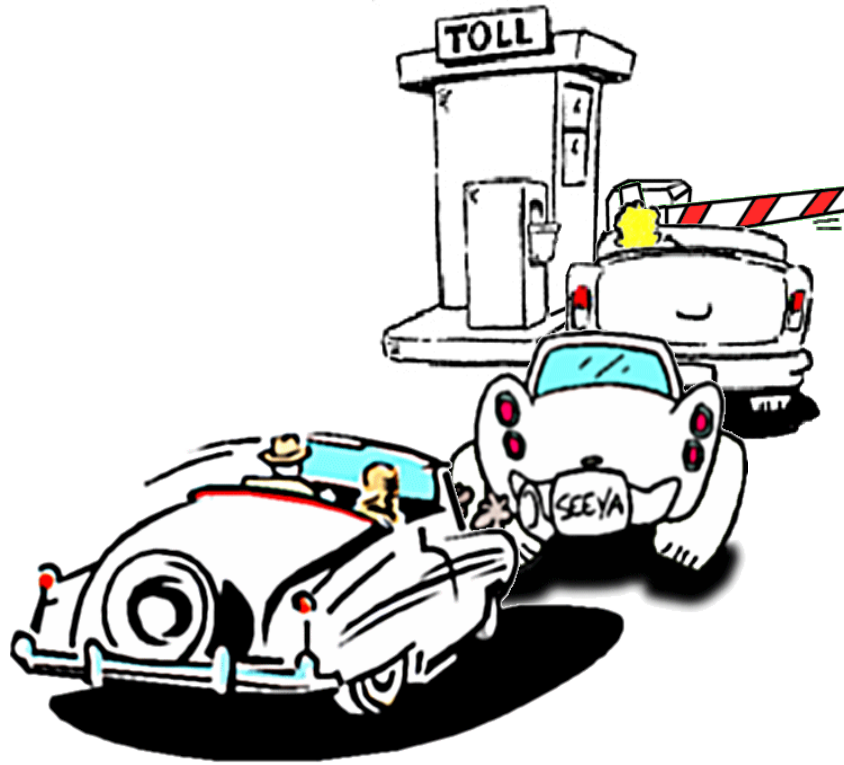
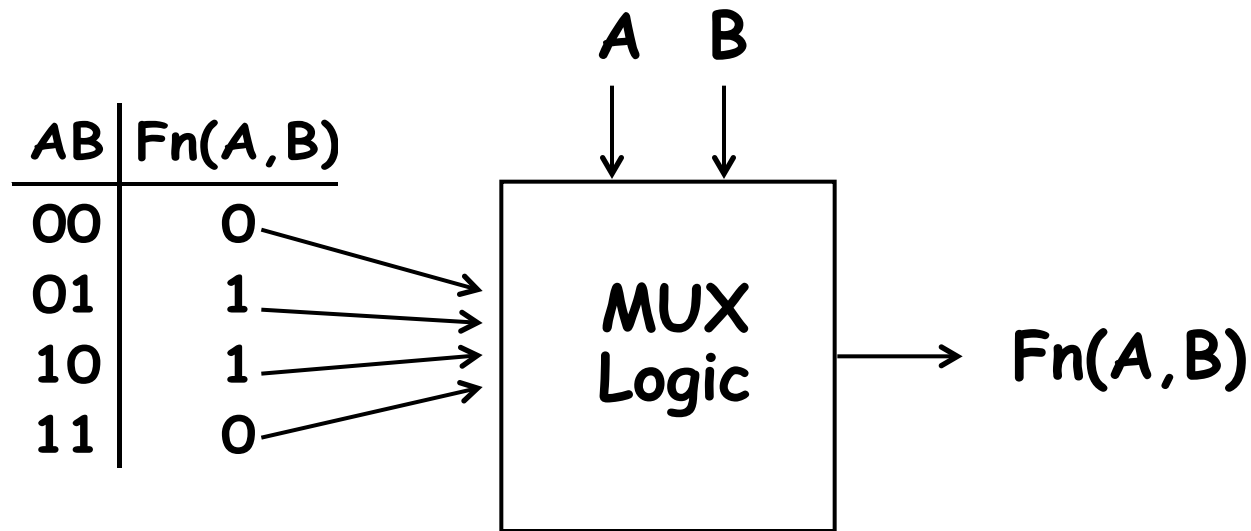


Memory, Latches, & Registers



- 1) Structured Logic Arrays
- 2) Memory Arrays
- 3) Transparent Latches
- 4) How to save
a few bucks
at toll booths
- 5) Edge-triggered Registers

General Table Lookup Synthesis



Generalizing:

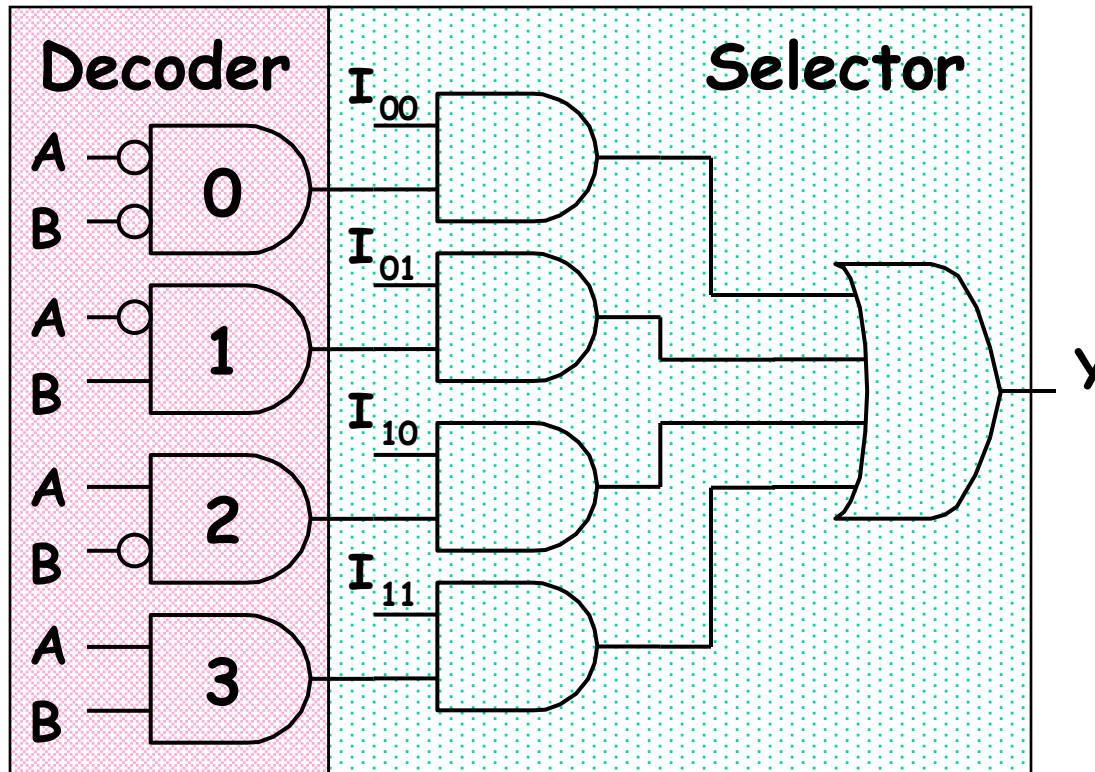
Remember from a few lectures ago that, in theory, we can build any 1-output combinational logic block with multiplexers.

For an N-input function we need a 2^N input multiplexer.

BIG Multiplexers? How about 10-input function? 20-input?

A Mux's Guts

A decoder generates all possible product terms for a set of inputs



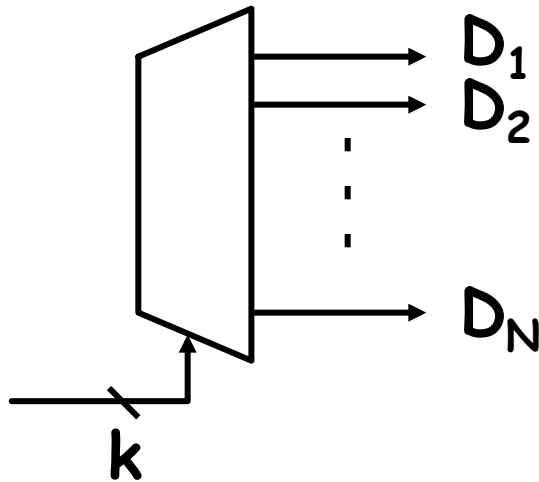
Multiplexers can be partitioned into two sections.

A DECODER that identifies the desired input, and

a SELECTOR that enables that input onto the output.

Hmmm, by sharing the decoder part of the logic MUXs could be adapted to make lookup tables with any number of outputs

A New Combinational Device



DECODER:

k SELECT inputs,

$N = 2^k$ DATA OUTPUTS.

Selected D_j HIGH;
all others LOW.

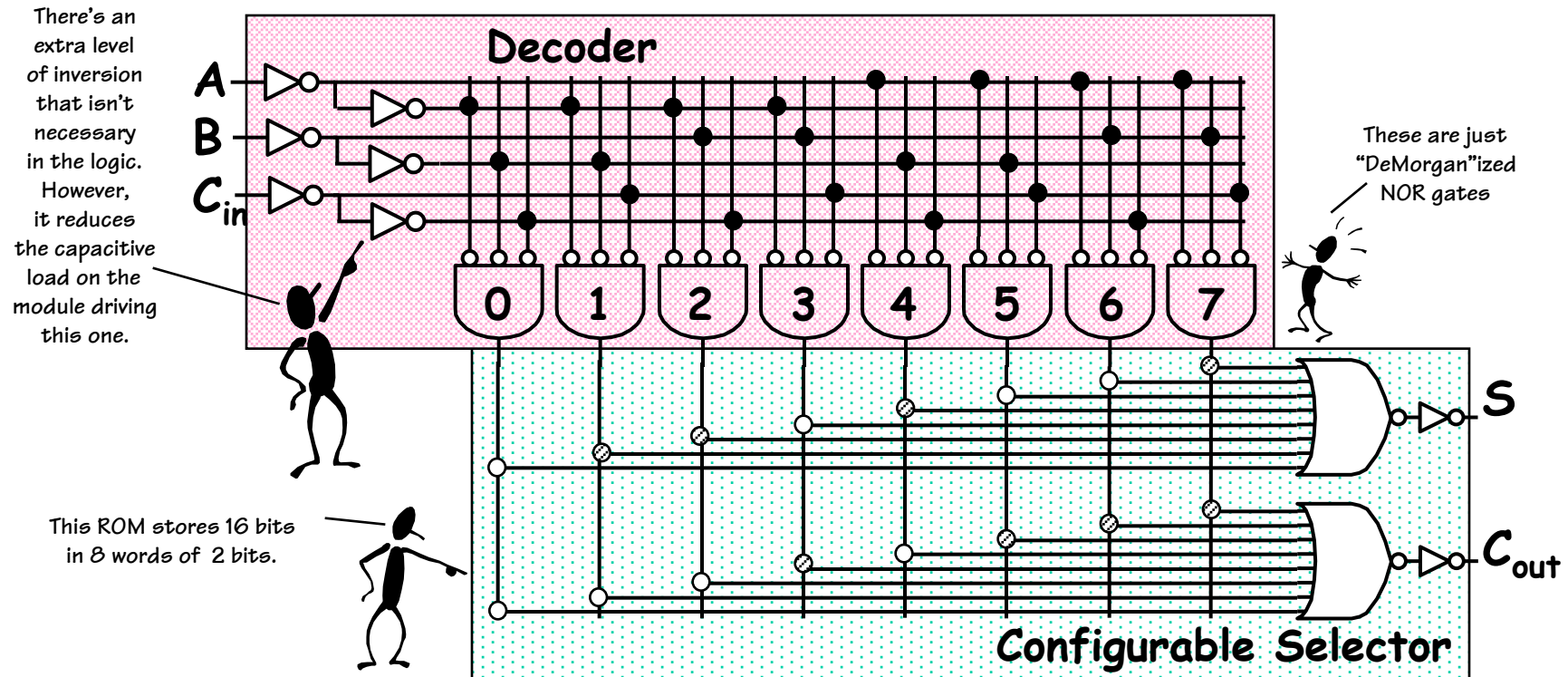
Have I
mentioned
that HIGH
is a synonym
for '1' and
LOW means
the same
as '0'



NOW, we are well on our way to building a general purpose table-lookup device.

We can build a 2-dimensional ARRAY of decoders and selectors as follows ...

Shared Decoding Logic

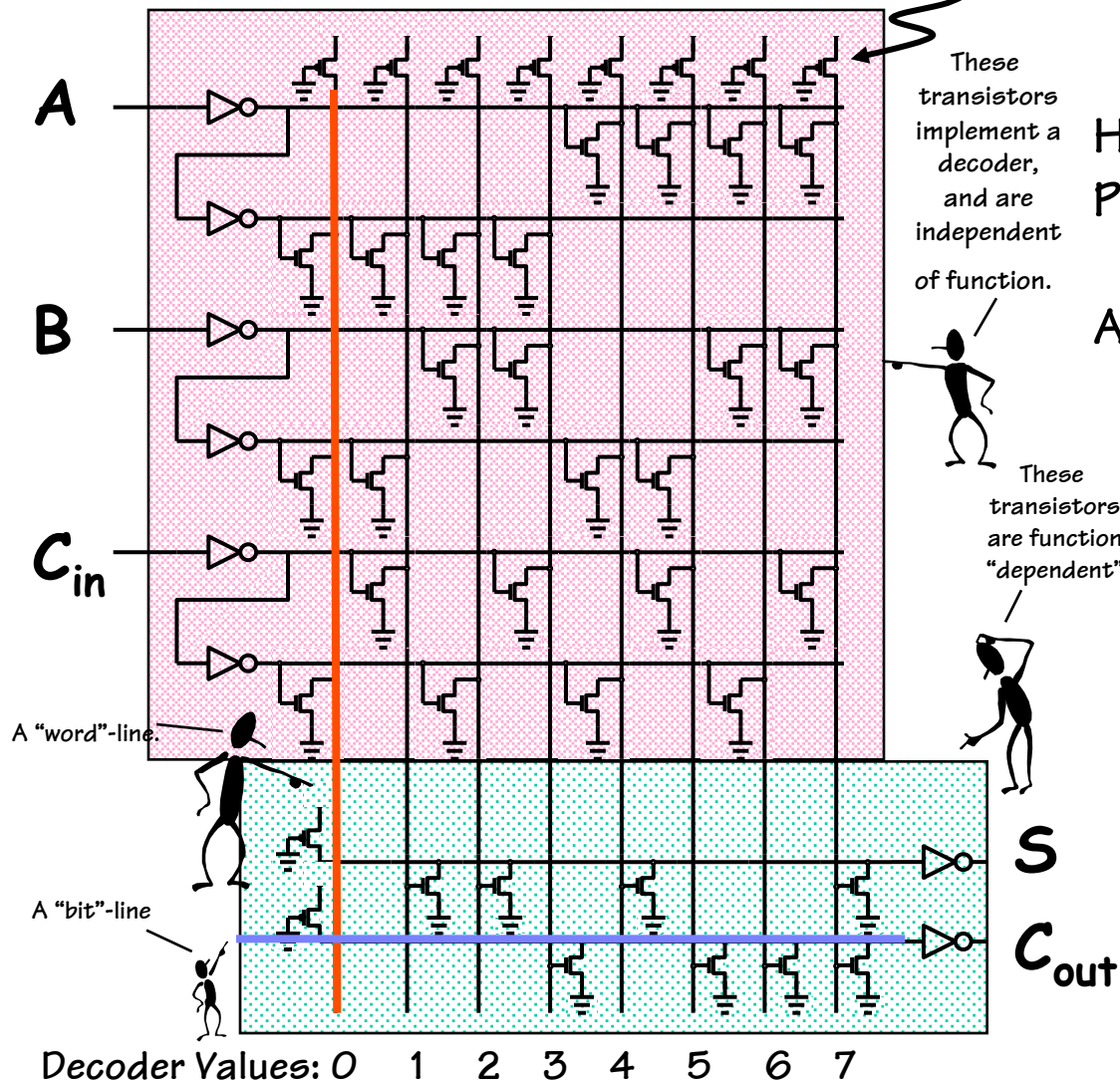


We can build a general purpose "table-lookup" device called a Read-Only Memory (ROM), from which we can implement any truth table and, thus, any combinational device

Made from PREWIRED connections ●, and CONFIGURABLE connections that can be either connected ⊗ or not connected ○

ROM Implementation Details

Tiny PFETs with gates tied to ground = resistor pullup that makes wire "1" unless one of the NFET pull-downs is on.



These transistors implement a decoder, and are independent of function.

Hardwired "AND" logic
Programmable "OR" logic

Advantages:

- Very regular design (can be entirely automated)

Problems:

- Active Pull-ups (Static Power)
- Long metal runs (Large Caps)
- Slow

JARGON: Inputs to a ROM are called ADDRESSES. The decoder's outputs are called WORD LINES, and the outputs lines of the selector are called BIT LINES.



Logic According to ROMs

ROMs *ignore* the structure of combinational functions ...

- Size, layout, and design are independent of function
- Any Truth table can be “programmed” by minor reconfiguration:

- Metal layer (masked ROMs)
- Fuses (Field-programmable PROMs)
- Charge on floating gates (EPROMs)
- ... etc.

Model: LOOK UP value of function in truth table...

Inputs: “ADDRESS” of a T.T. entry

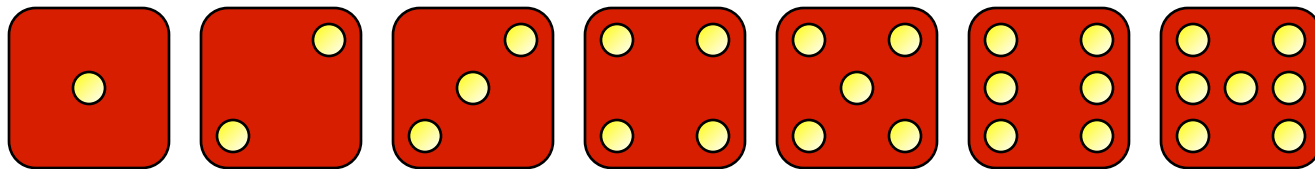
ROM SIZE = # TT entries...

... for an N-input boolean function, size = $2^N \times \text{\#outputs}$

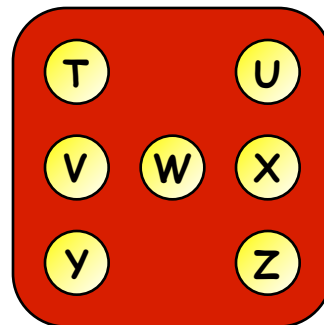
Example: 7-sided Die

What nature can't provide... electronics can
(and with the same number of LEDs!).

We want to construct a die with the following sides:



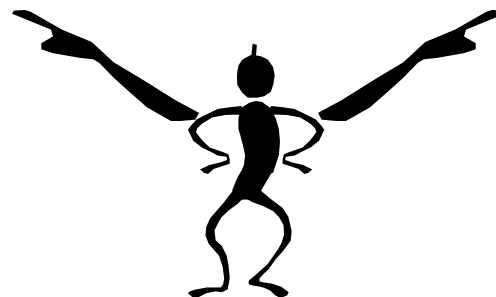
An array of LEDs, labeled as follows, can be used to display the outcome of the die:



ROM-Based Design

Truth Table for a 7-sided Die

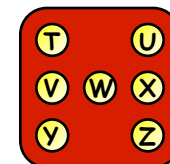
A	B	C	T	U	V	W	X	Y	Z
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	1	0	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
1	0	0	1	1	0	0	0	1	1
1	0	1	1	1	0	1	0	1	1
1	1	0	1	1	1	0	1	1	1
1	1	1	1	1	1	1	1	1	1



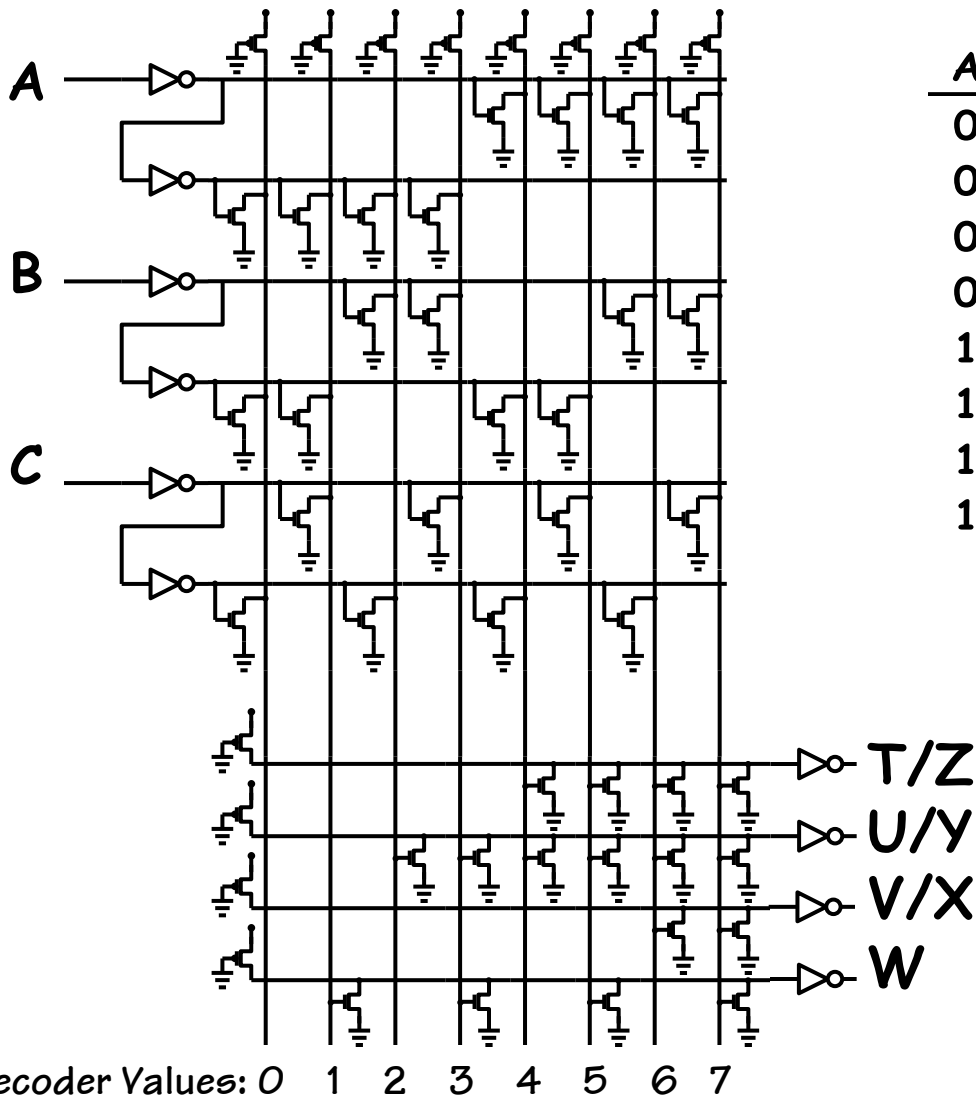
Once we've written out the truth table we've basically finished the design

Possible optimizations:

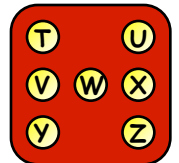
- Eliminate redundant outputs
- Addressing tricks



A Simple ROM implementation



A	B	C	T/Z	U/Y	V/X	W
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	1	0	0
0	1	1	0	1	0	1
1	0	0	1	1	0	0
1	0	1	1	1	0	1
1	1	0	1	1	1	0
1	1	1	1	1	1	1



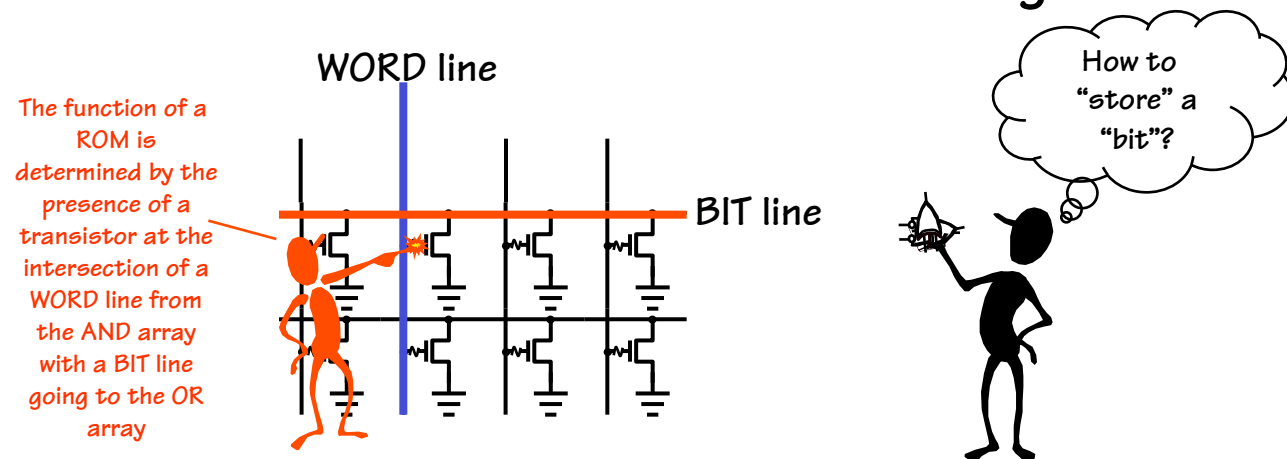
That was Easy!

ROMs are even more flexible than MUXes, because you can design the H/W first, and figure out the logic later!

This is the essence of programmability: “LATE-BINDING” logic specification.

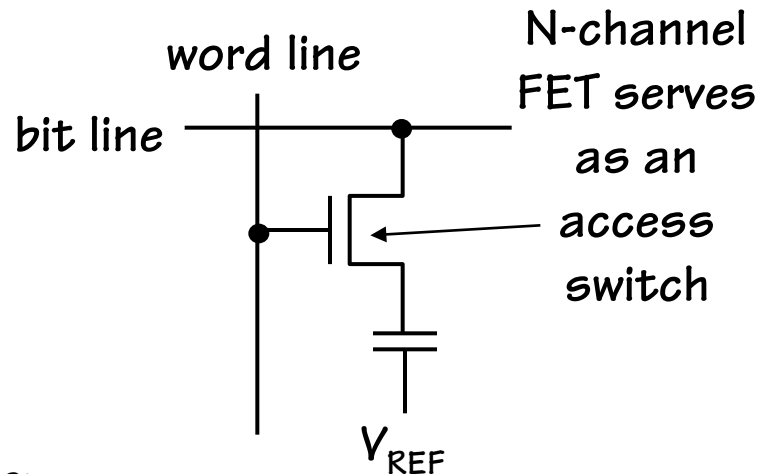
“Programmable” Look-up Tables

Remember, EVERY combinational circuit can be expressed as a lookup table. As a result a ROM is a universal logic device. Unfortunately, the ROMs we’ve built thus far are “HARDWIRED”. That is, the function that they compute is encoded by the pull-down transistors that are built into the OR-plane of the ROM. What we’d really like is a combinational gate that could be reconfigured dynamically. For this we’ll need some form of storage.



Analog Storage: Using Capacitors

We've chosen to encode information using voltages and we know from physics that we can "store" a voltage as "charge" on a capacitor:



To write:

Drive bit line, turn on access fet, force storage cap to new voltage

To read:

precharge bit line, turn on access fet, detect (small) change in bit line voltage

Pros:

- ◆ compact!

Cons:

- ◆ it leaks! \Rightarrow refresh

- ◆ complex interface

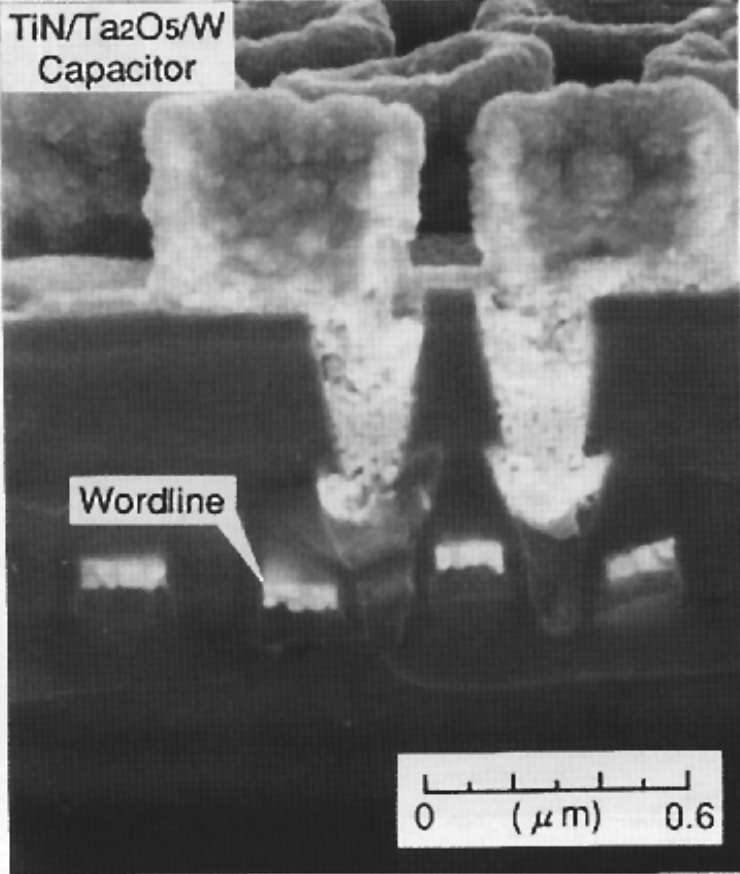
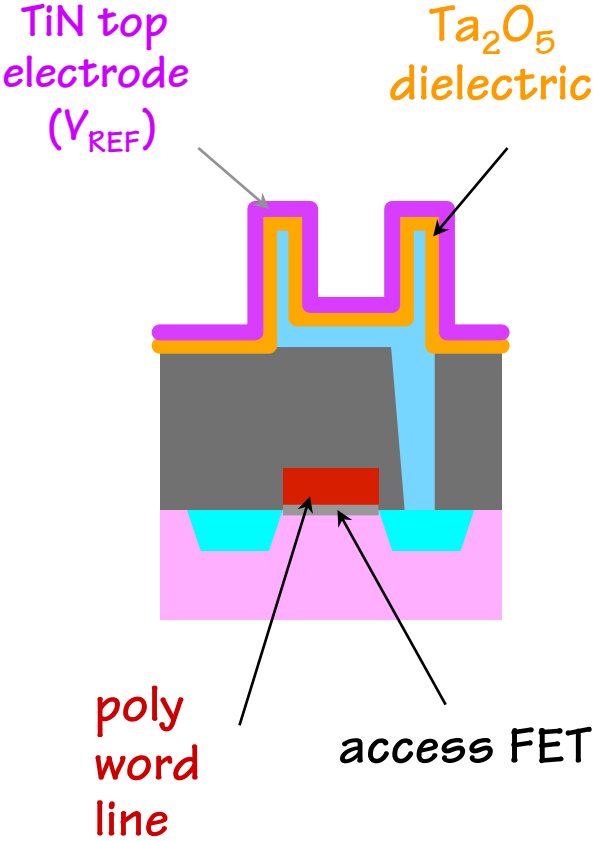
- ◆ reading a bit, destroys it

(you have to rewrite the value after each read)

- ◆ it's NOT a digital circuit

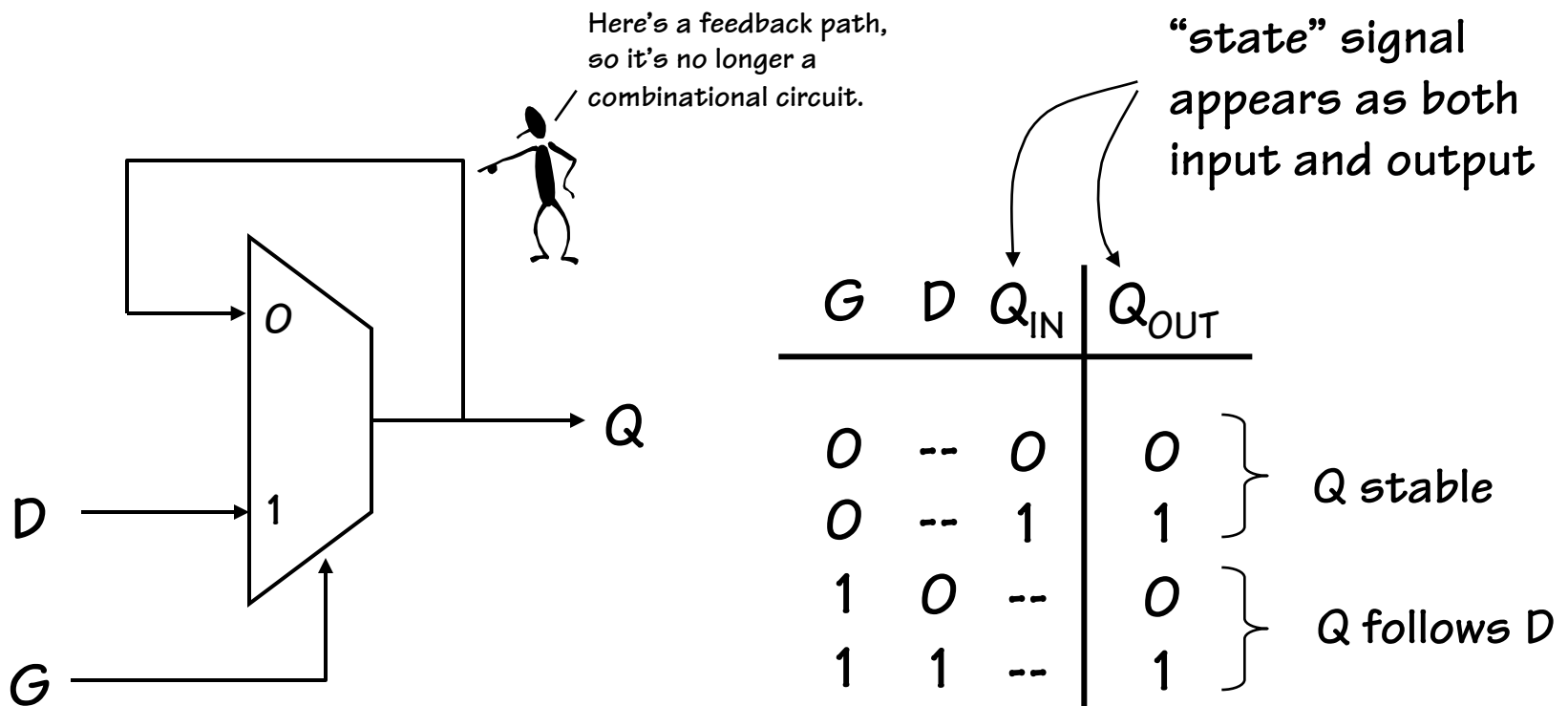
This storage circuit is the basis for commodity DRAMs

Dynamic Memory



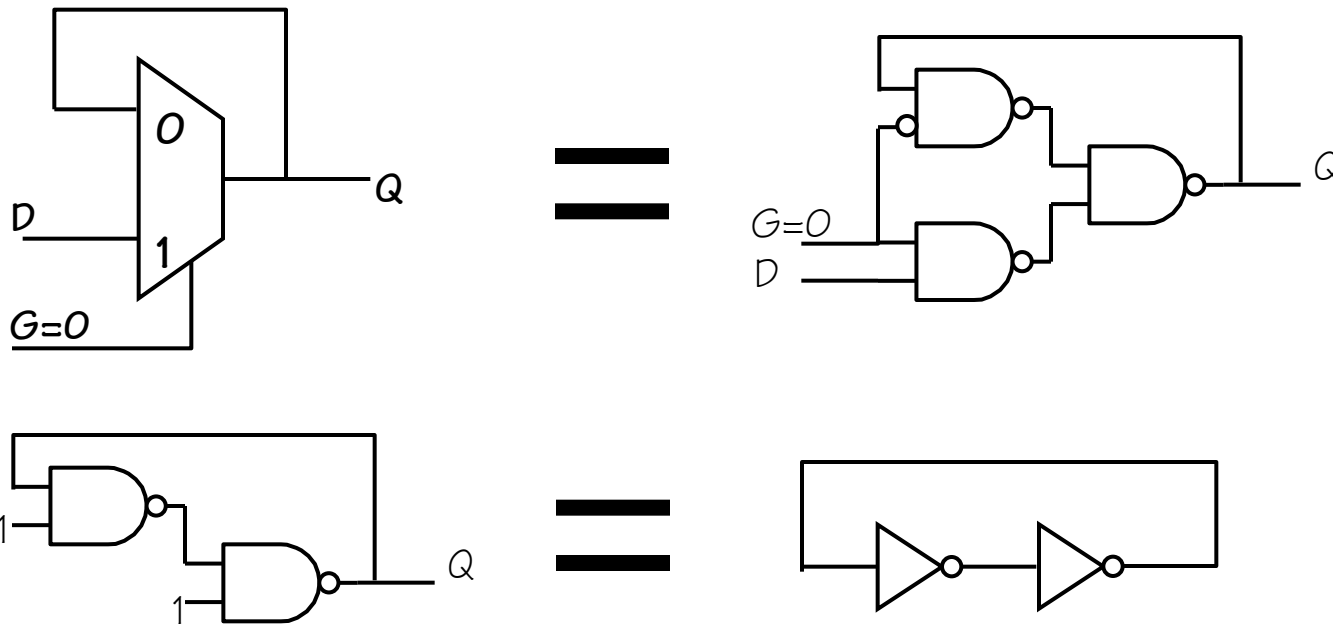
A "Digital" Storage Element

It's also easy to build a settable DIGITAL storage element (called a **latch**) using a MUX and FEEDBACK:



Looking Under the Covers

Let's take a quick look at the equivalent circuit for our MUX when the gate is LOW (the feedback path is active)



This storage circuit is the basis for commodity SRAMs

Advantages:

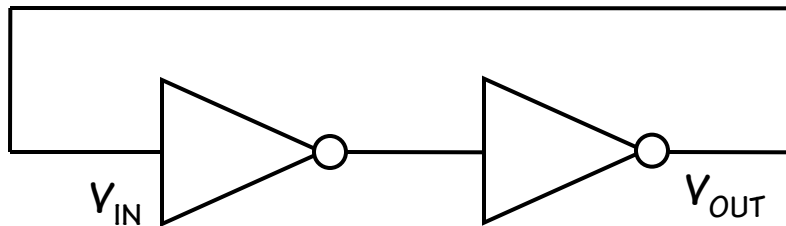
- 1) Maintains remembered state for as long as power is applied.
- 2) State is DIGITAL

Disadvantage:

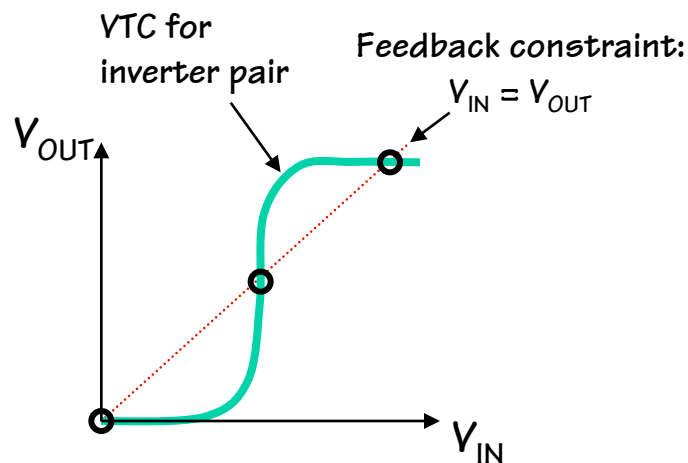
- 1) Requires more transistors

Why Does Feedback = Storage?

BIG IDEA: use **positive feedback** to maintain storage indefinitely. Our logic gates are built to restore marginal signal levels, so noise shouldn't be a problem!



Result: a **bistable storage element**



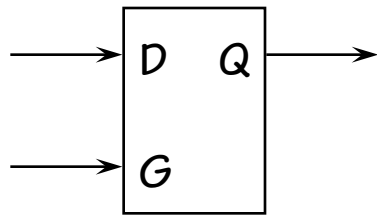
Not affected
by noise

Three solutions:

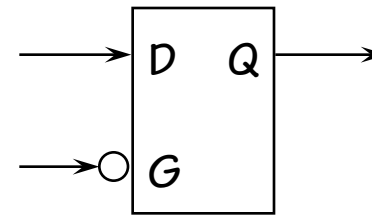
- ◆ two end-points are **stable**
- ◆ middle point is **unstable**

We'll get back to this!

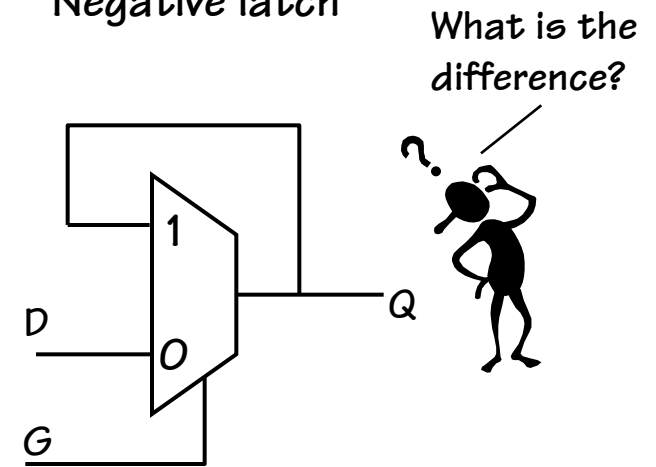
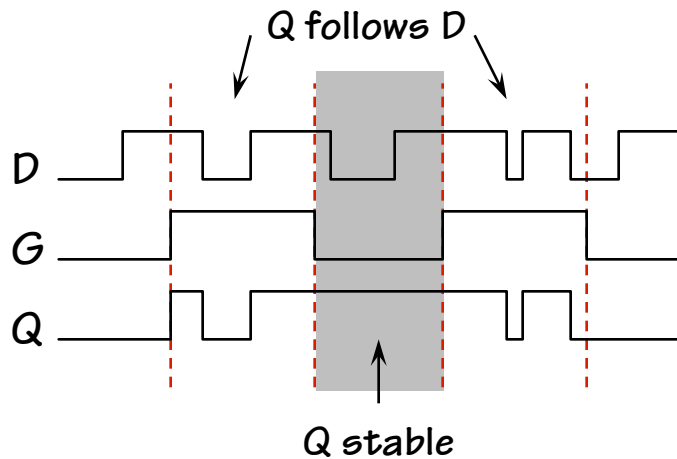
Static D Latch



Positive latch



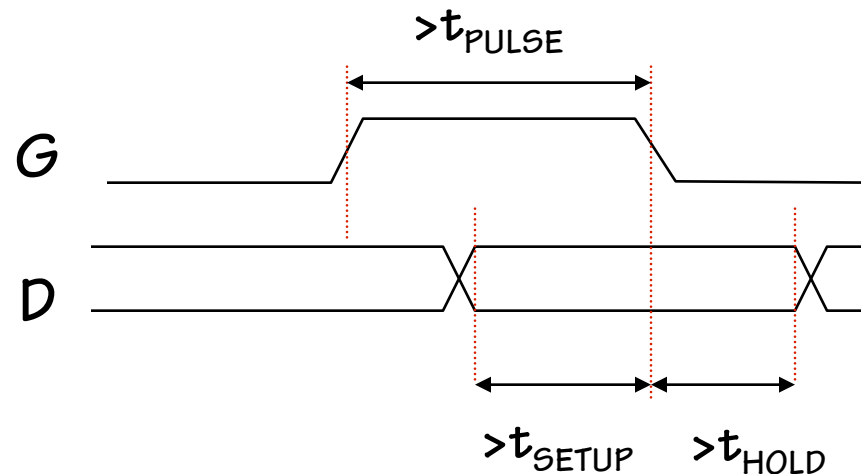
Negative latch



“static” means latch will hold data (i.e., value of Q) while G is inactive, however long that may be.

A DYNAMIC Discipline

Design of sequential circuits MUST guarantee that inputs to sequential devices are valid and stable during periods when they may influence state changes. **This is assured with additional timing specifications.**



t_{PULSE} : minimum pulse width

guarantee G is active for long enough for latch to capture data

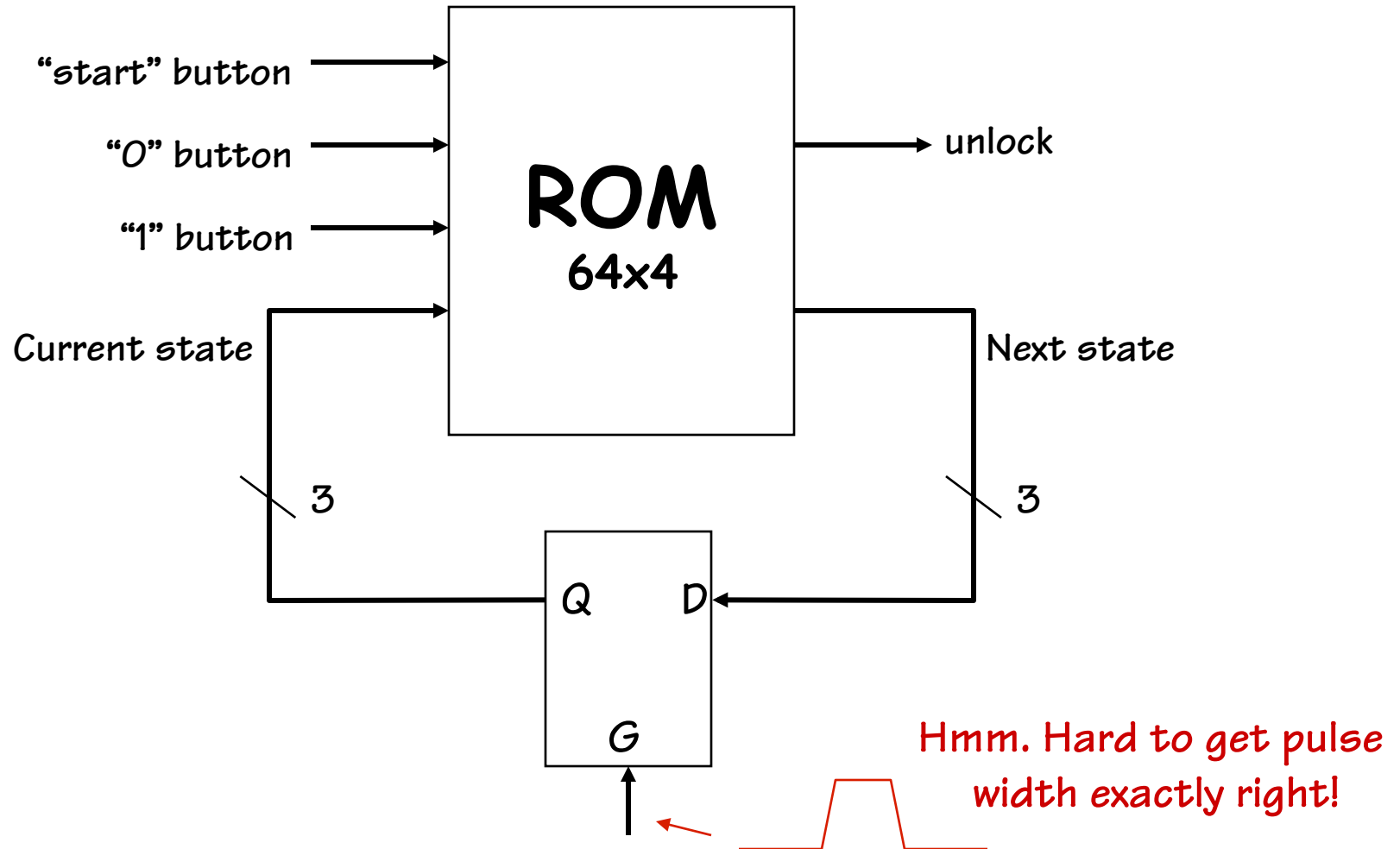
t_{SETUP} : setup time

guarantee that D value has propagated through feedback path before latch closes

t_{HOLD} : hold time

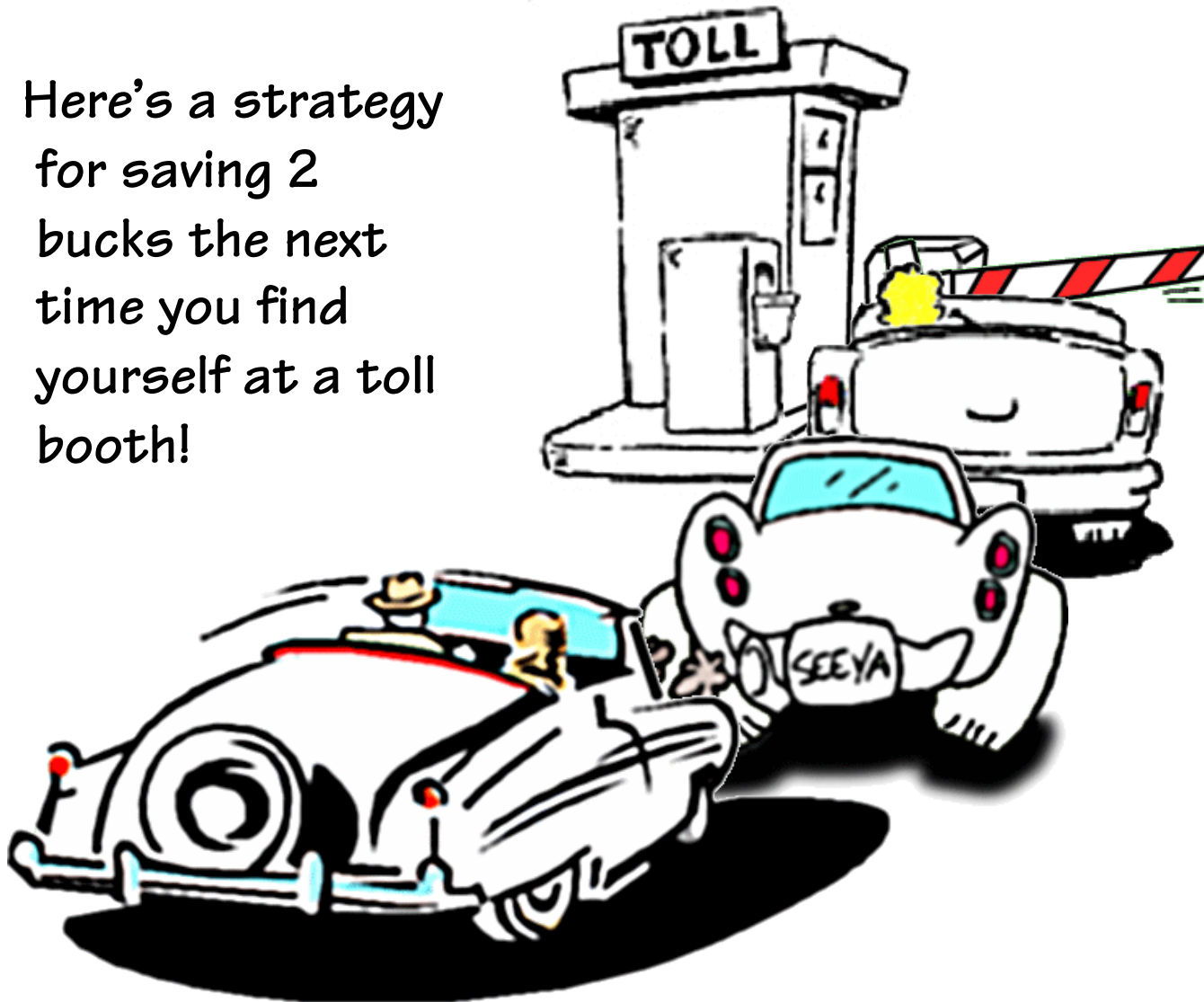
guarantee latch is closed and Q is stable before allowing D to change

Does this work?



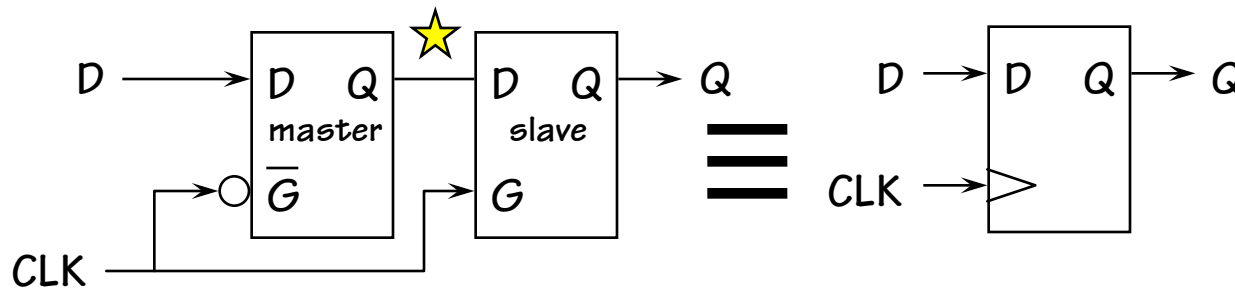
Flakey Control Systems

Here's a strategy
for saving 2
bucks the next
time you find
yourself at a toll
booth!



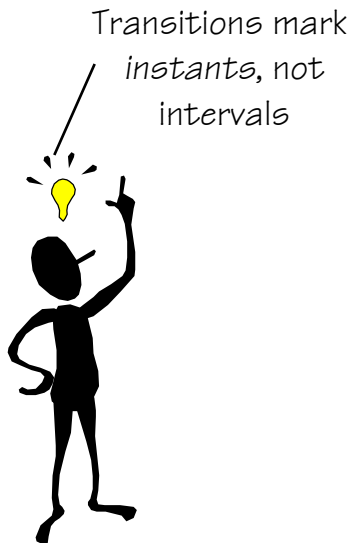
Edge-triggered Flip Flop

logical “escapement”

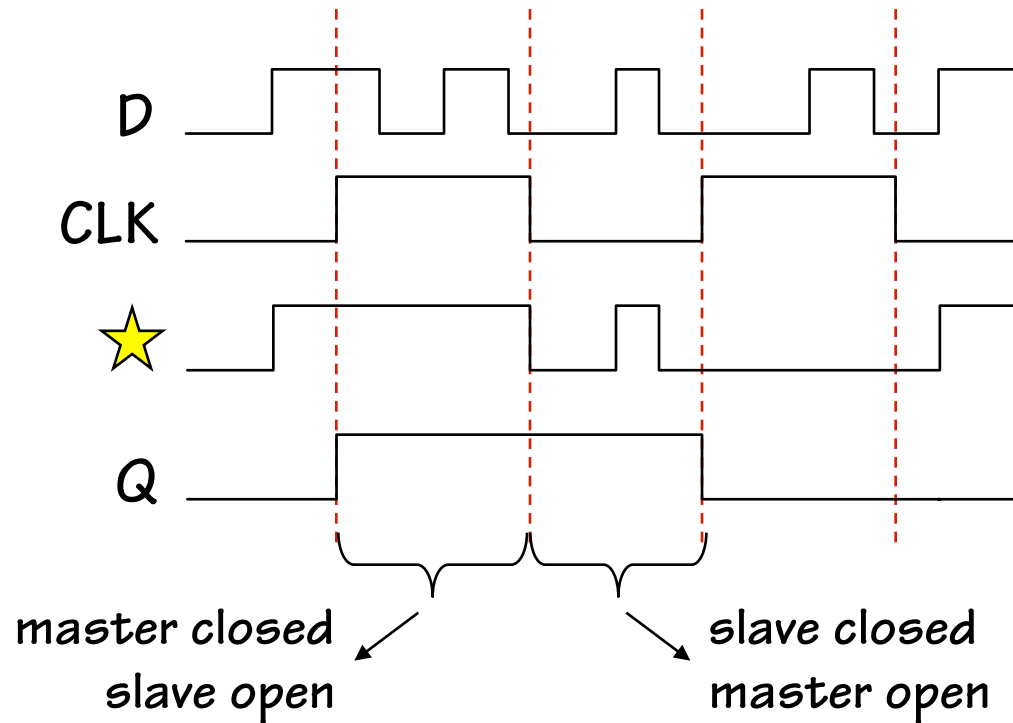
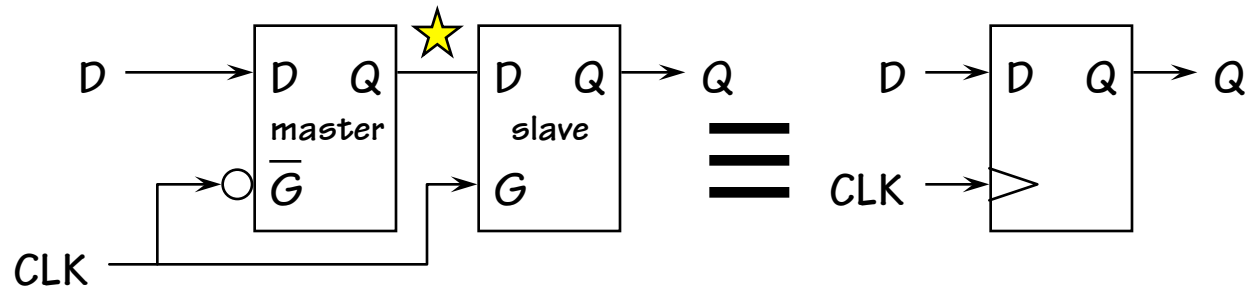


Observations:

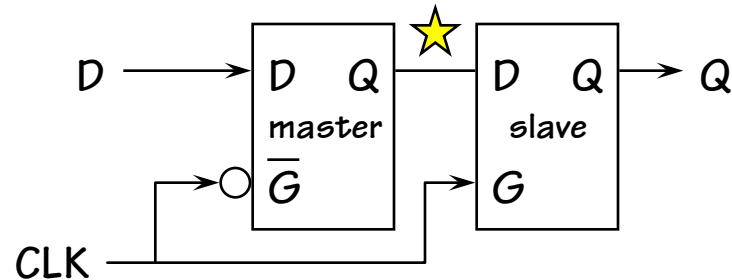
- ♦ only one latch “transparent” at any time:
 - ♦ master closed when slave is open (CLK is high)
 - ♦ slave closed when master is open (CLK is low)→ no combinational path through flip flop
- ♦ Q only changes shortly after 0 → 1 transition of CLK, so flip flop appears to be “triggered” by rising edge of CLK



Flip Flop Waveforms

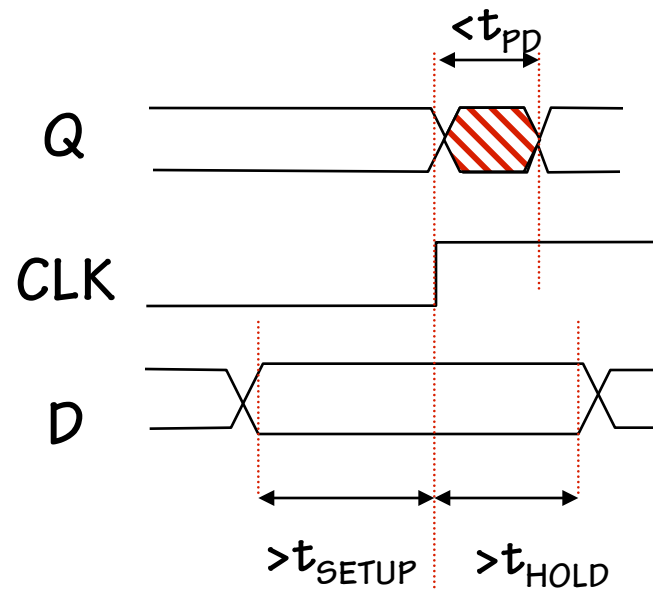
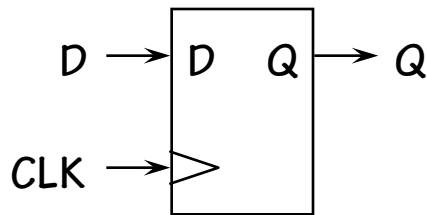


Two Issues



- Must allow time for the input's value to propagate to the Master's output while CLK is LOW.
 - This is called "SET-UP" time
- Must keep the input stable, just after CLK transitions to HIGH. This is insurance in case the SLAVE's gate opens just before the MASTER's gate closes.
 - This is called "HOLD-TIME"
 - Can be zero (or even negative!)
- Assuring "set-up" and "hold" times is what limits a computer's performance

Flip-Flop Timing Specs



t_{PD} : maximum propagation delay, CLK \rightarrow Q

t_{SETUP} : setup time

guarantee that D has propagated through feedback path before master closes

t_{HOLD} : hold time

guarantee master is closed and data is stable before allowing D to change

Summary

- Regular Arrays can be used to implement arbitrary logic functions
 - ROMs decode every input combination (fixed-AND array) and compute the output for it (customized-OR array)
 - PLAs decode an minimal set of input combinations (both AND and OR arrays customized)
- Memories
 - ROMs are HARDWIRED memories
 - RAMs include storage elements at each WORD-line and BIT-line intersection
 - dynamic memory: compact, only reliable short-term
 - static memory: controlled use of positive feedback
- Level-sensitive D-latches for static storage
- Dynamic discipline (setup and hold times)