GATE-LEVEL MODELING

VERILOG DESIGN STYLE

prepared by:

Gyro A. Madrona
Electronics Engineer

TOPIC OUTLINE

Logic Gate Primitives

Data Types

Bit and Bus

Gate-Level Modeling

LOGIC GATE PRIMITIVES

NUMBER REPRESENTATION

syntax

`size base number`

Decimal

`10'd8` → `00 0000 1000`

Binary

`8'b1100` → `0000 1100`

Hexadecimal

`9'hfe` → `0 1111 1110`

Octal

`7'o67` → `0 110 111`

Size matters

`7'hfe` → `0 1111 1110`

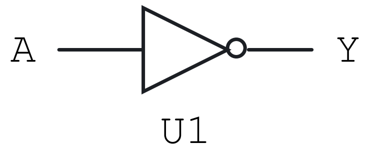
LOGIC GATE PRIMITIVES

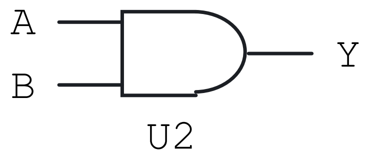
syntax

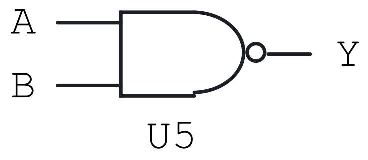
```
primitive instance_name(output,
                     input1,input2,...);
```

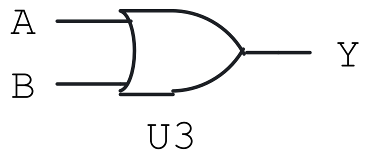
```
not U1(Y,A);

and U2(Y,A,B);          nand U5(Y,A,B);

or U3(Y,A,B);           nor U6(Y,A,B);

xor U4(Y,A,B);          xnor U7(Y,A,B);
```

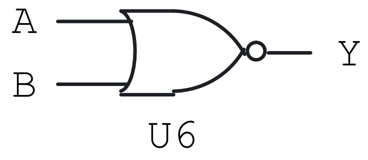
implementation

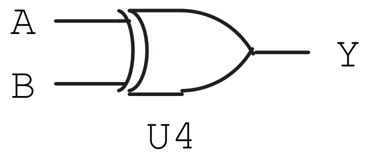


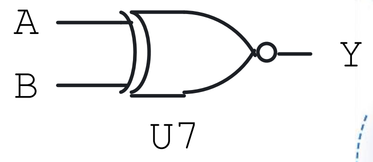










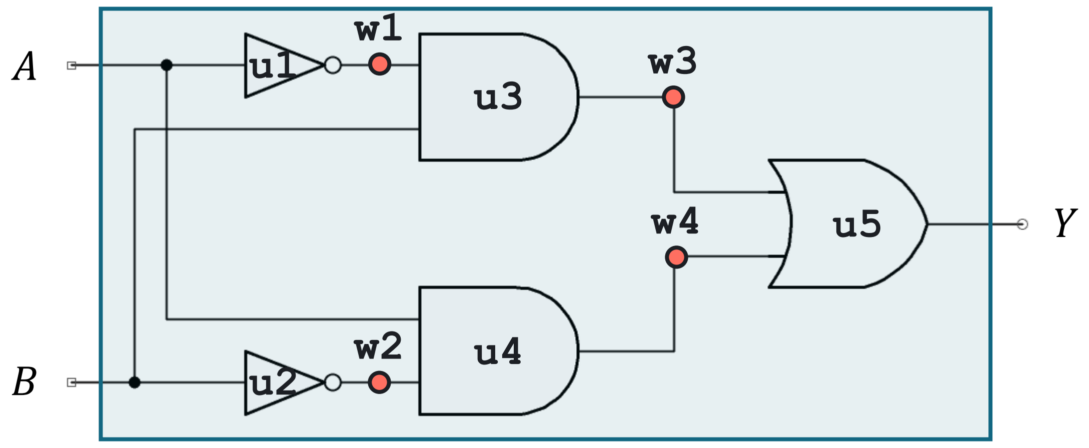


DATA TYPES

WIRE

module name

xor_gate



```
module xor_gate(Y,A,B);
    input A,B;
    output Y;
    wire w1,w2,w3,w4;

    or u5(Y,w3,w4);
    not u1(w1,A);
    not u2(w2,B);
    and u3(w3,w1,B);
    and u4(w4,A,w2);
endmodule
```

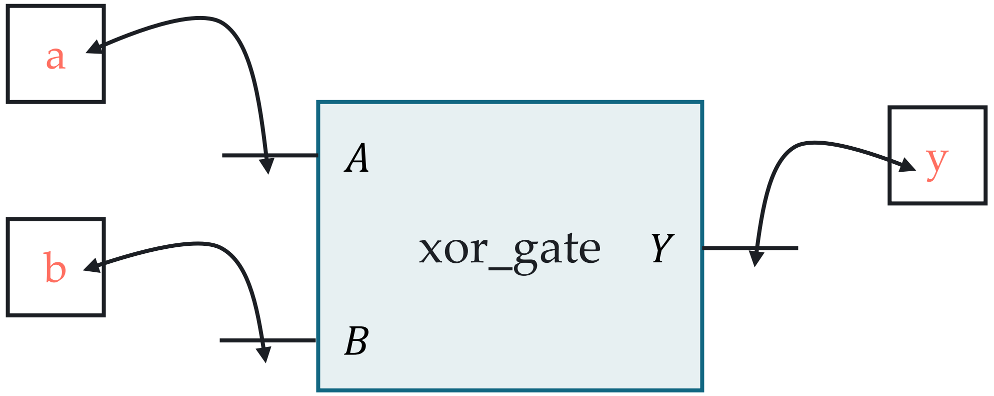
–it's all concurrent!

A **wire** represents a net (a logical connection) that reflects the value driven by its sources; but it **does not hold a value**.

REG

module name

testbench



DUT
(Design Under Test)

A **reg** represents a variable that can **store a value** assigned procedurally inside an **always** or **initial** block.

```
module testbench();
    reg a,b;
    wire y;

    xor_gate dut(y,a,b);

    initial begin
        a = 0; b = 0; #10;
        a = 0; b = 1; #10;
        a = 1; b = 0; #10;
        a = 1; b = 1; #10;
    end

endmodule
```

instantiate

sequential

BIT AND BUS

BIT AND BUS

Bit – a single binary signal

```
wire a; // 1-bit wire
```

```
reg b; // 1-bit reg
```

4-State Logic

1 – logic high

0 – logic low

x – unknown/don't care

z – high-impedance (not driven/open circuit)

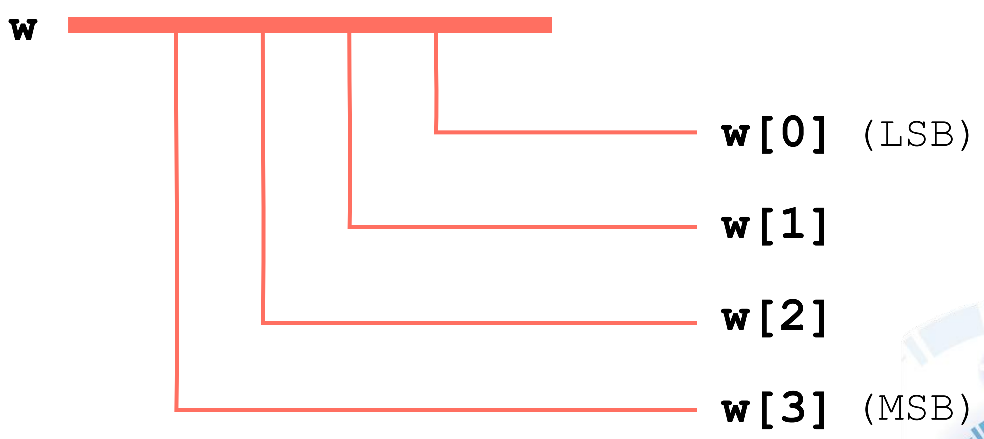
Bus – a multi-bit vector

```
wire [3:0] w; // 4-bit bus
```

```
reg [7:0] r; // 8-bit bus
```

syntax

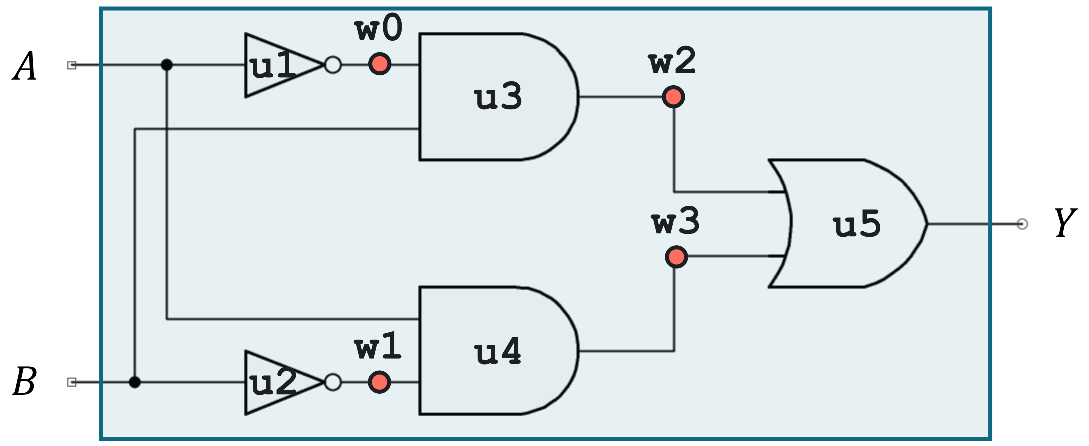
```
type [MSB:LSB] name;
```



USING A WIRE BUS

module name

xor_gate

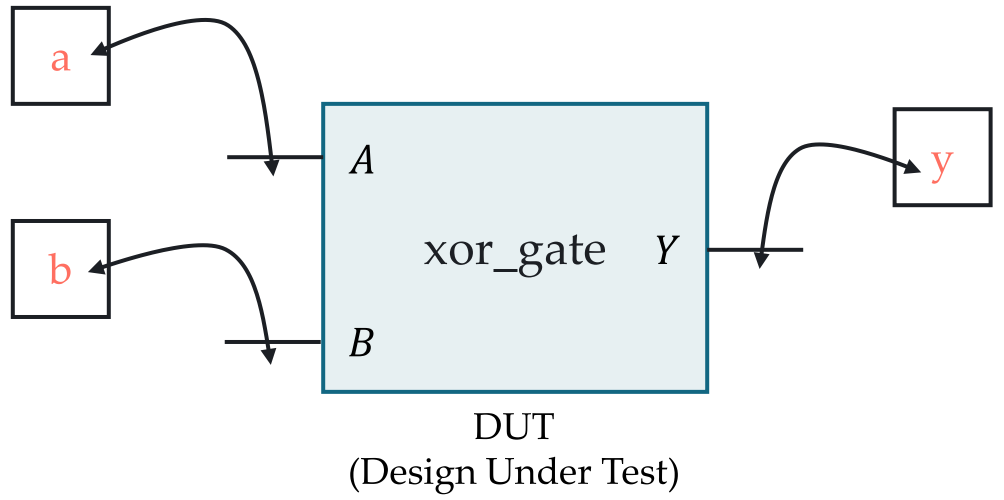


```
module xor_gate(Y,A,B);
    input A,B;
    output Y;
    wire [3:0] w;

    not u1(w[0],A);
    not u2(w[1],B);
    and u3(w[2],w[0],B);
    and u4(w[3],A,w[1]);
    or u5(Y,w[2],w[3]);
endmodule
```

TESTBENCH

module name

testbench

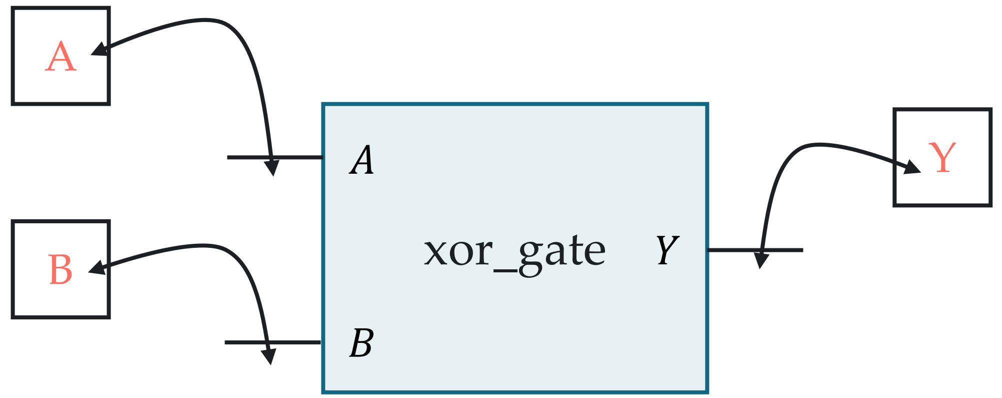


```
module testbench;
    reg a,b;
    wire y;

    xor_gate dut(.Y(y),.A(a),.B(b));

    initial begin
        a = 0; b = 0; #10;
        a = 0; b = 1; #10;
        a = 1; b = 0; #10;
        a = 1; b = 1; #10;
    end
endmodule
```

TESTBENCH

module name

testbench



DUT
(Design Under Test)

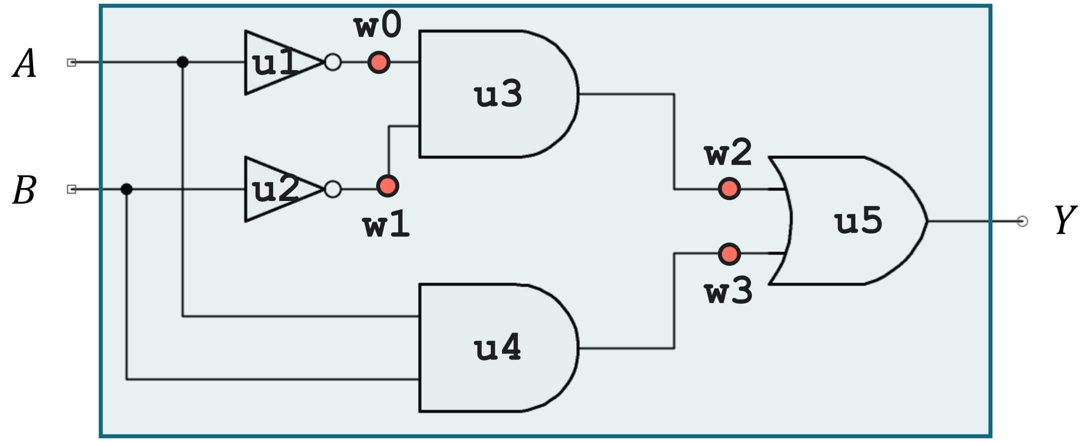
```
module testbench;
    reg A,B;
    wire Y;

    xor_gate dut(.Y(Y),.A(A),.B(B));

    initial begin
        a = 0; b = 0; #10;
        a = 0; b = 1; #10;
        a = 1; b = 0; #10;
        a = 1; b = 1; #10;
    end
endmodule
```

GATE-LEVEL MODELING

XNOR GATE MODULE

module name

xnor_gate



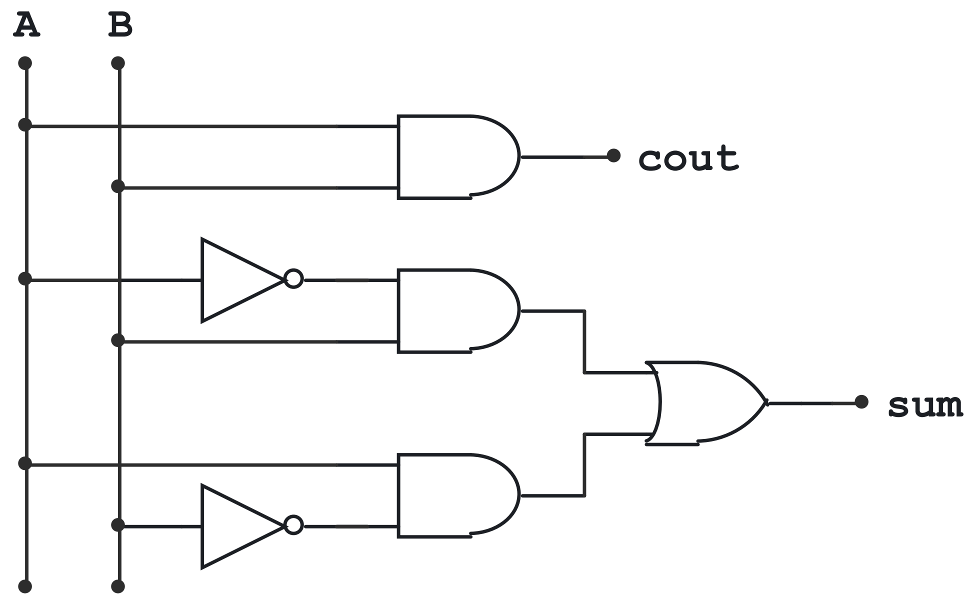
```
module xnor_gate(Y,A,B);
    input A,B;
    output Y;
    wire [3:0] w;

    not u1(w[0],A);
    not u2(w[1],B);
    and u3(w[0],w[1],w[2]);
    and u4(w[3],A,B);
    or u5(Y,w[2],w[3]);
endmodule
```

Gate-level modeling describes a circuit using built-in **logic gate primitives** (AND,OR,NOT,...etc.).

HALF-ADDER MODULE

```
module half_adder(sum,cout,A,B);




endmodule
```

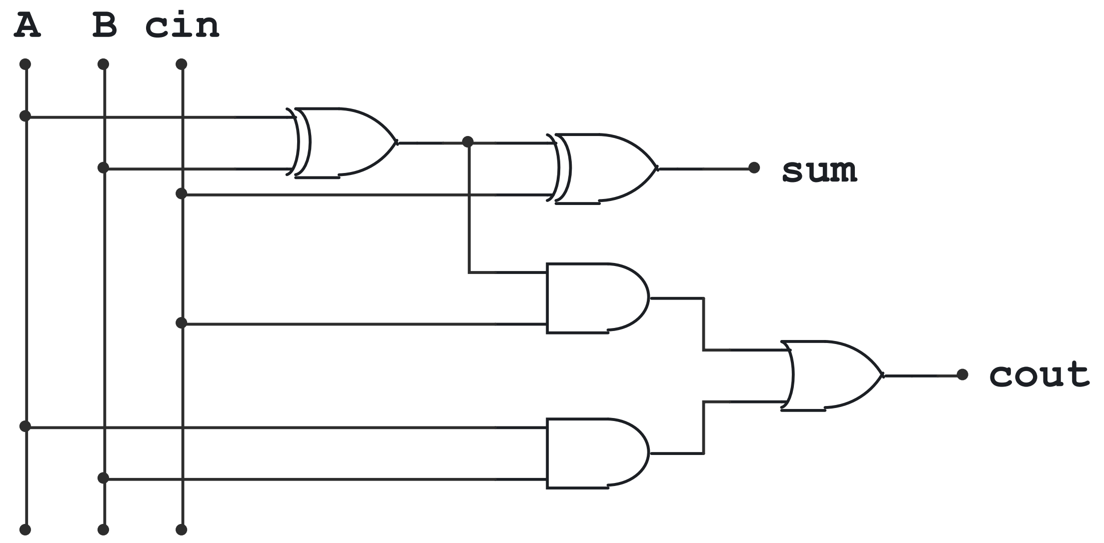
implementation



FULL-ADDER MODULE

```
module full_adder(sum,cout,A,B,cin);
```

implementation



```
endmodule
```

LABORATORY