


EE461 Verilog-HDL

Week 3 Operators & Gate Primitives

Alex Yang, Engineering School, NPU

Week 3 outlines - I

- Verilog Operators
  - Arithmetic Operators
  - Relational Operators
  - Equality Operators
  - Logical Operators
  - Bit-wise Operators
  - Reduction Operators
  - Shift Operators
  - Concatenation Operators
  - Replication Operators
  - Conditional Operators
  - Operator Precedence

Arithmetic Operators

- Binary(two operands): +, -, *, /, % (the modulus operator)
- Unary(one operand): +, - (This is used to specify the sign)
- Integer division truncates any fractional part
- The result of a modulus operation takes the sign of one of operand
- If any operand bit value is the unknown value x or z, then the entire result value is x
- Register data type is unsigned values (Negative numbers are stored in 2's complement number)

```
module testArith();
        reg[3:0]      a_r;
        reg[3:0]      b_r;
        wire[3:0]     c_w;

        assign c_w = a_r + b_r;

         initial begin
                  a_r = 4'b001x;
                  b_r = 4'b0001;
                  $display("c_w = %b", c_w);    //What is c_w's value? Ans: c_w = xxxx
                  #1 a_r = 4'b001z;
                  $display("c_w = %b", c_w);    //What is c_w's value? Ans: c_w = xxxx
                  #1 $finish;
         end
endmodule
```

Relational Operators

- Operators: a>b; a<b; a>=b; a<=b
- The value of (a>b) = 0, or 1, x, no z;
- If any bit in one operand is x or z, the result is x.

```
module testRel();
    reg[3:0]    a_r;
    reg[3:0]    b_r;
    wire[3:0]   c_w;


    initial begin
        a_r = 4'b001x;
        b_r = 4'b0001;
        $display("comp = %b", (a_r<=b_r));          //What is value of (a_r<=b_r)?
        #1 a_r = 4'b001z;
        $display("comp = %b", (a_r<=b_r));          //What is value of (a_r<=b_r)?
        #1 $finish;
    end
endmodule
```

Equality Operators

- Operators:

a === b	a equal to b, including x and z (Case equality); value = 0 or 1
a !== b	a not equal to b, including x and z (Case inequality); value = 0 or 1
a == b	a equal to b, result may be unknown (logical equality); value = 0 or 1 or X
a != b	a not equal to b, result may be unknown (logical equality) value = 0 or 1 or X

- Examples:

If a = 4'b01xz and b = 4'bzx10

(a === b) = ? // Ans: 0

(a !== b) = ? // Ans: 1

(a == b) = ? // Ans: X

(a != b) = ? // Ans: X

If a = 4'b01xz and b = 4'b01xz

(a === b) = ? // Ans: 1

(a !== b) = ? // Ans: 0

(a == b) = ? // Ans: X

(a != b) = ? // Ans: X

If a = 4'b01zz and b=4'b0100;

(a == b) = ? //Ans: 0

(a != b) = ? //Ans: 1

If a = 4'01zz and b=4'b01zz

(a == b) =? //Ans: X

(a !=b) = ? //Ans: X

Equality Operators

- summary:
  - If any oprerand has x bit, == or != value is x.
  - If any oprerand has z w/o x and Op1 is different from Op2, then (==)value is 0. Of course, (!=) value is 1. if Op1 is the same as Op2, then (==)/(!=) value is x.

"= = ": x <-> 0/1 = x;	x<-> x/z = x;
z <-> 0/1 = 0	z<-> x/z = x

Logical Operators

- Operators:

Operator	Desscription
!	logic negation; value = 0 or 1 or x (1 bit) , 3 values no z
&&	logical and
||	logical or

```
module testLog();
    reg[3:0]    a_r;
    reg[3:0]    b_r;
    reg         c_r;
    reg         d_r;

    initial begin
        a_r = 4'bx01x;      //Logically, (a_r) = 1;
        b_r = 4'b00xz;      //Logically, (b_r) = x;
        c_r = 1'bx;         //Logically, (c_r) = x;
        d_r = 1'bz;         //Logically, (d_r) = x;
        $display("!a_r = %b,!b_r= %b, !c_r=%b, !d_r=%b", (!a_r), (!b_r), (!c_r), (!d_r));
        #1 $finish;
    end
endmodule
```

summary:

Any one bit is 1 in a, (a) -> truth

Any one bit is x or z w/o 1 in a, (a) -> x

Results:

!a_r = 0,!b_r= x, !c_r=x, !d_r=x

Logical Operators

- Result of logical operation is ONE bit:

(logVariable1) && (logVariable2) = 0, 1, X
(logVariable1) = 0, 1, X not Z

(logVariable1) || (logVariable2) = 0, 1, X
(logVariable2) = 0, 1, X not Z

a && b	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

a || b	0	1	X
0	0	1	X
1	1	1	1
X	X	1	X

Questions:

1'bx && 2'bxz = ?
2'b00 && 2'b1z = ?

2'b0x || 1'bz = ?
2'b0z || 4'b01xz = ?

Bit-wise Operators

- Operators:

Operator	Description
~	Negation, number of bits of Value = number of operand (Calculate bit by bit)
&	and (bit by bit AND)
|	inclusive or (bit by bit OR)
^	exclusive or(bit by bit XOR)
^~ or ~^	exclusive nor (bit by bit XNOR)

~0 = 1 ~1 = 0 ~x = x ~z = x

A & B	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

Bit-wise Operators

- Operators:

A | B	0	1	X	Z
0	0	1	X	X
1	1	1	1	1
X	X	1	X	X
Z	X	1	X	X

A ^ B	0	1	X	Z
0	0	1	X	X
1	1	0	X	X
X	X	X	X	X
Z	X	X	X	X

Bit-wise Operators

- Operators:

A ^~ B:	0	1	X	Z
0	1	0	X	X
1	0	1	X	X
X	X	X	X	X
Z	X	X	X	X

Questions :

~4'b01xz = ? // Notice that ~z = x

4'b01xz & 4'bzx01 = ? 4'b01xz | 4'bzx01 = ?

4'b01xz ^ 4'bzx01 = ? 4'b01xz ^~ 4'bzx01 = ?

4'b01xz ^~ 2'bz1 = 4'b? //2'bz1 -> 4'b00z1

4'b01xz ^~ 2'bz1 = 2'b?

Reduction Operators

- **Operators:**

  There is only 1 operand, and result is just 1 bit.

Operator	Description
&	(Bit by bit within 1 operand) And; Result => 1 bit
~&	nand
|	or
~|	nor
^	xor
^~ or ~^	xnor

Question: & 4'b01xz = 1'b?

| 4'b01xz = 1'b?

^ 4'b01xz = 1'b?

Shift Operators

- Operators:

Operator	Description
<<	left shift, and 0 will be shifted in
>>	right shift, and 0 will be shifted in

```
module shiftOp();
        initial begin // Left shift
                $display (" 4'b1001 << 1 = %b", (4'b1001 << 1));
                $display (" 4'b10x1 << 1 = %b", (4'b10x1 << 1));
        $display (" 4'b10z1 << 1 = %b", (4'b10z1 << 1)); // Right shift
                $display (" 4'b1001 >> 1 = %b", (4'b1001 >> 1));
                $display (" 4'b10x1 >> 1 = %b", (4'b10x1 >> 1));
                $display (" 4'b10z1 >> 1 = %b", (4'b10z1 >> 1));
                #10 $finish;
        end
endmodule
```

Question: 4'b01xz << 3 = 4'b? 4'b10x1 >> 2 = 4'b?

4'b01xz << 1'bz = 4'b? //Ans: 4'bxxxx

4'b01xz << 2'b1x = 4'b? //Ans: 4'bxxxx

Concatenation Operators

- Operators:{…, …, …}
  - Example: if a = 2'b01, b=8'ha6, c = 4'b01xz,
    {a, b[3:0], c, 4'b1001} = ?
  - Unsized constant numbers are not allowed in concatenations.

Question: {'b01xz, 10} = ?

Ans: Get Warning.
'b01xz and 10: unsized number. Please don't do that

Replication Operators

- Operators:
  - {3{a}} => {a, a, a}
  - {b, {3{c, d}}} => {b, c, d, c, d, c, d}

Question: {3{10}} = ? {2'b00, 3{'b01xz,1'b1}} = ?

Ans: Still Get Warning.
'b01xz and 10: unsized number. Please don't do that

Conditional Operators

- Operators: var1 = (var2) ? expr1 : expr2

  out = (enable) ? data : 1'bz; => if(enable) out=data, else out= 1’bz

  Or more nested layers, like

  out = (enable==0)? 1’b0:
  (enable==1)? 1’b1:
  (enable==2)? 1’bx: 1’bz ;

  Important Info:

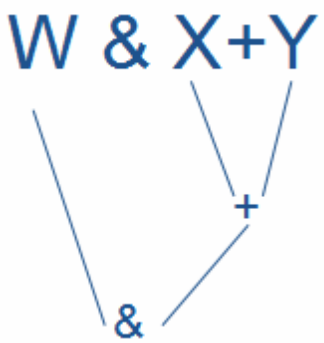
  1. LHS variable(like out) must be “wire” data type.
  2. Corresponding hardware: Mux
  3. If enable = x, or z, both expr1 and expr2 will be evaluated, and the result becomes the bit-by-bit combination of these two expressions.

  Example: out1 = X ? 4’b1100 : 4’b1ZX0 out1 = 4’b1XX0

Operator Precedence

- Precedence Table

Operator	Description
Unary, Multiply, Divide, Modulus	!, ~, *, /, %
Add, subtract, shift	+, - , <<, >>
Relation, Equality	<,>,<=,>=,==,!=,===,!==
Reduction	&, !&,^,^~,|,~|
Operator	&&, ||



Week 3 outlines - II

- Gate Level Modeling
  - Introduction
  - Gate Primitives
  - Transmission Gate Primitives
  - Switch Primitives
  - Logic Values and signal strengths
  - Verilog strength Levels
  - Designing Using Primitives
  - Gate and switch delays
  - N-Input Primitives
  - N-output Primitives

Introduction

- Introduction
  - Verilog provides built-in primitives like gates, transmission gates, and switches. These are rarely used in design (RTL Coding), but are used for modeling the ASIC/FPGA cells in post synthesis world. Those cells are then used for gate level simulation, or what is called as SDF (Standard Delay Format File) simulation with netlist.
  - Also the output netlist format from the synthesis tool, which is imported into the place and route tool, is in Verilog gate level primitives.
  - Note : RTL engineers still may use gate level primitivies or ASIC library cells in RTL when using IO CELLs.

Gate Primitives

- Gates:

Gate	Description
and	N-input AND gate
nand	N-input NAND gate
or	N-input OR gate
nor	N-input NOR gate
xor	N-input XOR gate
xnor	N-input XNOR gate

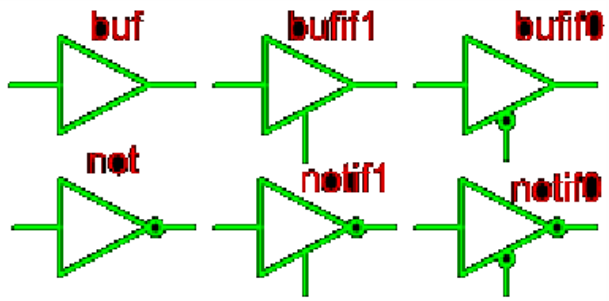
  - Each gate in above table has just one output and multiple inputs. The 1st one in the list of gate terminals is an output and others are inputs.

Gate Primitives

```
module gates();
        wire out0_w;
        wire out1_w;
        wire out2_w;
        reg  in1_r,in2_r,in3_r,in4_r;

        not uNot (out0_w, in1_r);
        and uAnd (out1_w, in1_r, in2_r, in3_r, in4_r);
        xor uXor (out2_w, in1_r, in2_r, in3_r);

        initial begin
                $monitor("in1_r=%b in2_r=%b in3_r=%b in4_r=%b out_0_w=%b out1_w=%b
        out_o2_w=%b",in1_r,in2_r,in3_r,in4_r,out0_w,out1_w,out2_w);
                in1_r = 0;
                in2_r = 0;
                in3_r = 0;
                in4_r = 0;
                #1 in1_r = 1;
                #1 in2_r = 1;
                #1 in3_r = 1;
                #1 in4_r = 1;
                #1 $finish;
                end
endmodule
```

Transmission Gate Primitives

- Logic symbols of Gate Primitives



Gate	Description
not	N-out_put (more than one) inverter
buf	N-out_put (more than one) buffer.
bufif0	Tri-state buffer, Active low en.
bufif1	Tri-state buffer, Active high en.
notif0	Tristate inverter, Low en.
notif1	Tristate inverter, High en.

Example: bufif0 U1 (data_bus, in, data_enable_low);
syntax: keyword unique_name (inout1, inout2, control);

Transmission Gate Primitives

- Logic symbols of Gate Primitives

buf	
inputs	outputs
0	0
1	1
x	x
z	x

bufif0		CONTROL			
		0	1	x	z
DATA	0	0	z	L	L
	1	1	z	H	H
	x	x	z	x	x
	z	x	z	x	x

bufif1		CONTROL			
		0	1	x	z
DATA	0	z	0	L	L
	1	z	1	H	H
	x	z	x	x	x
	z	z	x	x	x

not	
inputs	outputs
0	1
1	0
x	x
z	x

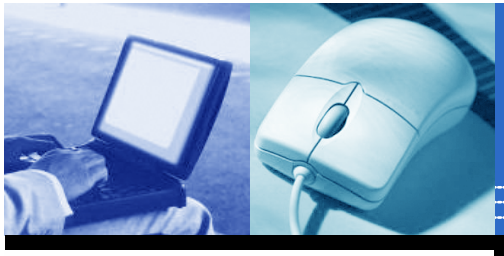
notif0		CONTROL			
		0	1	x	z
DATA	0	1	z	H	H
	1	0	z	L	L
	x	x	z	x	x
	z	x	z	x	x

notif1		CONTROL			
		0	1	x	z
DATA	0	z	1	H	H
	1	z	0	L	L
	x	z	x	x	x
	z	z	x	x	x

Note: L -> 0 or Z; H -> 1 or Z

Transmission Gate Primitives

```
module testTrans();
    reg data_enable_low, in;
    wire data_bus;

    bufif0 U1(data_bus,in, data_enable_low);

    initial begin
        $monitor("@%g in=%b data_enable_low=%b data_bus=%b", $time, in,
                 data_enable_low, data_bus);
        data_enable_low = 0;
        in = 0;
        #4 data_enable_low = 1;
        #4 data_enable_low = 1'bx;          //Ans: data_bus = 1'bx;
        #4 data_enable_low = 1'bz;          //Ans: data_bus = 1'bx;
        #8 $finish;
    end

    always #2 in = ~in;
endmodule
```



Switch Primitives

- MOS Switches
  - nmos, rnmos, pmos, rpmos

  Note: rmos/rpmos: resisitive transistors have significantly higher impedance between their sources and drains
- CMOS Switch
  - cmos rcmos

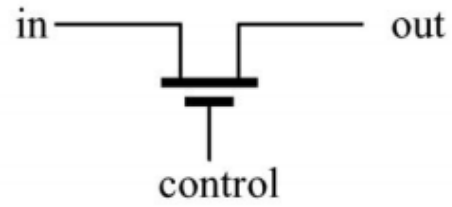
  Note: rcmos: higher impedance between input and output
- Bidirectional Switches
  - tran, tranif1, tranif0, rtran, rtranif1, rtranif0

    rxxxx: higher impedance between input and output
- Pullup & Pulldown

Switch Primitives

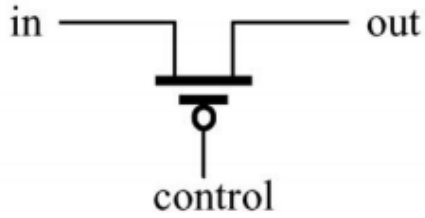
- MOS Switches
  - nmos, rnmos, pmos, rpmos

    r: higher impedance between in and out



nmos		control			
		0	1	x	z
in	0	z	0	L	L
	1	z	1	H	H
	x	z	x	x	x
	z	z	z	z	z

(a) nMOS switch



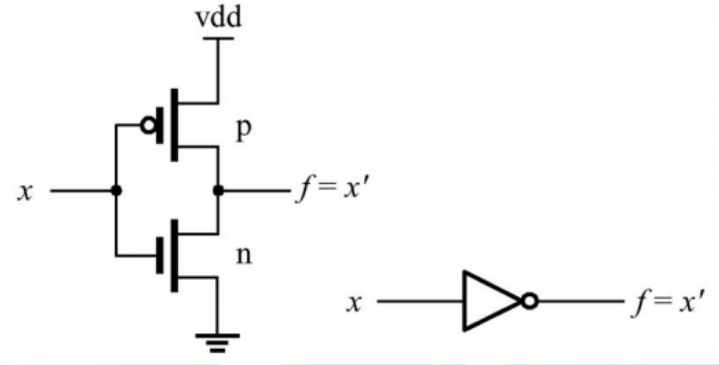
pmos		control			
		0	1	x	z
in	0	0	z	L	L
	1	1	z	H	H
	x	x	z	x	x
	z	z	z	z	z

(b) pMOS switch

Switch Primitives

- MOS Switches
  - switch_name [instance_name] (output, input, control);

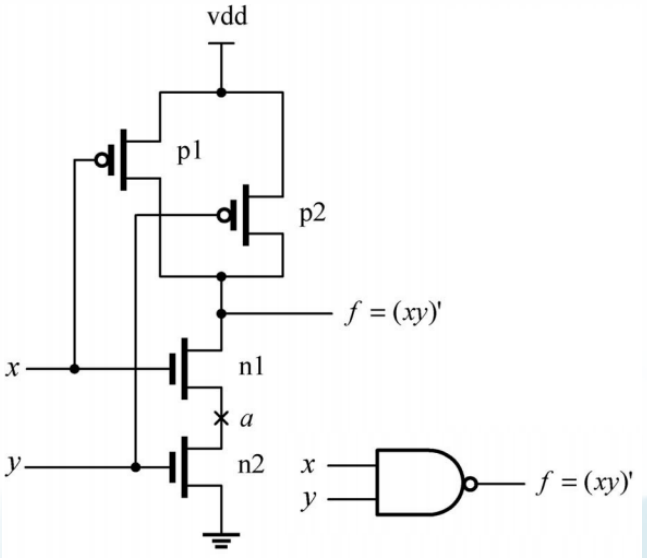
```
module myNot (input x_i, output f_o);
   // internal declaration
   supply1 vdd;
   supply0 gnd;
   // NOT gate body
   pmos uPmos (f_o, vdd, x_i);
   nmos uNmos (f_o, gnd, x_i);
endmodule
```



Switch Primitives

- MOS switches

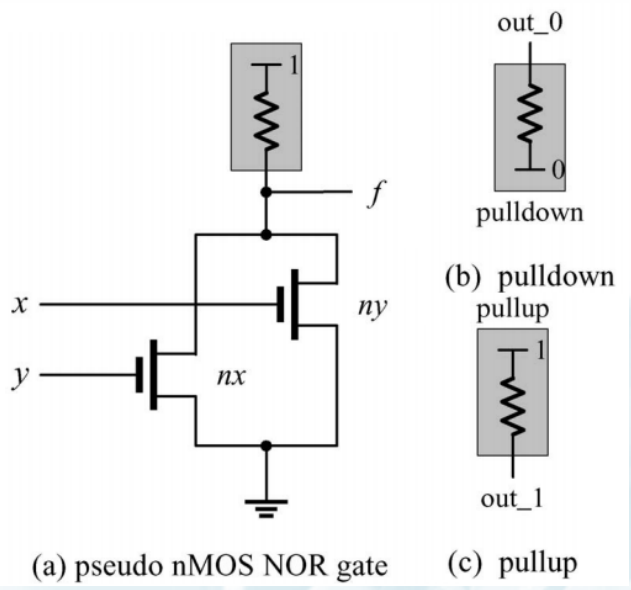
```
module myNand (input x_i, y_i, output f_o);
    supply1 vdd;
    supply0 gnd;
    wire a_w;

    // NAND gate body
    pmos uPmos1 (f_o, vdd, x_i);
    pmos uPmos2 (f_o, vdd, y_i);
    nmos uNmos1 (f_o, a_w, x_i);
    nmos uNmos2 (a_w, gnd, y_i);
endmodule
```



Switch Primitives

- MOS Switches

```
module myPseudoNor(input x_i, y_i, output f_o);
    supply0 gnd;
    // Pseudo nMOs gate body

    nmos uNmos1 (f_o, gnd, x_i);
    nmos uNmos2 (f_o, gnd, y_i);
    pullup uPullup (f_o);

endmodule
```

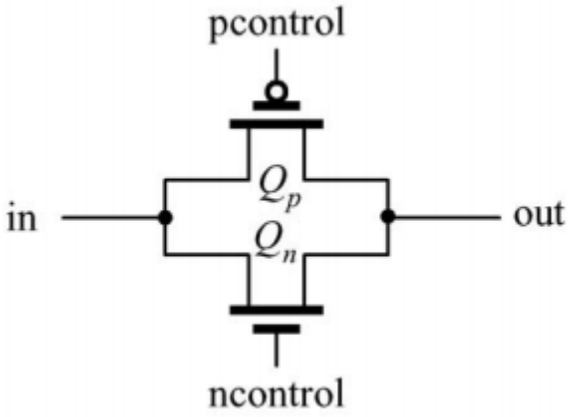


Switch Primitives

- CMOS Siwitches
  - cmos rcmos

cmos [instance_name]

(output, input, ncontrol, pcontrol);

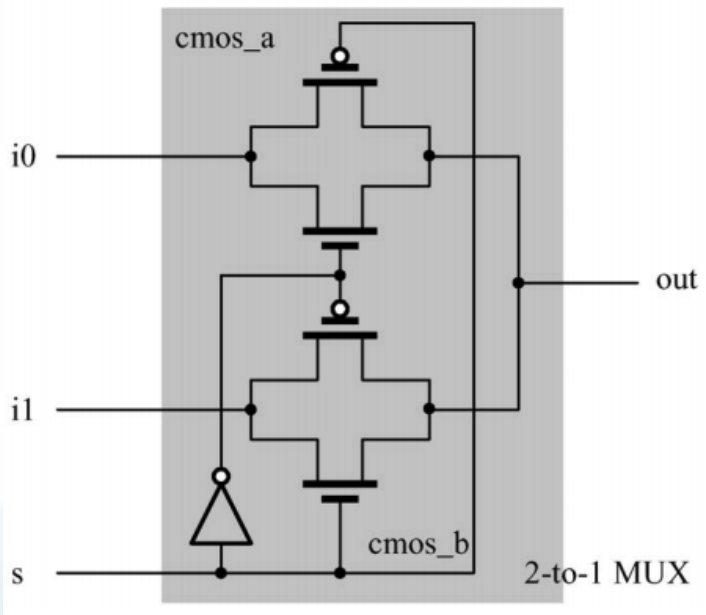


control		data			
n	p	0	1	x	z
0	0	0	1	x	z
0	1	z	z	z	z
0	x	L	H	x	z
0	z	L	H	x	z
1	0	0	1	x	z
1	1	0	1	x	z
1	x	0	1	x	z
1	z	0	1	x	z
x	0	0	1	x	z
x	1	L	H	x	z
x	x	L	H	x	z
x	z	L	H	x	z
z	0	0	1	x	z
z	1	L	H	x	z
z	x	L	H	x	z
z	z	L	H	x	z

Switch Primitives

- CMOS Switches

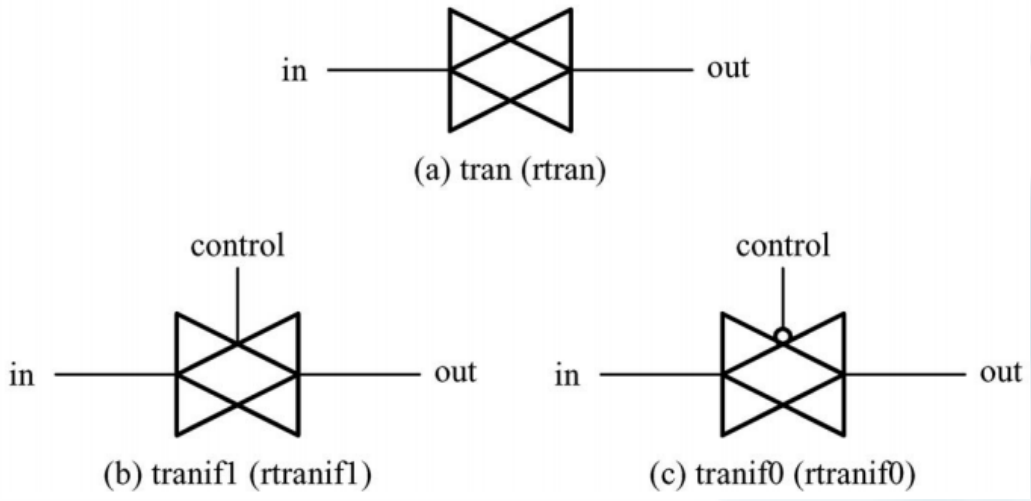
```
module myMux (out_o, s_i, i0_i, i1_i);
    output out_o;
    input s_i, i0_i, i1_i;
    //internal wire
    wire sbar_w; //complement of s_i

    not (sbar_w, s_i);
    //instantiate cmos switches
    cmos (out_o, i0_i, sbar_w, s_i);
    cmos (out_o, i1_i, s_i, sbar_w);
    //Instance name is optional

endmodule
```



Switch Primitives

- Bidirectional Switches
  - tran, tranif1, tranif0, rtran, rtranif1, rtranif0

tran [instance_name] (in, out);
tranif0 [instance_name] (in, out, control);
tranif1 [instance_name] (in, out, control);



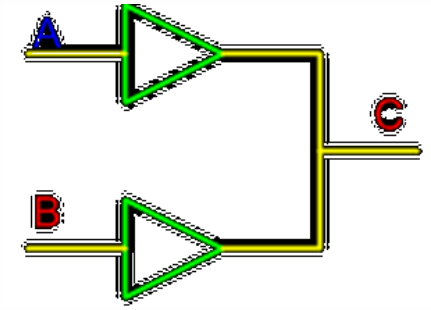
Logic Values & signal strengths

- Logic Value in Verilog

Logic Value	Description
0	zero, low, false
1	one, high, true
z or Z	high impedance, floating
x or X	unknown, uninitialized, contention

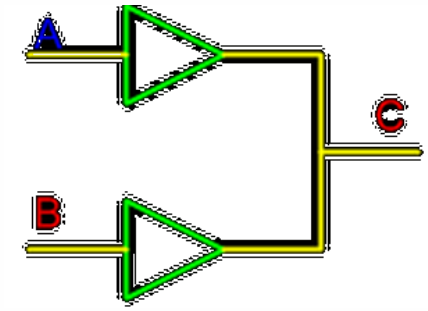
Verilog strength Levels

- Different Strength Signals to Drive Device



Two buffers that has output
A : Pull 1
B : Supply 0

Since supply 0 is stronger then pull 1,
Output C takes value of B.



Two buffers that has output
A : Supply 1
B : Large 1

Since Supply 1 is stronger then Large 1,
Output C takes the value of A

Verilog strength Levels

- Signal Strength Lookup Table

Strength	Strength0	Strength1	Type	Degree
supply	supply0	supply1	driving	strongest
strong	strong0	strong1	driving	
pull	pull0	pull1	driving	
large	large0	large1	storage	
weak	weak0	weak1	driving	
medium	medium0	medium1	storage	
small	small0	small1	storage	
highz	highz0	highz1	high Z	weakest

Verilog strength Levels

- Signal Strength
  - Drive Strength

    Example: and (strong1, weak0) b(o, i1, i2);
  - Charge Strength (Only for "trireg")

    Example: trireg (medium) t;
- Two Different Strength Signals to Drive

  buf (strong1, weak0) g1 (y, a);

  buf (pull1, supply0) g2 (y, b);

When a=0 & b=0, => y=0(weak0 in g1) y=0(supply0 in g2), final y=0
When a=0 & b=1, => y=0(weak0 in g1) y=1(pull1 in g2), final y=1
When a=1 & b=0, => y=1(strong1 in g1) y=0(supply0 in g2), final y=0
When a=1 & b=1, => y=1(strong1 in g1) y=1(pull1 in g2), final y=1

Verilog strength Levels

- Two Different Strength Signals to Drive

```
buf (strong1, weak0) g1 (y, a);
buf (weak1, strong0) g2 (y, b);
```

When a=1 & b=0, => y=1(strong1 in g1) y=0(strong0 in g2), final y=x

Note: If two drivers of a net have the same strength but different values then signal value will be unknown

```
bufif0 (strong1, weak0) g1 (y, i1, ctrl);
bufif0 (strong1, weak0) g2 (y, i2, ctrl);
```

If ctrl = x, i1 = 0, and i2 = 1 then y will be x with 36X strength,

because g1 will set y to L with weak strength (WeL - 3) and g2 will set y to H with strong strength (StH - 6).

3 and 6: number of strength level in the following table

Verilog strength Levels

- Strength Table

Strength	Value	Value displayed by display tasks
supply	7	Su
strong	6	St
pull	5	Pu
large	4	La
weak	3	We
medium	2	Me
small	1	Sm
highz	0	HiZ

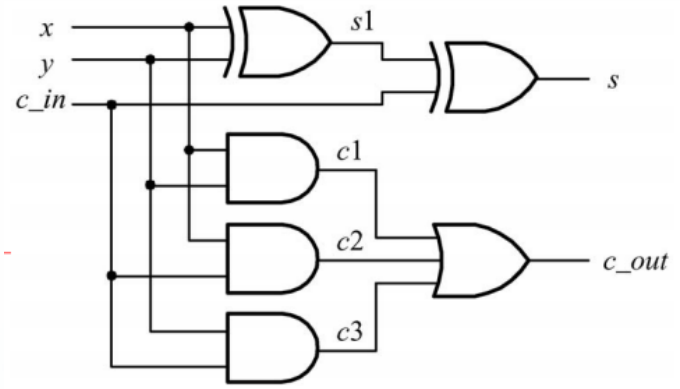
Verilog strength Levels

```
module testStr;
        wire    y_w;
        reg     i1_r, i2_r, ctrl_r, a_r, b_r;

        buf (strong1, weak0) g1 (y_w, a_r);
        buf (pull1, supply0)   g2 (y_w, b_r);

        //bufif0 (strong1, weak0) g1 (y_w, i1_r, ctrl_r);
        //bufif0 (strong1, weak0) g2 (y_w, i2_r, ctrl_r);

        initial begin
            /* i1_r=1'b0;
               i2_r=1'b1;
               ctrl_r=1'b1;
               $display("strength = %v", y_w);
             */
               a_r=1'b0;
               b_r=1'b1;
               $display("strength = %v", y_w);
        end
endmodule
```

//%v - signal strength

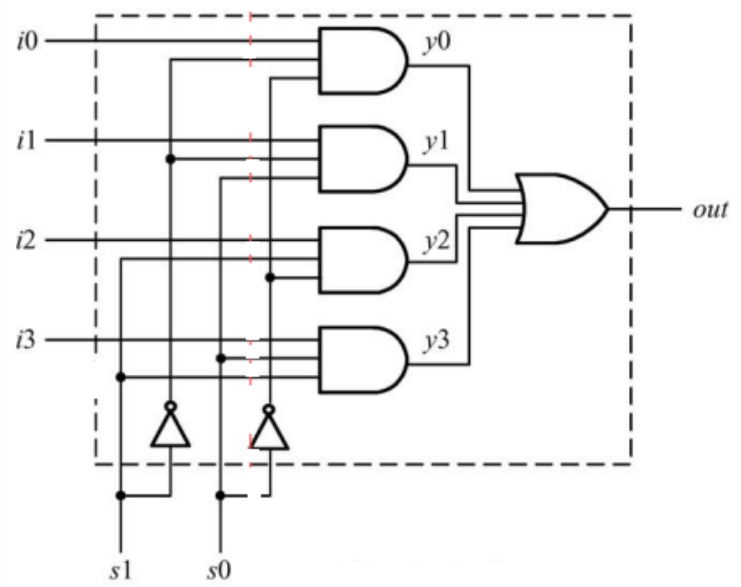
Designing Using Primitives

```
module oneBitFA(x_i, y_i, cin_i, s_o, co_o);
    // I/O port declarations
    input x_i, y_i, ci_i;
    output s_o, co_o;
    wire s1_w, c1_w, c2_w, c3_w;
    // Structural modeling of the 1-bit full adder.
    xor uXor1(s1_w, x_i, y_i);
    // compute sum.
    xor uXor2(s_o, s1_w, ci_i);
    and uAnd1(c1_w, x_i, y_i);
    // compute carry out.
    and uAnd2(c2_w, x_i, ci_i);
    and uAnd3(c3_w, y_i, ci_i);
    or uOr(co_o, c1_w, c2_w, c3_w);
endmodule
```

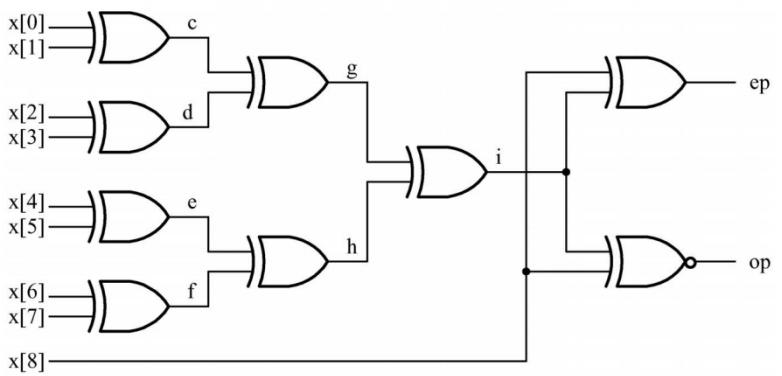


Designing Using Primitives

- More Design Examples



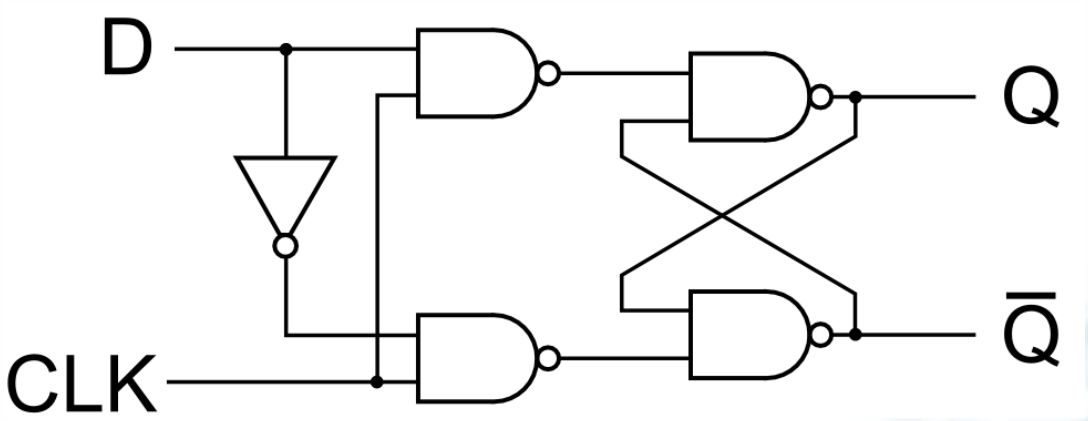
4-1 Mux



9-bit Parity Generator

Designing Using Primitives

- More Design Example - D Flip Flop



Gate delays

- Delay Specifications
  - Devices: AND, NAND, OR, NOR, XOR, XNOR, BUF, NOT.
    - Specify no delay

      e.g. and u0 (output, input, …);
    - Specify one delay only

      e.g. and #(5) u0 (output, input, …);

      Note: rise/fall time/(-> x) = 5;
    - Specify two delays

      e.g. and #(5, 10) u0 (output, input, …);

      Note: rise = 5, fall = 10, (->x) = 5

      (Select smaller one) ;

Buffer with ctrl delays

- Delay Specifications
  - Devices: bufif0, bufif1, notif0, notif1.
    - Specify no delay

      e.g. bufif0 u0 (output, input, …);
    - Specify one delay only

      e.g. bufif0 #(5) u0 (output, input, …);

      Note: rise/fall time/(-> x or ->L or ->H) = 5;
    - Specify two delays

      e.g. bufif0 #(5, 10) u0 (output, input, …);

      Note: rise = 5, fall = 10, (->x or ->L or ->H & ->z) = 5

      (Select smaller one) ;
    - Specify three delays

      e.g. bufif0 #(5, 10, 15) u0 (output, input, …);

      Note: rise = 5, fall = 10, (->z) = 15 , (->x or ->L or - >H) =5

      (Select smallest one) ;

Switch delays

- Delay Specifications
  - Devices: pmos, nmos, rpmos, rnmos, cmos, rcmos
    - Specify no delay

      e.g. pmos u0 (output, input, …);
    - Specify one delay only

      e.g. pmos #(5) u0 (output, input, …);

      Note: rise/fall time/(-> x or ->L or ->H) = 5;
    - Specify two delays

      e.g. pmos #(5, 10) u0 (output, input, …);

      Note: rise = 5, fall = 10, (->x or ->L or ->H & ->z) = 5 (Select smaller one) ;
    - Specify three delays

      e.g. pmos #(5, 10, 15) u0 (output, input, …);

      Note: rise = 5, fall = 10, (->z) = 15 , (->x or ->L or ->H) =5 (Select smallest one) ;

Bidirectional switch delays

- Delay Specifications
  - Devices: tran, tranif0, tranif1, rtran, rtranif0, rtranif1
    - Specify no delay

      e.g. tran u0 (output, input, …);

      Note: tran/rtran can't have delay
    - Specify one delay only

      e.g. tranif0 #(5) u0 (output, input, …);

      Note: turnON/turnOFF = 5;
    - Specify two delays

      e.g. tranif0 #(5, 10) u0 (output, input, …);

      Note: turnON = 5, turnOFF = 10;

Notice that all those delays discussed here are just for simulation, not real exact delays in P&R (Place and Route).

Pullup/Pulldown delays

- Delay Specifications
  - Devices: pullup, pulldown
    - Specify no delay

    e.g. pullup (strong1, strong0)(neta),(netb);

    Note: pullup is driving two wires, neta and netb to 1

pullup & pulldown can't take delays

N-Input Primitives

- AND, NAND, OR, NOR, XOR, and XNOR
  - 1 output and more inputs

```
and uAnd1 (out1, in1, in2);       //Two input AND gate
and uAnd2 (out2, in1, in2, in3, in4);// four input AND gate
xnor uAnor1 (out3, in1, in2, in3);// three input XNOR gate
```

N-output Primitives

- BUF & NOT
  - The outputs are the first terminals listed
  - The last terminal is the single input.

```
buf uBuf0 (out,in);          // one output Buffer gate
buf uBuf1 (out_0, out_1, out_2, out_3, in);
// four output Buffer gate

not uNot0 (out_a, out_b, out_c, in);// three output Invertor gate
```