ECT 203 – Logic Circuit Design

Course Project

Name : Amalkrishnan P
Register number : TVE22AE012
Class Roll number : 12

1. BCD Adder

- **Realisation of One Bit Parallel Adder**

Digital > fulladder > fulladder.v

```
// Full Adder (one bit parallel adder)

module fulladder(
  input a, b, c_in,
  output reg sum,
  output reg c_out
);

always @(*)
begin

    sum   = ((a)   ^    (b)   ^   (c_in));
    c_out = ((a&b) | (b&c_in) | (c_in&a));

end
endmodule

```

Design of one bit parallel adder

Digital > fulladder > fulladder_tb.v

```
// Test bench for full adder

module full_adder_tb;
// Testbench Variable declaration
reg     a;
reg     b;
reg     c_in;
wire    sum;
wire    c_out;
integer i;

// Instantiating and connecting to testbench variables
fulladder uut(
    .a      ( a         ),
    .b      ( b         ),
    .c_in   ( c_in      ),
    .sum    ( sum       ),
    .c_out  ( c_out     )
);

initial begin

    a = 0; b = 0; c_in = 0;
    $monitor("A=%b B=%b Sum=%b Cout=%b",a,b,sum,c_out);
    // Providing stimulus to test the design
    for( i = 0; i < 4; i = i + 1 )
        begin
            #10
            {a,b} = i; // giving two bits as inputs
        end
end
endmodule
```

Testbench of one bit parallel adder

```
amal  Aspire-A715-42G  ../Digital/fulladder  vvp fulladder
A=0 B=0 Sum=0 Cout=0
A=0 B=1 Sum=1 Cout=0
A=1 B=0 Sum=1 Cout=0
A=1 B=1 Sum=0 Cout=1
```

Test of one bit parallel adder

- **Realisation of 4 Bit Parallel Adder**

Digital > fulladder > 4bitadder.v

```verilog
// Full Adder (Using four one bit parallel adders)

module adder(

  input         [3:0] a,
  input         [3:0] b,
  input         c_in0  ,
  output        [3:0] s,
  output        c_out0 ,
  output        [5:0] w
);

wire    [3:0] a ;
wire    [3:0] b ;
wire    c_in0   ;
wire    [3:0] s ;
wire    c_out0  ;
wire    [5:0] w ;


fulladder s0( .a(a[0]) , .b(b[0]), .c_in(c_in0), .sum(s[0]), .c_out(w[0]) );
fulladder s2( .a(a[1]) , .b(b[1]), .c_in(w[1] ), .sum(s[1]), .c_out(w[2]) );
fulladder s3( .a(a[2]) , .b(b[2]), .c_in(w[3] ), .sum(s[2]), .c_out(w[4]) );
fulladder s4( .a(a[3]) , .b(b[3]), .c_in(w[5] ), .sum(s[3]), .c_out(c_out0));


assign   w[1] = w[0];
assign   w[3] = w[2];
assign   w[5] = w[4];

endmodule
module fulladder(
  input a, b, c_in,
  output sum, c_out
);

assign    sum   = ((a)   ^    (b)   ^ (c_in)  );
assign    c_out = ((a&b) | (b&c_in) | (c_in&a));

endmodule
```

Design of 4 bit parallel adder

Digital > fulladder > 4bit_tb.v

```verilog
// Test bench for full adder

module adder_tb;
// Testbench Variable declaration
reg     [3:0]a  ;
reg     [3:0]b  ;
reg     c_in    ;
wire    [3:0]sum;
wire    c_out   ;


// Instantiating and connecting to testbench variables
adder uut(
    .a          ( a         ),
    .b          ( b         ),
    .c_in0      ( c_in      ),
    .s          ( sum       ),
    .c_out0     ( c_out     )
);
initial begin

    $monitor("A=%b B=%b Sum=%b Cout=%b",a,b,sum,c_out);

    a = 4'd0;     b = 4'd0;    c_in = 0;#10
    a = 4'd1;     b = 4'd2;    c_in = 0;#10
    a = 4'd1;     b = 4'd3;    c_in = 0;#10
    a = 4'd1;     b = 4'd4;    c_in = 0;#10
    a = 4'd1;     b = 4'd1;    c_in = 0;#10
    a = 4'd15;    b = 4'd15;   c_in = 0;
    $monitor("A=%b B=%b Sum=%b Cout=%b",a,b,sum,c_out);

end
endmodule
```

Testbench of 4 bit parallel adder

```
amal  Aspire-A715-42G  ../Digital/fulladder  vvp 4bitadder
A=0000 B=0000 Sum=0000 Cout=0
A=0001 B=0010 Sum=0011 Cout=0
A=0001 B=0011 Sum=0100 Cout=0
A=0001 B=0100 Sum=0101 Cout=0
A=0001 B=0001 Sum=0010 Cout=0
A=1111 B=1111 Sum=1110 Cout=1
```

Test of 4 bit parallel adder

- **Realisation of BCD Adder**

Digital > fulladder > 4bcd.v

```
module bcd(

input       [3:0]a      ,
input       [3:0]b      ,
input       c_in        ,
output      c_out       ,
output      [3:0]s

);

wire    [3:0]a              ;
wire    [3:0]b              ;
wire    c_in                ;
wire    c_out               ;
wire    [3:0]s              ;
wire    [3:0]ws             ;
wire    wc                  ;
wire    [3:0]wa             ;
wire    cin                 ;
wire    wor1,wor2,wor3,wor4;

adder a1(   .a(a) ,  .b(b ),  .c_in0(c_in), .s(ws), .c_out0(wc    ) );
adder a2(   .a(wa) , .b(ws) , .c_in0(cin ), .s(s ) , .c_out0(c_out) );

assign wor1     =    wc                 ;
assign wor2     =    ws[3] & ws[2]      ;
assign wor3     =    ws[3] & ws[1]      ;
assign wor4     =    wor1 | wor2 |  wor3;
assign wa[2]    =    wor4               ;
assign wa[1]    =    wor4               ;
assign wa[0]    =    0                  ;
assign wa[3]    =    0                  ;
assign cin      =    0                  ;
assign c_in     =    0                  ;
endmodule
```

Digital > fulladder > 4bcd.v

```
module adder(
input           [3:0] a     ,
input           [3:0] b     ,
input           c_in0       ,
output          [3:0] s     ,
output          c_out0

);

wire                [3:0] a   ;
wire                [3:0] b   ;
wire                c_in0     ;
wire                [3:0] s   ;
wire                c_out0    ;
wire                [5:0] w   ;

full_adder s1(  .a(a[0])  , .b(b[0])  , .c_in( c_in0)   , .sum(s[0])    , .c_out(w[0])      );
full_adder s2(  .a(a[1])  , .b(b[1])  , .c_in( w[1] )   , .sum(s[1])    , .c_out(w[2])      );
full_adder s3(  .a(a[2])  , .b(b[2])  , .c_in( w[3] )   , .sum(s[2])    , .c_out(w[4])      );
full_adder s4(  .a(a[3])  , .b(b[3])  , .c_in( w[5] )   , .sum(s[3])    , .c_out(c_out0)    );

assign w[1] = w[0] ;
assign w[3] = w[2] ;
assign w[5] = w[4] ;
endmodule

module full_adder (

input a,b,c_in,
output sum,c_out
);

assign sum   = ((a) ^ (b) ^ (c_in));
assign c_out = (a & b) | (b & c_in)  | (c_in & a) ;
endmodule
```

Design of BCD adder

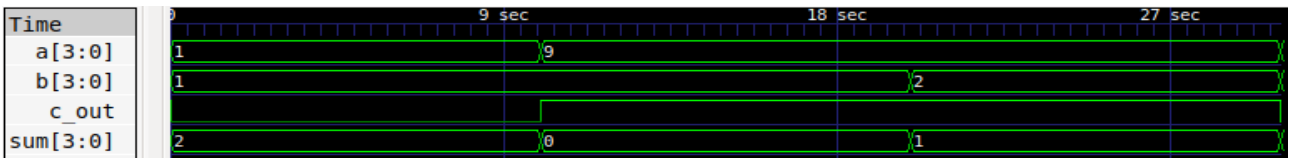
Digital > fulladder > 4bcd_tb.v

```verilog
module bcd_tb;

reg     [3:0]a              ;
reg     [3:0]b              ;
reg     c_in                ;
wire    [3:0]sum            ;
wire     c_out              ;

bcd      uut(
        .a          (   a       ),
        .b          (   b       ),
        .c_in       (   c_in    ),
        .s          (   sum     ),
        .c_out      (   c_out   )

    );

initial begin
$dumpfile("dp.vcd");$dumpvars;
$monitor("a = %b    b = %b  sum =%b     cout=%b",a,b,sum,c_out);

a = 4'd1;  b = 4'd1 ;  c_in = 0;#10  // 1 + 1 = 2
a = 4'd9;  b = 4'd1 ;  c_in = 0;#10  // 9 + 1 = 10
a = 4'd9;  b = 4'd2 ;  c_in = 0;     // 9 + 2 = 11
end
endmodule
```

Testbench of BCD adder

```
amal  Aspire-A715-42G  ../Digital/fulladder  vvp 4bcd
a = 0001        b = 0001        sum =0010       cout=0
a = 1001        b = 0001        sum =0000       cout=1
a = 1001        b = 0010        sum =0001       cout=1
```

Test of BCD adder



Waveform of BCD adder

Explanation

• Implemented a one bit parallel adder in verilog.

• Cascaded 4 such one bit parallel adders to create a 4 bit parallel adder.When bits are added one after the other to obtain the appropriate total (a1 plus b1 to get the corresponding sum (s1) and carry (cout1), this is known as cascading working. The first bit's carry (c) is treated as zero. The previous carryout (which we acquired during the previous addition of 2 bits) is now provided as the third input for the addition of the following 2 bits. In other words, we add a2, b2, and cout1 to yield s2 and cout 2.

• Then using 2 4 bit parallel adders BCD Adder was implemented.This kind of adder adds two BCD numbers together.The Bcd sum is the resulting sum if the total exceeds nine. Six is added to the total along with any carry, if any.When the total is less than nine, binary addition is done normally.

2 Realisation of BCD Subtractor

Digital > bcd_subtractor > subtractor.v

```
module subtractor(

input       [3:0]a      ,
input       [3:0]b      ,
output      [3:0]s      ,
output      sign        ,
output      c_out

);

wire    [3:0]a      ;
wire    [3:0]b      ;
wire    [3:0]bwire  ;
wire    c_out       ;
wire    [3:0]s      ;
wire    [3:0]wa     ;
wire    onewire     ;
wire    [3:0]ws1    ;
wire    [3:0]ws2    ;
wire    twowire     ;
wire    wc1,wc2     ;
wire    sign        ;

adder a1(   .a(a),  .b(bwire),  .c_in0(onewire),    .s(ws1), .c_out0(wc1)   );
adder a2(   .a(wa), .b(ws2) ,   .c_in0(wc2) ,       .s(s) ,  .c_out0(c_out) );
comparator c1(  .a(a),   .b(b),   .lt(sign) );

assign onewire = 1'd1;

assign bwire[3] = (b[3]) ^ (onewire);
assign bwire[2] = (b[2]) ^ (onewire);
assign bwire[1] = (b[1]) ^ (onewire);
assign bwire[0] = (b[0]) ^ (onewire);

assign wc2      = ~(wc1);

assign ws2[3]   = (ws1[3]) ^ (wc2)  ;
assign ws2[2]   = (ws1[2]) ^ (wc2)  ;
assign ws2[1]   = (ws1[1]) ^ (wc2)  ;
assign ws2[0]   = (ws1[0]) ^ (wc2)  ;
assign wa       =  4'd0;
endmodule

module adder(

    input           [3:0] a     ,
    input           [3:0] b     ,
    input           c_in0       ,
    output          [3:0] s     ,
    output          c_out0

);

wire                [3:0] a   ;
wire                [3:0] b   ;
wire                c_in0     ;
wire                [3:0] s   ;
wire                c_out0    ;
wire                [5:0] w   ;

full_adder s1(  .a(a[0])  , .b(b[0])  , .c_in( c_in0)   , .sum(s[0])    , .c_out(w[0])      );
full_adder s2(  .a(a[1])  , .b(b[1])  , .c_in( w[1] )   , .sum(s[1])    , .c_out(w[2])      );
full_adder s3(  .a(a[2])  , .b(b[2])  , .c_in( w[3] )   , .sum(s[2])    , .c_out(w[4])      );
full_adder s4(  .a(a[3])  , .b(b[3])  , .c_in( w[5] )   , .sum(s[3])    , .c_out(c_out0)    );

assign w[1] = w[0] ;
assign w[3] = w[2] ;
assign w[5] = w[4] ;
endmodule
```

```
module full_adder (

    input a,b,c_in,
    output sum,c_out
);

assign sum  = ((a) ^ (b) ^ (c_in));
assign c_out = (a & b) | (b & c_in)  | (c_in & a) ;
endmodule

module comparator(
input [3:0] a,b,
output reg lt
);

always @(*)
begin
 if (a<b)
    begin
    lt = 1'b1;
    end
 else
    begin
    lt = 1'b0;
    end
end
endmodule
```

Design of BCD Subtractor

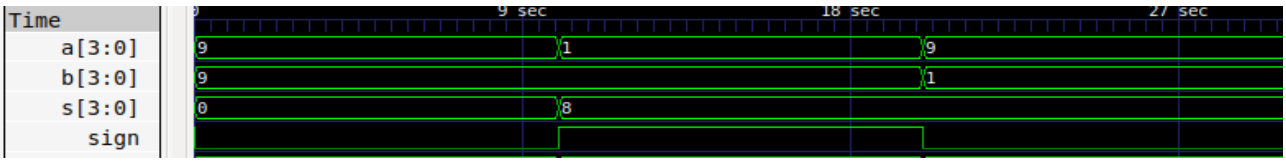
Digital > bcd_subtractor > subtractor_tb.v

```
module subtractor_tb;

reg     [3:0]a              ;
reg     [3:0]b              ;
wire    [3:0]s              ;
wire     c_out              ;
wire     sign               ;

subtractor       uut(
        .a          (   a       ),
        .b          (   b       ),
        .s          (   s       ),
        .c_out      (   c_out   ),
        .sign       (   sign    )

    );

initial begin
$dumpfile("dp.vcd");$dumpvars;
$monitor("a = %b    b = %b  signed binary representation of sum =%b000_%b ",a,b,sign,s);

a = 4'd9;  b = 4'd9 ;  #10
a = 4'd1;  b = 4'd9 ;  #10
a = 4'd9;  b = 4'd1;
end

endmodule
```

Testbench of BCD Subtractor

```
amal  Aspire-A715-42G  ../Digital/bcd_subtractor  vvp subtractor
a = 1001        b = 1001        signed binary representation of sum =0000_0000
a = 0001        b = 1001        signed binary representation of sum =1000_1000
a = 1001        b = 0001        signed binary representation of sum =0000_1000
```

Test of BCD Subtractor



Waveform of BCD Subtractor

Explanation

•Implemented a BCD Subtractor using 2 one bit parallel adders in verilog.Here, 2's complement subtraction is carried out when the mode is specified.The second number (b) is sent through a xor gate with cin' when Cin=1, which aids in determining the complement of the second number.
•Added a comparator to show the sign when a is less than b.

3.Logic Implementation with Multiplexer

testbench.sv

SV/Verilog Testbench

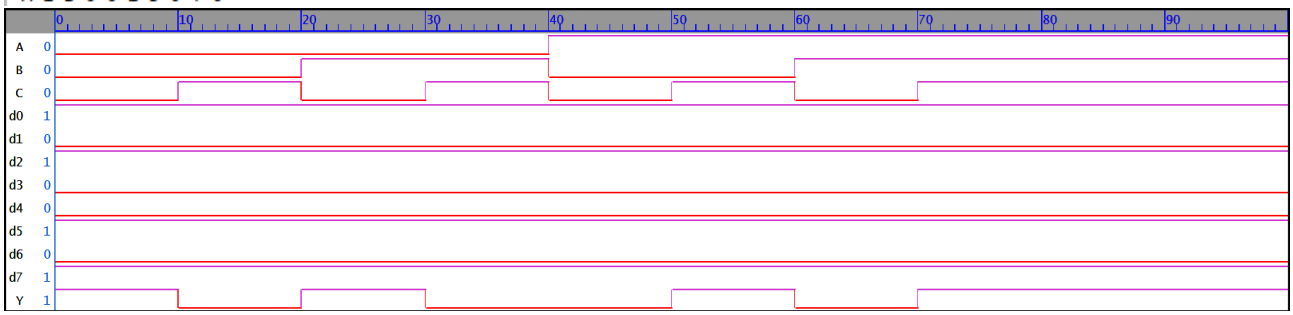
```
module mux_tb();
  reg A,B,C,D,d0,d1,d2,d3,d4,d5,d6,d7;
  wire Y;
  integer i;
  mux_8to1 dut(A,B,C,d0,d1,d2,d3,d4,d5,d6,d7,Y);
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
    #100 $finish;
  end
  initial begin
    A=0;B=0;C=0;D=0;
    $monitor("A=%0b B=%0b C=%0b D=%0b Y=%0b",A,B,C,D,Y);
    for(i=0;i<16;i=i+1)
    begin
      {A,B,C,D}=i;
      d0=D;d1=0;d2=~D;d3=0;d4=0;d5=0;d6=~D;d7=~D;
      #10;
   end
  end
endmodule
```

design.sv

```
//design
module mux_8to1(
  input A,B,C,d0,d1,d2,d3,d4,d5,d6,d7,
  output Y);
  wire y1,y2,y3,y4,y5,y6,y7,y8;
  and(y1,~A,~B,~C,d0);
  and(y2,~A,~B,C,d1);
  and(y3,~A,B,~C,d2);
  and(y4,~A,B,C,d3);
  and(y5,A,~B,~C,d4);
  and(y6,A,~B,C,d5);
  and(y7,A,B,~C,d6);
  and(y8,A,B,C,d7);
  or(Y,y1,y2,y3,y4,y5,y6,y7,y8);
endmodule
```

Log | Share

```
A=0 B=0 C=0 D=0 Y=0
A=0 B=0 C=0 D=1 Y=1
A=0 B=0 C=1 D=0 Y=0
A=0 B=0 C=1 D=1 Y=0
A=0 B=1 C=0 D=0 Y=1
A=0 B=1 C=0 D=1 Y=0
A=0 B=1 C=1 D=0 Y=0
A=0 B=1 C=1 D=1 Y=0
A=1 B=0 C=0 D=0 Y=0
A=1 B=0 C=0 D=1 Y=0
A=1 B=0 C=1 D=0 Y=0
```



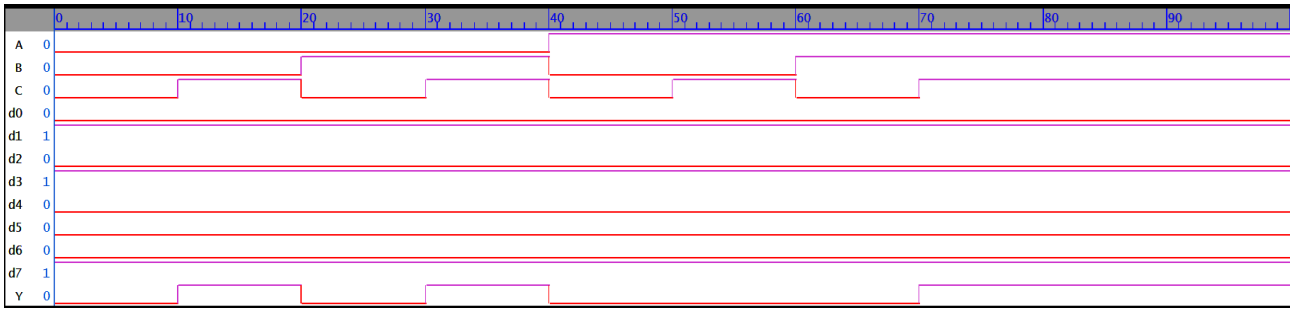
Design,Testbench,Test and Waveforms of 8 : 1 multiplexer

```
module mux_tb1();
  reg A,B,C,d0,d1,d2,d3,d4,d5,d6,d7;
  wire Y;
  integer i;
  mux_8to1 dut(A,B,C,d0,d1,d2,d3,d4,d5,d6,d7,Y);
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
    #100 $finish;
  end
  initial begin
    A=0;B=0;C=0;
    d0=0;d1=1;d2=0;d3=1;d4=0;d5=0;d6=0;d7=1;
    $monitor("A=%0b B=%0b C=%0b Y=%0b",A,B,C,Y);
    for(i=0;i<8;i=i+1)
    begin
      {A,B,C}=i;
      #10;
   end
  end
```

```
//design
 module mux_8to1(
  input A,B,C,d0,d1,d2,d3,d4,d5,d6,d7,
  output Y);
  wire y1,y2,y3,y4,y5,y6,y7,y8;
  and(y1,~A,~B,~C,d0);
  and(y2,~A,~B,C,d1);
  and(y3,~A,B,~C,d2);
  and(y4,~A,B,C,d3);
  and(y5,A,~B,~C,d4);
  and(y6,A,~B,C,d5);
  and(y7,A,B,~C,d6);
  and(y8,A,B,C,d7);
  or(Y,y1,y2,y3,y4,y5,y6,y7,y8);
endmodule
```

Log Share

```
[2023-12-19 11:25:35 UTC] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile dump.vcd opened for output.
A=0 B=0 C=0 Y=0
A=0 B=0 C=1 Y=1
A=0 B=1 C=0 Y=0
A=0 B=1 C=1 Y=1
A=1 B=0 C=0 Y=0
A=1 B=0 C=1 Y=0
A=1 B=1 C=0 Y=0
A=1 B=1 C=1 Y=1
Done
```



Design,Testbench,Test and Waveforms

testbench.sv

SV/Verilog Testbench

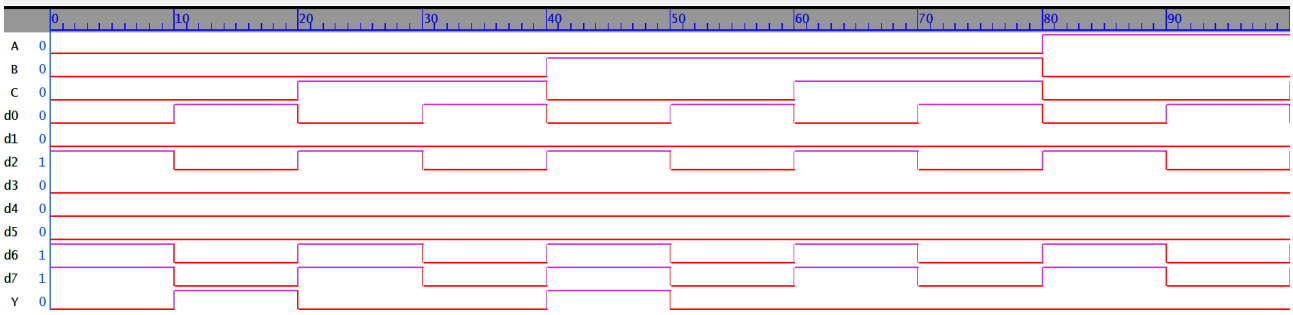
```
module mux_tb();
  reg A,B,C,D,d0,d1,d2,d3,d4,d5,d6,d7;
  wire Y;
  integer i;
  mux_8to1 dut(A,B,C,d0,d1,d2,d3,d4,d5,d6,d7,Y);
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
    #100 $finish;
  end
  initial begin
    A=0;B=0;C=0;D=0;
    $monitor("A=%0b B=%0b C=%0b D=%0b Y=%0b",A,B,C,D,Y);
    for(i=0;i<16;i=i+1)
    begin
      {A,B,C,D}=i;
      d0=D;d1=0;d2=~D;d3=0;d4=0;d5=0;d6=~D;d7=~D;
      #10;
   end
  end
endmodule
```

design.sv

```
//design
module mux_8to1(
  input A,B,C,d0,d1,d2,d3,d4,d5,d6,d7,
  output Y);
  wire y1,y2,y3,y4,y5,y6,y7,y8;
  and(y1,~A,~B,~C,d0);
  and(y2,~A,~B,C,d1);
  and(y3,~A,B,~C,d2);
  and(y4,~A,B,C,d3);
  and(y5,A,~B,~C,d4);
  and(y6,A,~B,C,d5);
  and(y7,A,B,~C,d6);
  and(y8,A,B,C,d7);
  or(Y,y1,y2,y3,y4,y5,y6,y7,y8);
endmodule
```

Log | Share

```
A=0 B=0 C=0 D=0 Y=0
A=0 B=0 C=0 D=1 Y=1
A=0 B=0 C=1 D=0 Y=0
A=0 B=0 C=1 D=1 Y=0
A=0 B=1 C=0 D=0 Y=1
A=0 B=1 C=0 D=1 Y=0
A=0 B=1 C=1 D=0 Y=0
A=0 B=1 C=1 D=1 Y=0
A=1 B=0 C=0 D=0 Y=0
A=1 B=0 C=0 D=1 Y=0
A=1 B=0 C=1 D=0 Y=0
```



Design,Testbench,Test and Waveforms

Explanation

•Implemented a 8:1 multiplexer using gates.An 8:1 multiplexer is a type of data selector that comprises one output line, three select lines, and eight input lines.Here, we assess two provided SOP expressions using the 8:1 mux module as a subcircuit.

•Used the above circuit to create a subcircuit and implemented the logic functions.

4. BCD to seven segment decoder

```
module seven_seg_disp(
  input A,B,C,D,
  output a,b,c,d,e,f,g);
  and(notbd,~B,~D);
  and(bd,B,D);
  and(notcd,~C,~D);
  and(cd,C,D);
  and(bnotc,~B,C);
  and(bcnotd,B,~C,D);
  and(cdnot,C,~D);
  and(bcnot,B,~C);
  and(bdnot,B,~D);
  or(a,notbd,C,bd,A);
  or(b,~B,notcd,cd);
  or(c,~C,D,B);
  or(d,notbd,bnotc,bcnotd,cdnot,A);
  or(e,notbd,cdnot);
  or(f,notcd,bcnot,bdnot,A);
  or(g,bnotc,bcnot,A,bdnot);
endmodule
```

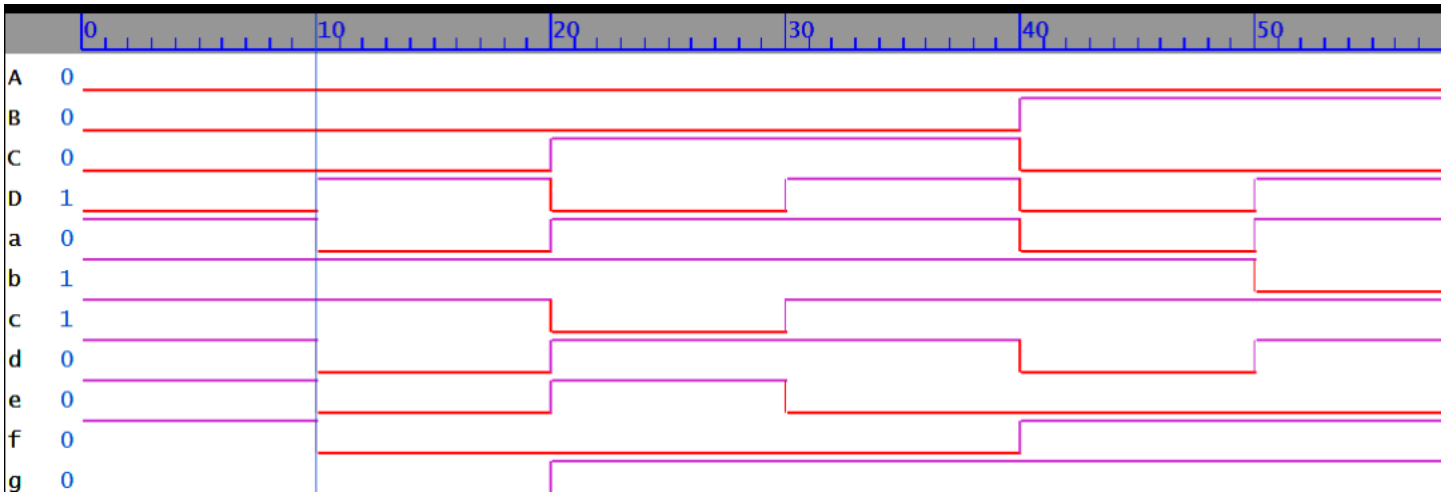
Design of BCD to seven segment decoder

```
module decode_tb;
  reg A,B,C,D;
  wire a,b,c,d,e,f,g;
  integer i;
  seven_seg_disp dut(A,B,C,D,a,b,c,d,e,f,g);
  initial begin
    $dumpfile("dump.vcd");      //waveform
    $dumpvars;
    #100 $finish;
  end
  initial begin
    A=0;B=0;C=0;D=0;
    $monitor("%0b%0b%0b%0b a=%0b b=%0b c=%0b d=%0b e=%0b f=%0b g=%0b",A,B,C,D,a,b,c,d,e,f,g);
    for(i=0;i<10;i=i+1)
      begin
        {A,B,C,D}=i;
        #10;
      end
  end
endmodule
```

Testbench of BCD to seven segment decoder

```
VCD info: dumpfile dump.vcd opened for output.
0000 a=1 b=1 c=1 d=1 e=1 f=1 g=0
0001 a=0 b=1 c=1 d=0 e=0 f=0 g=0
0010 a=1 b=1 c=0 d=1 e=1 f=0 g=1
0011 a=1 b=1 c=1 d=1 e=0 f=0 g=1
0100 a=0 b=1 c=1 d=0 e=0 f=1 g=1
0101 a=1 b=0 c=1 d=1 e=0 f=1 g=1
0110 a=1 b=0 c=1 d=1 e=1 f=1 g=1
0111 a=1 b=1 c=1 d=0 e=0 f=0 g=0
1000 a=1 b=1 c=1 d=1 e=1 f=1 g=1
1001 a=1 b=1 c=1 d=1 e=0 f=1 g=1
```

Test of BCD to seven segment decoder

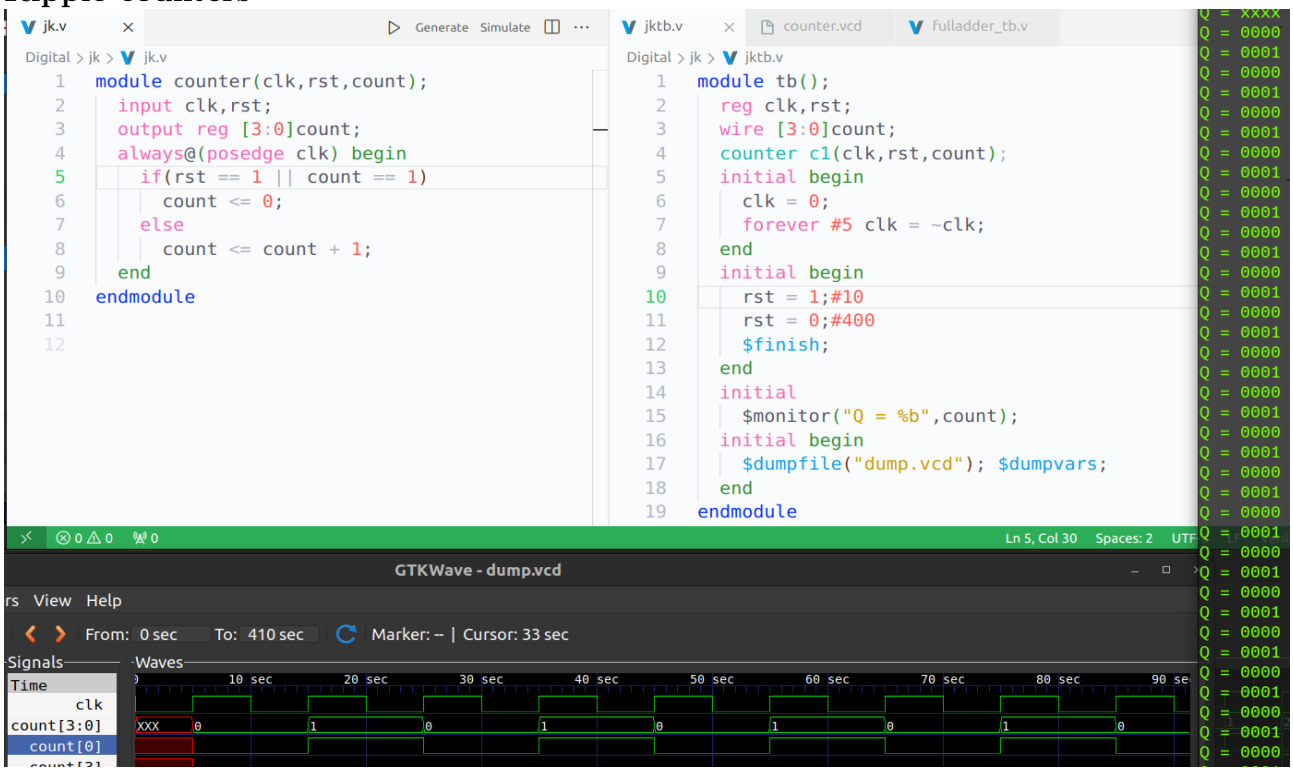


Explanation

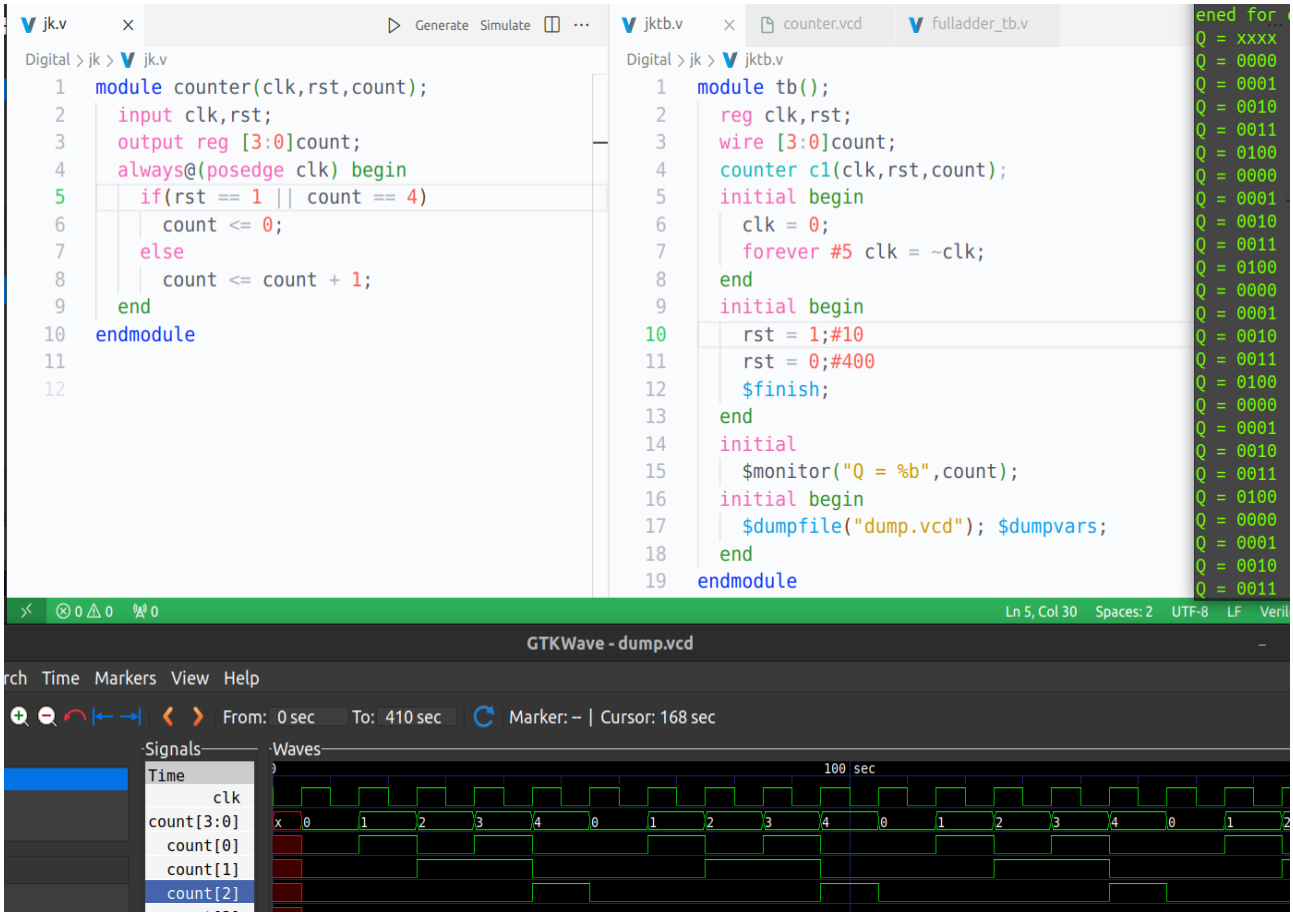
• Implemented a 4 BCD to seven segment decoder .

•The inputs (A, B, C, and D) is received by the seven-segment decoder, which has four input lines and seven output lines (a, b, c, d, e, f, and g). A seven-segment LED display receives the output and, based on the inputs, displays the decimal number.

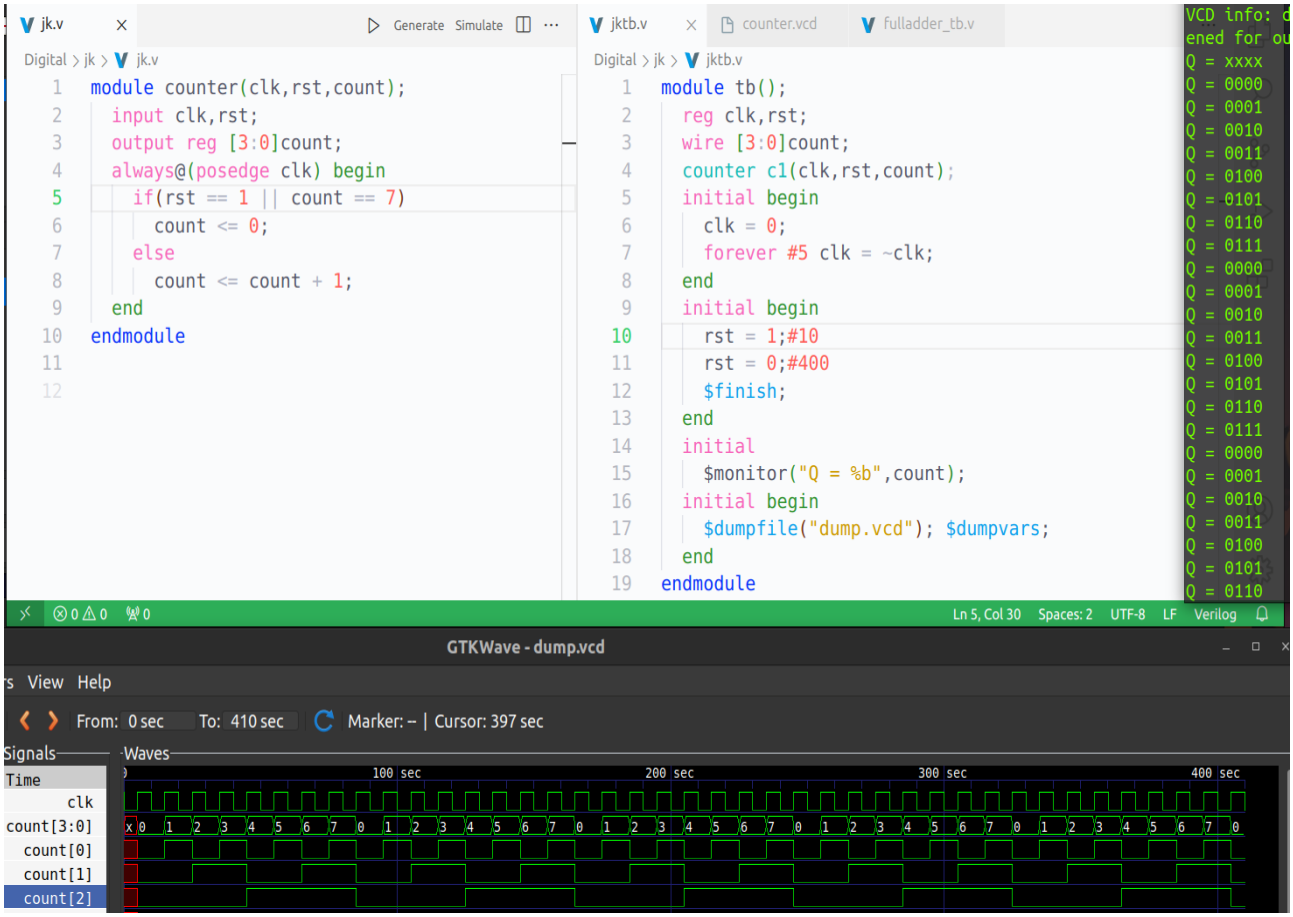
3.Implementing functionality of MOD 2, MOD 5, MOD 8, MOD 10 and MOD 40 Ripple counters



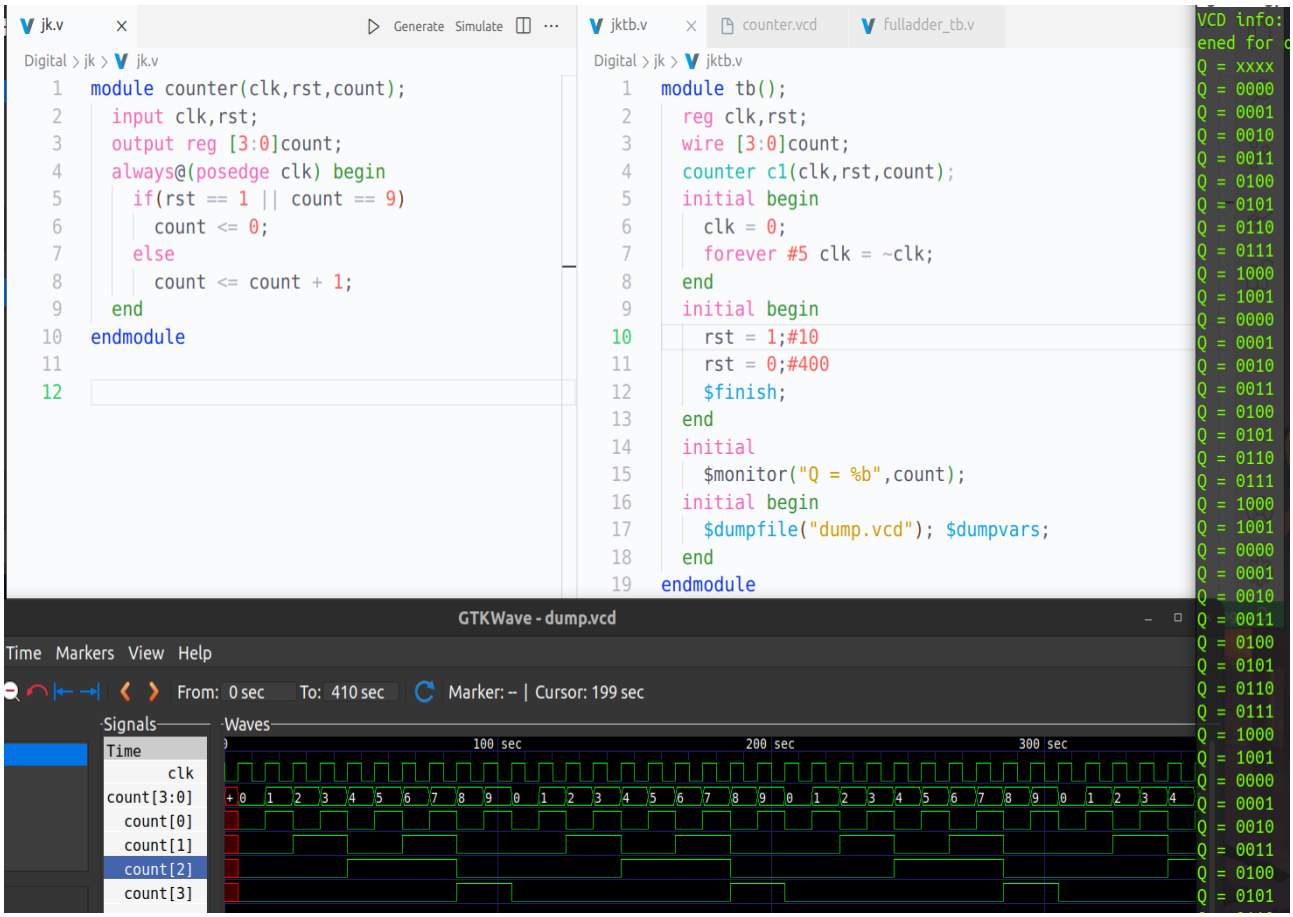
Design,Testbench,Test and Waveforms



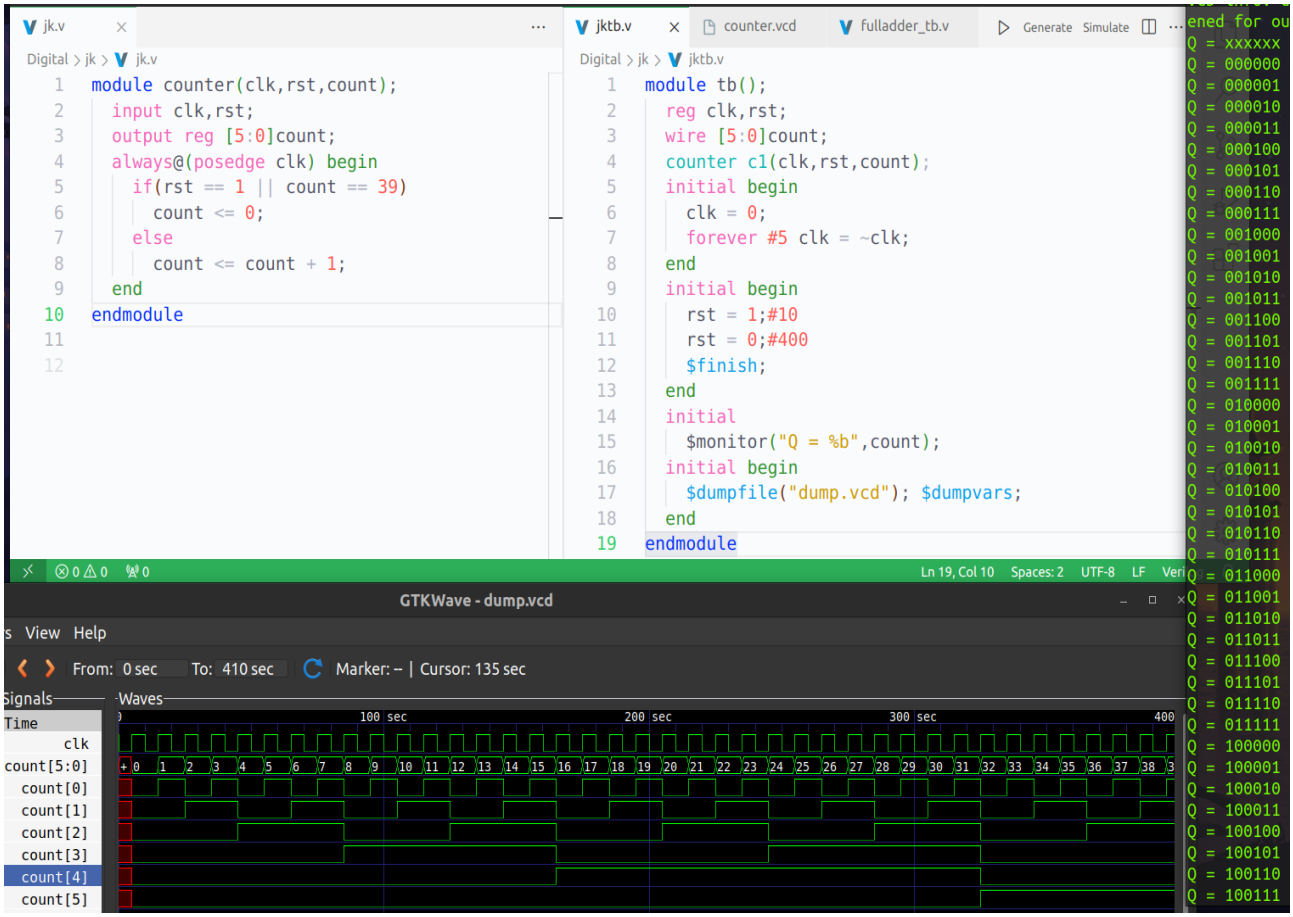
Design,Testbench,Test and Waveforms



Design,Testbench,Test and Waveforms



Design,Testbench,Test and Waveforms



Design,Testbench,Test and Waveforms

Explanation

•Implemented functionality of Ripple counters by understanding the state and timing diagrams.

•A n-bit ripple counter can count up to $(2^n) states$.

•It is also known as MOD n counter. It is known as ripple counter because of the way the clock pulse ripples its way through the flip-flops.Asynchronous counters are also called ripple-counters because of the way the clock pulse ripples its way through the flip-flops

6 Realisation of 4 Bit counter

Digital > jk > jk.v

```
module counter4bit(

input   clk  ,r   ,
output  [3:0]Q_OUT

);

wire clk          ;
wire j3,j2,j1,j0  ;
wire [3:0]Q_OUT   ;
wire r    ,y      ;

jk_flipflop a0(  .j(j0),  .k(j0),  .clk(clk), .reset(r), .Q(Q_OUT[0]) , .Q_bar(y)   );
jk_flipflop a1(  .j(j1),  .k(j1),  .clk(clk), .reset(r), .Q(Q_OUT[1]) , .Q_bar(y)   );
jk_flipflop a2(  .j(j2),  .k(j2),  .clk(clk), .reset(r), .Q(Q_OUT[2]) , .Q_bar(y)   );
jk_flipflop a3(  .j(j3),  .k(j3),  .clk(clk), .reset(r), .Q(Q_OUT[3]) , .Q_bar(y)   );

assign     j0 = 1                                  ;
assign     j1 = Q_OUT[0]                           ;
assign     j2 = Q_OUT[0] & Q_OUT[1]                ;
assign     j3 = Q_OUT[2] & Q_OUT[1] & Q_OUT[0]     ;
endmodule


module jk_flipflop(
input j,k,clk,reset,
output Q,Q_bar
);
wire        j        ;
wire        k        ;
wire        clk      ;
wire        reset    ;
reg         Q;
reg          Q_bar;

```

Digital > jk > jk.v

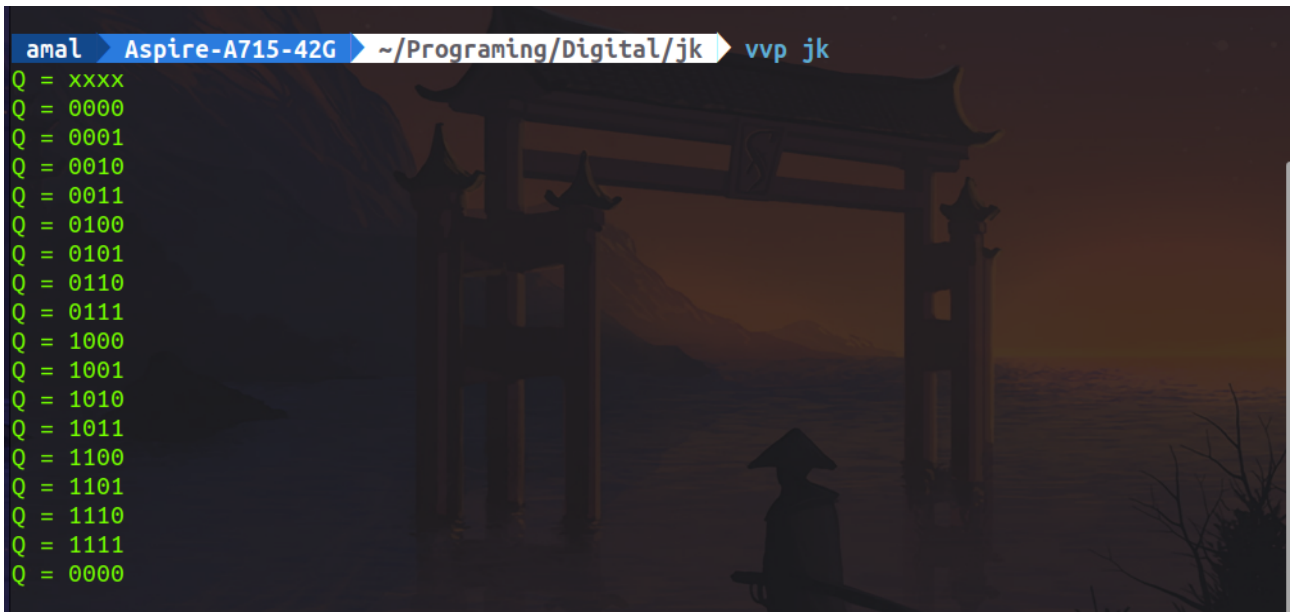
```
    always@(posedge clk)
        begin

          if({reset})
          {Q,Q_bar}<={1'b0,1'b1};

          else
              begin
              case({j,k})
              2'b00:{Q,Q_bar}<={Q,Q_bar};
              2'b01:{Q,Q_bar}<={1'b0,1'b1};
              2'b10:{Q,Q_bar}<={1'b1,1'b0};
              2'b11:{Q,Q_bar}<={~Q,Q};
              default:begin end
              endcase
              end
        end
endmodule

```

Design of 4 Bit counter

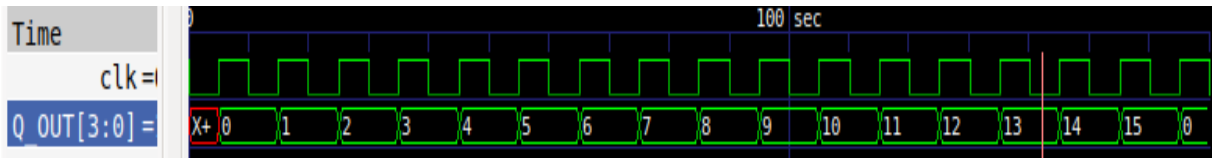
Digital > jk > jktb.v

```
// Test bench
module counter4bittb;

reg clk,rst;
wire [3:0]Q_OUT;

  counter4bit uut(

                .clk     (     clk     ),
                .Q_OUT   (     Q_OUT   ),
                .r       (     rst     )
  );

  initial begin
  clk=0;
  forever #5 clk=~clk;
  end

  initial
    begin
    $monitor("Q = %b  ",Q_OUT);
    $dumpfile("dp.vcd");$dumpvars;
     rst=1;#10
     rst=0;#160

     $finish;
    end
endmodule
```

Testbench of 4 Bit counter



Test of 4 Bit counter



Waveform of of 4 Bit counter

Explanation

•Implemented a 4 Bit counter(mod 16) using 4 jk flip flops.
•Design was obtained by drawing the state diagram, filling the table, drawing Karnaugh map for each FF input in terms of flip-flop output and then the design was carefully implemented in verilog.