ECE 550D
Fundamentals of Computer Systems and Engineering

Fall 2023

Storage and Clocking

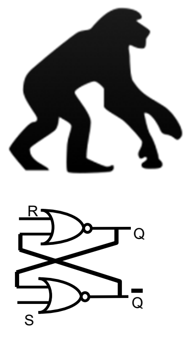
Xin Li & Dawei Liu
Duke Kunshan University

Slides are derived from work by
Andrew Hilton, Tyler Bletsch and Rabih Younes (Duke)

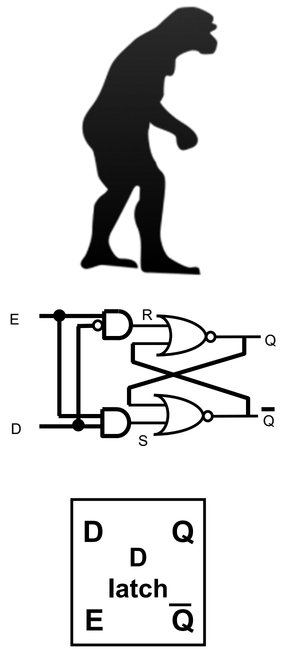
Last time…

- Who can remind us what we did last time?
  - SR latch
  - D latch
  - Clocks

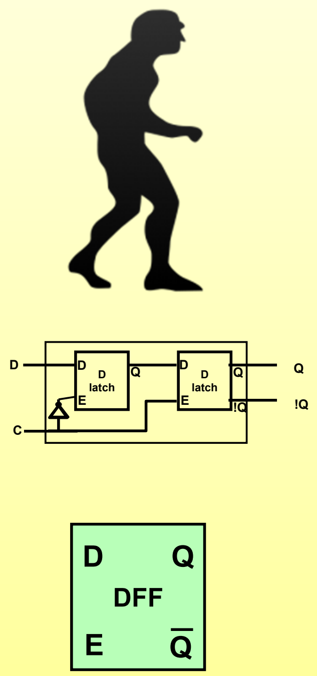
Building up to the D Flip-Flop and beyond



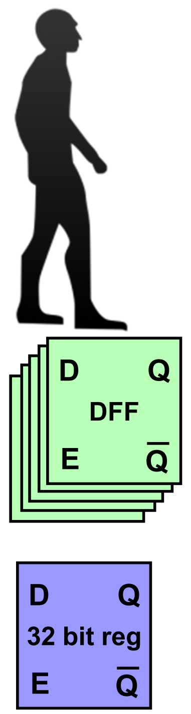
SR Latch
(too awkward)



D Latch
(bad timing)



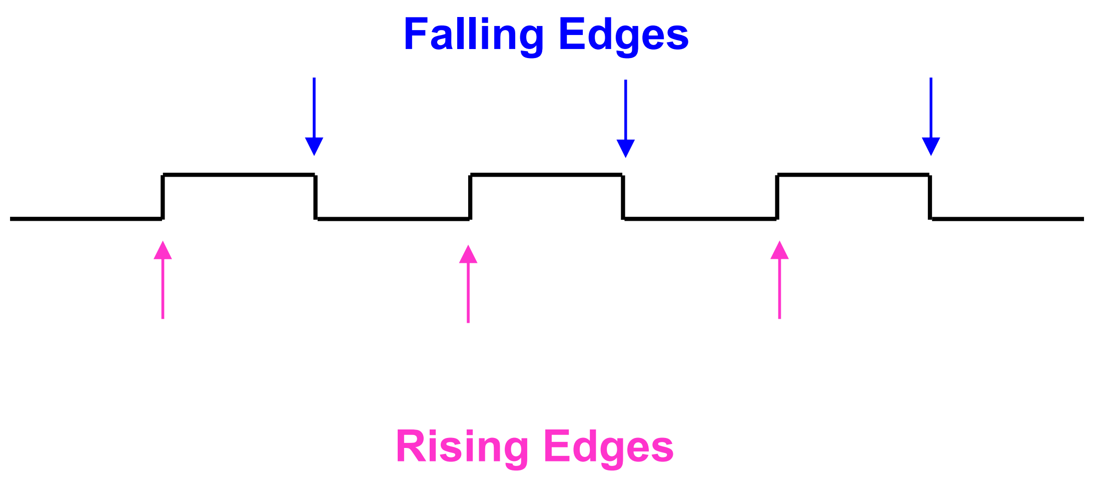
D Flip-Flop
(okay but only one bit)



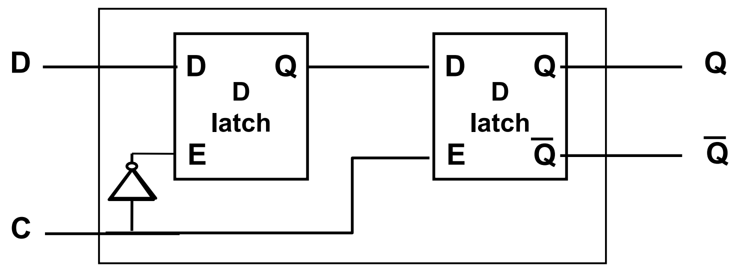
Register
(nice!)

DFF: Edge Triggered

- Instead of level triggered
  - Latch a new value at a clock level (high or low)
- We use edge triggered
  - Latch a value at an clock edge (rising or falling)

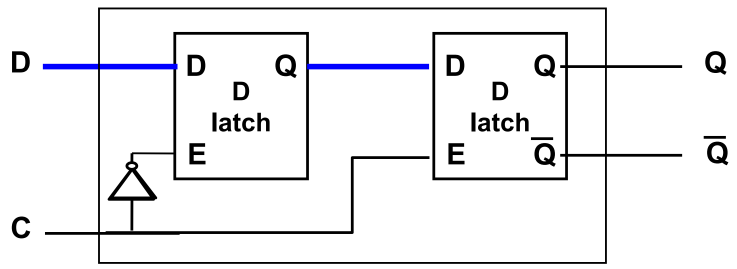


Our Ultimate Goal: D Flip-Flop



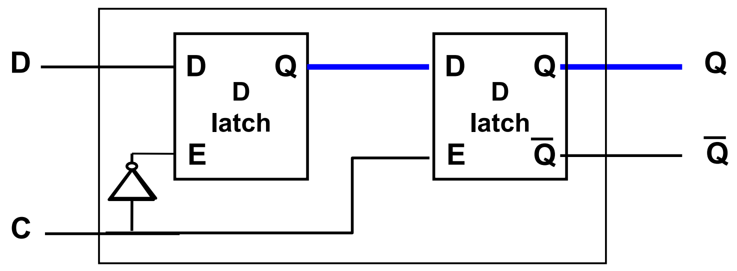
- Rising edge triggered D Flip-flop
  - Two D Latches w/ opposite clking of enables

D Flip-Flop



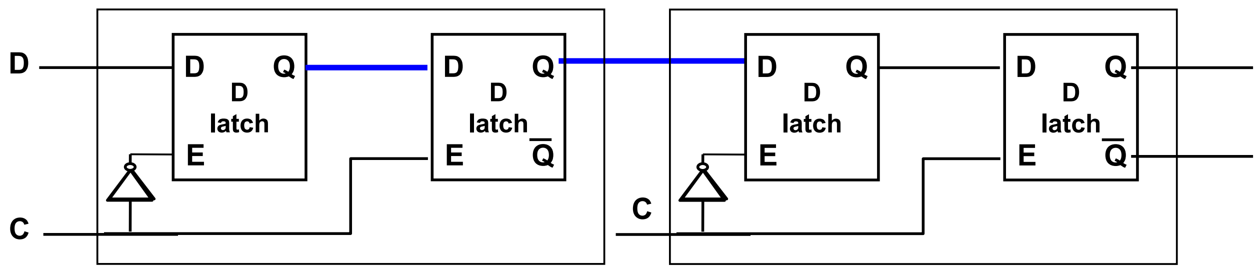
- Rising edge triggered D Flip-flop
  - Two D Latches w/ opposite clking of enables
  - On Low Clk, first latch enabled (propagates value)
    - Second not enabled, maintains value

D Flip-Flop



- Rising edge triggered D Flip-flop
  - Two D Latches w/ opposite clking of enables
  - On Low Clk, first latch enabled (propagates value)
    - Second not enabled, maintains value
  - On High Clk, second latch enabled
    - First latch not enabled, maintains value

D Flip-Flop



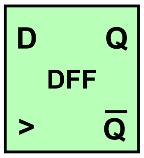
- No possibility of "races" anymore
  - Even if I put 2 DFFs back-to-back…
  - By the time signal gets through 2nd latch of 1st DFF
  1st latch of 2nd DFF is disabled
- Still must ensure signals reach DFF before clk rises
  - Important concern in logic design "making timing"

D Flip-flops

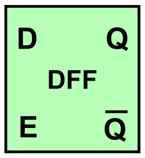
- Could also do falling edge triggered
  - Switch which latch has NOT on clk
- D Flip-flop is ubiquitous
  - Typically people just say "latch" and mean DFF
  - Which edge: doesn't matter
    - As long as consistent in entire design
    - We'll use rising edge

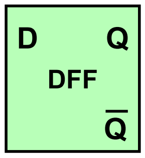
D flip flops

- Generally don't draw clk input
  - Have one global clk, assume it goes there
  - Often see > as symbol meaning clk



- Maybe have explicit enable
  - Might not want to write every cycle
  - If no enable signal shown, implies always enabled





- Get output and NOT(output) for "free"

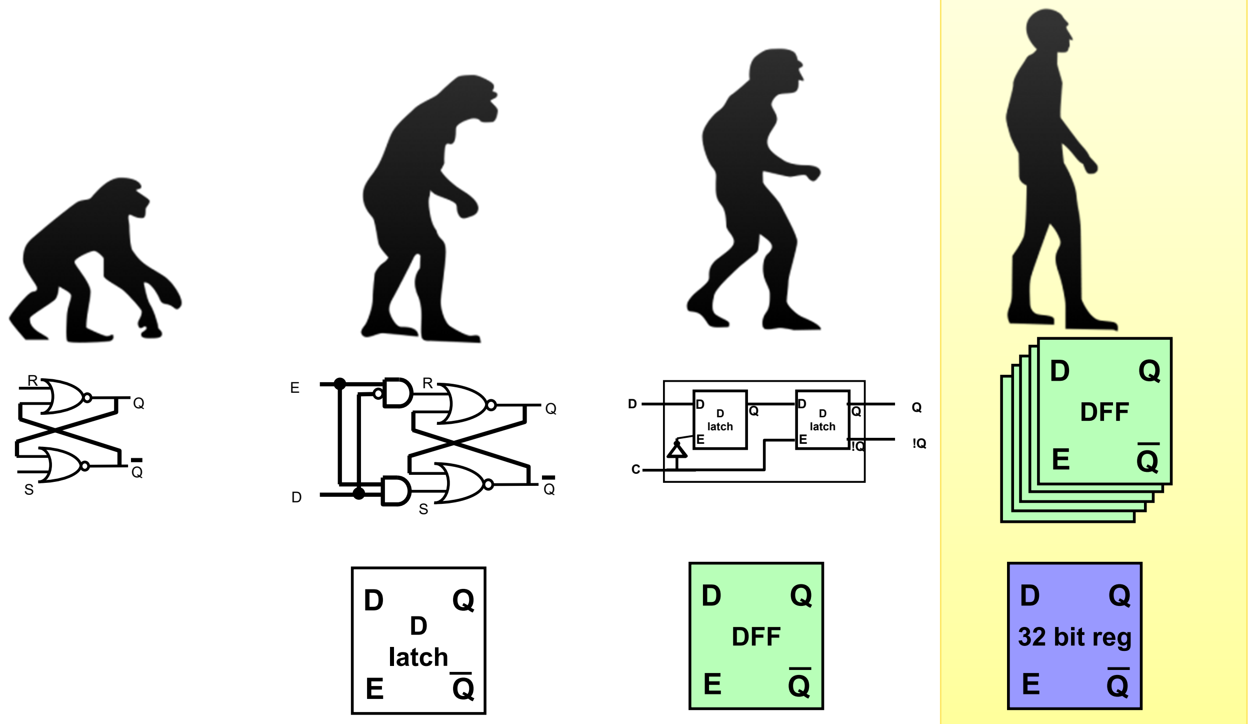
A few words about timing

- Quartus will tell you what timing you make
  - Fmax : how fast can this be clocked
  - Tells you your worst timing paths
    - From which dff to which dff
    - Can see in schematic viewer (usually)

Fixing timing misses

- Typical approach: reduce logic (gate delays)
  - Better adder?
  - Rethink approach?
  - Change “don’t care” behavior?
- Fix high fanout
  - Duplicate high FO/simple logic
- Also, feel free to ask for help from me/TAs
  - Quartus’s tools to help you fix them aren’t the best

Building up to the D Flip-Flop and beyond



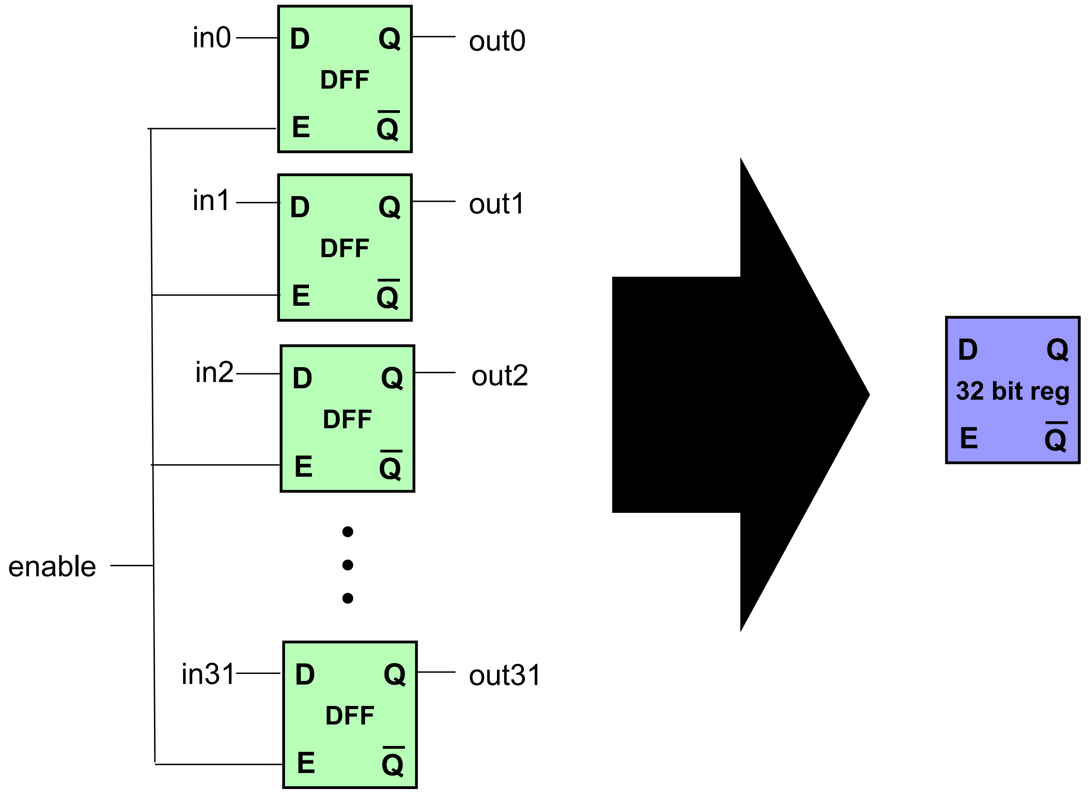
SR Latch
(too awkward)

D Latch
(bad timing)

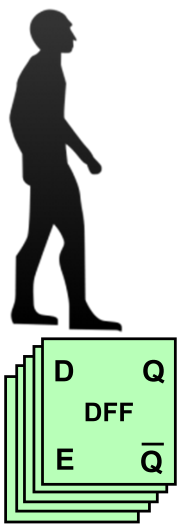
D Flip-Flop
(okay but only one bit)

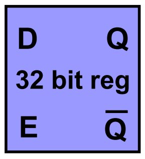
Register
(nice!)

Stick a bunch of DFFs together to make a register

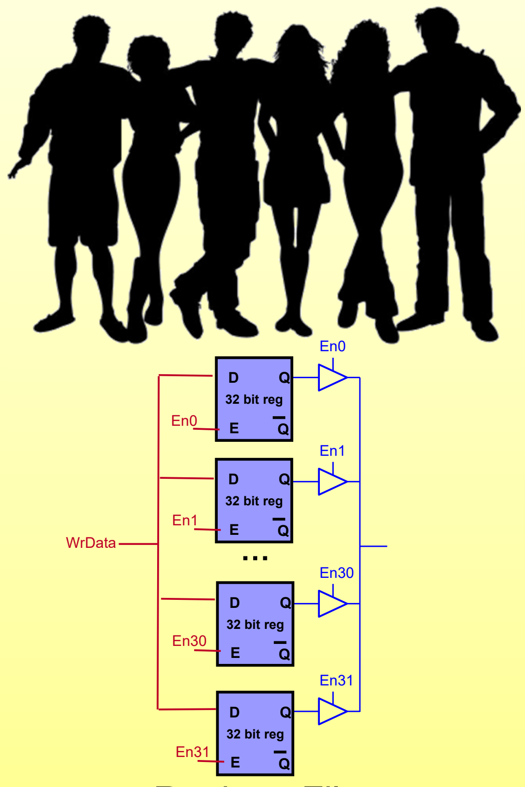


Next evolution: multiple registers





Register
(*nice!*)

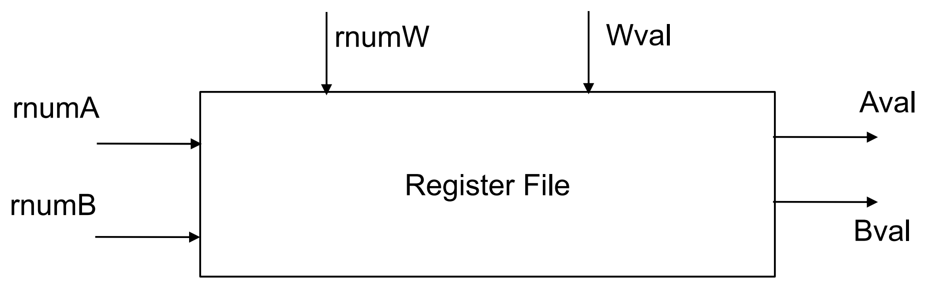


Register File
(*Tremendous!*)

Register File

- Can store one value… How about many?
- Register File
    - In processor, holds values it computes on
    - MIPS, 32 32-bit registers
- What other components do we need?

Register File: Interface



- 4 inputs
  - 3 register numbers (5 bit): 2 read, 1 write
  - 1 register write value (32 bits)
- 2 outputs
  - 2 register values (32 bits)

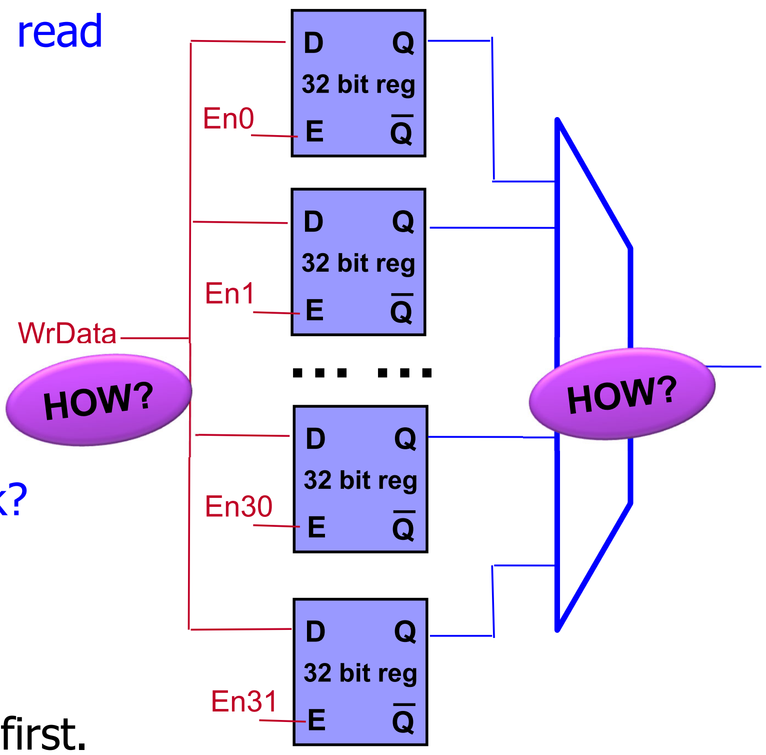
Register File Design

- Two problems: write and read

- **Writing** the registers
  - Need to pick which reg
  - Have reg num (e.g., 19)
  - Need to make En19=1
    - En0, En1,… = 0

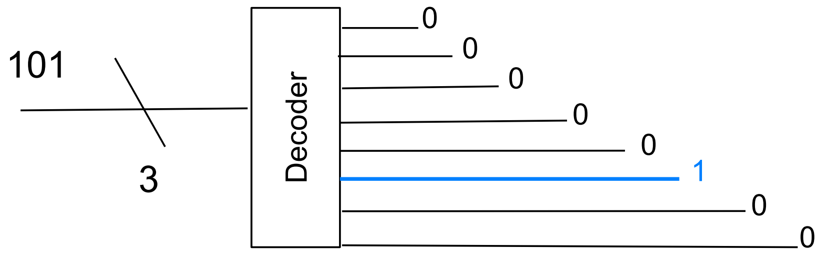
- **Read**: Use a mux to pick?
  - 32-input mux = slow
  - Need a better method…

- Let’s talk about **writing** first.



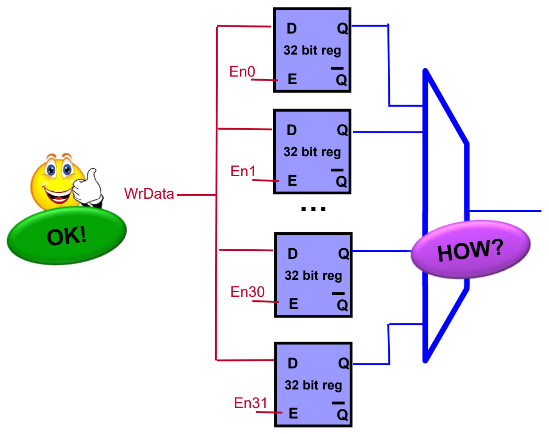
First: A Decoder

- First task: convert binary number to "one hot"
  - Saw this before
  - Take register number



Register File

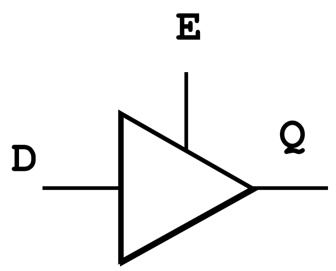
- Now we know how to **write**:
  - Use decoder to convert reg # to one hot
  - Send write data to all regs
  - Use one hot encoding of reg # to enable right reg
- Still need to fix **read** side
  - 32 input mux (the way we've made it) not realistic
  - To do this: expand our world from {1,0} to {1, 0, Z}



So this third option: Z

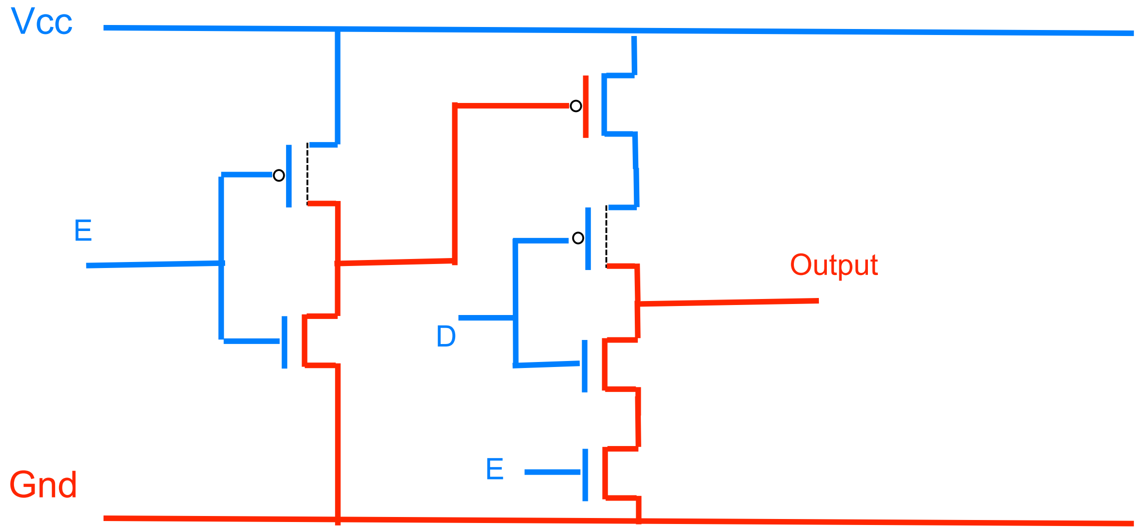
- There is a third possibility: Z ("high impedance")
  - Output is "disconnected"

- Gate that gives us Z : Tri-state



D	E	Q
0	1	0
1	1	1
-	0	Z

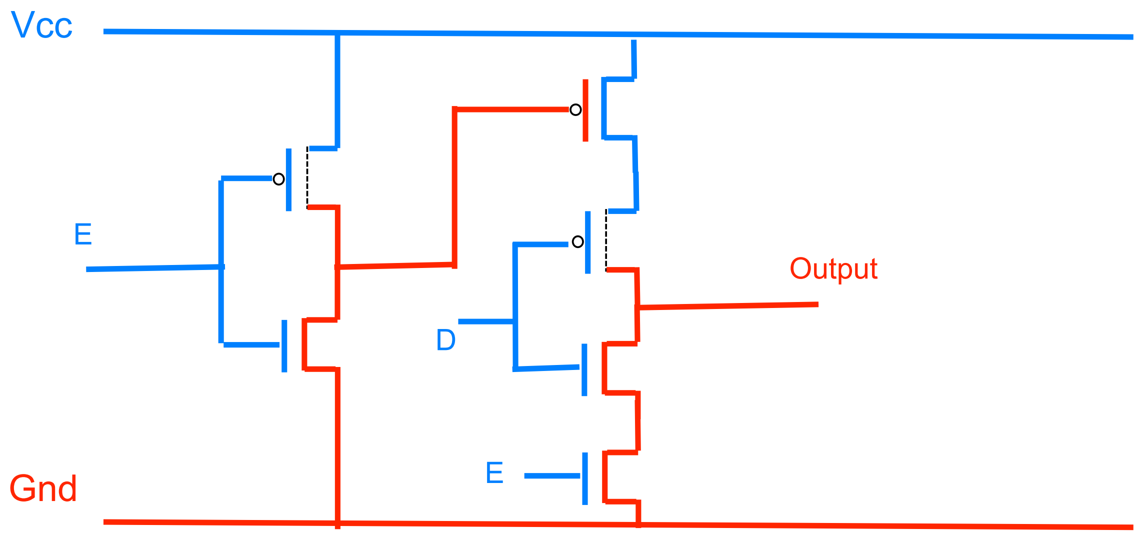
CMOS: Complementary MOS



- 2 inputs: E and D. What does this do?
  - Write truth table for output

E	D	Output
0	0	
0	1	
1	0	
1	1	

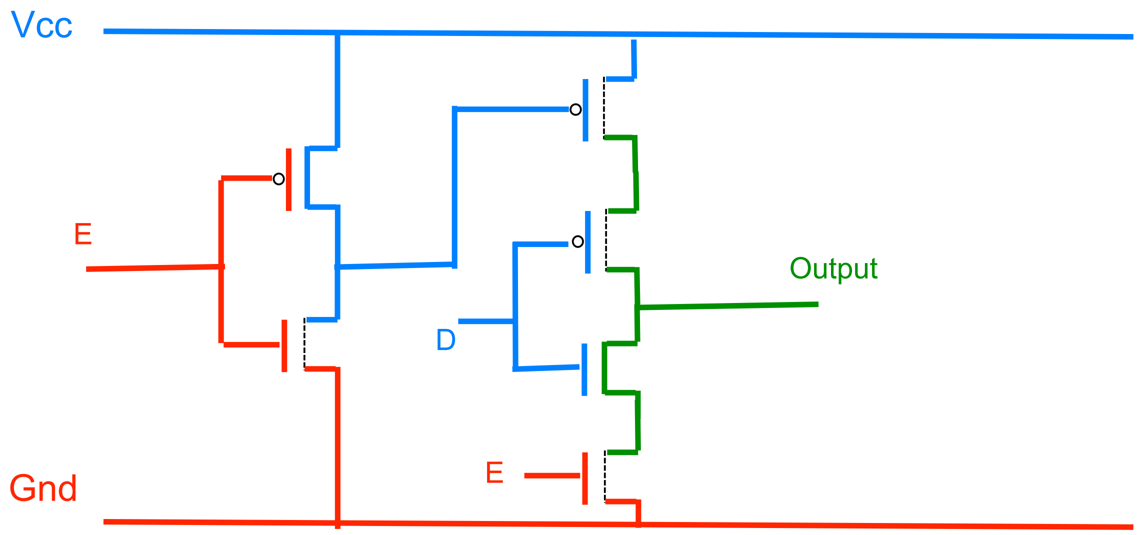
CMOS: Complementary MOS



- 2 inputs: E and D. What does this do?
  - Write truth table for output
  - When E =1, straightforward

E	D	Output
0	0	
0	1	
1	0	1
1	1	0

CMOS: Complementary MOS

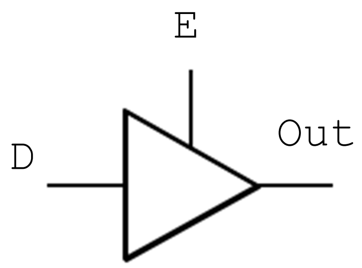


- 2 inputs: E and D. What does this do?
  - Write truth table for output
  - When E =1, straightforward
  - When E= 0, no connection: Z

E	D	Output
0	0	Z
0	1	Z
1	0	1
1	1	0

High Impedance: Z

- Z = High Impedance
  - No path to power or ground
  - "Gate" does not produce a 1 or a 0
- Previous slide: **tri-state** inverter
  - More commonly drawn: tri-state buffer
  - E = enable, D = data

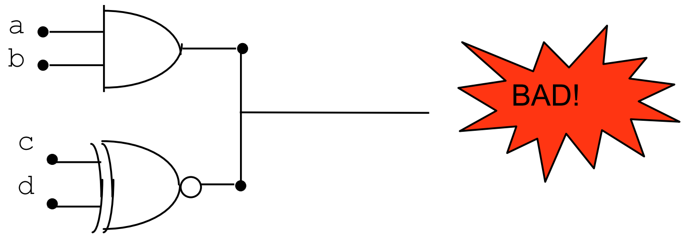


D	E	Out
0	1	0
1	1	1
X	0	Z

X="Don't care"

Remember this rule?

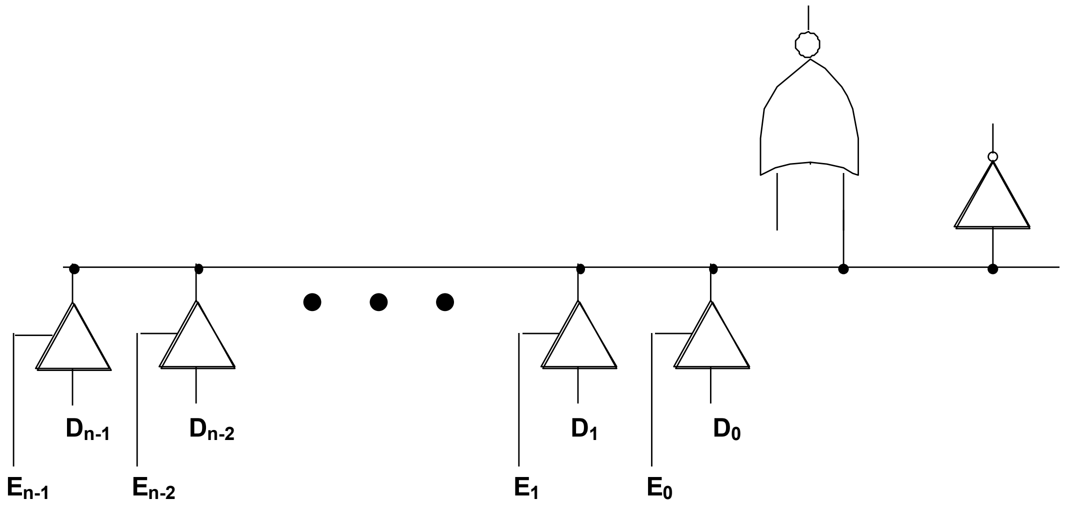
- Remember I told you not to connect two outputs?



- If one gate tries to drive a 1 and the other drives a 0
  - "Short circuit" — lots of current -> lots of heat

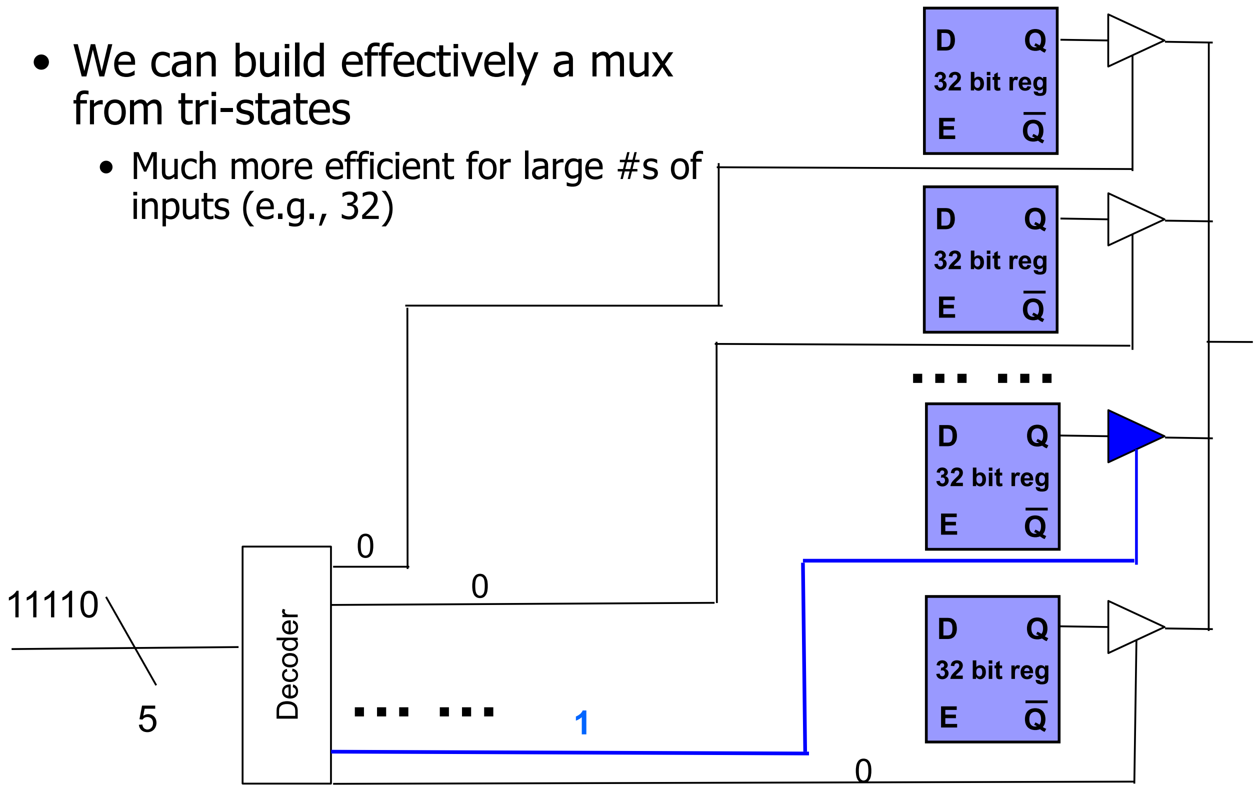
We've had this rule one day… and you break it

It's ok to connect multiple outputs together
Under one circumstance:
All but one must be outputting Z at any time



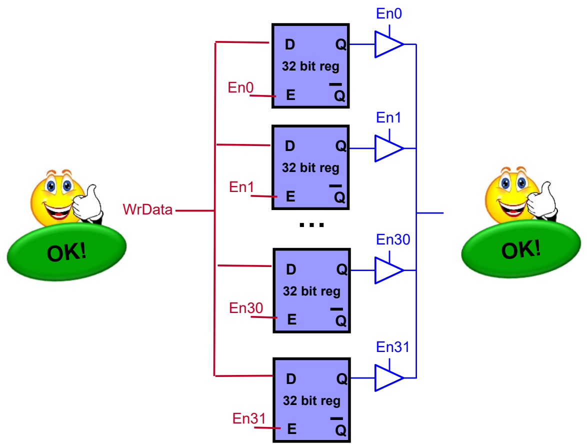
Mux, implemented with tri-states

- We can build effectively a mux from tri-states
  - Much more efficient for large #s of inputs (e.g., 32)



Register File

- Now we can **write** and **read** in one clock cycle!



Ports

- What we just saw: **read** port
  - Ability to do one read / clock cycle
  - May want more: read 2 source registers per instr
    - Maybe even more if we do many instrs at once
  - This design: can just replicate port
    - Another decoder
    - Another set of tri-states
    - Another output bus (wire connecting the tri-states)

- Earlier: **write** port
  - Ability to do one write/cycle
  - Could add more: need muxes to pick wr values

Minor Detail

- FYI: This is not how a register file is implemented
  - (Though it is how other things are implemented)
  - Actually done with SRAM
  - We'll see how those work soon

Summary

Storage devices

D flip-flops

Registers