

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;
```

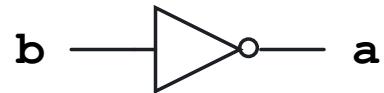
```
wire [3:0] x,y;
```

not (\sim)

```
a = ~b;
```

```
x = ~y;
```

implementation



BITWISE OR

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

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

or (|)

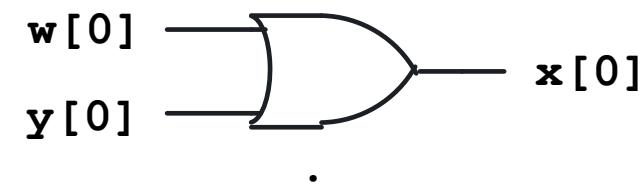
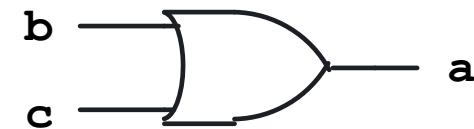
```
a = b | c;
```

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

Set bits

w 0010	
(mask) y 0101	
<hr/>	
x 0111	1-set

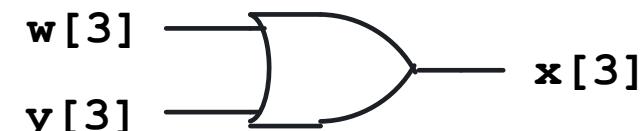
implementation



.

.

.



BITWISE AND

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

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

and (&)

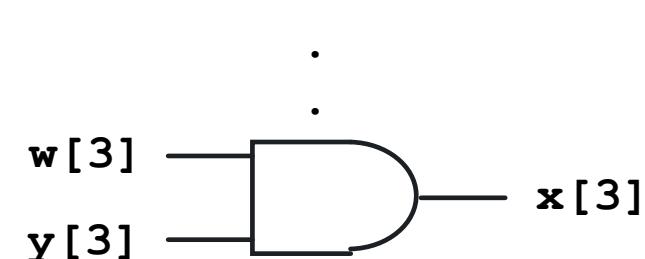
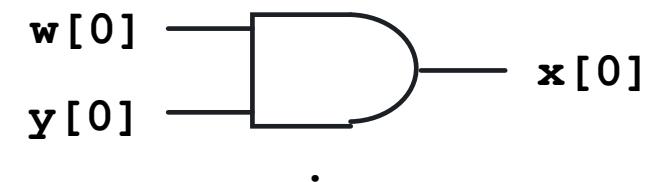
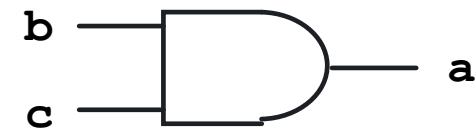
```
a = b & c;
```

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

Reset bits

$$\begin{array}{r} w \quad 1101 \\ (mask) \quad y \quad 0110 \\ \hline x \quad 0100 \end{array} \quad 0\text{-reset}$$

implementation



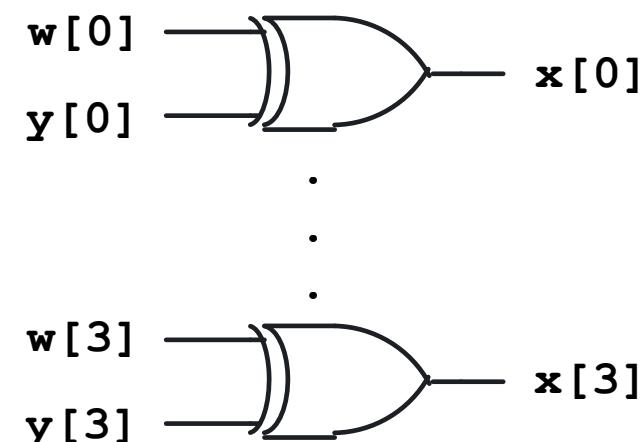
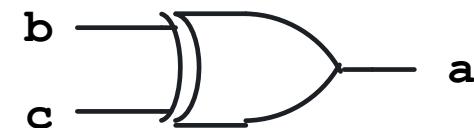
BITWISE XOR

```
wire a,b,c;  
wire [3:0] w,x,y;  
xor (^)  
    a = b ^ c;  
    x = w ^ y;
```

Flip bits

$$\begin{array}{r} w \text{ } 0101 \\ (mask) \text{ } y \text{ } 0110 \\ \hline x \text{ } 0011 \end{array} \quad \text{1-flip}$$

implementation



BITWISE NOR

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

nor ($\sim |$)

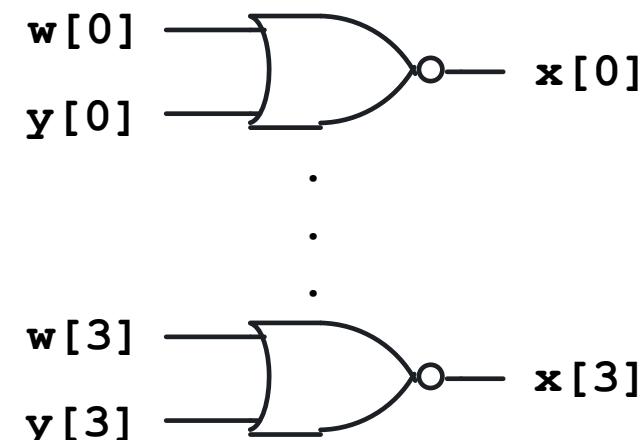
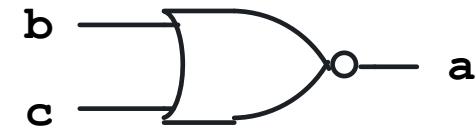
```
a = b ~|~ c;
```

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

Reset and flip bits

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

implementation



BITWISE NAND

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

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

nand (~&)

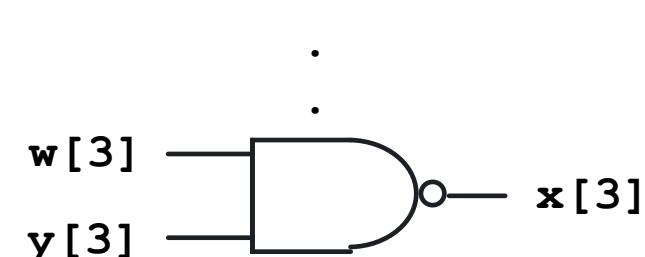
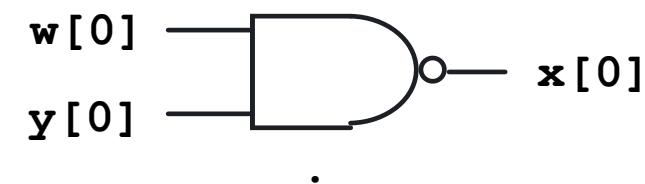
```
a = b ~& c;
```

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

Set and flip bits

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

implementation



BITWISE XNOR

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

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

xnor ($\sim \wedge$)

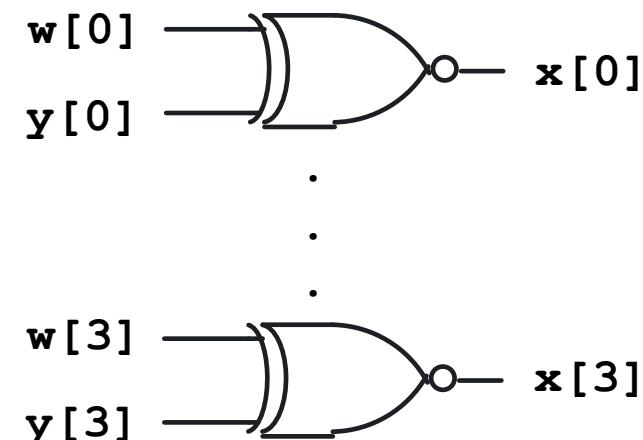
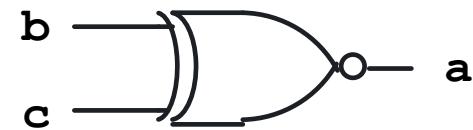
```
a = b  $\sim \wedge$  c;
```

```
x = w  $\sim \wedge$  y;
```

Equality check

$$\begin{array}{r} w \quad 0101 \\ (mask) \quad y \quad 0100 \\ \hline x \quad \textcolor{red}{1110} \\ 1\text{-equal} \end{array}$$

implementation



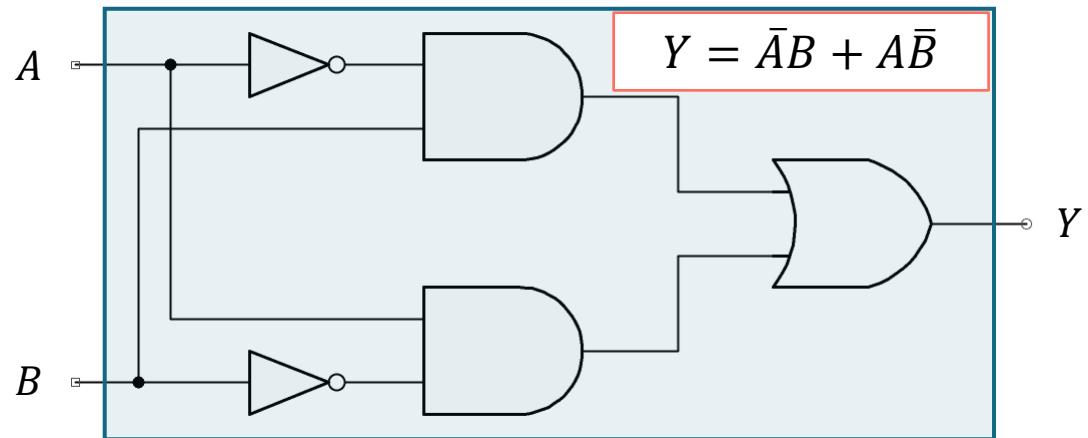
DATAFLOW MOELING



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.

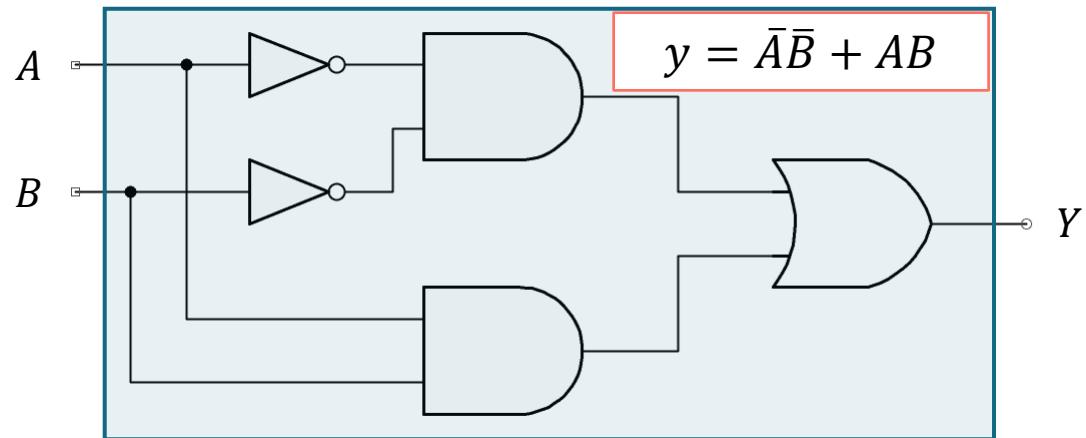
Dataflow modeling describes a circuit using continuous assignments and requires knowing the **logic expression** of the design.



XNOR MODULE

module name

xnor_gate



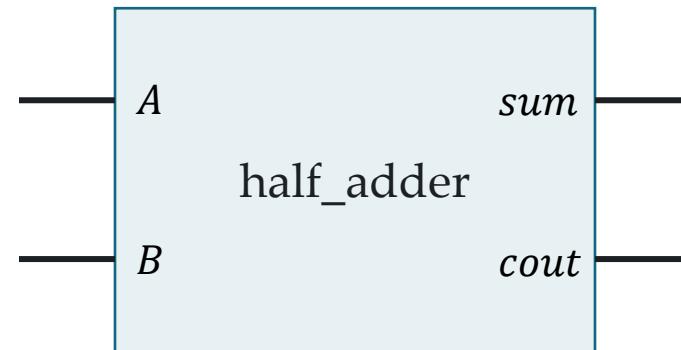
```
module xnor_gate(Y,A,B);  
  input A,B;  
  output Y;  
  
  assign Y = (~A & ~B) | (A & B);  
endmodule
```

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



$$cout = AB$$

$$sum = A \oplus B$$

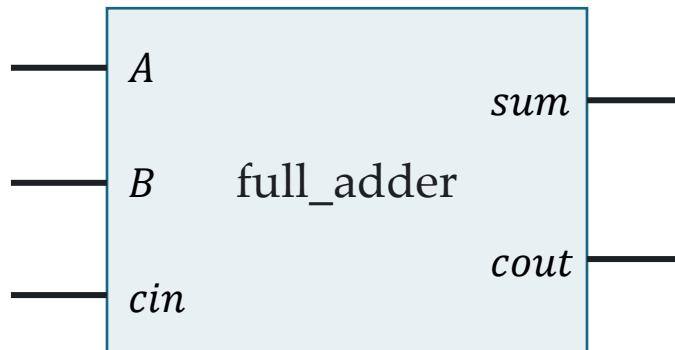


FULL-ADDER MODULE

module name

// Verilog code

half_adder



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

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



LABORATORY