```
module alu(
        input [7:0] A,B,  // ALU 8-bit Inputs
        input [3:0] ALU_Sel,// ALU Selection
        output [7:0] ALU_Out, // ALU 8-bit Output
        output CarryOut // Carry Out Flag
    );
    reg [7:0] ALU_Result;
    wire [8:0] tmp;
    assign ALU_Out = ALU_Result; // ALU out
    assign tmp = {1'b0,A} + {1'b0,B};
    assign CarryOut = tmp[8]; // Carryout flag
    always @(*)
    begin
      case(ALU_Sel)
      4'b0000: // Addition
        ALU_Result = A + B ;
      4'b0001: // Subtraction
        ALU_Result = A - B ;
      4'b0010: // Multiplication
        ALU_Result = A * B;
      4'b0011: // Division
        ALU_Result = A/B;
      4'b0100: // Logical shift left
        ALU_Result = A<<1;
       4'b0101: // Logical shift right
        ALU_Result = A>>1;
       4'b0110: // Rotate left
        ALU_Result = {A[6:0],A[7]};
       4'b0111: // Rotate right
        ALU_Result = {A[0],A[7:1]};
        4'b1000: //  Logical and
```

```
        ALU_Result = A & B;
       4'b1001: //  Logical or
        ALU_Result = A | B;
       4'b1010: //  Logical xor
        ALU_Result = A ^ B;
       4'b1011: //  Logical nor
        ALU_Result = ~(A | B);
       4'b1100: // Logical nand
        ALU_Result = ~(A & B);
       4'b1101: // Logical xnor
        ALU_Result = ~(A ^ B);
       4'b1110: // Greater comparison
        ALU_Result = (A>B)?8'd1:8'd0 ;
       4'b1111: // Equal comparison
         ALU_Result = (A==B)?8'd1:8'd0 ;
       default: ALU_Result = A + B ;
      endcase
  end

endmodule
`timescale 1ns / 1ps

module tb_alu;
//Inputs
 reg[7:0] A,B;
 reg[3:0] ALU_Sel;

//Outputs
 wire[7:0] ALU_Out;
 wire CarryOut;
 // Verilog code for ALU
```

```
 integer i;
 alu test_unit(
        A,B,  // ALU 8-bit Inputs
        ALU_Sel,// ALU Selection
        ALU_Out, // ALU 8-bit Output
        CarryOut // Carry Out Flag
    );
   initial begin
   // hold reset state for 100 ns.
     A = 8'h0A;
     B = 4'h0B;
     ALU_Sel = 4'h0;

     for (i=0;i<=15;i=i+1)
     begin
      ALU_Sel = ALU_Sel + 8'h01;
      #10;
     end


   end
endmodule
```

Synchronous Counter

```
module counter_behav ( count,reset,clk);
input wire reset, clk;
output reg [3:0] count;
 always @(posedge clk) if (reset)
count <= 4'b0000; else
count <= count + 4'b0001;
```

```
endmodule



module mycounter_t ;

 wire [3:0] count;
reg reset,clk; initial
clk = 1'b0; always
#5 clk = ~clk;
counter_behav m1 ( count,reset,clk);
initial
begin
reset = 1'b0 ;
#15 reset =1'b1;
#30 reset =1'b0;
#300 $finish; end
initial
$monitor ($time, "Output count = %d ",count );
Endmodule
```

Asynchronous Counter

```
module ripple_counter (clock, toggle, reset, count); input clock, toggle, reset;
output [3:0] count;
reg [3:0] count;
wire c0, c1, c2;
assign c0 = count[0],
 c1 = count[1],
 c2 = count[2];
```

```
always @ (posedge reset or posedge clock) if (reset == 1'b1) count[0] <= 1'b0;
else if (toggle == 1'b1) count[0] <= ~count[0]; always @ (posedge reset or negedge c0)
if (reset == 1'b1) count[1] <= 1'b0;
else if (toggle == 1'b1) count[1] <= ~count[1]; always @ (posedge reset or negedge c1)
if (reset == 1'b1) count[2] <= 1'b0;
else if (toggle == 1'b1) count[2] <= ~count[2]; always @ (posedge reset or negedge c2)
if (reset == 1'b1) count[3] <= 1'b0;
else if (toggle == 1'b1) count[3] <= ~count[3];
endmodule

module ripple_counter_t ; reg clock,toggle,reset; wire [3:0] count ;
ripple_counter r1 (clock,toggle,reset,count); initial
clock = 1'b0; always
#5 clock = ~clock; initial



begin
reset = 1'b0;toggle = 1'b0;
#10 reset = 1'b1; toggle = 1'b1;
#10 reset = 1'b0;
#190 reset = 1'b1;
#20 reset = 1'b0;
#100 reset = 1'b1;
#40 reset = 1'b0;
#250 $finish; end
initial
$monitor ($time, " output q = %d", count);
Endmodule
```

SRFlipflop

```
module SR_ff(q,qbar,s,r,clk); output q,qbar;
input clk,s,r; reg tq;
always @(posedge clk or tq) begin
if (s == 1'b0 && r == 1'b0) tq <= tq;
else if (s == 1'b0 && r == 1'b1) tq <= 1'b0;
else if (s == 1'b1 && r == 1'b0) tq <= 1'b1;
else if (s == 1'b1 && r == 1'b1) tq <= 1'bx;
end
assign q = tq; assign qbar = ~tq; endmodule

module SR_ff_test; reg clk,s,r;
wire q,qbar; wire s1,r1,clk1;
SR_ff sr1(q,qbar,s,r,clk); assign s1=s;
assign r1=r; assign clk1=clk; initial
clk = 1'b0; always
#10 clk = ~clk; initial
begin
s = 1'b0; r = 1'b0;
#30 s = 1'b1;
#29 s = 1'b0;
#1 r = 1'b1;
#30 s = 1'b1;
#30 r = 1'b0;
#20 s = 1'b0;
#19 s = 1'b1;



#200 s = 1'b1; r = 1'b1;
#50 s = 1'b0; r = 1'b0;
```

```
#50 s = 1'b1; r = 1'b0;
#10 ;
end always
#5 $display($time," clk=%b s=%b r=%b ",clk,s,r); initial
#500 $finish;
endmodule
```

TFlipflop

```
module t_ff(q,qbar,clk,tin,rst); output q,qbar;
input clk,tin,rst; reg tq;
always @(posedge clk or negedge rst) begin
if(!rst)
tq <= 1'b0; else
begin if (tin)
tq <= ~tq; end
end
assign q = tq; assign qbar = ~q; endmodule

module t_ff_test; reg clk,tin,rst; wire q,qbar;
t_ff t1(q,qbar,clk,tin,rst); initial
clk = 1'b0; always
#10 clk = ~clk; initial
begin



rst = 1'b0; tin = 1'b0;
#30 rst = 1'b1;
#10 tin = 1'b1;
#205 tin = 1'b0;
#300 tin = 1'b1;
#175 tin = 1'b0;
```

```
#280 rst = 1'b0;
#20 rst = 1'b1;
#280 tin = 1'b1;
#10 ;
end initial
#2000 $finish; endmodule
```