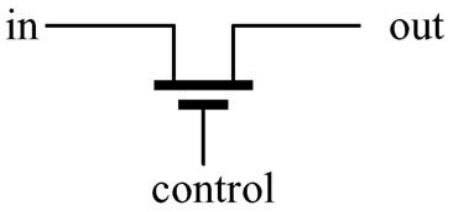
Switch Level Modeling in Verilog

Sayed Amirhossein Mirhosseini

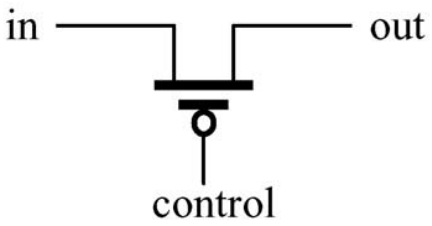
The nmos and pmos Switches

- To instantiate switch elements:
  - switch_name [*instance_name*] (output, input, control);
- The instance_name is optional



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

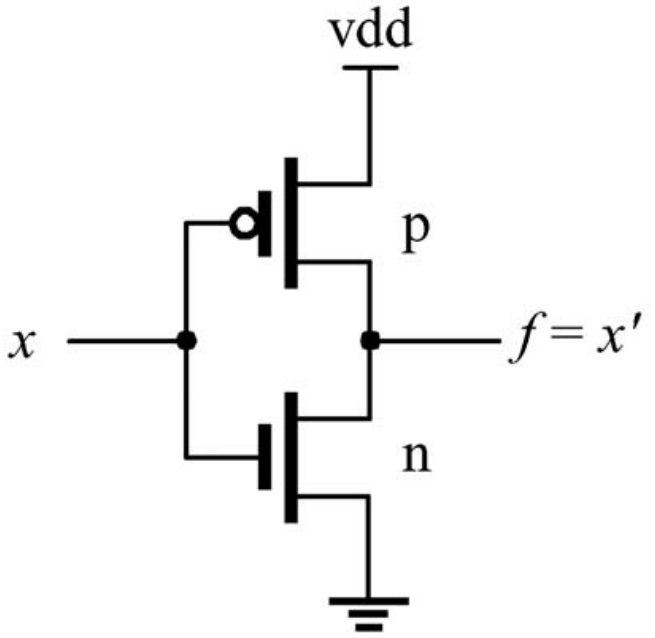


pmos		control			
		0	1	x	z
in	0	0	z	L	L
	1	1	z	H	H
	x	x	z	x	x
	z	z	z	z	z

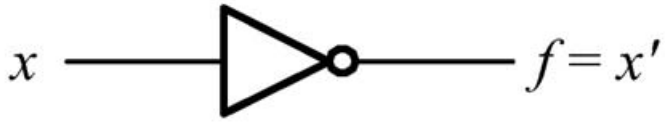
(b) pMOS switch

Example : The CMOS Inverter

```
module my_not (input x, output f);
// internal declaration
supply1 vdd;
supply0 gnd;
// NOT gate body
pmos p1 (f, vdd, x);
nmos n1 (f, gnd, x);
endmodule
```



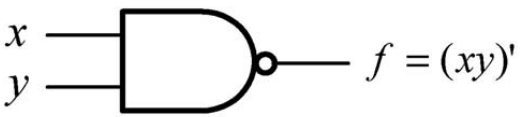
(a) Circuit



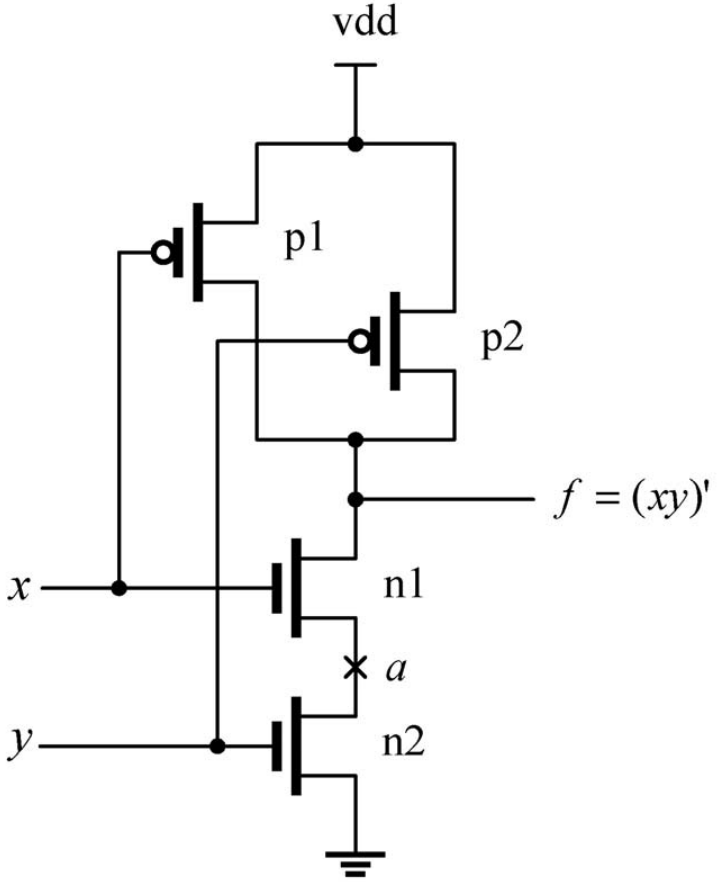
(b) Logic symbol

Example : CMOS NAND Gate

```
module my_nand (input x, y, output f);
supply1 vdd;
supply0 gnd;
wire a;
// NAND gate body
pmos p1 (f, vdd, x);
pmos p2 (f, vdd, y);
nmos n1 (f, a, x);
nmos n2 (a, gnd, y);
endmodule
```

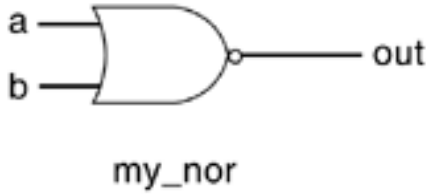


(b) Logic symbol

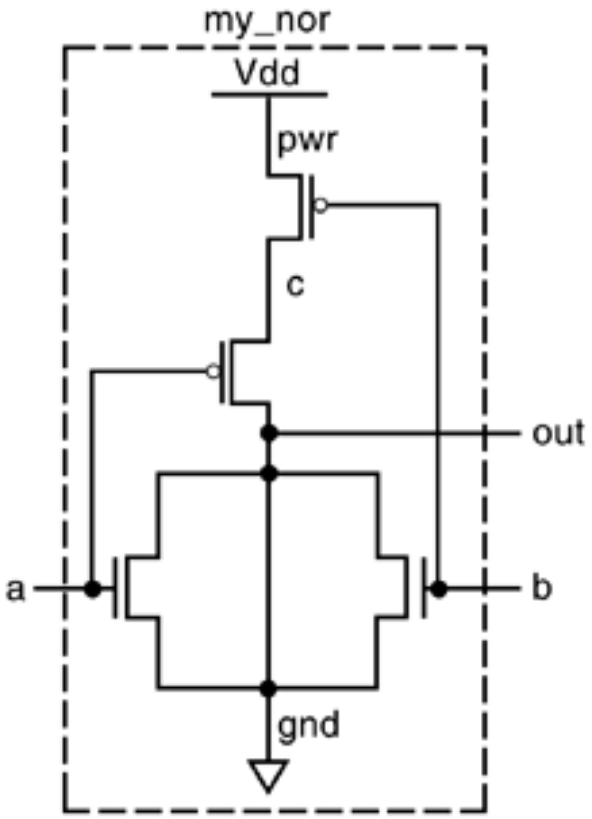


(a) Circuit

What about a NOR gate ?



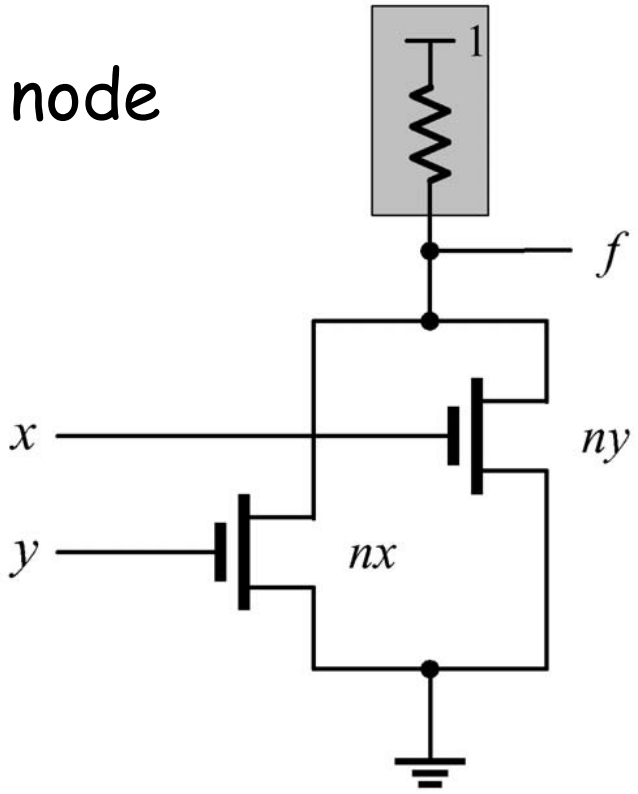
Any way to reduce the Number of switches?



CMOS NOR Gate

Example : An NMOS NOR Gate

```
module my_nor(input x, y, output f);
supply0 gnd;
tri1 f;          //To pullup the f node
// The nMOS gate body
nmos nx (f, gnd, x);
nmos ny (f, gnd, y);
endmodule
```

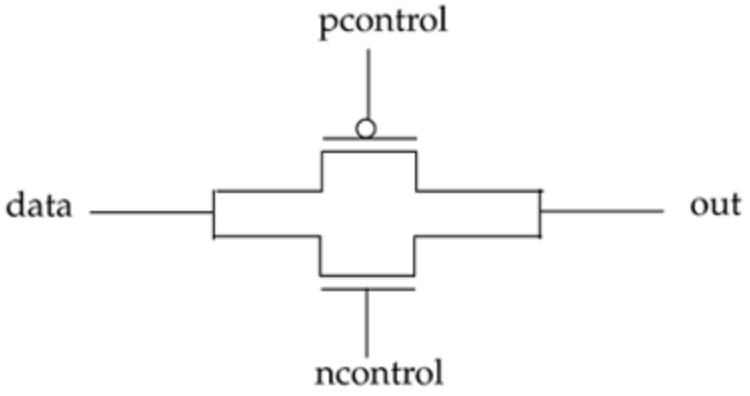


NMOS NOR Gate

CMOS Switch

- Is designed to propagate both 0 and 1 well
  - Nmos is only able to propagate 0 well
  - Pmos is only able to propagate 1 well
- To instantiate CMOS switches :
  - cmos [*instance_name*] (output, data, ncontrol, pcontrol);
- The instance_name is optional
- Exactly equivalent to the following :

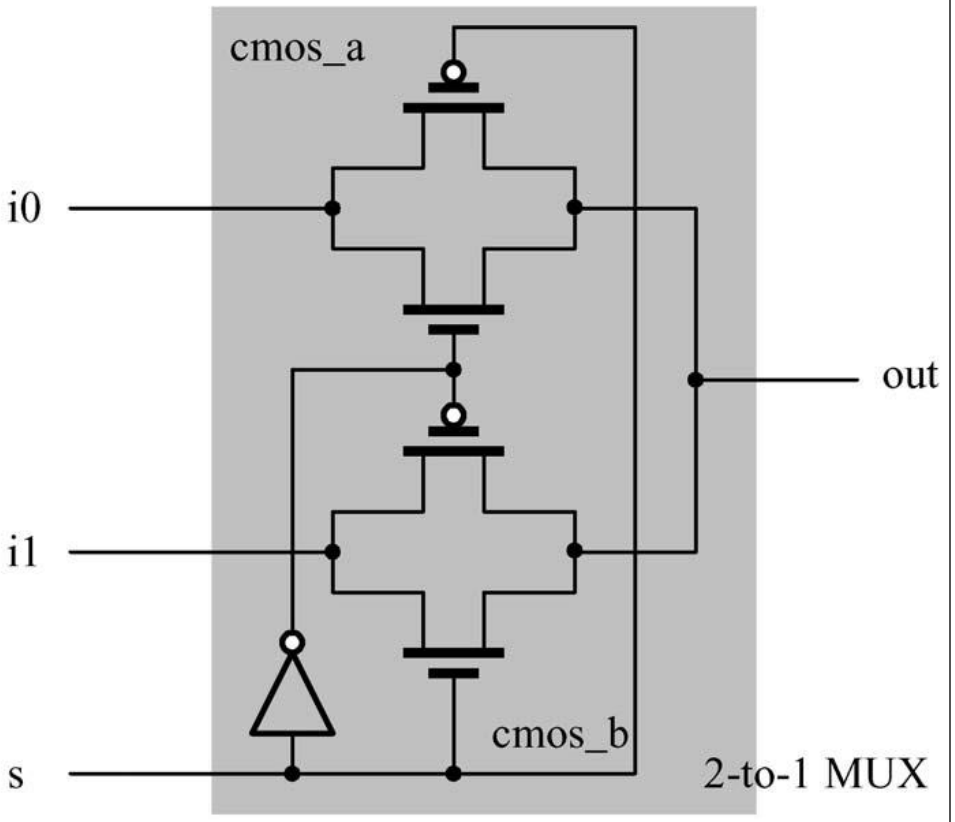
```
nmos (out, data, ncontrol);
pmos (out, data, pcontrol);
```



CMOS

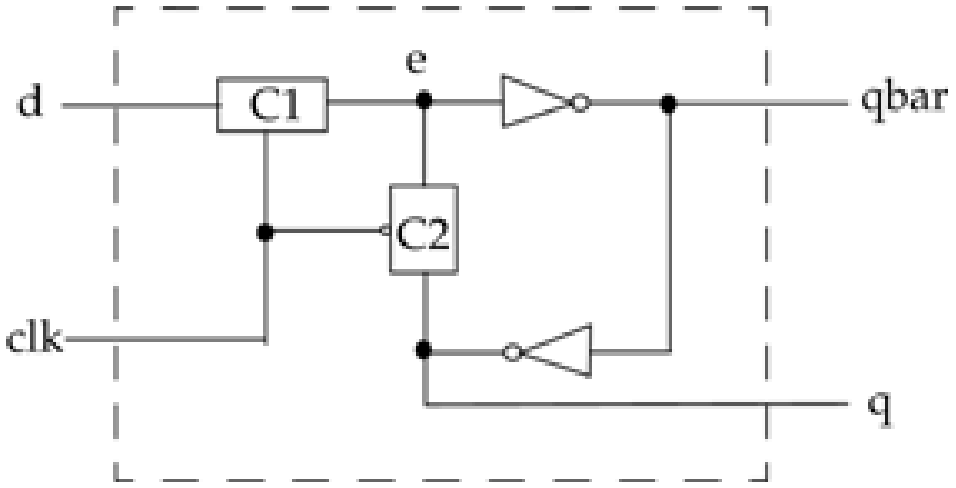
Example : A 2-to-1 Multiplexer

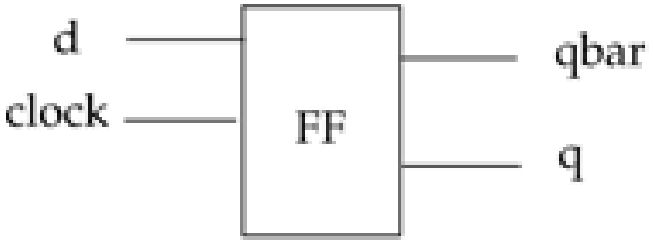
```
module my_mux (out, s, i0, i1);
output out;
input s, i0, i1;
//internal wire
wire sbar;
not (sbar, s);
//cmos switches
cmos (out, i0, sbar, s);
cmos (out, i1, s, sbar);
endmodule
```



Example : CMOS Flipflop

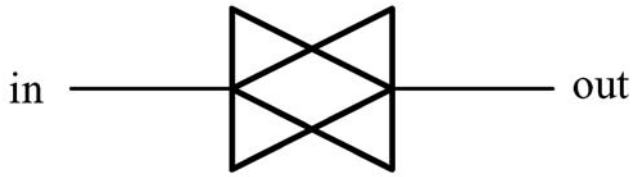
```
module cff ( q, qbar, d, clk);
output q, qbar;
input d, clk;
//internal nets
wire e,nclk;
//the clock inverter
my_not nt(nclk, clk);
//the CMOS switches
cmos  (e, d, clk, nclk); //switch C1 closed (e = d) when clk = 1.
cmos  (e, q, nclk, clk); //switch C2 closed (e = q) when clk = 0.
//instantiate the inverters
my_not nt1(qbar, e);
my_not nt2(q, qbar);
endmodule
```



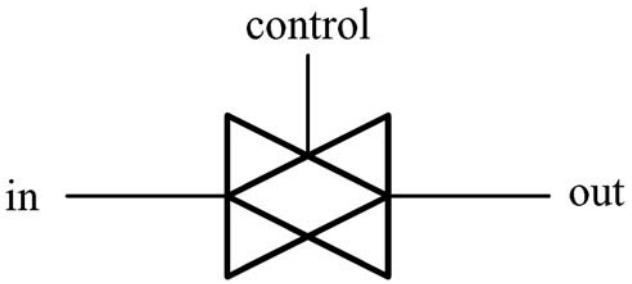


Bidirectional Switches

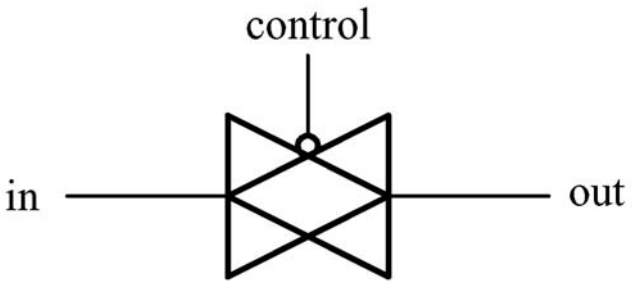
- To instantiate bidirectional switches:
  - tran [*instance_name*] (in, out);
  - tranif0 [*instance_name*] (in, out, control);
  - tranif1 [*instance_name*] (in, out, control);
- instance_name is optional



(a) tran (rtran)



(b) tranif1 (rtranif1)



(c) tranif0 (rtranif0)

Delay : Unidirectional Switches

- Specify no delay :
  - mos_sw [instance_name](output, input, …);
- Specify propagation delay only :
  - mos_sw #(prop_delay)[instance_name](output, input, …);
- Specify both rise and fall times :
  - mos_sw #(t_rise, t_fall)[instance_name](output, input, …);
- Specify rise, fall, and turn-off times :
  - mos_sw #(t_rise, t_fall, t_off)[instance_name](output, input, …);

Delay : Bidirectional Switches

- These switches do not delay signals passing through them. Instead, they have turn-on and turn-off delays while switching

- Specify no delay :
  - bdsw name [instance name](in, out, control);
- Specify a turn-on and turn-off delay :
  - bdsw name #(t_on_off)[instance name](in, out,control);
- Specify separately turn-on and turn-off delays :
  - bdsw name #(t_on, t_off)[instance name](in, out, control);

Resistive Switches

- MOS, CMOS, and bidirectional switches can be modeled as corresponding resistive devices

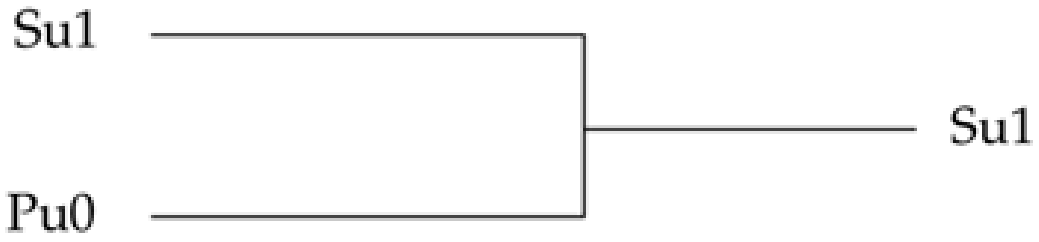
MOS	rnmos		rpmos
CMOS	rcmos		
Bidirectional	rtran	rtranif0	rtranif1

Signal Strengths

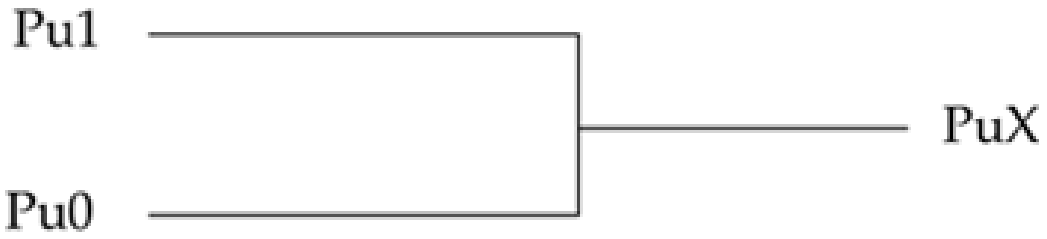
Strength	Strength0	Strength1	Type	Degree
supply	supply0	supply1	driving	strongest
strong	strong0	strong1	driving	
pull	pull0	pull1	driving	
large	large0	large1	storage	
weak	weak0	weak1	driving	
medium	medium0	medium1	storage	
small	small0	small1	storage	
highz	highz0	highz1	high Z	weakest

Signal Contention

- **Multiple Signals with Same Value and Different Strength**

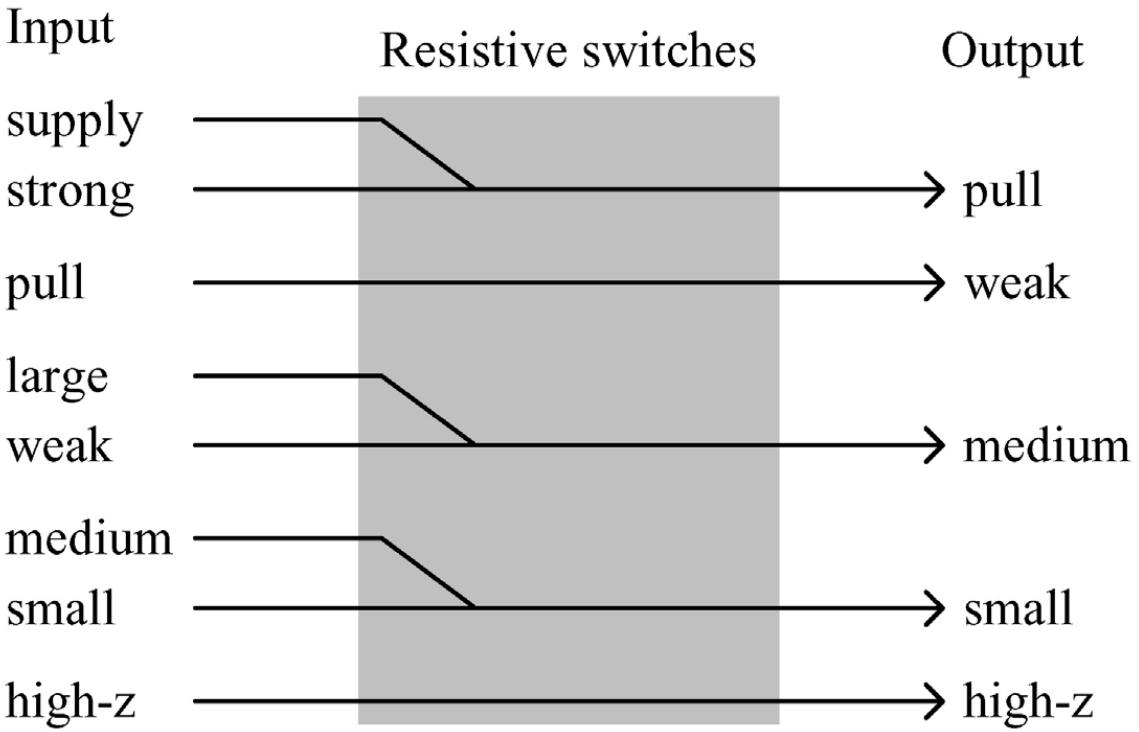


- **Multiple Signals with Opposite Value and Same Strength**



Signal Strength Reduction

- Non-resistive switches do not reduce the strength except that a supply strength is reduced to a strong strength.
- Resistive switches reduce the signal Strength as follows :



Net Types

- tri
  - Same as wire
  - Used to denote a multi-driver node
- tri0 and tri1
  - used to model resistive pulldown and pullup
    - tri0 net has a value 0 if nothing is driving the net
    - tri1 net has a value 1 if nothing is driving the net
  - The default strength is pull
- supply0 and supply1
  - used to model a power supply
  - have constant logic value and a strength level supply (strongest strength level)

Net Types (cont.)

- trireg
  - used to model nets having capacitance that stores values
  - The default strength for trireg nets is medium and the diffault value of the is X
  - Driven state
    - At least one driver drives a 0, 1, or x value on the net
    - The value is continuously stored in the trireg net
    - It takes the strength of the driver
  - Capacitive state
    - All drivers on the net have high impedance (z) value
    - The net holds the last driven value
    - The strength is small, medium, or large (default is medium)

trireg Net Charge Decay

- Like all nets, a trireg declaration's delay specification can contain up to three delays :
  - The first two delays specify the simulation time elapsed in a transition to the 1 and 0 logic values in the driven state
  - The third delay :
    - For the other net types shows the time that elapses in a transition to the z logic state
    - For the trireg net type specifies the charge decay time
- The charge decay process ends under one of the following two conditions:
  1. The specified number of time units elapse and the trireg makes a transition from 1 or 0 to x
  2. One of the drivers turn on and propagate a 1, 0 or x into it

Decay Example

```
module capacitor;
reg data,gate;

trireg (large) #(0,0,50) cap1;

nmos nmos1 (cap1,data,gate);

initial
   begin
      $monitor("%0d data = %v gate = %v cap1 = %v",
                $time,data,gate,cap1);
      data = 1;
      gate = 1;
      #10 gate = 0;
      #30 gate = 1;
      #10 gate = 0;
      #100 $finish;
   end

endmodule
```

`trireg` *declaration with a charge decay time of 50 time units*

`nmos` *switch that drives the* `trireg`

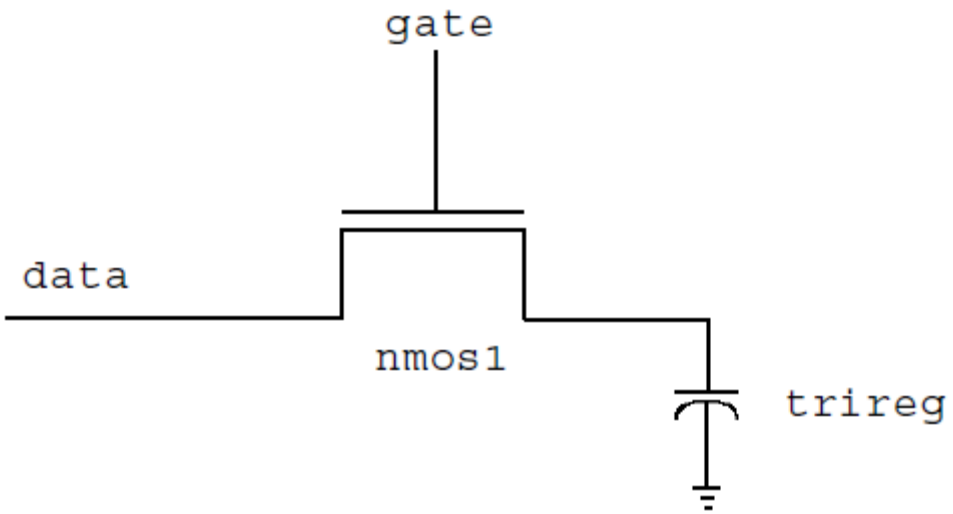
toggles the driver of the control input to the `nmos` *switch*

Decay Example (cont.)

```
 0 data = St1 gate = St1 cap1 = St1
10 data = St1 gate = St0 cap1 = La1
40 data = St1 gate = St1 cap1 = St1
50 data = St1 gate = St0 cap1 = La1

Warning! Time = 100: Charge on node capacitor.cap1 has
        decayed                                  [Verilog-DECAY]
          "trireg1.v", 4: cap1
100 data = St1 gate = St0 cap1 = LaX
```

trireg cap1 changes value to LaX at simulation time 100



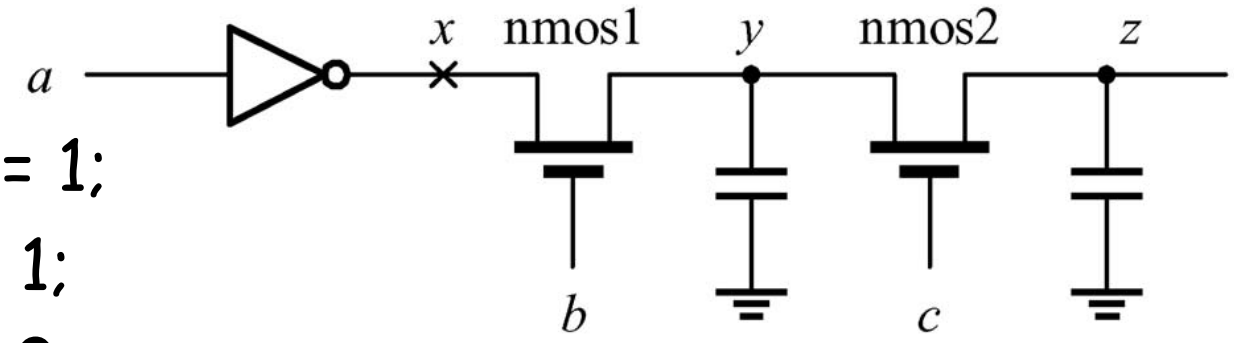
Charge Sharing Example

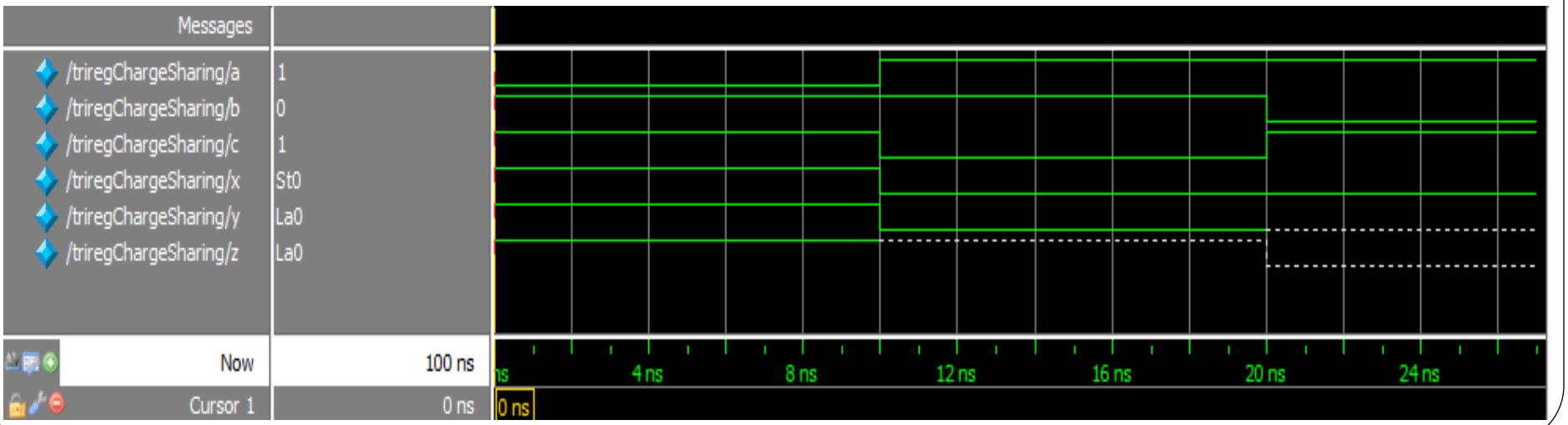
```
module triregChargeSharing;
reg  a, b, c;
wire x;
trireg (large) y; // declaration with large strength
trireg (small) z; // declaration with small strength

not  not1  (x, a);
nmos nmos1 (y, x, b);
nmos nmos2 (z, y, c); // nmos that drives the trireg
```

Charge Sharing Example (cont.)

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





References

- The Verilog® Hardware Description Language by Donald Thomas and Philip Moorby (2008)
- Digital System Designs and Practices: Using Verilog HDL and FPGAs by Ming-Bo Lin (2008)
- Verilog HDL (2nd Edition) by Samir Palnitkar (2003)