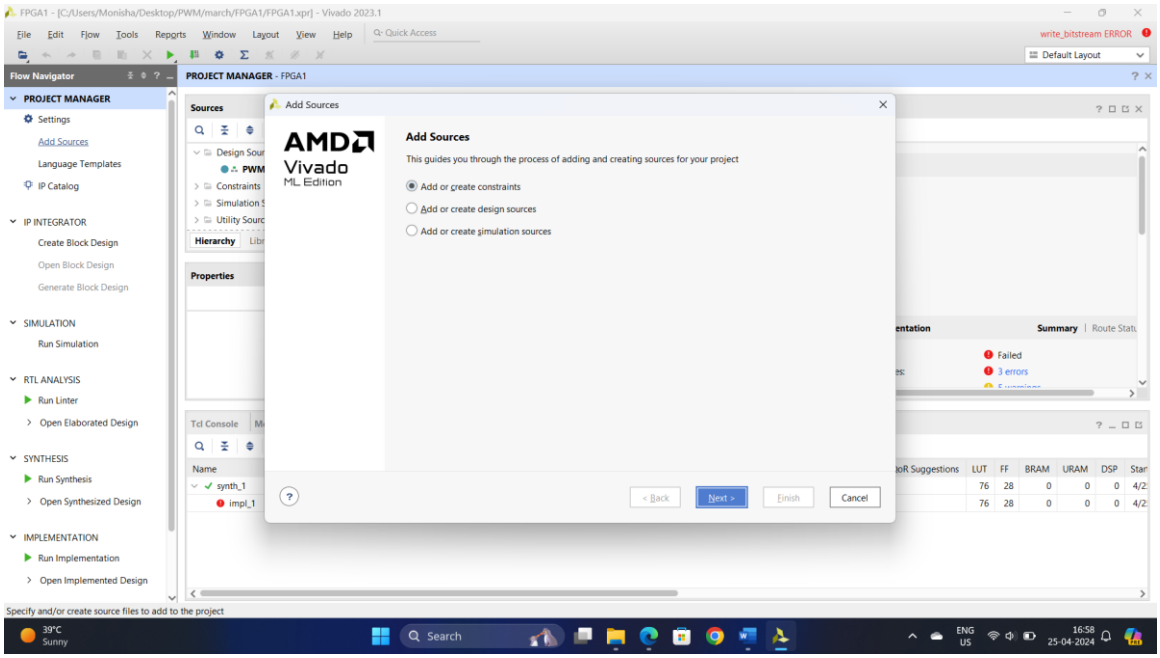

Figure 4. 8 Creating Constrain Source

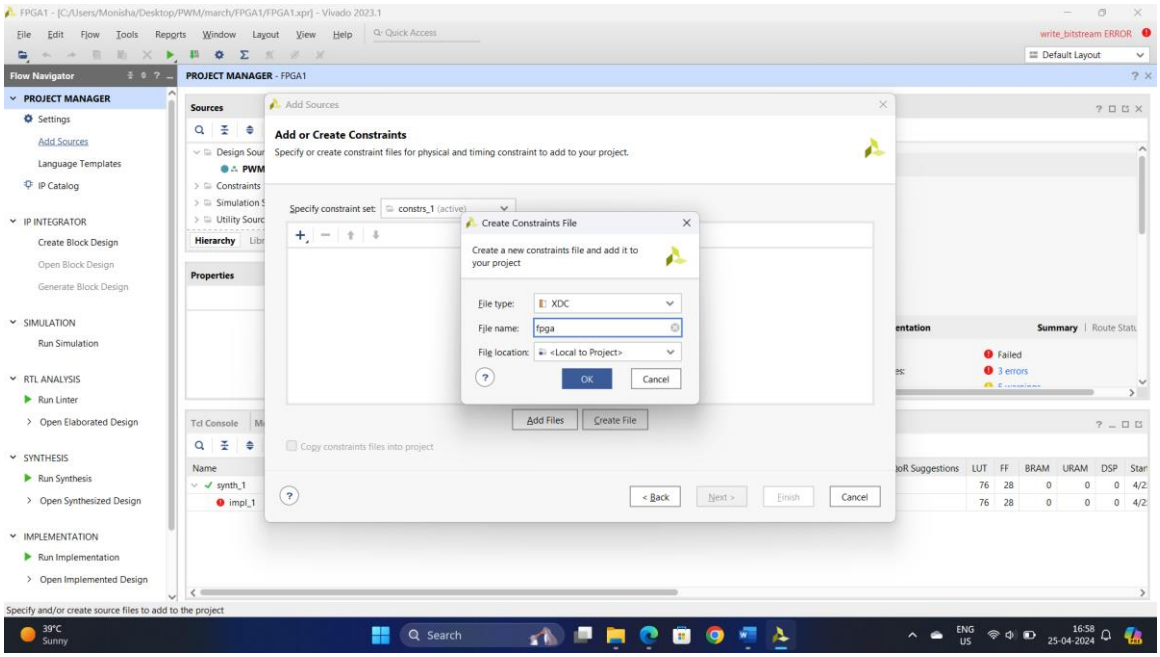

Figure 4. 9 Adding Constrain file

Step:7 Implement Design (Place and Route): After synthesis, implement the design which involves placing and routing the logical elements of your design onto the physical FPGA. Navigate to Flow Navigator > Run Implementation.

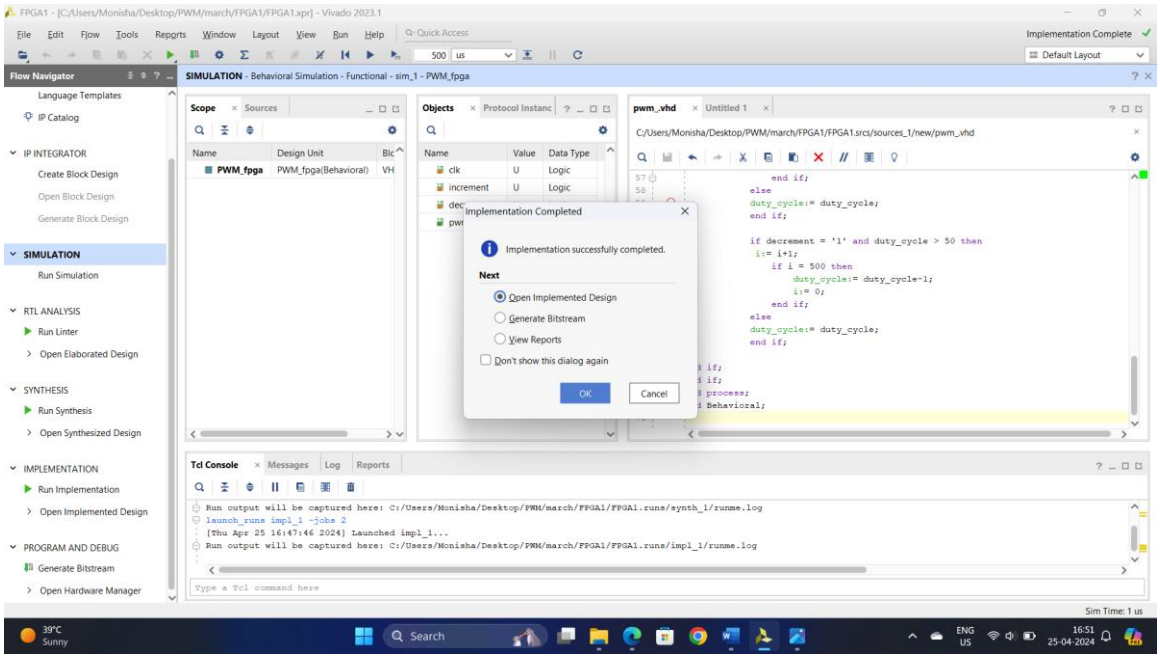

Figure 4. 10 Run Implementation

Step:8 Generate Bitstream: Once the design is successfully implemented, you can generate the bitstream, which is the configuration file for the FPGA.

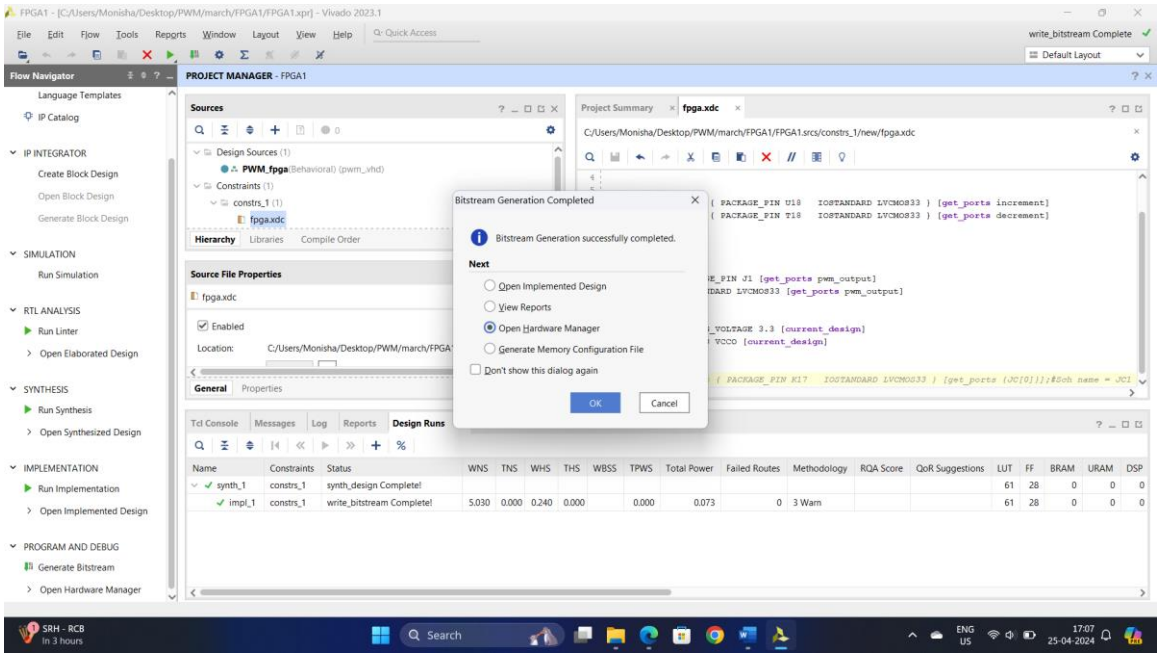

Figure 4. 11 Bit stream Generation

Step:9 Program FPGA: Use the generated bitstream file to program your FPGA device.

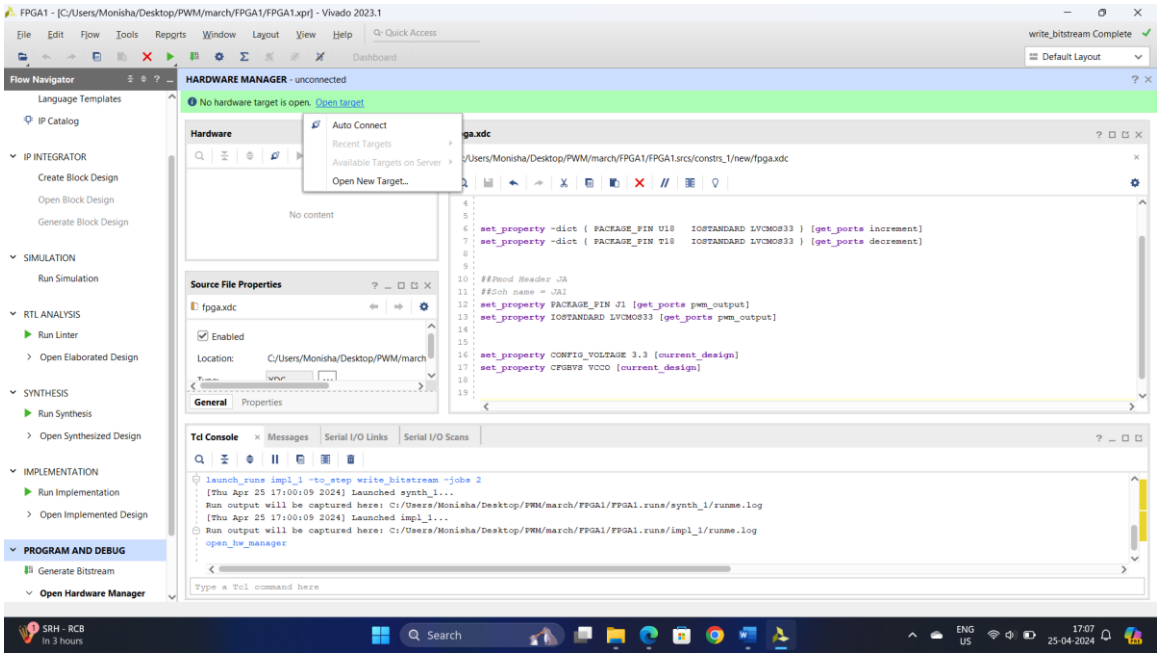

Figure 4. 12 Connecting VIVADO and FPGA board

4.3 Implementation on VIVADO

4.3.1 Design of PWM

At some simulation time, a module input may be stimulated by changing the value on an input port. The module reacts by running the code of its behavioral description and scheduling new values to be placed on the signals connected to its output ports at some later simulated time. A VHDL description is used.

Packages and Libraries: Libraries contain packages and Packages contain commonly used types, operators, constants, functions, etc. Both must be “opened” before their contents can be used in an entity or architecture.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
```

The entity of the design of PWM syntax is given below, which includes the input and the output interface signal.

```
entity PWM_FPGA is

    Port ( clk : in STD_LOGIC;

        increment : in STD_LOGIC;

        decrement : in STD_LOGIC;

        PWM_OUTPUT : out STD_LOGIC);

end PWM_FPGA;
```

Ports: Provide communication with other components, must have signal name, type and mode.

Port Modes :

in (data goes into entity only)

out (data goes out of entity only and not used internally)

in out (data is bi-directional)

buffer (data goes out of entity and used internally)

The internal aspect of the design unit can be behavioral (RTL), and always associated with single entity. Single entity can have multiple architectures. An implementation of the entity is described 48 in an architecture body. There may be more than one architecture body corresponding to a single entity specification, each of which describes a different view of the entity. A behavioral description of the pulse width modulation could be written as:

```
architecture Behavioral of PWM_fpga is
begin
process(clk)
variable count : integer range 0 to 500;
variable duty_cycle : integer range 100 to 500;
variable i : integer range 0 to 500;
begin
if rising_edge(clk) then
count := count+1;
   if count = duty_cycle then
   pwm_output<= '1';
   end if;
      if count = 500 then
      pwm_output <= '0';
      count:= 0;
```

in this if rising edge of clk is in sensitivity list and after that we compared our counter value and duty cycle value.

```
         if increment = '1' and duty_cycle < 450 then
         i:= i+1;
            if i = 500 then
```

```
            duty_cycle:= duty_cycle+1;
            i:= 0;
          end if;
        else
        duty_cycle:= duty_cycle;
        end if
        if decrement = '1' and duty_cycle > 50 then
         i:= i+1;
          if i = 500 then
            duty_cycle:= duty_cycle-1;
            i:= 0;
          end if;
        else
        duty_cycle:= duty_cycle;
                end if;
         end if;
end if;
end process;
end Behavioral;
```

Here, we have defined that if we push increment or decrement button then variable I have added 1 into it , it will lead to add in duty cycle value, similar logic has applied for decrement value.

Signals and Variables are two fundamental types of objects used to carry data from place to place in a VHDL design description: Signals are assigned via “<=”! Variables are assigned via “:=”. Signals are used in the connection parts for VHDL entities. Variables are declared with in the process block, procedures, and functions. Signals can only be declared with in architecture bodies. They can be passed as parameters to functions and procedures. A process is a body of code which is executed whenever any of the signals it is sensitive to changes value.

4.3.2 FPGA Constrain for PWM

A Xilinx Design Constraints file or **XDC** file is needed to interface between your System modules and the Basys 3. It hooks up the inputs and outputs of your module to pins, buttons, LEDs, switches, etc. on the board. An XDC file is *required* to generate configuration bit file.

An XDC file is simply a Tcl file with Tcl commands. These Tcl commands add properties to the ports of your System file (properties that indicate where the port should be hooked up). The file is then executed before bitstream generation so that these properties are attached to the design.

Creating XDC files is tedious and repetitive. To make things easier, you're provided with a master XDC file that contains all the XDC constraints for each pin on the Basys 3 that you will use in these labs.

Xdc file:

```
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports increment]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports decrement]
##Pmod Header JA
##Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports pwm_output]
set_property IOSTANDARD LVCMOS33 [get_ports pwm_output]
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```

CHAPTER 5 - RESULT AND CONCLUSION

5.1 PWM on Oscilloscope

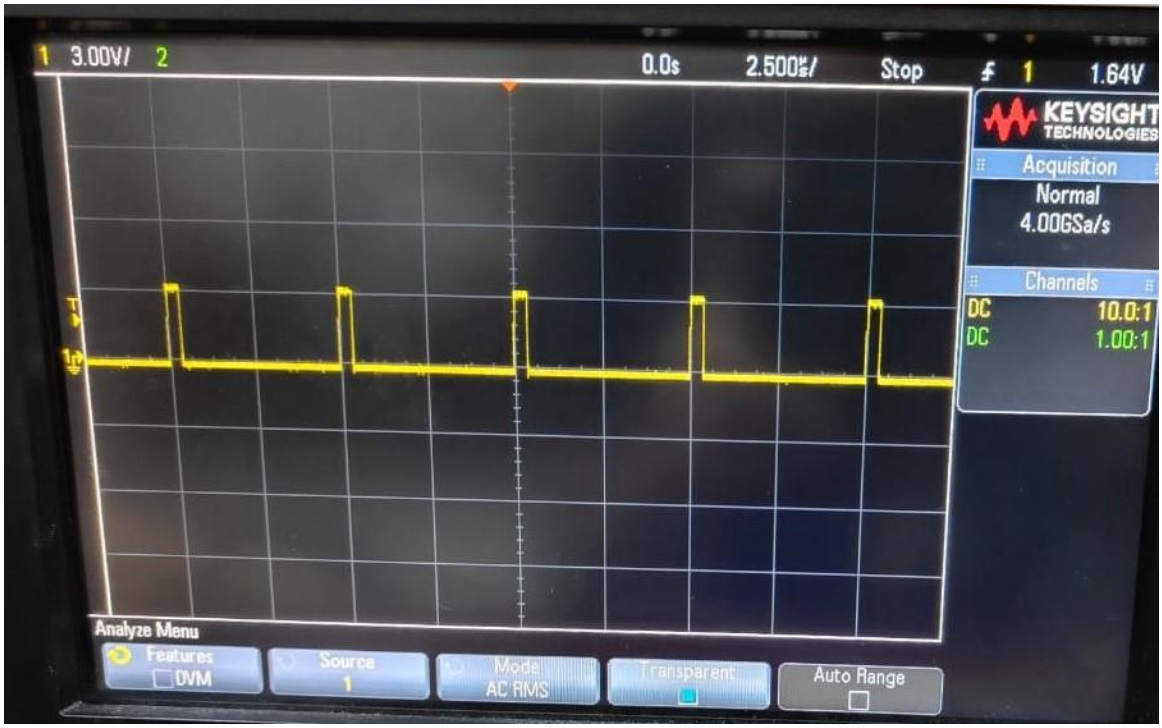

Figure 5. 1 duty cycle of 10%

A duty cycle of 10% indicates that the signal spends 10% of the total period in the high state and 90% in the low state within one complete cycle. Visually, on the oscilloscope, you will observe a short high pulse followed by a longer low period.

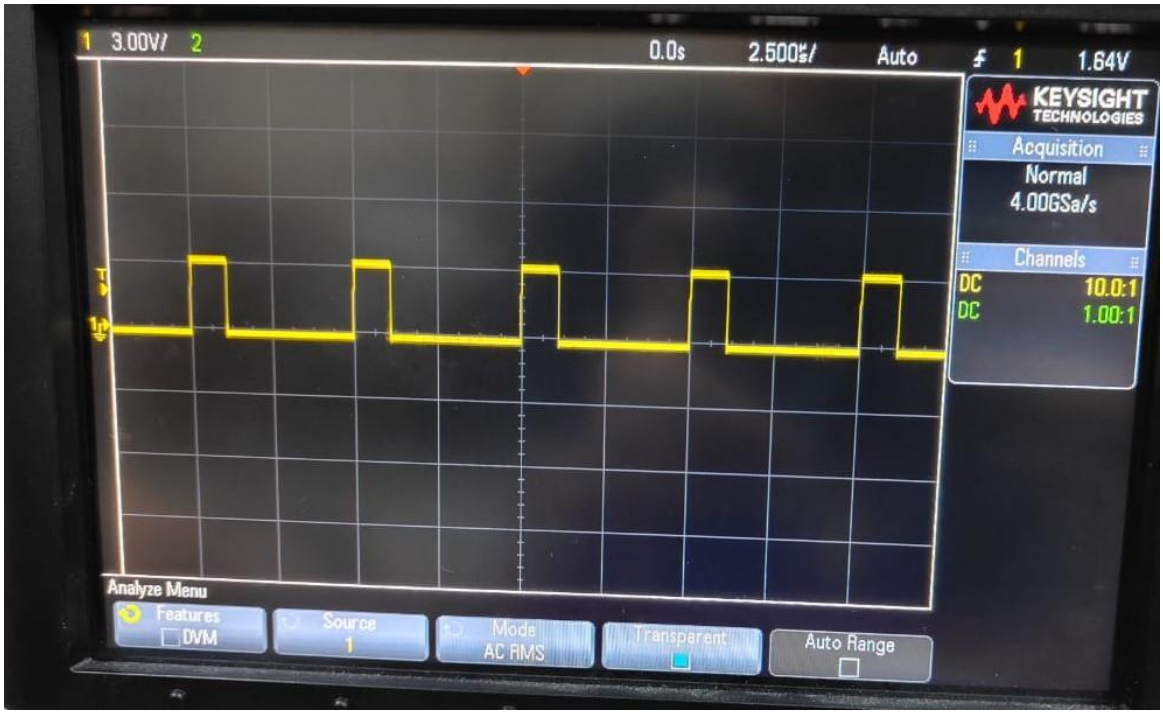

Figure 5. 2 duty cycle of 30%

A 30% duty cycle indicates that the signal spends 30% of the total period in the high state and 70% in the low state within one complete cycle. Visually, the oscilloscope will display a relatively short high pulse followed by a longer low period.

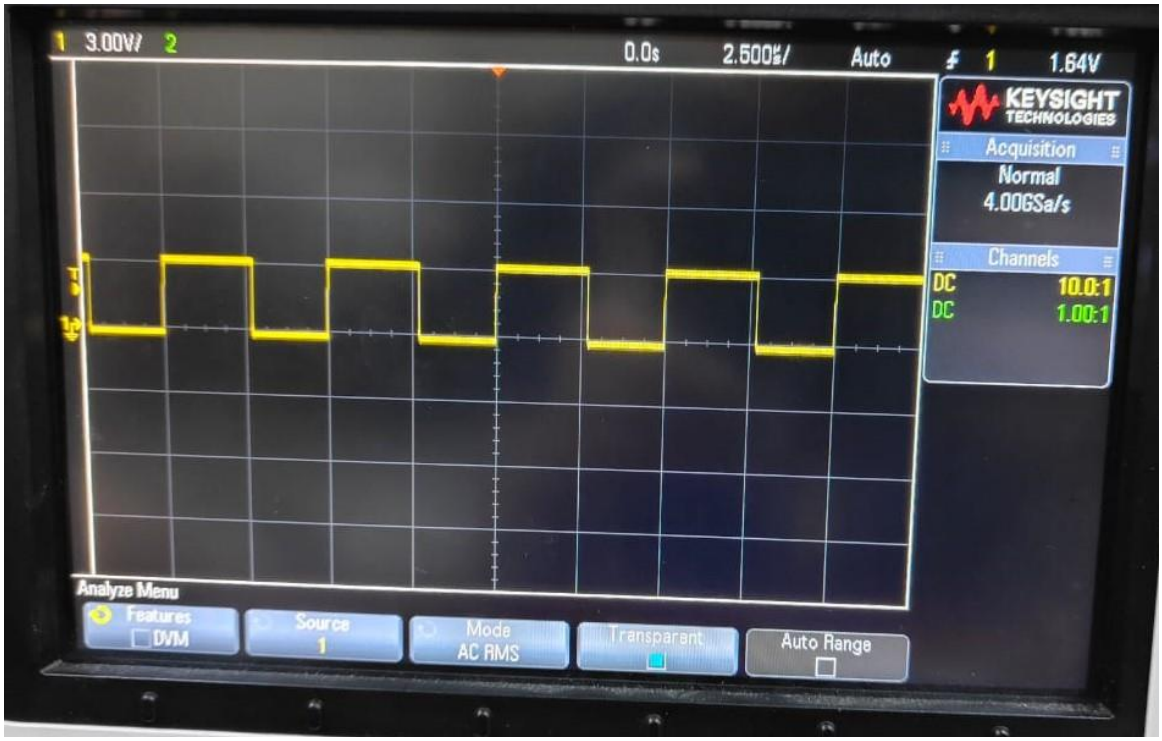

Figure 5. 3 duty cycle of 50%

A duty cycle value close to 50% indicates a square wave signal with equal time spent in the high and low states. Deviations from 50% indicate asymmetry in the signal, with higher duty cycles indicating longer high states and lower duty cycles indicates.

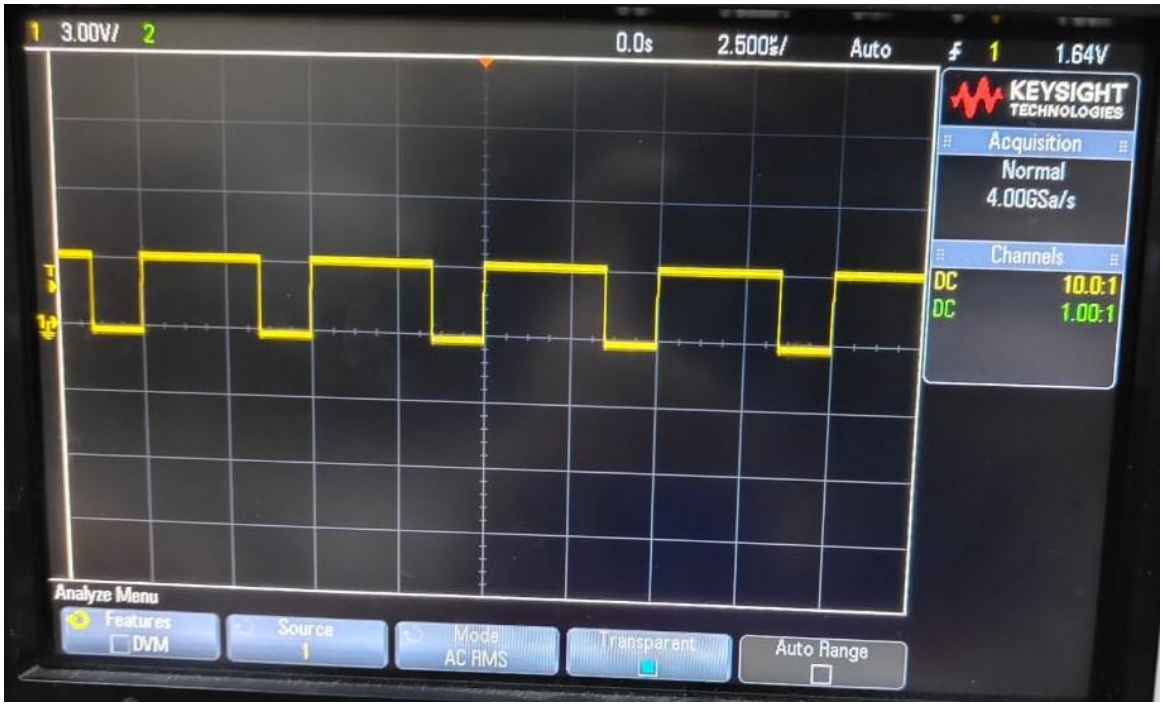

Figure 5. 4 duty cycle of 70%

A 70% duty cycle means the signal spends 70% of the total period in the high state and 30% in the low state within one complete cycle. Visually, on the oscilloscope, you'll observe a relatively long high pulse followed by a shorter low period.

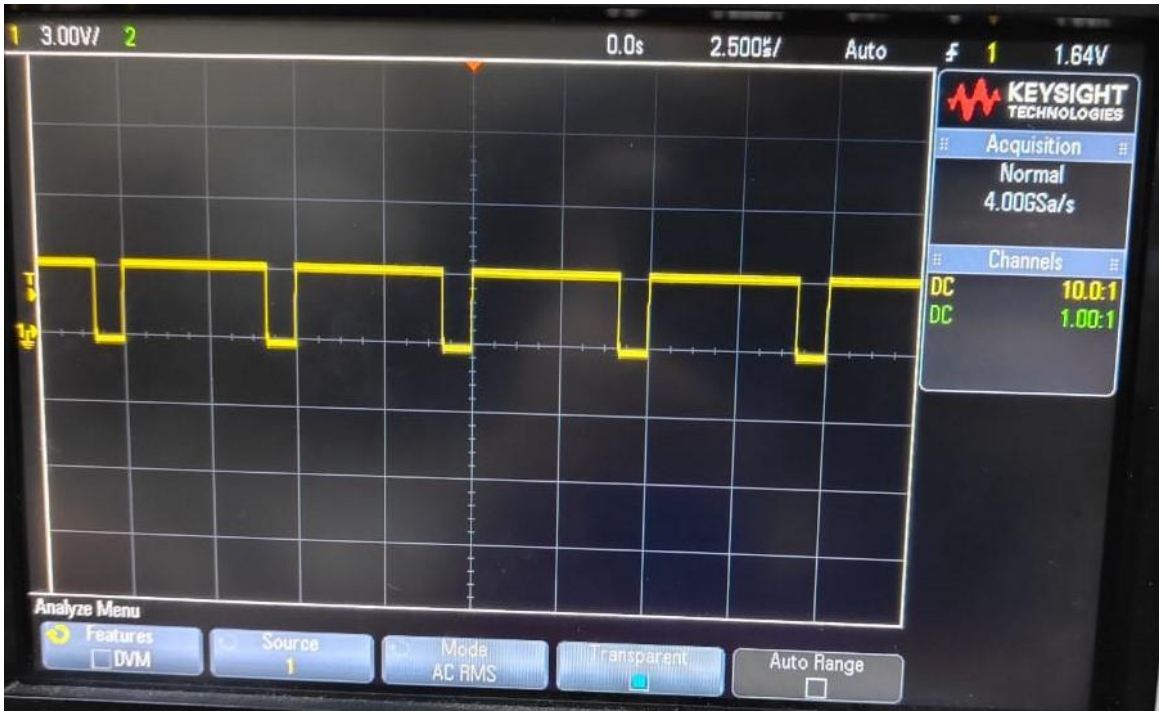

Figure 5. 5 duty cycle of 85%

High Time: 85% of the total period Low Time: 15% of the total period

Figure 5. 6 duty cycle of 92%

High Time: 92% of the total period Low Time: 8% of the total period

5.1.2 Limitations

Clock Jitter and Noise:Clock jitter (variation in clock edges) can introduce noise in the PWM signal. This noise affects the accuracy of the duty cycle.

Switching Losses:In applications like motor control, where PWM is commonly used, switching losses occur during transitions between high and low states.These losses can lead to inefficiencies and affect overall system performance.

Resource Utilization:Implementing PWM logic in an FPGA consumes resources (such as lookup tables, flip-flops, and routing channels)Careful design is necessary to optimize resource usage and avoid resource bottlenecks.

5.1.3 Further Implementations

Induction Heating: Induction heating utilizes high-frequency alternating current (AC) to generate heat in conductive materials through electromagnetic induction. PWM is commonly used to control the power delivered to the induction coil, which in turn regulates the heating intensity.

DC-to-DC Conversion: DC-to-DC conversion involves converting one DC voltage level to another. PWM is commonly used in switch-mode power supplies (SMPS) for efficient voltage regulation.

DC-to-AC Inversion: DC-to-AC inversion, commonly known as inverter operation, involves converting a DC input into an AC output. PWM is widely used in inverters to synthesize high-quality AC waveforms.

5.2 Conclusion

Using Xilinx VIVADO 2023.1 software we can develop the proposed PWM in Xilinx FPGA. Due to the need of design flexibility in FPGA, a PWM was developed using VHDL modeling in Xilinx Field Programmable Gate Array. A VHDL model was implemented on Artix-7 BASYS 3 and optimized for area. The simulation results prove that using the proposed method, PWM can be produced with a duty cycle and which is adequate for most applications like DC to DC converters , AC to AC converters especially nowadays It is Used in INTERNET OF THINGS. In addition to this the VHDL modeling and the

architectural features of XILINX FPGAs is studied. The PWM developed is used for DC motor speed control here by varying its duty cycle.

Bibliography

1. Stephen Brown, “Fundamentals of digital logic with VHDL design,” July 2002.
2. Gwaltney, “FPGA Implementation of controls,” 2003.
3. Peter J. Ashendenal , “The VHDL cookbook ’’ First Edition, July 1990, South Australia.
4. Karen Parenell, Nick Mehta,“ Programmable logic design Quick start Hand Book, ”Xilinx, Second edition, January 2002.
5. Karen Parenell , Nick Mehta, “ Programmable logic design Quick start Hand Book, ” Xilinx, Third edition, January 2002.
6. Angel Vpeterchev, “Digital pulse width modulation in power electronic circuits,” July2002.
7. S.poorani, K.udays Kumar, “FPGA based Fuzzy Logic controller for electric vehicle,” Vol.45 issue 5, 2005 Journal of IOE, Singapore.
8. Xilinx Integrated System Enviroment 2007 Manuel, Ver 8.2i.
9. Joannis Sourdis, “Efficient and High speed FPGA based string matching for Packet Inspection,” Msc thesis, Chania, July 2004.
10. Xilinx official Web site, http//www.xilinx.com. [11]Dougleas Perry, “VHDL,” Third edition, 2003.
11. Sunggu Lee, “Design of Computers and other complex digital devices,” July 2000.
12. Tomas, “the low carb Very High Sped Integrated Circuit Hardware Description Language Tutorial,” 2003.
13. Alexander Prodic,” Design and Implementation of a digital PWM controller for a High- frequency switching DC-DC Power Converter,” IECON’01, the 27th Annual conference of the IEEE industrial electronics society, 2001, USA.

14. M. Wang, A. Ranjan, S. Raje, “Multi-Million Gate FPGA Physical Design Challenges”, ICCAD, pp. 891-898, 2004.

15. J. M. Emmert, and D. Bhatia, “A Methodology for Fast FPGA Floorplanning”, Proc. FPGA, 1999.

16. R. Ramos, X. Roset, A. Manuel, Implementation of fuzzy logic controller for DC/DC converters using field programmable gate array, in: Proc. 17th IEEE Instrumentation and Measurement Technology Conference, vol. 1, 2000, pp. 160–163.

17. M.M. Islam, D. Allee, S. Konasani, A. Rodriguez, A lowcost digital controller for a switching DC converter with improved voltage regulation, IEEE Power Electronics Letters 2 (2004) 121–124.

18. http://www.digilentinc.com (Xilinx I/O board).

19. Diter Metzner, George Parz, “HDL based system engineering for Automotive Power Applications,” D-81609 Munchen, Germany, 2003.

20. Patrick Pelgrims, Tom Tirens,” Learning about VHDL and FPGA’s,”Version 1.0,2003.

21. http://www.quicklogic.com/support(official web site)

22. Xilinx. Basys 3 Platform FPGAs: Complete data sheet. DS031 v3.3, June 2005.

23. P. J. Ashenden, “The Designer’s Guide to VHDL,” 2nd ed. Morgan Kaufmann Publishers, 2002.

24. N. Mohan, T. Undeland, W. Robbins,” Power Electronics Converters, Applications and Design”, second ed., Wiley,1995.

Annexure

Xilinx Devices:

Artix-7 FPGAs

The Xilinx® Artix®-7 family of FPGAs has redefined cost-sensitive solutions by cutting power consumption in half from the previous generation while providing best-in-class transceivers and signal processing capabilities for high bandwidth applications. Built on the 28nm HPL process, these devices deliver best in class performance-per-watt. Together with the MicroBlaze(TM) soft processor, Artix-7 FPGAs are ideal for products like portable medical equipment, military radios, and compact wireless infrastructure. Artix-7 FPGAs meet the needs of size, weight, power, and cost (SWaP-C) sensitive markets like avionics and communications.

KEY CAPABILITY OVERVIEW :

New Levels of Performance

Smallest Package

Twice the Capacity, Half the Power, Comparable Cost

Low Risk, Rapid Ramp-Up

Basys3 FPGAs

The Basys 3 board is a complete, ready-to-use digital circuit development platform based on the latest Artix®-7 Field Programmable Gate Array (FPGA) from Xilinx®. With its high-capacity FPGA (Xilinx part number XC7A35T1CPG236C), low overall cost, and collection of USB, VGA, and other ports, the Basys 3 can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs, and other I/O devices to allow a large number of designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits. Artix-7 35T features include:

33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops)

1,800 Kbits of fast block RAM

Five clock management tiles, each with a phase-locked loop (PLL)

90 DSP slices

Internal clock speeds exceeding 450MHz

On-chip analog-to-digital converter (XADC)

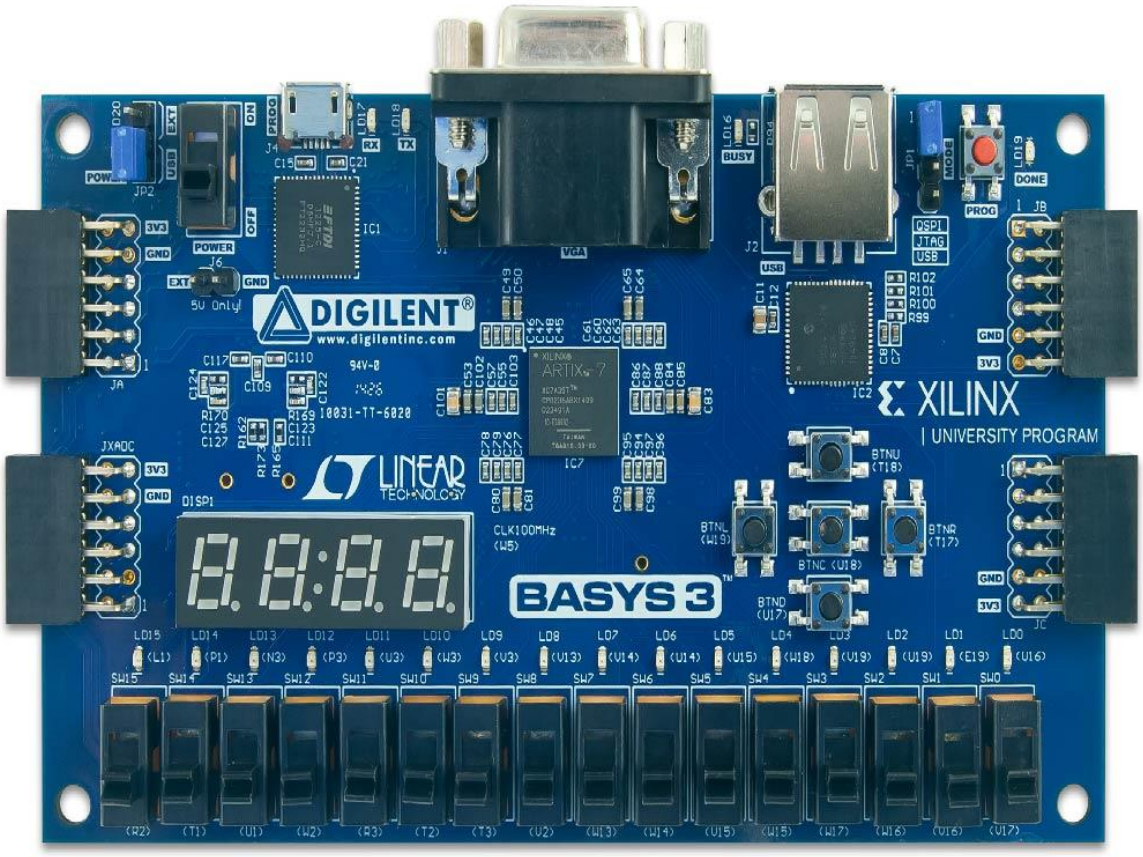

Figure 6. 1 ARTIX 7 BASYS 3 FPGA BOARD

The Basys 3 also offers an improved collection of ports and peripherals, including: 16 user switches ,16 user LEDs ,5 user pushbuttons ,4-digit 7-segment display ,Three Pmod ports ,Pmod for XADC signals,12-bit VGA output ,USB-UART Bridge ,Serial Flash,

DigilentUSB-JTAG port for FPGA programming and communication ,USB HID Host for mice, keyboards and memory sticks.

ULN2003 Pinout Configuration

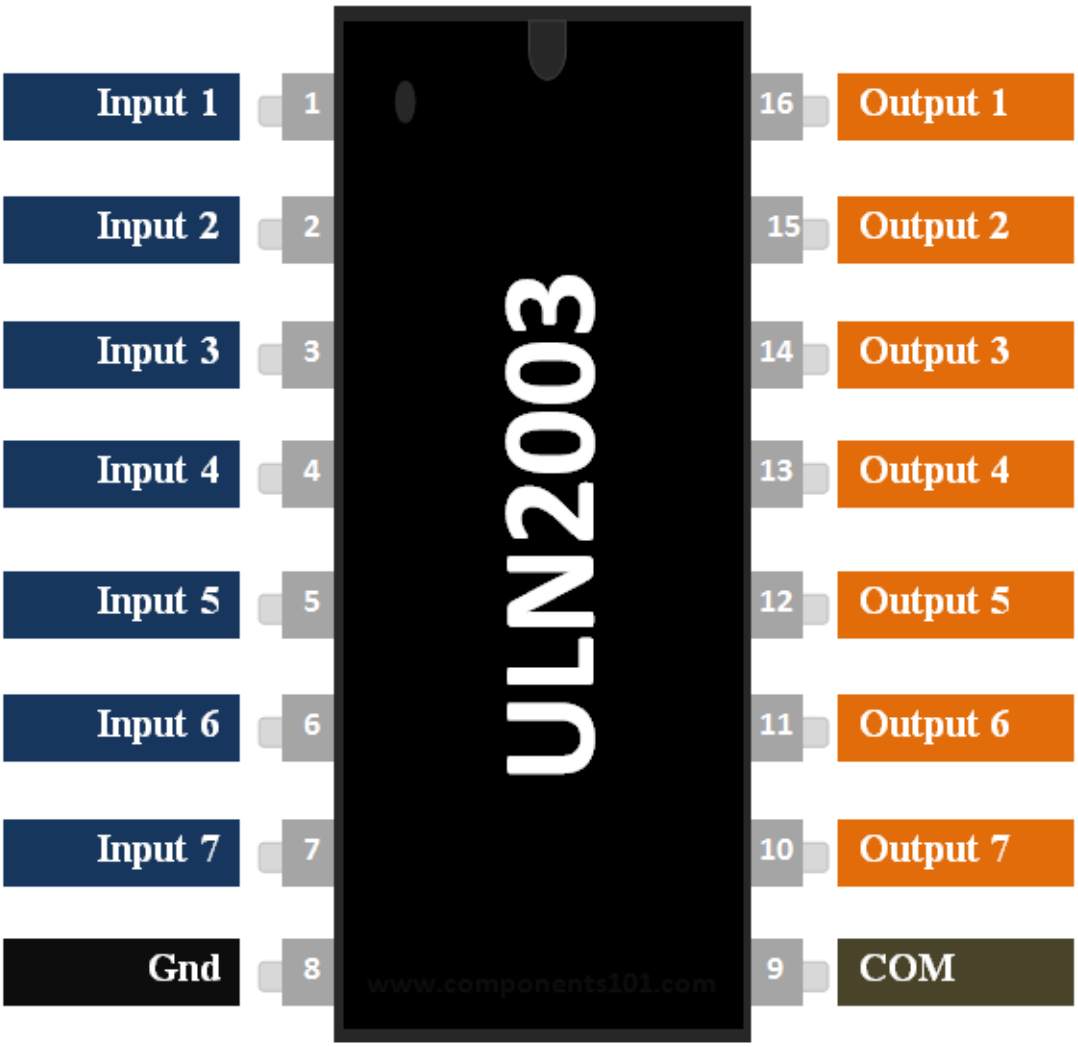

Figure 6. 2 PIN diagram of ULN2003APG Driver IC

Table 2 PIN Description

Pin Number	Pin Name	Description
1 to 7	Input 1 to Input 7	Seven Input pins of Darlington pair, each pin is connected to the base of the transistor and can be triggered by using +5V
8	Ground	Ground Reference Voltage 0V
9	COM	Used as test pin or Voltage suppresser pin (optional to use)
10 to 16	Output 1 to Output 7	Respective outputs of seven input pins. Each output pin will be connected to ground only when its respective input pin is high(+5V)

ULN2003 Features:

Contains 7 high-voltage and high current Darlington pairs

Each pair is rated for 50V and 500mA

Input pins can be triggered by +5V

All seven Output pins can be connected to gather to drive loads up to (7×500mA) ~3.5A.

Can be directly controlled by logic devices like Digital Gates, Arduino, PIC etc

Available in 16-pin DIP, TSSOP, SOIC packages.