**PWM Signal Verification with the help of LED**

**using Fuzzy Logic Controller**

**Verilog Code:**

// Code your design here

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/22/2024 06:30:56 PM

// Design Name:

// Module Name: TOP

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

// Code your design here

module TOP\_LED(

input CLK\_TOP,

input RESET\_TOP,

input [7:0] BCD\_TOP,

//output [7:0] digital\_freq,

output [7:0] DUTY\_TOP,

//output [7:0] fuzzy\_set\_id

output LED\_TOP

);

// PWM Output Pin of LED of FPGA/SoC Developement Board

wire PWM\_OUT\_TOP;

wire [7:0] DIGITAL\_FREQ\_TOP;

// Digital Output Frequency

wire [7:0] OUT\_DIG\_FREQ\_TOP;

//wire [7:0] output\_fuzzy\_set\_id;

// Input FUZZY FREQUENCY

wire [7:0] PTR\_FREQ\_FUZZY\_TOP;

wire [7:0] PTR\_INPUT\_FUZZY\_SET\_ID\_TOP;

//Input FUZZY SETS

wire [7:0] PTR\_negGIGANTIC\_TOP;

wire [7:0] PTR\_negHUGE\_TOP;

wire [7:0] PTR\_negBIG\_TOP;

wire [7:0] PTR\_negMEDIUM\_TOP;

wire [7:0] PTR\_negSMALL\_TOP;

wire [7:0] PTR\_ZERO\_TOP;

wire [7:0] PTR\_posSMALL\_TOP;

wire [7:0] PTR\_posMEDIUM\_TOP;

wire [7:0] PTR\_posBIG\_TOP;

wire [7:0] PTR\_posHUGE\_TOP;

wire [7:0] PTR\_posGIGANTIC\_TOP;

wire [7:0] OUTPUT\_FUZZY\_SET\_ID\_TOP;

wire [7:0] PWM\_DUTY\_CYCLE\_TOP;

//reg PROCESS\_Control = 2'b00;

// 1. MODULE to Digitize the BCD Value That is Received From The ADC

DIGITIZATION DIGITIZATION(

.CLK(CLK\_TOP),

.INPUT\_BIT(BCD\_TOP),

.FREQ(DIGITAL\_FREQ\_TOP),

.OUT\_DIG\_FREQ(OUT\_DIG\_FREQ\_TOP)

);

// 2. MODULE to Fuzzify The Input Frequency

FUZZIFICATION FUZZIFICATION(

.CLK(CLK\_TOP),

.FREQ\_CRISP(OUT\_DIG\_FREQ\_TOP),

.PTR\_FREQ\_FUZZY(PTR\_FREQ\_FUZZY\_TOP),

.PTR\_INPUT\_FUZZY\_SET\_ID(PTR\_INPUT\_FUZZY\_SET\_ID\_TOP),

.PTR\_negGIGANTIC(PTR\_negGIGANTIC\_TOP),

.PTR\_negHUGE(PTR\_negHUGE\_TOP),

.PTR\_negBIG(PTR\_negBIG\_TOP),

.PTR\_negMEDIUM(PTR\_negMEDIUM\_TOP),

.PTR\_negSMALL(PTR\_negSMALL\_TOP),

.PTR\_ZERO(PTR\_ZERO\_TOP),

.PTR\_posSMALL(PTR\_posSMALL\_TOP),

.PTR\_posMEDIUM(PTR\_posMEDIUM\_TOP),

.PTR\_posBIG(PTR\_posBIG\_TOP),

.PTR\_posHUGE(PTR\_posHUGE\_TOP),

.PTR\_posGIGANTIC(PTR\_posGIGANTIC\_TOP)

);

// 3. RULEBASE MODULE to Infer The Output Membership Function Based On The Input Membership Function

RULEBASE RULEBASE(

.CLK(CLK\_TOP),

.INPUT\_FUZZY\_SET\_ID(PTR\_INPUT\_FUZZY\_SET\_ID\_TOP),

.OUTPUT\_FUZZY\_SET\_ID(OUTPUT\_FUZZY\_SET\_ID\_TOP)

);

// 4. DEFUZZIFICATION MODULE to Defuzzify The Fuzzy Values Into a Crisp Output

DEFUZZIFICATION DEFUZZIFICATION(

.CLK(CLK\_TOP),

.OUTPUT\_FUZZY\_SET\_ID(OUTPUT\_FUZZY\_SET\_ID\_TOP),

.PWM\_DUTY(PWM\_DUTY\_CYCLE\_TOP)

);

PWM\_DUTY\_MOD PWM\_DUTY\_MOD(.CLK(CLK\_TOP),

.RESET(RESET\_TOP),

.PWM\_DUTY\_ONE(PWM\_DUTY\_CYCLE\_TOP),

.PWM\_OUT(PWM\_OUT\_TOP)

);

assign DUTY\_TOP = PWM\_DUTY\_CYCLE\_TOP;

//assign fuzzy\_set\_id = output\_fuzzy\_set\_id;

assign LED\_TOP = PWM\_OUT\_TOP;

endmodule

// Digitization Module

module DIGITIZATION(

input CLK,

input [7:0] INPUT\_BIT,

output [7:0] OUT\_DIG\_FREQ,

output [7:0] FREQ

);

reg BIT\_ONE = 1'b1;

reg BIT\_ZER = 1'b0;

reg [3:0] LSB\_DIG;

reg [3:0] MSB\_DIG;

reg [7:0] INTERNAL\_FREQ;

// This 'always' Block Only Converts The Data Bits Received In The PMOD Ports of The FPGA BOARD to Binary Digit Values.

always @(posedge CLK)

begin

// Aassigning Appropriate Values to The BCD LSB

if ((INPUT\_BIT[3] == BIT\_ZER) && (INPUT\_BIT[2] == BIT\_ZER) && (INPUT\_BIT[1] == BIT\_ZER) && (INPUT\_BIT[0] == BIT\_ZER))

begin

LSB\_DIG <= 4'b0000;

end

else if ((INPUT\_BIT[3] == BIT\_ZER) && (INPUT\_BIT[2] == BIT\_ZER) && (INPUT\_BIT[1] == BIT\_ZER) && (INPUT\_BIT[0] == BIT\_ONE))

begin

LSB\_DIG <= 4'b0001;

end

else if ((INPUT\_BIT[3] == BIT\_ZER) && (INPUT\_BIT[2] == BIT\_ZER) && (INPUT\_BIT[1] == BIT\_ONE) && (INPUT\_BIT[0] == BIT\_ZER))

begin

LSB\_DIG <= 4'b0010;

end

else if ((INPUT\_BIT[3] == BIT\_ZER) && (INPUT\_BIT[2] == BIT\_ZER) && (INPUT\_BIT[1] == BIT\_ONE) && (INPUT\_BIT[0] == BIT\_ONE))

begin

LSB\_DIG <= 4'b0011;

end

else if ((INPUT\_BIT[3] == BIT\_ZER) && (INPUT\_BIT[2] == BIT\_ONE) && (INPUT\_BIT[1] == BIT\_ZER) && (INPUT\_BIT[0] == BIT\_ZER))

begin

LSB\_DIG <= 4'b0100;

end

else if ((INPUT\_BIT[3] == BIT\_ZER) && (INPUT\_BIT[2] == BIT\_ONE) && (INPUT\_BIT[1] == BIT\_ZER) && (INPUT\_BIT[0] == BIT\_ONE))

begin

LSB\_DIG <= 4'b0101;

end

else if ((INPUT\_BIT[3] == BIT\_ZER) && (INPUT\_BIT[2] == BIT\_ONE) && (INPUT\_BIT[1] == BIT\_ONE) && (INPUT\_BIT[0] == BIT\_ZER))

begin

LSB\_DIG <= 4'b0110;

end

else if ((INPUT\_BIT[3] == BIT\_ZER) && (INPUT\_BIT[2] == BIT\_ONE) && (INPUT\_BIT[1] == BIT\_ONE) && (INPUT\_BIT[0] == BIT\_ONE))

begin

LSB\_DIG <= 4'b0111;

end

else if ((INPUT\_BIT[3] == BIT\_ONE) && (INPUT\_BIT[2] == BIT\_ZER) && (INPUT\_BIT[1] == BIT\_ZER) && (INPUT\_BIT[0] == BIT\_ZER))

begin

LSB\_DIG <= 4'b1000;

end

else if ((INPUT\_BIT[3] == BIT\_ONE) && (INPUT\_BIT[2] == BIT\_ZER) && (INPUT\_BIT[1] == BIT\_ZER) && (INPUT\_BIT[0] == BIT\_ONE))

begin

LSB\_DIG <= 4'b1001;

end

// Aassigning Appropriate Values to The BCD MSB

if ((INPUT\_BIT[7] == BIT\_ZER) && (INPUT\_BIT[6] == BIT\_ZER) && (INPUT\_BIT[5] == BIT\_ZER) && (INPUT\_BIT[4] == BIT\_ZER))

begin

MSB\_DIG <= 4'b0000;

end

else if ((INPUT\_BIT[7] == BIT\_ZER) && (INPUT\_BIT[6] == BIT\_ZER) && (INPUT\_BIT[5] == BIT\_ZER) && (INPUT\_BIT[4] == BIT\_ONE))

begin

MSB\_DIG <= 4'b0001;

end

else if ((INPUT\_BIT[7] == BIT\_ZER) && (INPUT\_BIT[6] == BIT\_ZER) && (INPUT\_BIT[5] == BIT\_ONE) && (INPUT\_BIT[4] == BIT\_ZER))

begin

MSB\_DIG <= 4'b0010;

end

else if ((INPUT\_BIT[7] == BIT\_ZER) && (INPUT\_BIT[6] == BIT\_ZER) && (INPUT\_BIT[5] == BIT\_ONE) && (INPUT\_BIT[4] == BIT\_ONE))

begin

MSB\_DIG <= 4'b0011;

end

else if ((INPUT\_BIT[7] == BIT\_ZER) && (INPUT\_BIT[6] == BIT\_ONE) && (INPUT\_BIT[5] == BIT\_ZER) && (INPUT\_BIT[4] == BIT\_ZER))

begin

MSB\_DIG <= 4'b0100;

end

else if ((INPUT\_BIT[7] == BIT\_ZER) && (INPUT\_BIT[6] == BIT\_ONE) && (INPUT\_BIT[5] == BIT\_ZER) && (INPUT\_BIT[4] == BIT\_ONE))

begin

MSB\_DIG <= 4'b0101;

end

else if ((INPUT\_BIT[7] == BIT\_ZER) && (INPUT\_BIT[6] == BIT\_ONE) && (INPUT\_BIT[5] == BIT\_ONE) && (INPUT\_BIT[4] == BIT\_ZER))

begin

MSB\_DIG <= 4'b0110;

end

else if ((INPUT\_BIT[7] == BIT\_ZER) && (INPUT\_BIT[6] == BIT\_ONE) && (INPUT\_BIT[5] == BIT\_ONE) && (INPUT\_BIT[4] == BIT\_ONE))

begin

MSB\_DIG <= 4'b0111;

end

else if ((INPUT\_BIT[7] == BIT\_ONE) && (INPUT\_BIT[6] == BIT\_ZER) && (INPUT\_BIT[5] == BIT\_ZER) && (INPUT\_BIT[4] == BIT\_ZER))

begin

MSB\_DIG <= 4'b1000;

end

else if ((INPUT\_BIT[7] == BIT\_ONE) && (INPUT\_BIT[6] == BIT\_ZER) && (INPUT\_BIT[5] == BIT\_ZER) && (INPUT\_BIT[4] == BIT\_ONE))

begin

MSB\_DIG <= 4'b1001;

end

end

always @\*

begin

INTERNAL\_FREQ <= (MSB\_DIG \* 4'b1010) + {3'b0, LSB\_DIG};

end

assign FREQ = INTERNAL\_FREQ;

reg [7:0] DIG\_FREQ;

parameter ZER = 5'b00000;

parameter ONE = 5'b00001;

parameter TWO = 5'b00010;

parameter THR = 5'b00011;

parameter FOR = 5'b00100;

parameter FIV = 5'b00101;

parameter SIX = 5'b00110;

parameter SEV = 5'b00111;

parameter EIG = 5'b01000;

parameter NIN = 5'b01001;

parameter TEN = 5'b01010;

parameter ELE = 5'b01011;

parameter TWE = 5'b01100;

parameter THI = 5'b01101;

parameter FRT = 5'b01110;

parameter FIF = 5'b01111;

parameter SXT = 5'b10000;

parameter SVT = 5'b10001;

parameter EIT = 5'b10010;

parameter NIT = 5'b10011;

parameter TWT = 5'b10100;

parameter TON = 5'b10101;

parameter FORTY\_eBIT = 8'b00101000;

parameter FORTY\_ONE\_eBIT = 8'b00101001;

parameter FORTY\_TWO\_eBIT = 8'b00101010;

parameter FORTY\_THREE\_eBIT = 8'b00101011;

parameter FORTY\_FOUR\_eBIT = 8'b00101100;

parameter FORTY\_FIVE\_eBIT = 8'b00101101;

parameter FORTY\_SIX\_eBIT = 8'b00101110;

parameter FORTY\_SEVEN\_eBIT = 8'b00101111;

parameter FORTY\_EIGHT\_eBIT = 8'b00110000;

parameter FORTY\_NINE\_eBIT = 8'b00110001;

parameter FIFTY\_eBIT = 8'b00110010;

parameter FIFTY\_ONE\_eBIT = 8'b00110011;

parameter FIFTY\_TWO\_eBIT = 8'b00110100;

parameter FIFTY\_THREE\_eBIT = 8'b00110101;

parameter FIFTY\_FOUR\_eBIT = 8'b00110110;

parameter FIFTY\_FIVE\_eBIT = 8'b00110111;

parameter FIFTY\_SIX\_eBIT = 8'b00111000;

parameter FIFTY\_SEVEN\_eBIT = 8'b00111001;

parameter FIFTY\_EIGHT\_eBIT = 8'b00111010;

parameter FIFTY\_NINE\_eBIT = 8'b00111011;

parameter SIXTY\_eBIT = 8'b00111100;

always @(posedge CLK)

begin

if(INTERNAL\_FREQ <= FORTY\_eBIT)

begin

DIG\_FREQ = ZER;

end

else if((INTERNAL\_FREQ > FORTY\_eBIT) && (INTERNAL\_FREQ <= FORTY\_ONE\_eBIT))

begin

DIG\_FREQ = ONE;

end

else if((INTERNAL\_FREQ > FORTY\_ONE\_eBIT ) && (INTERNAL\_FREQ <= FORTY\_TWO\_eBIT))

begin

DIG\_FREQ = TWO;

end

else if((INTERNAL\_FREQ > FORTY\_TWO\_eBIT) && (INTERNAL\_FREQ <= FORTY\_THREE\_eBIT))

begin

DIG\_FREQ = THR;

end

else if((INTERNAL\_FREQ > FORTY\_THREE\_eBIT) && (INTERNAL\_FREQ <= FORTY\_FOUR\_eBIT))

begin

DIG\_FREQ = FOR;

end

else if((INTERNAL\_FREQ > FORTY\_FOUR\_eBIT) && (INTERNAL\_FREQ <= FORTY\_FIVE\_eBIT))

begin

DIG\_FREQ = FIV;

end

else if((INTERNAL\_FREQ > FORTY\_FIVE\_eBIT) && (INTERNAL\_FREQ<= FORTY\_SIX\_eBIT))

begin

DIG\_FREQ = SIX;

end

else if((INTERNAL\_FREQ > FORTY\_SIX\_eBIT) && (INTERNAL\_FREQ <= FORTY\_SEVEN\_eBIT))

begin

DIG\_FREQ = SEV;

end

else if((INTERNAL\_FREQ > FORTY\_SEVEN\_eBIT) && (INTERNAL\_FREQ <= FORTY\_EIGHT\_eBIT))

begin

DIG\_FREQ = EIG;

end

else if((INTERNAL\_FREQ > FORTY\_EIGHT\_eBIT) && (INTERNAL\_FREQ <= FORTY\_NINE\_eBIT))

begin

DIG\_FREQ = NIN;

end

else if((INTERNAL\_FREQ > FORTY\_NINE\_eBIT) && (INTERNAL\_FREQ <= FIFTY\_eBIT))

begin

DIG\_FREQ = TEN;

end

else if((INTERNAL\_FREQ > FIFTY\_eBIT) && (INTERNAL\_FREQ <= FIFTY\_ONE\_eBIT))

begin

DIG\_FREQ = ELE;

end

else if((INTERNAL\_FREQ > FIFTY\_ONE\_eBIT) && (INTERNAL\_FREQ <= FIFTY\_TWO\_eBIT))

begin

DIG\_FREQ = TWE;

end

else if((INTERNAL\_FREQ > FIFTY\_TWO\_eBIT) && (INTERNAL\_FREQ <= FIFTY\_THREE\_eBIT))

begin

DIG\_FREQ = THI;

end

else if((INTERNAL\_FREQ > FIFTY\_THREE\_eBIT) && (INTERNAL\_FREQ <= FIFTY\_FOUR\_eBIT))

begin

DIG\_FREQ = FRT;

end

else if((INTERNAL\_FREQ > FIFTY\_FOUR\_eBIT) && (INTERNAL\_FREQ <= FIFTY\_FIVE\_eBIT))

begin

DIG\_FREQ = FIF;

end

else if((INTERNAL\_FREQ > FIFTY\_FIVE\_eBIT) && (INTERNAL\_FREQ <= FIFTY\_SIX\_eBIT))

begin

DIG\_FREQ = SXT;

end

else if((INTERNAL\_FREQ > FIFTY\_SIX\_eBIT) && (INTERNAL\_FREQ <= FIFTY\_SEVEN\_eBIT))

begin

DIG\_FREQ = SVT;

end

else if((INTERNAL\_FREQ > FIFTY\_SEVEN\_eBIT) && (INTERNAL\_FREQ <= FIFTY\_EIGHT\_eBIT))

begin

DIG\_FREQ = EIT;

end

else if((INTERNAL\_FREQ > FIFTY\_EIGHT\_eBIT) && (INTERNAL\_FREQ <= FIFTY\_NINE\_eBIT))

begin

DIG\_FREQ = NIT;

end

else if((INTERNAL\_FREQ > FIFTY\_NINE\_eBIT) && (INTERNAL\_FREQ <= SIXTY\_eBIT))

begin

DIG\_FREQ = TWT;

end

else if(INTERNAL\_FREQ > SIXTY\_eBIT)

begin

DIG\_FREQ = TON;

end

end

assign OUT\_DIG\_FREQ = DIG\_FREQ;

endmodule

// Fuzzification

module FUZZIFICATION (

input CLK,

input [7:0] FREQ\_CRISP,

output [7:0] PTR\_FREQ\_FUZZY,

output [7:0] PTR\_INPUT\_FUZZY\_SET\_ID,

output [7:0] PTR\_negGIGANTIC,

output [7:0] PTR\_negHUGE,

output [7:0] PTR\_negBIG,

output [7:0] PTR\_negMEDIUM,

output [7:0] PTR\_negSMALL,

output [7:0] PTR\_ZERO,

output [7:0] PTR\_posSMALL,

output [7:0] PTR\_posMEDIUM,

output [7:0] PTR\_posBIG,

output [7:0] PTR\_posHUGE,

output [7:0] PTR\_posGIGANTIC

);

reg [7:0] INT\_FREQ\_FUZZY;

reg [7:0] INT\_INPUT\_FUZZY\_SET\_ID;

reg [7:0] INT\_negGIGANTIC;

reg [7:0] INT\_negHUGE;

reg [7:0] INT\_negBIG;

reg [7:0] INT\_negMEDIUM;

reg [7:0] INT\_negSMALL;

reg [7:0] INT\_ZERO;

reg [7:0] INT\_posSMALL;

reg [7:0] INT\_posMEDIUM;

reg [7:0] INT\_posBIG;

reg [7:0] INT\_posHUGE;

reg [7:0] INT\_posGIGANTIC;

parameter ZERO\_eBIT = 8'b00000000;

parameter ONE\_eBIT = 8'b00000001;

parameter TWO\_eBIT = 8'b00000010;

parameter THREE\_eBIT = 8'b00000011;

parameter FOUR\_eBIT = 8'b00000100;

parameter FIVE\_eBIT = 8'b00000101;

parameter SIX\_eBIT = 8'b00000110;

parameter SEVEN\_eBIT = 8'b00000111;

parameter EIGHT\_eBIT = 8'b00001000;

parameter NINE\_eBIT = 8'b00001001;

parameter TEN\_eBIT = 8'b00001010;

parameter ELEVEN\_eBIT = 8'b00001011;

parameter TWELVE\_eBIT = 8'b00001100;

parameter THIRTEEN\_eBIT = 8'b00001101;

parameter FOURTEEN\_eBIT = 8'b00001110;

parameter FIFTEEN\_eBIT = 8'b00001111;

parameter SIXTEEN\_eBIT = 8'b00010000;

parameter SEVENTEEN\_eBIT = 8'b00010001;

parameter EIGHTEEN\_eBIT = 8'b00010010;

parameter NINETEEN\_eBIT = 8'b00010011;

parameter TWENTY\_eBIT = 8'b00010100;

// Internal Register to Hold The Crisp Value of Frequency

reg [7:0] INT\_FREQ\_CRISP;

always@ (posedge CLK)

begin

INT\_FREQ\_CRISP <= FREQ\_CRISP;

if (INT\_FREQ\_CRISP < ZERO\_eBIT)

begin

INT\_negGIGANTIC <= ONE\_eBIT; // Neg Gignatic

// Identifying The Selected Fuzzy Set

INT\_INPUT\_FUZZY\_SET\_ID <= ONE\_eBIT;

// The Input to Rule Base is Calculated With MAX Rule, Direct Assignment Works As Only One Value Exists

INT\_FREQ\_FUZZY <= INT\_negGIGANTIC;

/\* Set everything else to zero \*/

INT\_negHUGE <= 8'b00000000;

INT\_negBIG <= 8'b00000000;

INT\_negMEDIUM <= 8'b00000000;

INT\_negSMALL <= 8'b00000000;

INT\_ZERO <= 8'b00000000;

INT\_posSMALL <= 8'b00000000;

INT\_posMEDIUM <= 8'b00000000;

INT\_posBIG <= 8'b00000000;

INT\_posHUGE <= 8'b00000000;

INT\_posGIGANTIC <= 8'b00000000;

end

else if ((INT\_FREQ\_CRISP >= ZERO\_eBIT) && (INT\_FREQ\_CRISP < TWO\_eBIT))

begin

// Int\_negHuge <= FREQ\_CRISP

// int\_negGigantic <= (TWO\_eBIT - int\_freq\_crisp)

// Instead of a Division Algo, a Division by 2 LUT is created below.

case (FREQ\_CRISP)

ZERO\_eBIT:

begin

INT\_negHUGE <= ZERO\_eBIT; // 0 / 2 = 0

INT\_negGIGANTIC <= ONE\_eBIT; // (2-0) / 2 = 1

end

ONE\_eBIT:

begin

INT\_negHUGE <= ONE\_eBIT; // 1 / 2 = 1

INT\_negGIGANTIC <= ZERO\_eBIT; // (2-1) / 2 = 1

end

endcase

if (INT\_negHUGE >= INT\_negGIGANTIC)

begin

INT\_FREQ\_FUZZY <= INT\_negHUGE;

INT\_INPUT\_FUZZY\_SET\_ID <= TWO\_eBIT;

end

else

begin

INT\_FREQ\_FUZZY <= INT\_negGIGANTIC;

INT\_INPUT\_FUZZY\_SET\_ID <= ONE\_eBIT;

end

// Set Everything Else to Zero

INT\_negBIG <= 8'b00000000;

INT\_negMEDIUM <= 8'b00000000;

INT\_negSMALL <= 8'b00000000;

INT\_ZERO <= 8'b00000000;

INT\_posSMALL <= 8'b00000000;

INT\_posMEDIUM <= 8'b00000000;

INT\_posBIG <= 8'b00000000;

INT\_posHUGE <= 8'b00000000;

INT\_posGIGANTIC <= 8'b00000000;

end

else if ((INT\_FREQ\_CRISP >= TWO\_eBIT) && (INT\_FREQ\_CRISP < FOUR\_eBIT))

begin

case (FREQ\_CRISP)

TWO\_eBIT:

begin

// int\_negHuge <= (four\_ebit - int\_freq\_crisp) / two\_ebit;

INT\_negHUGE <= ONE\_eBIT; /\* (4 - 2) / 2 = 1 \*/

// int\_negBig <= (int\_freq\_crisp - two\_ebit) / two\_ebit;

INT\_negBIG <= ZERO\_eBIT; /\* (2 - 2) / 2 = 0 \*/

end

THREE\_eBIT:

begin

// int\_negHuge <= (four\_ebit - int\_freq\_crisp) / two\_ebit ;

INT\_negHUGE <= ZERO\_eBIT; /\* (4 - 3) / 2 = 1 \*/

// int\_negBig <= (int\_freq\_crisp - two\_ebit) / two\_ebit ;

INT\_negBIG <= ONE\_eBIT; /\* (3 - 2) / 2 = 0 \*/

end

endcase

if (INT\_negHUGE >= INT\_negBIG)

begin

INT\_FREQ\_FUZZY <= INT\_negHUGE;

INT\_INPUT\_FUZZY\_SET\_ID <= TWO\_eBIT;

end

else

begin

INT\_FREQ\_FUZZY <= INT\_negBIG;

INT\_INPUT\_FUZZY\_SET\_ID <= THREE\_eBIT;

end

// Set Everything Else to Zero

INT\_negGIGANTIC <= 8'b00000000;

INT\_negMEDIUM <= 8'b00000000;

INT\_negSMALL <= 8'b00000000;

INT\_ZERO <= 8'b00000000;

INT\_posSMALL <= 8'b00000000;

INT\_posMEDIUM <= 8'b00000000;

INT\_posBIG <= 8'b00000000;

INT\_posHUGE <= 8'b00000000;

INT\_posGIGANTIC <= 8'b00000000;

end

else if ((INT\_FREQ\_CRISP >= FOUR\_eBIT) && (INT\_FREQ\_CRISP < SIX\_eBIT))

begin

case (FREQ\_CRISP)

FOUR\_eBIT:

begin

// int\_negBig <= (six\_ebit - int\_freq\_crisp) / two\_ebit;

INT\_negBIG <= ONE\_eBIT; // (6 - 4) / 2 = 1 (44 Hz Should be Negative Big With 1 Membership Value)

// int\_negMedium <= (int\_freq\_crisp - 4) / two\_ebit; \*/

INT\_negMEDIUM <= ZERO\_eBIT; // (4 - 2) / 2 = 1 (44 Hz Should be Negative Medium with 0 Membership Value)

end

FIVE\_eBIT:

begin

// int\_negBig <= (six\_ebit - int\_freq\_crisp) / two\_ebit;

INT\_negBIG <= ZERO\_eBIT; // (6 - 5) / 2 = 0.5 (45 Hz should be negative Big with 0 membership value)

// int\_negMedium <= (int\_freq\_crisp - 4) / two\_ebit;

INT\_negMEDIUM <= ONE\_eBIT; // (5 - 4) / 2 = 0.5 (44 Hz should be negative medium with 1 membership value)

end

endcase

// Selecting the Maximum Value

if (INT\_negBIG >= INT\_negMEDIUM)

begin

INT\_FREQ\_FUZZY <= INT\_negBIG;

INT\_INPUT\_FUZZY\_SET\_ID <= THREE\_eBIT;

end

else

begin

INT\_FREQ\_FUZZY <= INT\_negMEDIUM;

INT\_INPUT\_FUZZY\_SET\_ID <= FOUR\_eBIT;

end

// Set Everything Else to Zero

INT\_negGIGANTIC <= 8'b00000000;

INT\_negHUGE <= 8'b00000000;

INT\_negSMALL <= 8'b00000000;

INT\_ZERO <= 8'b00000000;

INT\_posSMALL <= 8'b00000000;

INT\_posMEDIUM <= 8'b00000000;

INT\_posBIG <= 8'b00000000;

INT\_posHUGE <= 8'b00000000;

INT\_posGIGANTIC <= 8'b00000000;

end

else if ((INT\_FREQ\_CRISP >= SIX\_eBIT) && (INT\_FREQ\_CRISP < EIGHT\_eBIT))

begin

case (FREQ\_CRISP)

SIX\_eBIT:

begin

INT\_negMEDIUM <= ONE\_eBIT;

INT\_negSMALL <= ZERO\_eBIT;

end

SEVEN\_eBIT:

begin

INT\_negMEDIUM <= ZERO\_eBIT;

INT\_negSMALL <= ONE\_eBIT;

end

endcase

// Selecting the maximum value

if (INT\_negMEDIUM >= INT\_negSMALL)

begin

INT\_FREQ\_FUZZY <= INT\_negMEDIUM;

INT\_INPUT\_FUZZY\_SET\_ID <= FOUR\_eBIT;

end

else

begin

INT\_FREQ\_FUZZY <= INT\_negSMALL;

INT\_INPUT\_FUZZY\_SET\_ID <= FIVE\_eBIT;

end

// Set Everything Else to Zero

INT\_negGIGANTIC <= 8'b00000000;

INT\_negHUGE <= 8'b00000000;

INT\_negBIG <= 8'b00000000;

INT\_ZERO <= 8'b00000000;

INT\_posSMALL <= 8'b00000000;

INT\_posMEDIUM <= 8'b00000000;

INT\_posBIG <= 8'b00000000;

INT\_posHUGE <= 8'b00000000;

INT\_posGIGANTIC <= 8'b00000000;

end

else if ((INT\_FREQ\_CRISP >= EIGHT\_eBIT) && (INT\_FREQ\_CRISP < TWELVE\_eBIT))

begin

case (FREQ\_CRISP)

EIGHT\_eBIT:

begin

INT\_negSMALL <= ONE\_eBIT;

INT\_ZERO <= ZERO\_eBIT;

end

NINE\_eBIT:

begin

INT\_negSMALL <= ZERO\_eBIT;

INT\_ZERO <= ONE\_eBIT;

end

TEN\_eBIT:

begin

INT\_negSMALL <= ZERO\_eBIT;

INT\_ZERO <= ONE\_eBIT;

end

ELEVEN\_eBIT:

begin

INT\_negSMALL <= ZERO\_eBIT;

INT\_ZERO <= ONE\_eBIT;

end

endcase

// Selecting The Maximum Value

if (INT\_negSMALL >= INT\_ZERO)

begin

INT\_FREQ\_FUZZY <= INT\_negSMALL;

INT\_INPUT\_FUZZY\_SET\_ID <= FIVE\_eBIT;

end

else

begin

INT\_FREQ\_FUZZY <= INT\_ZERO;

INT\_INPUT\_FUZZY\_SET\_ID <= SIX\_eBIT;

end

// Set Everything Else to Zero

INT\_negGIGANTIC <= 8'b00000000;

INT\_negHUGE <= 8'b00000000;

INT\_negBIG <= 8'b00000000;

INT\_negMEDIUM <= 8'b00000000;

INT\_posSMALL <= 8'b00000000;

INT\_posMEDIUM <= 8'b00000000;

INT\_posBIG <= 8'b00000000;

INT\_posHUGE <= 8'b00000000;

INT\_posGIGANTIC <= 8'b00000000;

end

else if ((INT\_FREQ\_CRISP >= TWELVE\_eBIT) && (INT\_FREQ\_CRISP < FOURTEEN\_eBIT))

begin

case (FREQ\_CRISP)

TWELVE\_eBIT:

begin

INT\_ZERO <= ONE\_eBIT;

INT\_posSMALL <= ZERO\_eBIT;

end

THIRTEEN\_eBIT:

begin

INT\_ZERO <= ZERO\_eBIT;

INT\_posSMALL <= ONE\_eBIT;

end

endcase

// Selecting The Maximum Value

if (INT\_ZERO >= INT\_posSMALL)

begin

INT\_FREQ\_CRISP <= INT\_ZERO;

INT\_INPUT\_FUZZY\_SET\_ID <= SIX\_eBIT;

end

else

begin

INT\_FREQ\_FUZZY <= INT\_posSMALL;

INT\_INPUT\_FUZZY\_SET\_ID <= SEVEN\_eBIT;

end

// Set Everything Else to Zero

INT\_negGIGANTIC <= 8'b00000000;

INT\_negHUGE <= 8'b00000000;

INT\_negBIG <= 8'b00000000;

INT\_negMEDIUM <= 8'b00000000;

INT\_negSMALL <= 8'b00000000;

INT\_posMEDIUM <= 8'b00000000;

INT\_posBIG <= 8'b00000000;

INT\_posHUGE <= 8'b00000000;

INT\_posGIGANTIC <= 8'b00000000;

end

else if ((INT\_FREQ\_CRISP >= FOURTEEN\_eBIT) && (INT\_FREQ\_CRISP < SIXTEEN\_eBIT))

begin

case(FREQ\_CRISP)

FOURTEEN\_eBIT:

begin

INT\_posSMALL <= ONE\_eBIT;

INT\_posMEDIUM <= ZERO\_eBIT;

end

FIFTEEN\_eBIT:

begin

INT\_posSMALL <= ZERO\_eBIT;

INT\_posMEDIUM <= ONE\_eBIT;

end

endcase

if(INT\_posMEDIUM >= INT\_posSMALL)

begin

INT\_FREQ\_FUZZY <= INT\_posMEDIUM;

INT\_INPUT\_FUZZY\_SET\_ID <= EIGHT\_eBIT;

end

else

begin

INT\_FREQ\_FUZZY <= INT\_posSMALL;

INT\_INPUT\_FUZZY\_SET\_ID <= SEVEN\_eBIT;

end

// Set Everything Else to Zero

INT\_negGIGANTIC <= 8'b00000000;

INT\_negHUGE <= 8'b00000000;

INT\_negBIG <= 8'b00000000;

INT\_negMEDIUM <= 8'b00000000;

INT\_negSMALL <= 8'b00000000;

INT\_ZERO <= 8'b00000000;

INT\_posBIG <= 8'b00000000;

INT\_posHUGE <= 8'b00000000;

INT\_posGIGANTIC <= 8'b00000000;

end

else if ((INT\_FREQ\_CRISP >= SIXTEEN\_eBIT) && (INT\_FREQ\_CRISP < EIGHTEEN\_eBIT))

begin

case (FREQ\_CRISP)

SIXTEEN\_eBIT:

begin

INT\_posMEDIUM <= ONE\_eBIT;

INT\_posBIG <= ZERO\_eBIT;

end

SEVENTEEN\_eBIT:

begin

INT\_posMEDIUM <= ZERO\_eBIT;

INT\_posBIG <= ONE\_eBIT;

end

endcase

// Selecting The Maximum Value

if (INT\_posBIG >= INT\_posMEDIUM)

begin

INT\_FREQ\_FUZZY <= INT\_posBIG;

INT\_INPUT\_FUZZY\_SET\_ID <= NINE\_eBIT;

end

else

begin

INT\_FREQ\_FUZZY <= INT\_posMEDIUM;

INT\_INPUT\_FUZZY\_SET\_ID <= EIGHT\_eBIT;

end

// Set Everything Else to Zero

INT\_negGIGANTIC <= 8'b00000000;

INT\_negHUGE <= 8'b00000000;

INT\_negBIG <= 8'b00000000;

INT\_negMEDIUM <= 8'b00000000;

INT\_negSMALL <= 8'b00000000;

INT\_ZERO <= 8'b00000000;

INT\_posSMALL <= 8'b00000000;

INT\_posHUGE <= 8'b00000000;

INT\_posGIGANTIC <= 8'b00000000;

end

else if ((INT\_FREQ\_CRISP >= EIGHTEEN\_eBIT) && (INT\_FREQ\_CRISP <= TWENTY\_eBIT))

begin

case (FREQ\_CRISP)

EIGHTEEN\_eBIT:

begin

INT\_posBIG <= ONE\_eBIT;

INT\_posHUGE <= ZERO\_eBIT;

end

NINETEEN\_eBIT:

begin

INT\_posBIG <= ZERO\_eBIT;

INT\_posHUGE <= ONE\_eBIT;

end

endcase

// Selecting The Maximum Value

if (INT\_posBIG >= INT\_posHUGE)

begin

INT\_FREQ\_FUZZY <= INT\_posBIG;

INT\_INPUT\_FUZZY\_SET\_ID <= NINE\_eBIT;

end

else

begin

INT\_FREQ\_FUZZY <= INT\_posHUGE;

INT\_INPUT\_FUZZY\_SET\_ID <= TEN\_eBIT;

end

// Set Everything Else to Zero

INT\_negGIGANTIC <= 8'b00000000;

INT\_negHUGE <= 8'b00000000;

INT\_negBIG <= 8'b00000000;

INT\_negMEDIUM <= 8'b00000000;

INT\_negSMALL <= 8'b00000000;

INT\_ZERO <= 8'b00000000;

INT\_posSMALL <= 8'b00000000;

INT\_posMEDIUM <= 8'b00000000;

INT\_posGIGANTIC <= 8'b00000000;

end

else if (INT\_FREQ\_CRISP >= TWENTY\_eBIT)

begin

case (FREQ\_CRISP)

TWENTY\_eBIT:

begin

INT\_posGIGANTIC <= ONE\_eBIT;

end

endcase

// Selecting The Maximum Value

INT\_FREQ\_FUZZY <= INT\_posGIGANTIC;

INT\_INPUT\_FUZZY\_SET\_ID <= ELEVEN\_eBIT;

// Set Everything Else to Zero

INT\_negGIGANTIC <= 8'b00000000;

INT\_negHUGE <= 8'b00000000;

INT\_negBIG <= 8'b00000000;

INT\_negMEDIUM <= 8'b00000000;

INT\_negSMALL <= 8'b00000000;

INT\_ZERO <= 8'b00000000;

INT\_posSMALL <= 8'b00000000;

INT\_posMEDIUM <= 8'b00000000;

INT\_posBIG <= 8'b00000000;

INT\_posHUGE <= 8'b00000000;

end

end

assign PTR\_FREQ\_FUZZY = INT\_FREQ\_FUZZY;

assign PTR\_INPUT\_FUZZY\_SET\_ID = INT\_INPUT\_FUZZY\_SET\_ID;

assign PTR\_negGIGANTIC = INT\_negGIGANTIC;

assign PTR\_negHUGE = INT\_negHUGE;

assign PTR\_negBIG = INT\_negBIG;

assign PTR\_negMEDIUM = INT\_negMEDIUM;

assign PTR\_negSMALL = INT\_negSMALL;

assign PTR\_ZERO = INT\_ZERO;

assign PTR\_posSMALL = INT\_posSMALL;

assign PTR\_posMEDIUM = INT\_posMEDIUM;

assign PTR\_posBIG = INT\_posBIG;

assign PTR\_posHUGE = INT\_posHUGE;

assign PTR\_posGIGANTIC = INT\_posGIGANTIC;

endmodule

//RuleBase Mdoule

module RULEBASE(

input CLK,

input [7:0] INPUT\_FUZZY\_SET\_ID,

output [7:0] OUTPUT\_FUZZY\_SET\_ID

);

// Local Parameters to Represent The Numbers For Readability

localparam ONE\_eBIT = 8'b00000001;

localparam TWO\_eBIT = 8'b00000010;

localparam THREE\_eBIT = 8'b00000011;

localparam FOUR\_eBIT = 8'b00000100;

localparam FIVE\_eBIT = 8'b00000101;

localparam SIX\_eBIT = 8'b00000110;

localparam SEVEN\_eBIT = 8'b00000111;

localparam EIGHT\_eBIT = 8'b00001000;

localparam NINE\_eBIT = 8'b00001001;

localparam TEN\_eBIT = 8'b00001010;

localparam ELEVEN\_eBIT = 8'b00001011;

// Internal Registers to Handle the Input and Ouput Fuzzy Sets IDs

reg [7:0] INT\_INPUT\_ID;

reg [7:0] INT\_OUTPUT\_ID;

always@ (posedge CLK)

begin

INT\_INPUT\_ID = INPUT\_FUZZY\_SET\_ID;

if (INT\_INPUT\_ID == ONE\_eBIT)

begin

INT\_OUTPUT\_ID = ELEVEN\_eBIT;

end

else if (INT\_INPUT\_ID == TWO\_eBIT)

begin

INT\_OUTPUT\_ID = TEN\_eBIT;

end

else if (INT\_INPUT\_ID == THREE\_eBIT)

begin

INT\_OUTPUT\_ID = NINE\_eBIT;

end

else if (INT\_INPUT\_ID == FOUR\_eBIT)

begin

INT\_OUTPUT\_ID = EIGHT\_eBIT;

end

else if (INT\_INPUT\_ID == FIVE\_eBIT)

begin

INT\_OUTPUT\_ID = SEVEN\_eBIT;

end

else if (INT\_INPUT\_ID == SIX\_eBIT)

begin

INT\_OUTPUT\_ID = SIX\_eBIT;

end

else if (INT\_INPUT\_ID == SEVEN\_eBIT)

begin

INT\_OUTPUT\_ID = FIVE\_eBIT;

end

else if (INT\_INPUT\_ID == EIGHT\_eBIT)

begin

INT\_OUTPUT\_ID = FOUR\_eBIT;

end

else if (INT\_INPUT\_ID == NINE\_eBIT)

begin

INT\_OUTPUT\_ID = THREE\_eBIT;

end

else if (INT\_INPUT\_ID == TEN\_eBIT)

begin

INT\_OUTPUT\_ID = TWO\_eBIT;

end

else if (INT\_INPUT\_ID == ELEVEN\_eBIT)

begin

INT\_OUTPUT\_ID = ONE\_eBIT;

end

end

// Assigning The Value of The Internal Register to The Output

assign OUTPUT\_FUZZY\_SET\_ID = INT\_OUTPUT\_ID;

endmodule

// Defuzzification

module DEFUZZIFICATION(

input CLK,

input [7:0] OUTPUT\_FUZZY\_SET\_ID,

output [7:0] PWM\_DUTY

);

// Local Parameters to Represent The Numbers for Readability

parameter LEN\_FOUR = 3;

parameter LEN\_EIGHT = 7;

parameter ZERO\_eBIT = 8'b00000000;

parameter ONE\_eBIT = 8'b00000001;

parameter TWO\_eBIT = 8'b00000010;

parameter THREE\_eBIT = 8'b00000011;

parameter FOUR\_eBIT = 8'b00000100;

parameter FIVE\_eBIT = 8'b00000101;

parameter SIX\_eBIT = 8'b00000110;

parameter SEVEN\_eBIT = 8'b00000111;

parameter EIGHT\_eBIT = 8'b00001000;

parameter NINE\_eBIT = 8'b00001001;

parameter TEN\_eBIT = 8'b00001010;

parameter ELEVEN\_eBIT = 8'b00001011;

parameter TWENTY\_eBIT = 8'b00010100;

parameter THIRTY\_eBIT = 8'b00011110;

parameter FOURTY\_eBIT = 8'b00101000;

parameter FIFTY\_eBIT = 8'b00110010;

parameter SIXTY\_eBIT = 8'b00111100;

parameter SEVENTY\_eBIT = 8'b01000110;

parameter EIGHTY\_eBIT = 8'b01010000;

parameter NINETY\_eBIT = 8'b01011010;

parameter HUNDRED\_eBIT = 8'b01100100;

reg [LEN\_EIGHT:0] MEMO\_negGIGANTIC;

reg [LEN\_EIGHT:0] MEMO\_negHUGE;

reg [LEN\_EIGHT:0] MEMO\_negBIG;

reg [LEN\_EIGHT:0] MEMO\_negMEDIUM;

reg [LEN\_EIGHT:0] MEMO\_negSMALL;

reg [LEN\_EIGHT:0] MEMO\_ZERO;

reg [LEN\_EIGHT:0] MEMO\_posSMALL;

reg [LEN\_EIGHT:0] MEMO\_posMEDIUM;

reg [LEN\_EIGHT:0] MEMO\_posBIG;

reg [LEN\_EIGHT:0] MEMO\_posHUGE;

reg [LEN\_EIGHT:0] MEMO\_posGIGANTIC;

reg [7:0] DEGREE\_MEM;

reg [7:0] DUTY\_CYCLE;

always@ (posedge CLK)

begin

DEGREE\_MEM = OUTPUT\_FUZZY\_SET\_ID;

case (OUTPUT\_FUZZY\_SET\_ID)

ONE\_eBIT:

begin

// Division by 10 Causes a Floating Point Value, However, The Result Would Still be The Same Without it.

// Division by 10 is Removed;

MEMO\_negGIGANTIC = -(DEGREE\_MEM) + TEN\_eBIT; // -(1/10) + 10 = 9.9 : -(1) + 10 = 9 => NOT SELECTED

MEMO\_negHUGE = DEGREE\_MEM + 3'b100; // (1 / 10) = 0.1 : 1+4 = 5 => SELECTED

// Selecting The Lower Value of The Two

if (MEMO\_negGIGANTIC >= MEMO\_negHUGE)

DUTY\_CYCLE = MEMO\_negHUGE;

else

DUTY\_CYCLE = MEMO\_negGIGANTIC;

end

TWO\_eBIT:

begin

MEMO\_negHUGE = -(DEGREE\_MEM) + TWENTY\_eBIT; // (-2 / 10) + 20 = 19.8 : -2 + 20 = 18

MEMO\_negBIG = (DEGREE\_MEM) + TEN\_eBIT - 2'b10; // (2 / 10) + 10 = 10.2 : 2 + 10 - 2 = 12 => SELECTED\*

// Selecting The Lower of The Two Values

if (MEMO\_negHUGE >= MEMO\_negBIG)

DUTY\_CYCLE = MEMO\_negBIG;

else

DUTY\_CYCLE = MEMO\_negHUGE;

end

THREE\_eBIT:

begin

MEMO\_negBIG = -(DEGREE\_MEM) + THIRTY\_eBIT;

MEMO\_negMEDIUM = (DEGREE\_MEM) + TWENTY\_eBIT - 3'b100;

// Selecting The Lower of The Two Values

if (MEMO\_negBIG >= MEMO\_negMEDIUM)

DUTY\_CYCLE = MEMO\_negMEDIUM;

else

DUTY\_CYCLE = MEMO\_negBIG;

end

FOUR\_eBIT:

begin

MEMO\_negMEDIUM = -(DEGREE\_MEM) + FOURTY\_eBIT;

MEMO\_negSMALL = (DEGREE\_MEM) + THIRTY\_eBIT - 2'b10;

// Selecting The Lower of The Two Values

if (MEMO\_negMEDIUM >= MEMO\_negSMALL)

DUTY\_CYCLE = MEMO\_negSMALL;

else

DUTY\_CYCLE = MEMO\_negMEDIUM;

end

FIVE\_eBIT:

begin

MEMO\_negSMALL = -(DEGREE\_MEM) + FIFTY\_eBIT;

MEMO\_ZERO = (DEGREE\_MEM) + FOURTY\_eBIT; //

// Selecting the lower of the two values

if (MEMO\_ZERO >= MEMO\_negSMALL)

DUTY\_CYCLE = MEMO\_negSMALL;

else

DUTY\_CYCLE = MEMO\_ZERO;

end

SIX\_eBIT:

begin

// For The Singleton Function at The Middle, Only One Value Exists

MEMO\_ZERO = FIFTY\_eBIT;

DUTY\_CYCLE = MEMO\_ZERO;

end

SEVEN\_eBIT:

begin

MEMO\_ZERO = -(DEGREE\_MEM) + SIXTY\_eBIT;

MEMO\_posSMALL = (DEGREE\_MEM) + FIFTY\_eBIT;

// Selecting The Lower of The Two Values

if (MEMO\_ZERO >= MEMO\_posSMALL)

DUTY\_CYCLE = MEMO\_ZERO;

else

DUTY\_CYCLE = MEMO\_posSMALL;

end

EIGHT\_eBIT:

begin

MEMO\_posSMALL = -(DEGREE\_MEM) + SEVENTY\_eBIT;

MEMO\_posMEDIUM = (DEGREE\_MEM) + SIXTY\_eBIT;

// Selecting The Lower of The Two Values

if (MEMO\_posMEDIUM >= MEMO\_posSMALL)

DUTY\_CYCLE = MEMO\_posMEDIUM;

else

DUTY\_CYCLE = MEMO\_posSMALL;

end

NINE\_eBIT:

begin

MEMO\_posMEDIUM = -(DEGREE\_MEM) + EIGHTY\_eBIT;

MEMO\_posBIG = (DEGREE\_MEM) + SEVENTY\_eBIT;

// Selecting The Lower of The Two Values

if (MEMO\_posMEDIUM >= MEMO\_posBIG)

DUTY\_CYCLE = MEMO\_posMEDIUM;

else

DUTY\_CYCLE = MEMO\_posBIG;

end

TEN\_eBIT:

begin

MEMO\_posBIG = -(DEGREE\_MEM) + NINETY\_eBIT;

MEMO\_posHUGE = (DEGREE\_MEM) + EIGHTY\_eBIT;

// Selecting The Lower of The Two Values

if (MEMO\_posHUGE >= MEMO\_posBIG)

DUTY\_CYCLE = MEMO\_posHUGE;

else

DUTY\_CYCLE = MEMO\_posBIG;

end

ELEVEN\_eBIT:

begin

MEMO\_posHUGE = -(DEGREE\_MEM) + HUNDRED\_eBIT;

MEMO\_posGIGANTIC = (DEGREE\_MEM) + NINETY\_eBIT;

// Selecting The Lower of The Two Values

if (MEMO\_posHUGE >= MEMO\_posGIGANTIC)

DUTY\_CYCLE = MEMO\_posHUGE;

else

DUTY\_CYCLE = MEMO\_posGIGANTIC - 3'b101;

end

endcase

end

// Assigning The Value of Internal Register to Output Port

assign PWM\_DUTY = DUTY\_CYCLE;

endmodule

// For LED Intensity Variation

module PWM\_DUTY\_MOD(

input CLK,

input RESET,

input [7:0] PWM\_DUTY\_ONE,

output PWM\_OUT

);

//Up counter

reg[7:0] Q\_REG, Q\_NEXT;

//reg [8:0] duty = 0;

reg [7:0] MEM\_PWM\_DUTY;

always @(posedge CLK)

begin

MEM\_PWM\_DUTY = PWM\_DUTY\_ONE;

if(~RESET == 1)

begin

Q\_REG <= 1'b0;

end

else

begin

Q\_REG <= Q\_NEXT;

end

end

//Next State Logic

always @(\*)

begin

Q\_NEXT = Q\_REG + 1;

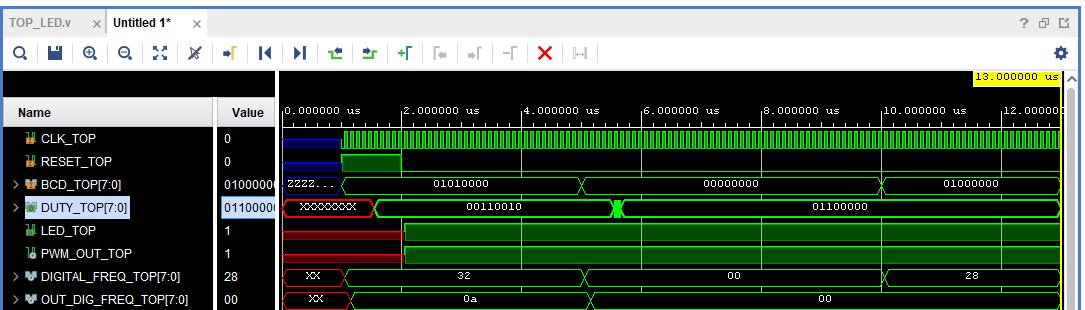
end

//Output Logic

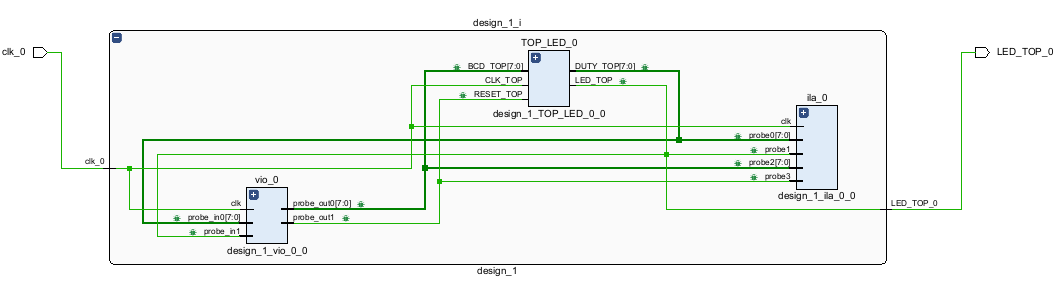
assign PWM\_OUT = (Q\_REG < MEM\_PWM\_DUTY);

endmodule

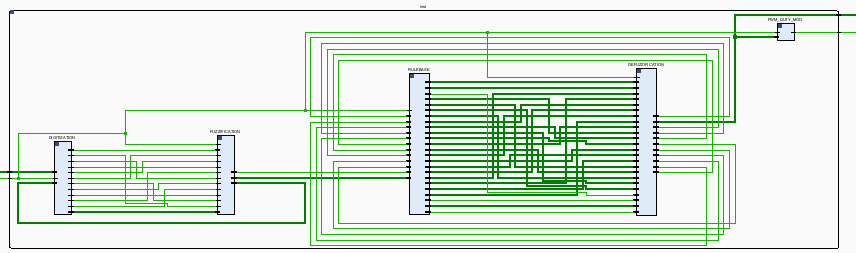
**Simulation Result on VIVADO:**

****

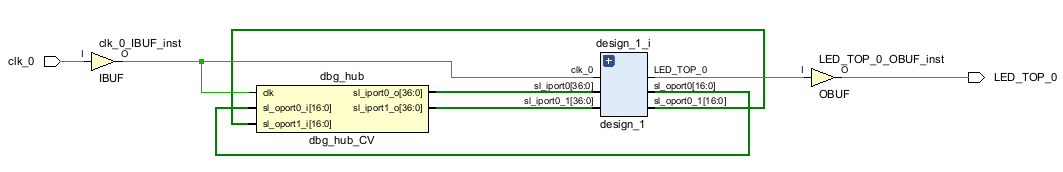
**Schematic View of Verification of TOP Module using VIO (Virtual Logic Analyzer) and ILA(Integrated Logic Analyzer) IPs of Vivado:**

****

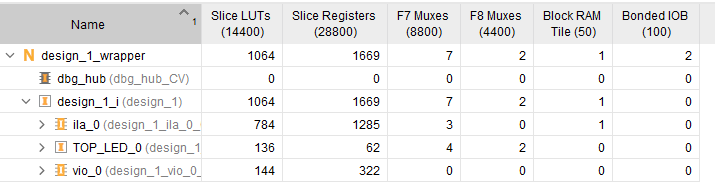
**Schematic View of TOP RTL Block:**

****

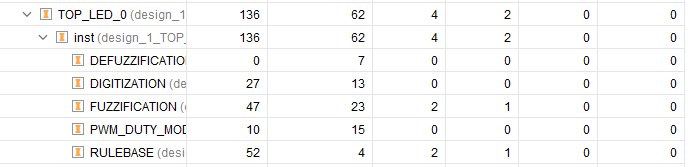
**Synthesized Schematic View after of Verification Setup:**

****

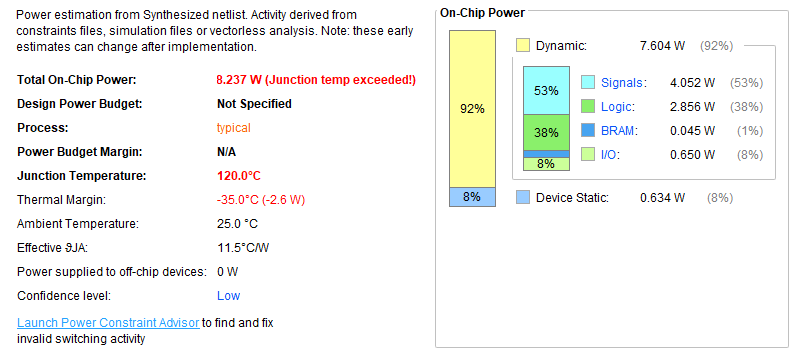
**Report Resourced Used after Synthesis:**



**Broad View of TOP RTL Block what I have designed is,**



**Power Report After Synthesis:**

****

**Table of Contents**

**1. Slice Logic**

**1.1 Summary of Registers by Type**

**2. Memory**

**3. DSP**

**4. IO and GT Specific**

**5. Clocking**

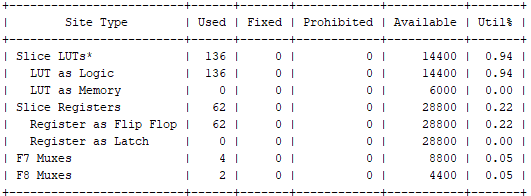
**6. Specific Feature**

**7. Primitives**

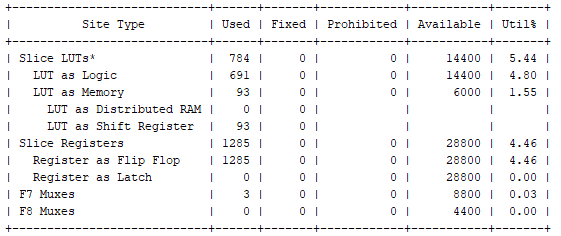
**8. Black Boxes**

**9. Instantiated Netlists**

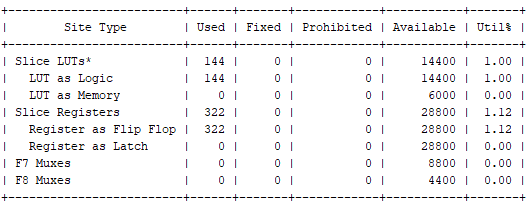
**1. Slice Logic by TOP Block:**

****

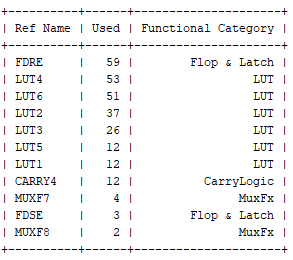
**By ILA:**

****

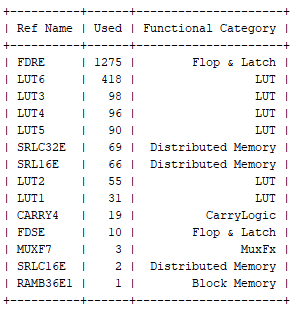
**By VIO:**

****

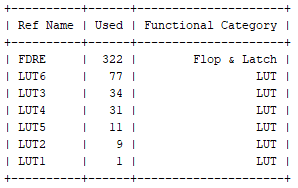
**7. Primitives by TOP block:**

****

**By ILA:**

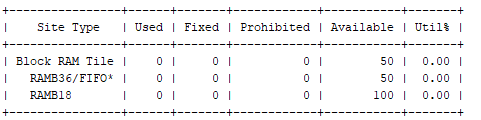
****

**By VIO:**

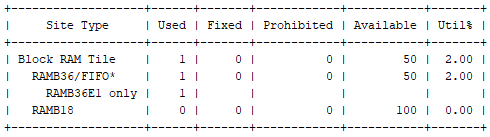
****

**Memory Used:**

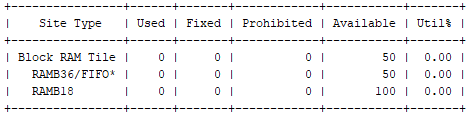
**By TOP:**

****

**By ILA:**

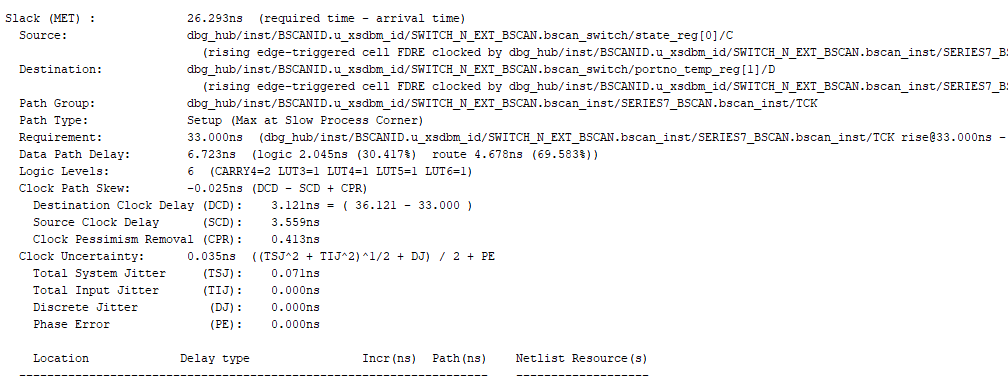
****

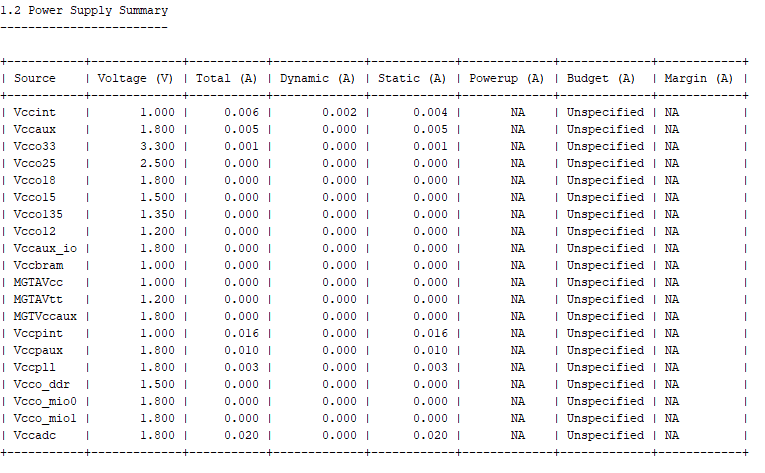
**By VIO:**

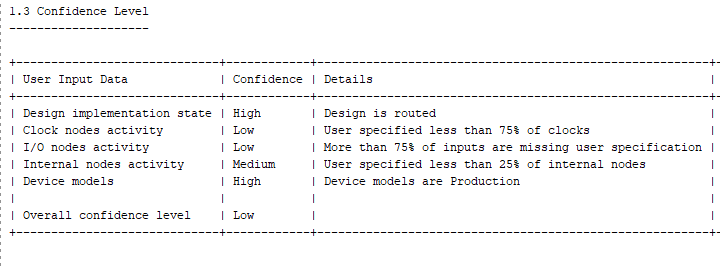
****

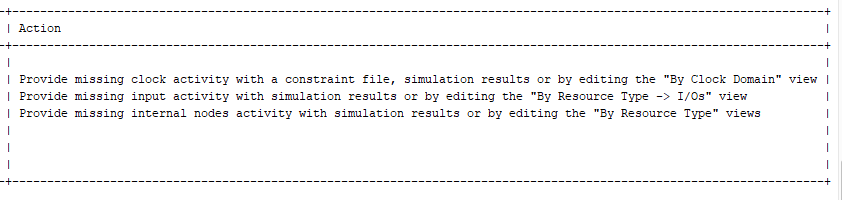
**Post Implementation Report:**

**Slack:**

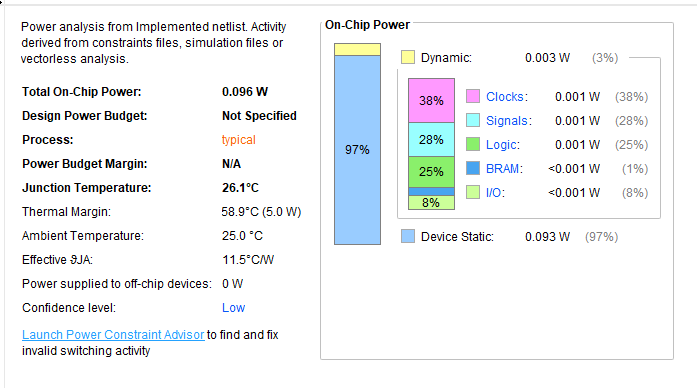
****

****

****

****

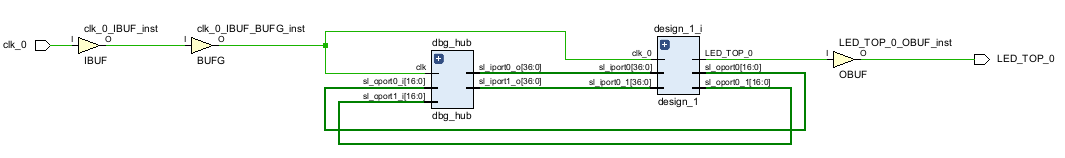
**Power Report:**

****

**Physical Constraints for I/O Pin:**

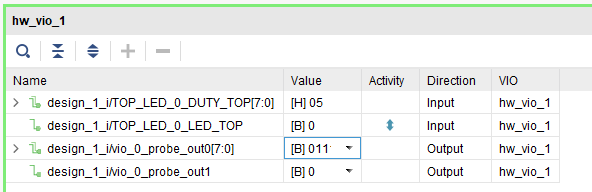
****

**Synthesized Schematic Diagram of Testing setup after Implementation:**

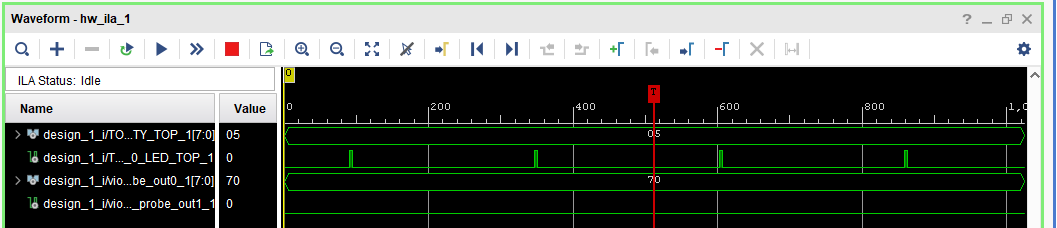
****

**Hardware Implementation and Verification using LED and VIO&ILA IPs(Hardware Manager Vivado) of FPGA/SoC (XC7Z007S-1CLG400C) of PWM Signal Generation:**

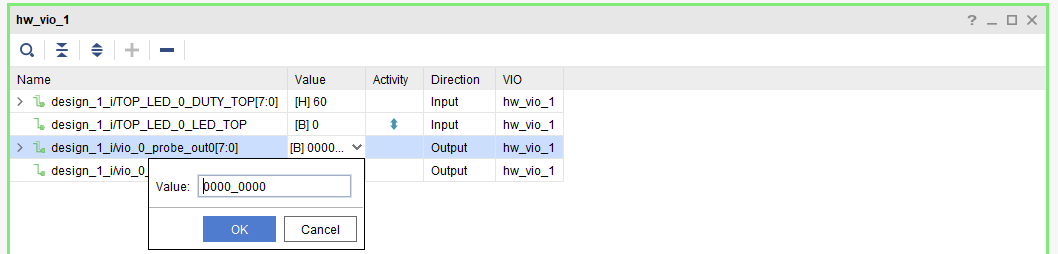
**VIO Inputs**

****

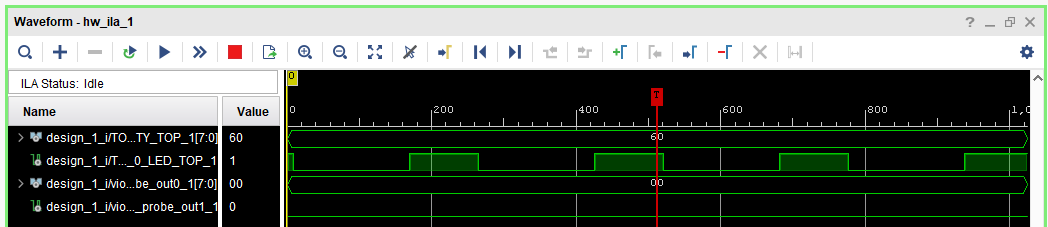
**ILA Outputs Waveform:**

****

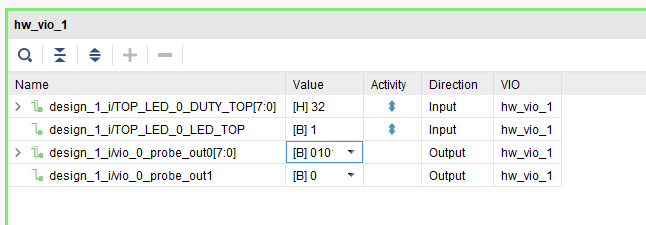
**2nd input**

****

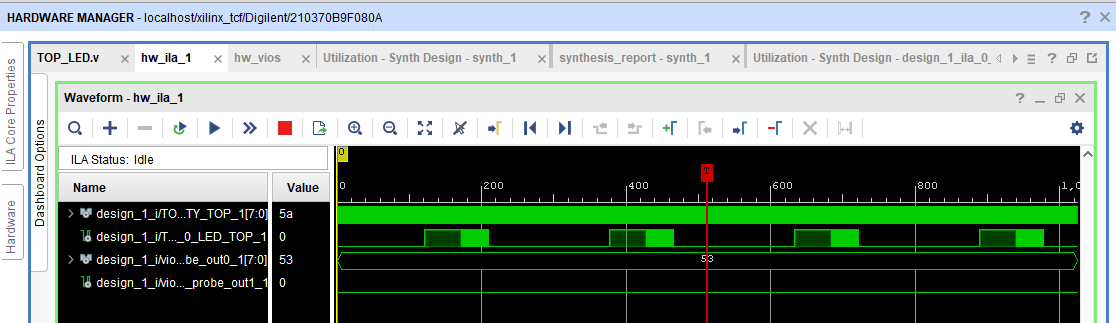
**2nd output:**

****

**3rd input:**

****

**3rd Output:**

****

**Hardware Implementation Video:**

****