**ASSIGNMENT 1**

1. **CODE:**

**module add(a,b,gt,lt,eq);**

**input [1:0] a,b;**

**output gt,lt,eq;**

**wire a1b,b1b,a0b,b0b,x,y,z,c,d,e,f,g,h,i;**

**not(a1b,a[1]);**

**not(a0b,a[0]);**

**not(b1b,b[1]);**

**not(b0b,b[0]);**

**assign a1=a[1];**

**assign b1=b[1];**

**assign b0=b[0];**

**assign a0=a[0];**

**and(x,a1,a0,b0b);**

**and(y,a0,b1b,b0b);**

**and(z,a1,b1b);**

**or(gt,x,y,z);**

**and(c,a1b,a0b,b1b,b0b);**

**and(d,a1b,a0,b1b,b0);**

**and(e,a1,a0,b1,b0);**

**and(f,a1,a0b,b1,b0b);**

**or(eq,c,d,e,f);**

**and(g,a1b,b1);**

**and(h,a0b,b1,b0);**

**and(i,a1b,a0b,b0);**

**or(lt,g,h,i);**

**endmodule**

**TEST BENCH**

**`timescale 1ns/1ps**

**module addtb;**

**reg [1:0] a, b;**

**wire gt, lt, eq;**

**bitcomp uut (**

**.a(a),**

**.b(b),**

**.gt(gt),**

**.lt(lt),**

**.eq(eq)**

**);**

**initial begin**

**a = 2'b00; b = 2'b11;**

**#5;**

**a = 2'b11; b = 2'b11;**

**#5;**

**a = 2'b11; b = 2'b10;**

**#5;**

**$finish;**

**end**

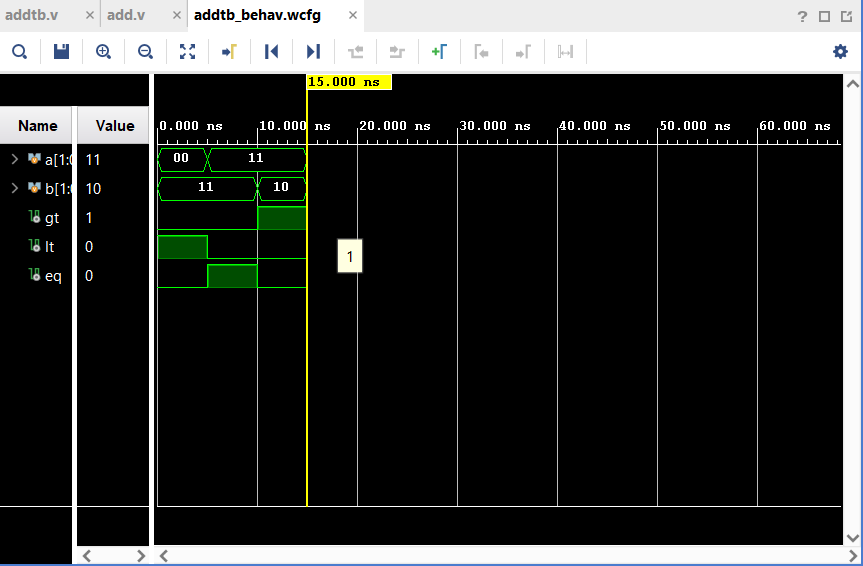
**initial begin**

**$monitor($time, " a=%b, b=%b, gt=%b, lt=%b, eq=%b", a, b, gt, lt, eq);**

**end**

**endmodule**

**WAVEFORM**

****

**2.ALU**

**CODE:**

**module fulladder(a,b,c,sum,carry);**

**input a,b,c;**

**output sum,carry;**

**wire x,y,z,d;**

**xor(d,a,b);**

**xor(sum,d,c);**

**and(x,a,b);**

**and(y,b,c);**

**and(z,c,a);**

**or(carry,x,y,z);**

**endmodule**

**module addsub(a,b,as,sum,carry);**

**input[3:0] a,b;**

**input as;**

**output[3:0] sum;**

**output carry;**

**wire[3:0] bx ,c;**

**xor(bx[0],b[0],as);**

**xor(bx[1],b[1],as);**

**xor(bx[2],b[2],as);**

**xor(bx[3],b[3],as);**

**fulladder f1(a[0],bx[0],as,sum[0],c[0]);**

**fulladder f2(a[1],bx[1],c[0],sum[1],c[1]);**

**fulladder f3(a[2],bx[2],c[1],sum[2],c[2]);**

**fulladder f4(a[3],bx[3],c[2],sum[3],carry);**

**endmodule**

**module alu(**

**input [1:0] s,**

**input [3:0] a, b,**

**output reg [3:0] out**

**);**

**wire [3:0] sum;**

**wire [3:0] org\_out, andg\_out;**

**wire carry;**

**reg as;**

**always @(\*) begin**

**case (s)**

**2'b00: as = 0;**

**2'b01: as = 1;**

**endcase**

**end**

**addsub A(a, b, as, sum, carry);**

**org O(a, b, org\_out, andg\_out);**

**always @(\*) begin**

**case (s)**

**2'b00: out = sum;**

**2'b01: out = sum;**

**2'b10: out = org\_out;**

**2'b11: out = andg\_out;**

**endcase**

**end**

**endmodule**

**`timescale 1ns/1ps**

**module alu\_tb;**

**reg [3:0] a, b;**

**reg [1:0] s;**

**wire [3:0] out;**

**alu uut (s,a,b,out);**

**initial begin**

**a = 7;**

**b = 4;**

**s = 0; #10;**

**s = 1; #10;**

**s = 2; #10;**

**s = 3; #10;**

**$finish;**

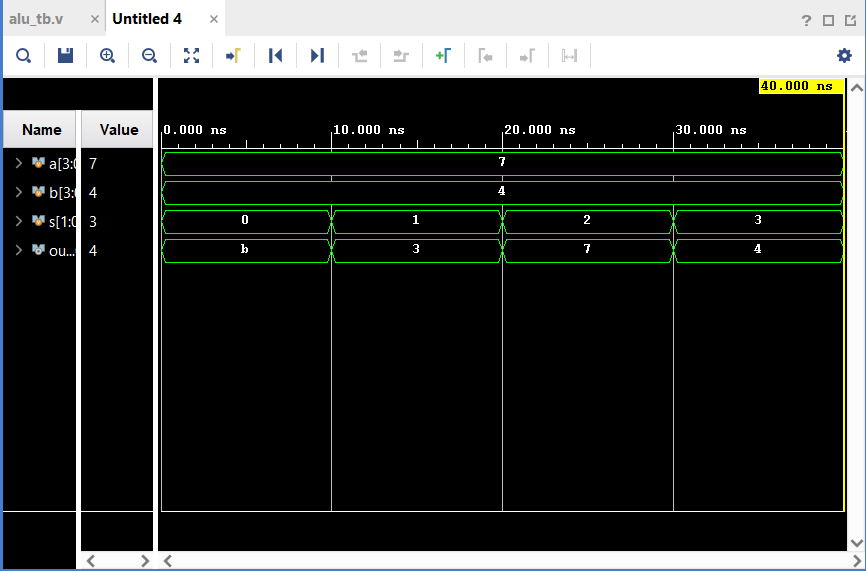
**end**

**initial begin**

**$monitor(" s=%b a=%b b=%b out=%b", $time, s, a, b, out);**

**end**

**endmodule**

****

**3.code**

**module mux4(i0,i1,i2,i3,s0,s1,y);**

**input i0,i1,i2,i3,s0,s1;**

**output y;**

**wire x,z,a,c;**

**not(s1b,s1);**

**not(s0b,s0);**

**and(x,s0b,s1b,i0);**

**and(z,s0,s1b,i1);**

**and(a,s0b,s1,i2);**

**and(c,s0,s1,i3);**

**or(y,x,a,c,z);**

**endmodule**

**module mux8(i0,i1,i2,i3,i4,i5,i6,i7,s0,s1,s2,y);**

**input i0,i1,i2,i3,i4,i5,i6,i7,s0,s1,s2;**

**wire y1,y2,s2bar,x,z;**

**output y;**

**mux4 m1(i0,i1,i2,i3,s0,s1,y1);**

**mux4 m2(i4,i5,i6,i7,s0,s1,y2);**

**not(s2bar,s2);**

**and(x,s2bar,y1);**

**and(z,s2,y2);**

**or(y,x,z);**

**endmodule**

**module muxtb;**

**reg i0,i1,i2,i3,i4,i5,i6,i7;**

**reg s0,s1,s2;**

**wire out;**

**mux8 uut (i0,i1,i2,i3,i4,i5,i6,i7,s0,s1,s2,out);**

**initial begin**

**i0=0;i1=1;i2=0;i3=0;i4=0;i5=1;i6=0;i7=1;**

**s0 = 0; s1=0; s2=0; #10;**

**s0 = 0; s1=0; s2=1; #10;**

**s0 = 0; s1=1; s2=0;#10;**

**s0 = 1; s1=0; s2=0; #10;**

**$finish;**

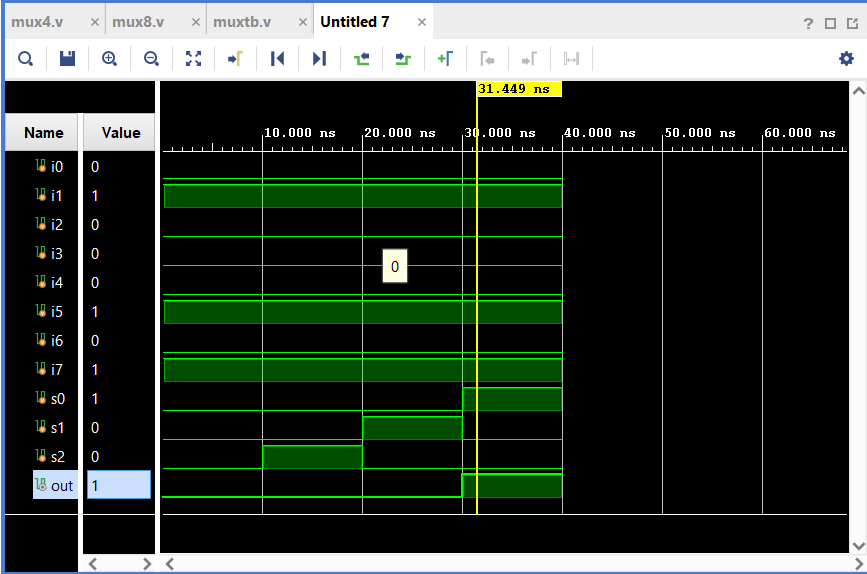
**end**

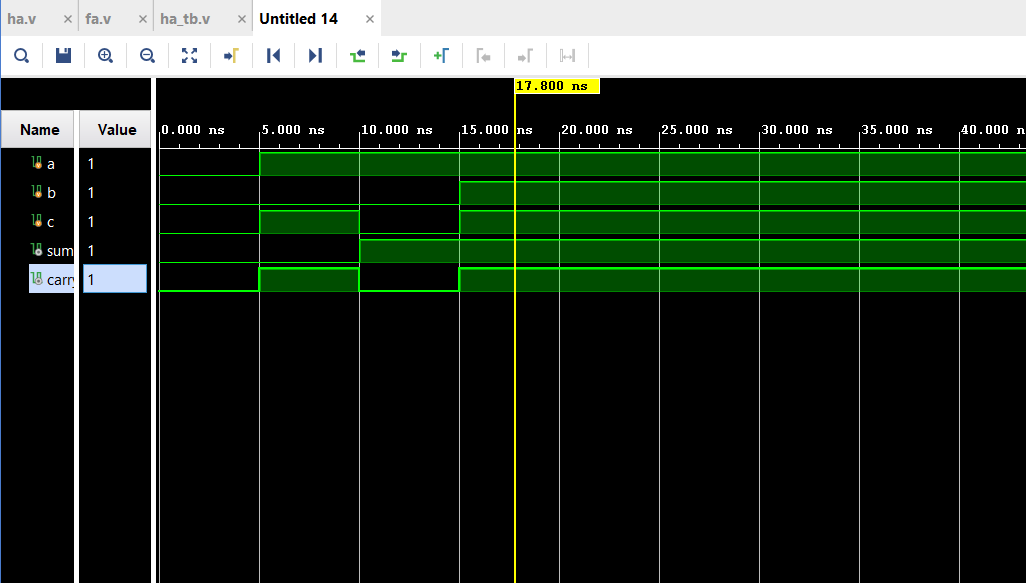
**initial begin**

**$monitor(" s0=%b s1=%b s2=%b i0=%b i1=%b i2=%b i3=%b i4=%b i5=%b i6=%b i7=%b y=%b", $time, i0,i1,i2,i3,i4,i5,i6,i7,s0,s1,s2,out);**

**end**

**endmodule**

****

**4.** ****

**module ha(**

**input a,**

**input b,**

**output sum,**

**output carry**

**);**

**xor(sum,a,b);**

**and(carry,a,b);**

**endmodule**

**module fa(**

**input a,**

**input b,**

**input c,**

**output sum,**

**output carry**

**);**

**wire sum1,carry1,carry2;**

**ha h0(a,b,sum1,carry1);**

**ha h1(sum1,c,sum,carry2);**

**or(carry,carry1,carry2);**

**endmodule**

**module ha\_tb;**

**reg a,b,c;**

**wire sum,carry;**

**fa uut(a,b,c,sum,carry);**

**initial begin**

**a=0;b=0;c=0;**

**#5 a=1;b=0;c=1;**

**#5 a=1;b=0;c=0;**

**#5 a=1;b=1;c=1;**

**end**

**initial begin**

**$monitor($time,"a=%b b=%b c=%b sum=%b carry=%c",a,b,c,sum,carry);**

**end**

**endmodule**

**5. no wires are required**

**6.**  **Instantiating the half adder code twice and using an "or" primitive**

**7.** **0000**

**8.** **Instantiating 4-bit ripple carry adder twice, eight "xor" one "not" and one "and"**

**Primitives**

**9.** **Instantiating two half adders and four "and" primitives**

**10.2**

**11.** **One "xnor", two "not", two "and" primitives**

**12.** **Two 1-bit comparators, three "and" gates, two "or" gates**

**13.** **module fulladder(a,b,c,sum,carry);**

**input a,b,c;**

**output sum,carry;**

**wire x,y,z,d;**

**xor(d,a,b);**

**xor(sum,d,c);**

**and(x,a,b);**

**and(y,b,c);**

**and(z,c,a);**

**or(carry,x,y,z);**

**endmodule**

**module add(a,b,cin,sum,carry);**

**input[3:0] a,b;**

**output[3:0] sum;**

**input cin;**

**wire[3:0] c;**

**output carry;**

**fulladder f1(a[0],b[0],cin,sum[0],c[0]);**

**fulladder f2(a[1],b[1],c[0],sum[1],c[1]);**

**fulladder f3(a[2],b[2],c[1],sum[2],c[2]);**

**fulladder f4(a[3],b[3],c[2],sum[3],carry);**

**endmodule**

**module addtb;**

**reg[3:0] a,b;**

**reg cin;**

**wire[3:0] sum;**

**wire carry;**

**add uut(a,b,cin,sum,carry);**

**initial begin**

**a=4;b=5;cin=0;**

**#5 a=8;b=9;**

**#5 a=7;b=9;**

**#5 a=6;b=7;**

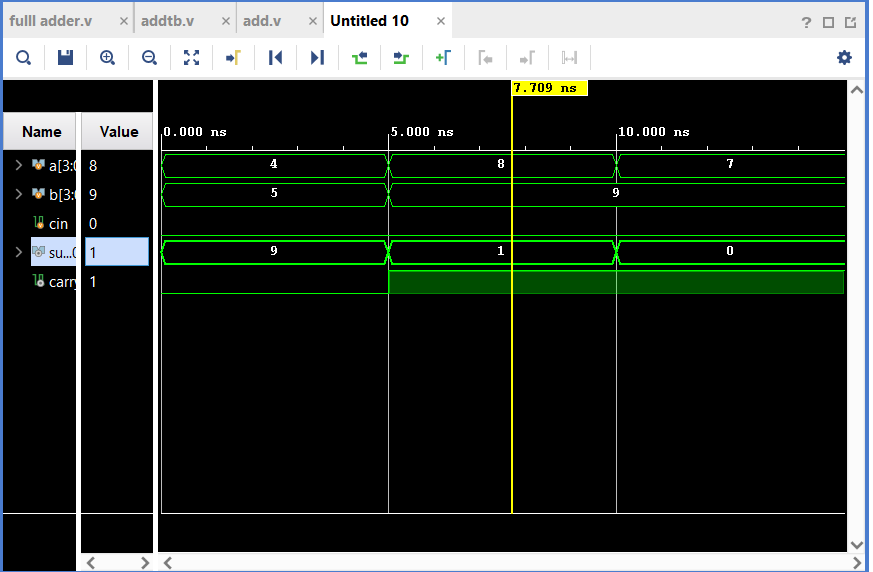
**$finish;**

**end**

**initial begin**

**$monitor($time," a=%b b=%b sum=%b carry=%b",a,b,sum,carry);**

**End**

****

**14. for verilog we should give the design specification in synthezing the circuit, it should not be changed during run time, so it is called static**

**15.8x8=64bits 1 character=8bits**

**16. setup time: time required to input signal maintained to be in stable state before clock**

**hold time: time required to input signal maintained to be in stable state after clock signal**

**17.** Verilog describes hardware circuits( logic gates, flip-flops). Programming languages describe sequential software logic running on a CPU.

18. verilog is C-like hardware description language widely used for designing and simulating digital circuits.  
VHDL is a strong language commonly used in safety-critical and defense applications.

19.w value is:1

20.

module nan(a,b,out);

input a,b;

output out;

supply1 vdd;

supply0 gnd;

wire w;

pmos(out,vdd,a);

pmos(out,vdd,b);

nmos(out,w,a);

nmos(w,gnd,b);

endmodule

module nan(a,b,out);

input a,b;

output out;

nor(out,a,b);

endmodule

21. module nand\_gate\_cmos (

input A, B,

output Y

);

supply1 Vdd;

supply0 Gnd;

wire w;

pmos (Y, Vdd, A);

pmos (Y, w, B);

nmos (Y, Gnd, A);

nmos (Y, Gnd, B);

endmodule

module nan(a,b,out);

input a,b;

output out;

nand(out,a,b);

endmodule