

**BROWN**  
School of Engineering

## DIGITAL ELECTRONICS SYSTEM DESIGN

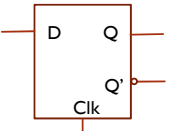
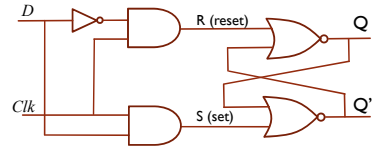
**FALL 2019**  
**PROF. IRIS BAHAR**  
OCTOBER 21, 2019  
LECTURE 13: TRANSMISSION GATES FOR LATCH/FF DESIGN

### ADJUSTED TA AND OFFICE HOURS

- McKenna will be out of town this Friday
  - No morning lab hours this Friday
  - Extra lunchtime hours Tues., and Thurs. this week
- I will have office hours today after class, but may have to leave a bit early
- No office hours this Tuesday, Oct. 22 (I will be out of town)
- If you would like to meet with me this week, please send me an email to schedule a separate time

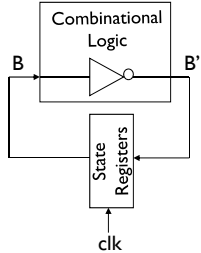
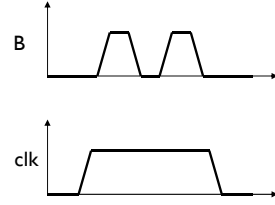
### D LATCH

Clk	D	R	S	Q	Q'
1	1	0	1	1	0
1	0	1	0	0	1
0	X	0	0	Q <sub>prev</sub>	Q' <sub>prev</sub>

- Circuit guarantees R=S=1 will never occur
- But now a single latch takes 22 transistors!

### LATCH RACE PROBLEM

Which value of B is stored?

Two-sided clock constraint

$$T \geq t_{c-q} + t_{plogic} + t_{su}$$

$$T_{high} < t_{c-q} + t_{cdlogic}$$

## FLIP-FLOP MADE FROM D-LATCHES

D	Clk	Q
0		0
1		1
X	0	Q <sub>prev</sub>
X	1	Q <sub>prev</sub>

- Copies D to Q on the rising edge of the clock
- Fixes race problem but now it take 44 transistors to make a single FF!

## TIMING METRICS

clock

In

Out

time

time

time

$t_{su}$  = setup time for data before clock edge

$t_{hold}$  = time data must remain valid after clock edge

$t_{c-q}$  = max FF delay from clock edge to output Q

## SYSTEM TIMING CONSTRAINTS

**Sequential Circuit**

inputs → Combinational Logic → outputs

Prev. State → State → Next State

clk

$t_{plogic}$  = worst case delay through combinational logic

$t_{cdlogic}$  = min. delay through combinational logic

$t_{cdreg}$  = min. delay through register logic

$T$  (clock period)

$T \geq t_{c-q} + t_{plogic} + t_{su}$

$t_{cdreg} + t_{cdlogic} \geq t_{hold}$

## NMOS TRANSISTORS IN SERIES/PARALLEL

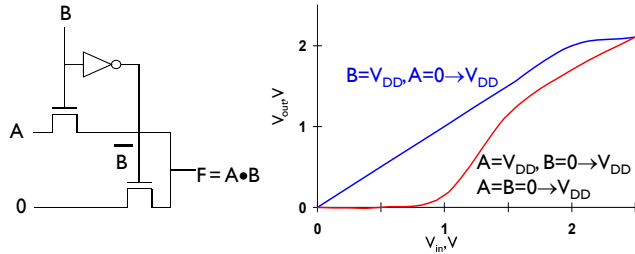
- So far we have assumed that primary inputs are only allowed to drive gate terminals of MOS transistors.
- Now assume primary inputs can drive both gate and source/drain terminals
- NMOS switch closes when the gate input is high

$X = Y$  if A and B

$X = Y$  if A or B

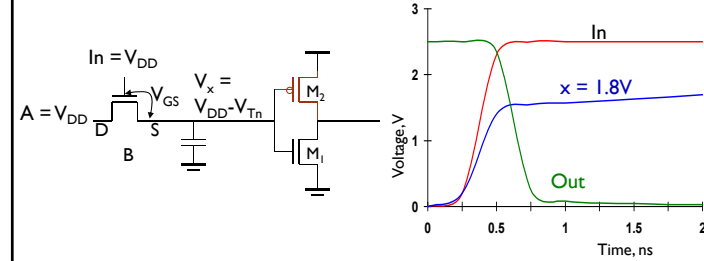
- Remember - NMOS transistors pass a strong 0 but a weak 1

### VTC OF PASS TRANSISTOR AND GATE



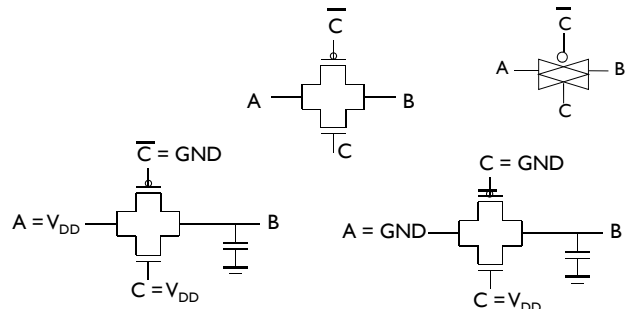
- Pure PT logic is not regenerative: signal gradually degrades after passing through a number of PTs
- ➔ fix with static CMOS inverter insertion

### NMOS ONLY PT DRIVING AN INVERTER



- $V_x$  does not pull up to  $V_{DD}$ , but  $V_{DD} - V_{TN}$
- Threshold voltage drop causes static power dissipation ( $M_2$  may be weakly conducting forming a path from  $V_{DD}$  to GND)

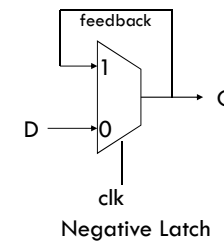
### TRANSMISSION GATES (TGS)



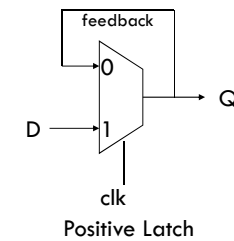
- Most widely used solution
- Full swing bidirectional switch controlled by gate signal C:  $A = B$  if  $C = 1$

### MUX BASED LATCHES

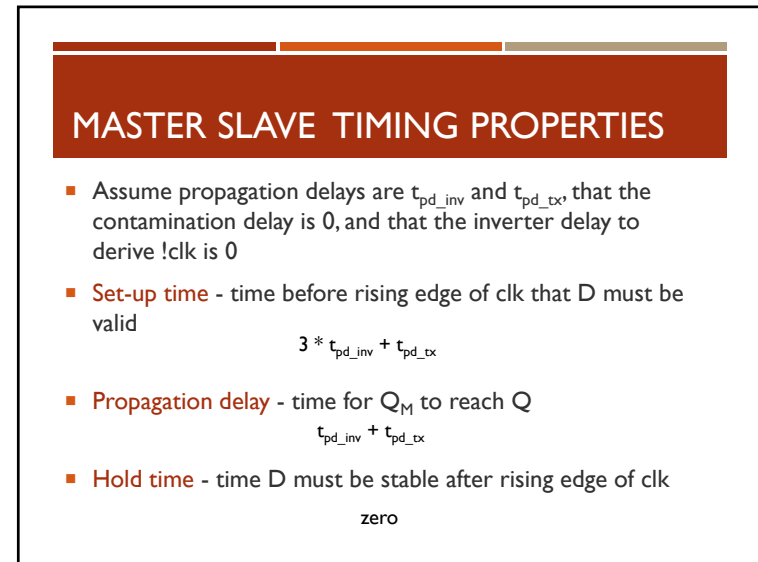
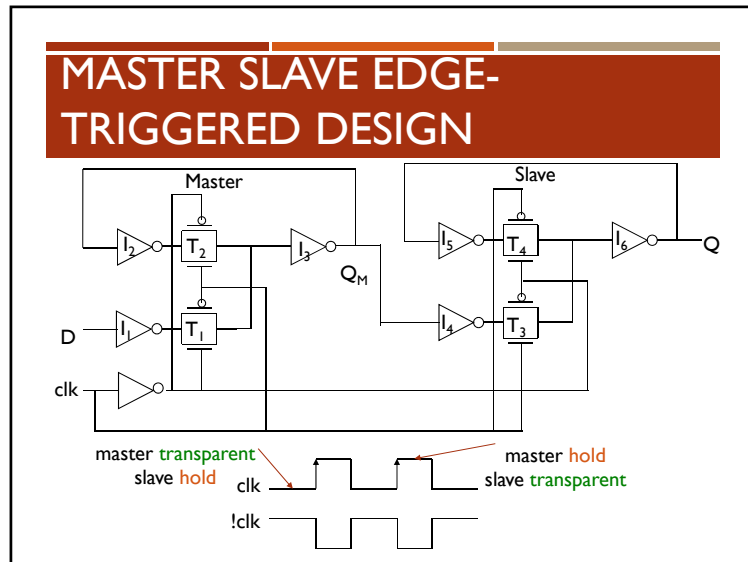
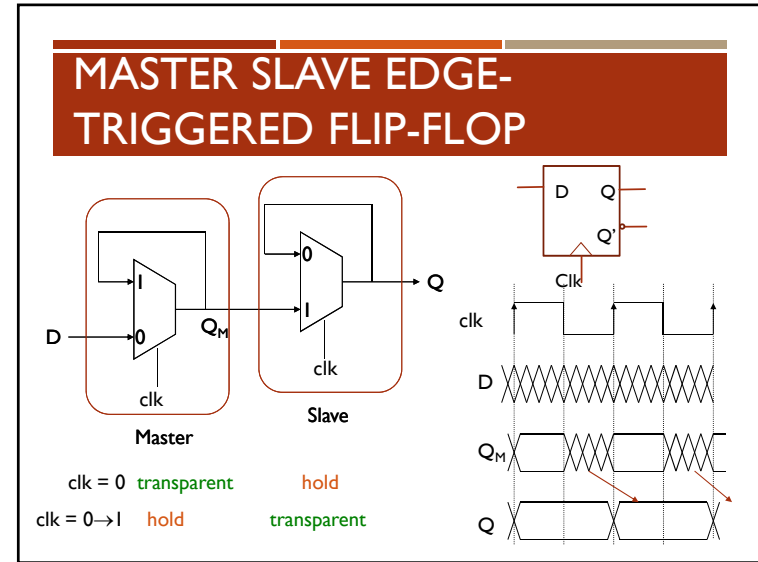
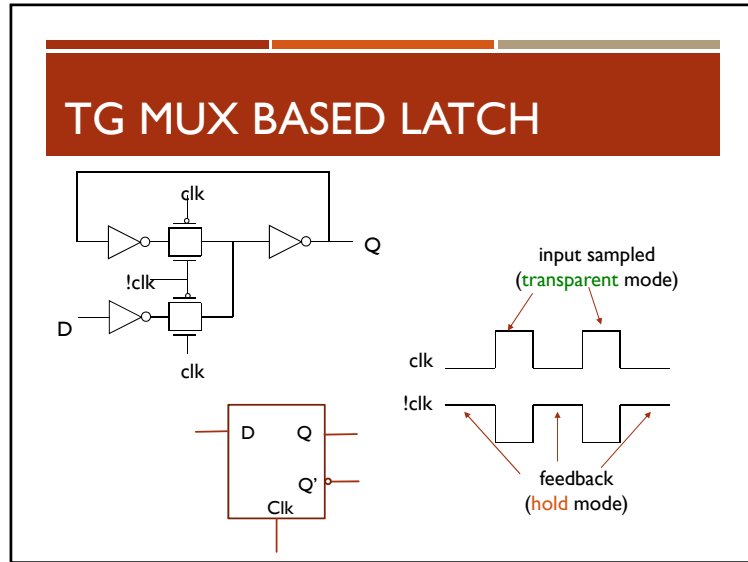
- Change the stored value by cutting the feedback loop



Negative Latch  
 $Q = \text{clk} \& Q \mid \text{!clk} \& D$   
 transparent when the clock is low



Positive Latch  
 $Q = \text{!clk} \& Q \mid \text{clk} \& D$   
 transparent when the clock is high



## SEGREGATING BLOCKING AND NON-BLOCKING ASSIGNMENTS

- Why segregate blocking and non-blocking assignments to separate *always* blocks?
  - *always* blocks start when triggered and scan their statements sequentially
  - Blocking assignment: = (completes assignment before next statement executes)
  - Non-blocking: <= (all such statements complete at once at end of the *always* block)

## MIXED ALWAYS BLOCK FOR A FUNNY SHIFTER

```

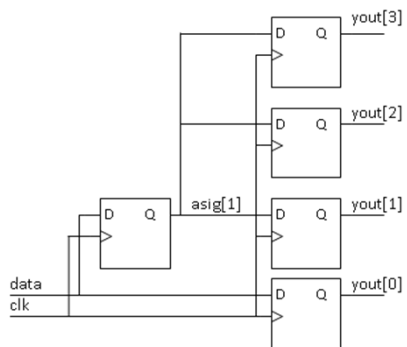
module funnyshifter (input data, clk, output reg [3:0] yout);
  reg [3:0] asig;

  initial asig = 4'b0000;

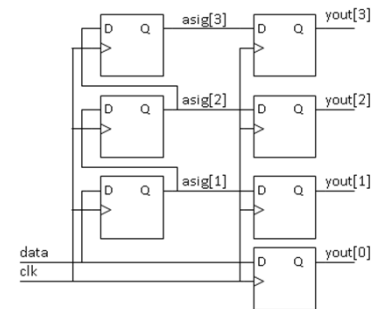
  always @ (posedge clk) begin
    asig[1] = asig[0];
    asig[2] = asig[1];
    asig[3] = asig[2];
    asig[0] = data;
    yout[3:1] <= asig[3:1];
    yout[0] <= data;
  end
endmodule

```

## WHAT YOU ACTUALLY GET



## DESIGNER'S PROBABLE INTENTION – BLOCK DIAGRAM



## DIFFERENT STATEMENT ORDER - DESIGNER'S PROBABLE INTENTION

```

module funnyshifter (input data, clk, output reg [3:0] yout);

reg [3:0] asig;
initial asig = 4'b0000;

always @ (posedge clk) begin
    asig[3] = asig[2];
    asig[2] = asig[1];
    asig[1] = asig[0];
    asig[0] = data;
    yout[3:1] <= asig[3:1];
    yout[0] <= data;
end

endmodule

```

## THE RIGHT WAY TO DO IT:

```

module funnyshifter (input data, clk, output reg [3:0] yout);
reg [3:0] asig;
initial asig = 4'b0000; // Note: this line initializes only the simulator

always @ (posedge clk) begin
    asig <= {asig[2:0], data}; // non-blocking unambiguous D-ff's
    yout[3:1] <= asig[3:1];
    yout[0] <= data;
end

endmodule

```

## SUGGESTED RULES AND STYLES

- “Avoid writing modules that... mix the creation of state... in an *always @ posedge* block with the definition of the next-state function... (This) sidesteps a tremendous amount of confusion and frustration that result from incorrect use of blocking = versus non-blocking <= assignment.”

Dally and Harting, *Digital Design a Systems Approach*, pp. 593-594

- “Two rules are so important this is the only place that you'll find bold font in this book.
    - Always use **blocking** assignments (=) in *always* blocks intended to create combinational logic.
    - Always use **non-blocking** assignments (<=) in *always* blocks intended to create registers.
    - Do not mix blocking and non-blocking logic in the same *always* block.”
- John F.Wakerly, *Digital Design Principles and Practices*, pg. 316.

## A STRONG STYLE PREFERENCE:

- Rule: in any *always* block, you must not leave ambiguity
  - all possible input conditions should have fully specified output conditions
- Common logic expression formats include:

**Option 1:** `always @ (*) begin`  
`if (adv = 1'b1) next = 0; // will get latch or permanent 0;`  
`end`

**Option 2:** `always @ (*) begin // PREFERRED STYLE`  
`case(adv) // USE case() for multiple choices`  
`1'b1: next = 0;`  
`1'b0: next = 1; // Preferable to include all cases`  
`default: next = 1; // ALWAYS supply a default.`  
`endcase`

**Option 3:** `assign next = adv ? 0 : 1; // Satisfactory for single bit usage inside or outside always block`