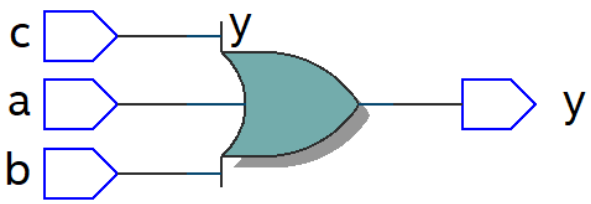
Introduction to Digital Design with Verilog HDL

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

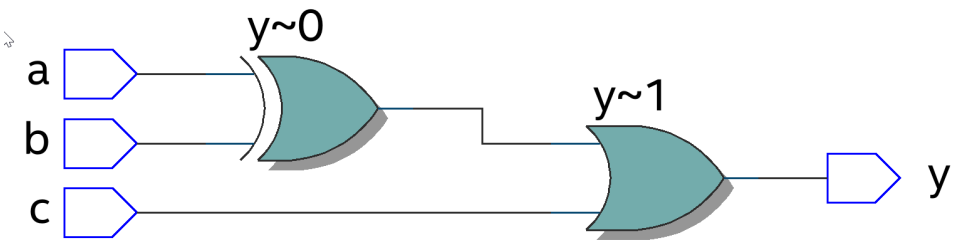
```
// AND gate in Verilog
module ex1 ( input logic a, b, c,
             output logic y ) ;

   assign y = a | b |c ;

endmodule
```



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

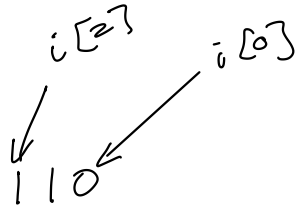


Exercise 3: What are the widths and values, in decimal, of the following:

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

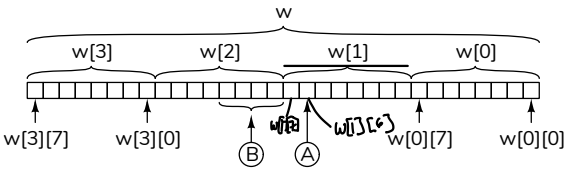
Exercise 4: If the signal `i` is declared as `logic [2:0] i;`, what is the 'width' of `i`? 3

If `i` has the value 6 (decimal), what is the value of `i[2]`? = 1



Of `i[0]`? = 0

Exercise 5:



How would you specify the bit marked A in the diagram above?

w[1][6]

The bits marked B?

w[2][3:0]

The least-significant byte? w[0]

Exercise 6: What are the values of the following expressions: `!4'b010`?

1'b0

`~4'b010`? ~4'b0010

4'b1101

32'b0 32'b0

`0+~(!0)`?

32'b0 32'h0

32'd0

→ !0 → 1'b1

→ ~1'b1 → 1'b0

→ 1'b0 + 32'd0 → 32'b0

↑ naive result

!0 → 32'b1 ← because result must be computed using width of final result (32 bits).

~32'b1 → 32'hffffffe

32'hffffffe

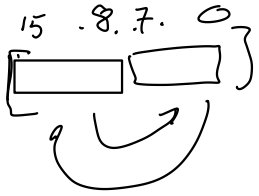
↑ actual result

Exercise 7: Use slicing and concatenation to compute the byte-swapped value of an array **n** declared as **`logic [15:0] n`**.

{010 , 101} → 010101 37

010 + 101 → 111 7

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



Exercise 8: If **n** has the value **16'h1234**, what is the value and width of:

0001 0010 0011 0100

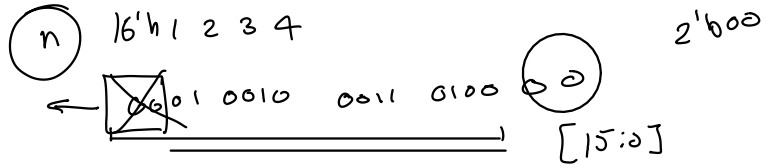
20 bits

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

0011 0100 0001 0010 1111

20'h 3 4 1 2 f

Exercise 9: Use concatenation to shift **n** left by two bits.



{ n[13:0] , 2'd0 }

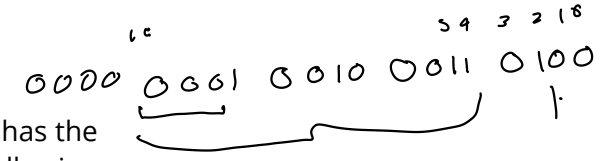
```
logic a [7:0];
logic b [7:0];
```

Exercise 10: Use concatenation to assign the high-order byte of **n** to **a** and the low-order byte to **b**.

assign { a , b } = n ;

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

16 ← 24



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?

`n[15:13]`

`!n`

`~n[3:0]`

`n>>4`

`n + 1'b1`

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

```
n >= 16'h1234
```

```
n ^ '1
```

16'hedcb

16'HEDCB

16'h1234

0001 0010 0011 0100

1111 1111 1111 1111

1110 1101 1100 1011

e d C b

```
(n && (!n))
```

1'b0

```
n * ( !n + 1'b1 )
```

16'h1234 → 16'h1234

16 bits

1'b0

1'b1 + 1'b1

2'b10

2'b0 + 2'b10

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

2 + 1

2

Exercise 12: What are the width and value of the expression: `16'd10 : 8'h20`?

3

width is 16

value 16'd10 ≡ 16'h000a

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

0

`x ?`

1'b0

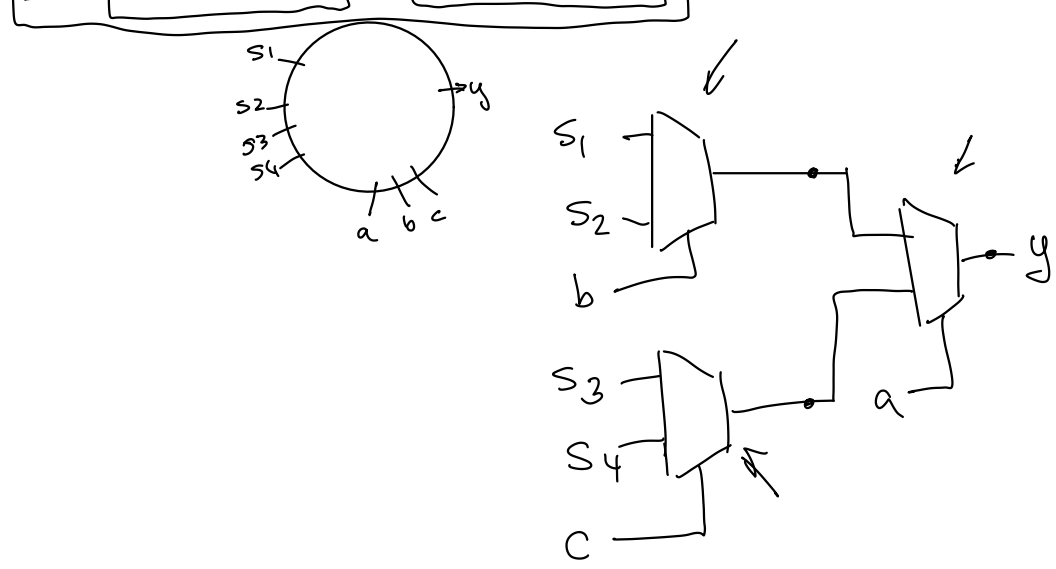
T F

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

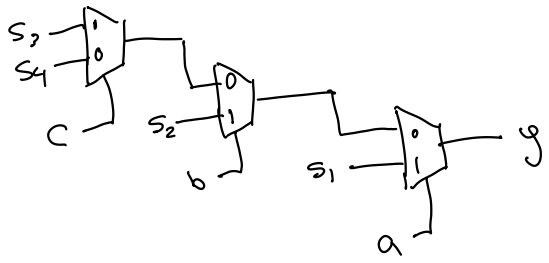
1'b1

Exercise 13: Draw the schematics corresponding to:

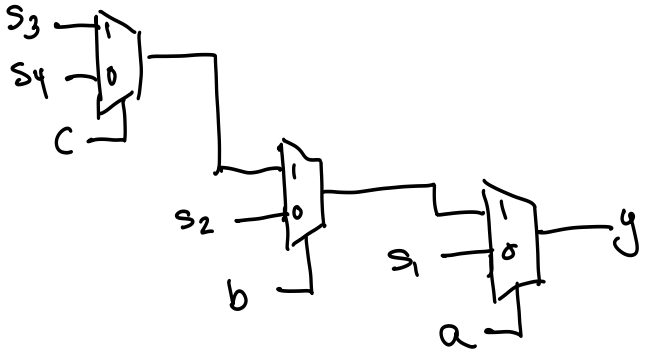
```
y = a ? ( b ? s1 : s2 ) : ( c ? s3 : s4 );
```




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




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



Exercise 14:

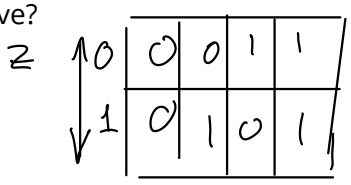
```
// concatenation:
logic [3:0] x = { 2'b00, 2'b11 } ;

// array literal
logic [0:1] [3:0] z = '{ 2'b11, 3'b101 } ;
```

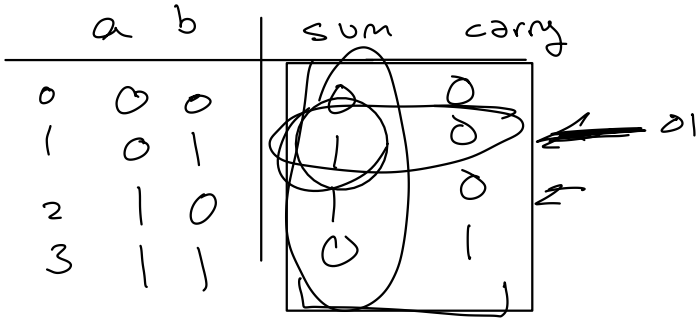
1 dim, 4...0

2 dim < 0..1 / 3..0

What are the dimensions and initial values of **x**, and **z** in the examples above?



Exercise 15: Write the truth table for a one-bit adder with 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**.



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

sum[{1'b0, 1'b1}] [1'b1]

0 0
0 1
1 0
1 1

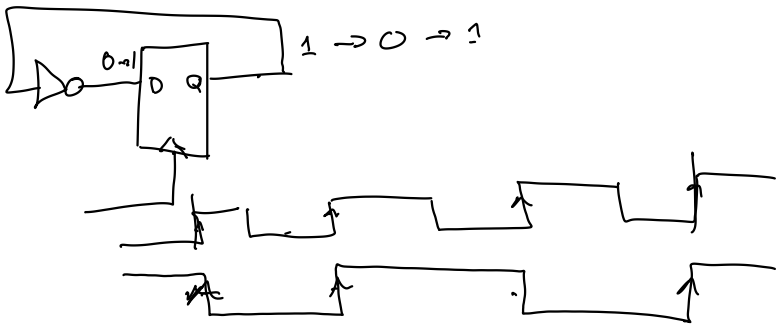
Exercise 16:

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

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

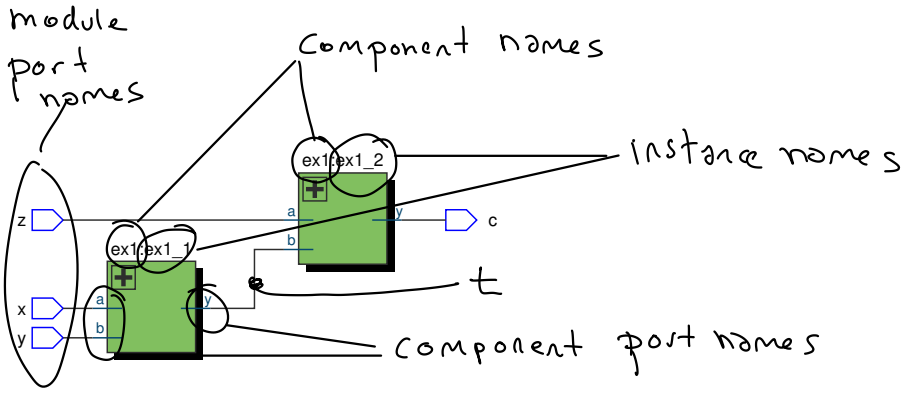
y = a + 1'b1;

Exercise 17: Write an **`always_ff`** statement that toggles (inverts) its output on each rising edge of the clock.




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

Exercise 18:



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

Exercise 19: 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 ) ;

  assign  c = x & y & z;

endmodule
```

← easier to understand.