Introduction to Digital Design with Verilog HDL

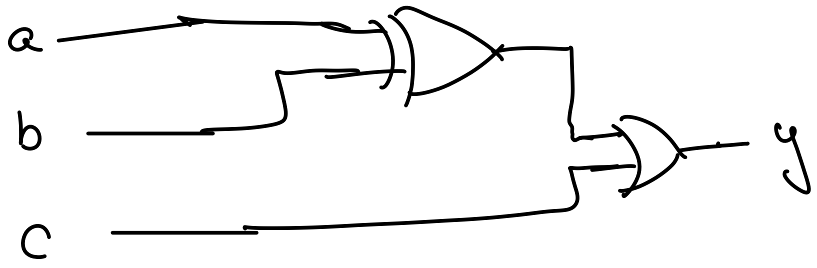
Exercise 1: What changes would result in a 3-input OR gate?

```
module or3 (
    input logic a, b, c,
    output logic y ) ;

   assign y = 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`**?

	2	1	0
i	1	1	0

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

i [2] = 1

[3:0]
[4:1]

Of **`i[0]`**?

i [0] = 0

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

`4'b1001`? 4 9

`5'd3`? 5 3

`6'h0_a`? 6 10

`3`? 32 3

Exercise 5: What are the values of the following expressions:

`!4'b010`? → 0

`~4'b010`? → 1101

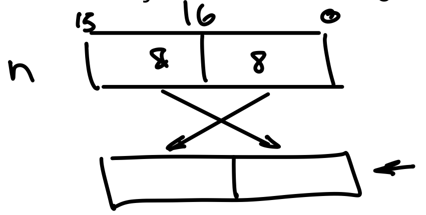
`|4'b0001`? |4'b0001 ≡ 0|0|0|1 = 1

`^4'b1001`? ^4'b1001 ≡ 1^0^0^1 = 0

`&4'b1111`? ≡ 1 & 1 & 1 & 1 → 1

`&4'b1110`? ≡ 1 & 1 & 1 & 0 → 0

Exercise 6: Use slicing and concatenation to compute the byte-swapped value of an array **n** declared as `logic [15:0] n`.



[a:b]

{ , }

n[15:8]

n[7:0]

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

Exercise 7: If **n** has the value **16'h1234**, what is the value and width of:

$16'h1234 = 16'b\underbrace{0001\ 0010}\underbrace{0011\ 0100}$

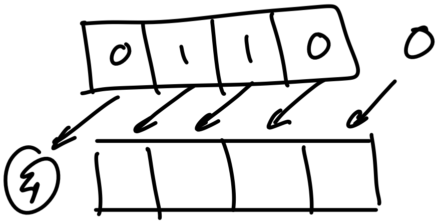
$\underbrace{1111}$

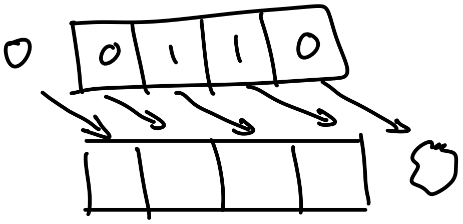
8'h34 8'h12 4'hf

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

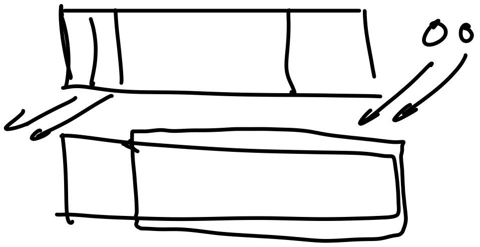
20'h 3412f

Exercise 8: Use concatenation to shift **n** left by two bits.

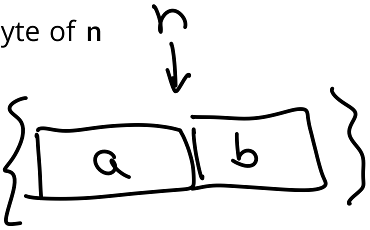




{n[13:0], 2'b00}



Exercise 9: Use concatenation to assign the high-order byte of **n** to **a** and the low-order byte to **b**.



```
logic [7:0] a,b;
logic [15:0] n;
```

```
assign a = n[15:8];      ≡   assign {a, b} = n;
assign b = n[7:0];           (OR {n[15:8], n[7:0]})
```

Exercise 10: What are the width and value of `{ {3{2'b10}}, 2'b11 }`?

{3{2'b10}} = 6'b 101010

{ {3{2'b10}}, 2'b11 } = 8'b 10101011

Exercise 11: An array declared as **`logic [15:0] n;`** and has the value **`16'h1234`**. What are the values and widths of the following expressions?

16'h 1234
16'b 0001 0010 0011 0100
16'b 1111 1111 1111 1111

1110 1101

`n[15:13]`

3'b000

`!n` → 1'b 0

0 → F ←
~~0~~ → T ←

4'b0100

`~n[3:0]` ~4'b0100 → 4'b1011

16'h1234 >>4 → 16'h123
or 16'h0123

`n>>4`

`n + 1'b1` → 16'h1235

`n[7:0] - n[3:0]`

8'h34 - 4'h4

8'h30

n >= 16'h1234

1'b1

n ^ '1 ≡ ~n

16'h1234 ^ 16'hffff

16'b1110 1101 1100 1011

n && !n

16'h1234 && 1'b0

1'b0

n * (!n + 1'b1)

$\underbrace{\text{16'h1234} * \underbrace{\text{1'b0 + 1'b1}}_{}\ \text{1'b1}}_{\text{16'h1234}}$

>=
<=
==
!=
<
>
'0 -
'1 -

& | ^ →

&& || →

Exercise 12: What are the width and value of the expression: 3 ? $\boxed{\text{16'd10}}$: 8'h20?

16'd10 ≡ 16'ha

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

1'b0

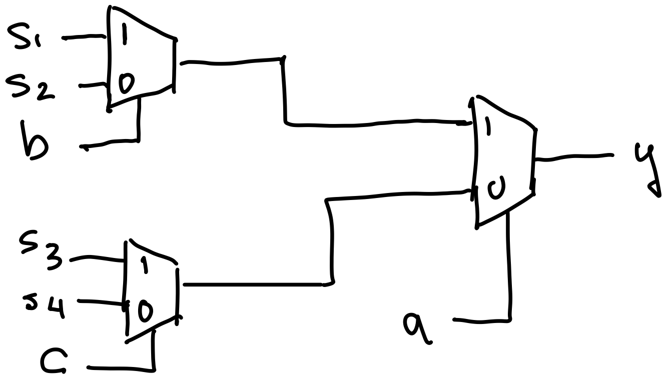
If **x** has the value -1?

-1 ? 1'b1 : 1'b0

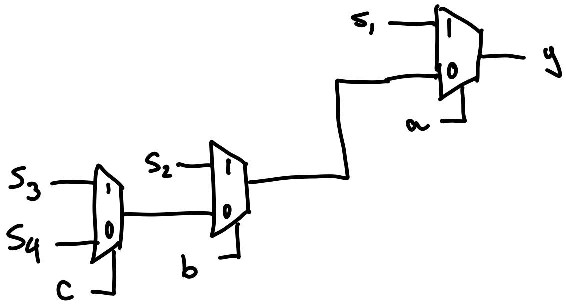
1'b1

Exercise 13: Draw the schematics corresponding to:

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



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



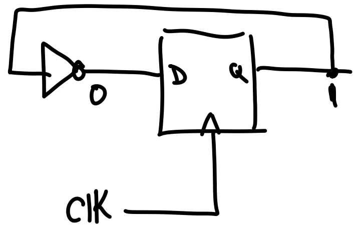
Exercise 14:

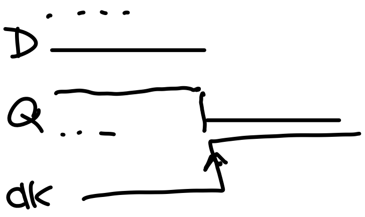
```
assign y = a + 1 ;
```

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

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

Exercise 15: Write an **`always_ff`** statement that toggles (inverts) its output on each rising edge of the clock.





~ 0 → 1
! 0 → 1

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