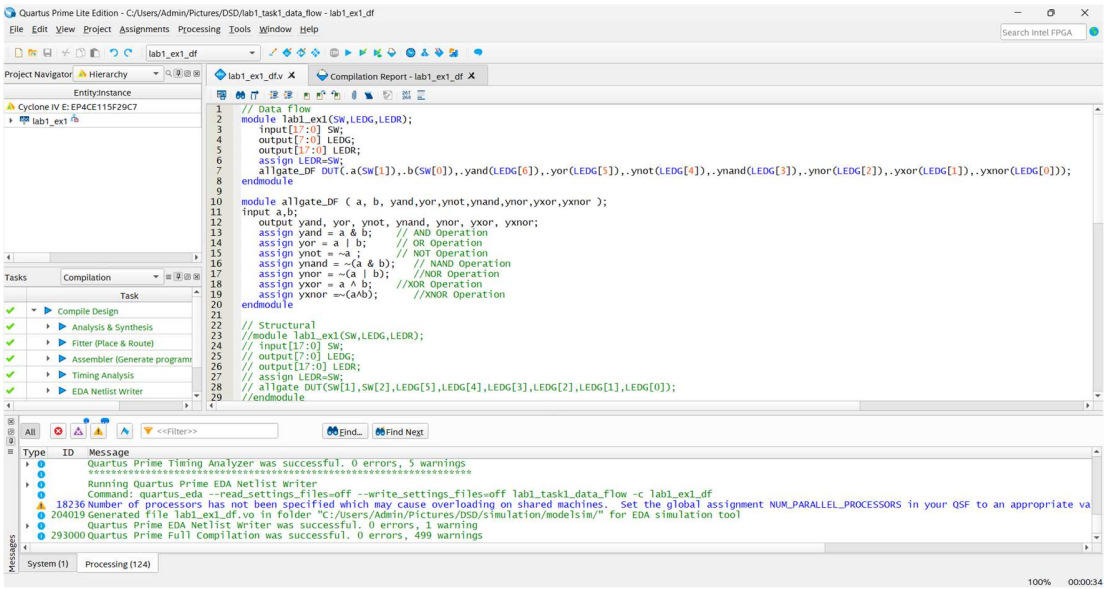
LAB 1

IMPLEMENTATION OF BASIC COMBINATION LOGIC CIRCUIT USING VERILOG

Phạm Quang Vinh ITITIU21347

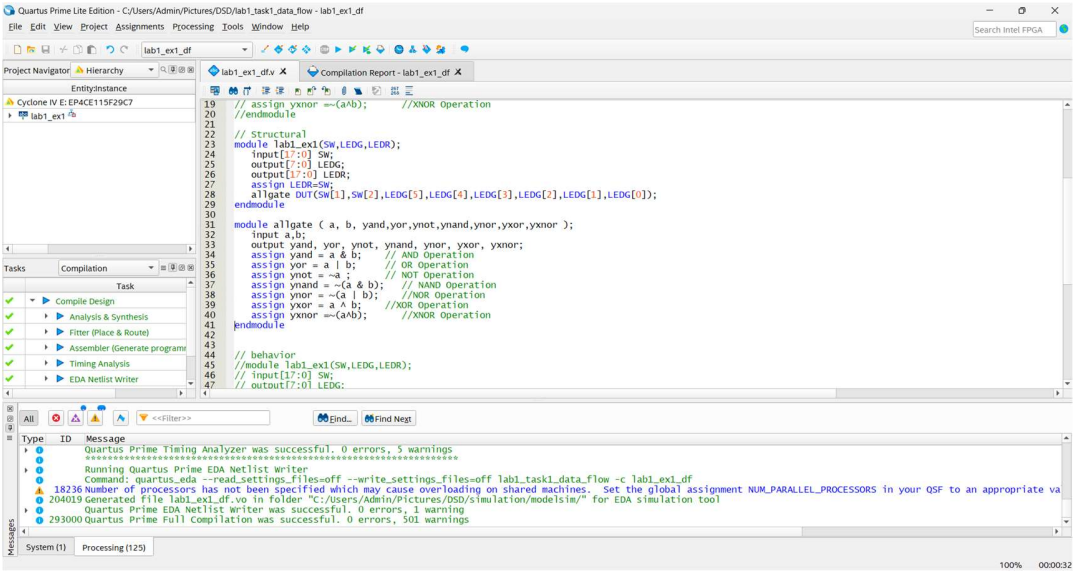
II.1 LAB EXPERIMENT 1 : WRITE HDL CODE TO REALIZE ALL LOGIC GATES

```
// 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(LE
DG[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
```



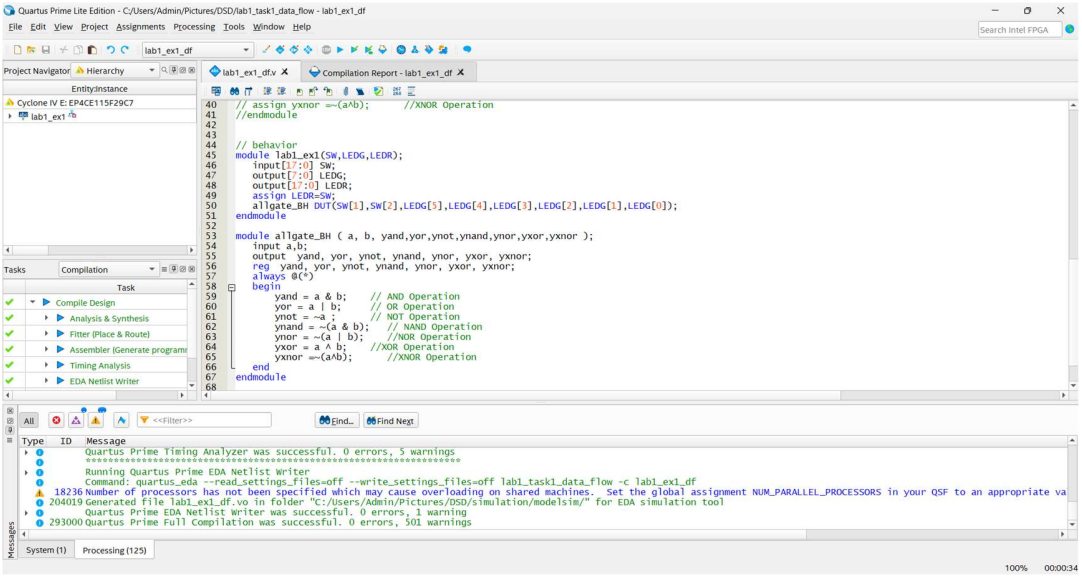
// **Structural**

```
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
```



// **behavior**

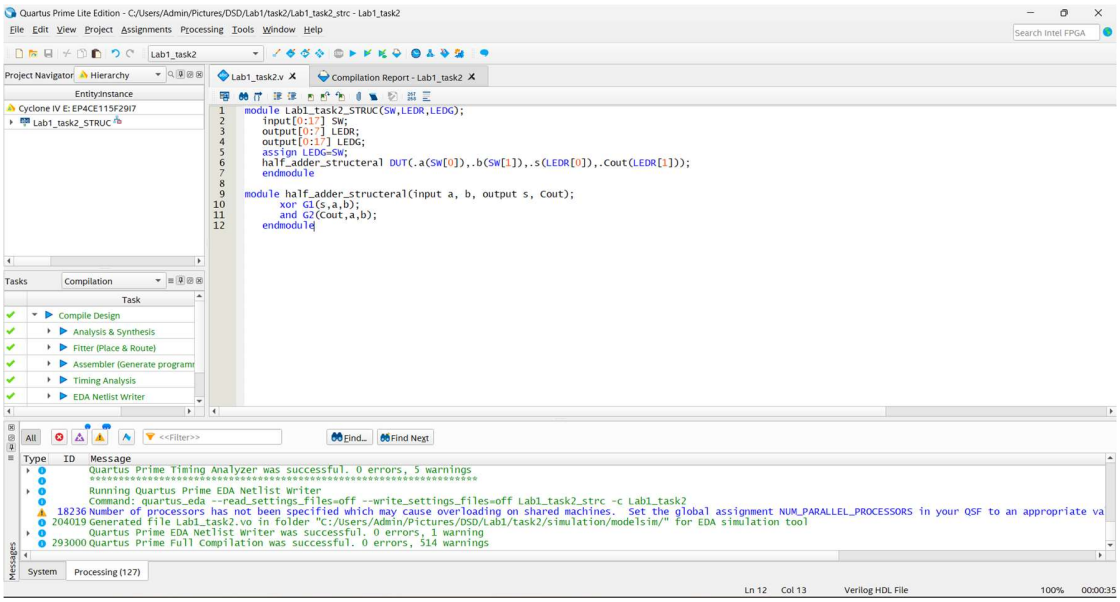
```
module lab1_ex1(SW,LEDG,LEDR);
        input[17:0] SW;
        output[7:0] LEDG;
        output[17:0] LEDR;
        assign LEDR=SW;
        allgate_BH DUT(SW[1],SW[2],LEDG[5],LEDG[4],LEDG[3],LEDG[2],LEDG[1],LEDG[0]);
endmodule

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
```



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

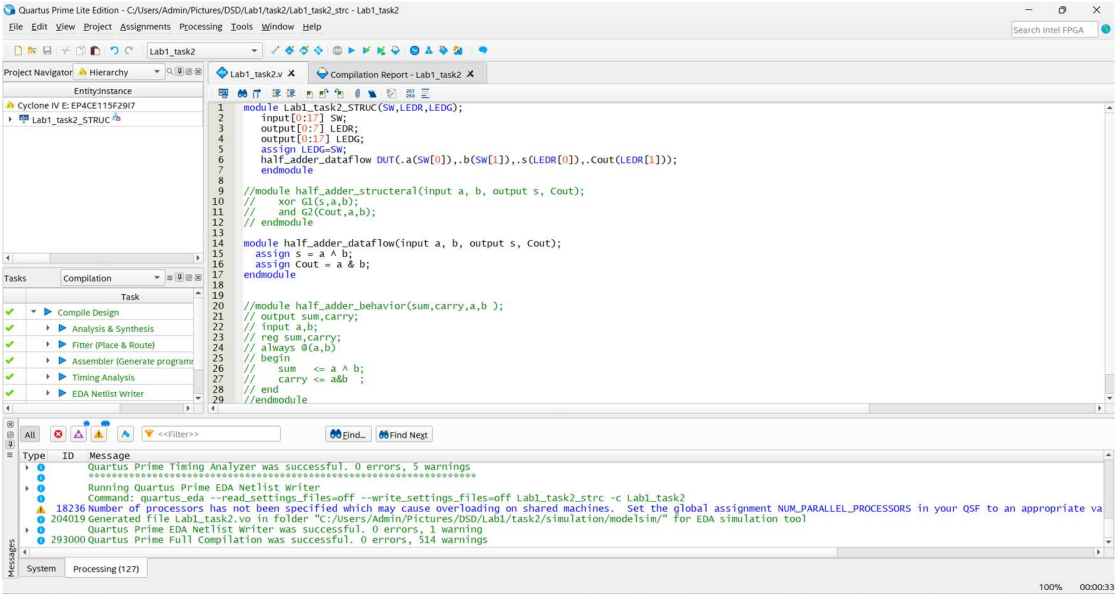
// **Structural**

```
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
```



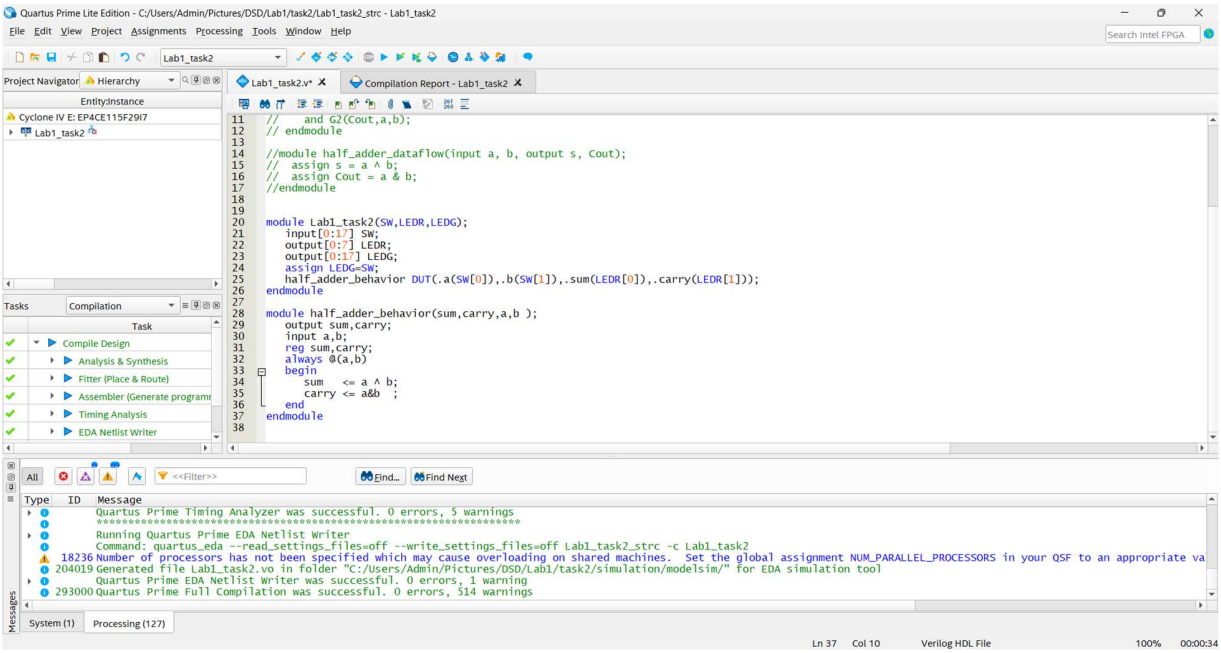
// **data flow**

```
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
```



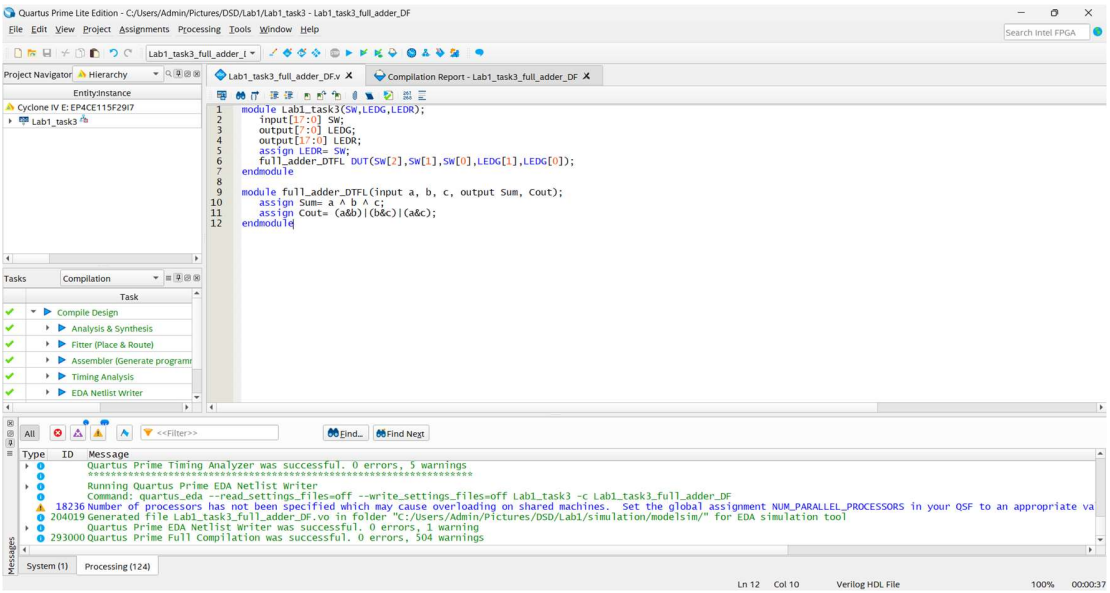
// **Behavior**

```
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
endmodule
```



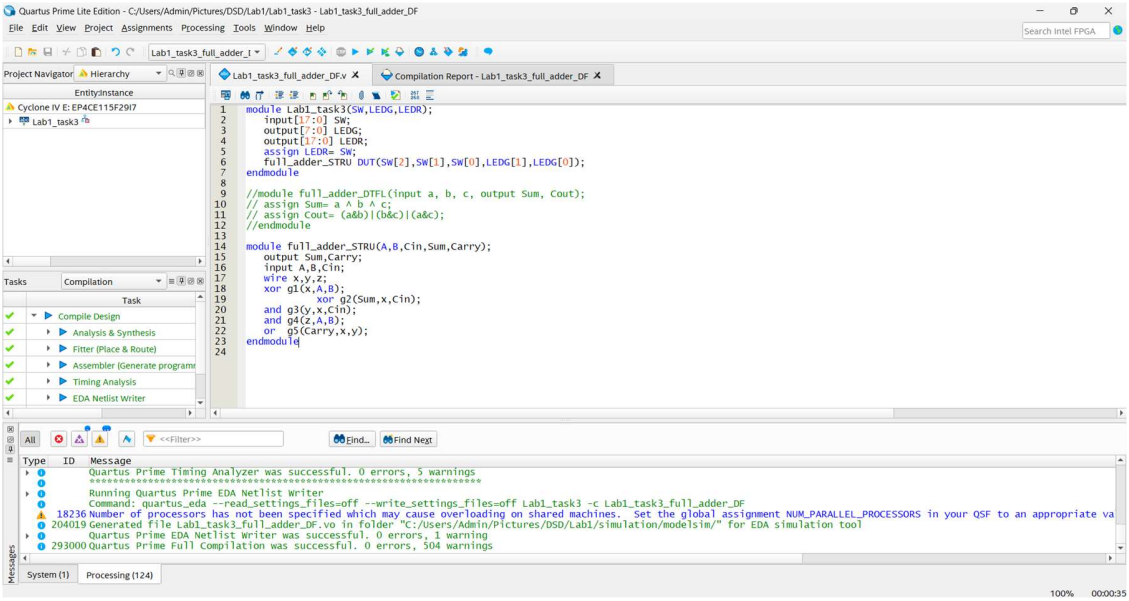
II.3 EXPERIMENT 3: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE FULL ADDER CIRCUIT:

```
// 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
```



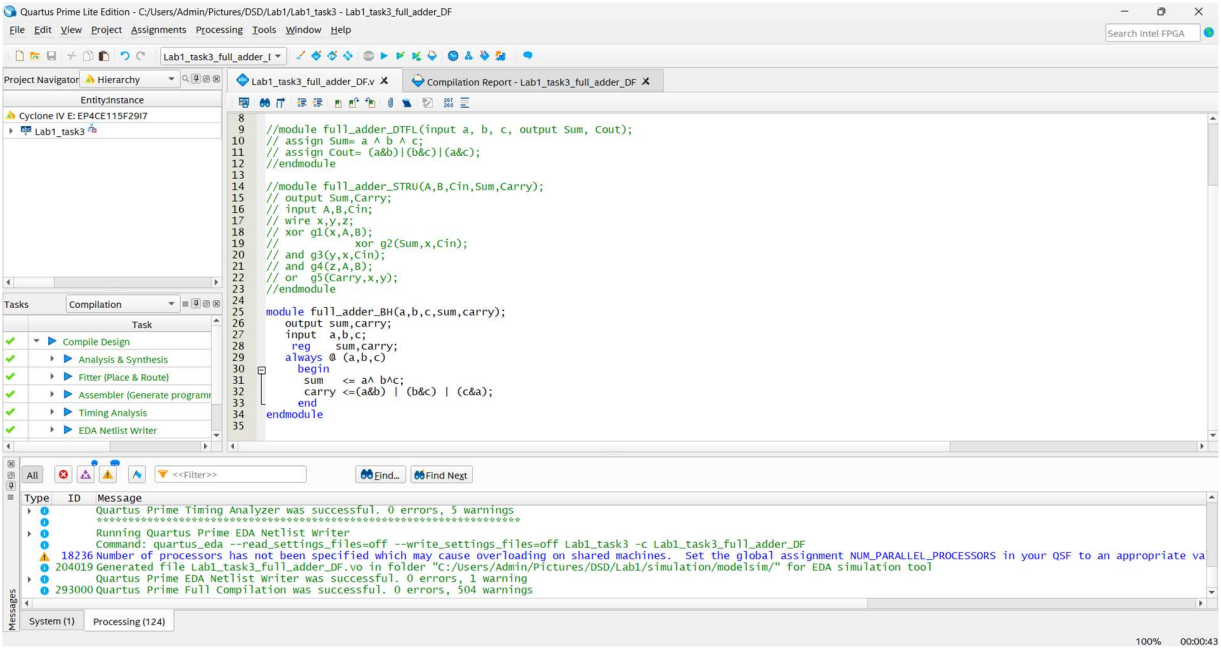
// **Structural**

```
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
```



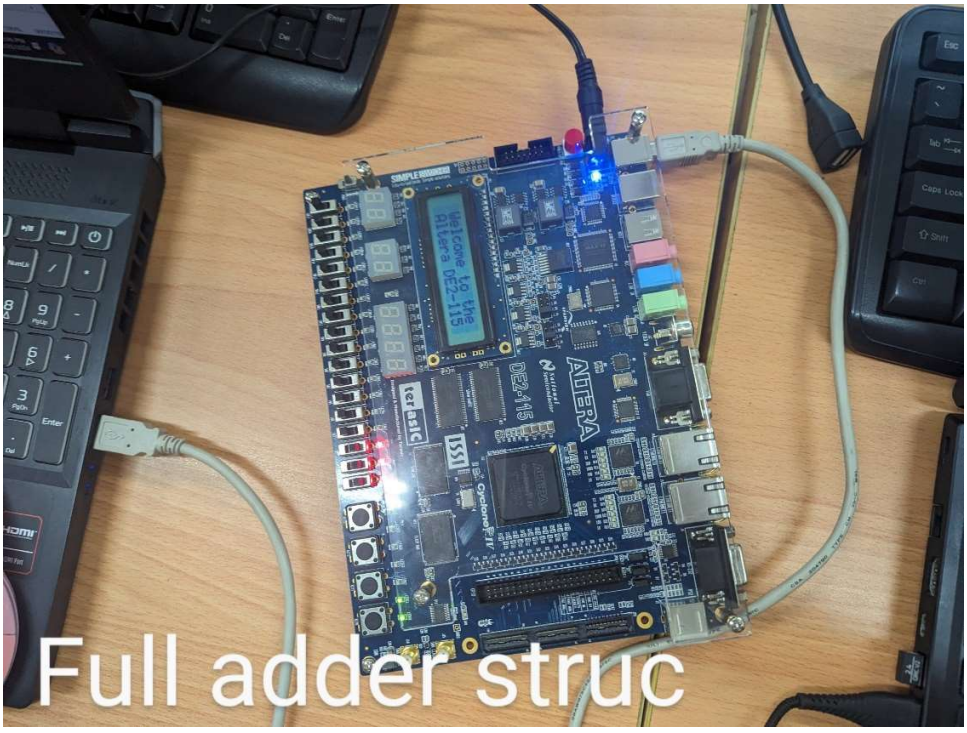
// **Behavior**

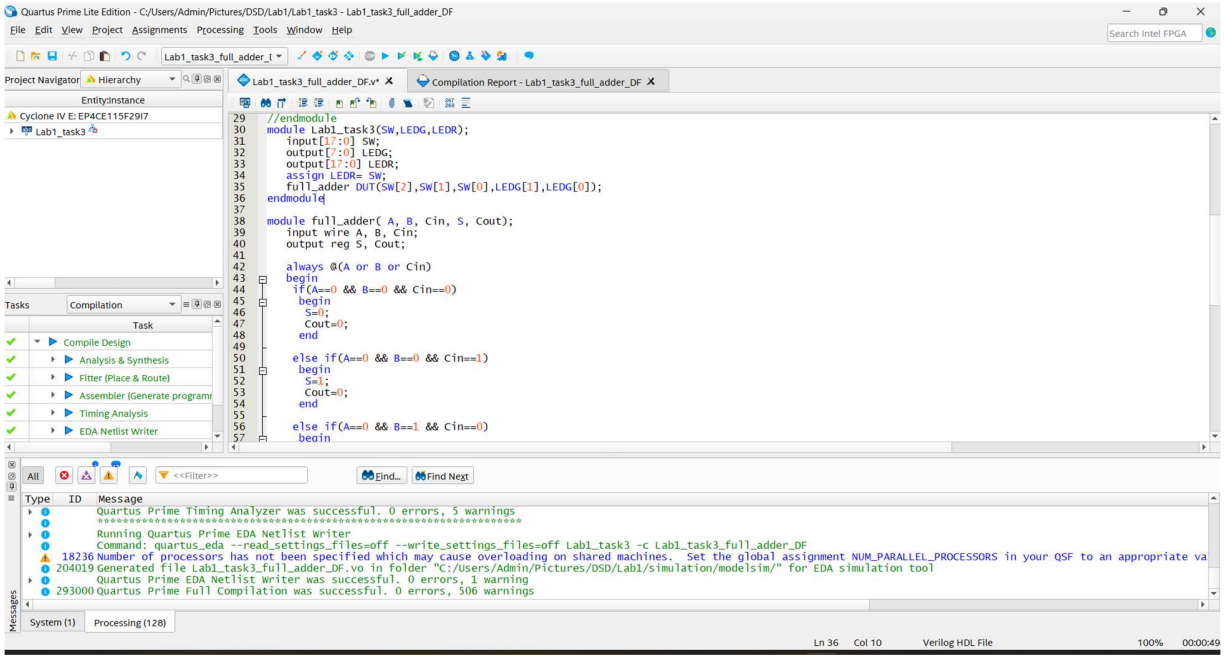
```
module Lab1_task3(SW,LEDG,LEDR);
        input[17:0] SW;
        output[7:0] LEDG;
        output[17:0] LEDR;
        assign LEDR= SW;
        full_adder_BH DUT(SW[2],SW[1],SW[0],LEDG[1],LEDG[0]);
endmodule

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
```




```
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
```

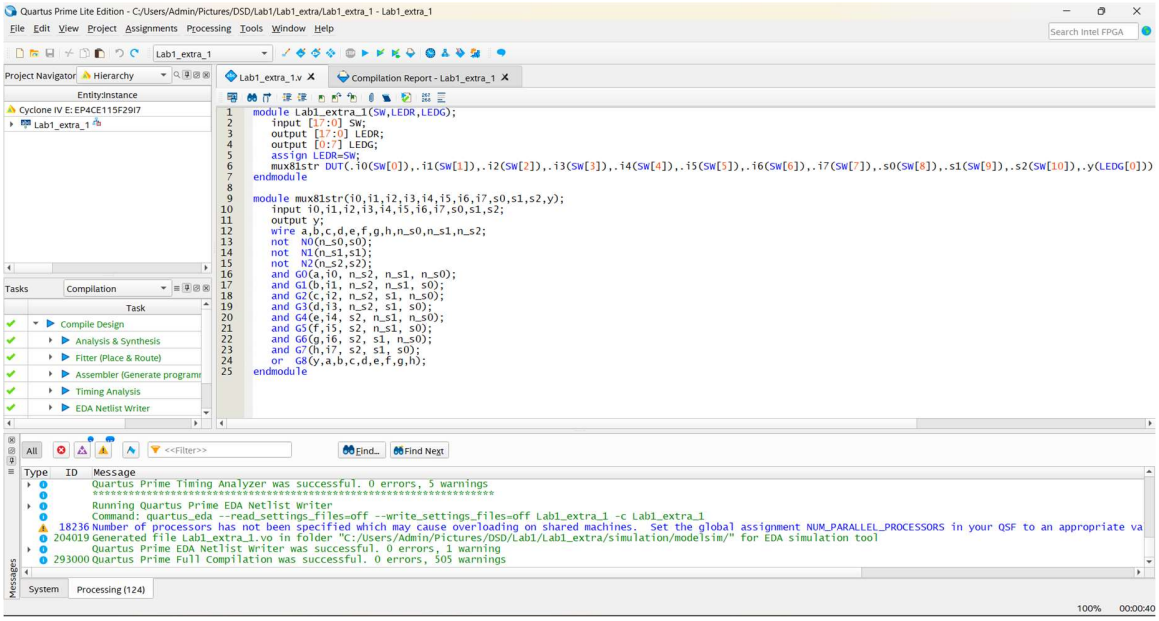





```
//endmodule
module Lab1_task3(SW,LEDG,LEDR);
   input[17:0] SW;
   output[7:0] LEDG;
   output[17:0] LEDR;
   assign LEDR= SW;
   full_adder DUT(SW[2],SW[1],SW[0],LEDG[1],LEDG[0]);
endmodule

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
```

EXTRA

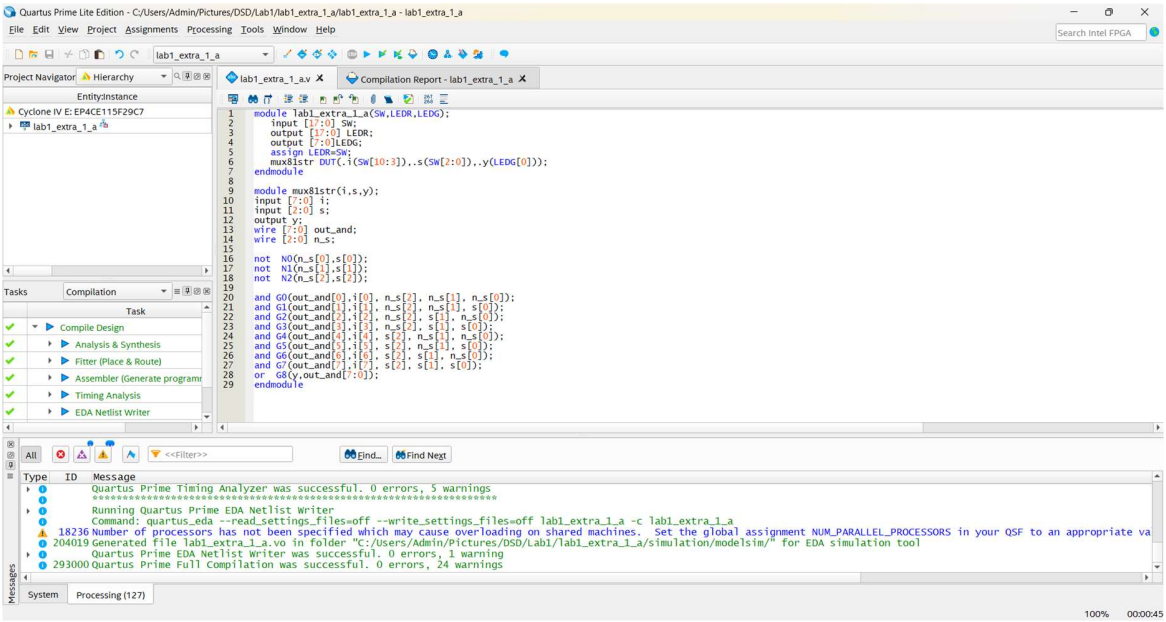
```
module Lab1_extra_1(SW,LEDR,LEDG);
        input [17:0] SW;
        output [17:0] LEDR;
        output [0:7] LEDG;
        assign LEDR=SW;
        mux81str
DUT(.i0(SW[0]),.i1(SW[1]),.i2(SW[2]),.i3(SW[3]),.i4(SW[4]),.i5(SW[5]),.i6(SW[6]),.i7(SW[7]),.s0(SW
[8]),.s1(SW[9]),.s2(SW[10]),.y(LEDG[0]));
endmodule

module mux81str(i0,i1,i2,i3,i4,i5,i6,i7,s0,s1,s2,y);
        input i0,i1,i2,i3,i4,i5,i6,i7,s0,s1,s2;
        output y;
        wire a,b,c,d,e,f,g,h,n_s0,n_s1,n_s2;
        not  N0(n_s0,s0);
        not  N1(n_s1,s1);
        not  N2(n_s2,s2);
        and G0(a,i0, n_s2, n_s1, n_s0);
        and G1(b,i1, n_s2, n_s1, s0);
        and G2(c,i2, n_s2, s1, n_s0);
        and G3(d,i3, n_s2, s1, s0);
        and G4(e,i4, s2, n_s1, n_s0);
        and G5(f,i5, s2, n_s1, s0);
        and G6(g,i6, s2, s1, n_s0);
        and G7(h,i7, s2, s1, s0);
        or  G8(y,a,b,c,d,e,f,g,h);
endmodule
```



Extra 1a

```
module lab1_extra_ex1a(SW,LEDR,LEDG);
        input [17:0] SW;
        output [17:0] LEDR;
        output LEDG;
        assign LEDR=SW;
        modulemux81str DUT(.i(S[10:3]),.s(SW[2:0]).y(LEDG[0]));
endmodule

modulemux81str(i,s,y);
input [7:0] i;
input [2:0] s;
output y;
wire [7:0] out_and;
wire [2:0] n_s;
;
not  N0(n_s[0],s[0]);
not  N1(n_s[1],s[1]);
not  N2(n_s[2],s[2]);

and G0(out_and[0],i[0], n_s[2], n_s[1], n_s[0]);
and G1(out_and[1],i[1], n_s[2], n_s[1], s[0]);
and G2(out_and[2],i[2], n_s[2], s[1], n_s[0]);
and G3(out_and[3],i[3], n_s[2], s[1], s[0]);
and G4(out_and[4],i[4], s[2], n_s[1], n_s[0]);
and G5(out_and[5],i[5], s[2], n_s[1], s[0]);
and G6(out_and[6],i[6], s[2], s[1], n_s[0]);
and G7(out_and[7],i[7], s[2], s[1], s[0]);
or  G8(y,out_and[7:0]);
```

endmodule


Quartus Prime Lite Edition - C:/Users/Admin/Pictures/DSD/Lab1/lab1_extra_1_a/lab1_extra_1_a - lab1_extra_1_a
File Edit View Project Assignments Processing Tools Window Help
Search Intel FPGA
Project Navigator Hierarchy
Entity:Instance
Cyclone IV E: EP4CE115F29C7
lab1_extra_1_a
lab1_extra_1_a.v
Compilation Report - lab1_extra_1_a
module lab1_extra_1_a(SW,LEDR,LEDG);
input [17:0] SW;
output [17:0] LEDR;
output [7:0]LEDG;
assign LEDR=SW;
mux81str DUT(.i(SW[10:3]),.s(SW[2:0]),.y(LEDG[0]));
endmodule
module mux81str(i,s,y);
input [7:0] i;
input [2:0] s;
output y;
wire [7:0] out_and;
wire [2:0] n_s;
not N0(n_s[0],s[0]);
not N1(n_s[1],s[1]);
not N2(n_s[2],s[2]);
and G0(out_and[0],i[0], n_s[2], n_s[1], n_s[0]);
and G1(out_and[1],i[1], n_s[2], n_s[1], s[0]);
and G2(out_and[2],i[2], n_s[2], s[1], n_s[0]);
and G3(out_and[3],i[3], n_s[2], s[1], s[0]);
and G4(out_and[4],i[4], s[2], n_s[1], n_s[0]);
and G5(out_and[5],i[5], s[2], n_s[1], s[0]);
and G6(out_and[6],i[6], s[2], s[1], n_s[0]);
and G7(out_and[7],i[7], s[2], s[1], s[0]);
or G8(y,out_and[7:0]);
endmodule
Tasks Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programm
Timing Analysis
EDA Netlist Writer
Find... Find Next
Type ID Message
Quartus Prime Timing Analyzer was successful. 0 errors, 5 warnings
Running Quartus Prime EDA Netlist Writer
Command: quartus_eda --read_settings_files=off --write_settings_files=off lab1_extra_1_a -c lab1_extra_1_a
18236 Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate va
204019 Generated file lab1_extra_1_a.vo in folder "C:/Users/Admin/Pictures/DSD/Lab1/lab1_extra_1_a/simulation/modelsim/" for EDA simulation tool
Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
293000 Quartus Prime Full Compilation was successful. 0 errors, 24 warnings
Messages
System
Processing (127)
100%
00:00:45