

Design of Digital Circuits and Systems

Algorithms to Hardware II, Timing Review

Instructor: Justin Hsia

Teaching Assistants:

Colton Harris

Deepti Anoop

Gayathri Vadhyan

Jared Yoder

Lancelot Wathieu

Matthew Hung

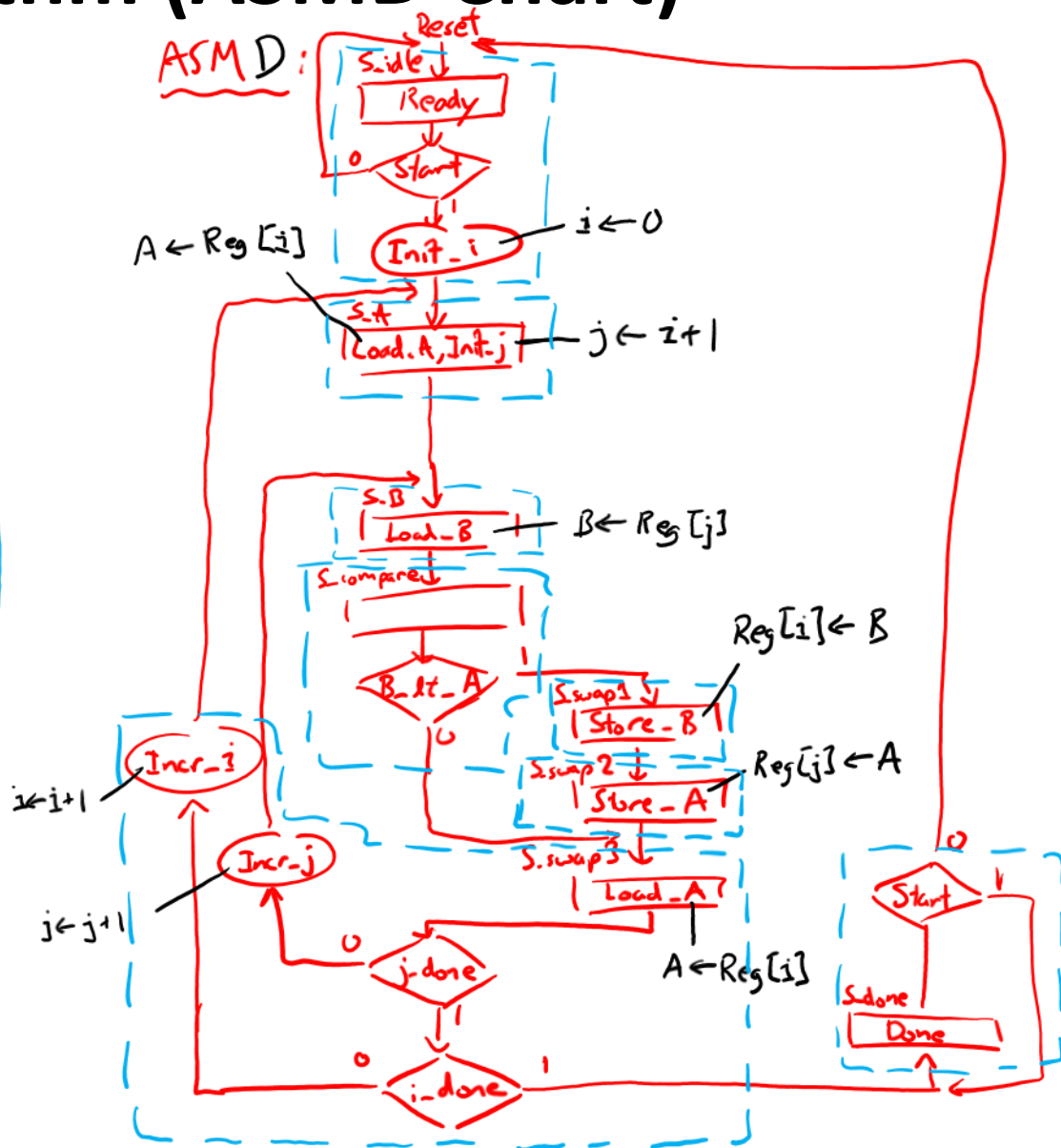
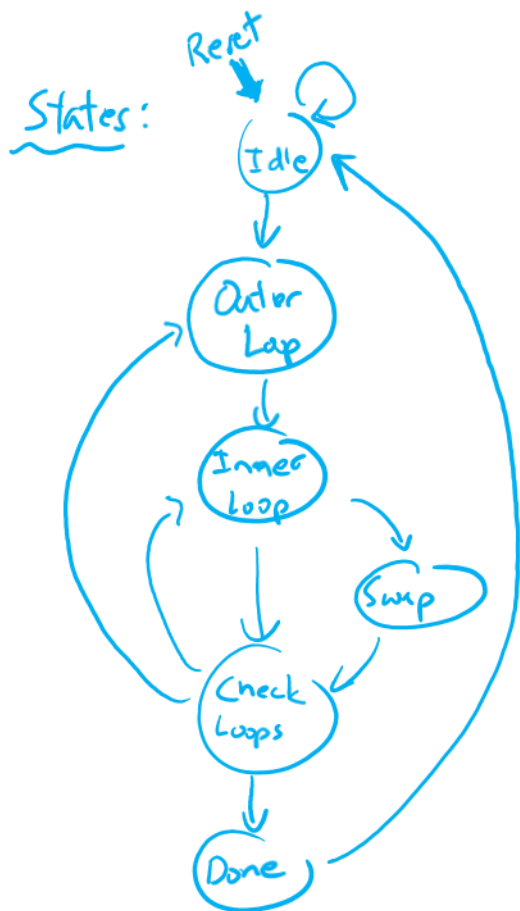
Relevant Course Information

- ❖ Anonymous mid-quarter survey on Canvas (due 4/29)
- ❖ Homework 4 due on Monday (4/29)
- ❖ Lab 3 due Friday (4/26)
- ❖ Lab 4 due next Friday (5/3)
- ❖ Lab 5 will be released next week, due 5/17
- ❖ Quiz 3 (ASM, ASMD) next Thursday (5/2)

Outline

- ❖ **Algorithm → Hardware Wrap-up**
- ❖ Timing Constraints Review

Sorting Algorithm (ASMD Chart)



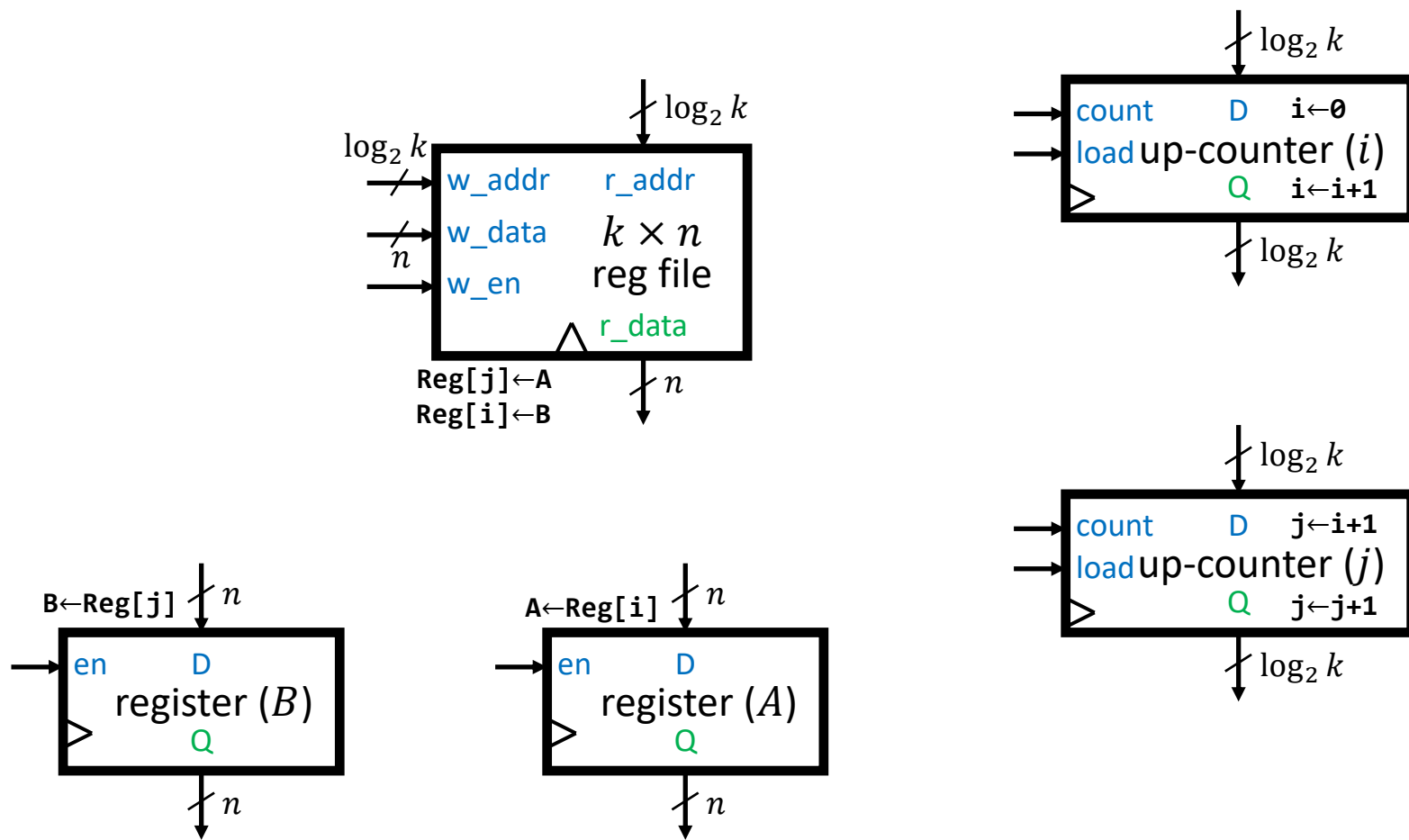
Sorting Algorithm Datapath

Control Signals:

Init_i, Incr_i
 Init_j, Incr_j
 Load_A, Store_A
 Load_B, Store_B

Status Signals:

B_lt_A
 i_done
 j_done



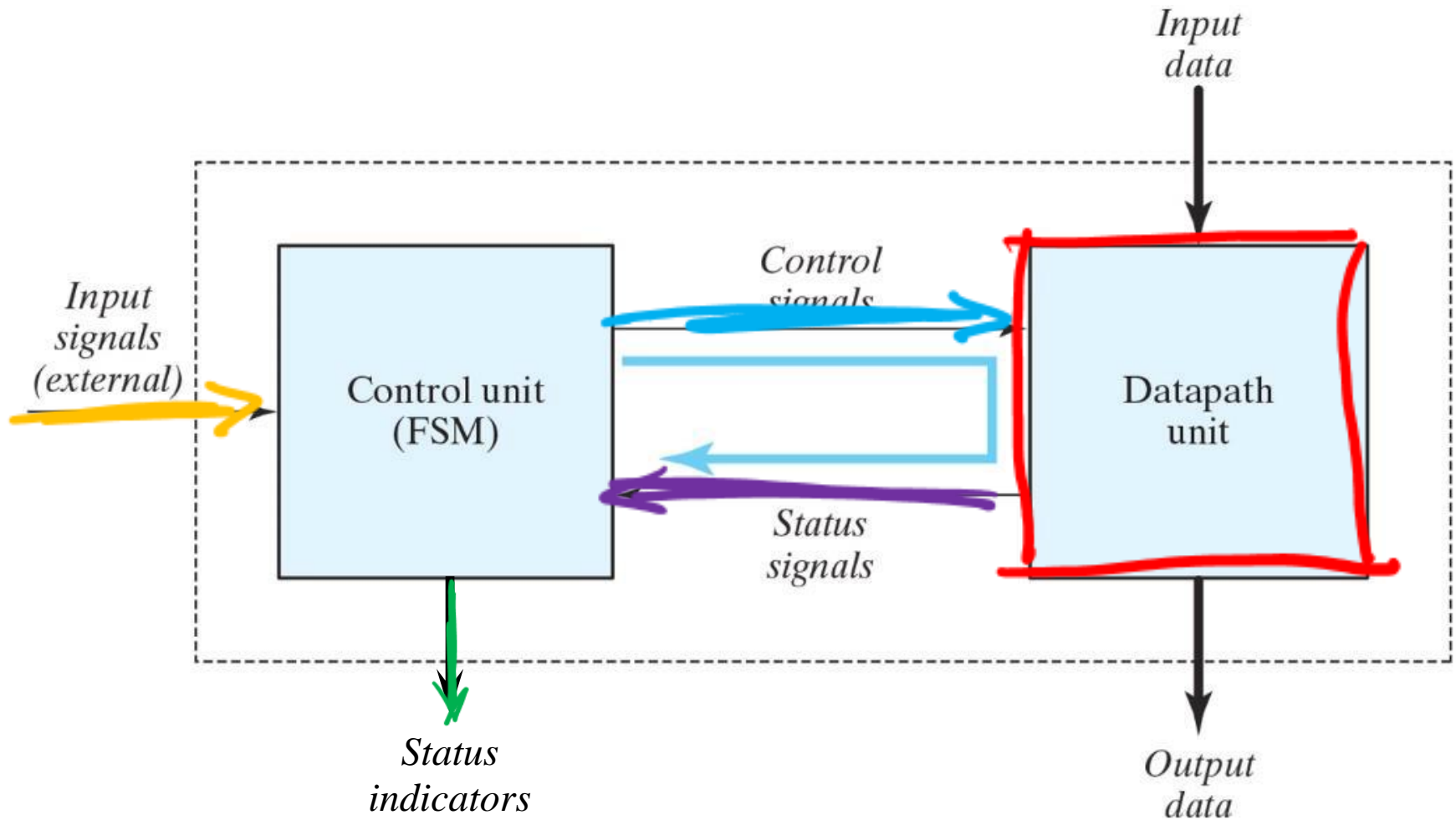
isPrime Algorithm

- ❖ Design a sequential circuit that determines if an input **Num** is a prime number (e.g., 2, 3, 5, 7) or not
 - **Num** has width **W**, output **P** is **1** (prime) or **0** (not prime)
 - Assume the usual **Ready**, **Done**, **Start**, **Reset**
- ❖ Algorithm:

```
N = Num
if N < 3 do           // handle special cases
    return (N == 2)
for F = 2 to N/2 do  // factor to check
    if (N % F == 0) do
        return 0
    endif
endfor
return 1
```

isPrime (ASMD Chart)

isPrime (Block Diagram)



Short Tech Break

isPrime Live Coding Demo

```
isPrime_control:
```

```
    // port definitions  
    // define state names and variables  
    // controller logic w/synchronous reset  
    // next state logic  
    // output assignments
```

```
isPrime_datapath:
```

```
    // port definitions  
    // internal datapath signals and registers  
    // datapath logic  
    // output assignments
```

```
isPrime:
```

```
    // port definitions  
    // define status and control signals  
    // instantiate control and datapath
```

Basic Testbench Review

❖ How to start a testbench:

- 1) Create a `<name>_tb` module with no ports
- 2) Create variables that match the module's ports
- 3) Instantiate an instance of the module can call it dut or uut
- 4) If needed, create a simulated clock
- 5) Create an `initial` block to define your test inputs

❖ Tools we know about:

- Timing controls: `#<time>;, @(posedge <signal>);`
- Loops: `integer i; for (i = 0; i < 2**3; i++)
repeat (2**4)`

Testing Your Algorithm

- ❖ Create a testbench for our `isPrime` module:

```
module isPrime_tb ();  
    // define parameters  
  
    // define module port connections  
  
    // instantiate module  
  
    // create simulated clock  
  
    // define test inputs  
  
endmodule
```

Testing Your Algorithm

- ❖ Create a testbench for our `isPrime` module:
 - Single `Num` value (arbitrary wait):

```
// define test inputs

initial begin
    Num = 3;
    Reset = 1; Start = 0; @(posedge clk);
    Reset = 0;          @(posedge clk);
    Start = 1;          @(posedge clk);
    Start = 0;

    repeat (2**4)      // wait 16 clock cycles
        @(posedge clk);

    @(posedge clk); // extra cycle of output
    $stop();
end
```

Testing Your Algorithm

- ❖ Create a testbench for our `isPrime` module:
 - Single `Num` value (check `Ready`):

```
// define test inputs

initial begin
    Num = 3;
    Reset = 1; Start = 0; @(posedge clk);
    Reset = 0;          @(posedge clk);
    Start = 1;          @(posedge clk);
    Start = 0;

    @(posedge Ready); // wait until module says it's ready

    @(posedge clk); // extra cycle of output
    $stop();
end
```

Testing Your Algorithm

- ❖ Create a testbench for our `isPrime` module:
 - All `Num` values:

```
// define test inputs
integer i;
initial begin
    Reset = 1; Start = 0; @(posedge clk);
    Reset = 0;           @(posedge clk);

    for (i = 0; i < 2**W; i++) begin
        Start = 1; Num = i; @(posedge clk);
        Start = 0;         @(posedge Ready);
    end

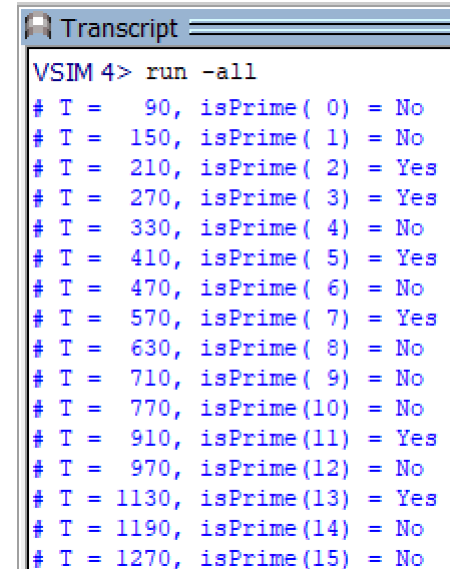
    @(posedge clk); // extra cycle of output
    $stop();
end
```

Testing Your Algorithm: \$display

- ❖ Triggers once when encountered, prints the given format string and adds a new line:

```
// define test inputs
integer i;
initial begin
  Reset = 1; Start = 0; @(posedge clk);
  Reset = 0;          @(posedge clk);
  for (i = 0; i < 2**W; i++) begin
    Start = 1; Num = i; @(posedge clk);
    Start = 0;          @(posedge Ready);

    $display("T = %4t, isPrime(%2d) = %s",
             $time, Num, P ? "Yes" : "No ");
  end
  @(posedge clk); // extra cycle of output
  $stop();
end
```



```
Transcript
VSIM 4> run -all
# T = 90, isPrime( 0) = No
# T = 150, isPrime( 1) = No
# T = 210, isPrime( 2) = Yes
# T = 270, isPrime( 3) = Yes
# T = 330, isPrime( 4) = No
# T = 410, isPrime( 5) = Yes
# T = 470, isPrime( 6) = No
# T = 570, isPrime( 7) = Yes
# T = 630, isPrime( 8) = No
# T = 710, isPrime( 9) = No
# T = 770, isPrime(10) = No
# T = 910, isPrime(11) = Yes
# T = 970, isPrime(12) = No
# T = 1130, isPrime(13) = Yes
# T = 1190, isPrime(14) = No
# T = 1270, isPrime(15) = No
```

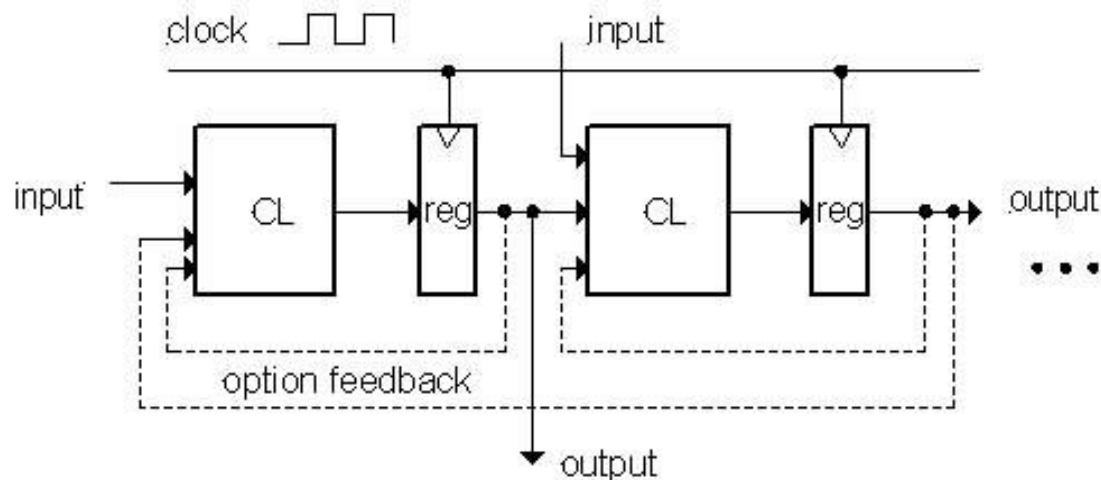

Short Tech Break

Outline

- ❖ Algorithm → Hardware Wrap-up
- ❖ **Timing Constraints Review**

Why Are Timing Constraints Important?

- ❖ Our model of synchronous digital systems relies on timing behavior of the clock and logic stages
 - Combinational logic blocks separated by registers

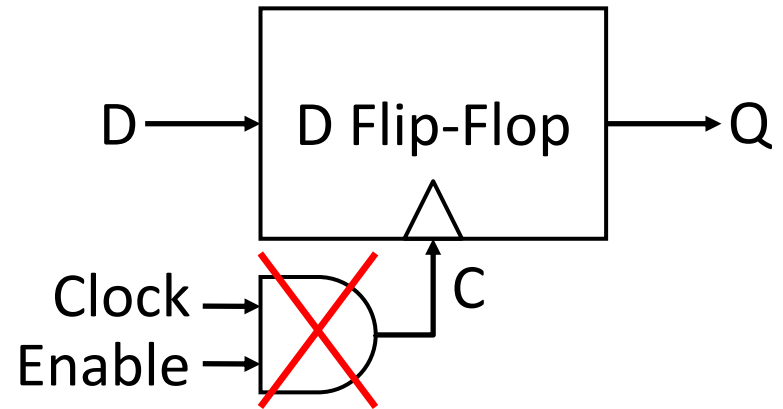


- Violations can cause incorrect behavior or can cause produced semiconductor circuits to fail!
- Timing considerations can limit how fast our system/clock can run

Review: Timing Issues

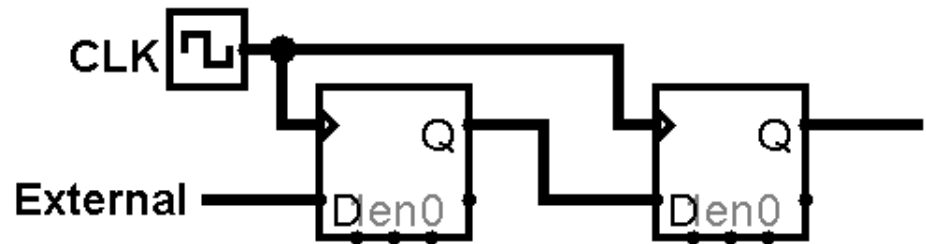
❖ NEVER GATE THE CLOCK!!!

- Delays can cause part of circuit to get out of sync with rest
 - Adds to *clock skew*
- Hard to track non-uniform triggers



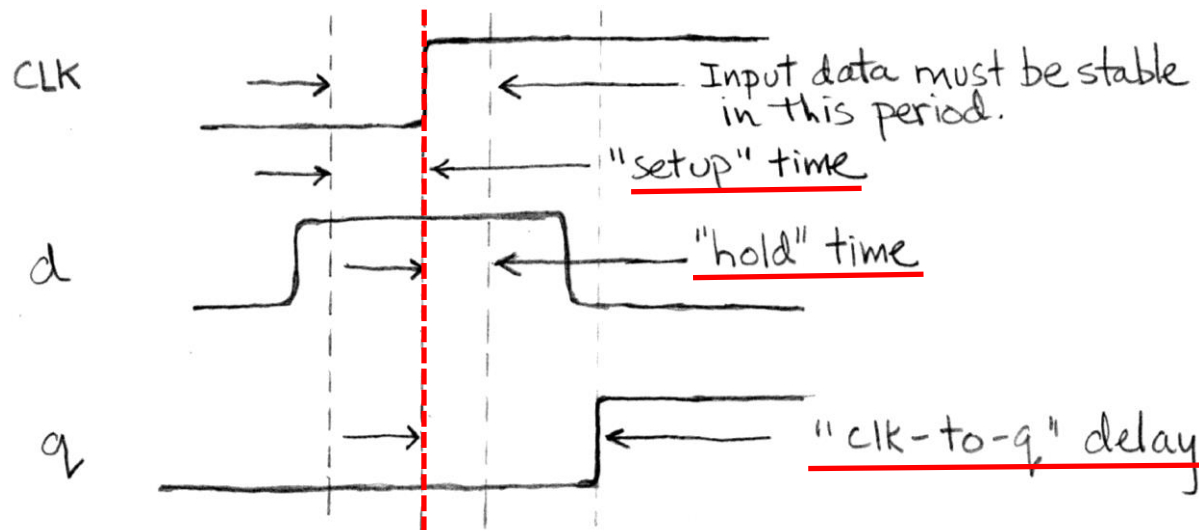
❖ **Metastability** is the ability of a digital system to persist for an unbounded time in an unstable equilibrium or metastable state

- Add chains of registers to allow to settle:



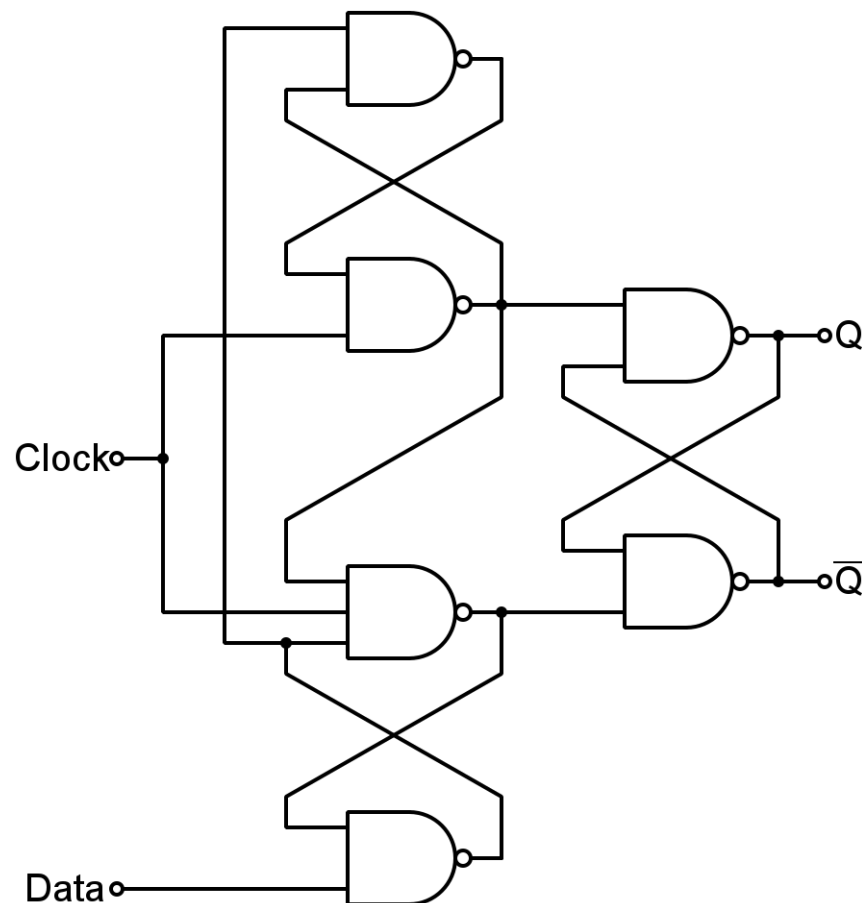
Review: Sequential Timing Constraints

- ❖ *Setup Time (t_s or t_{su}):* how long the input must be stable *before* the CLK trigger for proper input read
- ❖ *Hold Time (t_h):* how long the input must be stable *after* the CLK trigger for proper input read
- ❖ *"CLK-to-Q" Delay (t_{c2q} or t_{co}):* how long it takes the output to change, measured from the CLK trigger



Where Do Timing Terms Come From?

Edge-triggered
D flip-flop:



By Nolanjshettle at English Wikipedia, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=40852354>

When Can the Input Change?

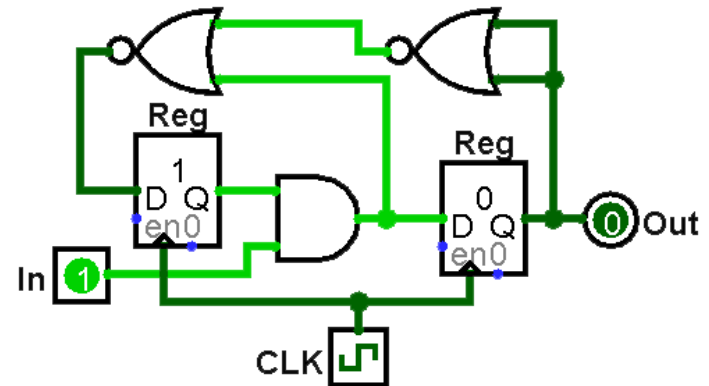
- ❖ When a register input changes shouldn't violate hold time (t_h) or setup time (t_{su}) constraints within a clock period (T)
- ❖ Let $t_{input,i}$ be the time it takes for the input of a register to change for the i -th time in a single clock cycle, measured from the CLK trigger:
 - Then we need $t_h \leq t_{input,i} \leq T - t_{su}$ for all i
 - Two separate constraints!

Timing Constraint Considerations

- ❖ Use *worst-case analysis*
 - Asynchronous inputs are *really* problematic
 - Use the **critical path** – the longest delay between *any* two registers in a circuit – for setup time constraint
 - Use the shortest path for the hold time constraint
- ❖ A timing violation could be caused by a signal change from the previous clock cycle
 - A really late input change could violate the hold time constraint in the next clock cycle
- ❖ Setup time constraint helps determine feasibility of clock frequency: $\text{max freq} = 1/(\text{min period})$

Timing Constraint Example

- ❖ $t_{su} = 12$ ns, $t_h = 18$ ns,
 $t_{CO} = 8$ ns, $t_{AND} = 25$ ns,
and $t_{NOR} = 20$ ns



- If In changes t_{CO} after each clock trigger, what is the **minimum clock period** we can use?
- If we use the minimum clock period, when within a clock period can In change without causing a timing violation?