**REPORT FILE**

1. **Group ID- 16**

**Name- Abhishek Rathod Enrollment No.- 17114004**

**Name- Anshuman Shakya Enrollment No.- 17114013**

1. **Group ID- 11**

**Name- Aman Jaiswal Enrollment No.- 17114008**

**Name- Amit Kumar Enrollment No.- 17114010**

**Description of the Project :**

We have made four .v files out of which 2 are modules which are Calculator.v and Seven\_seg\_output.v and 2 are test-benches which are Calc\_TestBench.v and calc\_tb2.v. We have implemented the GUI using Python. We have used the Tkinter library to create the GUI.

The modules are explained below -

The Calculator.v is our main design file. We have string as one input which takes a 14-bit instruction as input. The first 2 bits specify the operation code, the next 4-bits are for the first number and the next 4-bits are for the second number and the last 4-bits is for the output. We have 2 outputs , one as res which will be 4-bits BCD output and the other output as seven\_output which will be seven segment output.

The Seven\_seg\_output.v is the module for converting the input BCD as seven segment output.

The Calc\_TestBench.v is the test bench for the Calculator.v module which will give all the possible inputs to the module.

The calc\_tb2.v is the test bench which connects the GUI with the input and output files of the Verilog code. We have made 3 files, as interface between the GUI and the Verilog code. One file is for the input which takes the input from the GUI as entered by the user. The entered input in the GUI will then be stored in the input file. The data from the input file will be called by the test bench in the Xilinx. After getting the data, we will calculate the output using the Verilog module. And the result in the BCD form will then be stored in the output file. The GUI will then collect the data from the output file and display it on the user screen.

**VERILOG CODE of Design File(Calculator.v)-**

module Calculator(

input[13:0] string,

output reg[3:0] res,

output reg[6:0] seven\_output

);

always@\*

begin

if(string[13:12] == 2'b00)

assign res = string[11:8]+string[7:4] ;

else if(string[13:12] == 2'b01)

assign res = string[11:8]-string[7:4] ;

else if(string[13:12]== 2'b10)

assign res = string[11:8] | string[7:4] ;

else if(string[13:12] == 2'b11)

assign res = ~string[11:8]+4'b0001 ;

end

assign string[3:0] = res ;

//reg[6:0] seven\_output\_temp ;

/\*assign seven\_output = Seven\_seg\_output( .string(string[3:0])

);\*/

always @(res)

begin

case (res) //case statement

4'b0000 : seven\_output = 7'b0000001;

4'b0001 : seven\_output = 7'b1001111;

4'b0010 : seven\_output = 7'b0010010;

4'b0011 : seven\_output = 7'b0000110;

4'b0100 : seven\_output = 7'b1001100;

4'b0101 : seven\_output = 7'b0100100;

4'b0110 : seven\_output = 7'b0100000;

4'b0111 : seven\_output = 7'b0001111;

4'b1000 : seven\_output = 7'b0000000;

4'b1001 : seven\_output = 7'b0000100;

//switch off 7 segment character when the bcd digit is not a decimal number.

default : seven\_output = 7'b1111110;

endcase

end

endmodule

**Test Bench File-using VERILOG with output for all possible inputs (Calc\_TestBench.v)–**

module Calc\_TestBench;

// Inputs

reg [13:0] string;

// Outputs

wire [3:0] res;

wire [6:0] seven\_output;

// Instantiate the Unit Under Test (UUT)

Calculator uut (

.string(string),

.res(res),

.seven\_output(seven\_output)

);

integer i ;

integer j ;

initial begin

// Initialize Inputs

string[13:0]=14'b00000000000000 ;

for(i=0;i<16;i=i+1)

begin

for(j=0;j<16;j=j+1)

begin

string[7:4]= string[7:4]+4'b0001 ;

string[3:0]=string[11:8]+string[7:4] ;

string[13:12]=2'b00 ;

#0.5;

end;

string[11:8] = string[11:8]+4'b0001 ;

end;

string[13:0]=14'b01000000000000 ;

for( i=0;i<16;i=i+1)

begin

for( j=0;j<16;j=j+1)

begin

string[7:4]=string[7:4]+4'b0001;

string[3:0]=string[11:8]-string[7:4];

string[13:12] = 2'b01;

#0.5;

end;

string[11:8]=string[11:8]+4'b0001;

end;

string[13:0]=14'b10000000000000 ;

for( i=0;i<16;i=i+1)

begin

for( j=0;j<16;j=j+1)

begin

string[7:4]=string[7:4]+4'b0001;

string[3:0]=string[11:8]|string[7:4];

string[13:12] = 2'b10;

#0.5;

end;

string[11:8]=string[11:8]+4'b0001;

end;

string[13:0]=14'b11000000000000 ;

for( i=0;i<16;i=i+1)

begin

for( j=0;j<16;j=j+1)

begin

string[7:4]=string[7:4]+4'b0001;

string[3:0]=~string[11:8]+4'b0001;

string[13:12] = 2'b11;

#0.5;

end;

string[11:8]=string[11:8]+4'b0001;

end;

end

endmodule

**VERILOG CODE of BCD to Seven Segment Converter(Seven\_seg\_output.v) –**

module Seven\_seg\_output(

input[3:0] bcd\_op,

output reg[6:0] seven\_seg\_op

);

always @(bcd\_op)

begin

case (bcd\_op) //case statement

4'b0000 : seven\_seg\_op = 7'b0000001;

4'b0001 : seven\_seg\_op = 7'b1001111;

4'b0010 : seven\_seg\_op = 7'b0010010;

4'b0011 : seven\_seg\_op = 7'b0000110;

4'b0100 : seven\_seg\_op = 7'b1001100;

4'b0101 : seven\_seg\_op = 7'b0100100;

4'b0110 : seven\_seg\_op = 7'b0100000;

4'b0111 : seven\_seg\_op = 7'b0001111;

4'b1000 : seven\_seg\_op = 7'b0000000;

4'b1001 : seven\_seg\_op = 7'b0000100;

//switch off 7 segment character when the bcd digit is not a decimal number.

default : seven\_seg\_op = 7'b1111110;

endcase

end

endmodule

**Test Bench File-with GUI (calc\_tb2.v)–**

module calc\_tb2;

// Inputs

reg [13:0] string;

// Outputs

wire [3:0] res;

wire [6:0] seven\_output;

// Instantiate the Unit Under Test (UUT)

Calculator uut (

.string(string),

.res(res),

.seven\_output(seven\_output)

);

reg [13:0] mem[1:0];

integer id ;

integer i;

initial begin

// Initialize Inputs

id=$fopen("//home//ise//Documents//out1.txt","w" );

for(i=0;i<2;i=i+1)

begin

$readmemb("//home//ise//Documents//in.txt",mem);

string[13:0]=mem[i];

#10;

$fwrite(id,"%b\n",res);

#100;

// Add stimulus here

end

$fclose(id);

end

endmodule

**PYTHON CODE FOR GUI IMPLEMENTATION :**

# Python program to create a simple GUI

# calculator using Tkinter

#FROM GFG

# import everything from tkinter module

#pimport Tkinter as tk

from Tkinter import \*

# globally declare the expression variable

expression = ""

# Function to update expressiom

# in the text entry box

def press(num):

# point out the global expression variable

global expression

# concatenation of string

expression = expression + str(num)

# update the expression by using set method

equation.set(expression)

# Function to evaluate the final expression

def equalpress():

# Try and except statement is used

# for handling the errors like zero

# division error etc.

# Put that code inside the try block

# which may generate the error

try:

global expression

# eval function evaluate the expression

# and str function convert the result

# into string

# total = str(eval(expression))

exp=str(expression)

# print(exp)

ans=otb(exp[1])+dtb(exp[0])+dtb(exp[2])+"0000"

f=open('./in.txt','w')

f.write(str(ans))

f.close()

# equation.set("")

fi=open('./out1.txt','r')

# print("bjbjbjbj")

final=str(fi.read(4))

# print(final)

equation.set(str(btd(final)))

fi.close()

# initialze the expression variable

# by empty string

expression = ""

# if error is generate then handle

# by the except block

except:

equation.set(" error ")

expression = ""

def dtb(n):

if n=='1':

return "0001"

if n=='2':

return "0010"

if n=='3':

return "0011"

if n=='4':

return "0100"

if n=='5':

return "0101"

if n=='6':

return "0110"

if n=='7':

return "0111"

if n=='8':

return "1000"

if n=='9':

return "1001"

if n=='0':

return "0000"

def otb(n):

if n=='+':

return "00"

if n=='-':

return "01"

if n=='|':

return "10"

if n=='~':

return "11"

def btd(n):

if n=="0000":

return 0

if n=="0001":

return 1

if n=="0010":

return 2

if n=="0011":

return 3

if n=="0100":

return 4

if n=="0101":

return 5

if n=="0110":

return 6

if n=="0111":

return 7

if n=="1000":

return 8

if n=="1001":

return 9

# Function to clear the contents

# of text entry box

def clear():

global expression

expression = ""

equation.set("")

# Driver code

if \_\_name\_\_ == "\_\_main\_\_":

# create a GUI window

gui = Tk()

# set the background colour of GUI window

gui.configure(background="black")

# set the title of GUI window

gui.title("Simple Calculator")

# set the configuration of GUI window

gui.geometry("300x280+0+300")

# StringVar() is the variable class

# we create an instance of this class

equation = StringVar()

# create the text entry box for

# showing the expression .

expression\_field = Entry(gui, textvariable=equation)

# grid method is used for placing

# the widgets at respective positions

# in table like structure .

expression\_field.grid(columnspan=4, ipadx=70)

equation.set('0')

# create a Buttons and place at a particular

# location inside the root window .

# when user press the button, the command or

# function affiliated to that button is executed .

button1 = Button(gui, text=' 1 ', fg='black', bg='red',

command=lambda: press(1), height=1, width=7)

button1.grid(row=2, column=0)

button2 = Button(gui, text=' 2 ', fg='black', bg='red',

command=lambda: press(2), height=1, width=7)

button2.grid(row=2, column=1)

button3 = Button(gui, text=' 3 ', fg='black', bg='red',

command=lambda: press(3), height=1, width=7)

button3.grid(row=2, column=2)

button4 = Button(gui, text=' 4 ', fg='black', bg='red',

command=lambda: press(4), height=1, width=7)

button4.grid(row=3, column=0)

button5 = Button(gui, text=' 5 ', fg='black', bg='red',

command=lambda: press(5), height=1, width=7)

button5.grid(row=3, column=1)

button6 = Button(gui, text=' 6 ', fg='black', bg='red',

command=lambda: press(6), height=1, width=7)

button6.grid(row=3, column=2)

button7 = Button(gui, text=' 7 ', fg='black', bg='red',

command=lambda: press(7), height=1, width=7)

button7.grid(row=4, column=0)

button8 = Button(gui, text=' 8 ', fg='black', bg='red',

command=lambda: press(8), height=1, width=7)

button8.grid(row=4, column=1)

button9 = Button(gui, text=' 9 ', fg='black', bg='red',

command=lambda: press(9), height=1, width=7)

button9.grid(row=4, column=2)

button0 = Button(gui, text=' 0 ', fg='black', bg='red',

command=lambda: press(0), height=1, width=7)

button0.grid(row=5, column=0)

plus = Button(gui, text=' + ', fg='black', bg='red',

command=lambda: press("+"), height=1, width=7)

plus.grid(row=2, column=3)

minus = Button(gui, text=' - ', fg='black', bg='red',

command=lambda: press("-"), height=1, width=7)

minus.grid(row=3, column=3)

multiply = Button(gui, text=' | ', fg='black', bg='red',

command=lambda: press("|"), height=1, width=7)

multiply.grid(row=4, column=3)

divide = Button(gui, text='2s complement ', fg='black', bg='red',

command=lambda: press("~"), height=1, width=7)

divide.grid(row=5, column=3)

equal = Button(gui, text=' = ', fg='black', bg='red',

command=equalpress, height=1, width=7)

equal.grid(row=5, column=2)

clear = Button(gui, text='Clear', fg='black', bg='red',

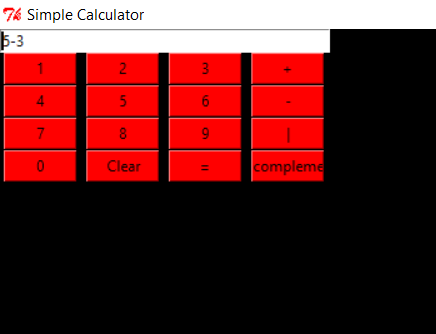
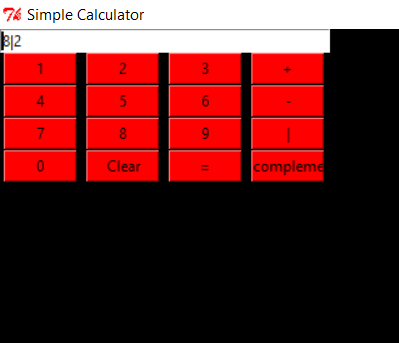
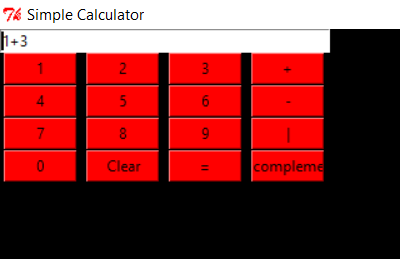
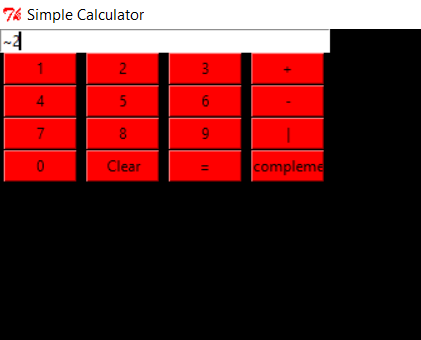
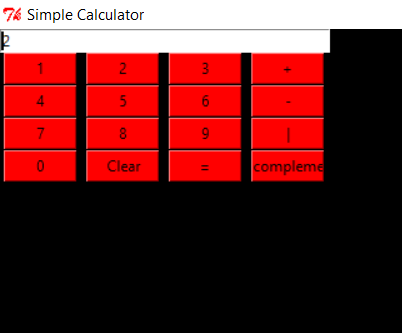
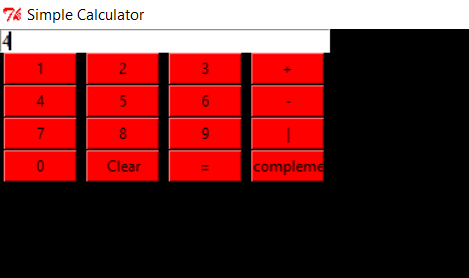
command=clear, height=1, width=7)

clear.grid(row=5, column='1')

# start the GUI

gui.mainloop()

**Screenshots for different inputs of GUI :**



**Screenshots of waveforms for different inputs :**

