|  |
| --- |
| **LAB 1** |

**IMPLEMENTATION OF BASIC COMBINATION LOGIC CIRCUIT USING VERILOG**

Nguyen Gia Cat Tuong ITITIU21117

### II.1 LAB EXPERIMENT 1 : WRITE HDL CODE TO REALIZE ALL LOGIC GATES

**// data flow**

module lab1\_ex1(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR=SW;

allgate\_DF DUT(.a(SW[1]),.b(SW[0]),.yand(LEDG[6]),.yor(LEDG[5]),.ynot(LEDG[4]),.ynand(LEDG[3]),.ynor(LEDG[2]),.yxor(LEDG[1]),.yxnor(LEDG[0]));

endmodule

module allgate\_DF ( a, b, yand,yor,ynot,ynand,ynor,yxor,yxnor );

input a,b;

output yand, yor, ynot, ynand, ynor, yxor, yxnor;

assign yand = a & b; // AND Operation

assign yor = a | b; // OR Operation

assign ynot = ~a ; // NOT Operation

assign ynand = ~(a & b); // NAND Operation

assign ynor = ~(a | b); //NOR Operation

assign yxor = a ^ b; //XOR Operation

assign yxnor =~(a^b); //XNOR Operation

endmodule  
 A screenshot of a computer

Description automatically generated

**// Structural**

module lab1\_ex1(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR=SW;

allgate DUT(SW[1],SW[2],LEDG[5],LEDG[4],LEDG[3],LEDG[2],LEDG[1],LEDG[0]);

endmodule

module allgate ( a, b, yand,yor,ynot,ynand,ynor,yxor,yxnor );

input a,b;

output yand, yor, ynot, ynand, ynor, yxor, yxnor;

assign yand = a & b; // AND Operation

assign yor = a | b; // OR Operation

assign ynot = ~a ; // NOT Operation

assign ynand = ~(a & b); // NAND Operation

assign ynor = ~(a | b); //NOR Operation

assign yxor = a ^ b; //XOR Operation

assign yxnor =~(a^b); //XNOR Operation

endmodule

A screenshot of a computer

Description automatically generated

**// behavior**

module lab1\_ex1(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR=SW;

allgate\_BH DUT(SW[1],SW[2],LEDG[5],LEDG[4],LEDG[3],LEDG[2],LEDG[1],LEDG[0]);

endmodule

module allgate\_BH ( a, b, yand,yor,ynot,ynand,ynor,yxor,yxnor );

input a,b;

output yand, yor, ynot, ynand, ynor, yxor, yxnor;

reg yand, yor, ynot, ynand, ynor, yxor, yxnor;

always @(\*)

begin

yand = a & b; // AND Operation

yor = a | b; // OR Operation

ynot = ~a ; // NOT Operation

ynand = ~(a & b); // NAND Operation

ynor = ~(a | b); //NOR Operation

yxor = a ^ b; //XOR Operation

yxnor =~(a^b); //XNOR Operation

end

endmodule

A screenshot of a computer

Description automatically generated

**II.2 LAB EXPERIMENT 2 : WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE HALF ADDER CIRCUIT:**

**// Structural**

module Lab1\_task2\_STRUC(SW,LEDR,LEDG);

input[0:17] SW;

output[0:7] LEDR;

output[0:17] LEDG;

assign LEDG=SW;

half\_adder\_structeral DUT(.a(SW[0]),.b(SW[1]),.s(LEDR[0]),.Cout(LEDR[1]));

endmodule

module half\_adder\_structeral(input a, b, output s, Cout);

xor G1(s,a,b);

and G2(Cout,a,b);

endmodule  
A screenshot of a computer

Description automatically generated

**// data flow**

module Lab1\_task2(SW,LEDR,LEDG);

input[0:17] SW;

output[0:7] LEDR;

output[0:17] LEDG;

assign LEDG=SW;

half\_adder\_dataflow DUT(.a(SW[0]),.b(SW[1]),.s(LEDR[0]),.Cout(LEDR[1]));

endmodule

module half\_adder\_dataflow(input a, b, output s, Cout);

assign s = a ^ b;

assign Cout = a & b;

endmodule  
A screenshot of a computer

Description automatically generated

// **Behavior**module Lab1\_task2(SW,LEDR,LEDG);

input[0:17] SW;

output[0:7] LEDR;

output[0:17] LEDG;

assign LEDG=SW;

half\_adder\_behavior DUT(.a(SW[0]),.b(SW[1]),.sum(LEDR[0]),.carry(LEDR[1]));

endmodule

module half\_adder\_behavior(sum,carry,a,b );

output sum,carry;

input a,b;

reg sum,carry;

always @(a,b)

begin

sum <= a ^ b;

carry <= a&b ;

end

endmoduleA screenshot of a computer

Description automatically generated

**II.3 EXPERIMENT 3: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE FULL ADDER CIRCUIT:**

**// dataflow**

module Lab1\_task3(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR= SW;

full\_adder\_DTFL DUT(SW[2],SW[1],SW[0],LEDG[1],LEDG[0]);

endmodule

module full\_adder\_DTFL(input a, b, c, output Sum, Cout);

assign Sum= a ^ b ^ c;

assign Cout= (a&b)|(b&c)|(a&c);

endmodule

A screenshot of a computer

Description automatically generated

// **Structural**

module Lab1\_task3(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR= SW;

full\_adder\_STRU DUT(SW[2],SW[1],SW[0],LEDG[1],LEDG[0]);

endmodule

module full\_adder\_STRU(A,B,Cin,Sum,Carry);

output Sum,Carry;

input A,B,Cin;

wire x,y,z;

xor g1(x,A,B);

xor g2(Sum,x,Cin);

and g3(y,x,Cin);

and g4(z,A,B);

or g5(Carry,x,y);

endmodule

A screenshot of a computer

Description automatically generated

**// Behavior**

module Lab1\_task3(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR= SW;

full\_adder\_BH DUT(SW[2],SW[1],SW[0],LEDG[1],LEDG[0]);

endmodule

module full\_adder\_BH(a,b,c,sum,carry);

output sum,carry;

input a,b,c;

reg sum,carry;

always @ (a,b,c)

begin

sum <= a^ b^c;

carry <=(a&b) | (b&c) | (c&a);

end

endmodule

A screenshot of a computer

Description automatically generated

module Lab1\_task3(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR= SW;

full\_adder DUT(SW[2],SW[1],SW[0],LEDG[1],LEDG[0]);

endmodule

module full\_adder( A, B, Cin, S, Cout);

input wire A, B, Cin;

output reg S, Cout;

always @(A or B or Cin)

begin

if(A==0 && B==0 && Cin==0)

begin

S=0;

Cout=0;

end

else if(A==0 && B==0 && Cin==1)

begin

S=1;

Cout=0;

end

else if(A==0 && B==1 && Cin==0)

begin

S=1;

Cout=0;

end

else if(A==0 && B==1 && Cin==1)

begin

S=0;

Cout=1;

end

else if(A==1 && B==0 && Cin==0)

begin

S=1;

Cout=0;

end

else if(A==1 && B==0 && Cin==1)

begin

S=0;

Cout=1;

end

else if(A==1 && B==1 && Cin==0)

begin

S=0;

Cout=1;

end

else if(A==1 && B==1 && Cin==1)

begin

S=1;

Cout=1;

end

end

endmodule

A screenshot of a computer

Description automatically generated