



# DATAFLOW MODELING

## VERILOG DESIGN STYLE

---

*prepared by:*

**Gyro A. Madrona**  
Electronics Engineer

# TOPIC OUTLINE

Bitwise Operator

Dataflow Modeling



# BITWISE OPERATORS



# BITWISE NOT

---

```
wire a,b;
```

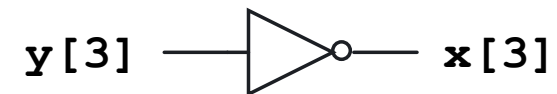
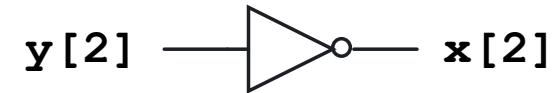
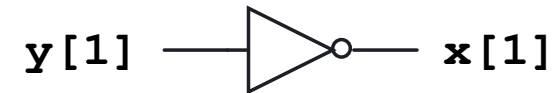
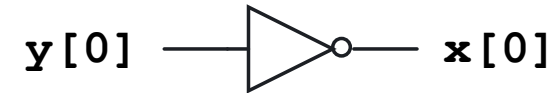
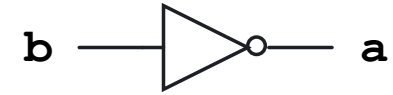
```
wire [3:0] x,y;
```

not ( $\sim$ )

```
a = ~b;
```

```
x = ~y;
```

implementation



# BITWISE OR

```
wire a,b,c;
```

```
wire [3:0] w,x,y;
```

or (|)

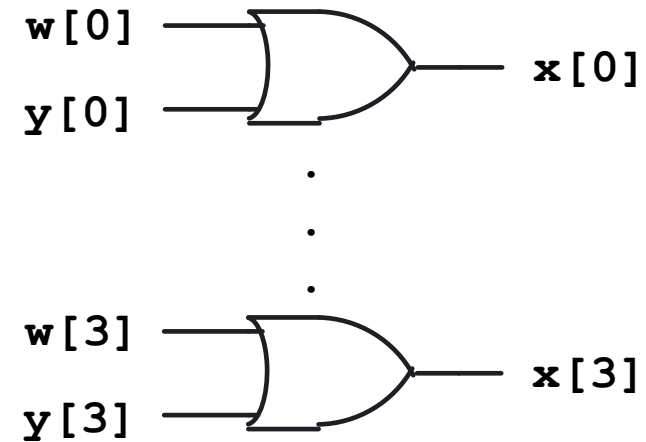
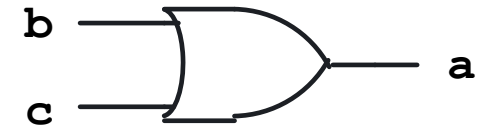
```
a = b | c;
```

```
x = w | y;
```

Set bits

	w	0	0	1	0	
(mask)	y	0	1	0	1	1-set
	<hr/>					
	x	0	1	1	1	

implementation



# BITWISE AND

```
wire a,b,c;
```

```
wire [3:0] w,x,y;
```

and (&)

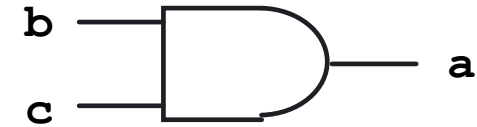
```
a = b & c;
```

```
x = w & y;
```

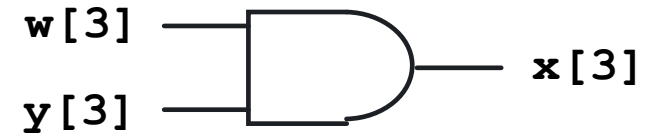
Reset bits

	<b>w</b>	<b>1101</b>	
(mask)	<b>y</b>	<b>0110</b>	0-reset
	<b>x</b>	<b>0100</b>	

implementation



⋮



# BITWISE XOR

```
wire a,b,c;
```

```
wire [3:0] w,x,y;
```

xor (^)

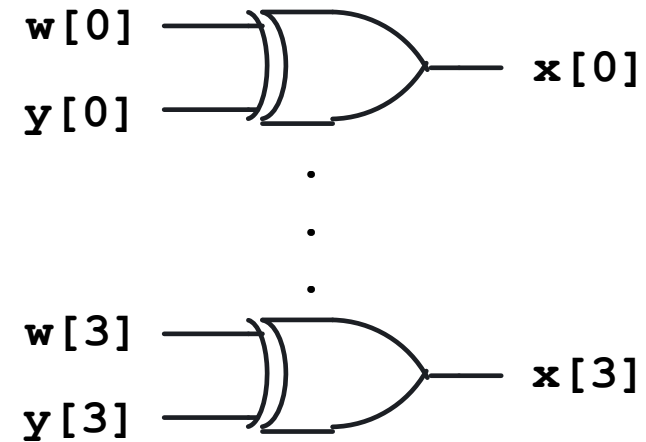
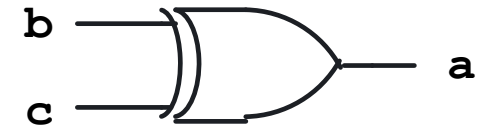
```
a = b ^ c;
```

```
x = w ^ y;
```

Flip bits

	w	0	1	0	1	
(mask)	y	0	1	1	0	1-flip
	x	0	0	1	1	

implementation



# BITWISE NOR

```
wire a,b,c;
```

```
wire [3:0] w,x,y;
```

nor (~|)

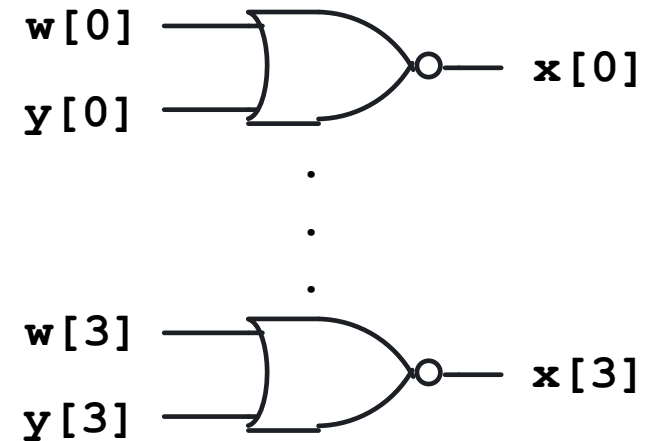
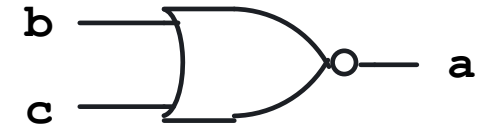
```
a = b ~| c;
```

```
x = w ~| y;
```

Reset and flip bits

	<b>w</b>	<b>0101</b>	<i>0-flip</i>
<i>(mask)</i>	<b>y</b>	<b>0110</b>	<i>1-reset</i>
	<hr/>		
	<b>x</b>	<b>1001</b>	

implementation





# BITWISE NAND

```
wire a,b,c;
```

```
wire [3:0] w,x,y;
```

nand (~&)

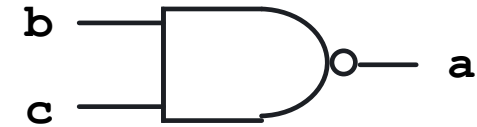
```
a = b ~& c;
```

```
x = w ~& y;
```

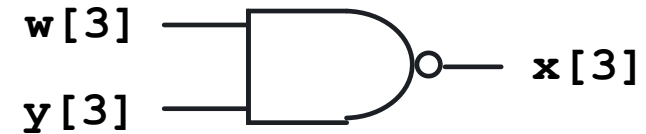
Set and flip bits

	<b>w</b>	<b>0101</b>	<i>0-set</i>
<i>(mask)</i>	<b>y</b>	<b>1001</b>	<i>1-flip</i>
	<hr/>		
	<b>x</b>	<b>1110</b>	

implementation



⋮



# BITWISE XNOR

```
wire a,b,c;
```

```
wire [3:0] w,x,y;
```

xnor ( $\sim^{\wedge}$ )

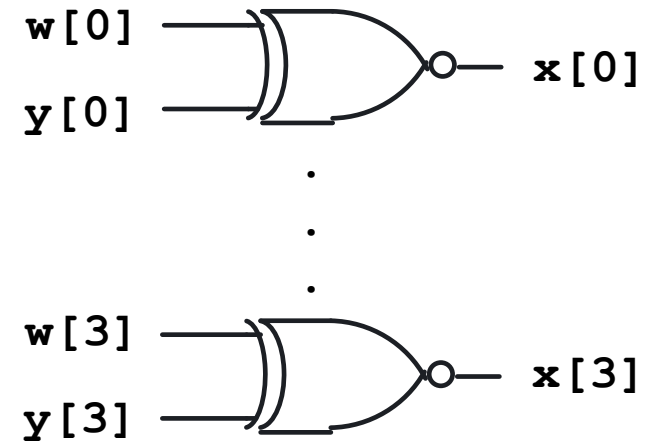
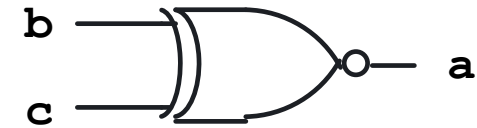
```
a = b ~^ c;
```

```
x = w ~^ y;
```

Equality check

	w	0101
(mask)	y	0100
	<hr/>	
	x	1110
		1-equal

implementation



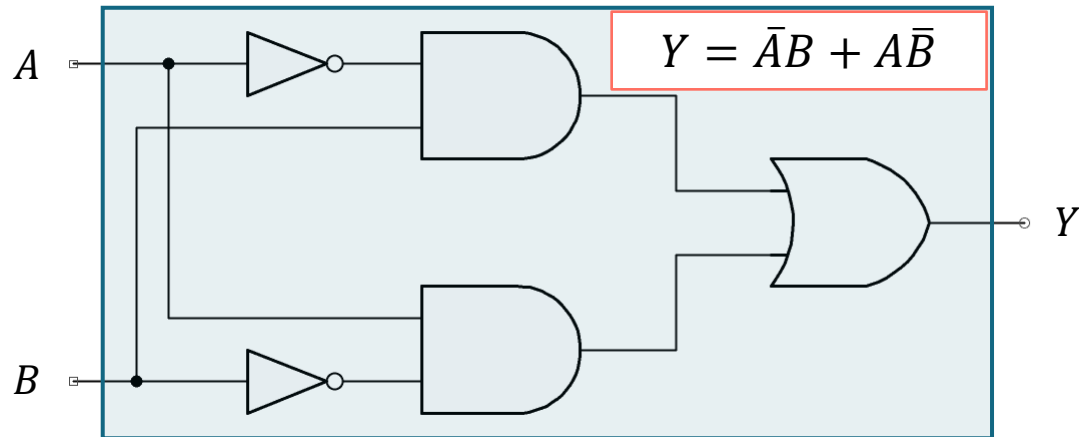
# DATAFLOW MODELING



# XOR MODULE

module name

**xor\_gate**



```
module xor_gate(Y,A,B) ;
```

```
    input A,B;
```

```
    output Y;
```

```
    assign Y = (~A & B) | (A & ~B) ;
```

```
endmodule
```

The assign keyword is used for continuous assignments – it continuously drives a value onto a net based on an expression.

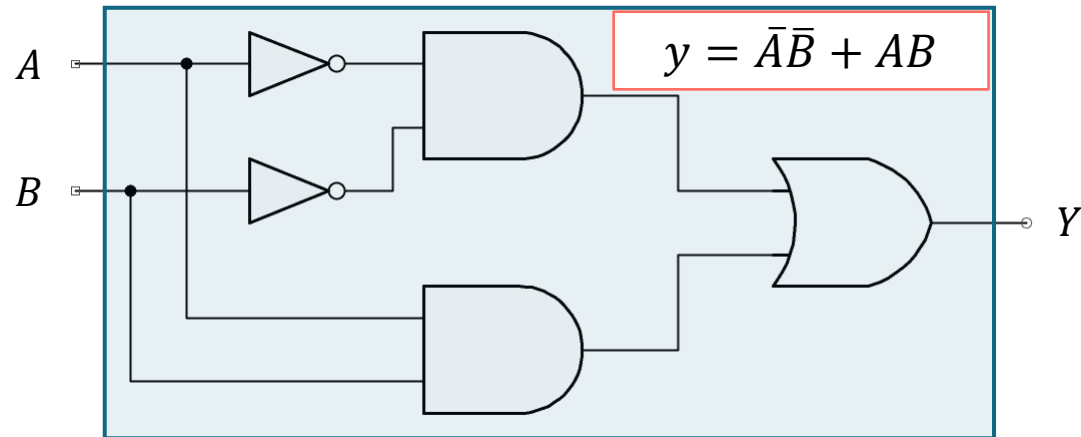
Dataflow modeling describes a circuit using continuous assignments and requires knowing the logic expression of the design.



# XNOR MODULE

module name

**xnor\_gate**



```
module xnor_gate(Y,A,B) ;
```

```
    input A,B;
```

```
    output Y;
```

```
    assign Y = (~A & ~B) | (A & B) ;  
endmodule
```

Dataflow modeling describes a circuit using continuous assignments and requires knowing the logic expression of the design.



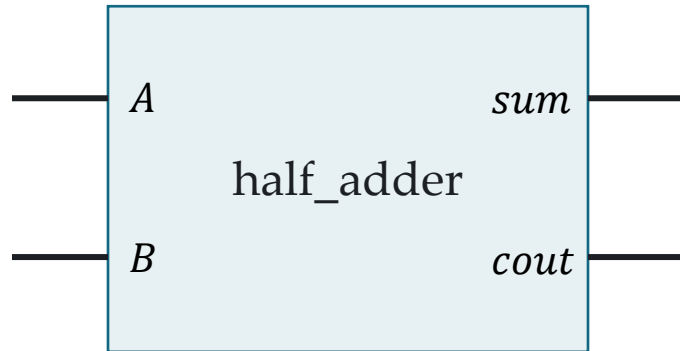
# HALF-ADDER MODULE

---

module name

// Verilog code

**half\_adder**



$$cout = AB$$

$$sum = A \oplus B$$

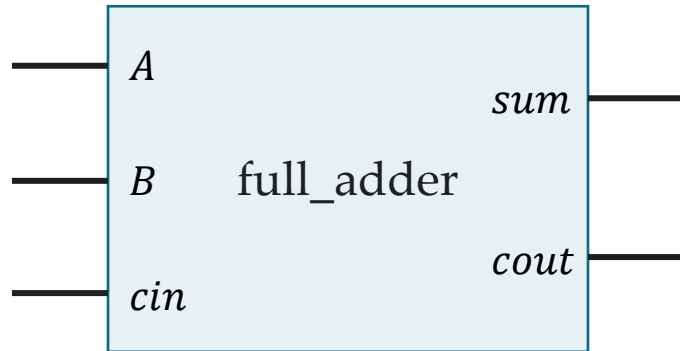


# FULL-ADDER MODULE

module name

// Verilog code

**half\_adder**



$$cout = cin(A \oplus B) + AB$$

$$sum = cin \oplus A \oplus B$$



# LABORATORY

