



**IIT MADRAS SUMMER FELLOWSHIP PROGRAMME  
(2019)**

**A STUDY OF DIGITAL CIRCUITS AND DESIGN FOR  
POWER ELECTRONIC APPLICATIONS**

**BY SURADHA SRINIVASAN IYER**

THIRD YEAR, DEPT. OF ELECTRONICS & TELECOMMUNICATION  
COLLEGE OF ENGINEERING, PUNE

**UNDER THE GUIDANCE OF DR. N. LAKSHMINARASAMMA**  
ASSOCIATE PROFESSOR, INDIAN INSTITUTE OF TECHNOLOGY, MADRAS

# **Contents**

## **PART A: FPGA BOARDS AND TESTING CRITERIA**

- 1. Abstract**
- 2. Problem Statement**
- 3. Objective**
- 4. Study**
  - i. About the Development Board**
  - ii. Identifying Parts of Designed Board**
  - iii. Comparison of 2 boards**
- 5. Testing Procedure & Results**
- 6. Proposed Testing & Conclusion**

## **PART B: VERILOG CODES FOR DIGITAL CIRCUITS**

- 1. Problem Statement & Objectives**
- 2. Tools & Hardware**
  - i. Verilog**
  - ii. Quartus**
  - iii. ATM code using RTL logic**
  - iv. Max V Board**
- 3. Precautions & Conclusion**
- 4. Appendix**
  - i. Part 0: Basic Verilog Logic Designs**
  - ii. Part I: Clock Division**
  - iii. Part II: PWM and control**

## **PART C: PCB DESIGN FOR SENSING CIRCUIT**

- 1. Problem Statement**
- 2. Objective**
- 3. Circuit**
  - i. Specifications**
  - ii. Schematic & Design**
  - iii. Cost Optimization**
- 4. Tool Used**
  - i. Eagle**
  - ii. PCB layout**
- 5. Precautions & Conclusion**

## **ACKNOWLEDGEMENTS**

# PART A

## STUDY AND TESTING OF INTEL MAX 10 FPGA BOARD

### 1. ABSTRACT

FPGA (Field Programmable Gate Array) boards can perform complex mathematical & logical operations at low power consumption than a DSP and are flexible, unlike ASICs. ASICs can be inexpensive and lower power than FPGAs when custom built and ordered in bulk, else FPGA boards make for good substitutes. This is a project understanding the parts of SOC FPGA boards like the Altera MAX 10 series and characterizing a board designed at IIT Madras with similar specifications. Certain suggestions have been made to test failures reported on the designed board (henceforth referred to as DB) and some possible failure parameters have been eliminated as well.

### 2. PROBLEM STATEMENT

A single DB was fabricated and plugged directly to a PC with a code that was tested on the Evaluation kit first. The DB had an issue with receiving code through the USB blaster II on the Quartus programmer. Evaluating if the problem is a software one or a hardware issue, and coming to conclusions on what the hardware issues are, if any.

### 3. OBJECTIVE

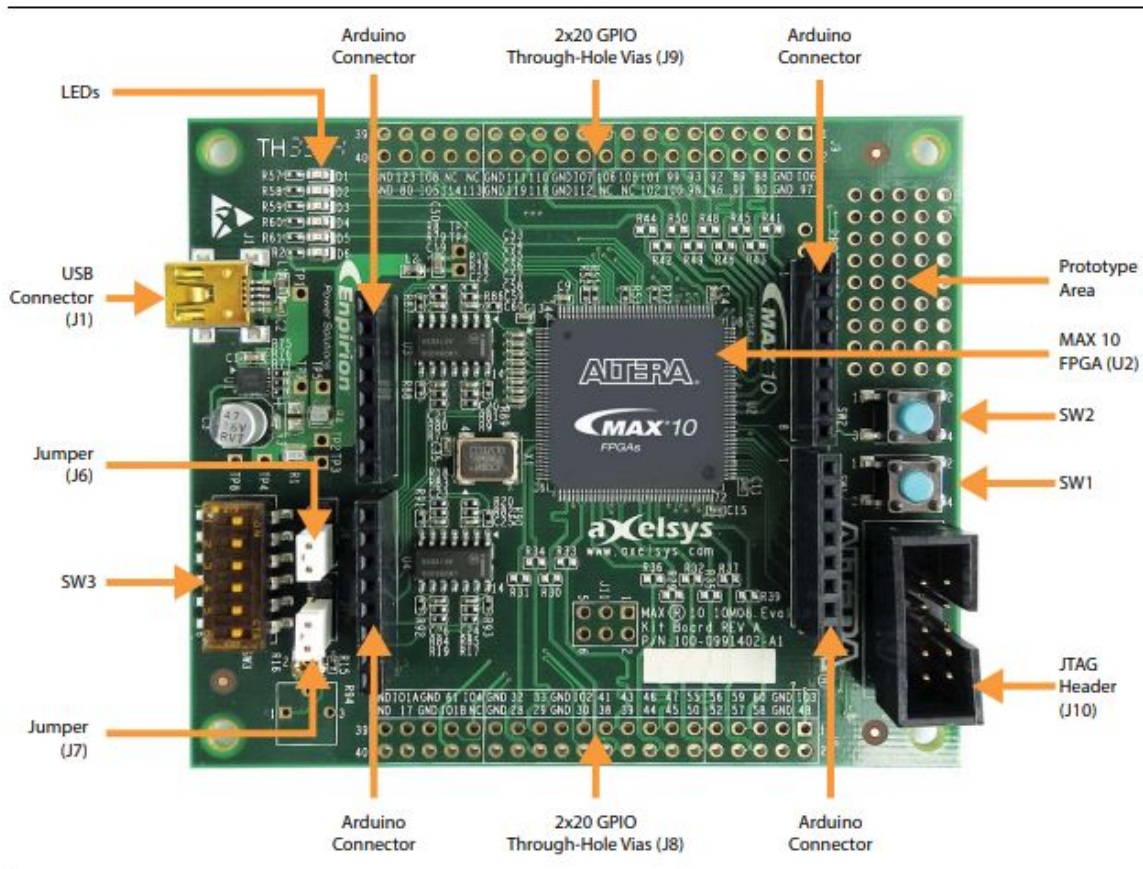
1. Identifying & evaluating the parts and the functions of the DB and comparing with MAX 10 Development Kit to understand the System-on-Chip.
2. Coming to conclusions about the problem areas with the DB

### 4. STUDY

#### i. ABOUT THE MAX10 DEVELOPMENT KIT

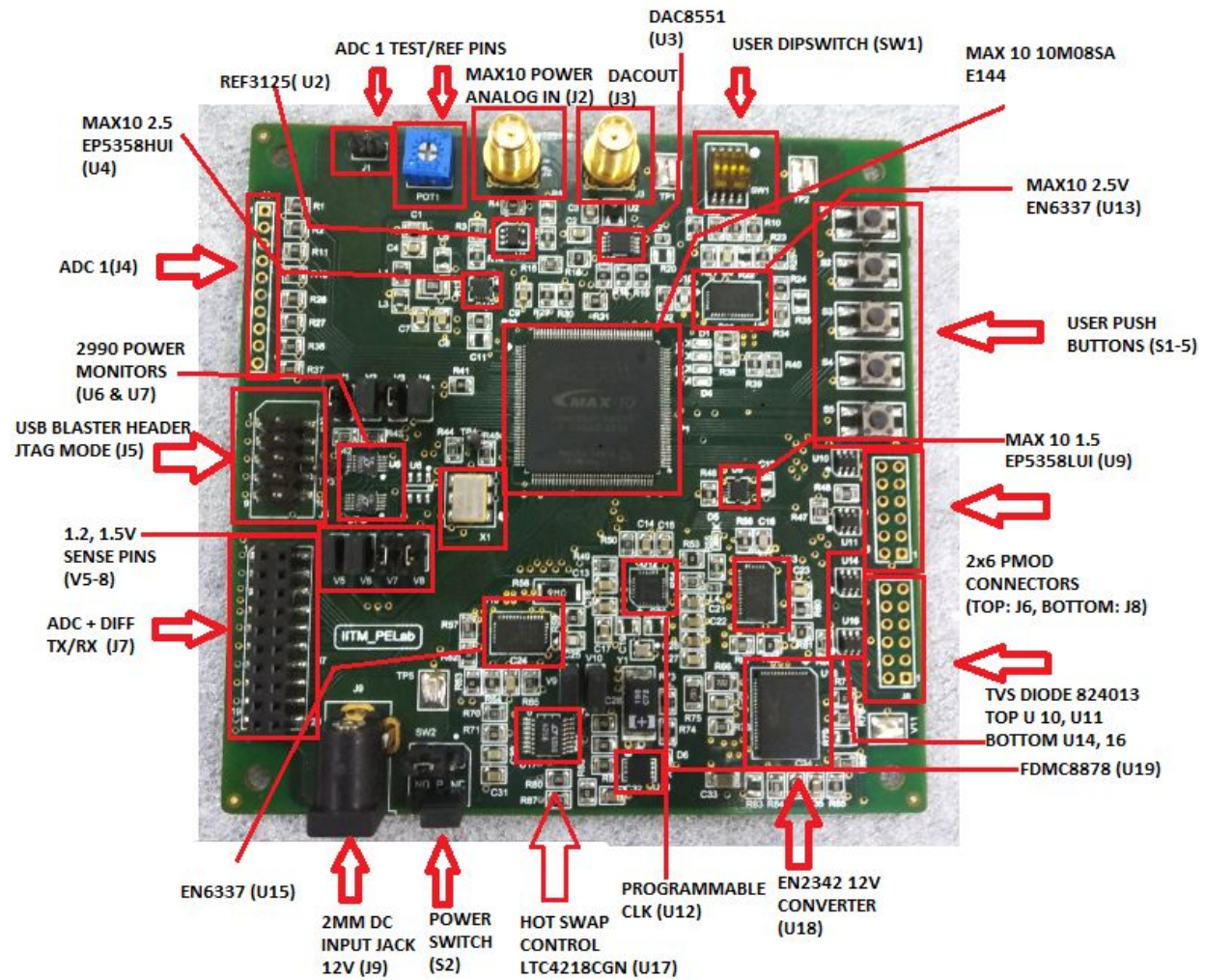
**Name of device: 10M08SA E144**

Features: Low power, directly compatible with Arduino boards, area to expand I/O and attach peripherals, and through hole vias for general purpose I/O jumpers.



## ii. IDENTIFYING PARTS OF THE DESIGNED BOARD

On going through the design schematic available of the designed board, I was able to understand and edit a photo of the DB with as much detail as was important. The schematic diagram available for the board is not user-friendly nor easily understandable to the user, so I have tried to make the graphic as simple and informative as possible. Without available thorough documentation, it fell to me to create a key and interpret datasheets and schematics to make sense of the FPGA design in a short time. Since the board had never been tested post fabrication, the entire scope of the problem was unknown as well.



Indexed Key for the schematic:  
(Schematic available [here](#))

I/O DESIGNATION	PART SPECIFICATION	USE
P1_2 J7	IO_1A, IO_1B	ADC + STD I/O
P1_4	IO_3	STD I/O PINS 38-60
P1_3	IO_2 (32, 33)	STD I/O FOR PLL_OUT
P1-9	IO_8	3 DAC pins, 3 SWITCH pins, 2 P/N TX RX

P1-5,6,7	IO_4,5,6	GENERAL HS TX/RX
P1-13	GND	12 PINS+ REF+ EP?
P1-11	IO_1B ,IO_8	2.5V JTAG EN
P1-10	IO_2, 6	CLK INPUTS
J5	USB BLASTER/ PROG. HEADER JTAG HEADER	CON 2X5
J4		CON 10 FOR 8 CH ADC W/ CAPACITORS PIN PROTECTION
J1		CON2
X1		2.5V 10MHZ DEFAULT CLK CONTROL
U12	Si5338 Programmable Oscillator	3 CLKS, A&B 1.5V POWER
U3	DAC8551	3.3V
U10, 11 & J6	824013 & 2X6 PMOD	3.3V
U16, 14 & J8	“	“
SW1	DIPSWITCH4	DEVICE JTAG CONFIG 3 DIPSWITCHES (1.5) + CONFIG SELECTION (3.3)
BUTTONS	USER PB- PUSH BUTTONS (4x)	1.5 V
	PULSE_NCONFIG	3.3 V
	CPU_RESET	3.3V
LED	USER LED- RED (5x)	TURNS ON AT 2.5V
	POWER LED- BLUE	12V
Hot Swap Controller Circuit	LTC4218CGN	12V
MAIN POWER 12V DCIN TO 5V	EN2342, POWER SOC 4A CONVERTER	5V/4A
3.3/2.5v 3A INPUTS	EN6337, POWER FROM 5V PREV	ENABLE SIGNALS DIFF FOR DIFF OUTPUTS + F/B SIGNALS

2.5V VCC(A, ADC)/ 600mA	EP5358HUI	POWER SoC
sns/rns		TEST PADS
MAX10 1.2V / 3A + 1.5V/ 600mA	EN6337	POWER SoC
U6, U7	POWER MONITOR LTC2900	BOTH 3.3V BUT 2.5V ADDITIONAL FOR U6 & I2C PINS TO U7
P1-12 AD8515	POWER TO ADC	ANALOG IN SIGNAL
P1-1	MAX10 POWER	4X 2.5 VCCIO 1X VCCA 1X 1.2 VCC 1X 3.3 1X 1.5VCCIO
DECOUPLING CAPS		0.1 UF, ARRANGED DIFFERENTLY FOR VARIOUS POWER PINS

### iii. COMPARATIVE ANALYSIS OF BOARD HARDWARE

The next step was understanding how the DB differs from the Evaluation Kit. Apart from being directly connected to a power source instead of the USB power from a computer, the DB is minimized in terms of peripheral handling capacity and has fewer I/Os and ADC channel. These reduce size, cost and power consumption of DB while restricting its potential applications and utility as well. While the following table is not all the parameters of difference, they form the salient differences.

CRITERION	MAX10 Eval Board	Designed Board
POWER	5V FROM PC	12V FROM SUPPLY
ADC (SAR)	8bit input *2	8 bit input *1
GPIO	80+ I/O POSSIBLE	60 STD I/O

PMOD (PERIPHERAL MODULE INTERAFCE)	4 regular 3.3V (2 2x6 connector )	1 3.3V 2x6 CONNECTOR
USB- JTAG specification	USB BLASTER PROGRAMMING HEADER	SAME JTAG CONNECTOR USED, one JTAG_SAFE testpoint present
Flash	QSPI FLASH	NOT PROVIDED

## 5. TESTING CRITERIA & RESULTS

On studying the circuits, a few questions came up.

### 1. What parameters needed to be to check the board?

On contacting the Intel Representative on the Intel Forum ([link](#)), the following interaction took place:

**Check the FPGA part number, Is it the same?**

>> yes, 10M08 Evaluation Kit and 10M08E144 has been used in the designed board as well.

**Are you able to “auto-detect” to detect the physical device in the JTAG chain?**

>> NO, check out the image attached at the end.

**Check the status of nCE,nCONFIG and nSTATUS pins.**

>> CAN WE GET A REFERENCE MANUAL FOR THE 10M08 EVAL KIT? WHAT IS THE EXPECTED OUTPUT FOR THESE PINS WHEN JTAG CHAIN IS PROPERLY DETECTED?

**Check pull-up/pull-down resistors and also check the soldering contact.**

>> FOR WHICH I/O PORT?

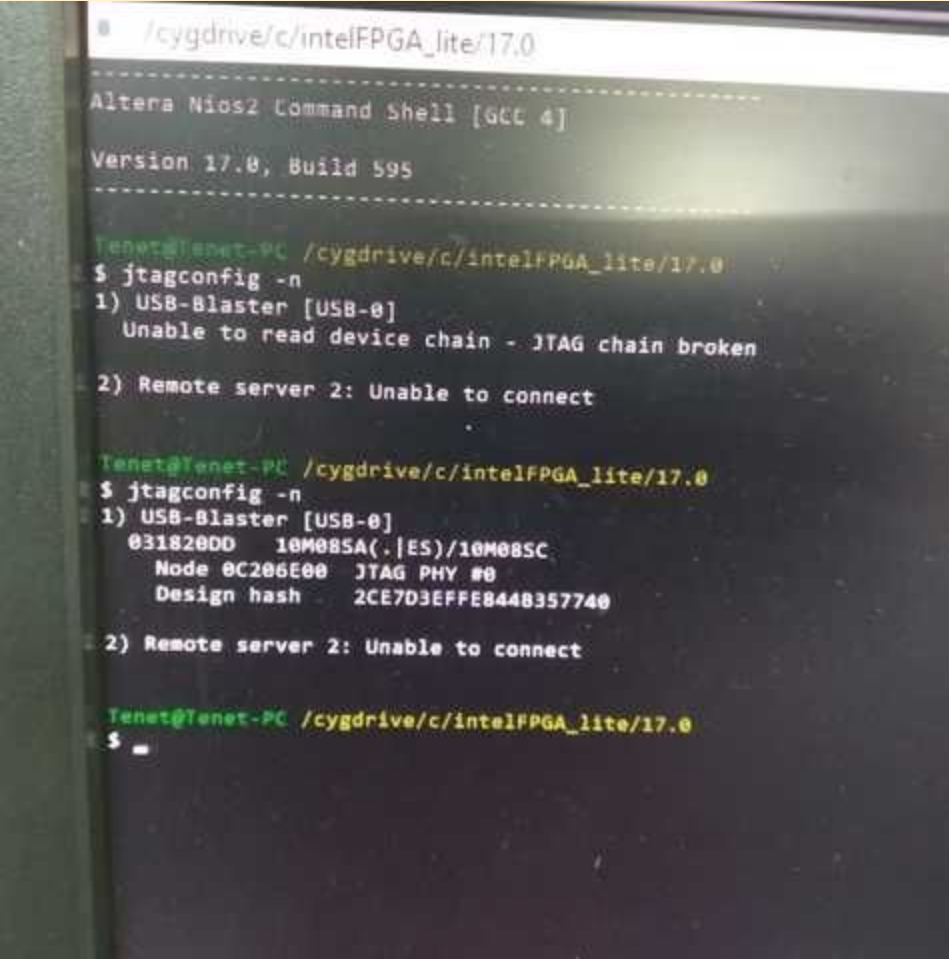
**Check the power supplies are ramped up to the appropriate voltage level according to the POR requirements. Refer below link for POR.**



[https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/mx-10/ug\\_m10\\_pwr.pdf](https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/mx-10/ug_m10_pwr.pdf)

>> Could you help by providing us with a schematic diagram? Where are the test pins/pads available in the Evaluation Board at least?

Attached some image on error message and pin status.



```
/cygdrive/c/intelFPGA_lite/17.0
-----
Altera Nios2 Command Shell [GCC 4]
Version 17.0, Build 595
-----

Tenet@Tenet-PC /cygdrive/c/intelFPGA_lite/17.0
$ jtagconfig -n
1) USB-Blaster [USB-0]
   Unable to read device chain - JTAG chain broken

2) Remote server 2: Unable to connect

Tenet@Tenet-PC /cygdrive/c/intelFPGA_lite/17.0
$ jtagconfig -n
1) USB-Blaster [USB-0]
   031820D0 10M085A(.|ES)/10M085C
   Node 0C206E00 JTAG PHY #0
   Design hash 2CE7D3EFFE844B357740

2) Remote server 2: Unable to connect

Tenet@Tenet-PC /cygdrive/c/intelFPGA_lite/17.0
$ _
```

The last picture was added by testing the JTAG chain on the NIOS II debugger that automatically downloads with the Quartus Prime software. The first output after `jtagconfig -n` command was when the Evaluation Kit was loaded & the second was when DB was connected.

The interaction gave us two leads:

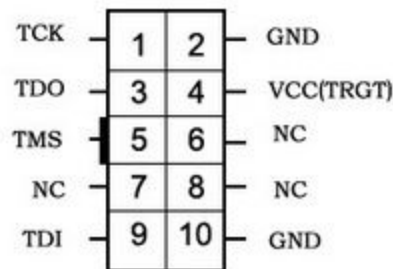
- a. The JTAG hardware is un-recognizeable.
- b. The critical pins to test any board with. However, the pins suggested by the representative were 1. Unavailable as testpoints on the either board and 2. Didn't

have standard values to compare with. So comparative testing became the only alternative.

2. *Where on the hardware could we test these FPGA pin statuses like VCC, VCCA, VCCIO also configuration pins like TCK, TDI, TMS nCE, nCONFIG and nSTATUS pins?*

Because the DB had no power USB connection to the PC, there was no way to check if it was correctly detected by the PC. The hardware was detectable on the Quartus programmer, but the loading of the programmer consistently failed.

We decided to measure the USB header values through the back of the board, at the solder points, which is where most of the critical pins mentioned are located.



*JTAG PINS ON USB BLASTER*

The DB only had 5 test points, and I was unsure of what values were expected at optimum working. The schematic diagram was incomplete in terms of testing conditions and the only option was to test comparatively between Evaluation Kit and DB.

SIGNAL	EVAL KIT TEST POINT	EXPECTED	MEASURED	DB/ TEST PT	EXPECTED VALUE	MEASURED
VCCa	TP_6	3.3	3.3	TP2	2.5	1. 2V across R13
VCC_CORE	TP_3/ TP_2	3.3	3.3	TP5	1.5	2.5 AT R13

						0V AT TP_5
VCC_IO	TP_4/ TP_5	3.3	3.3	13	2.5	V5- 1.5 V6- 1.5 V7- 1.19 V8- 1.19 V1- 2.5 V2- 2.5 V3- 2.5 V4- 2.5
nCONFI G	U2-9, 129	VCC_IO8	3.3	23	INPUT ONLY	-
nSTAT US	136	“	3.3	21	3.3	-
REF_G ND	TP_8, TP_9	0		TP1	0	-
JTAG_L OCK				TP3	3.3 (HIGH)	-

**NOTE: The - values on the last column are because we stopped testing the circuit. The power LED on the board started to flicker rapidly and the testpoint showed no power flowing in the primary power SoC circuit so we had to stop.**

### *3. Even after testing failure, what can we conclude?*

The DB is also the single board manufactured of its kind, so any results obtained are not reproducible and nothing specific concluded with certainty. This also makes the likelihood of a manufacturing fault exceedingly relevant in this scenario.

We purchased another power converter to power the board from the mains, and the board still didn't consume power and the power LED continued to flicker.

Concluding that the fault is internal to the circuit, and overlooking the possibility of fabrication error, there are certain reasonable conclusions to come to:

1. The board already had some power issues, as observed on the TP\_5 value on DB during testing.
2. The power SOC's were connected/ placed on PCB poorly and the power fluctuations after one round of careful testing support this theory.

3. The design needs to be verified and tested on an emulator first. Without that data, and being able to observe more quantities, the testing is largely guesswork.

## **6. PROPOSED TESTING & CONCLUSION**

1. On refabricating the board, the expected voltage vs output mismatch would have to be established conclusively
2. The JTAG header itself needs to be tested
3. Checking the PCB layout for any obvious faults will have to be done before refabrication of the board
4. More care needs to be taken to include more test-points on the board.
5. Adding a USB interface helps understand if the PC recognizes the hardware and data collection in the future

We can so far conclude that the design has issues and that the logical and design documentation needs to be improved and looked at once before the same board can be refabricated.

# Part B

## VERILOG CODES FOR DIGITAL CONTROL OF POWER CIRCUITS

### 1. Problem Statement & Objectives

Control circuits for power electronics applications are seeing a trend toward using reprogrammable logic circuits and quick processing using FPGAs and CPLD devices. Many programs from clock divider to ring counter to PWM generation have been executed on Quartus Prime with Verilog code (instead of using block diagrams and structural modelling).

#### **Need to control power circuits using logic circuits instead of DSPs:**

The tendency to use concurrent hardware for the control purpose, results in a custom hardware solution of implementing the digital control scheme in an FPGA instead of DSP. The continuous and simultaneous execution of all the internal logic elements of FPGA and also all the control procedures allows the usage of high-speed demanding algorithm for power electronics system control. These devices are also less prone to environmental degradation over long term use, and are dependable.

#### **Objectives:**

1. To learn & implement circuits with a software programming perspective, without designing the specific hardware for it.
2. To create simple and optimized programs for familiarity and testing of circuits
3. Using Altera MAX V and MAX 10 boards for power control techniques

## 2. TOOLS AND HARDWARE

### i. Using a HDL

HDL contains some high-level programming constructs along with constructs to describe the connectivity of hardware design, It allows the designer to describe various levels of abstractions without choosing a specific fabrication technology. It can describe functionality as well as timing and concurrency is possible. Easy to learn and use.

## **Language Used: Verilog**

The designers of Verilog wanted a language with a syntax similar to the C programming language, which was already widely used in engineering software development. Like C, Verilog is case-sensitive and has a basic preprocessor (though less sophisticated than that of ANSI C/C++). Its control flow keywords (if/else, for, while, case, etc.) are equivalent, and its operator precedence is compatible with C. Syntactic differences include: required bit-widths for variable declarations, demarcation of procedural blocks (Verilog uses begin/end instead of curly braces {}), and many other minor differences. Verilog requires that variables be given a definite size.

## **Features**

Hardware description languages such as Verilog are similar to software programming languages because they include ways of describing the propagation time and signal strengths (sensitivity). There are two types of assignment operators; a blocking assignment (=), and a non-blocking (<=) assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables. Since these concepts are part of Verilog's language semantics, designers could quickly write descriptions of large circuits in a relatively compact and concise form.

A Verilog design consists of a hierarchy of modules. Modules encapsulate design hierarchy, and communicate with other modules through a set of declared input, output, and bidirectional ports. Internally, a module can contain any combination of the following: net/variable declarations (wire, reg, integer, etc.), concurrent and sequential statement blocks, and instances of other modules (sub-hierarchies). Sequential statements are placed inside a begin/end block and executed in sequential order within the block. However, the blocks themselves are executed concurrently, making Verilog a dataflow language.

There are several statements in Verilog that have no analog in real hardware, e.g. \$display. Consequently, much of the language can not be used to describe hardware. The examples presented here are the classic subset of the language that has a direct mapping to real gates.

## **Using the Register Transfer Logic (RTL) level**

It uses a combination of behavioral and dataflow constructs. This is the most popular level because logic synthesis tools can create gate-level netlist from the RTL level design.

## ii. VERILOG SIMULATOR USED: QUARTUS PRIME 18.1

Intel Quartus Prime is programmable logic device design software produced by Intel, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Quartus Prime includes an implementation of VHDL and Verilog for hardware description, visual editing of logic circuits, and vector waveform simulation.

## iii. EXAMPLE: ATM

```
module ATM(
    input clk,
    input rst,
    input card_det,
        input card_valid,
    input [3:0] pin, // password ip
        input [3:0] cash_in,          //cash
    input in BINARY IN HUNDREDS
        output reg data_valid,        // confirmation
    message after entering valid pw
        input cash_rec,              //to reinstate s0

    output reg enter_pass,
    output reg enter_cash,
    output reg[3:0] notes100,
    output reg[1:0] notes500

);

parameter s0 = 2'b00;          // default state,
waits for card input
parameter s1 = 2'b01;          // checks pin
parameter s2 = 2'b10;
parameter s3 = 2'b11;
parameter password=4'b1000; //because we
don't make databases here

        reg [1:0] state = s0;

    initial begin
        enter_pass<=1'b0;
        enter_cash<=1'b0;

    end

    always@(posedge clk) begin
        if (rst) begin
            state <= s0;
        end

        else
            case (state)
                s0 : begin
                    if (card_det==1'b1) begin

                        notes100<=4'b0000;

                        notes500<=2'b00;

                        enter_pass<=1'b0;
```

```

state <= s1;

end
    else
        state <= s0;

$display("Enter card!");
end

s1 : begin
    if
((card_valid==1'b0)|| (pin[3:0]!=password))begin
        state <= s1;

enter_pass<=1'b1;

end
        else if
((card_valid==1'b1)&&(pin[3:0]==password))begin
n

enter_pass<=1'b0;

enter_cash<=1'b1;

state <= s2;

end
        else
            state <=s1;

end
s2 : begin
    if (pin!=password)

begin

data_valid=1'b0;

state

<= s2;

enter_cash<=1'b1;

```

```

end
        else if ( (pin==password))
begin

data_valid=1'b1;

state

<= s3;

enter_cash<=1'b0;

end
        else
            state <= s2;
end
s3 : begin
    if (cash_in[3:0] < 4'b0101)
begin

notes100<=cash_in;

notes500<=2'b00;

end
        else if ((cash_in [3:0]
>=4'b0101)&&(cash_in!=4'b1010))
begin
            notes100<=cash_in-(4'b0101);
            notes500<=2'b01;

end

if(cash_rec==1'b0) begin
    state <= s3;

end
else

if(cash_rec==1'b1)begin

state<=s0;

notes100<= 4'b0000;

notes500<=2'b00;

end
end
endcase
end

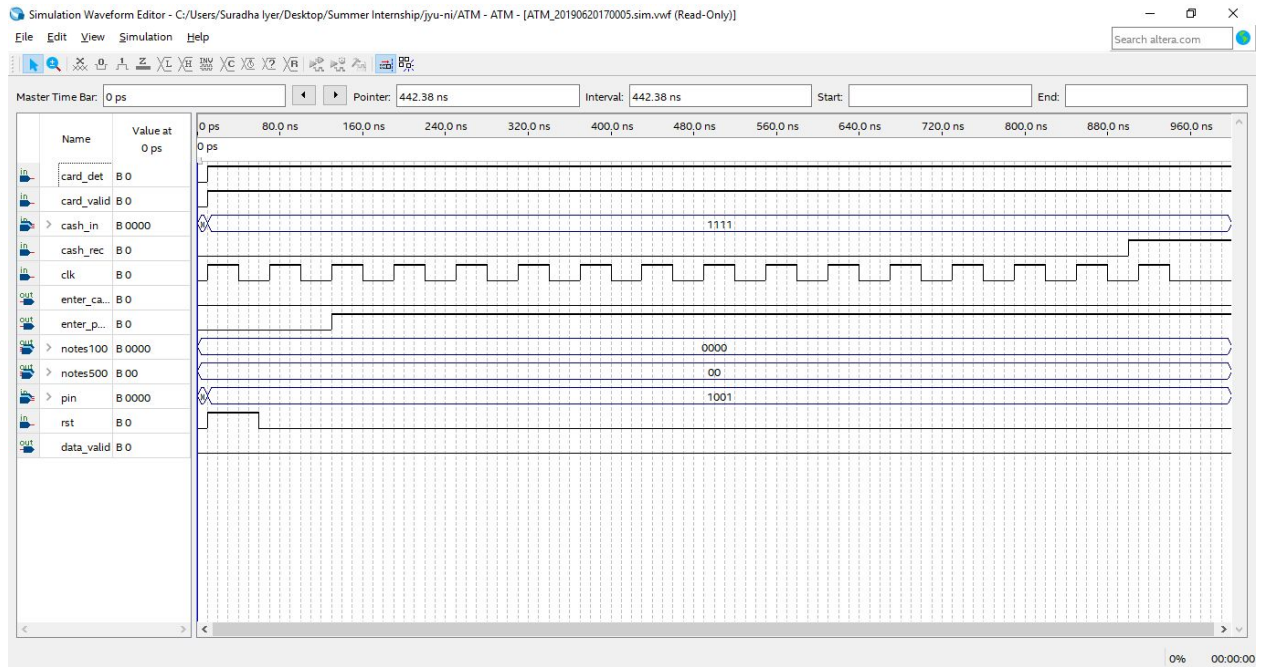
```



endmodule

## Output of Waveform Generator:

### a. WITH INVALID PIN (data\_valid line doesn't get activated)



### b. WITH VALID PIN (pin is accepted, verified and cash is dispensed)



**iv. HARDWARE USED: a. Max V Development Board**



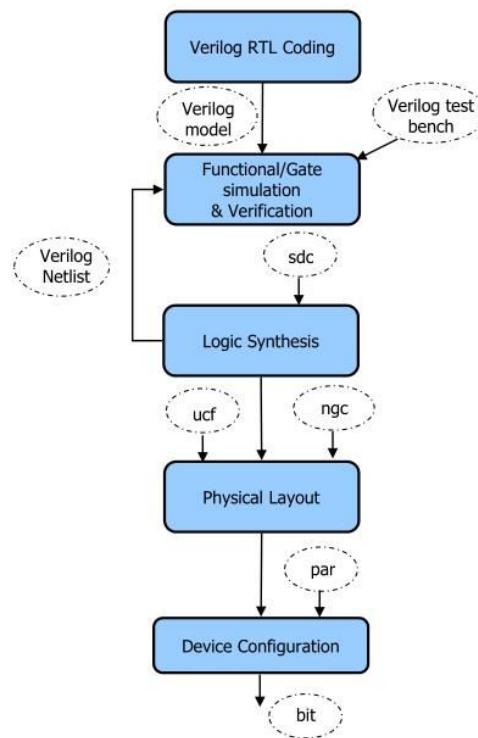
**Board Layout of MAX V**

**Features**

Low-cost hardware platform to quickly begin developing low-cost, low-power CPLD designs. The development board is designed to prototype the most common CPLD applications, including I/O expansion, interface bridging, power management control, configuration and initialization control, and analog interface control. Here we have designed systems and codes to understand clock division and the PWM control technique for DC power converters.

**b. MAX 10 Development Kit (see Part I of report for hardware description and features)**

## Design Process



*Image credit: CSRC*

1. Verilog Design (used because of ease in abstraction and C language type behavioural design)
2. Verification with Waveform (in .vwf format)
3. Pin assignment (10MHz Clock for MAX V appears on Pin\_H5)
4. Recompile
5. Uploading through USB Blaster- JTAG interface (Quartus-> Programmer-> Hardware Setup ->Choose target deviec->Add File-> choose output file -> check *Program Configure and Verify* checkboxes->Start )

## Conclusion & Precautions

Designs were tested on Intel MAX V Development Kit and MAX 10 Development Kit (based on availability of higher number of onboard LEDs).

1. Many real time waveforms are not available because of the limitation of Quartus software on longer time scale. The Part 0 set of codes in the Appendix are observable because they run at high frequencies.
2. Clock division programs (lower frequency of a few KHz) thus had to be tested on the boards only, and gave the approximate frequencies specified for PWM and

counter applications. Manual measurements are inaccurate, but the logic was verified.

3. The boards are easy to use and debug but output voltage levels are unlikely to drive real power circuits.

## Appendix: Codes Part 0, I & II

### Part 0: Basic Verilog Codes

#### a. 3:1 MUX

```
module mux3down1(
    input [7:0] X,
    input [2:0] A,
    output reg Y
);
    always@(A,X)
    begin
        case(A)
            3'b000: Y = X[0];
            3'b001: Y = X[1];
            3'b010: Y = X[2];
            3'b011: Y = X[3];
            3'b100: Y = X[4];
            3'b101: Y = X[5];
            3'b110: Y = X[6];
            3'b111: Y = X[7];
            default: Y= 1'b0;
        endcase
    end
endmodule
```

#### b. D- Flip Flop

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09:47:47
// 01/28/2019
// Design Name:
// Module Name: Dff
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:

/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
module Dff(
    input d,clock,reset,
    output reg q
);
    always@(posedge
clock)
    begin
        if(!reset)
            q=1'b0;
```

```

else
q=d;
end

```

```

endmodule

```

## c. FSMs

### 1. Sequence Detector- Mealy 111

```

module mealy1(
    input clk,
    input reset,
    input x,
    output reg z
);
    reg [0:1] y;
    parameter A=2'b00,
    B=2'b01, C=2'b10, D=2'b11;

    always@(posedge clk)
    begin
        if(reset==0)
            y<=A;
        else
            begin
                case(y)
                A: if(x) begin
                    y<=A;
                    z<=0;
                end
                else begin
                    y<=B;
                    z<=0;
                end

                B: if(x) begin
                    y<=C;
                    z<=0;
                end
            end
        end
    end

```

```

        else begin
            y<=B;
            z<=0;
        end
    end

```

```

        C: if(x) begin
            y<=D;
            z<=0;
        end
        else begin
            y<=B;
            z<=0;
        end
    end

```

```

        D: if(x) begin
            y<=A;
            z<=0;
        end
        else begin
            y<=B;
            z<=1;
        end
    end

```

```

        default: y<=A;
    endcase
end
end

```

```

endmodule

```

### 2. Digital Clock

```

module clock(
    input clk,
    input display, //0 =
    24hrs, 1 = 12hrs

```

```

    input display_sec,
    //0 = show, 1 = hide
    input reset,
    output reg [5:0] seconds,
    output reg [5:0] minutes,

```

```

        output reg [4:0] hours
    );

    always @(posedge(clk) or
    posedge(reset)) begin
        if(reset == 1'b1) begin
            //check for active high reset
            //reset the time
            seconds <= 0;
            minutes <= 0;
            hours <= 0;
        end
        else if(clk == 1'b1)
            begin //at the
                //beginning of each second

                seconds <=
                seconds + 1;
                //increment sec
                if(seconds ==
                59) begin //check
                    //for max value of sec

                    seconds <= 0;
                    //reset seconds

                    minutes
                    <= minutes + 1; //increment
                    minutes

                    if(minutes == 59) begin //check
                        //for max value of min

                        minutes <= 0;
                        //reset minutes

                        hours <= hours + 1;
                        //increment hours

                        if(hours == 23) begin //check
                            //for max value of hours

                            hours <= 0;
                            //reset hours

                        end

                    end

                end
            end
        end
    end
end
end

```

```

        end //end
        updating sec/min/hr

        if (display ==
        1'b0) begin

            $write("Time:
            %d:%d",hours,minutes);
            if
            (display_sec == 0)

            $display(":%d",seconds);
            else

            $display(.,.);
            end
            else if((display
            == 1'b1) & (hours>12)) begin

            $write("Time:
            %d:%d",(hours-12),minutes);
            if
            (display_sec == 0)

            $write(":%d",seconds);

            $display(" PM");
            end
            else begin
                //(((display == 1'b1) and
                (hours<=12))

                $write("Time:
                %d:%d",hours,minutes);
                if
                (display_sec == 0)

                $write(":%d",seconds);

                $display(" AM");
                end
            end
        end
    end

endmodule

```

### 3. Automatic Billing Machine

A machine which can vend four products that is coffee, cold drink, candies and snacks. Four select (select1, select2, select3, select4) inputs are taken for selection of products. Select1 is used for the selection of snacks. Similarly select2, select3, select4 are used for coffee, cold drink and candies respectively. Rs\_10 and Rs\_20 inputs represents rupees 10/- and 20/-notes respectively. A cancel input is also used when the user wants to withdraw his request and also the money will be returned through the return output. Return, product and change are the outputs. Return and change vectors are seven bits wide. Money is an in/out signal which can be updated with the total money of all products delivered at a time. Money signal is seven bits wide. Money\_count is an internal signal which can be updated at every transition. This signal is also seven bits wide. If the inserted money is more than the total money of products then the change will be returned through the change output signal. The products with their prices are shown by table 1. There are also two input signal clk and reset. The machine will work on the positive edge of clock and will return to its initial state when the reset button is pressed.

```
module billing(  
    input clk,  
    input reset, //  
    input sel1, //  
    input sel2, //  
    input sel3, //  
    input sel4, //  
    input cancel, //  
    input rs_10, //  
    input rs_20, //  
    output reg product,  
    output reg [6:0] change,  
    output reg [6:0] count,  
    output reg [6:0] z,  
    output reg [6:0] Return  
);  
parameter s0 = 4'b 0000,  
snacks = 4'b 0001, coffee = 4'b  
0010, cold    = 4'b 0011,  
    candies = 4'b 0100,  
rs10  = 4'b 0101, rs20  = 4'b  
0110, ret_money = 4'b 0111,  
    product_state = 4'b  
1000;  
// always block for select lines  
(selecting item)  
reg [3:0] Y, y;  
//output reg [6:0] z;  
//assign count = 7'b 0000000;  
  
always @ (posedge clk)  
begin  
    if (reset)  
    begin  
        y <= s0;  
        count <= 7'b  
0000000;  
        change <= 7'b  
0000000;  
        Return <= 7'b  
0000000;  
    end  
    else  
        y <= Y;  
    end  
  
always @ (posedge clk)  
begin  
    product <= 1'b 0;  
    case (y)  
        s0:    if (sel1)  
  
    begin  
        Y <= snacks;  

```

```

        z <= 7'b
0011110;

        end
        else if
(sel2)
        begin
            Y <= coffee;

```

```

        z <= 7'b
0101000;

        end
        else if
(sel3)
        begin
            Y <= cold;

```

```

        z <= 7'b
0101000;

        end
        else if
(sel4)
        begin
            Y <= candies;

```

```

        z <= 7'b
0011110;

        end
        else
        begin
            Y <= s0;

count <= 7'b 0000000;

```

```

change <= 7'b 0000000;

Return <= 7'b 0000000;
        end
        snacks: if
(cancel) Y <= s0;

        else if (rs_10)
        begin

```

```

Y <= rs10;

count <= count + 10; //7'b
0001010;

        end

```

```

        else if (rs_20)
        begin
            Y <= rs20;

```

```

count <= count + 20; //7'b
0010100;

        end

```

```

        else
            Y <=
snacks;

```

```

        coffee: if
(cancel) Y <= s0;

        else if (rs_10)

```

```

        begin

```



Y <= rs10;

count <= count + 10;//7'b  
0001010;

end

else if (rs\_20)

begin

Y <= rs20;

count <= count + 20;//7'b  
0010100;

end

else

Y <=

coffee;

cold:

if (cancel) Y <= s0;

else if (rs\_10)

begin

Y <= rs10;

count <= count + 10;//7'b  
0001010;

end

else if (rs\_20)

begin

Y <= rs20;

count <= count + 20;//7'b  
0010100;

end

else

Y <=

cold;

candies:

if (cancel) Y <= s0;

else if (rs\_10)

begin

Y <= rs10;

count <= count + 10;//7'b  
0001010;

end

else if (rs\_20)

begin

Y <= rs20;

count <= count + 20;//7'b 111  
0110;

end

else

```

candies;
Y <=
rs10:
if (rs_10)
begin
Y <= rs10;

count <= count + 10;//7'b
0001010;

end

else if (rs_20)
begin

Y <= rs20;

count <= count + 20;//7'b
0010100;

end

else if (cancel)
begin

Y <= ret_money;

Return <= count;

end

else
begin

```

```

Y <= product_state;

change <= count - z;

end

rs20:
if (rs_10)
begin

Y <= rs10;

count <= count + 10;//7'b
0001010;

end

else if (rs_20)
begin

Y <= rs20;

count <= count + 20;//7'b
0010100;

end

else if (cancel)
begin

Y <= ret_money;

Return <= count;

end

else

```

```

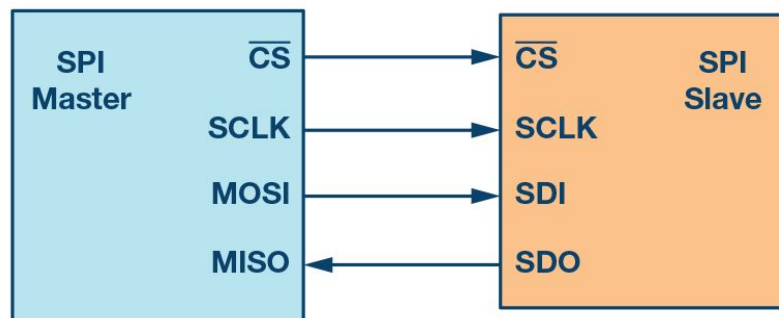
begin
    ret_money:
        Y <= product_state;
        change <= count - z;
    end
    product_state:
        begin
            product
        end
    end
    Y <=
    s0;
end
ret_money:
begin
    Y <=
    s0;
    count
    <= 7'b 0000000;
end
default: Y <=
    s0;
endcase
end
endmodule

```

## d. Communication Protocols

### 1. SPI

Devices communicating via SPI are in a master-slave relationship. The master is the controlling device (usually a microcontroller), while the slave (usually a sensor, display, or memory chip) takes instruction from the master. The simplest configuration of SPI is a single master, single slave system, but one master can control more than one slave



```

module spi(
    input [7:0] data,
    output reg
    [7:0]received_data,
    input sclk,

```

```

    output reg MOSI,
    input MISO,
    output reg slave_select1,
    output reg
    slave_select2,

```

```

input reset,
input
transmission_enable,
input reception_enable
);
parameter halt_transmission =
2'b00, transmission = 2'b01,
reception = 2'b10,
halt_reception = 2'b11;
reg [1:0] state1 =
halt_transmission;
reg [1:0] state2 = halt_reception;
integer i = 1'b0, index = 1'b0;

```

```

always @ (posedge sclk) //
always block for transmission
begin

```

```

        if (!reset) begin
            slave_select1 =
1'b1; // slave 0 for
transmission

```

```

        end

```

```

    else begin

```

```

        case (state1)

```

```

            halt_transmission :

```

```

        begin

```

```

            slave_select1 = 1'b1;

```

```

            if
(transmission_enable == 1)
begin

```

```

$display("Device ready for
transmission.\n");

```

```

            slave_select1 =
1'b0; // slave 1 selected

```

```

            state1 =
transmission;

```

```

        end

```

```

    else begin

```

```

            state1 =
halt_transmission;

```

```

        end

```

```

    end

```

```

transmission :

```

```

begin

```

```

    if (i < 8) begin

```

```

        MOSI = data[i];

```

```

        i = i + 1;

```

```

        state1 = transmission;

```

```

    end

```

```
else begin
```

```
    i = 0;
```

```
    $display("Transmission  
Completed.\n");
```

```
    state1 =  
    halt_transmission;
```

```
end
```

```
end
```

```
endcase;
```

```
    end
```

```
end
```

```
always @ (posedge sclk)  
// always block for reception  
begin
```

```
    if (!reset) begin  
        slave_select2 =  
1'b1;        // slave 2 for  
reception
```

```
    end
```

```
    else begin
```

```
        case (state2)
```

```
            halt_reception :
```

```
begin
```

```
    slave_select2 = 1'b1;
```

```
    if (reception_enable  
== 1) begin
```

```
        $display("Device ready for  
reception.\n");
```

```
        slave_select2 = 1'b0;  
// slave 1 selected
```

```
        state2 =  
reception;
```

```
    end
```

```
    else begin
```

```
        state2 =  
halt_reception;
```

```
    end
```

```
end
```

```
reception :
```

```
begin
```

```
    if (index<8) begin
```

```

received_data[index] = MISO;

state2 = halt_reception;

index = index +1;

state2 = reception ;

end

end

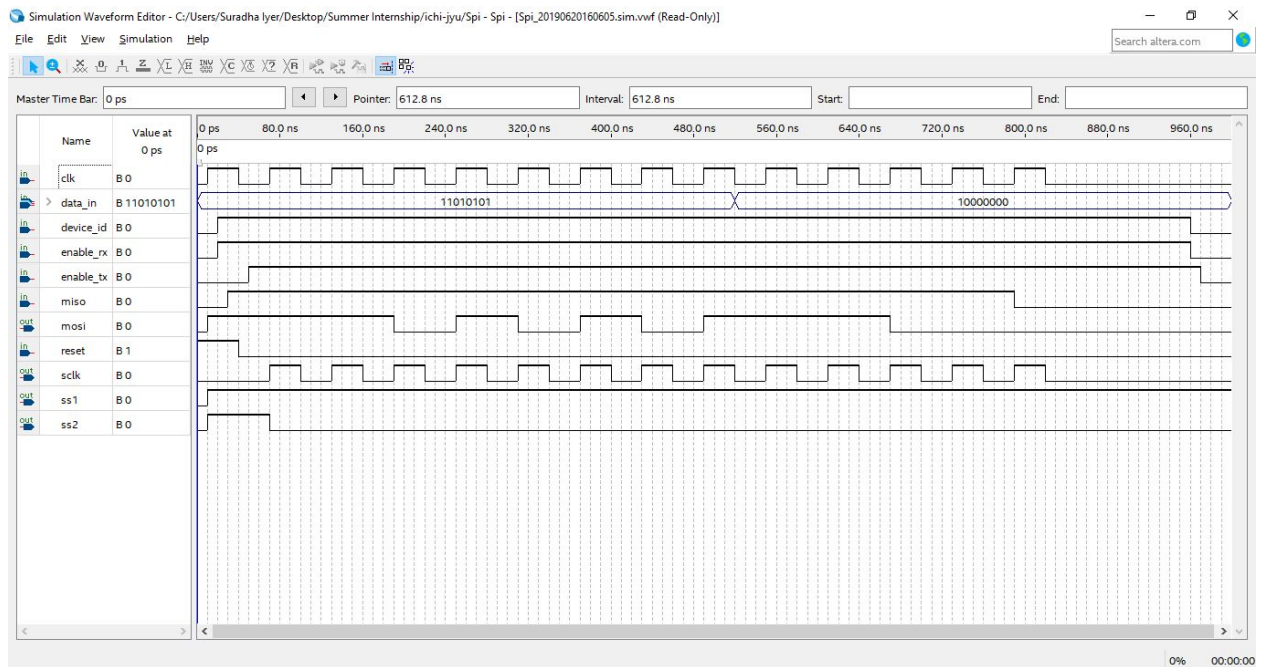
endcase;
end

end

endmodule

index = 0;

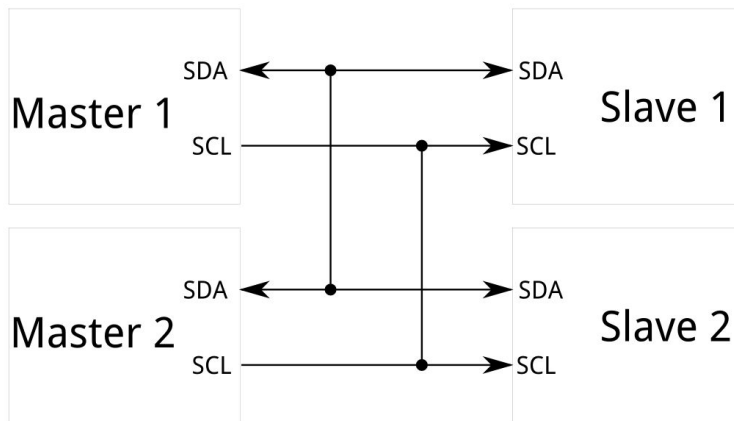
```



**SPI OUTPUT WAVEFORM**

## 2. I2C

I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.



```

module I2C
(
    input temp,
    input scl,
    output reg sda,
    input [6:0] device_address,
    input [7:0] memory_address,
    input [7:0] data,
    input rw
);
parameter idle = 3'b000, start = 3'b001,
device_address_transmission = 3'b010;
parameter read_write = 3'b011,
memory_address_transmission = 3'b100;
parameter data_transmission = 3'b101, stop = 3'b110;
integer i=0, n=0;
reg [2:0]state = idle;
always @ (posedge scl)
begin
    case (state)
        idle :
            begin
                if (temp == 0) begin
                    sda = 1;

                    $display("Start signal
received, moving to next
state.");

                    state = start;
                end
            end
    end
end

```

```

address transmission
completed.");

else begin

    state = read_write;

    sda = 1;

end

$display("communication not yet
started.");

    state = idle;
end
    end

start :

    begin

        sda = 0;

$display("Device ready for
Device address transmission.");

        state =
device_address_transmission;
    end

device_address_transmission :

    begin

if (n == 6) begin

    n=0;

    $display("Device
address transmission
completed.");

    state = read_write;

end

else begin

    sda =
device_address[6-n];

    n = n+1;

    state =
device_address_transmission;

end

    end

    read_write :

        begin

            if (rw == 0)

begin

                sda = 0;

                $display("write mode
enabled.");

                state =
memory_address_transmission;

```



```

        end

    else begin

        state = idle;

    end
end

memory_address_transmission
:

    begin
        if (n == 7) begin

            n=0;

            $display("Memory
address transmission
completed.");

            state =
data_transmission;

        end

    else begin

        sda =
memory_address[7-n];

        n = n+1;

```

```

        state =
memory_address_transmission;

    end

    end

    data_transmission :

        begin
            if (n == 7) begin

                n=0;

                $display("data
transmission completed.");

                state = stop;

            end

        else begin

            sda = data[7-n];

            n = n+1;

            state =
data_transmission;

        end

    end

    stop :

```

```

begin
state = idle;

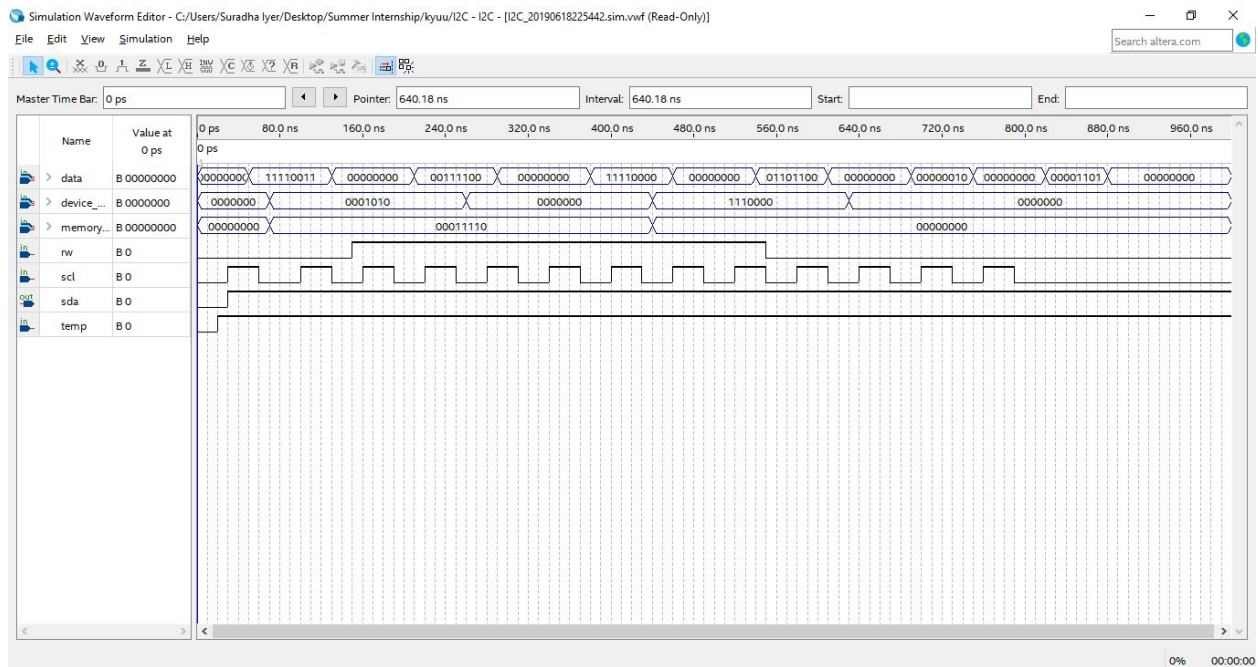
sda = 1;

end

endcase
end
endmodule

$display("stop.");

```



I2C OUTPUT WAVEFORM

## Part 1: Codes for Clock Division

### 1.1

Write a Verilog code to glow a LED for 100 ms and should be off for the next 100 ms.

```

// create module
module blinkswitch(
    input wire clk, // 10MHz input clock
    output reg [1:0] LED, // LED ouput
    output reg debug,
    output reg [7:0] lol
);

// create a binary counter
reg [19:0] count; // 20-bit
reg [19:0] flag;
reg dir;

initial begin

```

```

count = 10; // start at zero
//flag<= 1'b1;
dir<=0;
end

always @(posedge clk) begin
debug <=dir;
if (dir== 0)

begin
if (count < 15)
begin

count= count+1;
//flag<= count;
LED<= 2'b11;
end

if (count == 15)
begin
dir<=1;
debug<= dir;
LED<=2'b10;
end
end

//flag=20'b0;
count <= 20'b100; // start at
20'b11110100001001000000
dir<=0;
end
endmodule

```

## 1.2

**Write a Verilog code to blink three on board LEDs simultaneously ( LEDs should glow for 3s and be off for next 3s). (MAX 10)**

```

// create module
module blinky(
input wire clk, // 10MHz input clock
output wire LED0,
output wire LED2,
output wire LED1 // 3 LED ouput
);

// create a binary counter
reg [23:0] count; // 24-bit counter

initial begin

count <= 24'b00000000; // start at zero

always @(posedge clk) begin

count <= count + 1; // count up

end

//assign LED to 23rd bit of the counter to blink
the LED at a few Hz
assign LED0 = count[22];
assign LED2 = count[22];
assign LED1 = count[22];

endmodule
end

```

## 1.3

**Write a program to blink two LEDs simultaneously for 2 s and the third LED for 4 s. (Two LEDs should glow for 2s and be off for next 2s. Third LED should glow for 4s and be off for next 4s).**

```

module manyleds(
    input clk,
        //input reset,
    output reg [25:0] q,
        //output reg [25:0] q1,
    output reg [1:0] led0,
    output reg led1
);
reg [24:0] temp1 =
25'b000000000000000000000000;
reg [24:0] temp2 =
25'b100110001001011010000000; // counts
down for 3s
reg [25:0] temp3 =
26'b11001100010010110100000000;
reg m;
//reg n;
//reg state;
//reg [9:0] i;
//reg [16:0]j;

initial begin
m<=1;
//n<=1;
//state<= 1;
end

always@(posedge clk)
begin

    led1<= 0;

    if (m) begin // up counter

        //if (n) begin
        //led1= 1'b0; end
        //else begin
        //led1= 1'b1; end

        q = temp1;
        q = q+1;
        temp1 = q;
        led1<= 0;
        if (q> temp2)
        begin m<=0; temp1
=25'b000000000000000000000000;
        end
        end

    else
        begin

        q = temp1;
        q = q+1;
        temp1 = q;
        led1<=1;
        if (q> temp3)
        begin m<=1; temp1
=25'b000000000000000000000000;
        end
        end

        // n<= ~n;
    end
end
endmodule

if (m) begin // up counter

```

## 1.4

**Write a program to increment a 3 bit ring counter every 1s. Display the result of the counter using the on-board LEDs.**

```

module ring_timer(
    input clk,
        //input reset,
    output reg [24:0] q,
    output reg [2:0] ring
        //output led1;
);
reg [24:0] temp1 =
25'b000000000000000000000000;

```

```

reg [24:0] temp2 =
25'b0100110001001011010000000; // counts
down for 2s
//reg [24:0] temp3 =
20'b0111001110010101110110000; //counts up
to 6s

reg m;

initial begin
m<=1;
ring = 3'b000;
//state<= 1;
end

always@(posedge clk)
begin
if (m) begin // up counter
q = temp1;
q = q+1;
temp1 = q;
//led0= 2'b11;
if (q== temp2)
begin m<=0; temp1
=25'b00000000000000000000000000000000;
end
end

else
begin
q = temp2;
q = q-1;
temp2 = q;
// led0 = 2'b00;

ring = ring + 1'b1;
if (ring== 3'b111) begin
ring= 3'b000;
end

if (q==
25'b00000000000000000000000000000000)
begin m<= 1; temp2 =
25'b1110011100101011101100000; end
end

end
endmodule

```

## 1.5

**Write a program interrupt to increment a 3 bit ring counter every 1s. Display the result of the counter using the on-board LEDs. Now, using one of the DIP switches, counter should run as a 3 bit binary counter when the switch is open, it should run as 3 bit ring counter when closed.**

```

module bin_ring(
input clk,
output reg [24:0] q,
input dip, //pin120 or SW1
output reg [2:0] ring // led 123- 131,
134, 135
//output led1;
);
reg [24:0] temp1 =
25'b00000000000000000000000000000000;

reg [24:0] temp2 =
25'b0100110001001011010000000; // counts
down for 2s
//reg [24:0] temp3 =
20'b0111001110010101110110000; //counts up
to 6s

reg m;
reg flag;
reg i;

initial begin

```

```

m<=1;
ring = 3'b000;
flag=0;
//state<= 1;
end

always@(posedge clk)
begin

    if (dip==1) begin
        flag= ~flag;

    end

    if (flag) begin
        if (m) begin // up counter
            q = temp1;
            q = q+1;
            temp1 = q;
            if (q== temp2)
                begin m<=0; temp1
=25'b000000000000000000000000;
                end
        end

    else
        begin
            q = temp2;
            q = q-1;
            temp2 = q;
            if (q==
25'b000000000000000000000000)
                begin m<= 1; temp2 =
25'b1110011100101011101100000; end
        end

        ring= ring+1;
        if (ring== 3'b111)begin
            ring= 3'b000;end
        end
    end
    i= ring[2];
end

```

```

ring[2]= ring[1];
ring[1]= ring[0];
ring[0]= i;

end

else begin
    if (m) begin // up counter
        q = temp1;
        q = q+1;
        temp1 = q;
        if (q== temp2)
            begin m<=0; temp1
=25'b000000000000000000000000;
            end
        end

    else
        begin
            q = temp2;
            q = q-1;
            temp2 = q;
            if (q==
25'b000000000000000000000000)
                begin m<= 1; temp2 =
25'b1110011100101011101100000; end
            end

            ring= ring+1;
            if (ring== 3'b111)begin
                ring= 3'b000;end
            end
        end
    end
end
endmodule

```

## Part II: Codes for PWM and control

## 2.1

**Generate a 1kHz PWM signal with 30% duty in one of the available GPIO pins. Use up counting mode. View the result on the oscilloscope.**

```
module simple30DC(
    input clk,
        //input reset,
    output reg [10:0] q,
        //input dip,
        //output reg [2:0] ring
    output reg op
);
reg [10:0] temp1 = 11'b1101101011000; //7000
in bin
reg [10:0] temp2 = 11'b0101110111000; //3000
in bin
reg [10:0] temp3 = 11'b0000000000000;
//0

reg m;
reg i;

initial begin
    m<=1;

end

always@(posedge clk)
begin
    if (m) begin // up counter

        q = temp3;
        q = q+1;
        temp1 = q;
        op=1;
        if (q> temp2)
            begin m<=0; temp3
            =11'b0000000000000;
            end
        end

    else
        begin
            q = temp1;
            q = q-1;
            temp2 = q;
            op = 0;
            if (q== temp3)
                begin m<= 1; temp1 =
                11'b1101101011000; end
            end
        end
    end
endmodule
```

## 2.2

**//Generate a 3 kHz PWM signal with 40% duty. Use down counting mode. Show the result in the oscilloscope. Generate its complementary signal on another GPIO pin.**

```
module pwmupcounter(
    input clk,
    output reg [10:0] q,
        //input dip,
        //output reg [2:0] ring
    output reg op,
    output reg opp
);
reg [10:0] temp1 = 10'b11111010000; //2000
in bin
reg [10:0] temp2 = 10'b10100110101; //1333 in
bin
reg [10:0] temp3 = 10'b00000000000; //0

reg m;
reg i;
```

```

initial begin
m<=1;

end

always@(posedge clk)
begin
    if (m) begin // up counter
        q = temp2;
        q = q-1;
        temp1 = q;
        op=1;
        opp= ~op;
        if (q== temp3)
            begin m<=0; temp2
=10'b10100110101;
            end
        end
    end

    else
        begin
            q = temp1;
            q = q-1;
            temp2 = q;
            op = 0;
            opp= ~op;
            if (q== temp3)
                begin m<= 1; temp1 =
10'b111111010000; end
            end
        end

m<=1;

end
endmodule

```

## 2.3

**Generate a 2 kHz PWM signal with 30% duty. Use up counting mode. Configure another GPIO to generate a similar PWM but 120 degree phase shifted from that of first PWM. Use one of the onboard switches/push-buttons to decrement the duty in steps of 5% (The duty should decrement by 5% when the switch is toggled/pressed). Use the other switch/pushbutton to decrement the phase shift in steps of 10 degrees. Show the results in the oscilloscope.**

```

module pwmwithswitch(
    input clk,
    input pwm,
    output reg [9:0] q,
    input dip,
    //output reg [2:0] ring
    output reg op,
    output reg op120
);
reg [9:0] temp1 = 10'b110110101100; //3500
in bin
reg [9:0] temp2 = 10'b010111011100; //1500 in
bin
reg [9:0] temp3 = 10'b000000000000; //0
reg [9:0] pwm5 = 10'b000011111010;
reg [9:0] phase = 10'b011010000011; // .33 of
5000 divisions

reg [9:0] phasediff= 10'b0000010001011; // 139
or 10deg
reg m;
//reg m120;
reg i;

initial begin
m<=1;
//m120<=1;
end

always@(posedge clk)
begin
    if (pwm) begin
        temp1= temp2-pwm5;
        temp2= temp1+ pwm5;
    end
end

```



```

if (dip) begin
phase= phase+ phasediff;
end

if (m) begin // up counter
q = temp3;
    q = q+1;
    temp3 = q;
    op=1;
    if (q>phase) begin
        op120= 1; end

        if (q> temp2)
            begin m<=0; temp3
=10'b00000000000000;
            end

        end
end

```

```

else

```

```

begin
if (q> temp2+phase)
begin
op120<=0;
end

q = temp3;
    q = q+1;
    temp3 = q;
    op = 0;
    if (q== temp1)
        begin m<= 1; temp3 = 10'b000000000000;

end

end

end
endmodule

```

# PART C

## PCB DESIGN FOR SENSING CIRCUIT FROM DIGITAL SIGNAL PROCESSORS OR DC-DC CONVERTERS

### 1. PROBLEM STATEMENT & ABSTRACT

Creating a single side through-hole PCB schematic and layout for sensing digital or analog feedback outputs from

1. a DC-DC converter simulated on a DSP OR
2. a physical DC-DC converter with adequate voltage divider across its output.

This PCB is designed to be a bridge to check outputs and convert it to an analog format if needed before relaying it to a DAC and checking synchronization and fidelity of the signal output from the converter. The challenge was to create a schematic with the right trace as the ICs likely to be compatible with most hardware and optimize for minimum cost.

### 2. OBJECTIVE

1. Creating a design for a low cost board
2. Learning to use the Eagle software for PCB design

### 3. CIRCUIT DESIGN

Female jumpers will be used to connect to the board and take samples from it. Standard male jumper headers have been used for low power i/o connections.

Two options are available to provide inputs to the board-

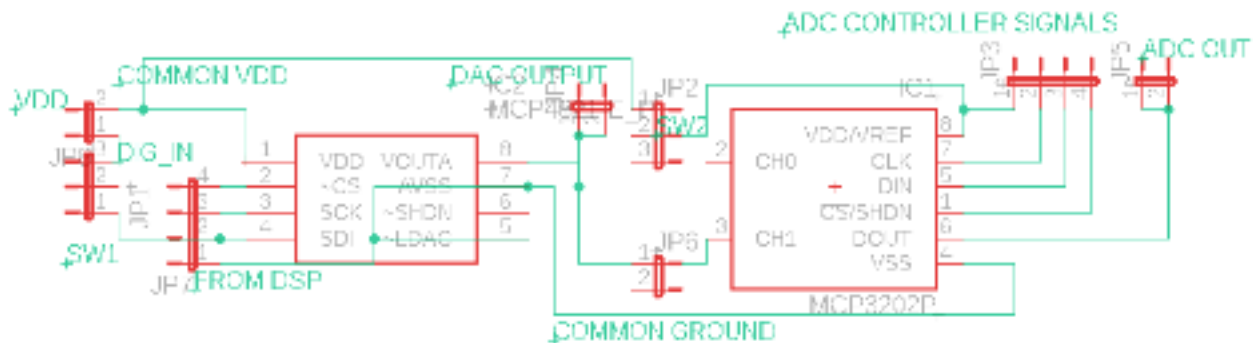
#### a. DIGITAL INPUT TO DAC:

DSP board digital output is converted into an analog signal (up to 3.3V) using the DAC chosen and can be tested. If needed, it can be sent to another controller via the jumper jack provided else the same signal can be converted back to digital using the low cost ADC chosen. We can provide a moderate voltage digital signal to employ digital control & feedback techniques externally.

#### b. ANALOG INPUT TO ADC:

A physical DC-DC converter can be stepped down to the input rail matching that of the ADC and from there can be sent forward to a DSP or any digital control circuit.

### i. FIG 1. SCHEMATIC



### ii. PARTS OF CIRCUIT (BASED ON SCHEMATIC IN FIG. 1)

1. DAC IC (IC #2), ADC IC (IC #1)
2. JP8- COMMON INPUT TO VCC OF BOTH ADC/DAC ICs, PROVIDED EXTERNALLY
3. SW2- CREATING A BUFFER TO CONNECT DAC IN THE CIRCUIT, USED TO PROVIDE DIGITAL INPUT FROM DSP TO THE DAC IC
4. JP4- JUMPERS TO CHECK DAC OUTPUT ON OSCILLOSCOPE
5. JP6- RELAY FROM DAC OUTPUT TO ADC INPUT
6. JP3- JUMPERS FROM MICROCONTROLLER OR DSP TO CONTROL THE ADC IC
7. JP5- ADC OUTPUT JUMPERS CAN BE CONNECTED

### iii. COST OPTIMIZATION

#### a. DAC SPECIFICATIONS

**Requisite features: Full-scale output of 2.5V, and operates from a single 2.7V to 5.5V supply**

The LTC2630 was recommended for use from standard DSP outputs. A cheaper alternative by Microchip was selected based on availability and standards of reliable operation.

LTC2630ISC6-LM1 2#TRMPBF	MCP4821: 12-Bit Voltage Output DAC
6 PIN PACKAGE \$4 (INR 276)	8 PIN PACKAGE INR 81

#### b. ADC SPECIFICATIONS

**Requisite features:**

12-Bit Resolution Zero Latency, Supply Range: 2.35 V to 5.25 V

Cheaper alternative procured at Microchip.

ADS7886SBDBVR	MCP 3202
Cost 284	Cost 183 rupees

**THE TOTAL COST REDUCTION IS INR 195+ 101= INR 296.**

### 4. ABOUT THE PCB DESIGN TOOL

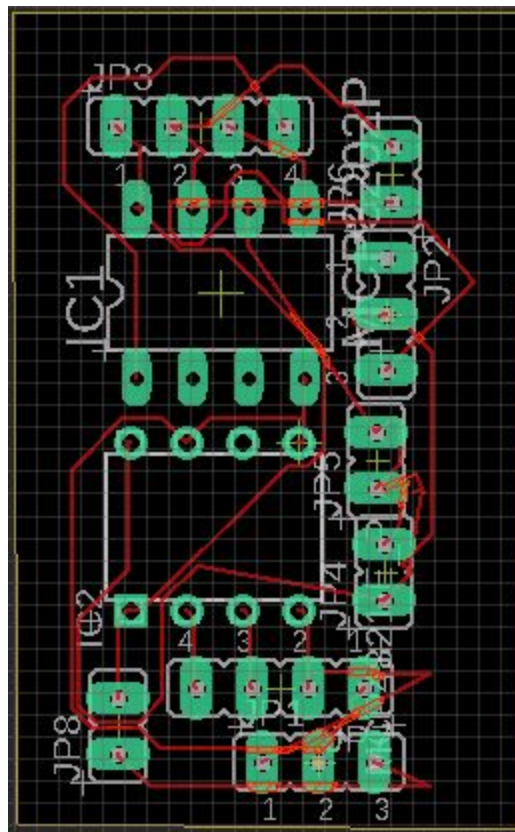
#### i. Eagle

An in-demand Electronic Design Automation software by AutoDesk. It offers several schematic editing and PCB layout tools. EAGLE contains a schematic editor, for designing circuit diagrams. Schematics are stored in files with .SCH extension, parts are defined in device libraries with .LBR extension. Parts can be placed on many sheets and connected together through ports. The PCB layout editor stores board files with the extension .BRD. It allows back-annotation to the schematic and auto-routing to automatically connect traces based on the connections defined in the schematic.

Track width and board size, as well as other hardware considerations need to be considered in the design before Autorouting.

EAGLE provides a multi-window graphical user interface and menu system for editing, project management and to customize the interface and design parameters. The system can be controlled via mouse, keyboard hotkeys or by entering specific commands at an embedded command line. Multiple repeating commands can be combined into script files (with file extension .SCR). It is also possible to explore design files utilizing an EAGLE-specific object-oriented programming language (with extension .ULP).

ii. FIG 2. PCB LAYOUT (grid size: 1mm)



## 5. PRECAUTIONS AND CONCLUSION:

1. The distance between the tracks and track width depends on current flowing through the circuit and for our circuit, the basic clearance of 5 mils is sufficient. It can support up to 0.5A with no trouble- the value is in accordance with the IPC Track Width Guidelines for 1oz Copper PCB.

2. The external connections outside the board have been placed at the periphery of the board, to reduce contact with sensitive components of the circuit. Other precautions on the board include A. allowing ample space between ICs to avoid criss-crossing and overlaps during routing & B. making sure the placement accounts for minimum track length- this is possible because we have no thermal considerations in this low power circuit.
3. An effective cost reduction of close to INR 300 is possible without affecting functionality of any applications.
4. The final single sided PCB layout is minimized in size for 100% efficiency and can be readily manufactured.

## **ACKNOWLEDGEMENTS**

Firstly, my wholehearted thanks to Dr. Lakshminarasamma under whose excellent guidance I was allowed to work for a duration of 2 months. I feel privileged to have been allowed to work on the problem statements that lend to ongoing research at IIT-M, an institute of worldwide renown. The access to the lab, the older students and discussions of the like I have been able to experience for the first time. I'd also like to thank the EE Department at IIT-M for organising and executing the Summer Fellowship Program seamlessly and facilitate my tenure there.

I would also thank all the faculty at the Department of Electronics and Telecommunication at College of Engineering, Pune and the Head of Department, Dr. S. P. Mahajan, for constant support and in their role in creating a foundation to build upon during this internship period.