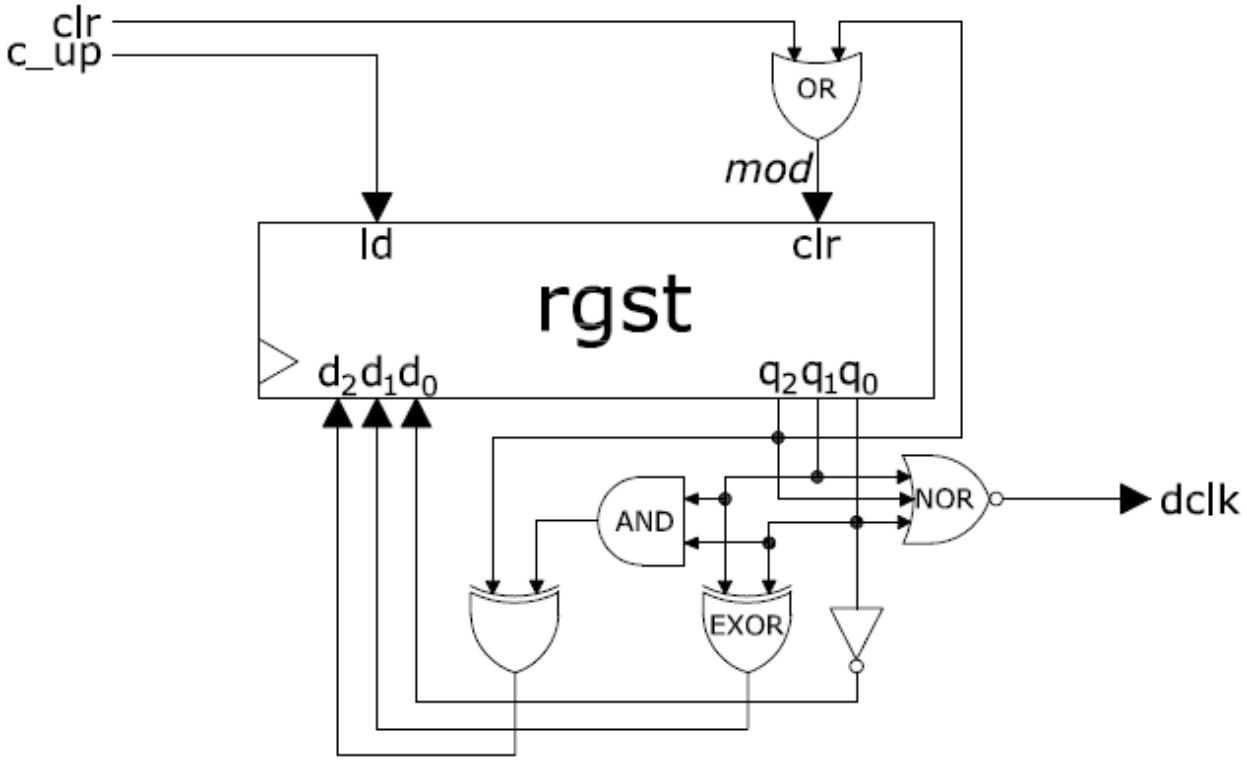
Week 7 – Control Unit Construction

One Hot encoding for FSM implementation

P.7.1 Implement, using Verilog language, a divide-by-5 counter described in the architecture below:

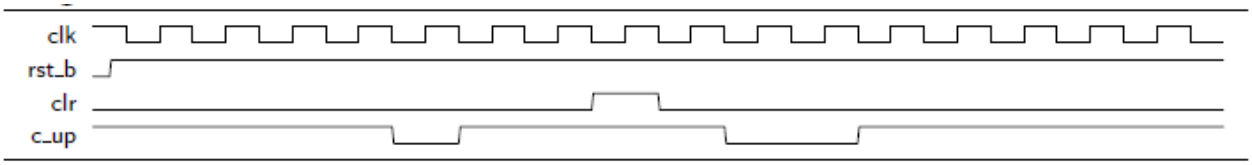



```
// Solution:
module register # (
parameter w=3,
parameter iv= { w { 1'b0}} )
( input [w-1:0] d,
input rst_b, clk, ld, clr,
output reg [w-1:0]q
);
```

```
always @ (posedge clk, negedge rst_b)
  begin
    if ((clr==1) || (!rst_b) )
      q<=iv;
    else if (ld==1)
      q<=d;
  end
endmodule


// Instantiation
module d5cntr(
  input clk, rst_b, c_up, clr,
  output dclk
  );
 wire mod;
 wire [2:0]q;
 assign mod= clr | q[2];
 register #(
.w(3),
.iv(3'd0))
register1 (
.d( { q[2]^ (q[1]&q[0]) , (q[1] ^q[0])  , ~q[0] } ),
.clk(clk), .ld(c_up), .rst_b(rst_b), .clr(mod),
.q(q));
assign dclk= ~(q[2] | q[1] | q[0]);
endmodule
```

P.7.2 Verify the ***d5cntr*** module with a testbench that generates the inputs as shown in the diagram below.



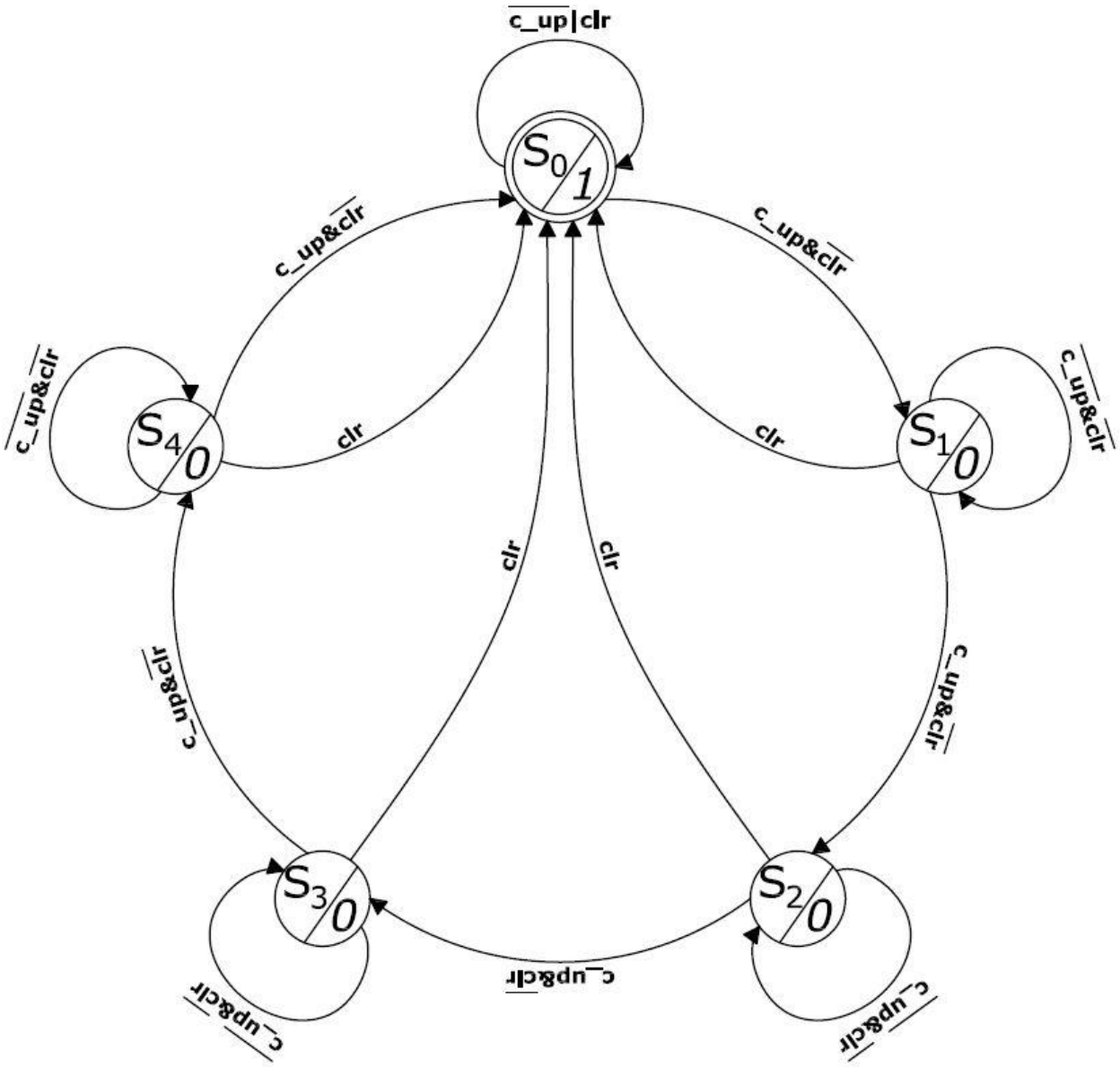

```
// Solution:
module d5cntr_tb (
 output reg clk,
 output reg rst_b, //asynch
 output reg clr,
 output reg c_up,
 output dclk
  );
d5cntr d5( .clk(clk),
.rst_b(rst_b),
.clr(clr),
.c_up(c_up),
.dclk(dclk));
 initial begin
 clk=1'd1;
  repeat(200) #20 clk=~clk;
  end
    initial begin
    rst_b=1'd0;
    #10 rst_b=1'd1;
  end
```

```
initial begin
   clr=1'd0;
   #300 clr=1'd1;
   #40 clr=1'd0;
  end

  initial begin
   c_up=1'd1;
   #180 c_up=1'd0;
   #40 c_up=1'd1;
   #160 c_up=1'd0;
   #80 c_up=1'd1;
  end
endmodule
```

P.7.3 Construct, using Verilog language, the transition diagram corresponding to the divide-by-5 counter described in the figure below:

```
// Solution:
module fsm_d5cntr(
input clk,
input rst_b, //asynch
input clr,
input c_up,
output dclk
);
```


$\overline{c\_up|clr}$
$S_0/1$
$c\_up\&\overline{clr}$
$\overline{c\_up}\&clr$
$S_1/0$
$\overline{c\_up\&clr}$
clr
$S_2/0$
$c\_up\&\overline{clr}$
clr
$S_3/0$
$c\_up\&\overline{clr}$
clr
$S_4/0$
$\overline{c\_up\&clr}$
$\overline{c\_up\&clr}$
$c\_up\&\overline{clr}$
$\overline{c\_up\&clr}$

```
// Declaring the registers for keeping the current state and the next state
reg [4:0] din;
wire [4:0] din_nxt;
// Writing the Boolean equations for each type D flip-flop
assign din_nxt[1] = (din[0] & c_up & (~clr)) | (din[1] & (~c_up) & (~clr)) ;
assign din_nxt[2] = (din[1] & c_up & (~clr)) | (din[2] & (~c_up) & (~clr)) ;
assign din_nxt[3] = (din[2] & c_up & (~clr)) | (din[3] & (~c_up) & (~clr)) ;
assign din_nxt[4] = (din[3] & c_up & (~clr)) | (din[4] & (~c_up) & (~clr)) ;
assign din_nxt[0] = (din[4] & c_up & (~clr)) | (din[0] & (~c_up) | clr) | (din[1] & clr) | (din[2]
& clr) | (din[3] & clr) | (din[4] & clr) ;
assign dclk = din[0];
always @ (posedge clk, negedge rst_b)
            if (! rst_b) din<=5'd1; // the default state will be S0
             else din <= din_nxt;
endmodule
```

P.7.4 Verify the ***fsm_d5cntr*** module using the same timing diagram as provided in ***P.7.2***.

```
// Solution:
module fsm_d5cntr_tb (
output reg clk,
output reg rst_b, //asynch
output reg clr,
output reg c_up,
output dclk
);
```

```
fsm_d5cntr FSM ( .clk(clk),
.rst_b(rst_b),
.clr(clr),
.c_up(c_up),
.dclk(dclk));
 initial begin
 clk=1'd1;
  repeat(200) #20 clk=~clk;
  end
    initial begin
    rst_b=1'd0;
    #10 rst_b=1'd1;
  end
initial begin
    clr=1'd0;
    #300 clr=1'd1;
    #40 clr=1'd0;
  end
  initial begin
    c_up=1'd1;
    #180 c_up=1'd0;
    #40 c_up=1'd1;
    #160 c_up=1'd0;
    #80 c_up=1'd1;
  end


endmodule
```