Introduction to Digital Design with Verilog HDL

Exercise 1: What changes would result in a 3-input OR gate?

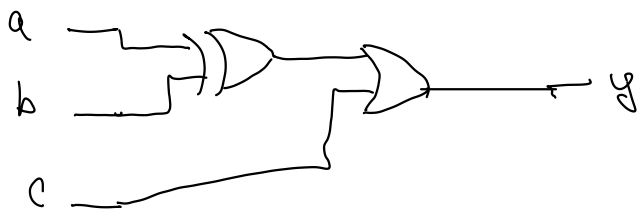
```
// AND gate in Verilog

module ex1 ( input logic a, b, c,
             output logic y ) ;

   assign y = a & b; a | b | c ;

endmodule
```

Exercise 2: What schematic would you expect if the statement was `assign y = ( a ^ b ) | c ;`?



Exercise 3: If the signal **i** is declared as **`logic [2:0] i;`**, what is the 'width' of **i**?

3 bits

If **i** has the value 6 (decimal), what is the value of **i[2]**?

$6_{10} = 110_2$ $i = (1, 1, 0)$ $i[2]$ $i[1]$ $i[0]$

Of **i[0]**? 0

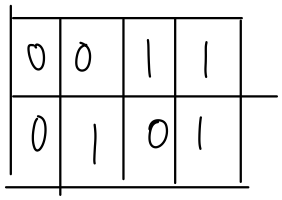
Exercise 4: What are the widths and values, in decimal, of the following:

	width	value (decimal)
`4'b1001`?	4	9
`5'd3`?	5	3
`6'h0_a`?	6	10
`3`?	32	3

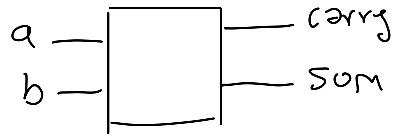
Exercise 5:

```
logic [3:0] z [0:1] = ' 3'b11, 3'b101 ;
```

Draw a two-dimensional table showing the values of the bits in **z**.



Exercise 6: Write the truth table for a one-bit adder with carry. There are three inputs (**a**, **b** ~~and carry~~) and two outputs (**sum** and **carry**). Define an array that implements this function. Write an expression that uses this array to find the sum and carry of `logic` signals **a** and **b**.



a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

{a, b} →

	sum	carry
00	0	0
01	1	0
10	1	0
11	0	1

```
logic [1:0] tt [0:3] =
    '{ 2'b00, 2'b10, 2'b10, 2'b01 };

assign y = tt[ {a,b} ];
```

Exercise 7: What are the values of the following expressions: `!4'b010`?

`!4'b010` → `1'b0`

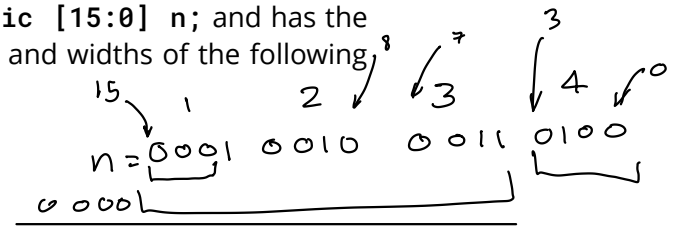
`~4'b010`? → `4'b1101`

width { `0+~(!0)?` (32, 1, 1)

value { 0, 0, 1

32'b0

Exercise 8: An array declared as `logic [15:0] n;` and has the value `16'h1234`. What are the values and widths of the following expressions?



`n[15:13]` — 3

3'b000
3'd0

`!n`

1'h0

`~n[3:0]`

4'b1011
4'hb

`n>>4`

16'h0123

`n + 1'b1`

16'h1235

`(n[7:0]) - (n[3:0])` — 8

8'h34 − 4'h4
8'h30

`n >= 16'h1234`

1 bit

1'h1

`n ^ '1`

= n ^ 16'hffff
= 16'hedcb

0001 0010 0011 0100
1111 1111 1111 1111
1110 1101 1100 1011

⊕	0	1
0	0	1
1	1	0

`n && (!n)`

1'b1 && 1'b0
1'b0

`n * ((!n) + 1'b1)` — 16

1'b0 + 1
1'b1

`1'b1 + 1'b1` → 1'b0

1 + 1
0

(1'b1 + 1'b1) + 2'b0

n = (1'b1 + 1'b1)

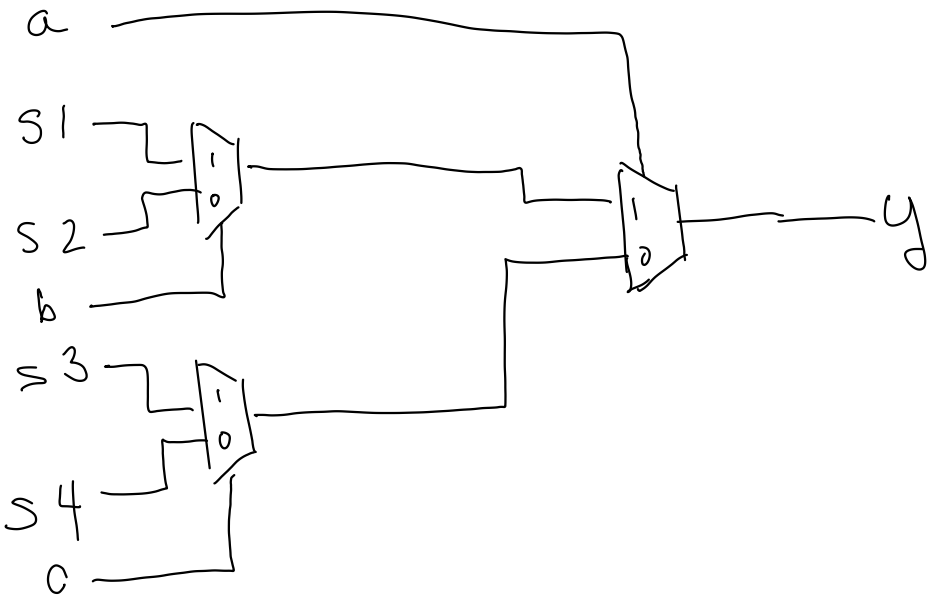
Exercise 9: What are the width and value of the expression: **`3 ? 16'd10 : 8'h20`**?

If **x** has the value 0, what is the value of the expression: **`x ? 1'b1 : 1'b0`**?

If **x** has the value -1?

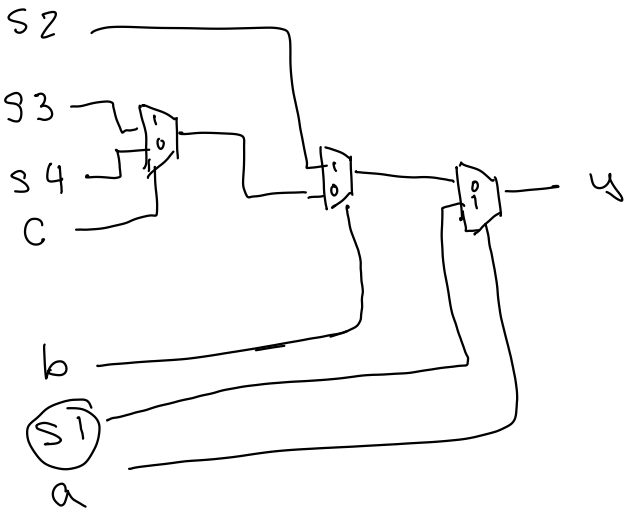
Exercise 10: Draw the schematics corresponding to:

```
y = a ? ( b ? s1 : s2 ) : ( c ? s3 : s4 );
```

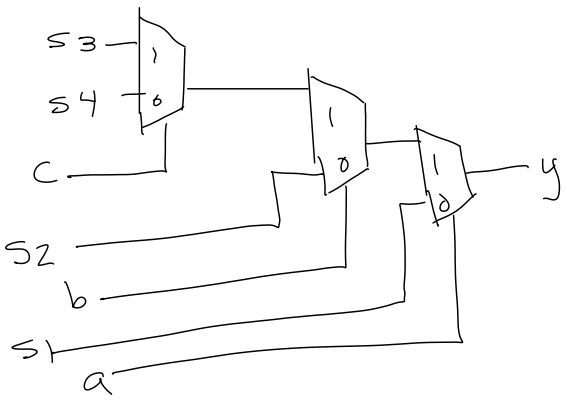


evaluate Right to Left.

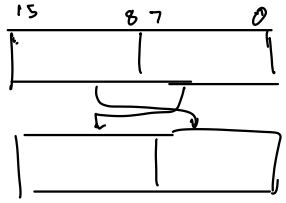
```
y = (a ? s1 : (b ? s2 : (c ? s3 : s4)));
```




```
y = (a ? (b ? (c ? s3 : s4) : s2) : s1);
```



Exercise 11: Use slicing and concatenation to compute the byte-swapped value of an array **n** declared as `logic [15:0] n`.



{ } [7:0]

{n[7:0], n[15:8]}

```
logic [15:0] m;
assign m = {n[7:0], n[15:8]};
```

Exercise 12: If **n** has the value `16'h1234`, what are the value and width of:

`{n[7:0], n[15:8], 4'b1111}`?

20'h 3412f

Exercise 13: Use concatenation to shift **n** left by two bits.

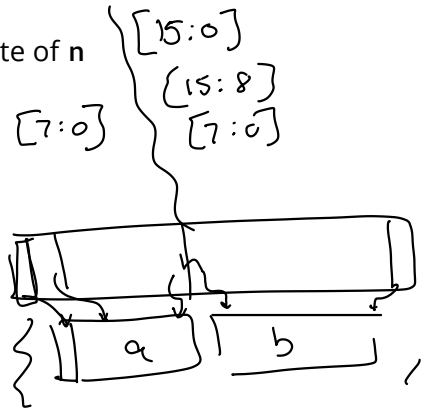
{ n[13:0] , 2'b00}

Exercise 14: Use concatenation to assign the high-order byte of **n** to **a** and the low-order byte to **b**.

assume a,b are declared

{ a , b, c } = n ;

[7:0] [7:0] [7:0] 15:0



logic X ;

logic [0:0] X ; ← X = 1'b1

X & 1

X [0]

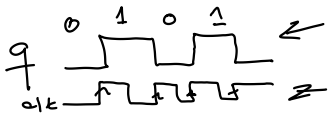
Exercise 15:

```
assign y = a + 1 ;
```

Some software warns about truncation. How could you re-write the **`assign`** statement to avoid such a warning?

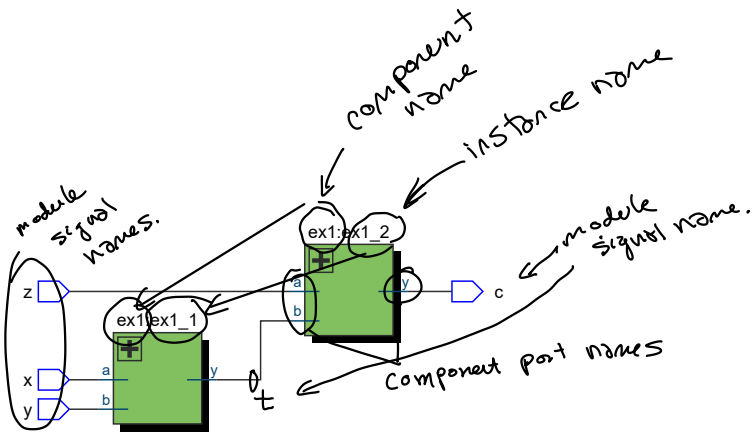
```
y = a + 1'b1;
```

Exercise 16: Write an **`always_ff`** statement that toggles (inverts) its one-bit output on each rising edge of the clock.




```
always_ff @( posedge clk )
    q <= !q ;
```

Exercise 17:



Identify the following in the diagram above: component names, component "instance names," component port names, module ~~port~~ signal names. Label the signal **`t`** in the schematic.

Exercise 18: Rewrite the **`ex60`** module using operators. Which version – "structural" or "behavioural" – is easier to understand?

```
module ex60 ( input logic x, y, z,
              output logic c ) ;

   logic t ;

   ex1 ex1_1 ( x, y, t ) ;
   ex1 ex1_2 ( z, t, c ) ;

endmodule
```

→ `assign c = x | y | z ;`

behavioural description (preferred)