Verilog Manual

PART B (Using Xilinx Vivado tool)

5. Write a Verilog code and implement Adders & Subtractors.
6. Design Comparators.
7. Realize SR, JK, T & D Flip-Flops.
8. Design Counters & Shift Registers.
9. Design Clock Pulse Generator.
10. Design and implement 4-bit ALU using Verilog program.

Write a Verilog code and implement Adders & Subtractors.

Aim:

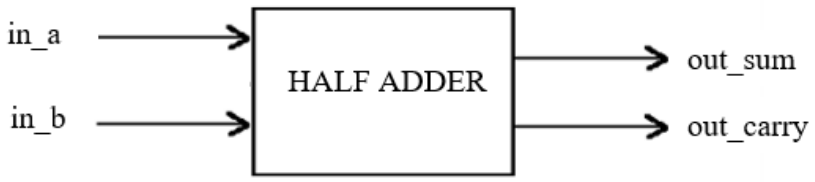
1. Model in Verilog for a 1-bit half adder.
2. Model in Verilog for a full adder and add functionality to perform logical operations of XOR, XNOR, AND, and OR gates. Write a test bench with appropriate input patterns to verify the modeled behaviour.

Software Required:

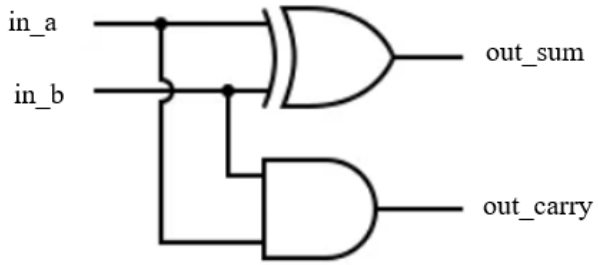
Xilinx Vivado

1-bit Half Adder

Block Diagram:



Logic Diagram:



Truth table:

INPUT		OUTPUT	
a	**b**	**sum**	**carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

BOOLEAN EXPRESSIONS:

out_sum = a ⊕ b

out_carry = a . b

CODE:

Dataflow:

```
module half_adder(sum,carry,a,b);
        input a,b;
        output sum,carry;
         assign sum=a^b;
         assign carry= a & b;
endmodule
```

Testbench Code

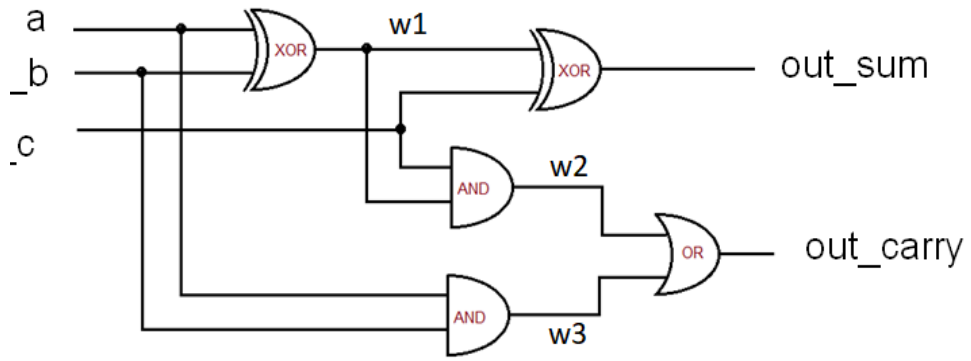
```
module half_adder_tb();
reg a,b;
wire sum,carry;
half_adder U2(sum,carry,a,b);
initial
```

```
begin
    a=1'b0; b = 1'b0;
#5 a=1'b0; b = 1'b1;
#5 a=1'b1; b = 1'b0;
#5 a=1'b1; b = 1'b1;
#5
$finish;
end
endmodule
```

Full Adder

Block Diagram:

Logic Diagram:



Truth table:

INPUT			OUTPUT	
in	**in**	**in**	**sum**	**carry**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

BOOLEAN EXPRESSIONS:

sum = a ⊕ b ⊕ c

carry = ab+bc+ca

CODE:

Dataflow:

```
module full_adder (out_sum, out_carry, in_a, in_b, in_c);
module full_adder(sum,carry,a,b,c);
input a,b,c;
output sum,carry;
assign sum=a^b^c;
assign carry=(a & b)|(a & c)|(b & c);
endmodule
```

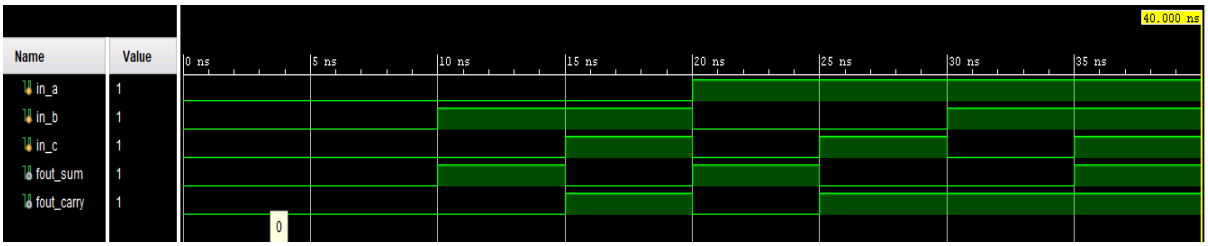
Testbench Code

```
module full_adder_tb();
reg a,b,c;
wire sum,carry;
full_adder U5(sum,carry,a,b,c);
initial
begin
a=1'b0;
b = 1'b0;
c =1'b0;
#5 c =1'b1;
#5 b=1'b1; c=1'b0;

#5 c=1'b1;
```

```
#5 a=1'b1;b=1'b0;c=1'b0;
#5 c=1'b1;
#5 b=1'b1; c=1'b0;
#5 c=1'b1;
#5
$finish;
end
endmodule
```

EXPECTED WAVEFORM:



Realize SR, JK, T & D Flip-Flops.

Aim:

Simulation and realization of following Flip Flop

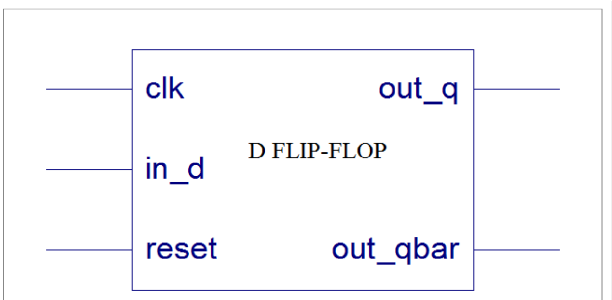
A. D Flip Flop
B. JK Flip Flop
C. SR Flip Flop

Software Required:

Xilinx Vivado

A. D Flip Flop

Block Diagram:



Truth table:

INPUT			OUTPUT	
clk	**reset**	**in_d**	**out_q**	**out_qbar**
↑	0	0	0	1
↑	0	1	1	0
↑	1	X	0	1
↓	X	X	No change	

CODE:

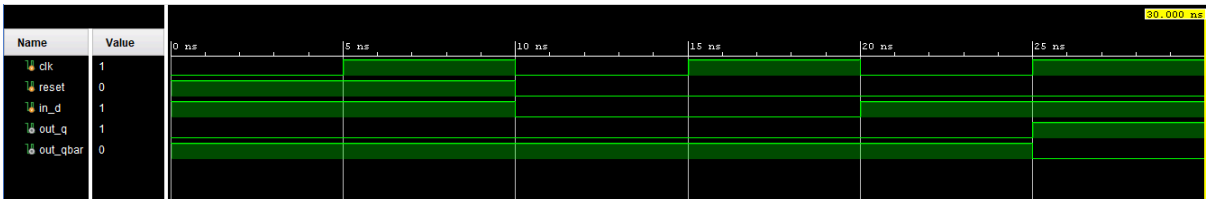
Behavioural:

```
module d_ff (clk, reset, in_d, out_q, out_qbar);
   input clk, reset, in_d;
   output out_q, out_qbar;
   reg out_q;
   always @ (posedge clk or posedge reset)
begin
        if(reset)
        out_q <= 1'b0;
        else
        out_q <= in_d;
  end
  assign out_qbar = ~out_q;
endmodule
```

Testbench code:

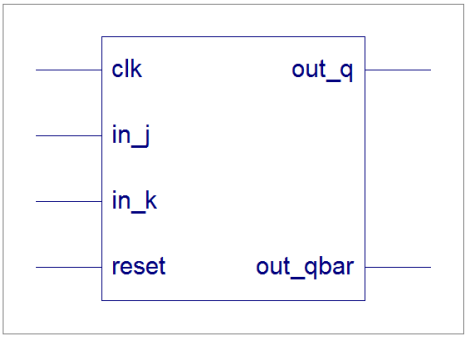
```
module d_fftest;
   reg clk;
   reg reset;
   reg in_d;
   wire out_q;
   wire out_qbar;
   d_ff d1 (.clk(clk), .reset(reset),.in_d(in_d),.out_q(out_q),.out_qbar(out_qbar));
   initial
   begin
      clk = 1'b0;
      forever
      #5 clk=~clk;
   end
initial
   begin
 $display("Time | clk | reset | in_d |out_q | out_qbar ");
$monitor("%4d | %3b | %5b | %3b| %6b| %7b|",
$time, clk, reset, in_d, out_q,out_qbar);
      reset = 1'b1; in_d = 1’b1;
      #10;
      reset =1'b0;
      in_d=1'b0;
      #10; in_d= 1'b1;
      #10; $finish;
  end
endmodule
```

EXPECTED WAVEFORM:



B. JK Flip Flop

Block Diagram:



Truth table:

INPUT				OUTPUT	
clk	**reset**	**in_j**	**in_k**	**out_q**	**Out_qbar**
↑	0	0	0	(No change)	
↑	0	0	1	0	1
↑	0	1	0	1	0
↑	0	1	1	Toggles	
↑	1	X	X	0	1

CODE:

Behavioural:

```
module jk_ff (clk, reset, in_j, in_k, out_q, out_qbar);
  input clk, reset, in_j, in_k;
  output out_q, out_qbar;
  reg out_q;
always @(posedge clk or posedge reset)
  begin
    if(reset)
      out_q<=1'b0;
else
    begin
      case ({in_j, in_k})
        2'b00: out_q<=out_q;
        2'b01: out_q<=1'b0;
        2'b10: out_q<=1'b1;
        2'b11: out_q<=~out_q;
        default: out_q<=out_q;
      endcase
    end
  end
  assign out_qbar=~out_q;
endmodule
```

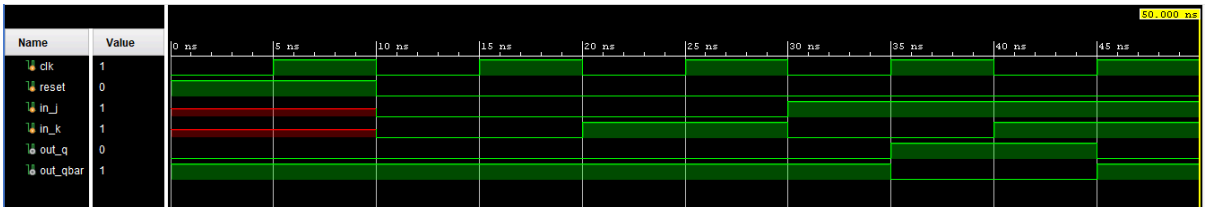
Testbench code:

```
module jk_ff_tb;
  reg clk;
```

```
   reg reset;
   reg in_j;
   reg in_k;
   wire out_q;
   wire out_qbar;
  jk_ff jk1(.clk(clk), .reset(reset), .in_j(in_j), .in_k(in_k), .out_q(out_q),
.out_qbar(out_qbar));
initial
begin
   clk=1'b0;
   forever
   #5 clk= ~clk;
end
initial
begin
$display("Time | clk | reset | in_j | in_k| out_q | out_qbar ");
$monitor("%4d | %3b | %5b | %3b | %3b| %6b| %7b|",
$time, clk, reset, in_j, in_k, out_q,out_qbar);
 reset=1'b1;
#10;
reset=1'b0;
   in_j=1'b0;
   in_k=1'b0;
   #10;
   in_k=1'b1;
   #10;
   in_j=1'b1;
   in_k=1'b0;
   #10;
   in_k=1'b1;
```

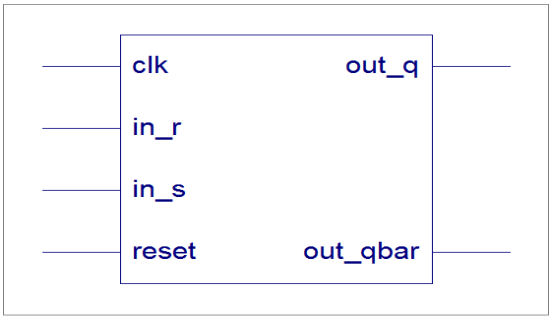
```
    #10;
$finish;
end
endmodule
```

EXPECTED WAVEFORM:



C. SR Flip Flop

Block Diagram:



Truth table:

INPUT				OUTPUT	
clk	**reset**	**in_s**	**in_r**	**out_q**	**Out_qbar**
clk	0	0	0	(No change)	
↑	0	0	1	0	1
↑	0	1	0	1	0
↑	0	1	1	X (Invalid)	
↑	1	X	X	0	1
↑					

CODE:

Behavioural:

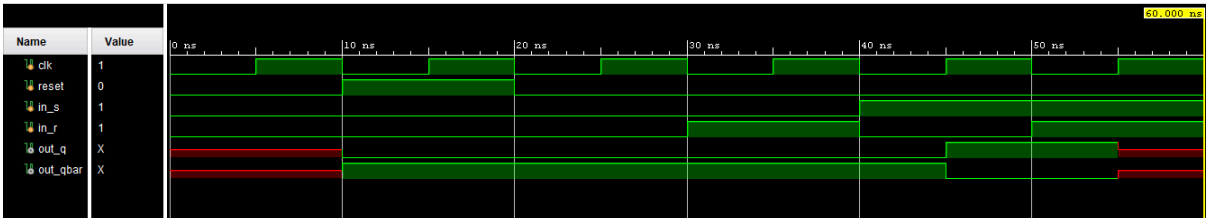
```
module sr_ff (clk,reset,in_s,in_r, out_q,out_qbar);
input clk, reset, in_s, in_r;
output out_q, out_qbar;
reg out_q;
always @(posedge clk or posedge reset)
begin
if(reset)
out_q<=1'b0;
else
begin
case({in_s, in_r})
2'b00:out_q<=out_q;
2'b01:out_q<=1'b0;
2'b10:out_q<=1'b1;
2'b11:out_q<=1'bx;
default:out_q<=out_q;
endcase
end
end
assign out_qbar = ~out_q;
endmodule
```

Testbench code:

```
module sr_ff_tb();
reg clk, reset, in_s, in_r;
wire out_q, out_qbar;
```

```
sr_ff sr1 (.clk(clk), .reset(reset), .in_s(in_s), .in_r(in_r), .out_q(out_q),
.out_qbar(out_qbar));
initial
begin
clk=1'b0;
forever
#5 clk= ~clk;
end
initial
begin
$display("Time | clk | reset | in_s | in_r | out_q | out_qbar ");
$monitor("%4d | %3d | %5d | %b | %b| %b| %b|",
$time,clk, reset, in_s, in_r, out_q,out_qbar);
reset=1'b0;
in_s=1'b0;
in_r=1'b0;
#10;
reset=1'b1;
#10;
reset=1'b0;
#10;
in_r=1'b1;
#10;
in_s=1'b1;
in_r=1'b0;
#10;
in_r=1'b1;
#10 $finish;
end
endmodule
```

EXPECTED WAVEFORM:



Design and implement 4-bit ALU using Verilog program.

Aim:

Simulation and realization of 4-bit ALU

Software Required:

Xilinx Vivado

Block Diagram:

Basic Operation based on 3-bit opcode

Op-code	ALU Operation
000	A + B
001	A – B
010	Increment by 1

011	Decrement by 1
100	True
101	Ones complement
110	AND operation
111	OR operation

CODE:

Dataflow:

```
module alu4 (in_a, in_b, c_opcode, o_result);
input [3:0] in_a, in_b;
input [2:0] c_opcode;
output [3:0] o_result;
reg [3:0] o_result;
always@(in_a, in_b, c_opcode)
begin
case (c_opcode)
  3'b000: o_result = in_a + in_b;
  3'b001: o_result = in_a - in_b;
  3'b010: o_result = in_a + 1;
  3'b011: o_result = in_a - 1;
  3'b100: o_result = in_a?1:0; // true
  3'b101: o_result = ~in_a; // Complement
  3'b110: o_result = in_a & in_a; //AND Bitwise
  3'b111: o_result = in_a | in_a; // OR Bitwise
default : o_result = 32'bx;
endcase
end
```

```
endmodule
```

Test bench code:

```
module alu4_tb;
 reg [3:0] in_a, in_b;
 reg [2:0] c_opcode;
 wire [3:0] o_result; integer i;
 alu32 U0 (in_a, in_b, c_opcode, o_result);
 initial
 begin
$display("|Time| in_a| in_b |   c_opcode | o_result|");
$monitor("%4d | %4d | %4d | %3d | %8d",$time,in_a,in_b,c_opcode,o_result);
in_a = 10;
in_b = 4;
c_opcode = 0;
for (i = 0; i < 8; i = i+1)
#5 c_opcode = i;
#5
$finish;
end
endmodule
```

EXPECTED WAVEFORM:

OUTPUT:

Design Counters & Shift Registers.

Aim:

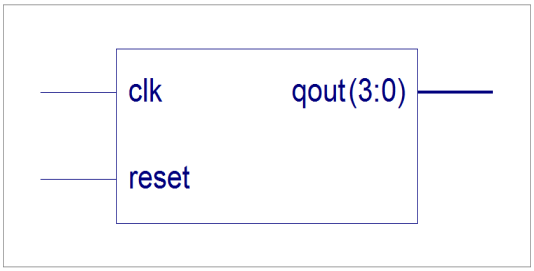
Simulation and realization of 4-bit binary and BCD synchronous counters

Software Required:

Xilinx Vivado

A. 4-bit Synchronous Binary Counter:

Block Diagram:



Truth table:

Clk	Reset	Present State				Next State			
		qout3	qout2	qout1	qout0	qout3	qout2	qout1	qout0
↑	0	0	0	0	0	0	0	0	1
↑	0	0	0	0	1	0	0	1	0
↑	0	0	0	1	0	0	0	1	1
↑	0	0	0	1	1	0	1	0	0
↑	0	0	1	0	0	0	1	0	1

Up Counter:

CODE:

Behavioural:

```
module binary_sync_up (clk, reset, qout);
  input clk;
  input reset;
  output [3:0] qout;
  reg [3:0] qout;
always @ (posedge clk)
begin
  if(reset)
    qout <= 4'b0000;
  else
    qout <= qout+1;
  end
endmodule
```

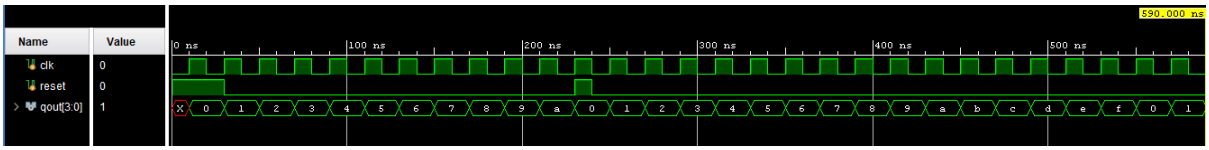
Testbench code:

```
module binary_Sync_up_tb();
  reg clk;
  reg reset;
  wire [3:0] qout;
binary_sync_up u0 (.clk(clk),.reset(reset),.qout(qout));
initial
begin
  clk=1'b0;
  forever
```

```
   #10 clk=~clk;
end

initial
begin
   reset=1'b1;
   #30;
   reset=1'b0;
   #200;
     reset=1'b1;
    #10;
      reset=1'b0;
#350;
   $finish;
end
endmodule
```

EXPECTED WAVEFORM:



Down Counter:

CODE:

Behavioural:

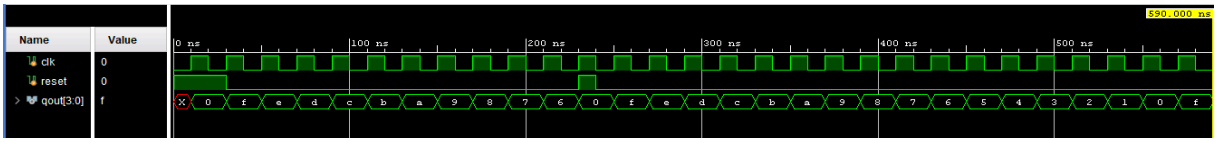
```
module binary_sync_down (clk, reset, qout);
   input clk;
   input reset;
   output [3:0] qout;
   reg [3:0] qout;
```

```
always @ (posedge clk)
begin
   if(reset)
      qout <= 4'b0000;
   else if
  (qout == 4'b0000)
  qout = 4'b1111;
else
      qout <= qout-1;
   end
endmodule
```

Testbench code:

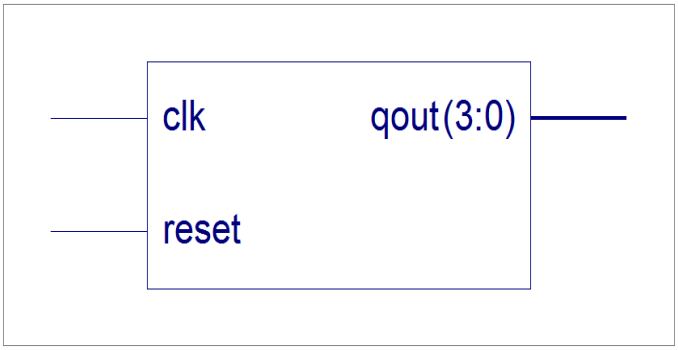
```
module binary_Sync_down_tb();
   reg clk;
   reg reset;
   wire [3:0] qout;
binary_sync_down u0 (.clk(clk),.reset(reset),.qout(qout));
initial
begin
   clk=1'b0;
   forever
   #10 clk=~clk;
end
initial
begin
   reset=1'b1;
   #30;
   reset=1'b0;
   #200;
     reset=1'b1;
    #10;
      reset=1'b0;
#350;
   $finish;
end
endmodule
```

EXPECTED WAVEFORM:



B. 4-bit BCD Synchronous Counter

Block Diagram:



Truth table:

Clk	Reset	Present State				Next State			
		qout3	qout2	qout1	qout0	qout3	qout2	qout1	qout0
↑	0	0	0	0	0	0	0	0	1
↑	0	0	0	0	1	0	0	1	0
↑	0	0	0	1	0	0	0	1	1
↑	0	0	0	1	1	0	1	0	0
↑	0	0	1	0	0	0	1	0	1
↑	0	0	1	0	1	0	1	1	0
↑	0	0	1	1	0	0	1	1	1
↑	0	0	1	1	1	1	0	0	0
↑	0	1	0	0	0	1	0	0	1
↑	0	1	0	0	1	0	0	1	0

Up Counter:

CODE:

Behavioural:

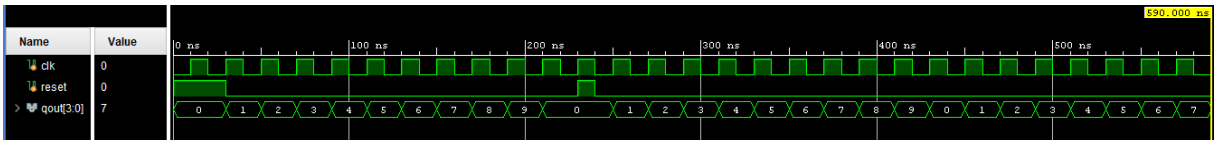
```
module BCDsyn_up (reset, clk, qout);
input clk;
input reset;
output [3:0] qout;
reg [3:0] qout=4'b0000;
always@(posedge clk)
begin
if(reset==1)
qout =4'b0000;
else if (qout>=4'b1001)
qout =4'b0000;
else
qout = qout+1;
end
endmodule
```

Testbench code:

```
module BCD_Sync_up_tb( );
   reg clk;
   reg reset;
   wire [3:0] qout;
BCDsyn_up u0 (.clk(clk),.reset(reset),.qout(qout));
initial
begin
   clk=1'b0;
   forever
   #10 clk=~clk;
```

```
end
initial
begin
   reset=1'b1;
   #30;
   reset=1'b0;
   #200;
     reset=1'b1;
    #10;
      reset=1'b0;
#350;
   $finish;
end
endmodule
```

EXPECTED WAVEFORM:



Down Counter:

CODE:

Behavioural:

```
module BCDsyn_down (reset, clk, qout);
input clk;
input reset;
output[3:0] qout;
reg [3:0]qout=4'b0000;
always@(posedge clk)
```

```
begin
 if(reset==1)
qout=4'b0000;
else if (qout<=4'b0000)
qout =4'b1001;
else
qout = qout-1;
end
endmodule
```

Testbench code:

```
module BCD_sync_down();
   reg clk;
   reg reset;
   wire [3:0] qout;
BCDsyn_down  u0 (.clk(clk),.reset(reset),.qout(qout));
initial
begin
   clk=1'b0;
   forever
   #10 clk=~clk;
end
initial
begin
reset=1'b1;
   #30;

reset=1'b0;
   #200;
     reset=1'b1;
    #10;
```

```
      reset=1'b0;
#350;
   $finish;
end
endmodule
```

EXPECTED WAVEFORM: