# Birla Vishvakarma Mahavidyalaya

## Engineering Collage (An Autonomous Institution)

**Subject: -** Digital System Design (3EL42)

Prof. Chintan Patel

**Name: -** Shrey Shah

**ID Number: -** 21EL080

**Division: -** 11

**Year: -** 2023-24

**Branch: -** Electronics

# Assignment 2

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

VERILOG CODE :

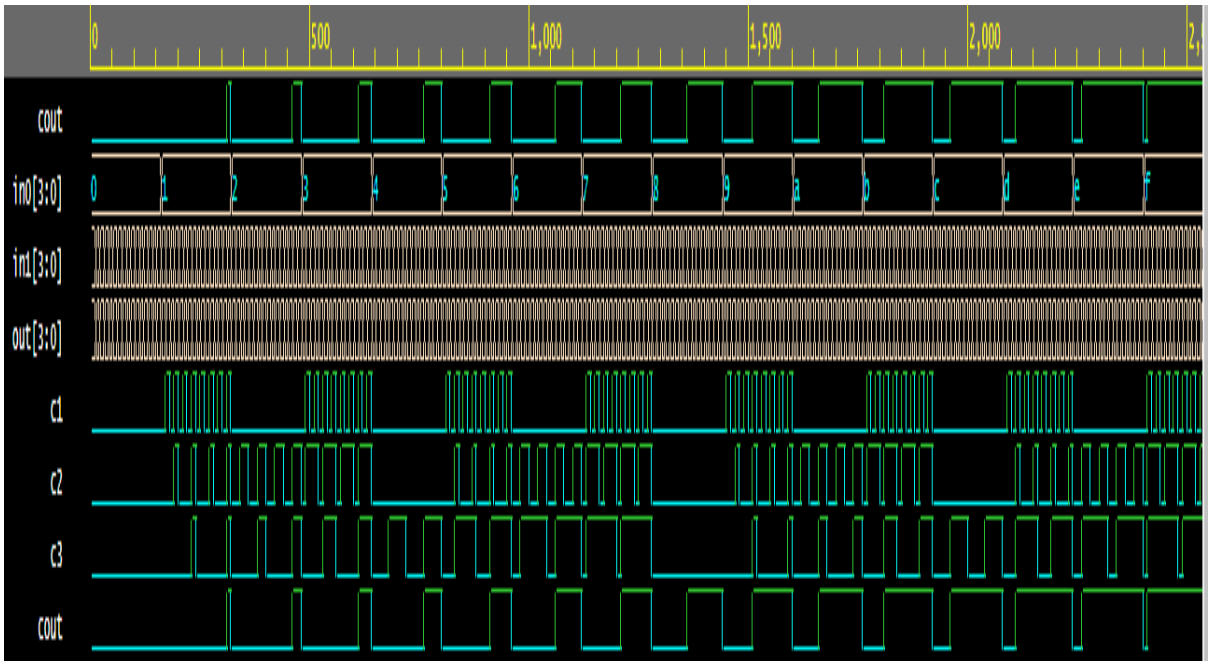
```
module full_adder(
    input a,
    input b,
    input Cin,
    output sum,
    output cout
    );

    assign sum = a ^ b ^ Cin;
    assign cout = (a * b) |  (b * Cin) | (a * Cin);

endmodule
```

```
module ripple_carry(
    input [3:0] a,
    input [3:0] b,
    input cin,
    output [3:0] s,
    output cout
    );

     wire c1,c2,c3;

       full_adder fa1(a[0],b[0],cin,s[0],c1);
       full_adder fa2(a[1],b[1],c1,s[1],c2);
       full_adder fa3(a[2],b[2],c2,s[2],c3);
       full_adder fa4(a[3],b[3],c3,s[3],cout);

endmodule
```

## TESTBENCH :

```
1   module ripple_carry_tb;
2
3       reg [3:0] a;
4       reg [3:0] b;
5       reg cin;
6       wire [3:0] s;
7       wire cout;
8
9     initial begin
10
11  $monitor("a = %b | b = %b | cin = %b | s = %b | cout = %b",a,b,cin,s,cout);
12
13    end
14
15      ripple_carry uut(a,b,cin,s,cout);
16
17      initial begin
18
19      #100 a = 4'b0000 ; b = 4'b0000 ; cin = 0;
20      #100 a = 4'b0001 ; b = 4'b0001 ; cin = 0;
21      #100 a = 4'b0011 ; b = 4'b0011 ; cin = 0;
22      #100 a = 4'b0111 ; b = 4'b0111 ; cin = 0;
23      #100 a = 4'b1111 ; b = 4'b1111 ; cin = 0;
24      #100 a = 4'b0000 ; b = 4'b0000 ; cin = 1;
25      #100 a = 4'b0001 ; b = 4'b0001 ; cin = 1;
26      #100 a = 4'b0011 ; b = 4'b0011 ; cin = 1;
27      #100 a = 4'b0111 ; b = 4'b0111 ; cin = 1;
28      #100 a = 4'b1111 ; b = 4'b1111 ; cin = 1;
29      #100 $finish;
30
31      end
32  initial begin
33      $dumpfile("dump.vcd");
34      $dumpvars(0);
35    end
36  endmodule
```

## OUTPUT :



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

## VERILOG CODE :

```
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
```

```
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
```

## TESTBENCH :

```
1   module rbit_johnson_counter_tb;
2
3     reg clk,reset;
4     wire [3:0] q;
5
6     _4bit_johnson_counter uut(clk,reset,q);
7
8     initial
9     begin
10
11        reset = 1'b1;
12        clk = 1'b1;
13
14    end
15
16    always #10 clk = ~clk;
17
18    initial
19    begin
20      #00  reset = 1'b1;
21      #20 reset = 1'b0;
22      #500 $finish;
23
24    end
25
26  endmodule
```

## OUTPUT:

```
  0 clk=0, out= xxxx, reset=1
  5 clk=1, out= 0000, reset=1
 10 clk=0, out= 0000, reset=1
 15 clk=1, out= 0000, reset=1
 20 clk=0, out= 0000, reset=0
 25 clk=1, out= 0001, reset=0
 30 clk=0, out= 0001, reset=0
 35 clk=1, out= 0011, reset=0
 40 clk=0, out= 0011, reset=0
 45 clk=1, out= 0111, reset=0
 50 clk=0, out= 0111, reset=0
 55 clk=1, out= 1111, reset=0
 60 clk=0, out= 1111, reset=0
 65 clk=1, out= 1110, reset=0
 70 clk=0, out= 1110, reset=0
 75 clk=1, out= 1100, reset=0
 80 clk=0, out= 1100, reset=0
 85 clk=1, out= 1000, reset=0
 90 clk=0, out= 1000, reset=0
 95 clk=1, out= 0000, reset=0
100 clk=0, out= 0000, reset=0
```

# Q-3 Reuse 2:1 Mux code to implement 8:1 Mux.

## VERILOG CODE :

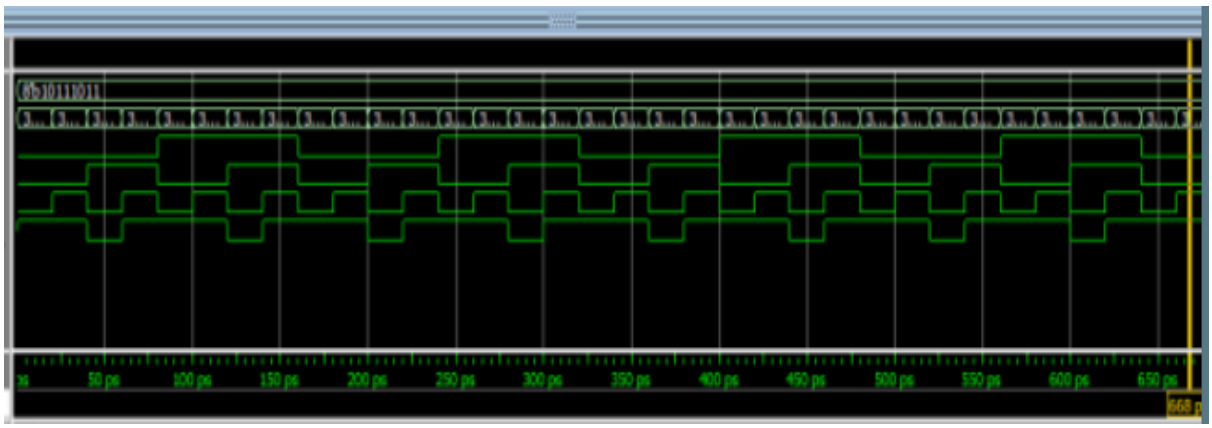
```
1   module mux__21(
2       input x0,
3       input x1,
4       input s,
5       output out
6       );
7
8       assign out = x0*~s | x1*s;
9
10  endmodule
```

```
1   module mux_81(
2       input [7:0] x,
3       input [2:0] s,
4       output out
5       );
6
7       wire k1,k2,k3,k4,k5,k6;
8
9       mux__21  m1(x[0],x[1],s[0],k1);
10      mux__21  m2(x[2],x[3],s[0],k2);
11      mux__21  m3(x[4],x[5],s[0],k3);
12      mux__21  m4(x[6],x[7],s[0],k4);
13      mux__21  m5(k1,k2,s[1],k5);
14      mux__21  m6(k3,k4,s[1],k6);
15      mux__21  m7(k5,k6,s[2],out);
16
17
18  endmodule
```

## TESTBENCH :

```
1   module mux_81_tb;
2
3   reg [7:0]x;
4   reg[2:0]s;
5   wire out;
6
7     initial begin
8
9     $monitor( " %t | x = %b | s2 = %b | s1 = %b | s0 = %b | out = %b " ,$time,x,s[2],s[1],s[0],out);
10
11    end
12
13    mux_81 uut(x,s,out);
14
15    initial begin
16    x = 8'b10101010;
17
18    #100  x=8'b10101010; s[2]= 0 ; s[1]= 0 ; s[0]= 0;
19    #100  x=8'b10101010; s[2]= 0 ; s[1]= 0 ; s[0]= 1;
20    #100  x=8'b10101010; s[2]= 0 ; s[1]= 1 ; s[0]= 0;
21    #100  x=8'b10101010; s[2]= 0 ; s[1]= 1 ; s[0]= 1;
22    #100  x=8'b10101010; s[2]= 1 ; s[1]= 0 ; s[0]= 0;
23    #100  x=8'b10101010; s[2]= 1 ; s[1]= 0 ; s[0]= 1;
24    #100  x=8'b10101010; s[2]= 1 ; s[1]= 1 ; s[0]= 0;
25    #100  x=8'b10101010; s[2]= 1 ; s[1]= 1 ; s[0]= 1;
26    #100 $finish;
27
28    end
29    initial begin
30
31    $dumpfile("dump.vcd");
32    $dumpvars(0);
33
34    end
35  endmodule
```

## OUTPUT:-



# Q-4 Design a Full Subtractor with Gate Level Modelling Style (Use primitive gates).

## VERILOG CODE :

```
1   module and_21(
2       input x,
3       input y,
4       output z
5       );
6
7       assign z = x*y;
8   endmodule
```

```
1   module or_21(
2       input x,
3       input y,
4       output z
5       );
6
7       assign z = x|y;
8   endmodule
```

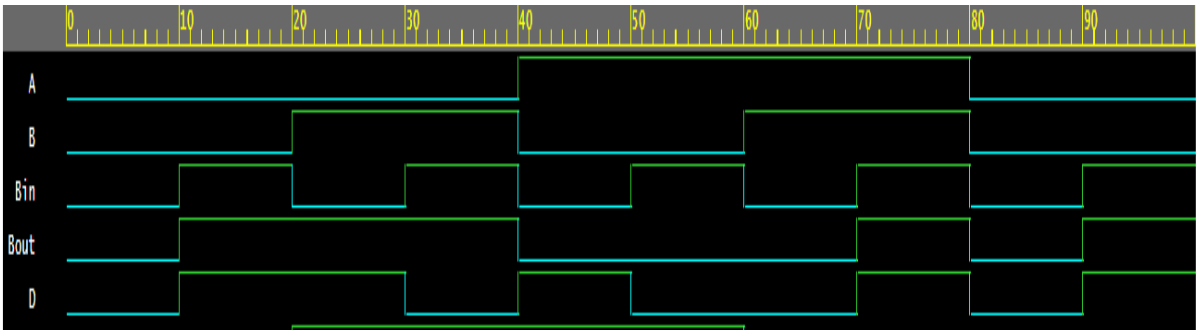
```
1   module xor_21(
2       input x,
3       input y,
4       output z
5       );
6
7       assign z=x^y;
8   endmodule
```

```
1   module full_subtractor_gate_level(
2       input x,
3       input y,
4       input z,
5       output difference,
6       output borrow
7       );
8
9       xor_21 x1(x,y,k1);
10      xor_21 x2(k1,z,difference);
11
12      and_21 a1(~x,y,k2);
13      and_21 a2(~x,z,k3);
14      and_21 a3(y,z,k4);
15
16      or_21 o1(k2,k3,k5);
17      or_21 o2(k5,k4,borrow);
18
19  endmodule
```

## TESTBENCH :

```
1   module full_subtractor_tb;
2
3     reg x,y,z;
4     wire difference,borrow;
5
6     initial begin
7
8       $monitor($time |"x = %b | y = %b | z = %b | diff = %b | borrow = %b ",x,y,z,difference,borrow);
9
10    end
11
12     full_subtractor_gate_level uut(x,y,z,difference,borrow);
13
14
15    initial begin
16
17      #000 x=0; y=0; z=0;
18      #100 x=0; y=0; z=1;
19      #100 x=0; y=1; z=0;
20      #100 x=0; y=1; z=1;
21      #100 x=1; y=0; z=0;
22      #100 x=1; y=0; z=1;
23      #100 x=1; y=1; z=0;
24      #100 x=1; y=1; z=1;
25
26      #100 $finish;
27
28    end
29
30    initial begin
31
32      $dumpfile("dump.vcd");
33      $dumpvars(0);
34
35    end
36  endmodule
```

## OUTPUT :



# Q-5 Design a 2X4 decoder using gate level modelling.

## VERILOG CODE :

```
module decoder_24_gate_level(
    input x1,
    input x2,
    output y1,
    output y2,
    output y3,
    output y4
    );

    and a1(y1,~x1,~x2);
    and a2(y2,~x1,x2);
    and a3(y3,x1,~x2);
    and a4(y4,x1,x2);

endmodule
```

## TESTBENCH :

```
module decoder_24_tb;
  reg x1,x2;
  wire y1,y2,y3,y4;

  initial begin

     $monitor(" %t | x1 = %b | x2 = %b | y1 = %b | y2 = %b | y3 = %b | y4 = %b ",$time,x1,x2,y1,y2,y3,y4);

  end
   decoder_24_gate_level uut(x1,x2,y1,y2,y3,y4);
    initial begin

    #000 x1=0; x2=0;
    #100 x1=0; x2=1;
    #100 x1=1; x2=0;
    #100 x1=1; x2=1;
    #100 $finish;

 end

  initial begin

    $dumpfile("dump.vcd");
    $dumpvars(0);

  end

endmodule
```

## OUTPUT:-

```
# KERNEL: en=0 a=x b=x y=xxxx
# KERNEL: en=1 a=0 b=0 y=1111
# KERNEL: en=1 a=0 b=1 y=1111
# KERNEL: en=1 a=1 b=0 y=1111
# KERNEL: en=1 a=1 b=1 y=1111
```

# Q-6 Design a 4x1 mux using operators. (Use data flow)

## VERILOG CODE :

```
1     module mux_41__operator(
2         input x1,
3         input x2,
4         input x3,
5         input x4,
6         input s0, s1,
7         output y
8         );
9         assign  y = s1 ?  ( s0 ? x4:x3) : ( s0 ? x2:x1);
10    endmodule
```

## TESTBENCH :

```
1   module mux_41__tb;
2
3   reg x1,x2,x3,x4,s0,s1;
4     wire y;
5
6     initial begin
7
8        $monitor(" %t | s0 = %b | s1 = %b | y = %b ",$time,s0,s1,y);
9
10    end
11     mux_41__operator uut(x1,x2,x3,x4,s0,s1,y);
12      initial begin
13     x1=1;
14     x2=1;
15     x3=1;
16     x4=1;
17
18      #000 s0=0; s1=0;
19      #100 s0=0; s1=1;
20      #100 s0=1; s1=0;
21      #100 s0=1; s1=1;
22      #100 $finish;
23
24   end
25    initial begin
26
27      $dumpfile("dump.vcd");
28      $dumpvars(0);
29
30    end
31
32  endmodule
```

## OUTPUT :

```
sel = 00 -> i3 = 0, i2 = 1 ,i1 = 0, i0 = 1 -> y = 1
sel = 01 -> i3 = 0, i2 = 1 ,i1 = 0, i0 = 1 -> y = 0
sel = 11 -> i3 = 0, i2 = 1 ,i1 = 0, i0 = 1 -> y = 0
sel = 01 -> i3 = 0, i2 = 1 ,i1 = 0, i0 = 1 -> y = 0
```

# Q-7 Design a Full adder using half adder.

## VERILOG CODE:

```
module half_adder(
    input x,
    input y,
    output sum,
    output carry
    );

  assign  sum = x^y;
  assign  carry = x*y;
endmodule
```

```
module or_21(
    input x,
    input y,
    output z
    );
    assign z= x|y;
endmodule
```

```
module fulladder_using_halfadder(
    input x,
    input y,
    input z,
    output sum,
    output carry
    );

    half_adder ha1 ( .x(x), .y(y), .sum(k1), .carry(c1));
    half_adder ha2 ( .x(k1), .y(z), .sum(sum), .carry(c2));

    or_21 o1( .x(c1),.y(c2),.z(carry));
endmodule
```

## TESTBENCH :

```
1   module fulladder_using_half_tb;
2
3     reg x,y,z;
4     wire carry,sum;
5
6     initial begin
7
8        $monitor($time |"x = %b | y = %b | z = %b | sum = %b | carry = %b ",x,y,z,carry,sum);
9
10    end
11
12     fulladder_using_halfadder uut(x,y,z,carry,sum);
13
14
15    initial begin
16
17      #000 x=0; y=0; z=0;
18      #100 x=0; y=0; z=1;
19      #100 x=0; y=1; z=0;
20      #100 x=0; y=1; z=1;
21      #100 x=1; y=0; z=0;
22      #100 x=1; y=0; z=1;
23      #100 x=1; y=1; z=0;
24      #100 x=1; y=1; z=1;
25
26      #100 $finish;
27
28    end
29
30    initial begin
31
32      $dumpfile("dump.vcd");
33      $dumpvars(0);
34
35    end
36
37  endmodule
```

## OUTPUT :

```
At time 0: a=0 b=0, cin=0, sum=0, carry=0
At time 1: a=0 b=0, cin=1, sum=1, carry=0
At time 2: a=0 b=1, cin=0, sum=1, carry=0
At time 3: a=0 b=1, cin=1, sum=0, carry=1
At time 4: a=1 b=0, cin=0, sum=1, carry=0
At time 5: a=1 b=0, cin=1, sum=0, carry=1
At time 6: a=1 b=1, cin=0, sum=0, carry=1
At time 7: a=1 b=1, cin=1, sum=1, carry=1
```