Study of triggering schemes for pulsed power systems using FPGA

INTERNSHIP REPORT

Submitted by

JAYESH MULCHANDANI

in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND TELECOMMUNICATION

INSTITUTE OF ENGINEERING & TECHNOLOGY

INDORE

DEVI AHILYA VISHWAVIDYALAYA- 452020

GOVERNMENT OF INDIA

DEPARTMENT OF ATOMIC ENERGY

RAJA RAMANNA CENTER OF ADVANCED TECHNOLOGY

CERTIFICATE

This is to certify that Mr. Jayesh Mulchandani (Enrollment No. DE20663), student of Institute of Engineering and Technology , Indore has worked with us as internship trainee and participated in the work being carried out in our division on the subject " **Study of triggering schemes for Pulsed Power Systems using FPGA"** during the period 14rd February 2023 to 30th June 2023 at Accelerator Test Facility Building, Raja Ramanna Centre for Advanced Technology(RRCAT), Indore under our supervision. We recommend this work towards this partial fulfillment for the award of the degree of Bachelor of Engineering in Electronics and Telecommunication. We wish him all the professional success in the future.

Shri Mahendra Lad	Shri T.Reghu	Shri Mohan Tiwari
OS	SO/G	SO/C
AD, PAG & Head, RFSD	Head, KMPSL,RFSD	KMPSL,RFSD

Director

Institute of Engineering & Technology

Devi Ahilya Vishwavidyalaya,

Indore (M.P.)

Report Approval Sheet

The INTERNSHIP REPORT submitted by JAYESH MULCHANDANI (DE20663) is approved as partial fulfillment for the award of Bachelor of Engineering in ELECTRONICS AND TELECOMMUNICATION degree by Institute of Engineering and Technology Devi Ahilya Vishwavidyalaya, Indore.

Internal Examiner	External Examiner
IET-DAVV	

Endorsed By :

Head, Department of Electronics and Telecommunication

Candidate Declaration

I hereby declare that the work which is being presented in this internship report in partial fulfillment of degree of Bachelor of Engineering in Electronics and Telecommunication at Institute of Engineering and Technology, Devi Ahilya Vishwavidyalaya, Indore is an authentic record of our own work carried out under the supervision and guidance of Shri Mohan Tiwari, KMPSL, RFSD, RRCAT Indore.

We are fully responsible for the matter embodied in this internship report in case of any discrepancy found in the internship report and the internship report has not been submitted for the award of any other degree.

Date:

Place:

Jayesh Mulchandani

ACKNOWLEDGEMENTS

I wish to express my sincere and heartfelt gratitude to my internship guide Shri T. Reghu Sir, Head, KMPSL, RFSD and my technical supervisor Shri Mohan Tiwari Sir, SO/C, KMPSL, RFSD for their invaluable guidance, constant inspiration, mentorship and immense support during the course of the project as well as writing of this report.

I also wish to express my sincere gratitude to RF systems division head, Shri Mahendra Lad Sir for his valuable suggestions, inspiration, encouragement and immense support during the course of the project.

I sincerely acknowledge the efforts and useful guidance of my internal supervisor Shri Ravi Sindal Sir, Professor of Institute of Engineering and Technology, DAVV.

I also wish to thank the Project Placement Committee of RRCAT for giving me a chance to work as project trainee at RRCAT. I also wish to thank our Placement officer Dr. Govind Maheshwari Sir, Professor of Institute of Engineering and Technology, DAVV for permitting me and arranging my internship program at RRCAT, Indore.

I am thankful to Shri Deodatta Baxy Sir and Shri Prem Baboo Sir for their technical guidance during the assembly and programming. I am also thankful to Shri Ashwin Pandey Sir, Shri Jaikishan Mulchandani Sir, Shri Ankit Gupta Sir and Shri Vinod Phadnis Sir for insightful conversations and discussions. I am profoundly grateful to all the officers, assistants and technicians at ATF for their enormous help and guidance in this project.

I am deeply indebted to my parents , brother and extended family whose infallible love and support has been my greatest strength.

CONTENTS

List of Figures **Page No**

List of Tables

Chapter 1

Introduction to the Project

Overview on Particle Accelerators

A particle accelerator is a machine that accelerates charged subatomic particles and ions to high energies while confining them in a narrow beam. In the field of scientific research, particle accelerators are used for high-energy physics experiments and the discovery of fundamental particles. They are used in medicine for isotope production and cancer therapy. Accelerators have industrial applications such as food preservation, sterilization, and ion implantation .

Particle accelerators can be divided into two categories: linear accelerators that accelerate the beam in one pass and circular accelerators that allow multiple passes of the beam through the accelerating voltage. Van de Graaff generator, Cockcroft-Walton generator, drift-tube linac and radio frequency quadrupole are common types of linear accelerators. Circular accelerators include cyclotrons, microtrons and synchrotrons . Either static electric fields or changing electromagnetic fields could be used for acceleration.

Charged particles are accelerated in RF accelerators using RF power in resonating cavities. The cavity's resonant frequency is matched to the RF power frequency fed into the cavity. Magnetrons, klystrons, and solid-state power amplifiers generate this RF power. Klystrons are commonly used for high power and high frequency applications. Figure 1.1 depicts a block diagram of a typical particle accelerator's RF power system. The klystron receives a low-level RF (LLRF) signal, which it then amplifies before feeding to the cavity. A high voltage pulse generator system provides biasing power to the klystron. The high voltage pulse generator is also known as a pulse modulator because it provides a modulating signal to the klystron.

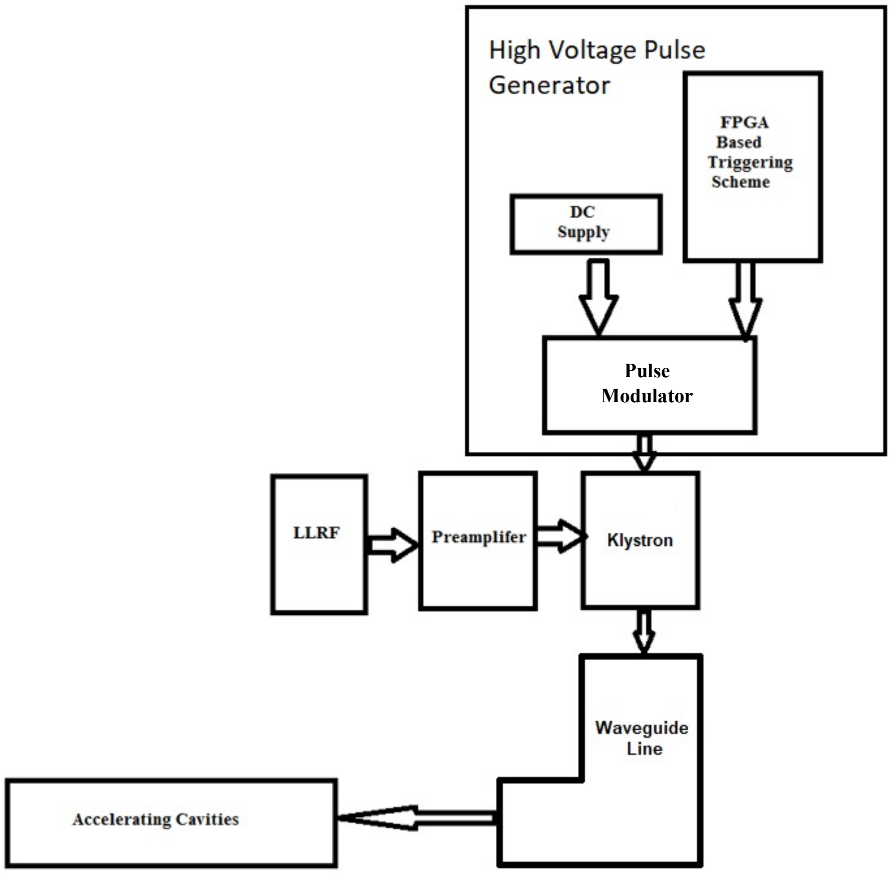


Fig (1.1): RF power system for particle accelerators

1.1) **Study of Pulse Power Topologies**

High Voltage Pulse Generators

A high voltage pulse generator is a device that produces short, high voltage pulses of electrical energy. These pulses are typically used in scientific and medical research, as well as in industrial applications such as welding and cutting. High voltage pulse generators deliver short-term biasing pulses to loads such as klystrons, magnetrons, and electron guns.

The basic components of a high voltage pulse generator include a power source, a capacitor, and a fully controlled switch. The dc power supply continuously charges the energy storing element. The stored energy is released to the load for a short period of time, resulting in a high peak power. The voltage pulse required by the load is in the hundreds of kV range. Except in Marx generator systems, such high voltage pulses are difficult to generate directly. A pulse transformer is used in most systems to step-up the pulse generated by the pulse generating unit.

High Voltage Pulse Generator Classification

The topology of a pulse generator is chosen based on the output pulse requirements. Pulse generators are divided into two types: line-type and hard-switched. A pulse-forming network (PFN), which is essentially an L-C network, is used in line-type pulse generators. A PFN discharges the energy stored in either L or C to the load.[4]. This type, however, cannot be used for variable pulse widths. Hard-switch pulse generators make use of an energy storage capacitor that is partially discharged to the load via a fully controllable switch such as an IGBT or MOSFET. This type has the advantage of allowing the pulse width to be varied by simply controlling the switch on time.

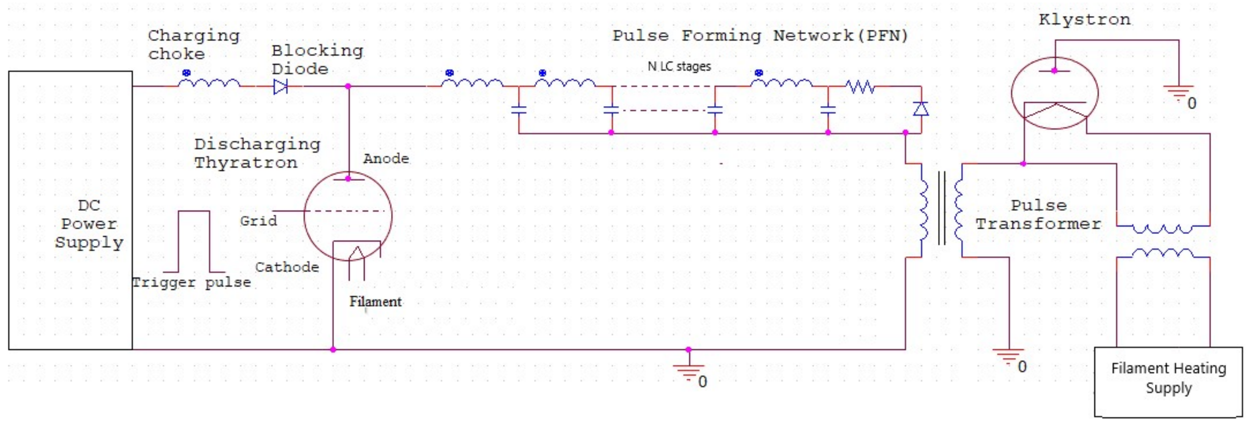


Fig (1.2): Line type pulse generator

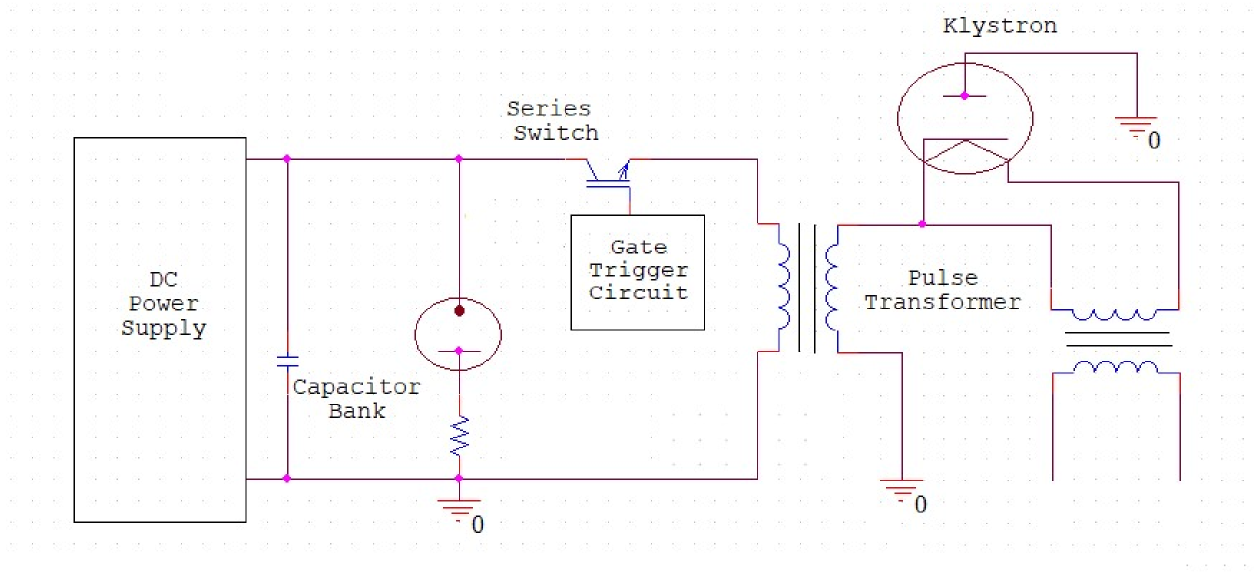


Fig (1.3): Hard switch type pulse generator

1.2) **Specification of Microwave System used in 10 MeV electron Linear Accelerator**

ARPF is the radiation processing facility developed and run by RRCAT Indore. This facility is used for irradiation of agriculture and medical products. This facility is based on 10MeV electrons linear accelerator. It contains 50keV electron gun as source of electrons. These electrons are injected into a multicavity accelerating structure. The cavities are energized by pulsed microwave source known as klystron to accelerate the electrons. The klystron is the RF amplifier tube operating in pulsed mode and its biasing voltage is provided by high voltage pulse modulator as shown in figure. The RF signal to be amplified is given its input cavity via a soild state RF amplifier of low power. The specifications of microwave systems and modulator are shown below;

Table (1.1): Microwave Systems Specifications

Peak output Power	6 MW
Average output Power	25 kW
Operating Frequency	2856 MHz
Pulse Duration	11 microsec
Pulse Repetition Rate	1-300Hz

Table (1.2): Modulator Specifications

Pulse Output Power	12MW
Pulse Voltage Output	130 kV
Pulse Current Output	90 A
Pulse duration	12 microsec
Output Power	42 kW

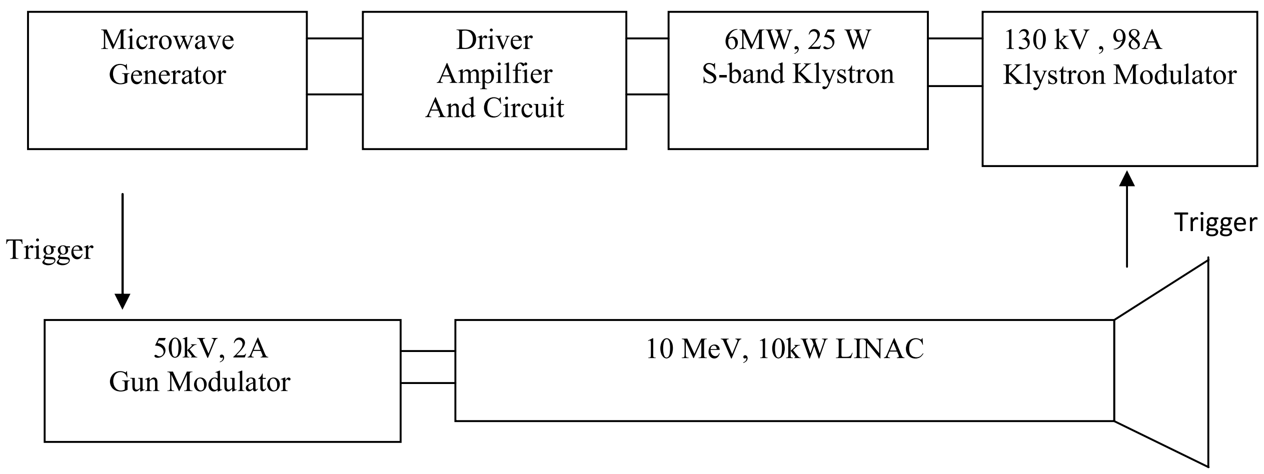

Microwave Generator
Driver Ampilfier And Circuit
6MW, 25 W S-band Klystron
130 kV , 98A Klystron Modulator
Trigger
Trigger
50kV, 2A Gun Modulator
10 MeV, 10kW LINAC

Fig (1.4): Diagram of Linac System

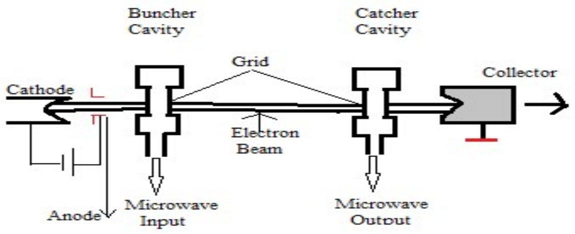
Microwave Systems Components

1.3) **Klystron**

Klystron is a specialized vacuum tube that is used for generating high-power microwaves. It is commonly used in radar systems, satellite communication, and particle accelerators. The klystron works by using an electron beam to interact with a resonant cavity, which causes the electrons to bunch together and emit microwave radiation. One of the main advantages of klystrons is their ability to generate high-power microwaves, which makes them ideal for use in a variety of applications. However, klystrons are also relatively large and expensive compared to other types of microwave generators, and they require a high voltage power supply and cooling system to operate. The klystron operates on the principle of velocity modulation, in which a beam of electrons is accelerated and decelerated by a series of resonant cavities called "bunchers" and "catchers." The electron beam enters the first cavity, or buncher, where it is subjected to a high-frequency electric field that causes the electrons to bunch together in groups. As the bunched electrons move through the cavity, they encounter a series of gaps that are tuned to the same frequency as the input signal. As the electrons pass through the gaps, they are either accelerated or decelerated depending on the polarity of the electric field.

The bunched electrons then enter the second cavity, or catcher, where they are subjected to a similar electric field that extracts energy from the bunched electrons and transfers it to the output signal. The output signal is then amplified and sent to the next stage of the circuit.

The klystron can operate at very high frequencies, up to tens or even hundreds of gigahertz. It is also capable of producing very high levels of power, up to several megawatts in some applications. This makes the klystron a popular choice for many applications, including radar, satellite communications, and particle accelerators .One type of klystron that is commonly used in particle accelerators is the klystron oscillator. In this configuration, the klystron is used to generate a high-power, high-frequency signal that is used to accelerate particles in a linear accelerator. The klystron oscillator consists of a series of resonant cavities that are tuned to the desired frequency and connected in a loop. The loop is fed by a low-power input signal, which is amplified by the klystron and output as a high-power signal.



Fig(1.5):Klystron

S-band klystron is a type of vacuum tube that is used for generating high-power microwaves in the range of 2 to 4 GHz. It is widely used in applications such as radar systems, satellite communication, and particle accelerators. High peak power S- band Klystrons are generally designed and operated for pulsed mode operation.

TH 2173 F

Fig (1.6): Photograph of Thales make S- band Klystron (TH2173 F)

Table (1.3): Specifications of S- band klystron (TH 2173 F)

Frequency Range	2856 MHz
RF output : Peak output power Average output power	 6 MW 36kW
Gain	51 dB
Efficiency	50%
Heater Voltage	7.5V
Heater Current	26A
Beam peak voltage	140 kV
Beam peak current	95 A

Applications of S- band Klystron

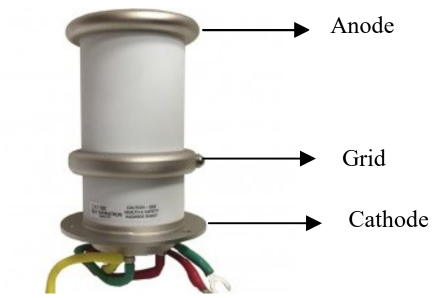
- Radar: S band klystrons are widely used in radar systems for air traffic control, weather monitoring, and military applications. They are used to generate high-power signals that are transmitted and received by radar antennas.
- Communications: S band klystrons are also used in satellite communication systems, particularly for downlink transmission of data from the satellite to the ground station. They can provide high power output and good efficiency, making them a popular choice for this application.
- Particle accelerators: S band klystrons are used in linear particle accelerators to accelerate charged particles to high energies. They are used to generate high-power radio frequency fields that are used to accelerate the particles.
- Medical applications: S band klystrons are used in medical linear accelerators for cancer treatment. The high-power radio frequency fields generated by the klystron are used to accelerate electrons, which are then used to produce X-rays for radiation therapy.
- Industrial heating: S band klystrons are used for industrial heating applications, such as in the food processing industry. They are used to generate microwave energy that is used to heat or cook food products.
- Plasma physics: S band klystrons are used in plasma physics research to generate high-power radio frequency fields that are used to heat and confine plasmas in fusion reactors.

1.4) **Thyratron**

Thyratron is a type of gas-filled tube that is used as a high-power electrical switch. It was first invented by William W. Hansen in 1928 and was widely used in early electronic devices. Thyratrons are commonly used in applications where high voltage and high current switching is required, such as in radar systems, particle accelerators, and other high-energy physics experiments. Thyratrons work by using a control electrode to regulate the flow of electrons between two other electrodes. When the control electrode is activated, it allows a large current to flow between the other two electrodes, effectively closing the switch. This electrode is called Grid or Trigger electrode of Thyratron. A fast rising trigger pulse of 1-2 kV and pulse width of 2-3 μs is applied on this electrode which in turns energize and injects free electrons from thermionic cathode towards the Anode region. These free electrons ionize the gas filled in the Thyratron very fast by avalanche action and thus the tube becomes conductive.

One of the main advantages of thyratrons is their ability to handle high power levels without overheating or breaking down. They are also very reliable and have a long lifespan compared to other types of switches. Thyratrons are also able to operate at very high frequencies, making them useful in applications such as microwave amplifiers and radio transmitters.

Thyratrons have a wide range of applications, from high-energy physics experiments to industrial welding machines. They are commonly used in radar systems and particle accelerators, as well as in medical equipment such as X-ray machines. Thyratrons can also be found in military applications, such as in missile guidance systems and electronic counter measures. Thyratron are semi controlled switch and are used in switching of PFN in line type modulator.



Fig(1.7): Thyratron switch

Table (1.4): Specifications of Thyratron (CX1154)

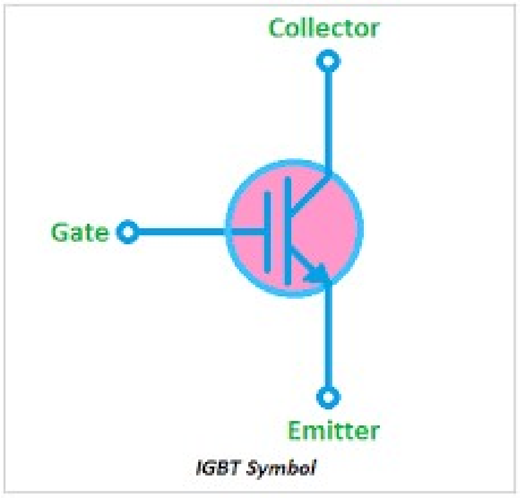
Peak Anode voltage	40 kV
Peak Anode current	3 kA
Average Anode current	2 A
Heater current	22 A
Heater Voltage	6.3 V
Trigger Grid voltage pulse	500 – 2000 V
Trigger Grid drive current	0.3 – 1A
Anode delay time	0.1 – 0.25 μs

1.5) **IGBT Switch**

IGBT stands for Insulated Gate Bipolar Transistor. It is a power semiconductor device that is used for switching and amplifying the electrical power. The IGBT switch combines the advantages of both the MOSFET and bipolar junction transistor (BJT) to achieve high efficiency, fast switching speed and high voltage capability.

The IGBT switch consists of three terminals: the emitter, collector and gate. The emitter and collector are connected to the main circuit, while the gate is connected to a control signal. When a voltage is applied to the gate, it creates an electric field that allows or blocks the flow of current between the emitter and collector. This enables the IGBT switch to turn on and off rapidly, controlling the flow of power through the circuit.

The IGBT switch has a low on-state resistance, which reduces power loss and increases efficiency. It also has a fast switching speed, allowing it to rapidly turn on and off in response to changing signals. The special feature of IGBT switch is its ability to handle high voltage and current levels without overheating or breaking down.



Fig(1.8): IGBT Switch

Table (1.5): Specifications of IGBT switch (CM1200HB-66H)

Collector-Emitter voltage	3.3 kV max.
Collector current	1.2 kA max.
Pulsed collector current	2.4 kA max.
Collector cut-off current	15 mA max.
Gate-Emitter Voltage	± 20 V
Turn ON delay time	1.6 µs
Turn ON rise time	2 µs
Turn OFF delay time	2.5 µs
Turn OFF fall time	1 µs

1.6) **Trigger Generation System :**

It is the important part of Pulse modulator. It generates repetitive trigger pulses to the drivers of Thyratron / IGBT switches in the Pulse modulator. If there are more than one switches used in the Modulator, it provides synchronized and low jitter multiple pulses for each switch. A high power pulse modulator is a very strong source of conductive noise so the trigger generator system should be immune to this noise. Hence the electrical trigger pulses from trigger source are sent to the drivers using optical fibers. The trigger generator system is also required to be operated in two modes – Internal and External. In internal mode, it generates its own pulses. In external mode, an external trigger pulse is used to generate synchronized pulses required for all the parts of modulator.

The pulse parameters of a typical high voltage pulse generated in a pulse modulator are shown in figure and explained also below-

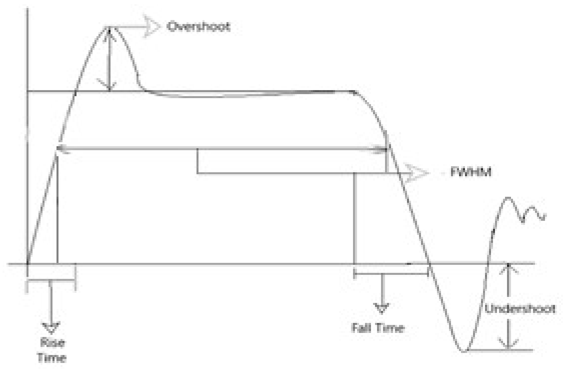


Fig (1.9): Typical High Voltage Pulse

Pulse Parameters

- **Rise Time**: In electronics, when describing a voltage or current step function, rise time is the time taken by a signal to change from a specified low value (generally 10%) to a specified high value (generally 90%).
- **Fall Time**: Fall time is the duration between the instants where the falling transition of each pulse crosses from the upper to the lower reference levels i.e. (90% to 10%).
- **FWHM**: The pulse duration is defined as a full width at half maximum (FWHM), that is, the width of the time interval within which the power is at least half the peak power.
- **Overshoot and Undershoot**: Overshoot occurs on rising and falling edges when the waveform exceeds the desired value. Undershoot is when they are lower than the final value.

1.7) **Need of Digital Control of power electronic systems**

- It provides better accuracy, flexibility, and reliability, which are essential for the efficient operation of power electronic systems.
- Digital control systems can process information more quickly and accurately than analog systems, leading to better control of power electronic systems.
- Digital systems easily adapt to changes in power electronic control needs. This flexibility allows for more efficient use of power electronic systems and reduces the need for costly hardware modifications.
- Digital control systems monitor power electronics in real-time, detecting issues early to prevent failure and reduce downtime, thus improving reliability.
- Software can be used to change hardware design and make interfaces between hardware structure.
- Digital Power Management IC : Controller, ADC as interface are integrated in single IC only inductor, capacitor outside Board. So number of components reduce as compares to Analog Board.
- Rapid Prototyping ; As multiple topologies can be implement in same chip , Product Development time reduces.

Comparison of various digital control systems

1.8) **Digital control hardware – PLC, FPGA, MICROCONTROLLER**

FPGA	Microcontroller	PLC
It is based on hardware connection or wiring through programming	It is based on software programming to execute the instruction.	PLC is made up of both hardware and software components.
FPGAs can be reprogrammed to adapt to changing requirements or to fix errors without the need for costly hardware changes.	Microcontrollers are highly efficient and cost-effective for their intended applications.	PLCs typically have input and output modules that allow them to interact with sensors, actuators, and other devices in the system they are controlling
FPGAs are integrated circuits that can be programmed to perform a wide range of functions, from simple logic operations to complex digital signal processing tasks	Microcontrollers are small, self-contained computers that are used in a variety of embedded systems, such as appliances, automobiles, and medical devices.	PLCs are specialized computers used to control industrial processes and machines

1.9) **Difference between FPGA and Microcontroller**

FPGA	Microcontroller
It is based on hardware connection or wiring through programming	It is based on software programming to execute the instruction.
It supports Hardware Description Language.	It supports C, C++, C#, and many more.
Processing power is high.	Processing power is low.
It require more skill to use.	Easy to use.
It is costly.	They are affordable.

1.10**)** **Difference between PLC and Microcontroller**

PLC	Microcontroller
PLC is made up of both hardware and software components	It is based on software programming to execute the instruction.
Microcontrollers are small, self-contained computers that are used in a variety of embedded systems, such as appliances, automobiles, and medical devices.	PLCs are specialized computers used to control industrial processes and machines
PLC Programming is Ladder Type Programming & it's easy.	Microcontroller Programming depends on Hand Coding Program

Chapter 2

STUDY OF VERILOG PROGRAMMING LANGUAGE

Verilog is a hardware description language that is used to model and simulate digital circuits. It was developed in the 1980s by Phil Moorby and Prabhu Goel at Gateway Design Automation. Since then, it has become one of the most widely used HDLs in the industry.Verilog is a powerful language that allows designers to describe complex digital systems using a concise syntax. It supports both behavioral and structural modeling styles, which makes it suitable for a wide range of applications. Verilog supports a variety of data types, including integers, real numbers, and arrays. It also provides a set of operators for performing arithmetic, logical, and bitwise operations on these data types.

One of the unique features of Verilog is its support for time-based simulation. This means that designers can specify the timing behavior of their circuits, which is critical for ensuring correct operation in real-world scenarios.

In Verilog, a module is a self-contained unit of code that represents a functional block in a digital system. Modules can be instantiated multiple times to create hierarchical designs. Verilog also supports parameterized modules, which allow designers to create reusable components that can be customized for different applications. This feature promotes modularity and reduces design time and effort. It is a text-based language that allows designers to create, simulate and debug their designs. Verilog is used in a variety of applications, from FPGAs to ASICs to embedded systems.

2.1) **Verilog Syntax**

Verilog is a human-readable language and has a straightforward syntax. It is based on C and shares many of the same concepts, such as variables, data types, and functions. Verilog also has several unique constructs, such as modules and tasks, which are used to create digital logic circuits. In Verilog, data types are used to define the type of data that can be stored in a variable.

The most commonly used data types in Verilog are wire, reg, integer, and parameter. Each data type has its own properties and uses, making it important to understand their differences.

i) **Wire Data Type in Verilog**

Wire data type is used to represent continuous signals in Verilog. It is used to connect different components in a circuit and transmit values between them. Wire data type only supports assignment using continuous assignments or module instances. It cannot be assigned a value inside an always block or initial block.

Example-

```
module my_module(input A, input B, output Y);
  wire C;
  wire D;

  and(C, A, B);
  or(D, C, B);

  // Output
  assign Y = D;
endmodule
```

ii) **Register Data Type in Verilog**

Register data type is used to store values in Verilog. It is used to represent variables that can be changed during simulation. Register data type can be assigned a value inside an always block or initial block. It is also used for sequential logic and storing state information in a circuit

Example-

```
module my_module(input clk, input reset, input data, output reg output_data);
  reg [7:0] internal_register; // 8-bit register
  always @(posedge clk, posedge reset)
     begin
    if (reset)
   begin
      internal_register <= 8'b0; // Reset the register to 0
    end
   else
   begin
      internal_register <= data; // Load data into the register
    end
  end


 always @(posedge clk) begin
    output_data <= internal_register; // Output the register value
  end
endmodule
```

iii) Integer Data Type in Verilog

Integer data type is used to represent whole numbers in Verilog. It is used for arithmetic operations and counting.

Integer data type can be signed or unsigned and can have a width specified by the user. It is commonly used for loop counters and indexing arrays.

Example-

```
module my_module(input integer A, input integer B, output integer Y);
  integer sum;
  always @* begin
    sum = A + B;
    Y = sum;
  end
endmodule
```

2.2) **Verilog Simulation**

Verilog can be used to simulate digital logic circuits. This allows designers to test their designs before they are implemented in hardware. Verilog simulations can be used to verify the correctness of a design, as well as to identify potential problems.

Verilog simulations are also used to optimize designs. By running simulations with different parameters, designers can determine the best design for their application. Verilog simulations are a powerful tool for digital logic design.

2.3) **Types of Modelling in Verilog**

i) Behavioral Modeling in Verilog

Behavioral modeling is a style of modeling in Verilog that focuses on the functionality of a digital system, rather than its implementation details. It uses high-level constructs such as always blocks and case statements to describe the behavior of a circuit. Behavioral modeling is particularly useful during the early stages of the design process, when designers are still exploring different architectures and algorithms. It allows them to quickly prototype and test their ideas before committing to a specific implementation.

```
Ex- 6 bit counter
`timescale 1us/1us
module counter(input clk,input rst,output reg[5:0] out);
always @(posedge clk)
 begin
 if(rst)
 out <= 0;
 else if(out>=52)
 out<=0;
 else
 out <= out + 1;
 end
 endmodule
```

```
// Testbench
`include “counter.v”
`timescale 1us/1us
module counter_tb;
reg clk;
reg rst;
wire[5:0] out;
counter c0(clk,rst,out);
always
#5 clk = ~clk;
initial
 begin
 $dumpfile(“counter_dump.vcd”);
   $dumpvars;
   $monitor($time,”clk=%b rst=%b,out=%b”,clk,rst,out);
    clk <= 0;
    rst <=0;
    # 20 rst <= 1;
    # 20 rst <= 0;
    #700 $finish;
    end
endmodule
```

ii). Gate level Modelling in Verilog

Gate level modeling is used to implement the lowest-level modules in a design, such as multiplexers, full-adders, and so on. Verilog provides gate primitives for all basic gates.Gate level odeling offers several benefits over other design methods. One advantage is that it allows for precise control over the circuit’s behavior, making it ideal for optimizing performance or reducing power consumption. Additionally, gate level odeling enables designers to create custom circuits that meet specific requirements, rather than relying on pre-built components. This flexibility can lead to more efficient and cost-effective designs.

In Verilog, gate level odeling involves defining a module that describes the circuit using a list of gates and their connections. The module includes input and output ports, which are used to connect the circuit to other modules or external devices.

```
Ex- 1-bit Full Adder

module Fulladder(A,B,Cin,Sum,Cout);
input A,B,Cin;
output  Sum,Cout;
 xor (Sum, A, B, Cin);
 and  G2(Signal1,A,B);
 and  G3(Signal2,B,Cin);
 and  G4(Signal3,Cin,A);
 or    G5(Cout,Signal1,Signal2,Signal3);
endmodule
```

```
// Testbench
`include “Fulladder.v”
`timescale 1us/1us
module Fulladder_tb;
reg A,B,Cin;
wire Sum,Cout;
Fulladder test(A,B,Cin,Sum,Cout);
initial
    begin
    $dumpfile(“Fulladder_dump.vcd”);
    $dumpvars;
    $monitor($time,”A=%b,B=%b Cin=%b
Sum=%b,Cout=%b”,A,B,Cin,Sum,Cout);
   #10 A=0;B=0;Cin=0;
   #10 A=0;B=1;Cin=0;
   #10 A=0;B=0;Cin=1;
   #10 A=1;B=1;Cin=0;
   #10 A=1;B=0;Cin=1;
   #10 A=1;B=1;Cin=1;
   #10 A=0;B=0;Cin=0;
   #10 $finish;
    end
  endmodule
```

iii). Structural Modeling in Verilog

Structural modeling is a style of modeling in Verilog that focuses on the physical implementation of a digital system, using low-level constructs such as gates and flip-flops. It is typically used for final implementation and verification of a design. Structural modeling allows designers to precisely control the layout and connectivity of their circuits, which is essential for meeting performance and power requirements. However, it can be more time-consuming and error-prone than behavioral modeling.

```
Ex- 4 bit adder
`include "Fulladder.v"
module fulladder_4bit( A,B,Cin,Sum,Carry);
input [3:0] A,B;
input Cin;
output [3:0] Sum ,Carry;
Fulladder bit0(A[0],B[0],Cin,Sum[0],Carry[0]);
Fulladder bit1(A[1],B[1],Carry[0],Sum[1],Carry[1]);
Fulladder bit2(A[2],B[2],Carry[1],Sum[2],Carry[2]);
Fulladder bit3(A[3],B[3],Carry[2],Sum[3],Carry[3]);
endmodule
```

```
 //testbench

 `include "fulladder_4bit.v"
 `timescale 1us/1us
 module fulladder_4bit_tb;
 reg [3:0] A,B;
 reg Cin;
wire [3:0] Sum, Carry;
fulladder_4bit test( A,B,Cin,Sum,Carry);
initial
 begin
   $dumpfile("fulladder_4bit_dump.vcd")
  $dumpvars;
   $monitor($time,"A=%b,B=%b,Cin=%b,Carry=%b,Sum=%b"
   ,A,B,Cin,Carry,Sum);
  #10 A=4'b1001;B=4'b0110;Cin=0;
  #10 A=4'b1001;B=4'b0110;Cin=0;
  #10 A=4'b1001;B=4'b0110;Cin=1;
 #10 $finish;
  end
 endmodule
```

Chapter 3

Study of FPGA architecture

An FPGA (Field Programmable Gate Array) is a type of integrated circuit that can be programmed to perform specific tasks. It is made up of configurable logic blocks and programmable interconnects, which allow users to customize the chip to suit their needs. FPGAs are widely used in applications such as digital signal processing, communications, and embedded systems. They are also used in consumer electronics, automotive, aerospace, and military applications. FPGAs are also more cost-effective than ASICs, since the cost of designing and manufacturing an FPGA is much lower than that of an ASIC. This makes them ideal for prototyping and small-scale production runs.

The FPGA design process involves a number of steps, including hardware design, software design, and verification. The hardware design involves creating a schematic and a netlist, while the software design involves creating a hardware description language (HDL) code. The verification step involves testing the design to ensure it meets the specified requirements. Once the design is verified, it is ready to be implemented on the FPGA. This involves programming the FPGA with the design, which can be done using a variety of tools, such as Xilinx ISE or Altera Quartus II.

FPGAs can be programmed using a variety of languages, such as Verilog, VHDL, and SystemVerilog. Verilog and VHDL are hardware description languages (HDLs), while System Verilog is a hardware verification language (HVL). Each language has its own advantages and disadvantages, so it is important to choose the language that best suits the application. In addition to HDLs, FPGAs can also be programmed using C/C++, Java, and other high-level languages. These languages are easier to learn and use than HDLs, but they are not as efficient or as powerful as HDLs.

FPGAs began as competitors to CPLDs in the implementation of glue logic for printed circuit boards. FPGAs took on additional functions as their size, capabilities, and speed increased, to the point where some are now marketed as full systems on chips. (SoCs). With the introduction of dedicated multipliers into FPGA architectures in the late 1990s, applications that had previously relied solely on digital signal processor hardware (DSPs) began to incorporate FPGAs.

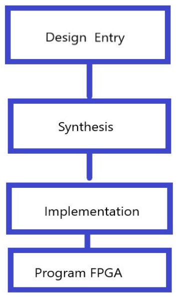


Fig (3.1): FPGA DESIGN FLOW

3.1) **ARTY-7**

The Arty A7 FPGA board is a powerful and versatile development platform that allows designers to create custom digital circuits with ease. It is built around the Xilinx Artix-7 FPGA, which features a high-performance architecture and a range of advanced features. It features a Xilinx Artix-7 FPGA with a capacity of 15,850 logic cells and 240 KB of block RAM, making it ideal for a wide range of applications.

With the Arty A7 board, designers can implement a wide range of applications, from simple logic circuits to complex systems with multiple processors and peripherals. The board comes with a range of built-in peripherals, including Ethernet, USB, and HDMI interfaces, making it easy to connect to other devices and networks.It has 16 user switches and LEDs, 4-digit seven-segment display, two Pmod connectors, and an Arduino/chipKIT shield connector. It also has a microSD card slot for data storage and a USB-UART bridge for easy programming and debugging. The Arty A7 FPGA board can be used in a variety of applications, from simple LED blinking projects to complex digital signal processing algorithms. Because FPGAs are highly customizable, they can be used to implement a wide range of functions, including image and audio processing, cryptography, and control systems.

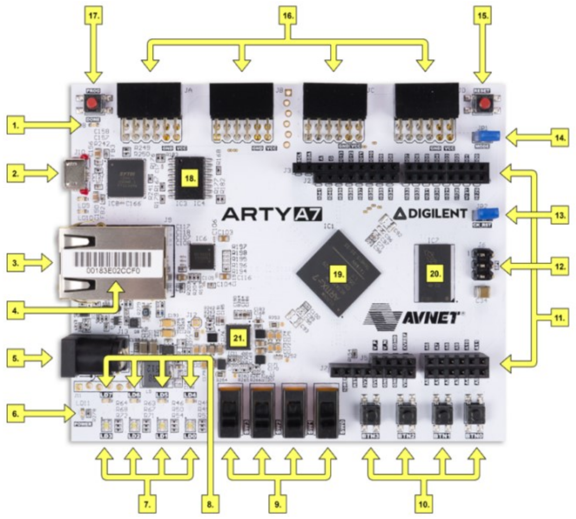


Fig (3.2): ARTY 7 Board

Table(3.1): PIN DESCRIPTION OF ARTY 7 FPGA BOARD

Pin Number	**Description**	**Pin Number**	**Description**	**Pin Number**	**Description**
1	FPGA programming DONE LED	8	User LEDs	15	chipKIT processor reset
2	Shared USB JTAG / UART port	9	User slide switches	16	Pmod connectors

3	Ethernet connector	10	User push buttons	17	FPGA programming reset button
4	MAC address sticker	11	Arduino/chipKIT shield connectors	18	SPI flash memory
5	Power jack for optional external supply	12	Arduino/chipKIT shield SPI connector	19	Artix FPGA
6	Power good LED	13	chipKIT processor reset jumper	20	Micron DDR3 memory
7	User RGB LEDs	14	FPGA programming mode	21	Dialog Semiconductor DA9062 power supply

3.2) Memory

Arty A7 contains two external memories: DDR3L SDRAM and a 128Mb (16MB) non-volatile serial Flash device. The DDR3L module is connected to the FPGA using the industry standard interface. The serial Flash is on a dedicated quad-mode (x4) SPI bus.

Table (3.2): DDR3L Specifications

Setting	Value
Memory type	DDR3 SDRAM
MAX.CLOCK PERIOD	667Mbps data rate
MEMORY VOLTAGE	1.35V
DATA WIDTH	16
INTERNAL IMPEDENCE	50 OHMS

The Arty A7 FPGA board comes equipped with 256MB of DDR3 SDRAM memory, providing high-speed data transfer capabilities for a variety of applications.This memory can be used to store program code, data, and other critical information, allowing for fast and efficient execution of complex algorithms and processes.

In addition to DDR3 SDRAM, the Arty A7 FPGA board also includes non-volatile memory in the form of QSPI Flash memory.This memory provides a reliable and secure way to store critical data, even when power is lost or the system is shut down.

The Arty A7 FPGA board features a range of memory interfaces, including DDR3 SDRAM, QSPI Flash, and SD/MMC card slots.These interfaces provide developers with a flexible and versatile platform for implementing a wide range of applications and use cases.

3.3) **Oscillators/Clocks**

There are several types of oscillators that can be generated on the Arty A7 board, including ring oscillators, relaxation oscillators, and crystal oscillators. The Arty A7 includes a single 100 MHz crystal oscillator connected to pin E3 (E3 is a MRCC input on bank 35). The input clock can drive MMCMs or PLLs to generate clocks of various frequencies and with known phase relationships that may be needed throughout a design. Some rules restrict which MMCMs and PLLs may be driven by the 100 MHz input clock

The Arty A7 board can generate clocks with a wide range of frequencies, allowing for precise control over the timing of a system. These clocks can be used to drive other components on the board or to synchronize external devices.

3.4) **Basic I/O**

The four push buttons are "momentary" switches that normally generate a low output when they are at rest, and a high output only when they are pressed. Slide switches generate constant high or low inputs depending on their position. The red reset button labeled "RESET" generates a high output when at rest and a low output when pressed. The RESET button is intended to be used in Microblaze designs to reset the processor, but you can also use it as a general purpose push button

One of the main advantages of the Basic I/O Arty 7 FPGA Board is its flexibility. It can be used for a wide range of projects, from simple LED blinking to complex image processing. Additionally, the board is compatible with various software tools, such as Vivado Design Suite and Xilinx SDK, which makes it easy to develop and test applications.

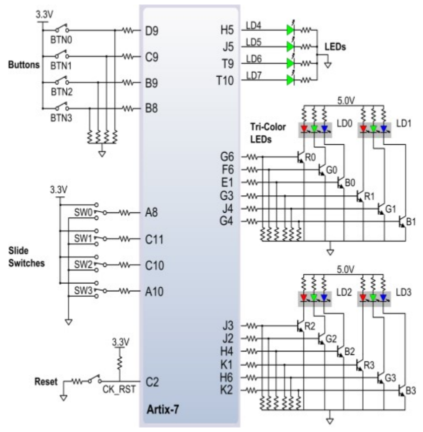


Fig (3.3): Basic I/0 of Arty 7 diagram

3.5) **Counters**

Counters are a fundamental building block in digital circuits. They are used to count the number of clock cycles, events or sequences that occur within a given period of time. In Verilog, counters can be implemented using either combinational or sequential logic. Combinational counters use logic gates to implement the counter, while sequential counters use flip-flops.

3.5) Combinational counters are made up of logic gates that are connected together to form a counting circuit. These counters are faster than sequential counters because they do not have any memory elements like flip-flops.

One example of a combinational counter is the ripple counter, which uses a series of D flip-flops connected in a chain. Each flip-flop is triggered by the output of the previous flip-flop, resulting in a binary count sequence.

Sequential counters use flip-flops to store the current count value. These counters are slower than combinational counters because of the delay introduced by the flip-flops.

One example of a sequential counter is the Johnson counter, which uses a shift register to generate a pseudo-random sequence of bits. The sequence is then decoded to produce a binary count sequence.

Synchronous Counters

Synchronous counters are a type of sequential counter that use a common clock signal to synchronize the operation of all the flip-flops in the counter.One advantage of synchronous counters is that they are less prone to glitches and race conditions compared to asynchronous counters. However, they require more hardware to implement the clock distribution network.

Asynchronous Counters

Asynchronous counters are a type of sequential counter that do not use a common clock signal. Instead, each flip-flop is triggered by the output of the previous flip-flop. One disadvantage of asynchronous counters is that they are more prone to glitches and race conditions compared to synchronous counters. However, they require less hardware to implement the clock distribution network.

Chapter 4

Introduction to Xilinx Vivado Software

Xilinx Vivado is a powerful software suite used for designing and developing programmable logic devices. It offers a wide range of features and tools that allow engineers to create complex designs with ease.

Vivado includes a range of debugging features that make it easy to identify and fix issues in designs. These include interactive logic analyzers, waveform viewers, and system-level debuggers. Vivado also offers a powerful integrated logic analyzer that allows designers to capture and analyze signals in real-time, providing valuable insights into system behavior.

As technology continues to advance, the demand for powerful, flexible programmable logic devices will only increase. Xilinx Vivado is poised to meet this demand, offering a comprehensive suite of tools and features that enable designers to create innovative, cutting-edge designs. With its intuitive user interface, powerful optimization features, and robust collaboration and debugging tools, Vivado is the clear choice for FPGA design and development now and in the future.

4.1) **Creating a New Project in Xilinx Vivado**

To get started with Xilinx Vivado, you need to create a new project. This can be done by selecting 'File' > 'New Project' from the main menu. You will then be prompted to specify the project name, location, and target device.

Once you have created a new project, you can start adding design sources, constraints, and simulation models to it. Xilinx Vivado provides a wide range of design entry options, including schematic capture, HDL coding, and IP integration.

4.2) **Designing and Simulating Digital Circuits in Xilinx Vivado**

Xilinx Vivado provides a powerful suite of tools for designing and simulating digital circuits. These tools include a waveform viewer, a logic analyzer, a power estimator, and a timing analyzer. They allow designers to visualize and analyze the behavior of their circuits at various levels of abstraction.

In addition to these built-in tools, Xilinx Vivado also supports third-party simulators, such as ModelSim and QuestaSim. These simulators provide advanced features, such as mixed-language support, assertion-based verification, and coverage analysis.

4.3) **Implementing and Debugging FPGA-Based Designs in Xilinx Vivado**

Xilinx Vivado provides a comprehensive set of features for implementing and debugging FPGA-based designs. These features include synthesis, place-and-route, bitstream generation, and programming. They allow designers to optimize their designs for performance, area, and power consumption.

To debug their designs, designers can use the integrated logic analyzer, the on-chip debug (OCD) module, or the JTAG interface. These tools enable designers to trace signals, set breakpoints, and inspect the internal state of their circuits in real-time.

4.4) **Simulation and Verification in Vivado**

Vivado includes a powerful simulation and verification environment that allows designers to test and debug their designs before implementation. The simulator includes support for both behavioral and timing simulations, allowing designers to verify the correctness and performance of their designs under a variety of conditions.

In addition, Vivado includes advanced verification features such as formal verification and assertion-based verification, which can help identify and eliminate potential design errors early in the development process. This can save significant time and resources compared to traditional manual verification methods.

4.5) **Timing Analysis in Vivado**

Timing analysis is a critical step in the design process, ensuring that all signals in the design arrive at their destinations within the required timing constraints. Vivado includes a comprehensive timing analysis tool that can analyze the timing characteristics of the entire design, from the highest-level modules down to individual logic cells.

The timing analysis tool includes advanced features such as path tracing and slack analysis, allowing designers to identify and address timing violations quickly and efficiently. This can help ensure that the final design meets all timing requirements and operates correctly in the target system.

Chapter 5

Introduction to State Machines in Verilog

State machines are a fundamental concept in digital circuit design, and they play a critical role in the development of complex systems. In essence, a state machine is a model that describes how a system behaves over time. It consists of a set of states, transitions between those states, and actions that occur when a transition occurs.

In Verilog, state machines are implemented using a combination of combinational and sequential logic. The combinational logic is used to determine the next state based on the current state and inputs, while the sequential logic is used to store the current state and update it on each clock cycle. By carefully designing the state machine, it is possible to create highly efficient and reliable circuits that can perform complex tasks.

5.1) **Types of State Machines in Verilog**

There are two main types of state machines Mealy machines and Moore machines. Mealy machines produce outputs based on both the current state and inputs, while Moore machines produce outputs based only on the current state. Both types of machines have their advantages and disadvantages, and the choice between them depends on the specific requirements of the system being designed.

Another important distinction is between synchronous and asynchronous state machines. Synchronous state machines use a clock signal to synchronize the state updates, while asynchronous state machines do not rely on a clock and instead use feedback loops to update the state. Asynchronous state machines are more complex to design and analyze, but they offer some advantages in terms of speed and power consumption.

5.2) **Designing State Machines in Verilog**

The process of designing a state machine in Verilog involves several steps. First, the behavior of the system must be analyzed to identify the states and transitions that are required. This is typically done using a state diagram, which provides a visual representation of the system's behavior. Once the states and transitions have been identified, the state machine can be implemented in Verilog using a combination of combinational and sequential logic.

One important consideration when designing state machines is to ensure that they are free from race conditions and other hazards. Race conditions occur when two or more signals arrive at the same time and cause unpredictable behavior in the circuit. Hazards occur when changes in one input cause temporary glitches in the output. By carefully designing the state machine and using appropriate techniques such as edge detection and synchronization, these issues can be avoided.

5.3) **Applications of State Machines in Verilog**

State machines have a wide range of applications in digital circuit design, including control systems, communication protocols, and data processing. For example, a state machine can be used to control the operation of a robot arm, where the states represent the different positions of the arm and the transitions represent the movements between those positions. Similarly, a state machine can be used to implement a communication protocol, where the states represent the different stages of the communication process.

State machines are also used extensively in software development, particularly in the field of compilers and interpreters. In this context, state machines are used to parse and interpret programming languages, with the states representing the different syntactic constructs of the language. By carefully designing the state machine, it is possible to create highly efficient and reliable compilers and interpreters that can handle complex programs.

Example of State Machine:

```
`timescale 1us / 1us
module pm_trigger(output trigger_out,input reset);
reg trigger_out;
reg clk=0;
always
#1 clk=~clk;

reg [1:0] state=2'b0;
reg [12:0] count=13'b0;
always@(posedge clk)
begin
if(reset==1)
begin
    state<=0;
    trigger_out<=0;
end
else
    case(state)
    2'b00   : // wait state
    begin
        trigger_out<=0;
         count<=count+1;
        if(count>=50)
        begin
          state<=state+1;
          count<=0;
         end
    end
    2'b01   :// trigger on state , gives high pulse for 20 us
    begin
        trigger_out<=1;
         count<=count+1;
        if(count>=9)
        begin
          state<=state+1;
          count<=0;
         end
    end
    2'b10   :// triggen off state , gives high pulse for 10000 us
    begin
        trigger_out<=0;
         count<=count+1;
        if(count>=50)
        begin
          state<=1;
          count<=0;
         end
    end
    endcase
end
endmodule
```

```
//Testbench
`include "pm_trigger.v"
`timescale 1us / 1us
module pm_trigger_tb;
    wire trigger_out;
    reg reset=1'b0;
    pm_trigger Dut(trigger_out,reset);

    initial
        begin
        $monitor($time,"reset=%b,output=%b",reset,trigger_out);
        $dumpfile("pm_trigger_tb.vcd");
        $dumpvars;
        #600 reset=1'b1;
        #200 reset=1'b0;
        #10000 $finish;
        end
    endmodule
```

Simulation Output:

```
VCD info: dumpfile pm_trigger_tb.vcd opened for output.
                   0reset=0,output=x
                   1reset=0,output=0
                 103reset=0,output=1
                 123reset=0,output=0
                 225reset=0,output=1
                 245reset=0,output=0
                 347reset=0,output=1
                 367reset=0,output=0
                 469reset=0,output=1
                 489reset=0,output=0
                 591reset=0,output=1
                 600reset=1,output=1
                 601reset=1,output=0
                 800reset=0,output=0
                 893reset=0,output=1
                 913reset=0,output=0
                1015reset=0,output=1
                1035reset=0,output=0
                1137reset=0,output=1
                1157reset=0,output=0
```

```
1259reset=0,output=1
1279reset=0,output=0
1381reset=0,output=1
1401reset=0,output=0
1503reset=0,output=1
1523reset=0,output=0
1625reset=0,output=1
1645reset=0,output=0
1747reset=0,output=1
1767reset=0,output=0
1869reset=0,output=1
1889reset=0,output=0
1991reset=0,output=1
2011reset=0,output=0
2113reset=0,output=1
2133reset=0,output=0
2235reset=0,output=1
2255reset=0,output=0
2357reset=0,output=1
2377reset=0,output=0
2479reset=0,output=1
2499reset=0,output=0
2601reset=0,output=1
2621reset=0,output=0
2723reset=0,output=1
2743reset=0,output=0
2845reset=0,output=1
2865reset=0,output=0
2967reset=0,output=1
2987reset=0,output=0
3089reset=0,output=1
3109reset=0,output=0
3211reset=0,output=1
3231reset=0,output=0
3333reset=0,output=1
3353reset=0,output=0
3455reset=0,output=1
3475reset=0,output=0
3577reset=0,output=1
3597reset=0,output=0
3699reset=0,output=1
3719reset=0,output=0
3821reset=0,output=1
3841reset=0,output=0
3943reset=0,output=1
3963reset=0,output=0
4065reset=0,output=1
4085reset=0,output=0
4187reset=0,output=1
4207reset=0,output=0
4309reset=0,output=1
4329reset=0,output=0
4431reset=0,output=1
4451reset=0,output=0
4553reset=0,output=1
4573reset=0,output=0
4675reset=0,output=1
```

```
4695reset=0,output=0
4797reset=0,output=1
4817reset=0,output=0
4919reset=0,output=1
4939reset=0,output=0
5041reset=0,output=1
5061reset=0,output=0
5163reset=0,output=1
5183reset=0,output=0
5285reset=0,output=1
5305reset=0,output=0
5407reset=0,output=1
5427reset=0,output=0
5529reset=0,output=1
5549reset=0,output=0
5651reset=0,output=1
5671reset=0,output=0
5773reset=0,output=1
5793reset=0,output=0
5895reset=0,output=1
5915reset=0,output=0
6017reset=0,output=1
6037reset=0,output=0
6139reset=0,output=1
6159reset=0,output=0
6261reset=0,output=1
6281reset=0,output=0
6383reset=0,output=1
6403reset=0,output=0
6505reset=0,output=1
6525reset=0,output=0
6627reset=0,output=1
6647reset=0,output=0
6749reset=0,output=1
6769reset=0,output=0
6871reset=0,output=1
6891reset=0,output=0
6993reset=0,output=1
7013reset=0,output=0
7115reset=0,output=1
7135reset=0,output=0
7237reset=0,output=1
7257reset=0,output=0
7359reset=0,output=1
7379reset=0,output=0
7481reset=0,output=1
7501reset=0,output=0
7603reset=0,output=1
7623reset=0,output=0
7725reset=0,output=1
7745reset=0,output=0
7847reset=0,output=1
7867reset=0,output=0
7969reset=0,output=1
7989reset=0,output=0
8091reset=0,output=1
8111reset=0,output=0
```

```
                8213reset=0,output=1
                8233reset=0,output=0
                8335reset=0,output=1
                8355reset=0,output=0
                8457reset=0,output=1
                8477reset=0,output=0
                8579reset=0,output=1
                8599reset=0,output=0
                8701reset=0,output=1
                8721reset=0,output=0
                8823reset=0,output=1
                8843reset=0,output=0
                8945reset=0,output=1
                8965reset=0,output=0
                9067reset=0,output=1
                9087reset=0,output=0
                9189reset=0,output=1
                9209reset=0,output=0
                9311reset=0,output=1
                9331reset=0,output=0
                9433reset=0,output=1
                9453reset=0,output=0
                9555reset=0,output=1
                9575reset=0,output=0
                9677reset=0,output=1
                9697reset=0,output=0
                9799reset=0,output=1
                9819reset=0,output=0
                9921reset=0,output=1
                9941reset=0,output=0
               10043reset=0,output=1
               10063reset=0,output=0
               10165reset=0,output=1
               10185reset=0,output=0
               10287reset=0,output=1
               10307reset=0,output=0
               10409reset=0,output=1
               10429reset=0,output=0
               10531reset=0,output=1
               10551reset=0,output=0
               10653reset=0,output=1
               10673reset=0,output=0
               10775reset=0,output=1
               10795reset=0,output=0
pm_trigger_tb.v:15: $finish called at 10800 (1us)
PS C:\iverilog\bin> gtkwave
```

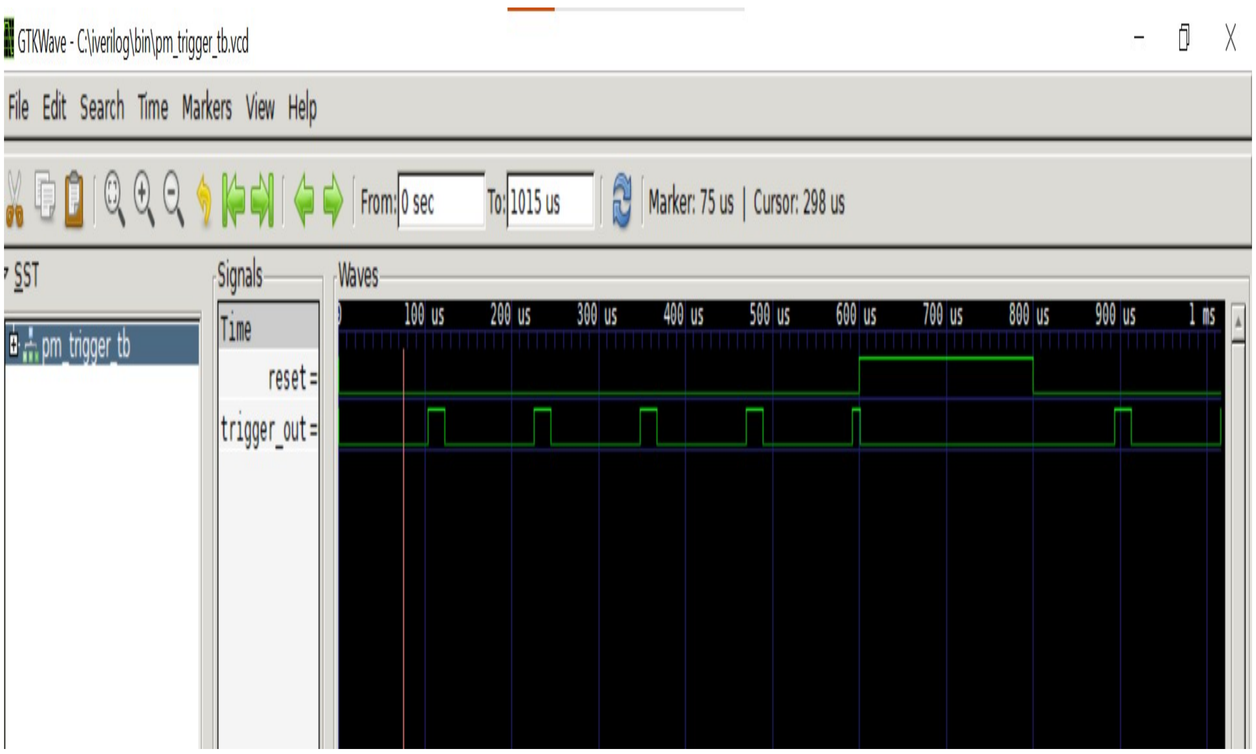


Fig (5): Simulation waveform of Pulse Modulation Trigger

Chapter 6

Implementation and Result

Implementation

Code of Arty 7

Constrain File describing Arty A7 Pins:- Constraint File describing Arty A7 Pins:- The Input/ Output variables used in Verilog module are mapped to pins in Arty – A7 Board as described in constraints file. :

```
# This file is a general .xdc for the Arty A7-35 Rev. D and Rev. E
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project


##for master_spi mode{to be able to store program in flash}
#set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
#set_property CONFIG_VOLTAGE 3.3 [current_design]
#set_property CFGBVS VCCO [current_design]
#set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
#set_property CONFIG_MODE SPIx4 [current_design]

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }];
#IO_L12P_T1_MRCC_35 Sch=gclk[100]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { CLK100MHZ }];

## Switches
set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33 } [get_ports { reset }];
#IO_L12N_T1_MRCC_16 Sch=sw[0]
#set_property -dict { PACKAGE_PIN C11   IOSTANDARD LVCMOS33 } [get_ports { switch[1] }];
#IO_L13P_T2_MRCC_16 Sch=sw[1]
#set_property -dict { PACKAGE_PIN C10   IOSTANDARD LVCMOS33 } [get_ports { switch[2] }];
#IO_L13N_T2_MRCC_16 Sch=sw[2]
#set_property -dict { PACKAGE_PIN A10   IOSTANDARD LVCMOS33 } [get_ports { switch[3] }];
#IO_L14P_T2_SRCC_16 Sch=sw[3]

## RGB LEDs
#set_property -dict { PACKAGE_PIN E1    IOSTANDARD LVCMOS33 } [get_ports { led0_b }];
#IO_L18N_T2_35 Sch=led0_b
#set_property -dict { PACKAGE_PIN F6    IOSTANDARD LVCMOS33 } [get_ports { led0_g }];
#IO_L19N_T3_VREF_35 Sch=led0_g
set_property -dict { PACKAGE_PIN G6    IOSTANDARD LVCMOS33 } [get_ports { p_danger_led }];
#IO_L19P_T3_35 Sch=led0_r
#set_property -dict { PACKAGE_PIN G4    IOSTANDARD LVCMOS33 } [get_ports { led1_b }];
#IO_L20P_T3_35 Sch=led1_b
#set_property -dict { PACKAGE_PIN J4    IOSTANDARD LVCMOS33 } [get_ports { led1_g }];
#IO_L21P_T3_DQS_35 Sch=led1_g
#set_property -dict { PACKAGE_PIN G3    IOSTANDARD LVCMOS33 } [get_ports { led1_r }];
#IO_L20N_T3_35 Sch=led1_r
```

```
#set_property -dict { PACKAGE_PIN H4    IOSTANDARD LVCMOS33 } [get_ports { led2_b }];
#IO_L21N_T3_DQS_35 Sch=led2_b
#set_property -dict { PACKAGE_PIN J2    IOSTANDARD LVCMOS33 } [get_ports { led2_g }];
#IO_L22N_T3_35 Sch=led2_g
#set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 } [get_ports { led2_r }];
#IO_L22P_T3_35 Sch=led2_r
#set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { led3_b }];
#IO_L23P_T3_35 Sch=led3_b
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { led3_g }];
#IO_L24P_T3_35 Sch=led3_g
#set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33 } [get_ports { led3_r }];
#IO_L23N_T3_35 Sch=led3_r

## USB-UART Interface
#set_property -dict { PACKAGE_PIN D10   IOSTANDARD LVCMOS33 } [get_ports { uart_rxd_out }];
#IO_L19N_T3_VREF_16 Sch=uart_rxd_out
#set_property -dict { PACKAGE_PIN A9    IOSTANDARD LVCMOS33 } [get_ports { uart_txd_in }];
#IO_L14N_T2_SRCC_16 Sch=uart_txd_in

## ChipKit Outer Digital Header
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { p_bouncap_charge  }];
#IO_L16P_T2_CSI_B_14          Sch=ck_io[0]
#set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { p_bouncap_discharge  }];
#IO_L18P_T2_A12_D28_14        Sch=ck_io[1]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { p_maincap_discharge  }];
#IO_L8N_T1_D12_14             Sch=ck_io[2]
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { p_CLK1MHZ  }];
#IO_L19P_T3_A10_D26_14        Sch=ck_io[3]
set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports { fault  }];
#IO_L5P_T0_D06_14             Sch=ck_io[4]
#set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { ck_io5  }];
#IO_L14P_T2_SRCC_14           Sch=ck_io[5]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { ck_io6  }];
#IO_L14N_T2_SRCC_14           Sch=ck_io[6]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { ck_io7  }];
#IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=ck_io[7]
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { ck_io8  }];
#IO_L11P_T1_SRCC_14           Sch=ck_io[8]
#set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports { ck_io9  }];
#IO_L10P_T1_D14_14            Sch=ck_io[9]
#set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { ck_io10 }];
#IO_L18N_T2_A11_D27_14        Sch=ck_io[10]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { ck_io11 }];
#IO_L17N_T2_A13_D29_14        Sch=ck_io[11]
#set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { ck_io12 }];
#IO_L12N_T1_MRCC_14           Sch=ck_io[12]
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { ck_io13 }];
#IO_L12P_T1_MRCC_14           Sch=ck_io[13]
```

Clock divider Module:

```
//this module will divide the 100MHz clk onboard arty A7 (max upto 1 Hz) by a factor of 2*div( div is a parameter
created below)

module clk_divisor(
    input CLK100MHZ,
    output clk_out
    );

    parameter div=50;
    reg [26 :0] count=27'b0;
    reg out=1'b0;
    assign clk_out=out;

    always@(posedge CLK100MHZ )
    begin
        if(count<(div-1))
        begin
            count<=count+1;
        end
        else if(count>=(div-1))
        begin
            out<=~out;
            count<=0;
        end
    end

endmodule
```

Main code for generating trigger of modulator:

```
module main(
    output wire p_maincap_discharge,
    output wire p_CLK1MHZ,
    input wire fault,
    input wire reset,
    output wire p_danger_led,
    input CLK100MHZ
    );



reg maincap_discharge=1'b0;

//p_<portname> indicates that <portname> variable is being sent out as a wire as a port
assign p_CLK1MHZ=CLK1MHZ;
assign p_maincap_discharge=maincap_discharge;
assign p_danger_led=danger_led;

wire CLK1MHZ;
```

```
reg [0:17] count=18'b0;
reg [0:1] state=2'b01;    //state of finite stae machine
reg [0:3] state1;
reg danger_led=1'b0;
clk_divisor DUT1(           //will generate a 1MHZ clock from internal 100MHz clock
.CLK100MHZ(CLK100MHZ),     //discussed in Appendix-4
.clk_out(CLK1MHZ)
);

parameter period_pulse=20;      //in us
parameter state1count=50;     //blank state after state 0
parameter state2count=10000;  //count corresponding to 10ms delay, for 100Hz trigerring

always@(posedge CLK1MHZ)
begin
   if( fault==0)
   begin
      state<=0;
      maincap_discharge<=0;
      danger_led<=1;
   end
   else
   begin
   case (state)
      2'b00:             //reset state
            begin
            danger_led<=1;
            if(reset==1)
               begin
               danger_led<=0;
               state<=state+1;
               end
             end
      2'b01:             //state to discharge main cpacacitor
         begin
         if(count<=period_pulse-2)
            begin
            maincap_discharge<=1;
            count<=count+1;
            end
         else
            begin
            count<=0;
            state<=state+1;
            maincap_discharge<=0;
            end
         end
      2'b10:         //blank state after state 1
         begin
         if(count<=state1count)
            begin
            maincap_discharge<=0;
            count<=count+1;
            end
         else
            begin
            maincap_discharge<=0;
            count<=0;
            state<=state+1;
            end
          end
       2'b11:        //state to generate time lag 10ms, corresponding to 100 Hz
          begin
          if(count<=state2count)
            begin
            maincap_discharge<=0;
            count<=count+1;
            end
          else
            begin
            maincap_discharge<=1;
            state<=2'b01; //return to state 0 for main capacitor discharging
            count<=0;
            end
          end

   endcase
   end
end

endmodule
```

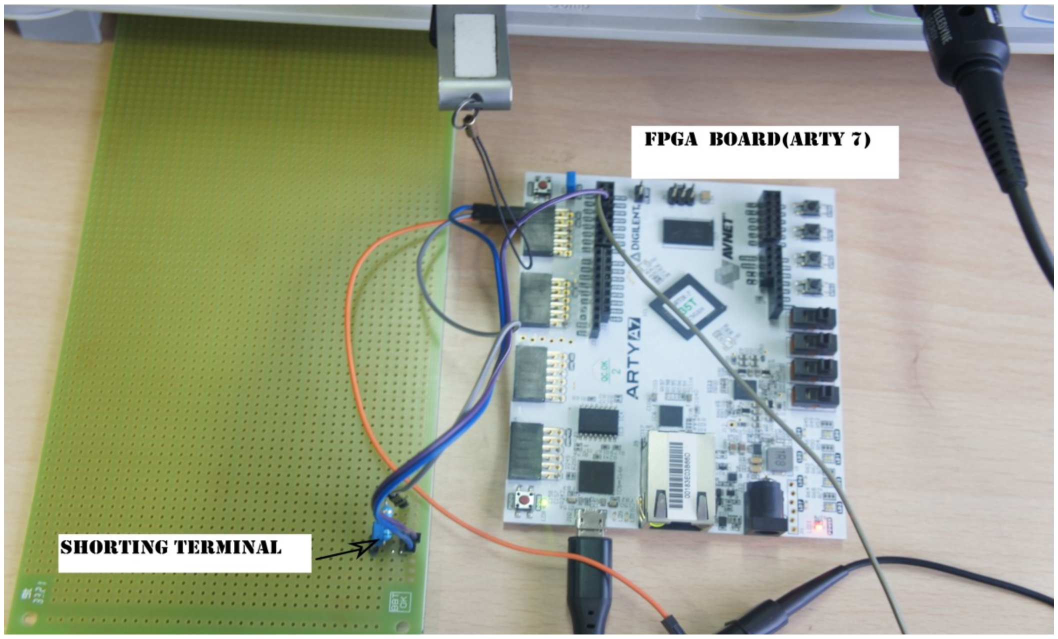
Result

Components used for Project:

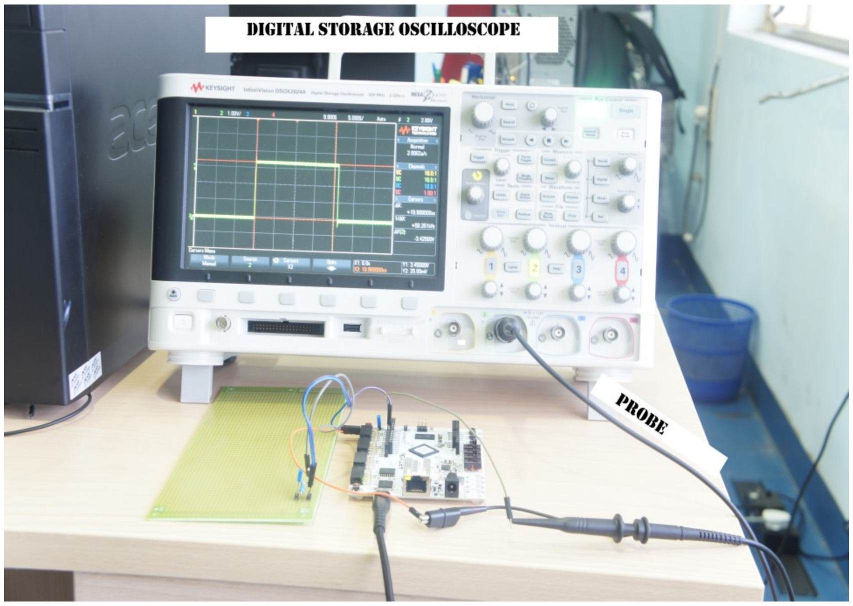
- FPGA Board Model :Arty 7 Make: Digilent
- Female Header
- Shorting header
- Digital Storage Oscilloscope Model:DSOX2024A Make: Keysight
- Resistor Value: 560kohm
- Jumper Wires
- Probes Model: PPO26 Make: TELEDYNE

Software:

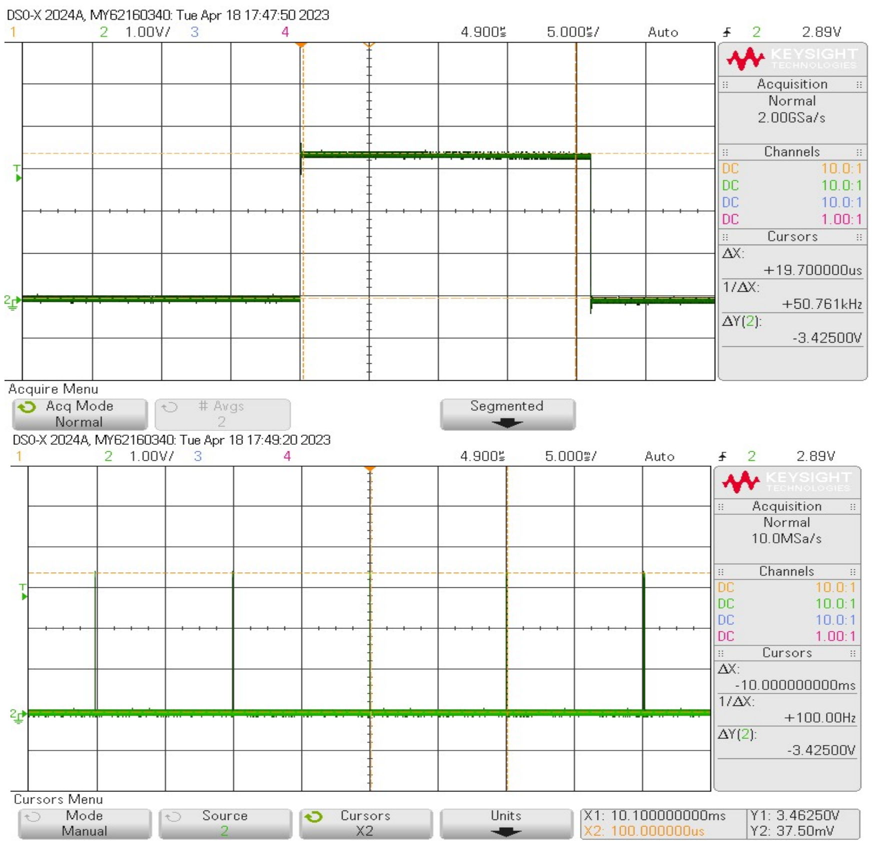
- Iverilog
- Vs code
- Vivado



Fig(6)1: Setup of FPGA Board



Fig(6)(2):FPGA Board connected with DSO



DSO-X 2024A, MY62160340: Tue Apr 18 17:47:50 2023

1 2 1.00V/ 3 4 4.900s 5.000s/ Auto 2 2.89V

Acquisition
Normal
2.00GSa/s

Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1

Cursors
ΔX:
+19.700000us
1/ΔX:
+50.761kHz
ΔY(2):
-3.42500V

Acquire Menu
Acq Mode Normal
Avgs 2
Segmented

DSO-X 2024A, MY62160340: Tue Apr 18 17:49:20 2023

1 2 1.00V/ 3 4 4.900s 5.000s/ Auto 2 2.89V

Acquisition
Normal
10.0MSa/s

Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1

Cursors
ΔX:
-10.000000000ms
1/ΔX:
+100.00Hz
ΔY(2):
-3.42500V

Cursors Menu
Mode Manual
Source 2
Cursors X2
Units
X1: 10.100000000ms
X2: 100.000000us
Y1: 3.46250V
Y2: 37.50mV

Fig(6):Simulation Waveforms