Introduction to Digital Design with Verilog HDL

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

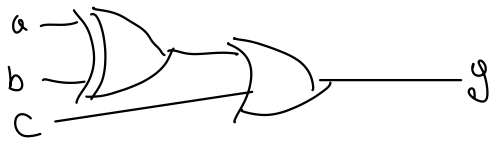
```
// AND gate in Verilog

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 ;`?

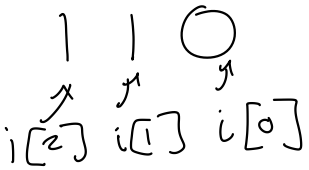


Exercise 3: If the signal `i` is declared as `logic [2:0] i;`, what is the 'width' of `i`?

3, bits 2, 1, 0

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

1



Of `i[0]`?

0.

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

	width	value in decimal
`4'b1001`?	4	9
`5'd3`?	5	3
`6'h0_a`?	6	10
same as `6'h00a`		
`3`?	32	3

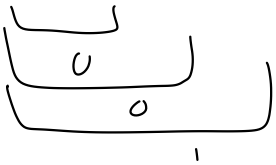
6
06
006
00006
↑
leading zeros don't change value

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

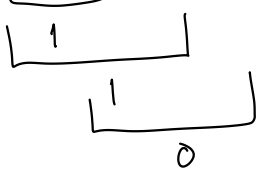
`!4'b010`? $1'b0 \equiv 1'h0$

`~4'b010`? $4'b1101$ ← 4 bits

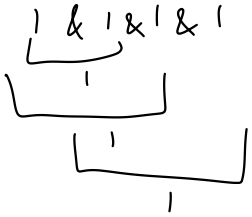
`|4'b0001`? $0 | 0 | 0 | 1$ (or-reduction)



`^4'b1001`? $1 \wedge 0 \wedge 0 \wedge 1 \rightarrow 1'h0$ (xor-reduction)

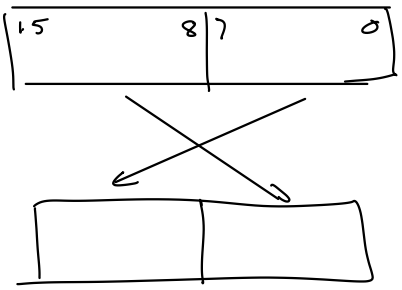


`&4'b1111`? $1'h1$



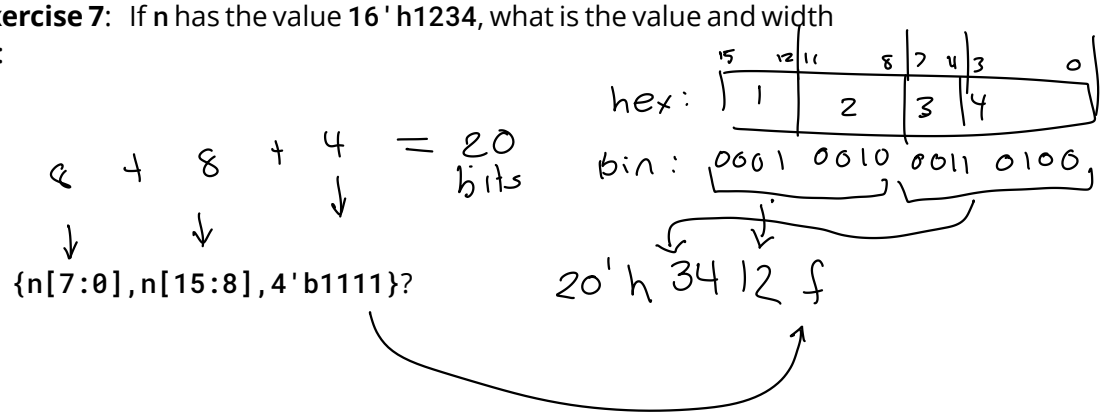
`&4'b1110`? $1'h0$ → and-reduction

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



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

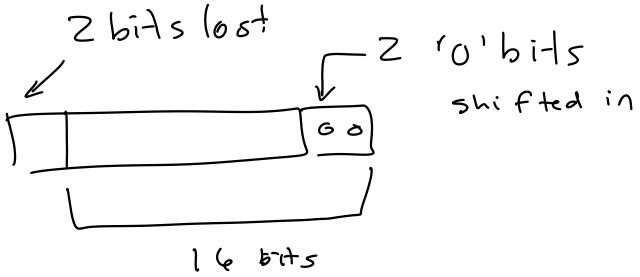
Exercise 7: If **n** has the value **16'h1234**, what is the value and width of:



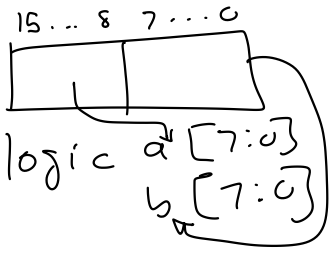
Exercise 8: Use concatenation to shift **n** left by two bits.

n[15:0]

{n[13:0], 2'b0}



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



assign {a,b} = n;

some results

assign a = n[15:8];
assign b = n[7:0];

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

replication

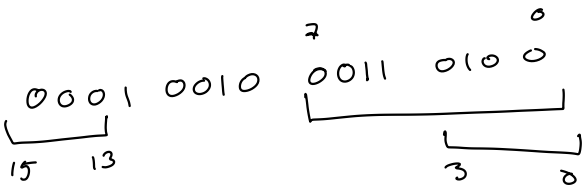
concatenation

{6'b101010 , 2'b11} → 8'b10101011

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]` 3'h0

`!n` 1'h0



`~n[3:0]` 0100 → 4'b1011 = 4'hb

`n>>4` 16'h0123

`n + 1'b1` 16'h1235

`n[7:0] - n[3:0]` 8'h34 − 4'h4 = 8'h30

`n >= 16'h1234` 1'h1

`n ^ '1`

```
  16'h1234
^ 16'hffff
----------
  16'hedcb
```

0001 0010 0011 0100
1111 1111 1111 1111
1110 1101 1100 1011 ← xor inverts (negates) each bit

(same result as ~n)

`n && !n` 16'h1234 & ~16'h0 = 1'h0

`n * ( !n + 1'b1 )` (1'h0, 1'b1)

16'h1234 (1'h1)

16'h1234

Exercise 12: What are the width and value of the expression: `3 ? 16'd10 : 8'h20`?

16'ha

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

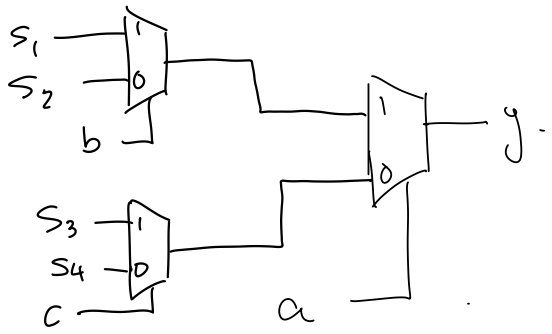
1'h0

If **x** has the value -1? 1'b1 == 1'h1

true

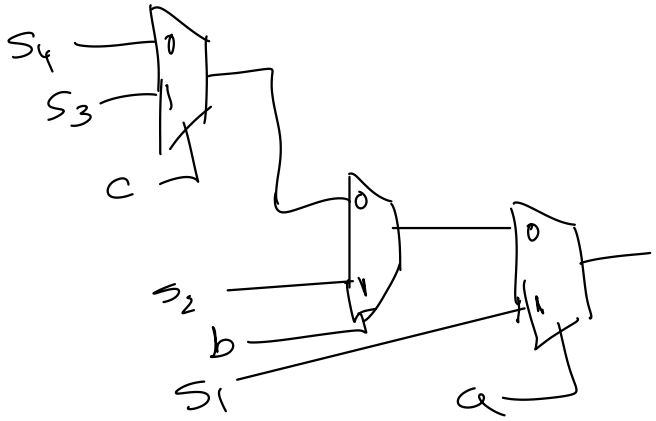
Exercise 13: Draw the schematics corresponding to:

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




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

```
y = a ? 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?

Use `assign y = a + {1'b1};`

1 bit

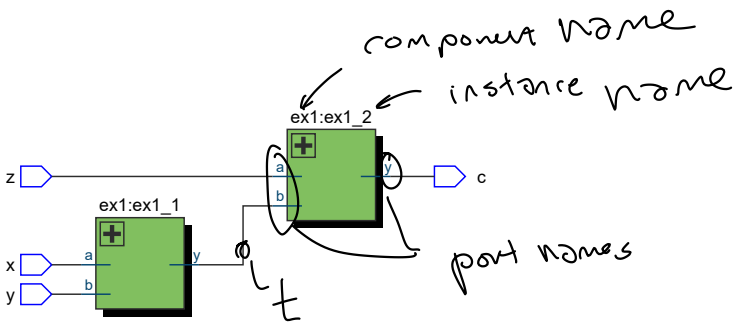
16 bits

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

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

(or !q if q is 1 bit)

Exercise 16:



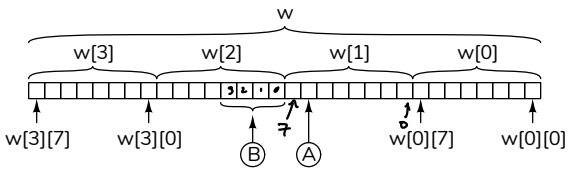
Identify the following in the diagram above: component names, component "instance names," component port names, module port names. Label the signal **`t`** in the schematic.

Exercise 17: Rewrite the **`ex60`** module using operators. Which version – "structural" or "behavioural" – is easier to understand?

```
module ....
  assign c = x & y & z ;
endmodule
```

← behavioural description (easier to understand)

Exercise 18:



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 19:

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

// replication (z=8'b1010_1010):
logic [7:0] y = { 4{2'b10} } ;

// array literal
logic [0:1] [3:0] z = '{ 4'b0011, 4'b1010 } ;
```

x is 4 bits: 4'b0011

y is 8 bits: 8'b10101010

z is 2x4 (0 upto 1) by (3 downto 0)

value is 4'b0011, 4'b1010

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

another example: if logic [3:0][7:0] x = '{1,2,3,4};

↑ array literal

x is 32'h1234

but logic [3:0][7:0] x = {1,2,3,4};

↑ concatenation.

x is 32'h1