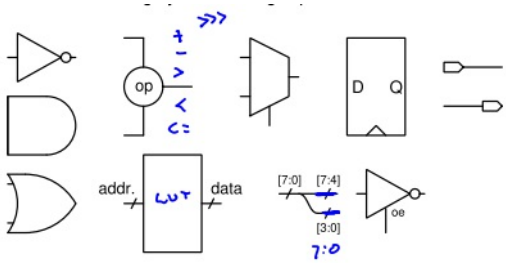
HDL Idioms

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




```
// assume the following declarations:
logic a, b, c, reset, up, down, oe ;
logic [15:0] x, y, y_next ;
logic [3:0] z ;
logic [3:0] tab [64] ;
```

```
assign c = a ^ b ;
```

```
assign a = x < y ;
```

```
assign y_next = y + 1 ;
```

'1 → expands

```
assign y = x[3] ;
```

```
assign y = 1'b1 << x ;  // decoder
```

≡ 2^x

```
assign z = tab[x] ;
```

```
assign y_next = y < x ? y + 1 : y - 1 ;
```

```
assign y = z[3] ? 4 :  // priority encoder
           z[2] ? 3 :
           z[1] ? 2 :
           z[0] ? 1 : 0;
```

```
assign y = oe ? x : 16'hzzzz ;
```

```
assign y_next = reset ? 16'h0 :
                up ? y + 1 :
                down ? y - 1 :
                y ;
```

{ braces

[brackets

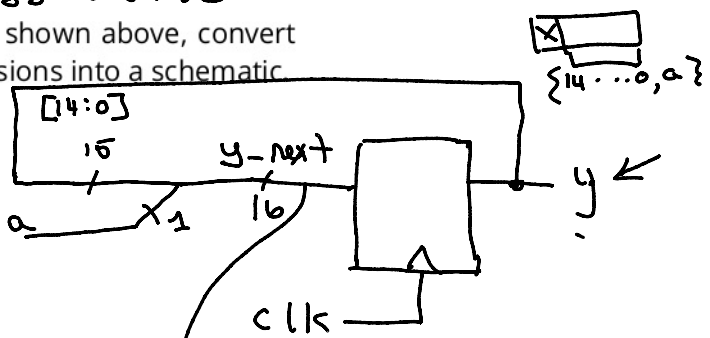
(parentheses

$y = (a + (b + c))$ right-associative

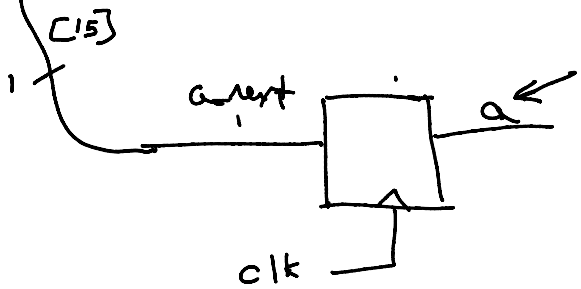
$y = ((a + b) + c)$ left-associative

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

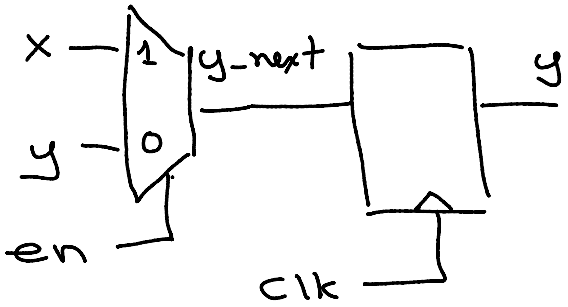
```
// shift register
assign y_next = {y[14:0],a} ;
always_ff@(posedge clk)
   y = y_next ;
```



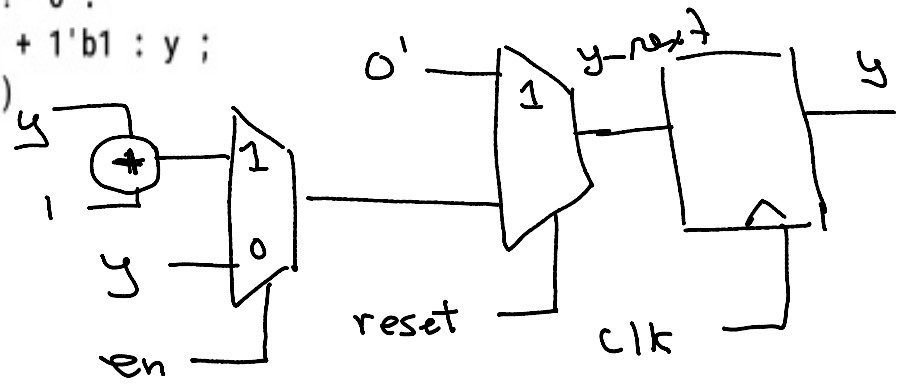

```
// FF showing if y (above) is even
assign a_next = y_next [15] ;
always_ff@(posedge clk)
   a = a_next ;
```




```
// register with load enable
assign y_next = en ? x : y ;
always_ff@(posedge clk)
   y = y_next ;
```

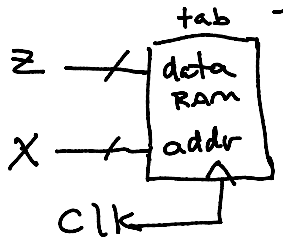



```
// counter with reset and count enable
assign y_next = reset ? '0 :
                en ? y + 1'b1 : y ;
always_ff@(posedge clk)
   y = y_next ;
```

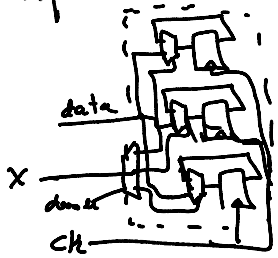


```
// counter with reset and count enable
assign y_next = reset ? '0 :
                on ? y + 1'b1 : y ;
always_ff@(posedge clk)
   y = y_next ;
```

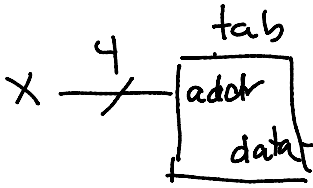
```
// writeable memory (RAM)
always_ff@(posedge clk)
   tab[x] = z ;
```



modelled by unpacked array

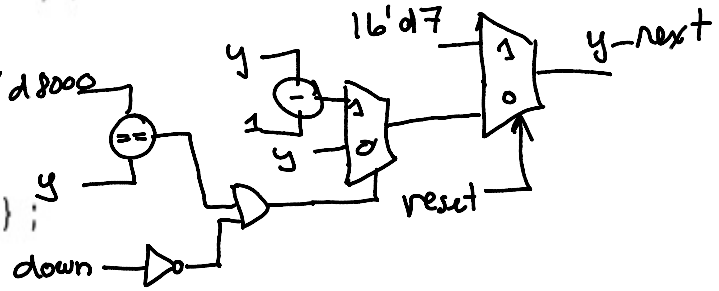



```
// read port
assign z = tab[x] ;
```



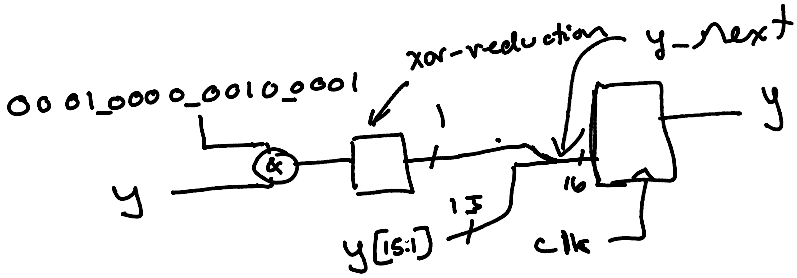
modelled using unpacked array (lookup table)

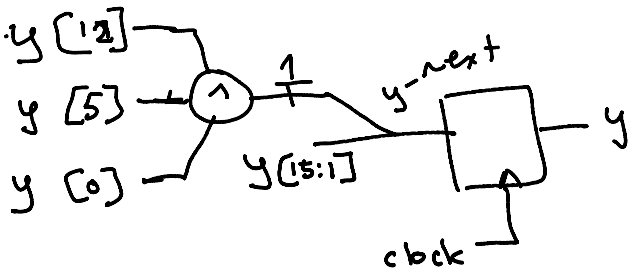
```
assign y_next = reset ? 16'd7 :
                ( y == 16'd8000 && !down ) ? y-1 : y ;
```



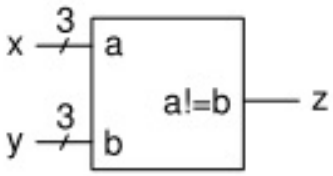
xor reduction

```
// CRC
assign y_next = { ^( y & 16'h0x1021 ), y[15:1] } ;
always_ff@(posedge clk)
   y = y_next ;
```

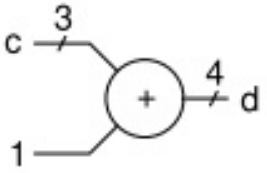




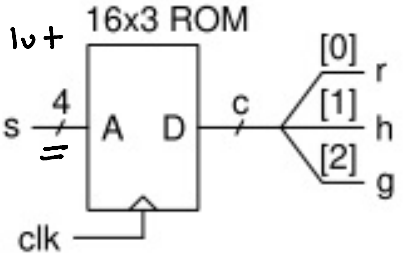
Exercise 3: Write System Verilog that would generate each of the following schematics. Include any required signal declarations (using `logic`).



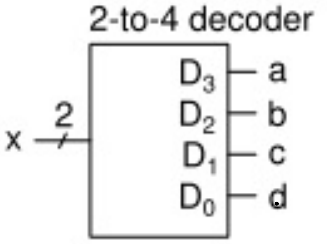

```
logic [2:0] a, b;
logic z;
assign z = a != b
```




```
logic [2:0] c;
logic [3:0] d;
assign d = c + 1'b1;
```




```
logic [2:0] lut [16], c;
logic [3:0] s;
logic r, h, g;
always @(posedge clk)
    c = lut[s];
assign {g,h,r} = c;
```



x	a	b	c	d
0 0	0	0	0	1
0 1	0	0	1	0
1 0	0	1	0	0
1 1	1	0	0	0

(1)
```
assign {a,b,c,d} =
    x == 2'b00 ? 4'b0001 :
    x == 2'b01 ? 4'b0010 :
    :
```

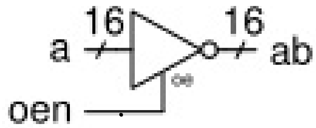
(2)
```
assign {a,b,c,d} = 4'b1 << x;
```

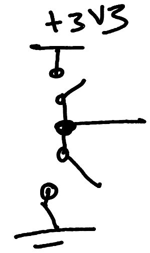
```
logic [3:0] lut [4] = '{4'b0001,
                        4'b0010,
                        :
                        };
```

(3)
```
assign {a,b,c,d} = lut[x];
```

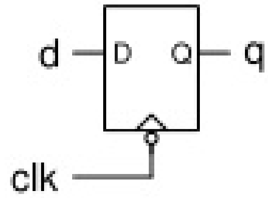
! ~   assign ab = oen ? ~a : 16'hzzzz;

- 0
- 1
- z
- x



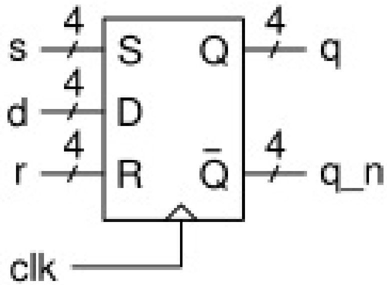


always_ff @(negedge clk)
q = d



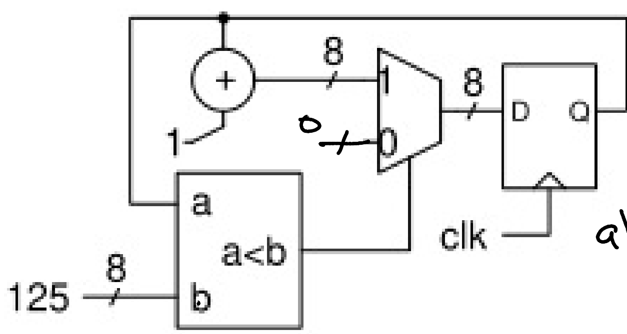
assign q_next = s ? '1 :
r ? '0 : d; (if only 1 bit)

assign q_next = (s | d) & ~r; (bit masking)



always_ff @(posedge clk)
q = q_next;

always_ff @(posedge clk)
q_n = q_n_next;



assign q_next = q < 125 ? q+1 : 0;

always_ff @(posedge clk)
q = q_next;

```
  0 0100
+ 1 0010
--------
```

&
&&

| - bitwise → bit by bit
vs || - logical → result either 0 or 1

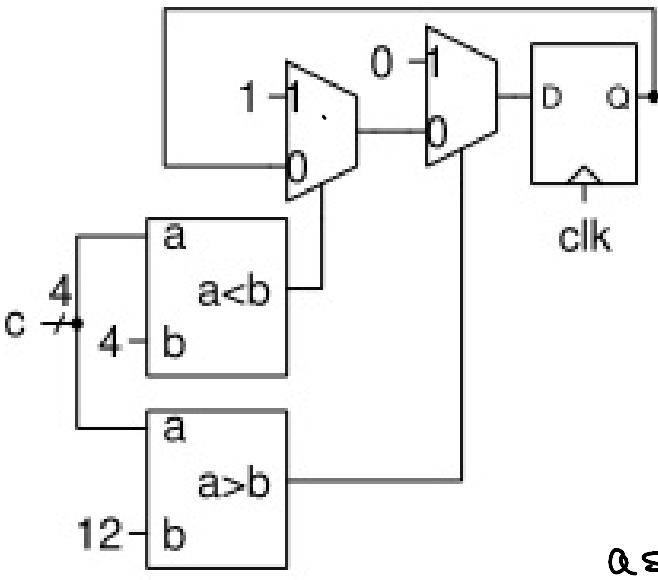
```
  0110         0110 →    1        0000 → 0
| 0101     || 0101 → || 1     || 0101 → 1
------     ----------------    -----------
  0111          1        1          1    1

                              || 0000
                                 0000
                                 ----
                                  0.

                               0110
                           (^) 0101
                               ----
                               0011
```

same as bitwise equal.

=   ==
assign   equality.




```
assign ud_next=
c>12 ? 0 : c<4 ? 1 : ud;
always_ff @(posedge clk)
    ud = ud_next;
```

```
assign c_next = ud ? c+1 : c-1;
always_ff @(posedge clk)
    c = c_next;
```