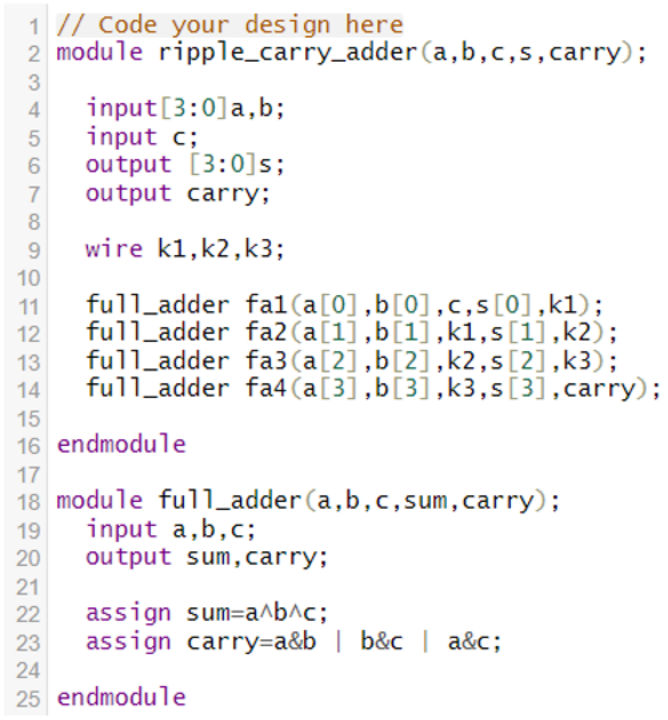
Assignment 2

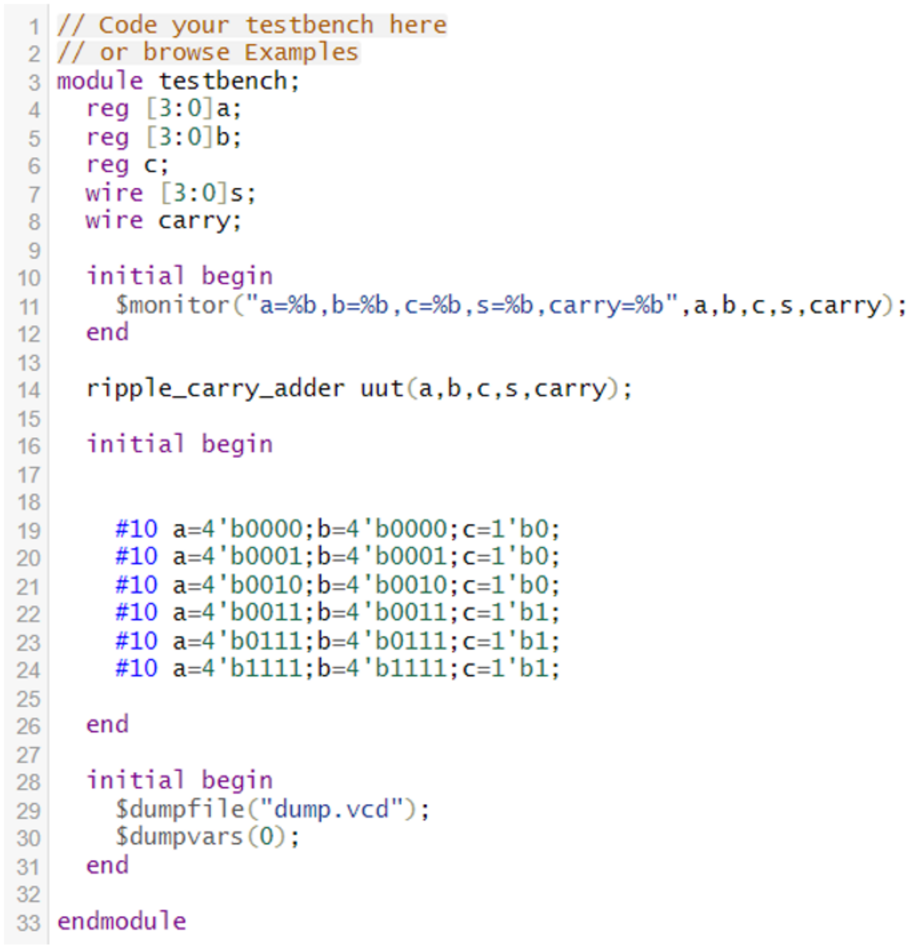
1] Design 4-bit Ripple Carry Adder with the help of 1-bit adder

CODE:



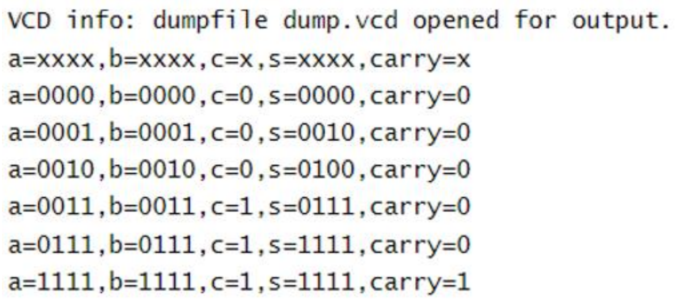
```verilog
// Code your design here
module ripple_carry_adder(a,b,c,s,carry);

  input[3:0]a,b;
  input c;
  output [3:0]s;
  output carry;

  wire k1,k2,k3;

  full_adder fa1(a[0],b[0],c,s[0],k1);
  full_adder fa2(a[1],b[1],k1,s[1],k2);
  full_adder fa3(a[2],b[2],k2,s[2],k3);
  full_adder fa4(a[3],b[3],k3,s[3],carry);

endmodule

module full_adder(a,b,c,sum,carry);
  input a,b,c;
  output sum,carry;

  assign sum=a^b^c;
  assign carry=a&b | b&c | a&c;

endmodule
```

TEST BENCH:



```
// Code your testbench here
// or browse Examples
module testbench;
  reg [3:0]a;
  reg [3:0]b;
  reg c;
  wire [3:0]s;
  wire carry;

  initial begin
    $monitor("a=%b,b=%b,c=%b,s=%b,carry=%b",a,b,c,s,carry);
  end

  ripple_carry_adder uut(a,b,c,s,carry);

  initial begin


    #10 a=4'b0000;b=4'b0000;c=1'b0;
    #10 a=4'b0001;b=4'b0001;c=1'b0;
    #10 a=4'b0010;b=4'b0010;c=1'b0;
    #10 a=4'b0011;b=4'b0011;c=1'b1;
    #10 a=4'b0111;b=4'b0111;c=1'b1;
    #10 a=4'b1111;b=4'b1111;c=1'b1;

  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end

endmodule
```

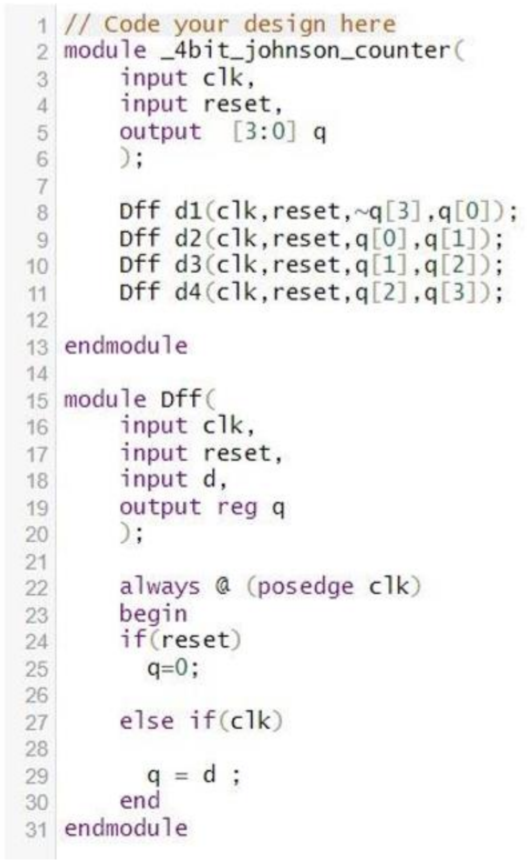
OUTPUT:



```
VCD info: dumpfile dump.vcd opened for output.
a=xxxx,b=xxxx,c=x,s=xxxx,carry=x
a=0000,b=0000,c=0,s=0000,carry=0
a=0001,b=0001,c=0,s=0010,carry=0
a=0010,b=0010,c=0,s=0100,carry=0
a=0011,b=0011,c=1,s=0111,carry=0
a=0111,b=0111,c=1,s=1111,carry=0
a=1111,b=1111,c=1,s=1111,carry=1
```

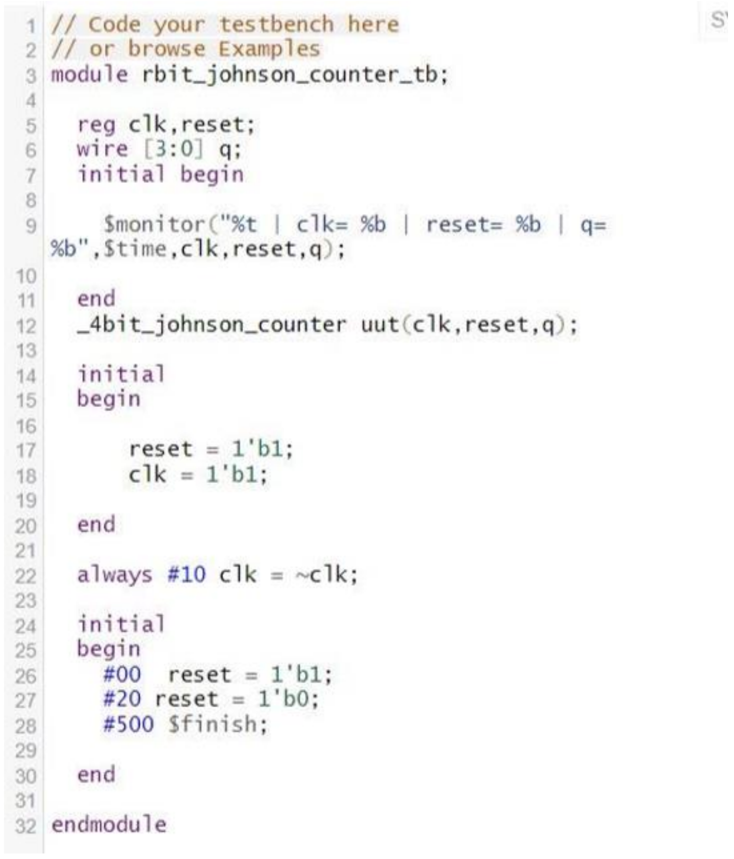
2] Design D-flipflop and reuse it to implement 4- bit Johnson Counter.]

CODE:



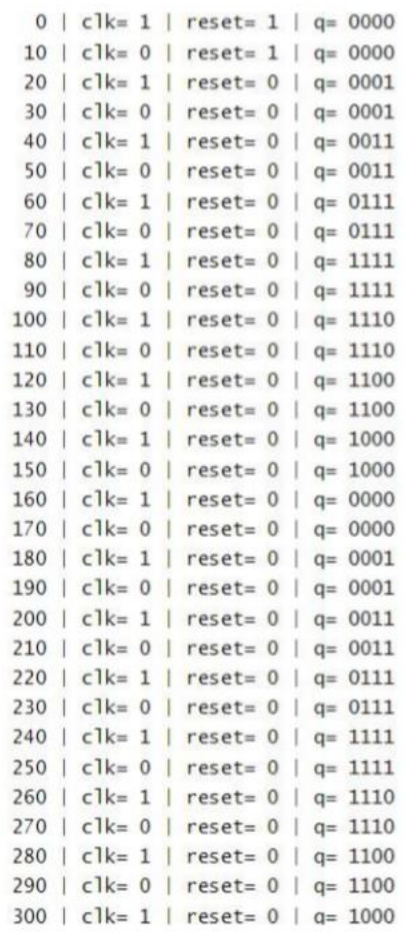
```
// Code your design here
module _4bit_johnson_counter(
    input clk,
    input reset,
    output  [3:0] q
    );

    Dff d1(clk,reset,~q[3],q[0]);
    Dff d2(clk,reset,q[0],q[1]);
    Dff d3(clk,reset,q[1],q[2]);
    Dff d4(clk,reset,q[2],q[3]);

endmodule

module Dff(
    input clk,
    input reset,
    input d,
    output reg q
    );

    always @ (posedge clk)
    begin
    if(reset)
      q=0;

    else if(clk)

      q = d ;
    end
endmodule
```

TEST BENCH:



```
// Code your testbench here
// or browse Examples
module rbit_johnson_counter_tb;

  reg clk,reset;
  wire [3:0] q;
  initial begin

    $monitor("%t | clk= %b | reset= %b | q=
%b",$time,clk,reset,q);

  end
  _4bit_johnson_counter uut(clk,reset,q);

  initial
  begin

      reset = 1'b1;
      clk = 1'b1;

  end

  always #10 clk = ~clk;

  initial
  begin
    #00  reset = 1'b1;
    #20 reset = 1'b0;
    #500 $finish;

  end

endmodule
```

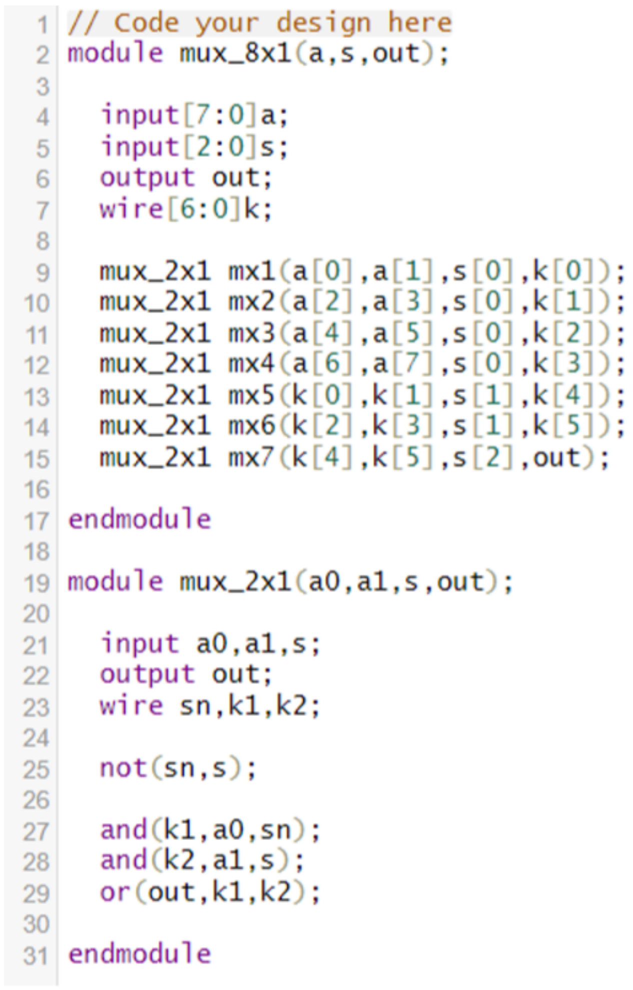
OUTPUT:



```
  0 | clk= 1 | reset= 1 | q= 0000
 10 | clk= 0 | reset= 1 | q= 0000
 20 | clk= 1 | reset= 0 | q= 0001
 30 | clk= 0 | reset= 0 | q= 0001
 40 | clk= 1 | reset= 0 | q= 0011
 50 | clk= 0 | reset= 0 | q= 0011
 60 | clk= 1 | reset= 0 | q= 0111
 70 | clk= 0 | reset= 0 | q= 0111
 80 | clk= 1 | reset= 0 | q= 1111
 90 | clk= 0 | reset= 0 | q= 1111
100 | clk= 1 | reset= 0 | q= 1110
110 | clk= 0 | reset= 0 | q= 1110
120 | clk= 1 | reset= 0 | q= 1100
130 | clk= 0 | reset= 0 | q= 1100
140 | clk= 1 | reset= 0 | q= 1000
150 | clk= 0 | reset= 0 | q= 1000
160 | clk= 1 | reset= 0 | q= 0000
170 | clk= 0 | reset= 0 | q= 0000
180 | clk= 1 | reset= 0 | q= 0001
190 | clk= 0 | reset= 0 | q= 0001
200 | clk= 1 | reset= 0 | q= 0011
210 | clk= 0 | reset= 0 | q= 0011
220 | clk= 1 | reset= 0 | q= 0111
230 | clk= 0 | reset= 0 | q= 0111
240 | clk= 1 | reset= 0 | q= 1111
250 | clk= 0 | reset= 0 | q= 1111
260 | clk= 1 | reset= 0 | q= 1110
270 | clk= 0 | reset= 0 | q= 1110
280 | clk= 1 | reset= 0 | q= 1100
290 | clk= 0 | reset= 0 | q= 1100
300 | clk= 1 | reset= 0 | q= 1000
```

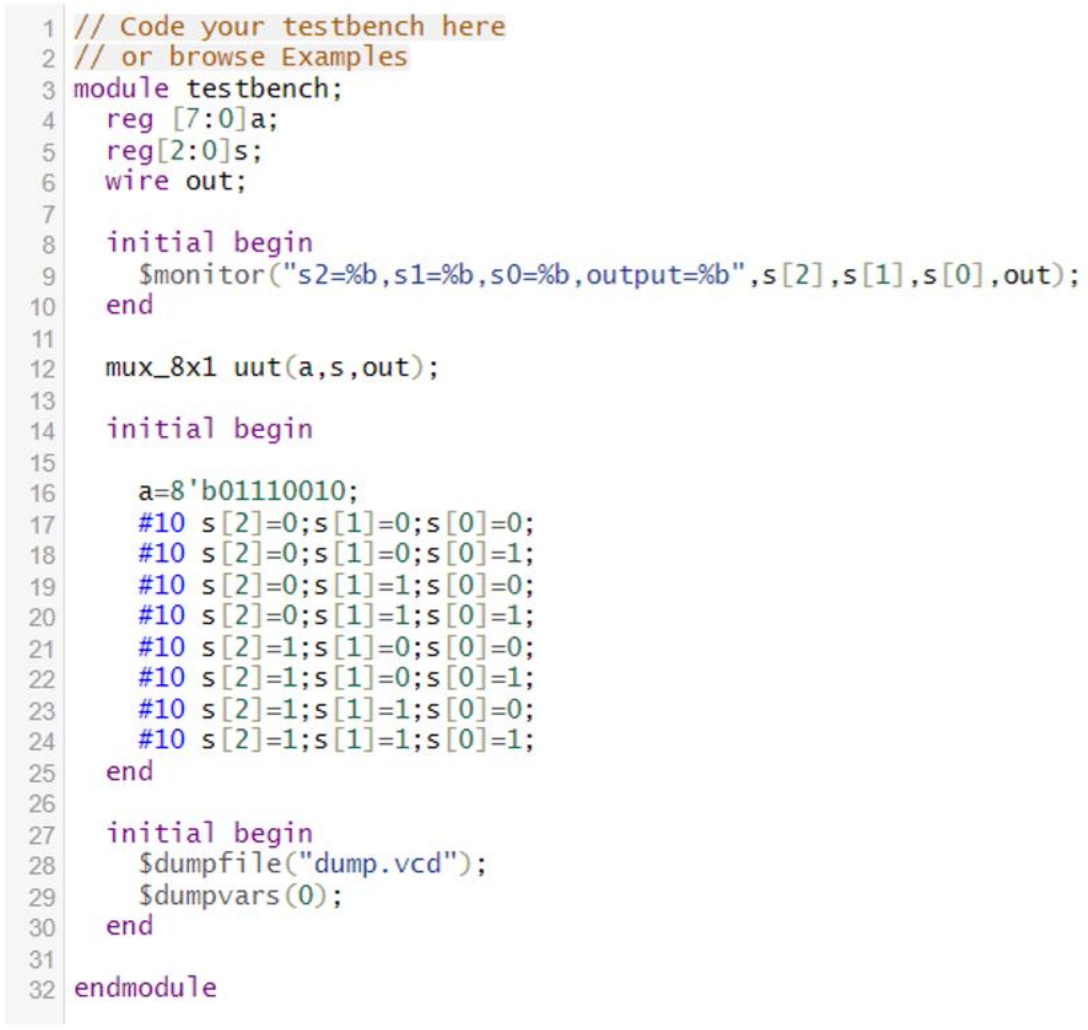
3] Reuse 2:1 Mux code to implement 8:1 Mux.

CODE:



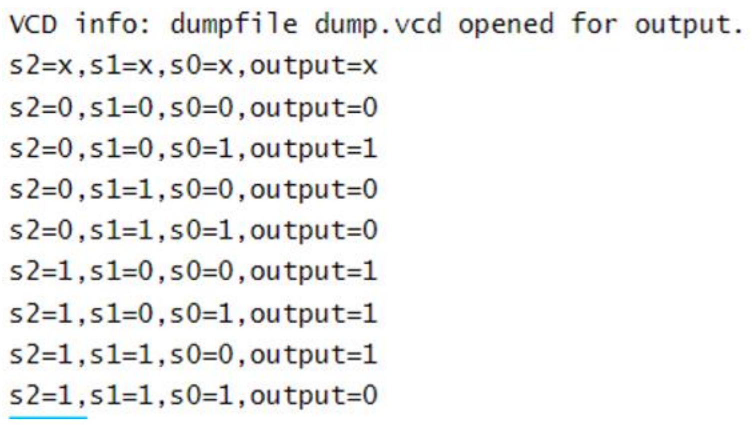
```
// Code your design here
module mux_8x1(a,s,out);

  input[7:0]a;
  input[2:0]s;
  output out;
  wire[6:0]k;

  mux_2x1 mx1(a[0],a[1],s[0],k[0]);
  mux_2x1 mx2(a[2],a[3],s[0],k[1]);
  mux_2x1 mx3(a[4],a[5],s[0],k[2]);
  mux_2x1 mx4(a[6],a[7],s[0],k[3]);
  mux_2x1 mx5(k[0],k[1],s[1],k[4]);
  mux_2x1 mx6(k[2],k[3],s[1],k[5]);
  mux_2x1 mx7(k[4],k[5],s[2],out);

endmodule

module mux_2x1(a0,a1,s,out);

  input a0,a1,s;
  output out;
  wire sn,k1,k2;

  not(sn,s);

  and(k1,a0,sn);
  and(k2,a1,s);
  or(out,k1,k2);

endmodule
```

TEST BENCH:



```
// Code your testbench here
// or browse Examples
module testbench;
  reg [7:0]a;
  reg[2:0]s;
  wire out;

  initial begin
    $monitor("s2=%b,s1=%b,s0=%b,output=%b",s[2],s[1],s[0],out);
  end

  mux_8x1 uut(a,s,out);

  initial begin

    a=8'b01110010;
    #10 s[2]=0;s[1]=0;s[0]=0;
    #10 s[2]=0;s[1]=0;s[0]=1;
    #10 s[2]=0;s[1]=1;s[0]=0;
    #10 s[2]=0;s[1]=1;s[0]=1;
    #10 s[2]=1;s[1]=0;s[0]=0;
    #10 s[2]=1;s[1]=0;s[0]=1;
    #10 s[2]=1;s[1]=1;s[0]=0;
    #10 s[2]=1;s[1]=1;s[0]=1;
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end

endmodule
```

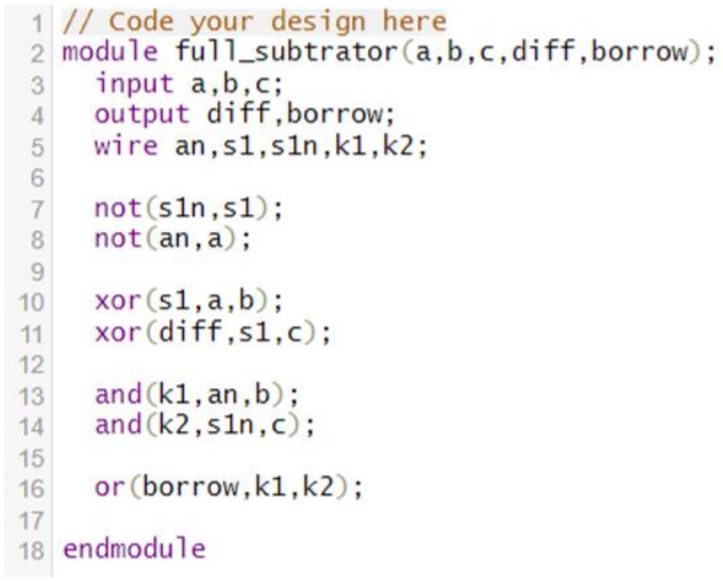
OUTPUT:



```
VCD info: dumpfile dump.vcd opened for output.
s2=x,s1=x,s0=x,output=x
s2=0,s1=0,s0=0,output=0
s2=0,s1=0,s0=1,output=1
s2=0,s1=1,s0=0,output=0
s2=0,s1=1,s0=1,output=0
s2=1,s1=0,s0=0,output=1
s2=1,s1=0,s0=1,output=1
s2=1,s1=1,s0=0,output=1
s2=1,s1=1,s0=1,output=0
```

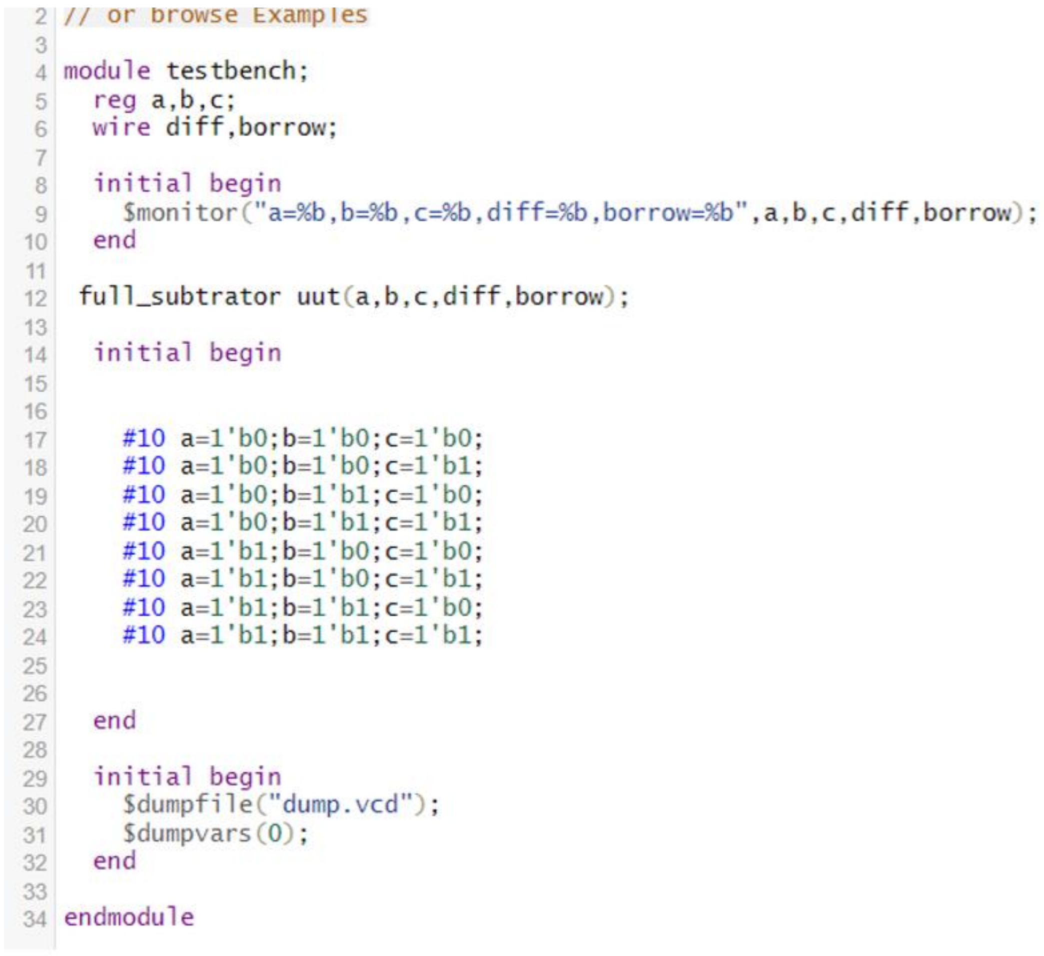
4] Design a Full Subtractor with Gate Level Modelling Style. (use primitive gates)

CODE:



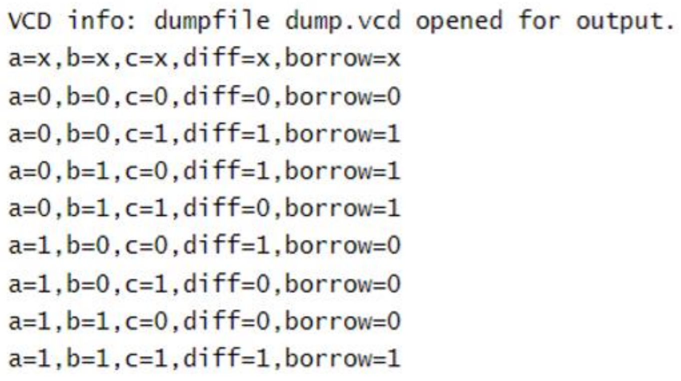
```
// Code your design here
module full_subtrator(a,b,c,diff,borrow);
  input a,b,c;
  output diff,borrow;
  wire an,s1,s1n,k1,k2;

  not(s1n,s1);
  not(an,a);

  xor(s1,a,b);
  xor(diff,s1,c);

  and(k1,an,b);
  and(k2,s1n,c);

  or(borrow,k1,k2);

endmodule
```

TEST BENCH:



```
// or browse Examples

module testbench;
  reg a,b,c;
  wire diff,borrow;

  initial begin
    $monitor("a=%b,b=%b,c=%b,diff=%b,borrow=%b",a,b,c,diff,borrow);
  end

 full_subtrator uut(a,b,c,diff,borrow);

  initial begin


    #10 a=1'b0;b=1'b0;c=1'b0;
    #10 a=1'b0;b=1'b0;c=1'b1;
    #10 a=1'b0;b=1'b1;c=1'b0;
    #10 a=1'b0;b=1'b1;c=1'b1;
    #10 a=1'b1;b=1'b0;c=1'b0;
    #10 a=1'b1;b=1'b0;c=1'b1;
    #10 a=1'b1;b=1'b1;c=1'b0;
    #10 a=1'b1;b=1'b1;c=1'b1;


  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end

endmodule
```

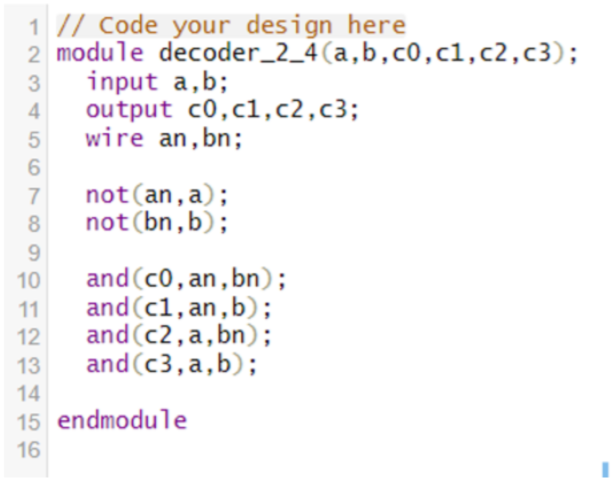
OUTPUT:



```
VCD info: dumpfile dump.vcd opened for output.
a=x,b=x,c=x,diff=x,borrow=x
a=0,b=0,c=0,diff=0,borrow=0
a=0,b=0,c=1,diff=1,borrow=1
a=0,b=1,c=0,diff=1,borrow=1
a=0,b=1,c=1,diff=0,borrow=1
a=1,b=0,c=0,diff=1,borrow=0
a=1,b=0,c=1,diff=0,borrow=0
a=1,b=1,c=0,diff=0,borrow=0
a=1,b=1,c=1,diff=1,borrow=1
```

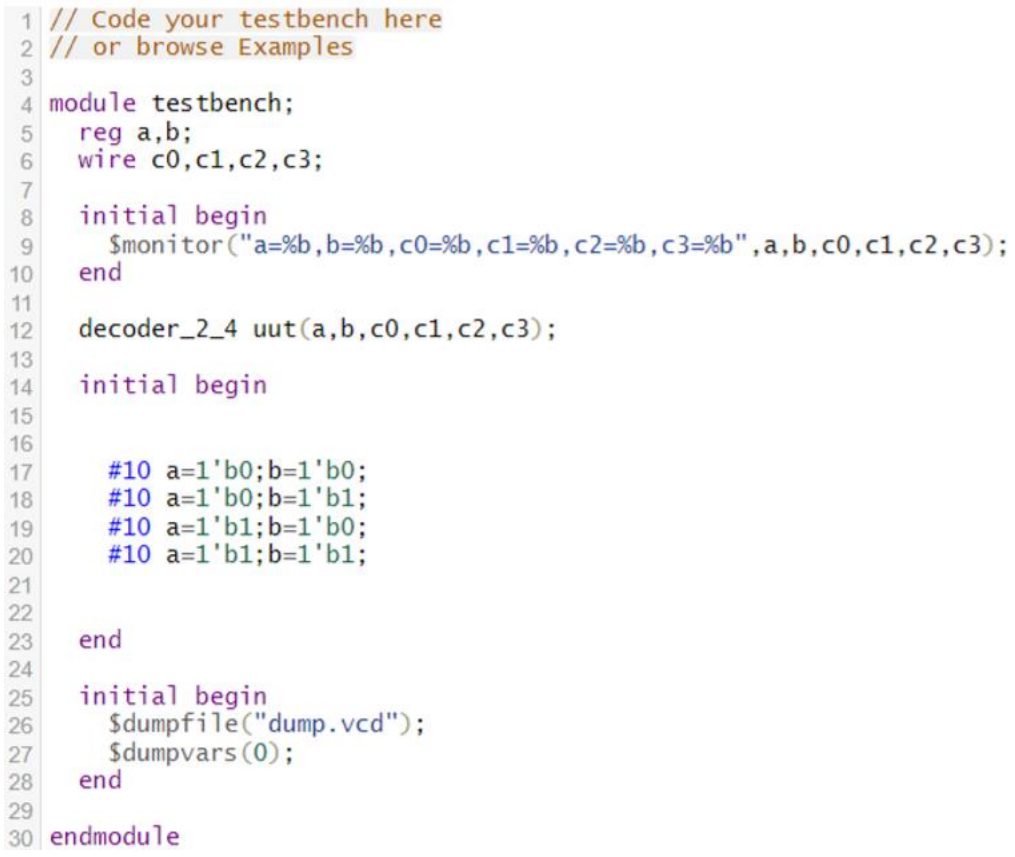
5] Design a 2X4 decoder using gate level modelling.

CODE:



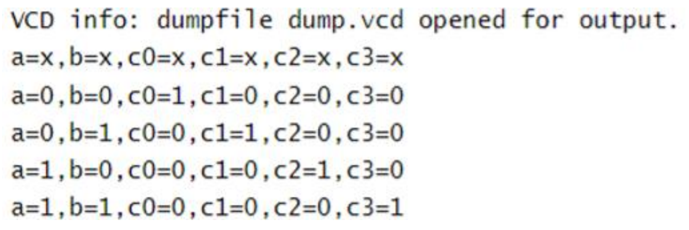
```verilog
// Code your design here
module decoder_2_4(a,b,c0,c1,c2,c3);
  input a,b;
  output c0,c1,c2,c3;
  wire an,bn;

  not(an,a);
  not(bn,b);

  and(c0,an,bn);
  and(c1,an,b);
  and(c2,a,bn);
  and(c3,a,b);

endmodule

```

TEST BENCH:



```verilog
// Code your testbench here
// or browse Examples

module testbench;
  reg a,b;
  wire c0,c1,c2,c3;

  initial begin
    $monitor("a=%b,b=%b,c0=%b,c1=%b,c2=%b,c3=%b",a,b,c0,c1,c2,c3);
  end

  decoder_2_4 uut(a,b,c0,c1,c2,c3);

  initial begin


    #10 a=1'b0;b=1'b0;
    #10 a=1'b0;b=1'b1;
    #10 a=1'b1;b=1'b0;
    #10 a=1'b1;b=1'b1;


  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end

endmodule
```

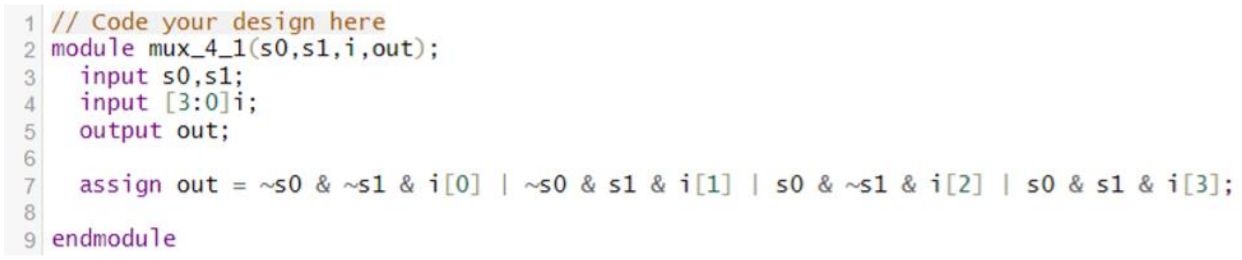
OUTPUT:



```
VCD info: dumpfile dump.vcd opened for output.
a=x,b=x,c0=x,c1=x,c2=x,c3=x
a=0,b=0,c0=1,c1=0,c2=0,c3=0
a=0,b=1,c0=0,c1=1,c2=0,c3=0
a=1,b=0,c0=0,c1=0,c2=1,c3=0
a=1,b=1,c0=0,c1=0,c2=0,c3=1
```

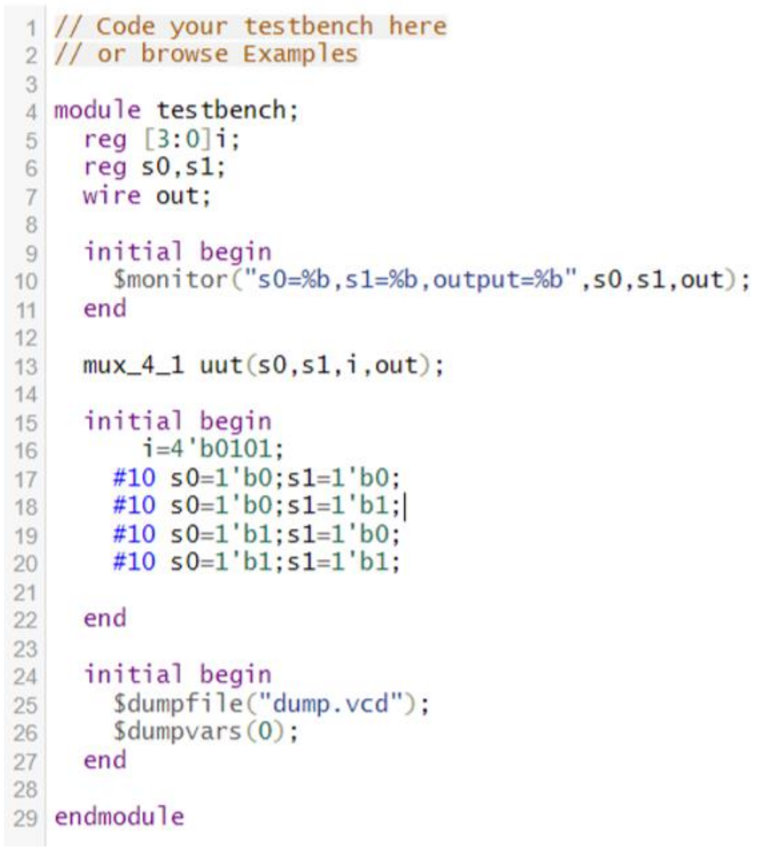
6] Design a 4x1 mux using operators. (use data flow)

CODE:



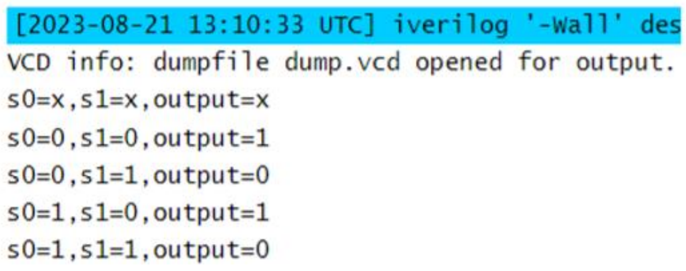
```
// Code your design here
module mux_4_1(s0,s1,i,out);
  input s0,s1;
  input [3:0]i;
  output out;

  assign out = ~s0 & ~s1 & i[0] | ~s0 & s1 & i[1] | s0 & ~s1 & i[2] | s0 & s1 & i[3];

endmodule
```

TEST BENCH:



```
// Code your testbench here
// or browse Examples

module testbench;
  reg [3:0]i;
  reg s0,s1;
  wire out;

  initial begin
    $monitor("s0=%b,s1=%b,output=%b",s0,s1,out);
  end

  mux_4_1 uut(s0,s1,i,out);

  initial begin
      i=4'b0101;
    #10 s0=1'b0;s1=1'b0;
    #10 s0=1'b0;s1=1'b1;
    #10 s0=1'b1;s1=1'b0;
    #10 s0=1'b1;s1=1'b1;

  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end

endmodule
```

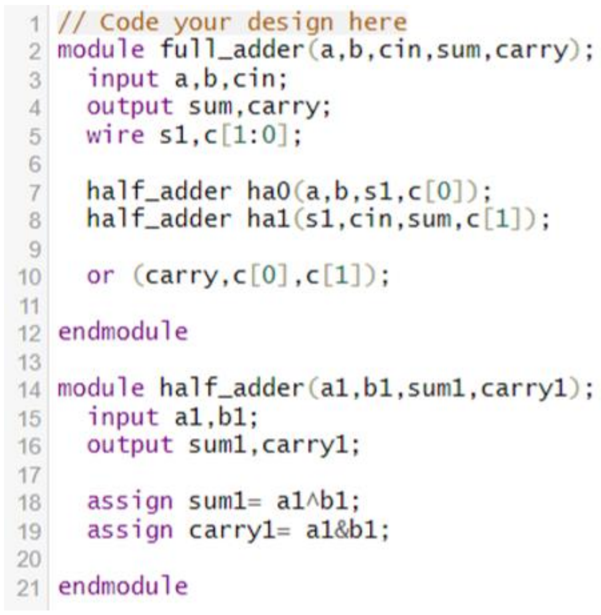
OUTPUT:



```
[2023-08-21 13:10:33 UTC] iverilog '-Wall' des
VCD info: dumpfile dump.vcd opened for output.
s0=x,s1=x,output=x
s0=0,s1=0,output=1
s0=0,s1=1,output=0
s0=1,s1=0,output=1
s0=1,s1=1,output=0
```

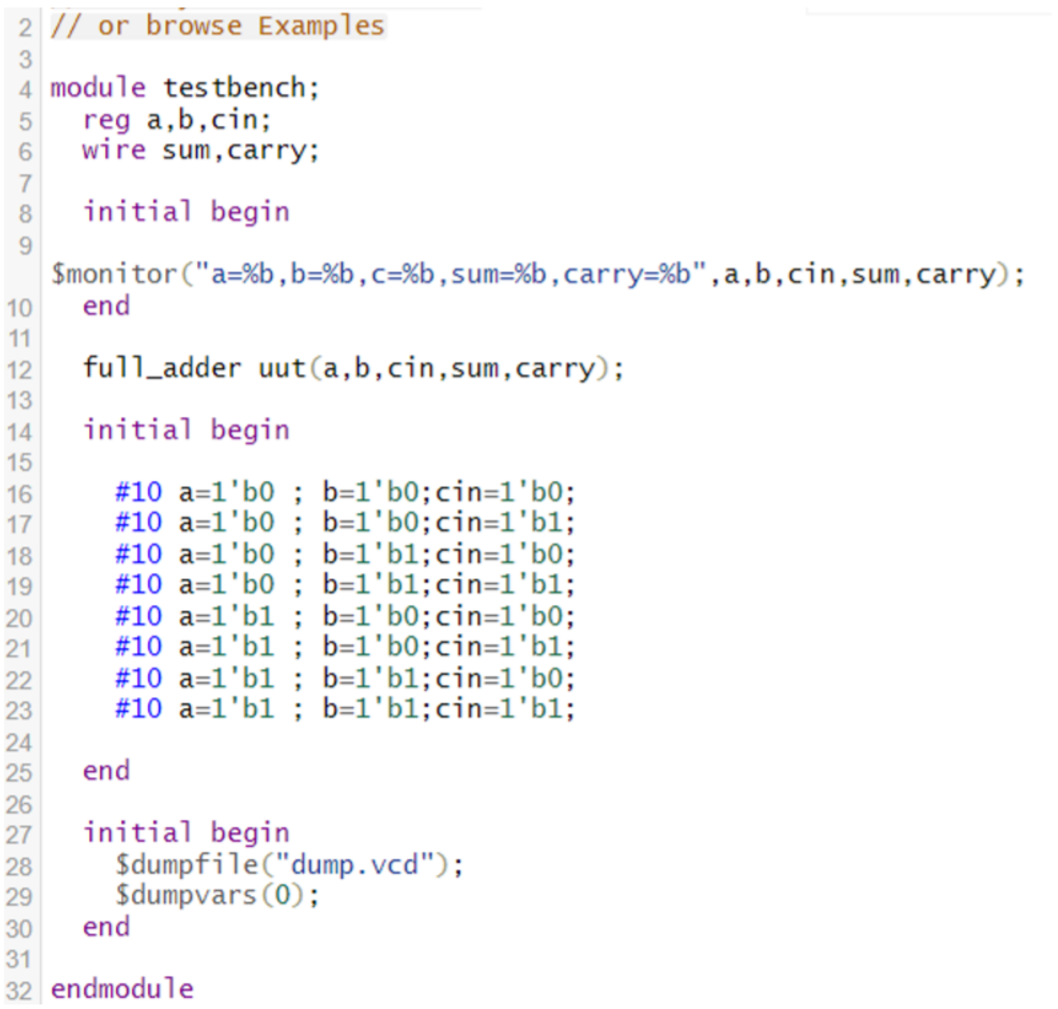
7] Design a Full adder using half adder.

CODE:



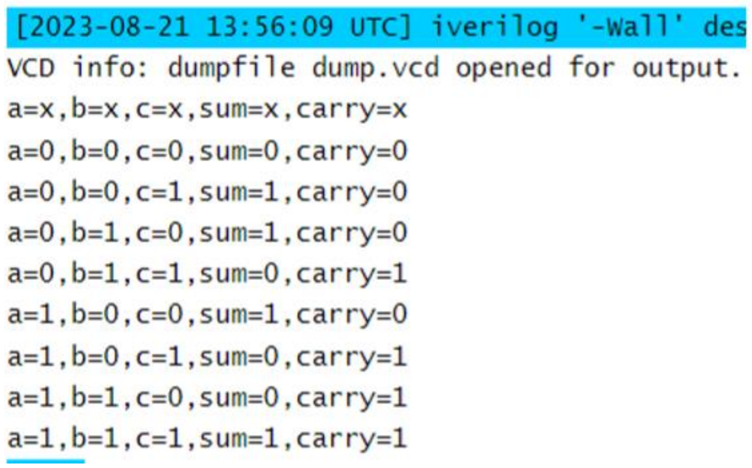
```
// Code your design here
module full_adder(a,b,cin,sum,carry);
  input a,b,cin;
  output sum,carry;
  wire s1,c[1:0];

  half_adder ha0(a,b,s1,c[0]);
  half_adder ha1(s1,cin,sum,c[1]);

  or (carry,c[0],c[1]);

endmodule

module half_adder(a1,b1,sum1,carry1);
  input a1,b1;
  output sum1,carry1;

  assign sum1= a1^b1;
  assign carry1= a1&b1;

endmodule
```

TEST BENCH:



```
// or browse Examples

module testbench;
  reg a,b,cin;
  wire sum,carry;

  initial begin

$monitor("a=%b,b=%b,c=%b,sum=%b,carry=%b",a,b,cin,sum,carry);
  end

  full_adder uut(a,b,cin,sum,carry);

  initial begin

    #10 a=1'b0 ; b=1'b0;cin=1'b0;
    #10 a=1'b0 ; b=1'b0;cin=1'b1;
    #10 a=1'b0 ; b=1'b1;cin=1'b0;
    #10 a=1'b0 ; b=1'b1;cin=1'b1;
    #10 a=1'b1 ; b=1'b0;cin=1'b0;
    #10 a=1'b1 ; b=1'b0;cin=1'b1;
    #10 a=1'b1 ; b=1'b1;cin=1'b0;
    #10 a=1'b1 ; b=1'b1;cin=1'b1;

  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end

endmodule
```

OUTPUT:



```
[2023-08-21 13:56:09 UTC] iverilog '-Wall' des
VCD info: dumpfile dump.vcd opened for output.
a=x,b=x,c=x,sum=x,carry=x
a=0,b=0,c=0,sum=0,carry=0
a=0,b=0,c=1,sum=1,carry=0
a=0,b=1,c=0,sum=1,carry=0
a=0,b=1,c=1,sum=0,carry=1
a=1,b=0,c=0,sum=1,carry=0
a=1,b=0,c=1,sum=0,carry=1
a=1,b=1,c=0,sum=0,carry=1
a=1,b=1,c=1,sum=1,carry=1
```