CSE 2600
Intro. To Digital Logic & Computer Design

Bill Siever
&
Michael Hall

This week

- Studio 4A: Bring a Micro-USB cable to class. You will need it!
- Next week (Thursday, October 8th): Exam 1 at class time

Studio 3B: Space Use

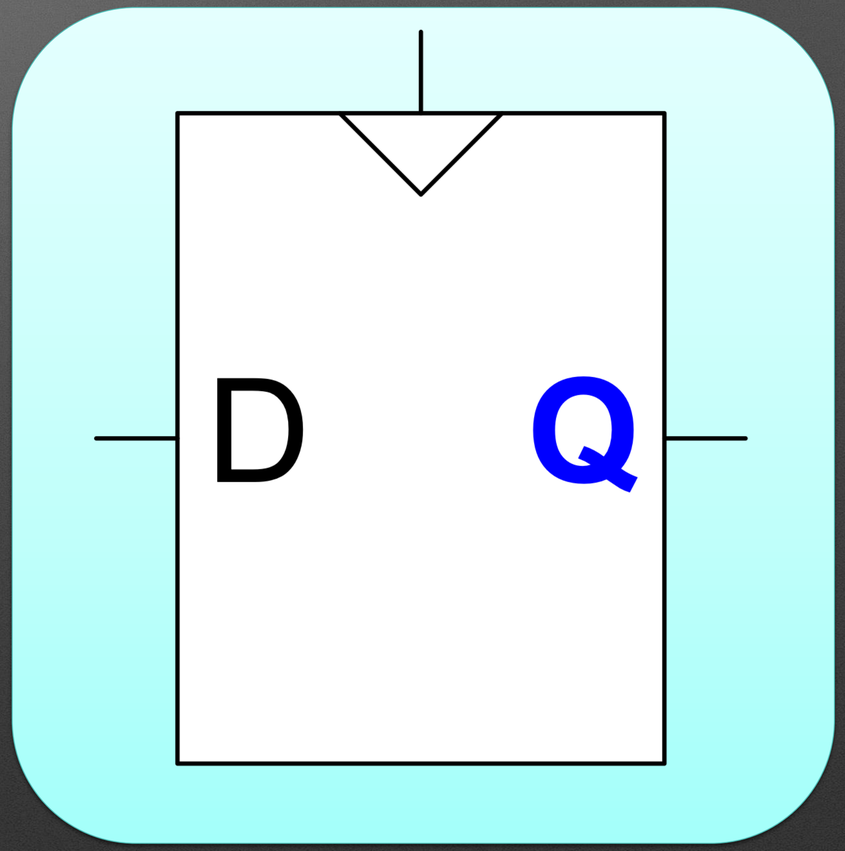
- n states
- ~8 Gates / Flip-flop (Via in-class approach)
- Assume simple approaches for decoding

ENCODING	GATES FOR STORAGE	GATES FOR DECODING	TOTAL (SPACE) GROWTH
BINARY	8*ceil(log2(n))	Likely n product terms of ~log2(n) variables	~ 8*log2(n)+n*log2(n) = O(n log2(n))
ONE-HOT	8*n	0	~ 8n = O(n)

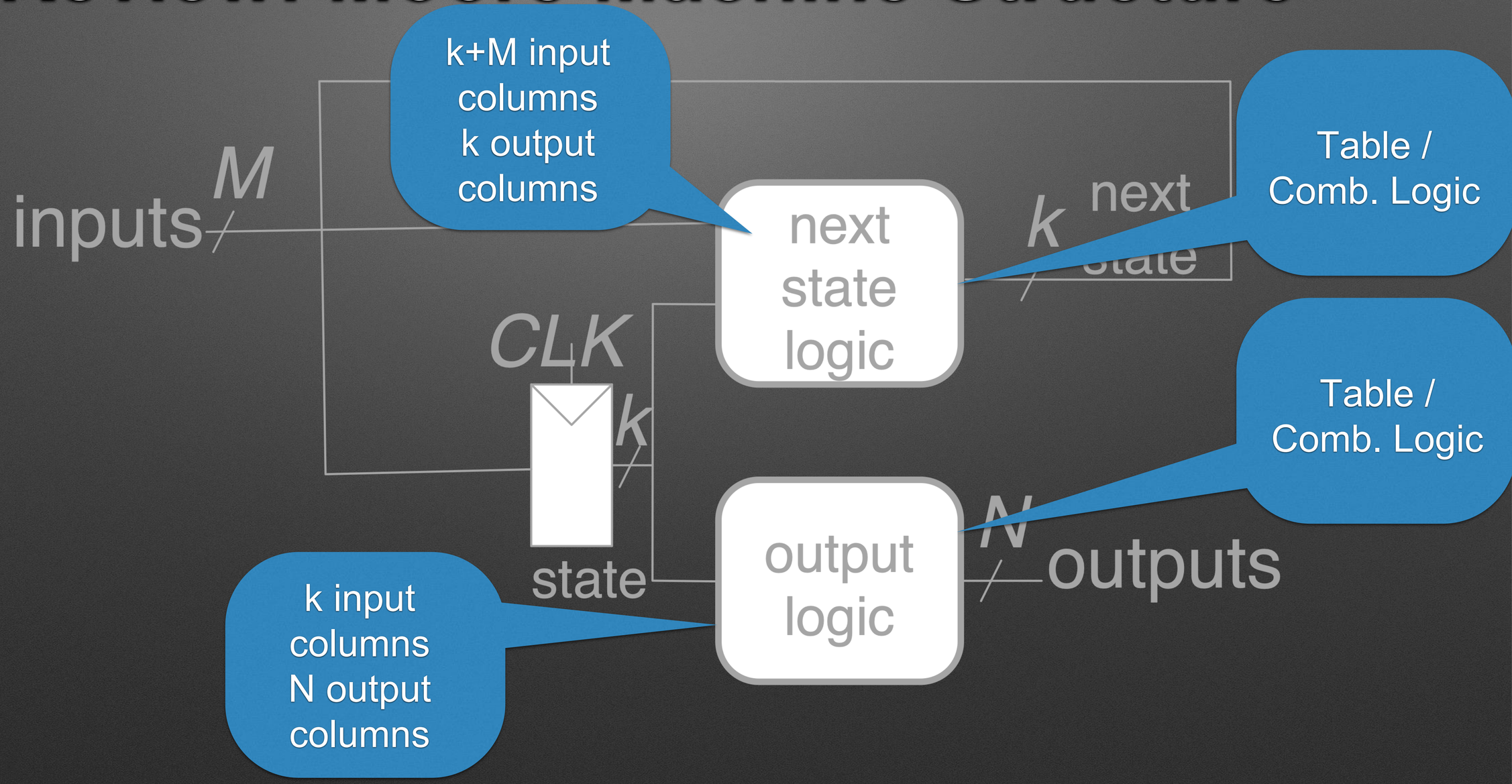
Rough Graph

D-flip flop Time Parameters

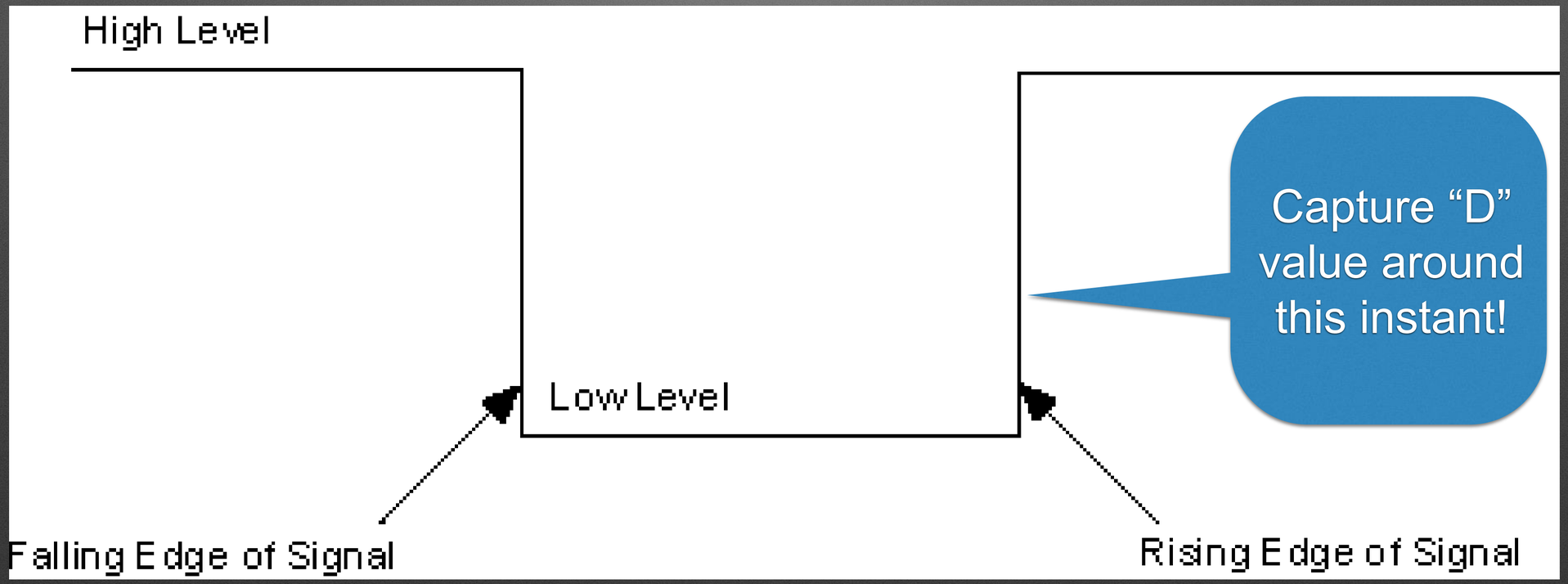
- t_{pcq}: Propagation delay from Clock to Q (pcq)
- t_{ccq}: Contamination delay from Clock to Q (ccq)
- t_{setup}: Setup time (for d before clock)
- t_{hold}: Hold time (for d after clock)



Review: Moore Machine Structure



D-Flip-Flop Clock

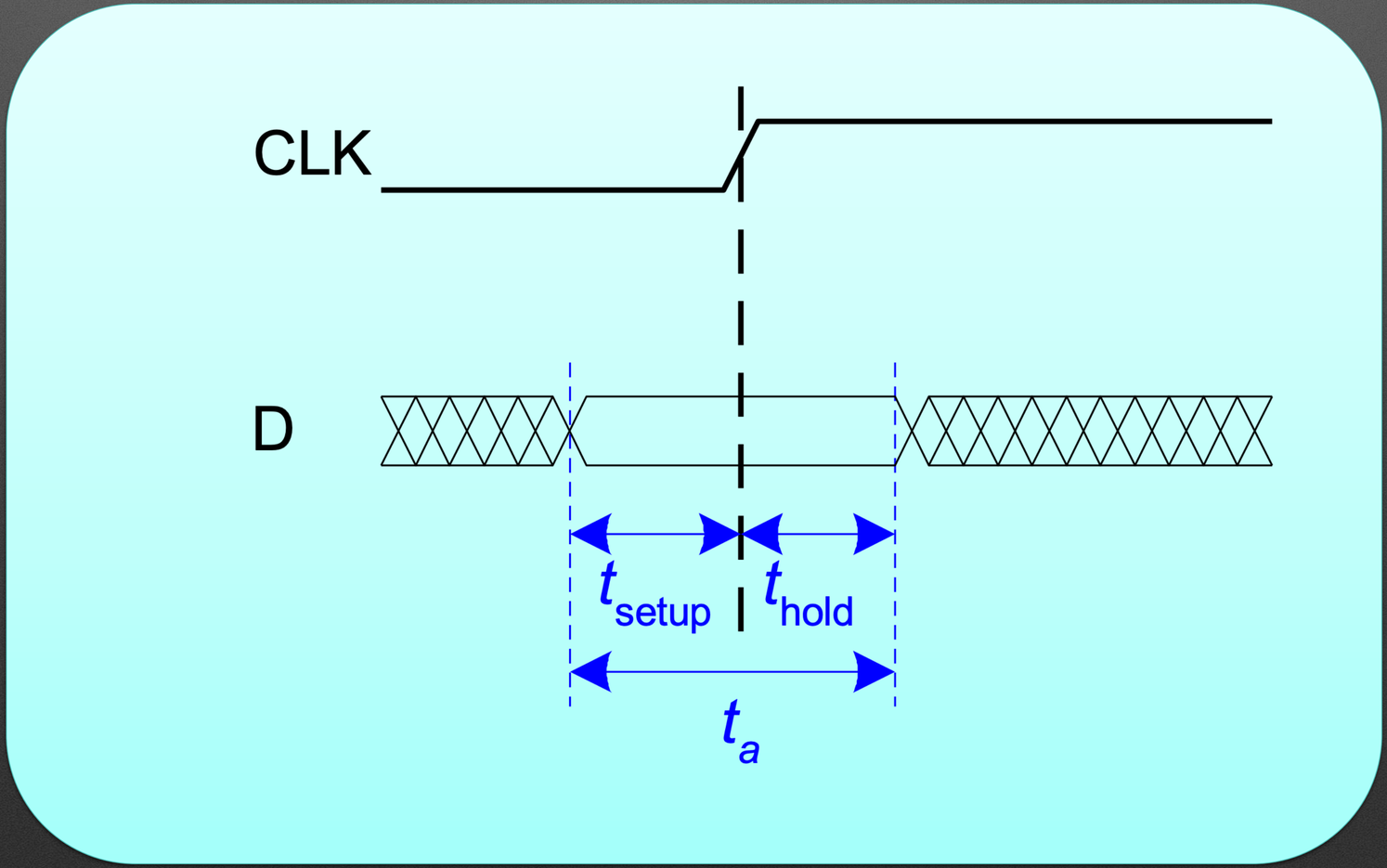


https://www.ni.com/docs/en-US/bundle/ni-hsdio/page/hsdio/fedge_trigger.html

Studio 3B Review

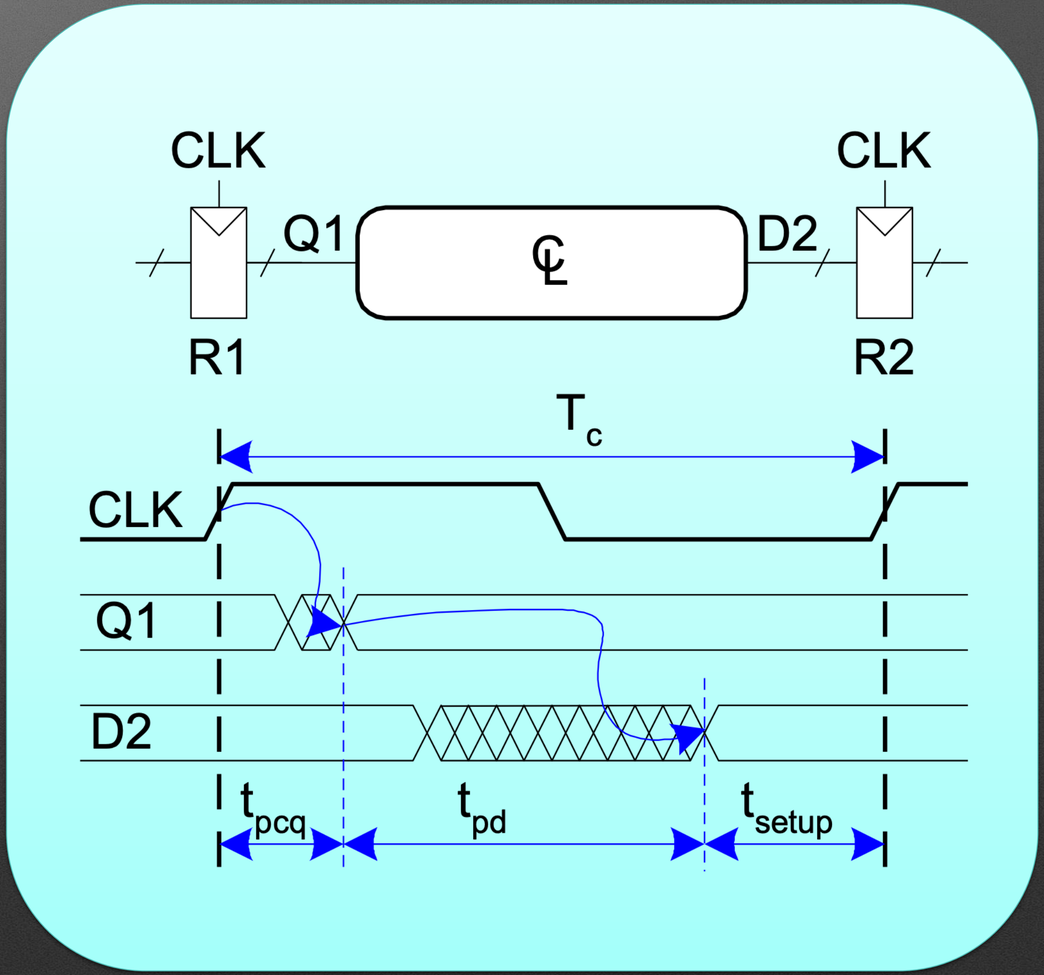
- D-FF Model
  - NOR oscillation - Why?
    - *Important*: Circuits are composed of continually, concurrently executing elements!
  - D-Latch Setup / Hold: Shaping waveforms and measuring around clock
    - Demonstrates concept, but shouldn't be measured empirically in practice! (Rely on data sheets in this class)

D flip flop: Setup & Hold Time

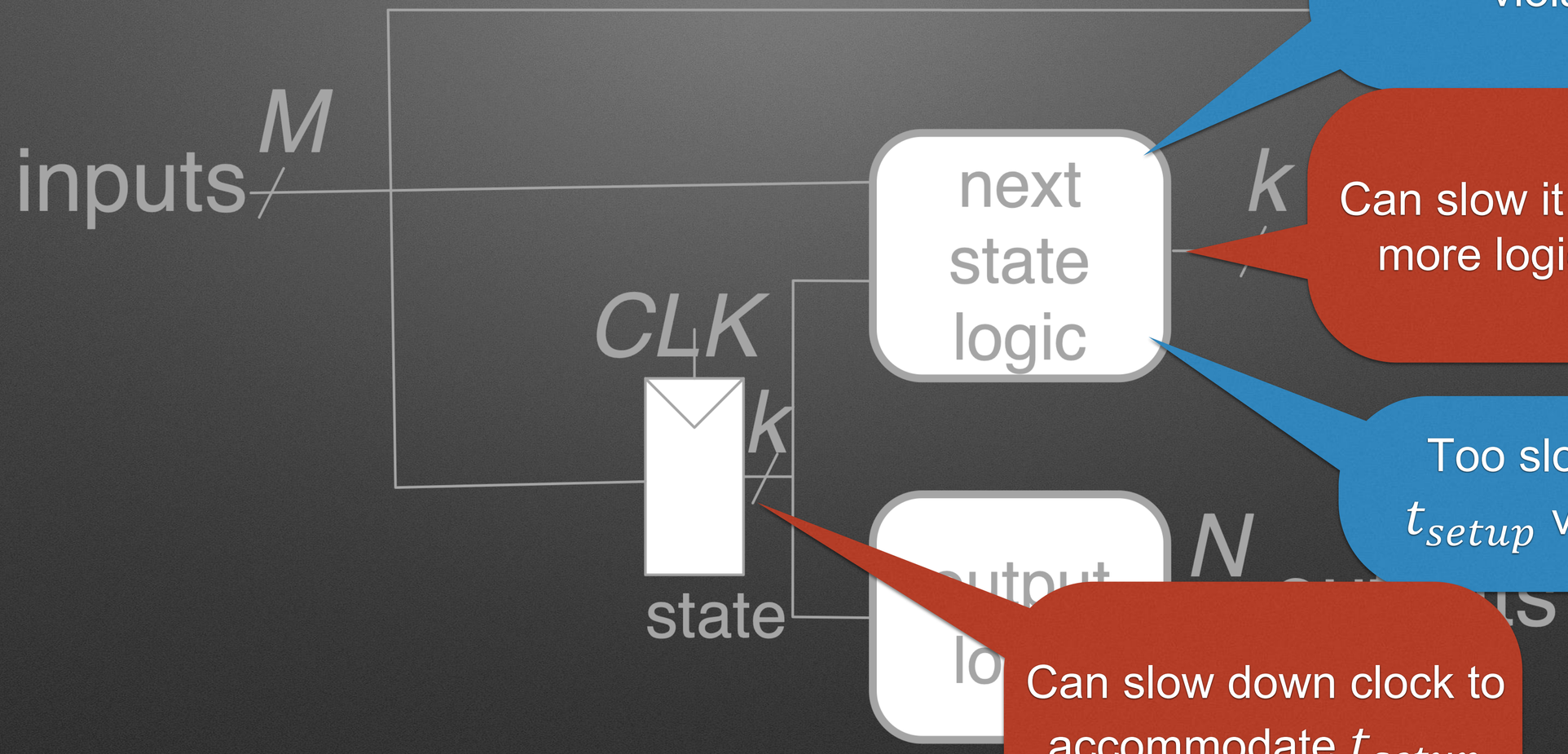


Setup Time Constraint

- Max time from R1 through CL
  - R2's input needs to be stable t_{setup} before clock



Review: Moore Machine Stru



Too fast and t_{hold} violated!

Can slow it down with more logic/buffers

Too slow and t_{setup} violated!

Can slow down clock to accommodate t_{setup}

Chapter 4

Hardware Description Languages (HDLs)

- Specifies logic function only
- Computer-aided design (CAD) tool produces or synthesizes the optimized gates
- Most commercial designs built using HDLs

VHDL

- VHDL 2008
- Developed in 1981 by the Department of Defense
- IEEE standard (1076) in 1987
- Updated in 2008 (IEEE STD 1076-2008)

Questions this week:
Verilog?
VHDL?
Both?

Questions this week:
Verilog?

Both?

We will use Verilog

Not VHDL

(System) Verilog

- Developed in 1984 by Gateway Design Automation
- IEEE standard (1364) in 1995
- Extended in 2005 (IEEE STD 1800-2009)

HDL

- A HDL is not a computer program.
- A HDL is not a computer program!
- A HDL is *not* a computer program!
- A HDL is *not* a computer program!
- A HDL is ***NOT*** a computer program!

HDL

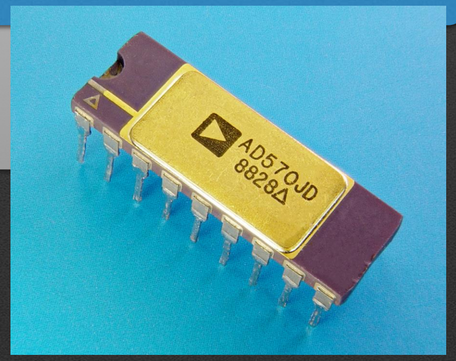
- HDL is a way to *describe* hardware
- HDLs typically can describe hardware in different ways
- HDLs describe hardware in modules

(System) Verilog Module Example

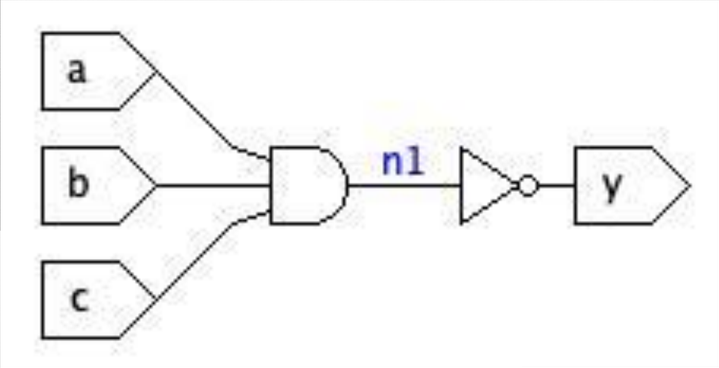
(System) Verilog Module Example

```
module example(input  logic a, b, c,
                      output logic y);
  // module body goes here
endmodule
```

Input & Output are like the Pins On chips or in JLS



HDL: *Structural* (Verilog)




```
module nand3(input  logic a, b, c
             output logic y);
  logic n1;                        // internal signal

  and  my_and(n1, a, b, c);        // instance of and3
  not  my_inverter(y,n1);          // instance of inv
endmodule
```

Example & Tour of Environment

(System) Verilog

- Case sensitive
- Identifiers follow rules like in programming languages
  - Ex: 2inputGate not allowed, but gateWithTwoInputs is.
- Whitespace ignored
- C/Java Style comments:

```
// Comment to end of line
/*
   Block Comment
 */
```

(System) Verilog

- Basic logic operators
  - Binary: &, |, ^
  - Unary: ~

HDL: *Structural* (Verilog)

```
module nand3(input  logic a, b, c
             output logic y);
  logic n1;                        // internal signal

  and  my_and(n1, a, b, c);        // instance of and3
  not  my_inverter(y,n1);          // instance of inv
endmodule
```

HDL: *Structural* (Verilog) (Without named instances)

```
module nand3(input  logic a, b, c
             output logic y);
  logic n1;              // internal signal

  and(n1, a, b, c);      // instance of and3
  not(y,n1);             // instance of inv
endmodule
```

HDL: *Behavioral* (Verilog)

```
module nand3(input  logic a, b, c
             output logic y);
   assign y = ~(a & b & c);
endmodule
```

(System) Verilog

- Multi-bit values: Array-like notation with bit ordering:

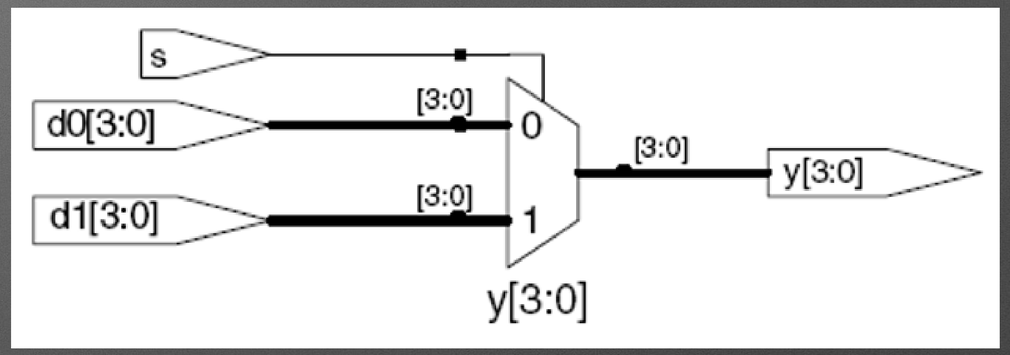
```
logic [7:0] a;
```

- Multi-bit reductions by pre-fix notation on multi-bit values

```
assign y = &a;
```

(System) Verilog

- Conditionals via Ternary operator (? :)




```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);
   assign y = s ? d1 : d0;
endmodule
```

Example

Operator Precedence

~	NOT
*, /, %	mult, div, mod
+, -	add, sub
<<, >>	shift
<<<, >>>	arithmetic shift
<, <=, >, >=	comparison
==, !=	equal, not equal
&, ~&	AND, NAND
^, ~^	XOR, XNOR
|, ~|	OR, NOR
?:	ternary operator

Number Formats

Number	# Bits	Base	Decimal Equivalent	Stored
3'b101	3	binary	5	101
'b11	unsized	binary	3	00…0011
8'b11	8	binary	3	00000011
8'b1010_1011	8	binary	171	10101011
3'd6	3	decimal	6	110
6'o42	6	octal	34	100010
8'hAB	8	hexadecimal	171	10101011
42	Unsized	decimal	42	00…0101010

Verilog Bit Manipulations

- Subscripting and grouping

```
assign y = {a[2:1], {3{b[0]}}, a[0], 6'b100_010};
```

- Underscores can be used for clarity

Questions

- How can I tell whether my HDL code describes a combinational circuit or accidentally creates a circuit that stores a value?
- I am still a little unsure about when exactly blocking versus nonblocking assignments should be used and what hardware problems can happen if the wrong one is used.
- What's the difference between SystemVerilog and VHDL?
- When exactly does an ""always"" block accidentally create a latch instead of combinational logic? is it only a missing default or are there other cases?
- How can we tell whether a piece of HDL code is synthesizable into real hardware or if it can only be used for simulation?