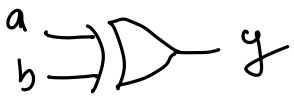
HDL Synthesis Idioms

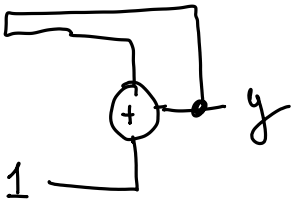
Exercise 1: Using the schematic symbols shown below, convert each of the following System Verilog expressions into a schematic.

```
y = a ^ b ;
```

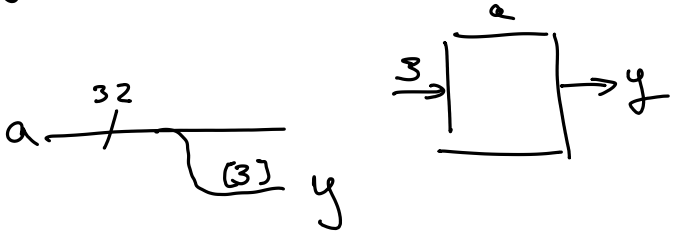



```
y = a < b ;
```

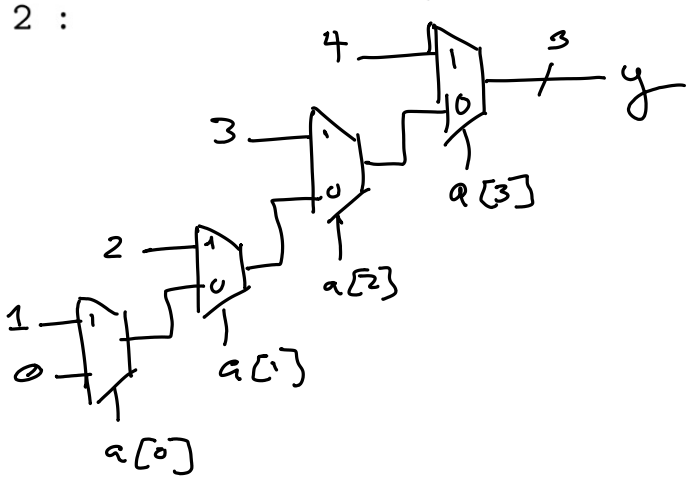

```
y = y+1 ;
```



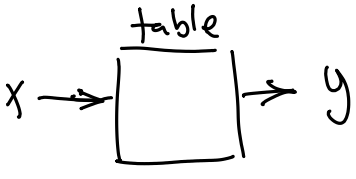

```
y = a[3] ;
```




```
y = a[3] ? 4 : a[2] ? 3 : a[1] ? 2 :
    a[0] ? 1 : 0;
```




```
y = table[x] ;
```

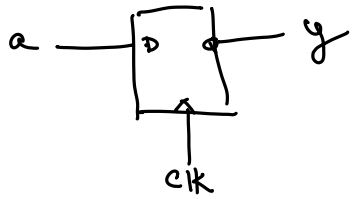



```
if ( y < b )
  y = y+1 ;
else
  y = y-1 ;
```

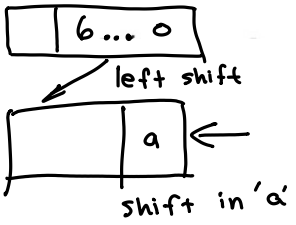

```
y = oe ? d : 16'hzzzz ;
```

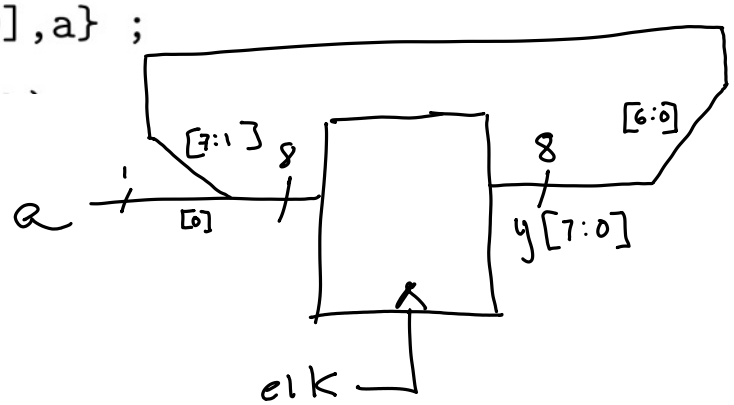
Exercise 2: Using the schematic symbols shown above, convert each of the following System Verilog expressions into a schematic.

```
always@(posedge clk)
  y <= a ;
```

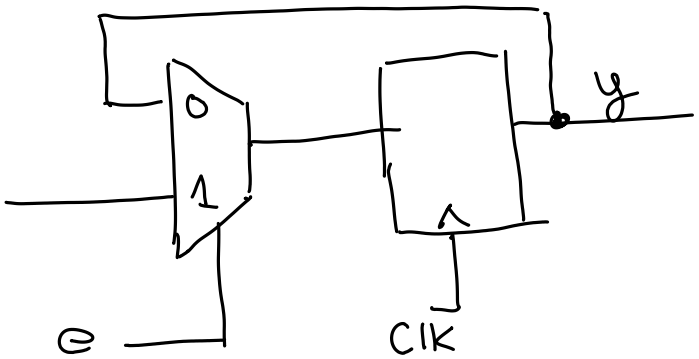



```
always@(posedge clk)
  y[7:0] <= {y[6:0],a} ;
```

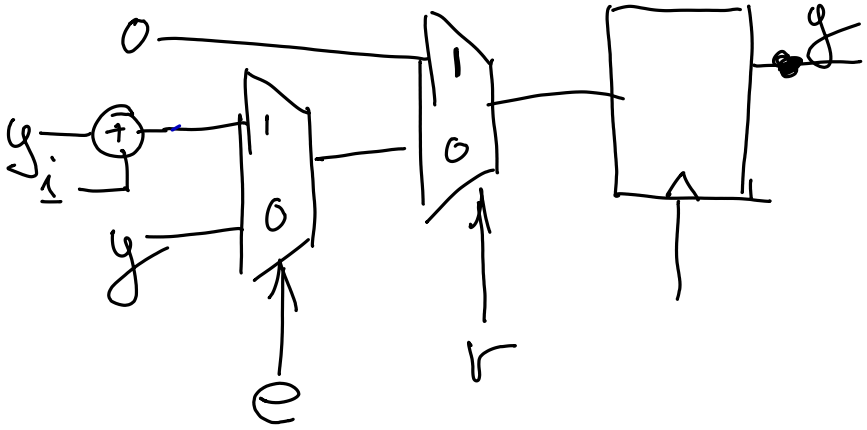




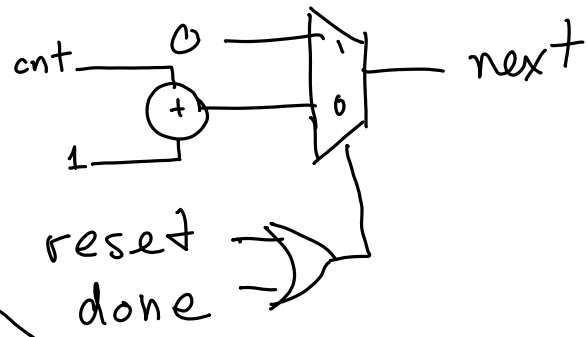

```
always@(posedge clk)
  if ( e )
     y <= a ;
  else
     y <= y ;
```



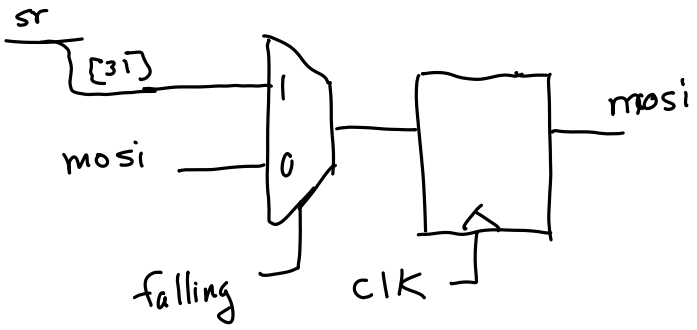

```
always@(posedge clk)
  if ( r )
    y <= '0 ;
  else
    if ( e )
       y <= y+1'b1 ;
    else
       y <= y ;
```



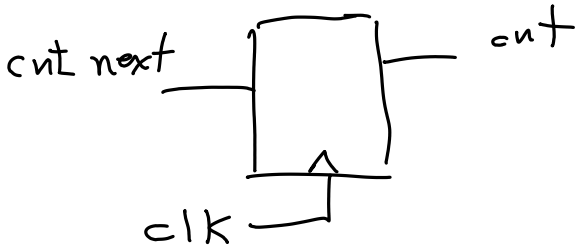
```
next = ( reset || done ) ? '0 : cnt+'b1 ;
```



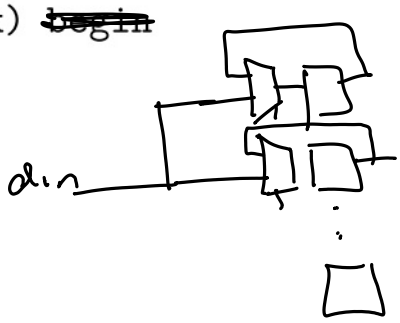

```
always@(posedge clk)
  if ( falling )
     mosi <= sr[31] ;
```

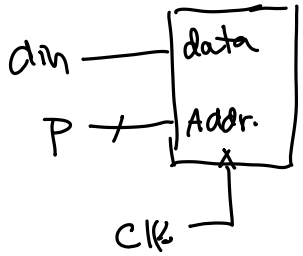



```
always@(posedge clk)
   cnt <= cnt_next ;
```

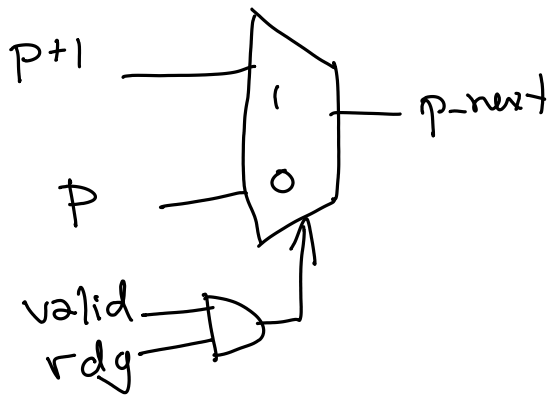



```
// logic [31:0] mem [15:0]
always_ff@(posedge clk) begin
   mem[p] <= din ;
```

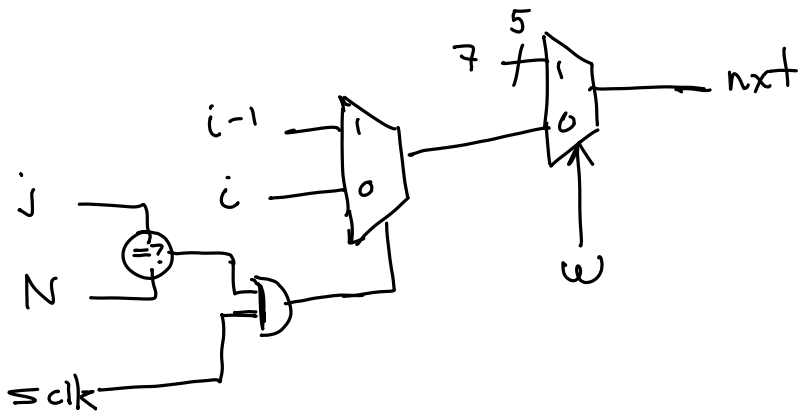




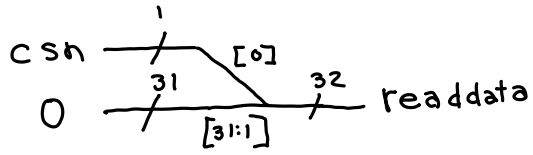

```
p_next = (valid && rdy) ? p + 1'b1 : p ;
```



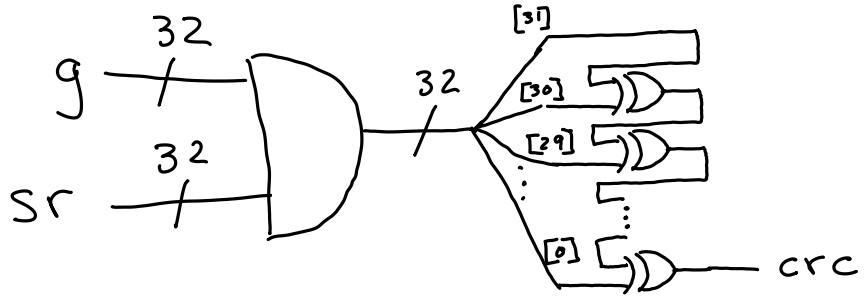
```
// i, j are logic[4:0]; w, sclk are logic
nxt = w ? 5'd7 : ( j==N && sclk ) ? i-1 : i ;
```



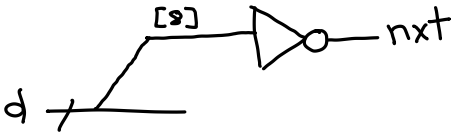

```
readdata = {31'b0,csn} ; // csn is logic
```



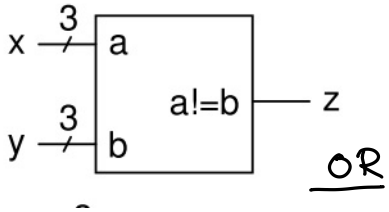

```
crc = ^ (g&sr) ; // g and sr are logic[31:0]
```




```
nxt = ~d[8] ;
```



Exercise 3: Write System Verilog that would generate each of the following schematics. Include any required signal declarations (using `logic`).

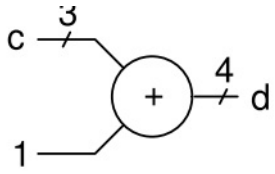



```
logic [2:0] x, y;
logic z;

z = x != y;

if (x != y)
    z = '1;
else
    z = '0;
```

OR

```
z = ~|(x ^ y)
```

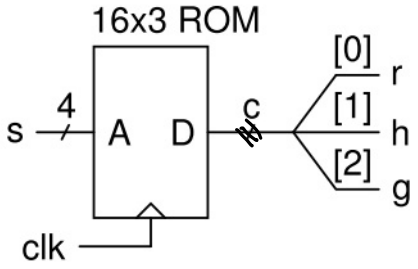
unary or (or-reduction)



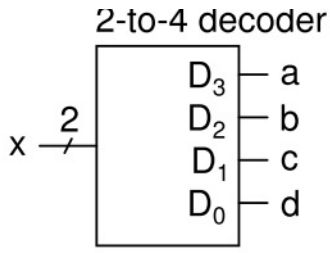

```
logic [2:0] c;
d = {1'b0, c} + 1'b1;
```

OR

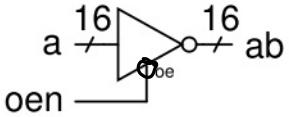
```
d = c + 4'b1;
```



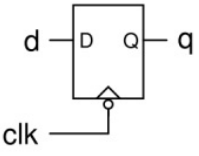

```
always @(posedge clk)
    c <= rom[s]

assign r = c[0]
       h = c[1]
       g = c[2]

r <= rom[s][0]
```



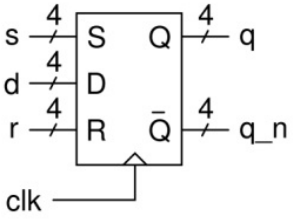

```
{a, b, c, d} = 4'b1 << x;
```



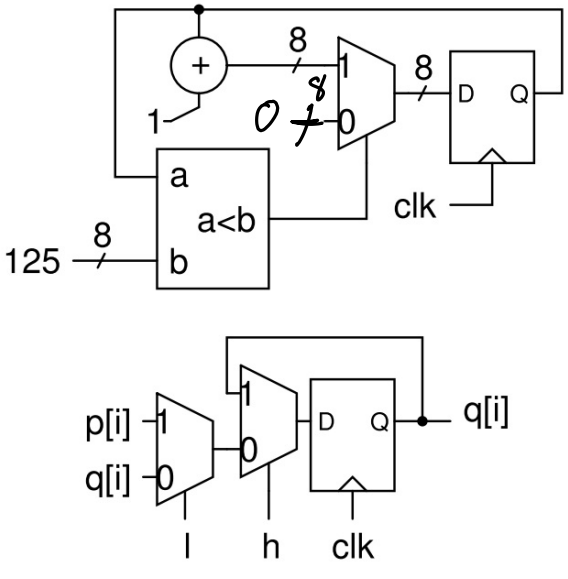

```
ab = oen ? 16'bz : ~a;
```




```
always @(negedge clk)
    q <= d;
```



(see below)

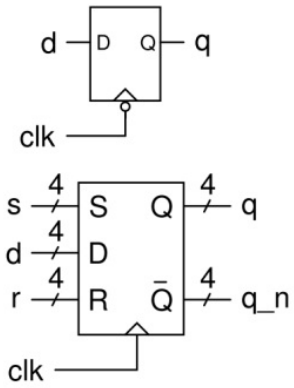



```
logic [7:0] q, d;
always_comb
    if (q<125)
        d = q+1;
    else
        d = 0;
always_ff @(posedge clk)
    q <= d;
```



```
always_comb
    d = h ? q[i] : (l ? p[i] : q[i]);
always_ff @(posedge clk)
    q <= d;
```

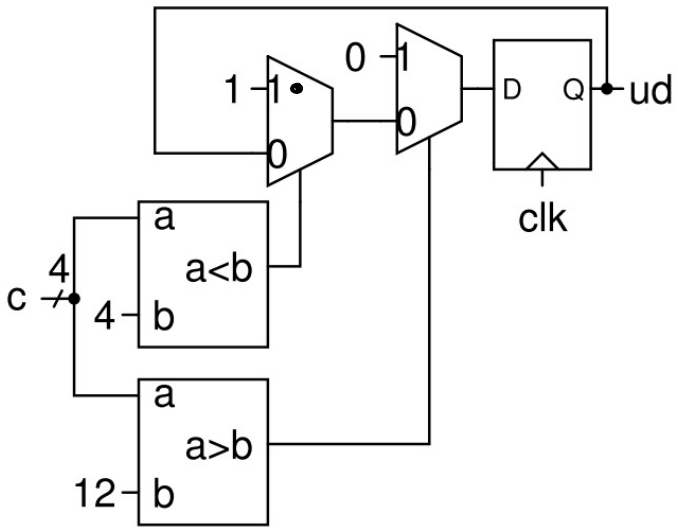
FF with Synchronous set/reset inputs:



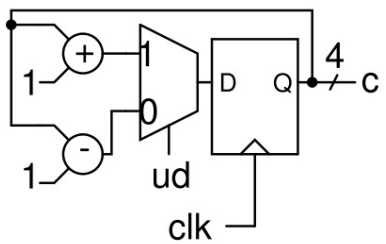

```
logic [3:0] q, q_n, s, d, r;

always_comb
    for (int i=0; i<4; i++)
        if (s[i])                    //set has priority
            q_next[i] = 1;
        else if (r[i])
            q_next[i] = 0;
        else
            q_next[i] = d[i];

always_ff @(posedge clk) begin
        q <= q_next;
        q_n <= ~q_next;
end;
```

asynchronous set/reset (not recommended):

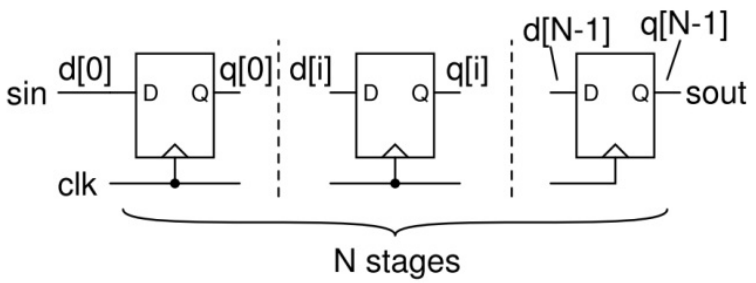
```
always @(s, r, clk)
    for (int i=0; i<4; i++)
        if (s[i] && !r[i]) q[i] = 1;
        else if (r[i] && !s[i]) q[i]=0;
        else if (posedge clk) q[i] = d[i];
```




```
always_comb
if (c > 12)
    d = 0;
else
    if (c < 4)
        d = 1;
    else
        d = ud;
always_ff @(posedge clk)
    ud <= d;
```




```
always_comb
    d = ud ? c+1 : c-1

always_ff @(posedge clk)
    c <= d;
```




```
always_comb begin
    for (int i=0; i<N-1; i++)
        d[i+1] = q[i];
    d[0] = sin;
end;

always_ff @(posedge clk)
    q <= d;

assign sout = q[N-1];
```