

**ECE 2300**  
**Digital Logic & Computer Organization**  
**Spring 2022**

**More Sequential Logic**  
**Verilog**



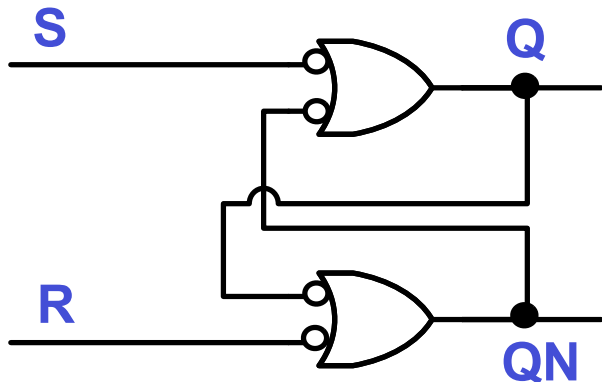
Cornell University

# Announcements

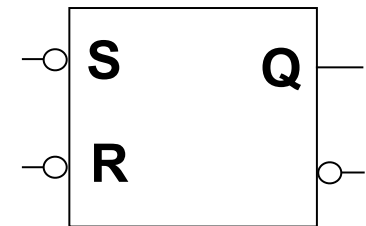
- **Updates to TA OH posted on Ed**
- **Prelab 2A due tomorrow**
  - **Form groups on CMS for Lab 2A**
  
- **Prelim 1**
  - **Thursday Feb 24, 1:00-2:15pm in class**
    - **closed book, closed notes, closed Internet**
  - **Coverage: Lectures 1~7**
    - **Binary number, Boolean algebra, CMOS, combinational logic, sequential logic, and Verilog**
  - **An old prelim exam will be posted tomorrow**
  - **More information to be announced soon**
    - **TA-led review session will be scheduled (& recorded)**

# $\overline{S}$ - $\overline{R}$ Latch

- ***S-bar-R-bar* latch**
  - Built from NAND gates
  - Inputs are active low rather than active high
  - **When both inputs are 0,  $Q = QN = 1$  (avoid!)**



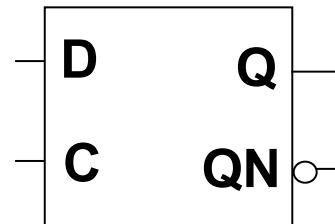
S	R	Q	QN
0	0	1	1
0	1	1	0
1	0	0	1
1	1	Last Q	Last QN



# D Latch and Flip-Flop

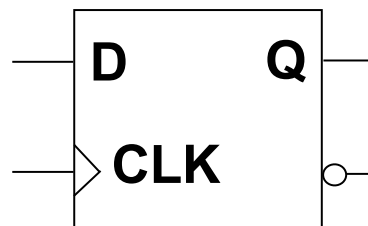
- **Latch: level sensitive**

- Captures the input when enable signal asserted

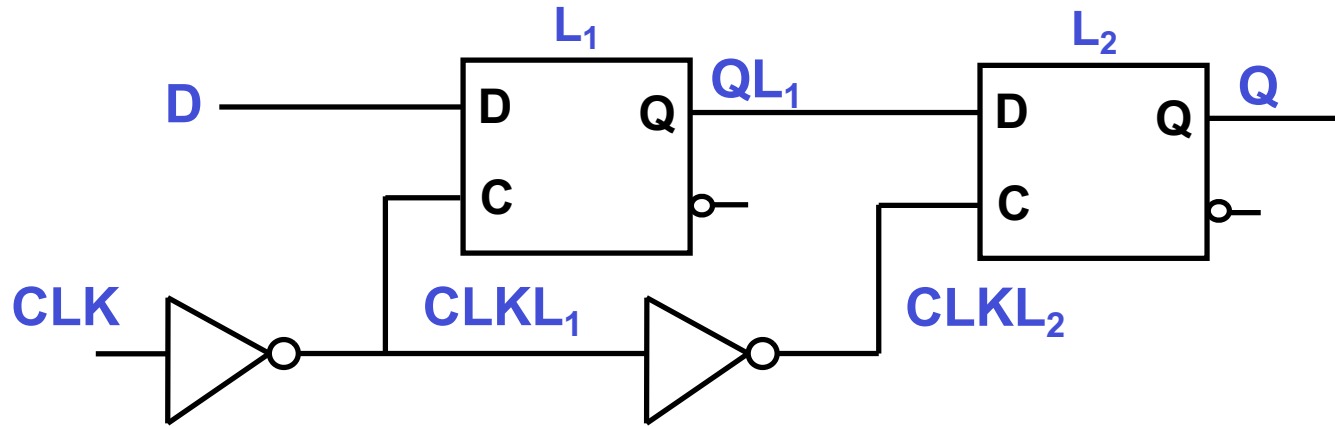


- **Flip-Flop (FF): edge sensitive**

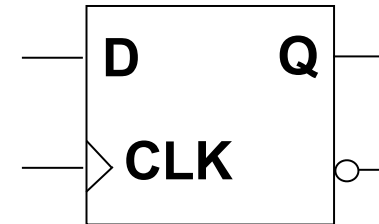
- Captures the input at the triggering clock edges (e.g., L→H)
- A single FF is also called a one-bit register



# Recap: D Flip-Flop (DFF)

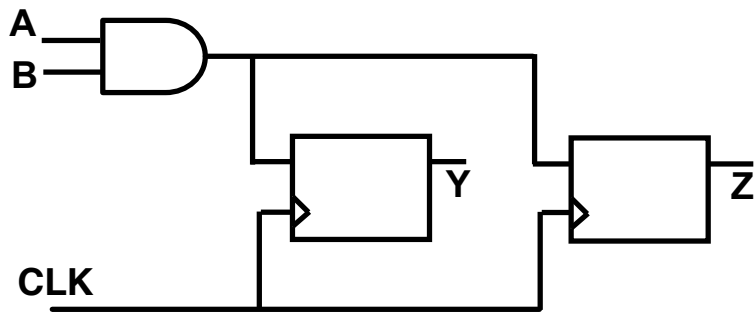


D	CLK	Q	QN
0		0	1
1		1	0
X	0	Last Q	Last QN
X	1	Last Q	Last QN

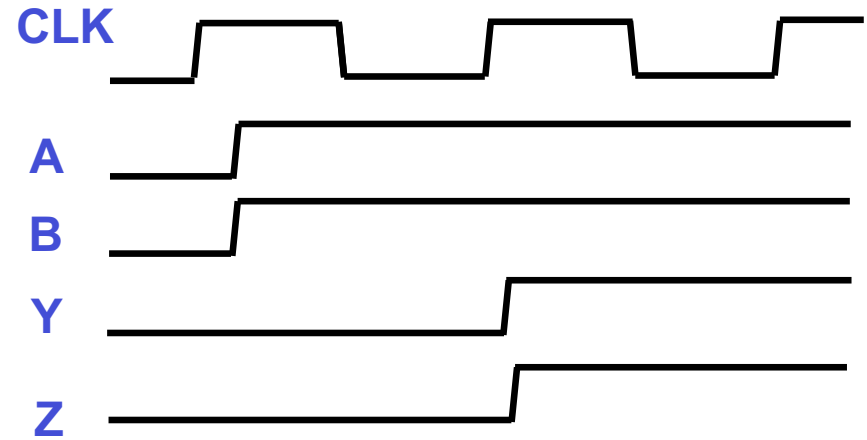


- Copies  $D$  to  $Q$  on the rising edge of the clock

# DFF Timing Example

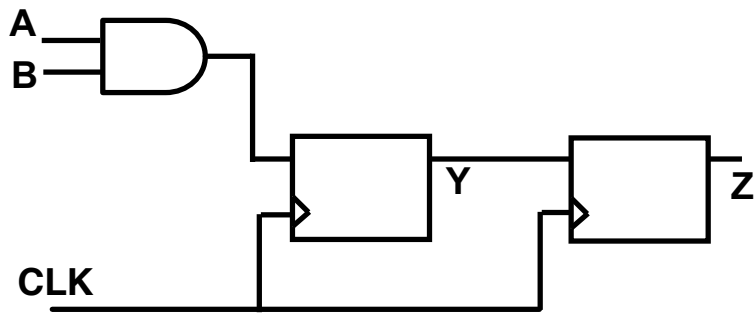


Circuit diagram

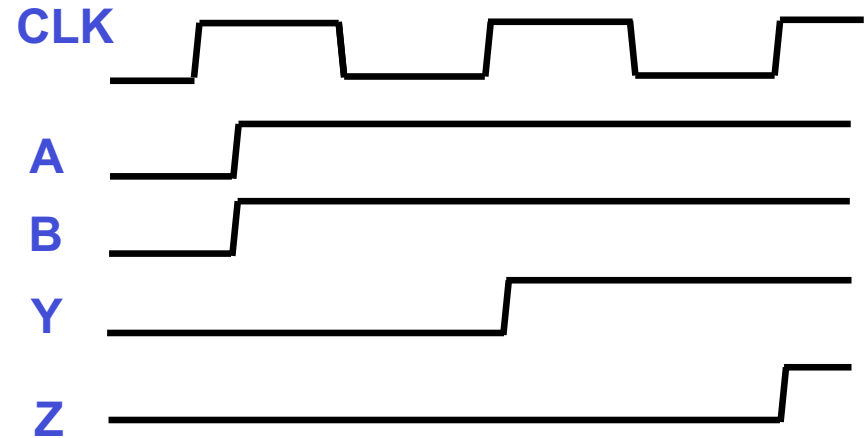


Waveform  
(assuming Y & Z are  
initialized with 0s)

# Another Example



Circuit diagram

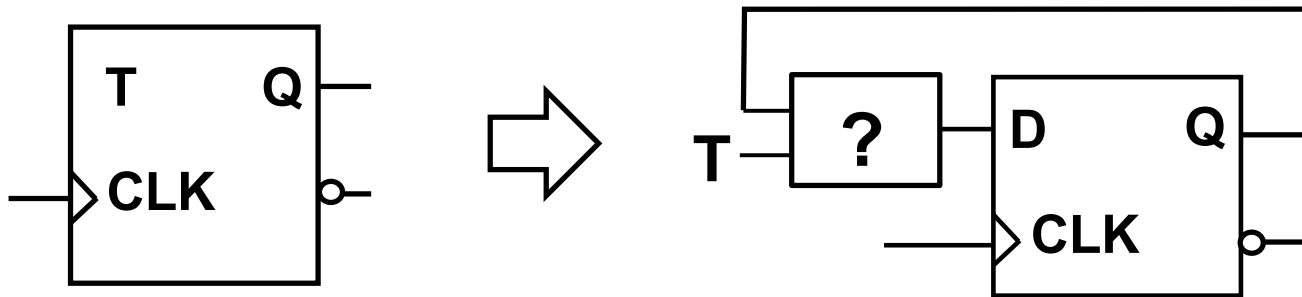


Waveform  
(assuming Y & Z are  
initialized with 0s)

# T (*Toggle*) Flip-Flop

- Output toggles only if  $T=1$
- Output does not change if  $T=0$
- Useful for building counters

**Q: 0, 1, 0, 1, 0, 1, 0, ...**

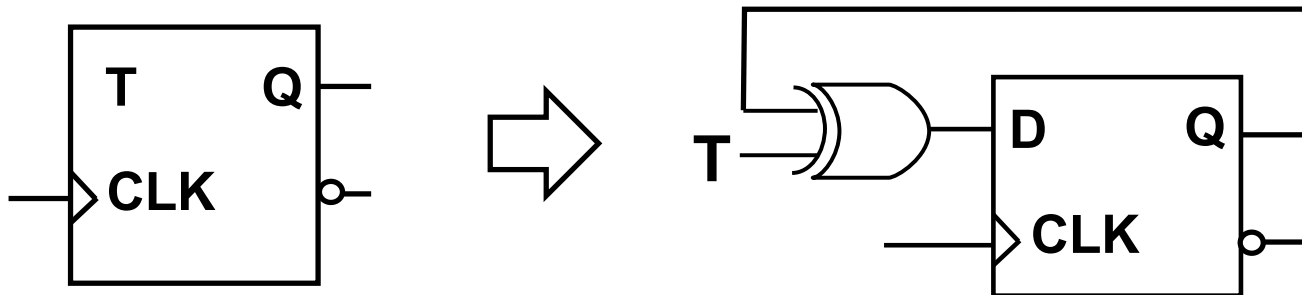




# T (*Toggle*) Flip-Flop

- Output toggles only if T=1
- Output does not change if T=0
- Useful for building counters

Q: 0, 1, 0, 1, 0, 1, 0, ...



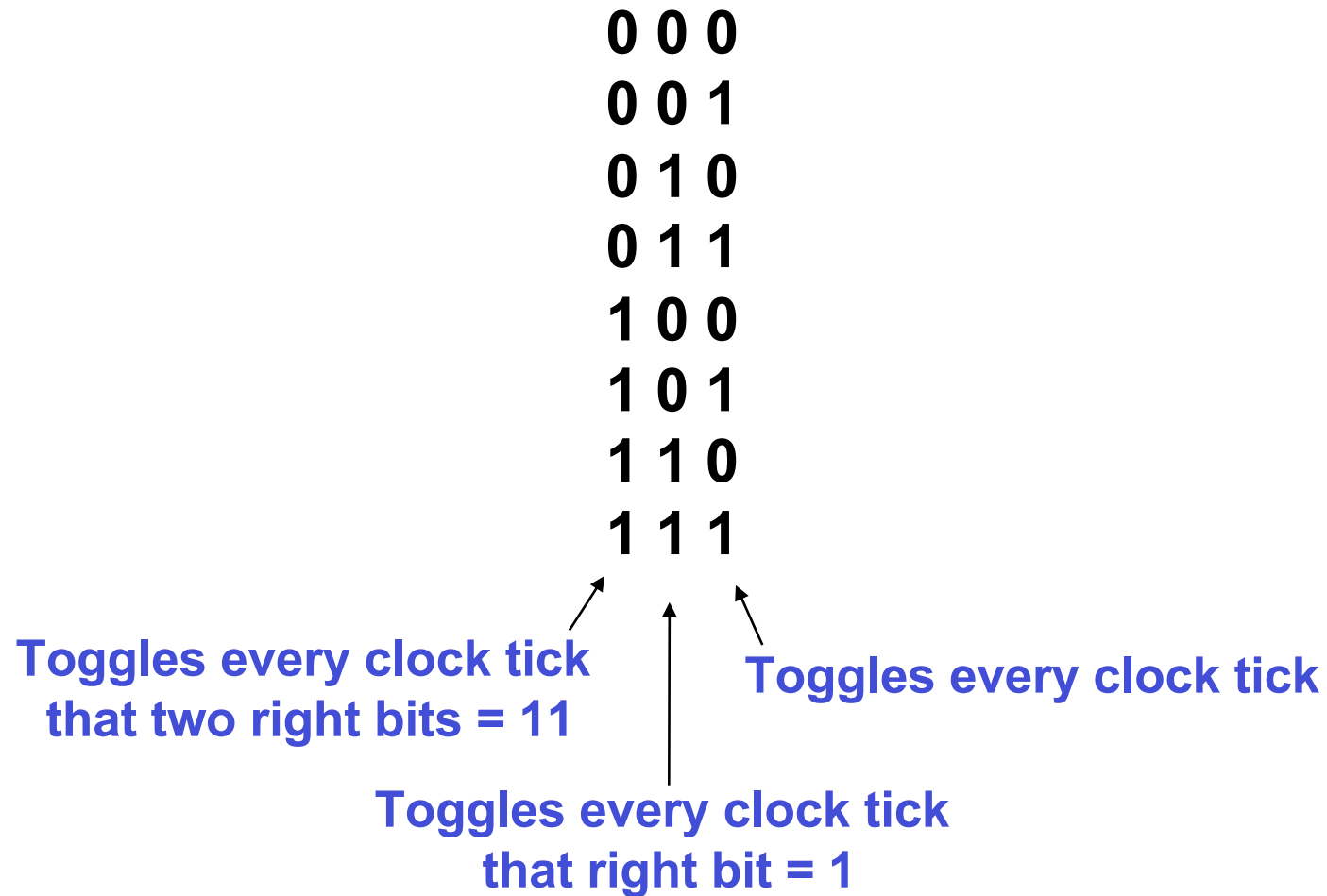
$$Q_{\text{next}} = T \cdot Q' + T' \cdot Q$$

# Binary Counters

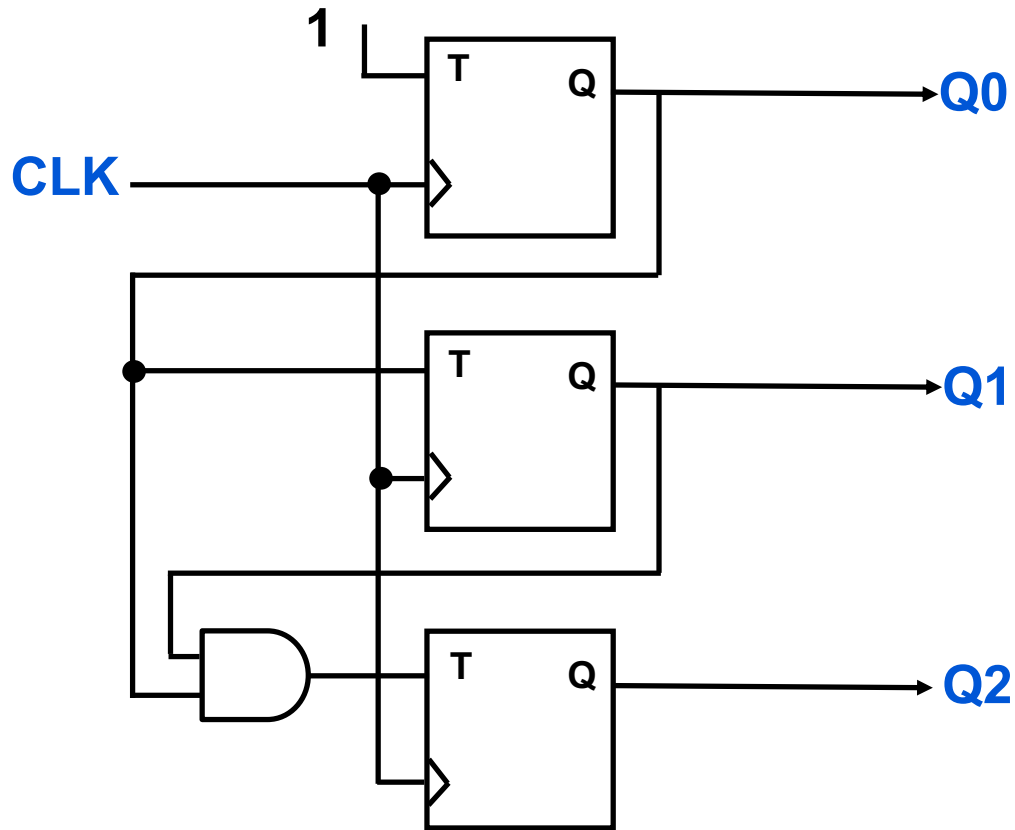
- Counts in binary in a particular sequence
- Advances at every tick of the clock
- Many types

		Divide-	
Up	Down	by-n	n-to-m
0 0 0	1 1 1	0 0 0	n
0 0 1	1 1 0	0 0 1	n+1
0 1 0	1 0 1	0 1 0	n+2
0 1 1	1 0 0	0 1 1	⋮
1 0 0	0 1 1	1 0 0	m-1
1 0 1	0 1 0	⋮	m
⋮	⋮	n-1	n
		0 0 0	n+1
		0 0 1	⋮

# Up Counter Sequence



# Building Binary Up Counter



$Q_2 Q_1 Q_0$

0 0 0

0 0 1

0 1 0

0 1 1

1 0 0

1 0 1

1 1 0

1 1 1

0 0 0

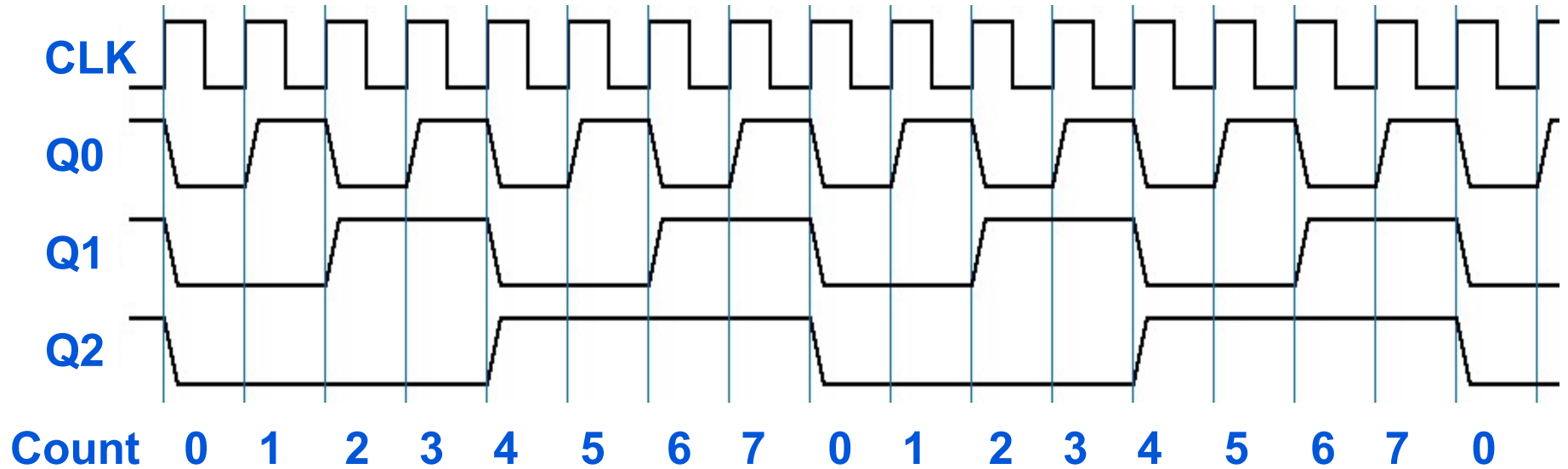
0 0 1

Q0 toggles at every rising edge

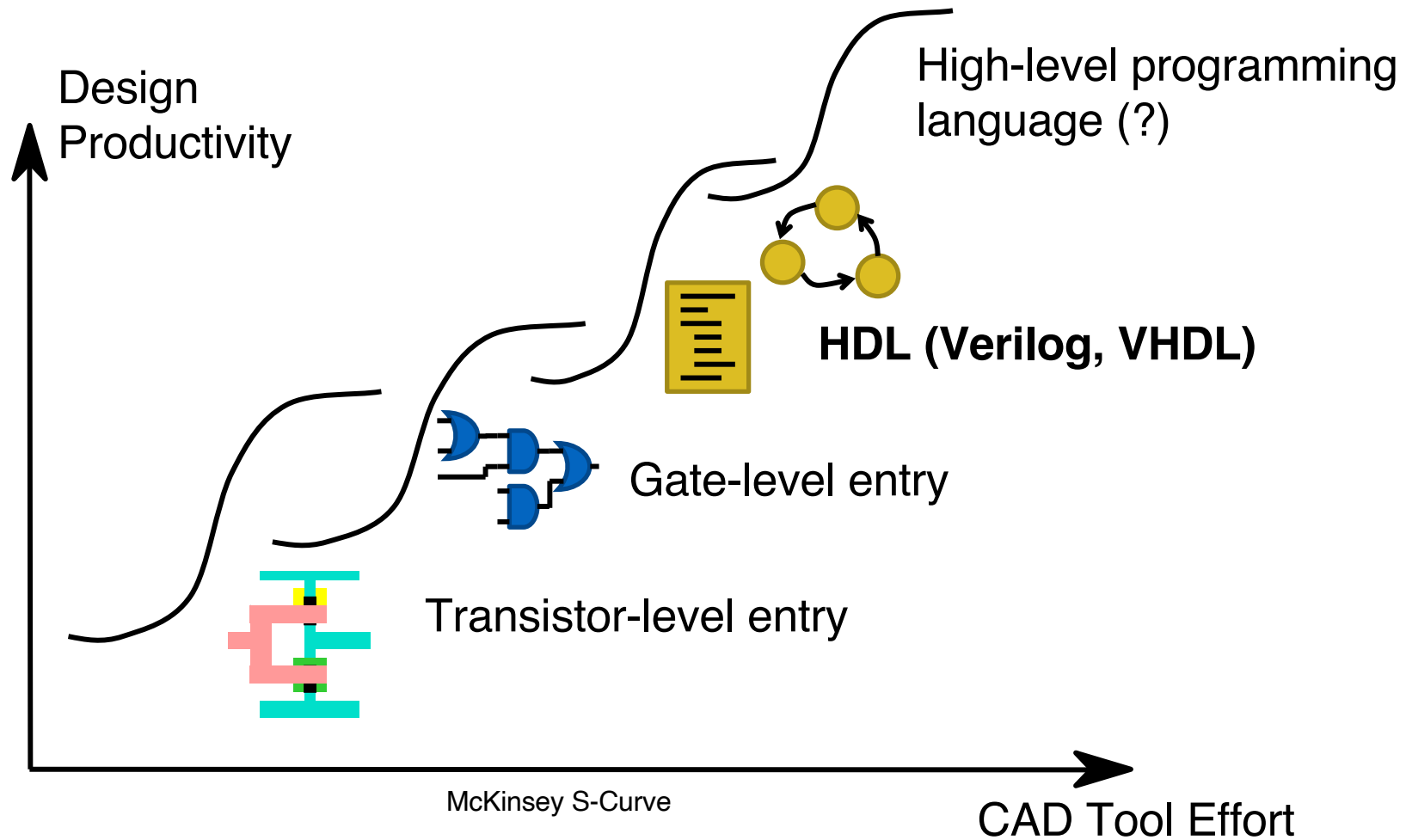
Q1 toggles at the rising edge when Q0=1

Q2 toggles at the rising edge when Q0=Q1=1

# Up Counter Timing Diagram



# Evolution of Design Abstractions



[Figure credit: Kurt Keutzer]

# Hardware Description Languages

- **Hardware Description Language (HDL):**  
a language for describing hardware
  - Efficiently code large, complex designs
    - Programming at a more abstract level than schematics
  - CAD tools can automatically synthesize circuits
- **Industry standards:**
  - Verilog: We will use it from Lab 2
  - SystemVerilog: Successor to Verilog, gaining wide adoption
  - VHDL (Very High Speed Integrated Circuit HDL)

# Verilog

- **Developed in the early 1980s by Gateway Design Automation (later bought by Cadence)**
- **Supports modeling, simulation, and synthesis**
  - **We will use a (synthesizable) subset of the language features**
- **Major language features (in contrast to software programming languages)**
  - **Structure and instantiation**
  - **Concurrency**
  - **Bit-level behavior**



# Values

- **Verilog signals can take 4 values**

- 0 Logical 0, or false**

- 1 Logical 1, or true**

- x Unknown logical value**

- z High impedance (Hi-Z), floating/non-connected**

**x might be a 0, 1, z, or in transition, or don't cares**

**Sometimes useful debugging and often exploited by CAD tools during optimization**

# Bit Vectors

- **Multi-bit values are represented by bit vectors (i.e., grouping of 1-bit signals)**
  - Right-most bit is always least significant
  - **Example**
    - `input[7:0] a1, a2, a3; /* three 8-bit inputs */`

- **Constants**

**4'b1001**



Base format (b,d,h,o)

Decimal number representing bit width

- **Binary Constants**
  - `8'b00000000`
  - `8'b0xx01xx1`
- **Decimal Constants**
  - `4'd10`
  - `32'd65536`

# Operators

- **Bitwise Boolean operators**

~ NOT

& AND

^ Exclusive OR

| OR

- **Arithmetic operators**

+ Addition

/ Division

<< Shift left

– Subtraction

% Modulus

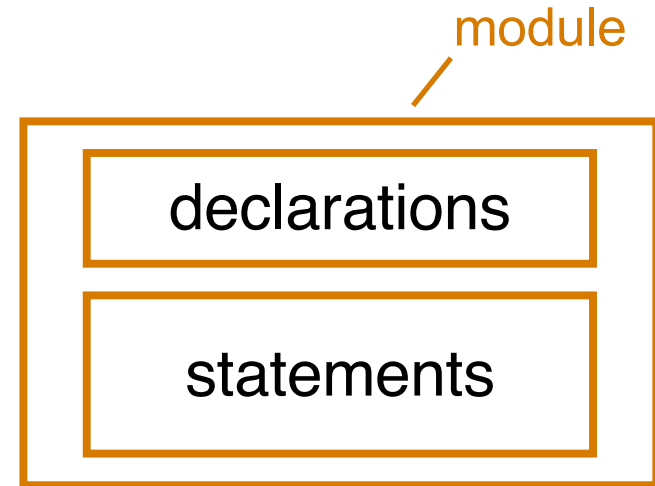
>> Shift right

\* Multiplication

# Verilog Program Structure

- **System is a collection of *modules***

- **Module corresponds to a single piece of hardware**



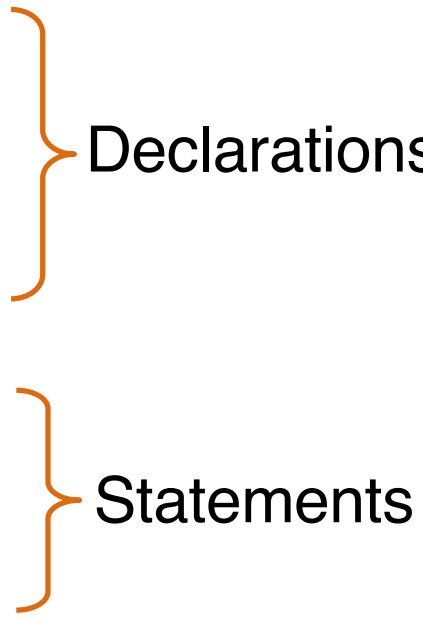
- **Declarations**

- **Describe names and types of inputs and outputs**
  - **Describe local signals, variables, constants, etc.**

- **Statements specify what the module does**

# Verilog Program Structure

```
module M_2_1 (x, y, sel, out);  
  input x, y;  
  input sel;  
  output out;  
  wire tx, ty;  
  
  AND and0 (x, ~sel, tx);  
  AND and1 (y, sel, ty);  
  OR or0 (tx, ty, out);  
  
endmodule
```

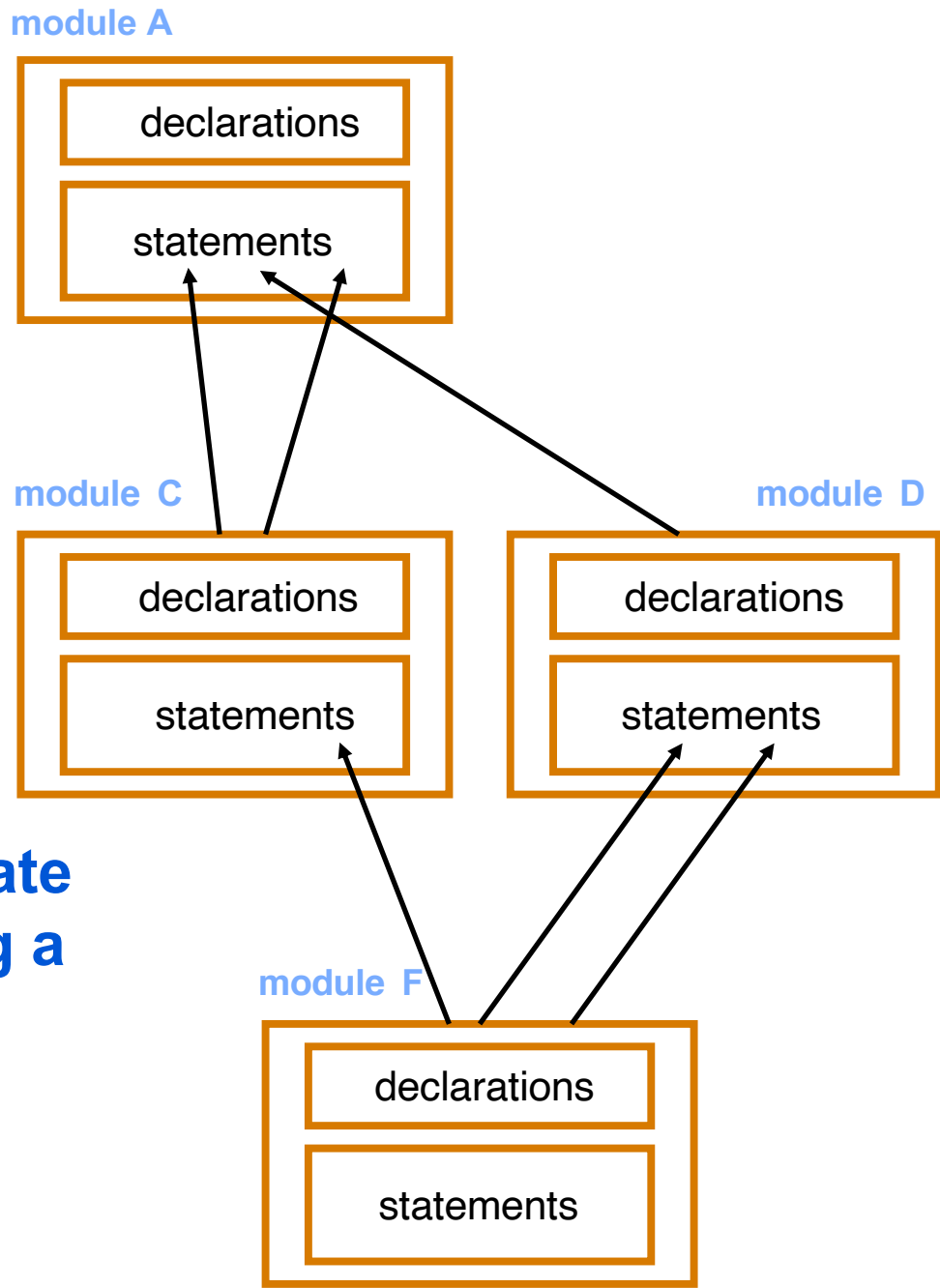


Declarations

Statements

# Verilog Hierarchy

A module can instantiate other modules forming a module hierarchy



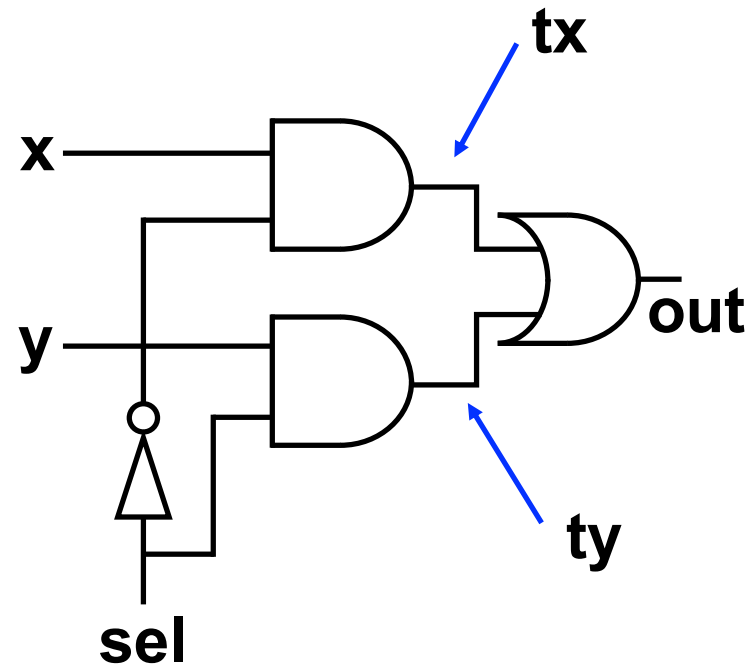
# Verilog Programming Styles

- **Structural**
  - Shows how a module is built from other modules via *instance* statements
  - Textual equivalent of drawing a schematic
- **Behavioral**
  - Specify what a module does in high-level description
  - Use procedural code (e.g., in *always* block) and continuous assignment (i.e., *assign*) constructs to indicate what actions to take

**We can mix the structural and behavioral styles  
in a Verilog design**

# Structural Style

```
module M_2_1 (x, y, sel, out);  
  input x, y;  
  input sel;  
  output out;  
  
  wire tx, ty;  
  
  AND and0 (x, ~sel, tx);  
  AND and1 (y, sel, ty);  
  OR or0 (tx, ty, out);  
  
endmodule
```



**The order of the module instantiation does not matter**  
**Essentially describing the schematic textually**



# Behavioral Style with Continuous Assignments

- An *assign* statement represents continuously executing combinational logic

```
module MUX2_1 (x, y, sel, out);
```

```
  input x, y;
```

```
  input sel;
```

```
  output out;
```

```
  assign out = (~sel & x) | (sel & y);
```

```
endmodule
```

- Multiple continuous assignments happen in parallel; the order does not matter

# Assignments in Verilog

- **Continuous assignments apply to combinational logic only**
- ***Always blocks* contain a set of procedural assignments (blocking or nonblocking)**
  - **Can be used to model either combinational or sequential logic**
  - **Always blocks execute concurrently with other always blocks, instance statements, and continuous assignment statements in a module**

# (Behavioral Style) Combinational Logic with Always Blocks

```
module MUX2_1 (x, y, sel, out);  
  input x, y;  
  input sel;  
  output reg out;  
  
  always @(x, y, sel)  
  begin  
    out <= (~sel & x) | (sel & y);  
  end  
  
endmodule
```

- An *always* block is reevaluated whenever a signal in its sensitivity list changes
- Formed by *procedural assignment* statements
  - *reg* needed on the LHS of a procedural assignment

# Sequential Logic in Always Blocks

```
reg Q;
```

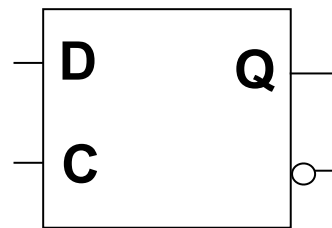
```
always @( clk, D )
```

```
begin
```

```
  if ( clk )
```

```
    Q <= D;
```

```
end
```



D latch

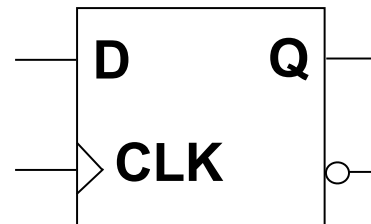
- Sequential logic can only be modeled using always blocks

```
always @( posedge clk )
```

```
begin
```

```
  Q <= D;
```

```
end
```



DFF

- Q must be declared as a “reg”

# Blocking Assignments ( = )

- Blocking assignments ( = )

```
input A, B;  
reg Y, Z;  
always @ (posedge clk)  
begin  
    Y = A & B;  
    Z = ~Y;  
end
```

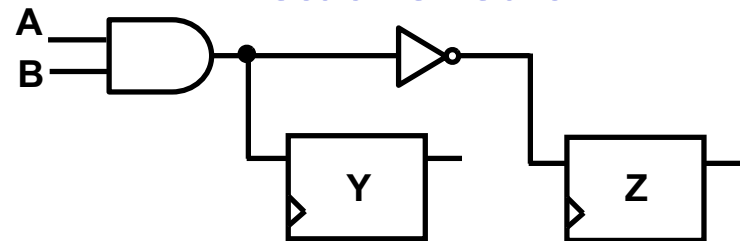
Y and Z are inferred as FFs since the always block is sensitive to the clock edge

## Simulation behavior

$Y_{\text{next}} = A \& B$   
 $Z_{\text{next}} = \sim(A \& B)$  /\* negating the new Y \*/

- Right-hand side (RHS) evaluated sequentially
- Assignment to LHS is immediate

## Actual circuit



When a reg appears on RHS of a blocking assignment ("Y" here), use its input for connection ("A&B" here)

# Nonblocking Assignments

- Nonblocking assignment ( `<=` )

```
input A, B;  
reg Y, Z;  
always @ (posedge clk)  
begin  
    Y <= A & B;  
    Z <= ~Y;  
end
```

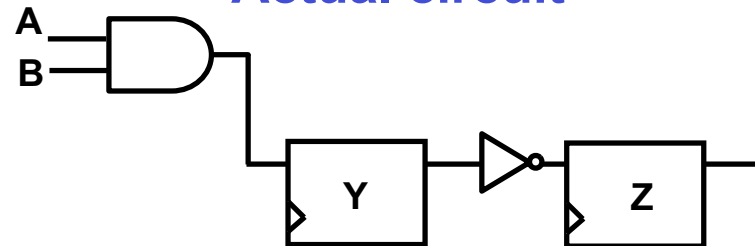
Y and Z are inferred as FFs since the always block is sensitive to the clock edge

## Simulation behavior

$Z_{\text{next}} = \sim Y$  /\* negating the old Y \*/  
 $Y_{\text{next}} = A \& B$

- RHS evaluated in parallel (order doesn't matter)
- Assignment to LHS is delayed until end of always block

## Actual circuit



When a reg appears on RHS of a nonblocking assignment (Y here), use its output for connection

# Before Next Class

- H&H 3.4, 4.6

## Next Time

### Finite State Machines