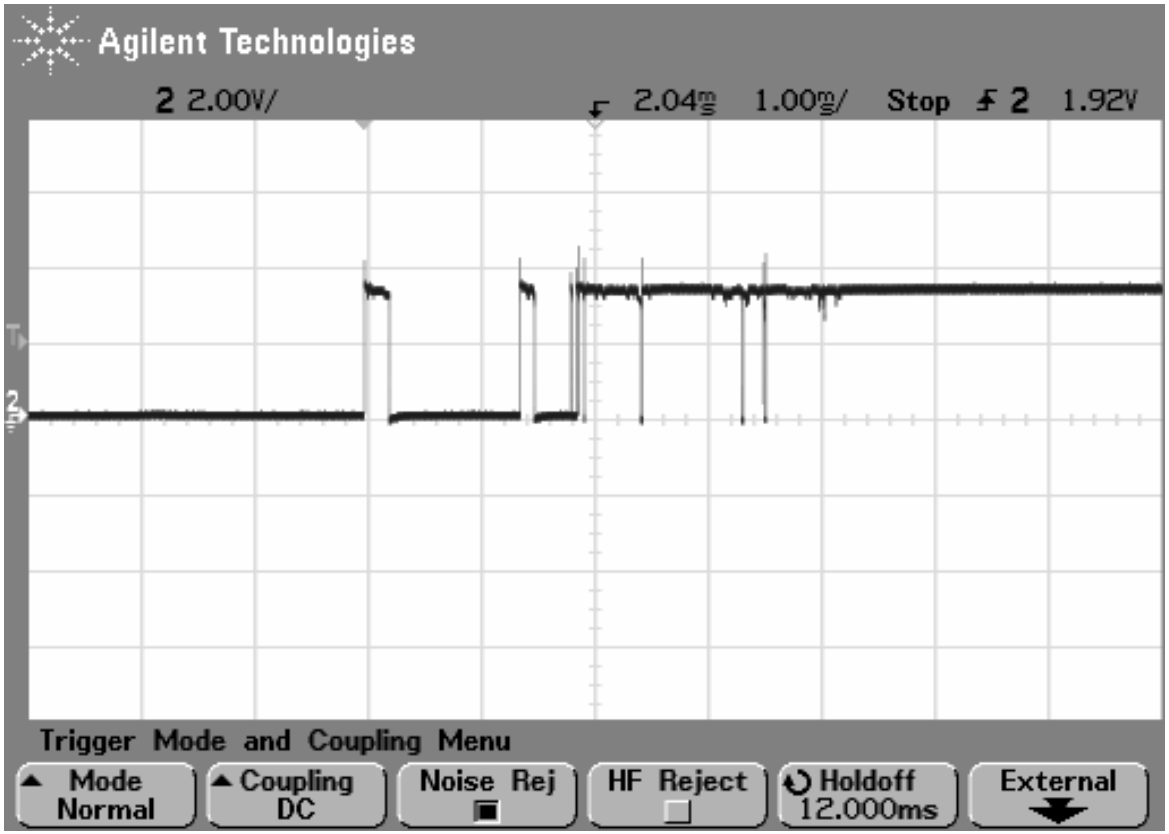
Debouncing a Switch

A Design Example

Background and Motivation

When you throw a switch (button or two-pole switch)…

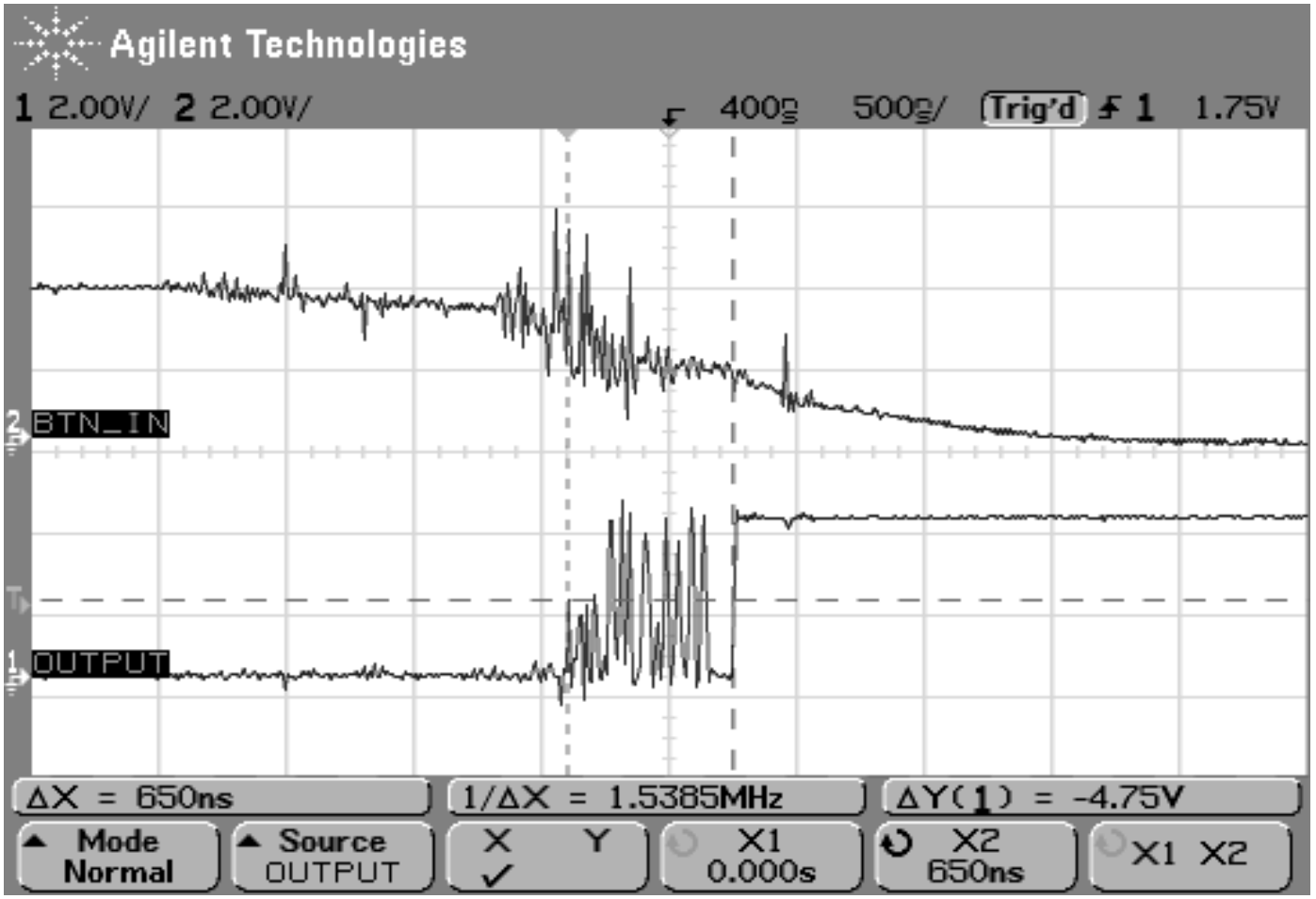
- It often bounces…



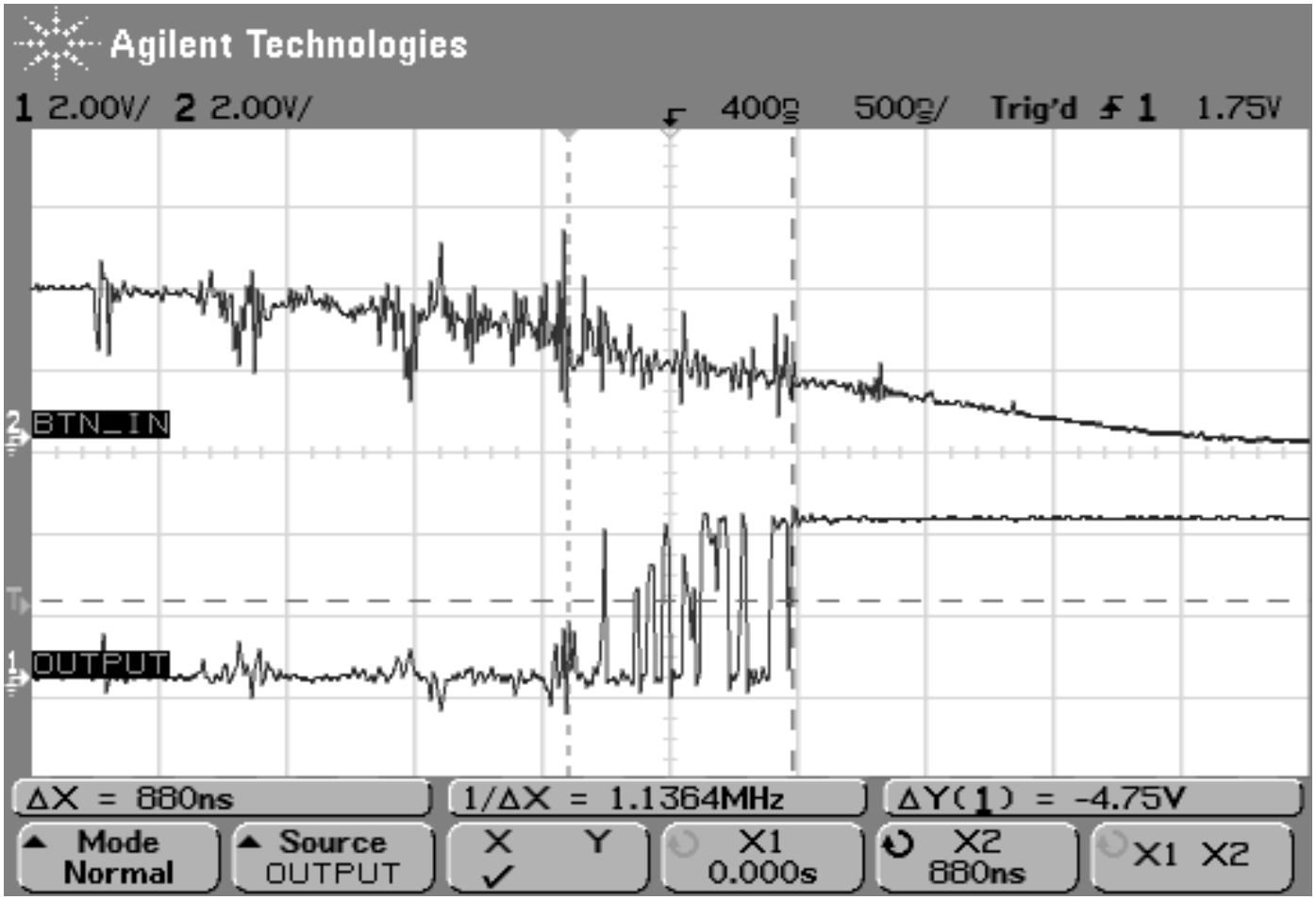
Another switch…

switch

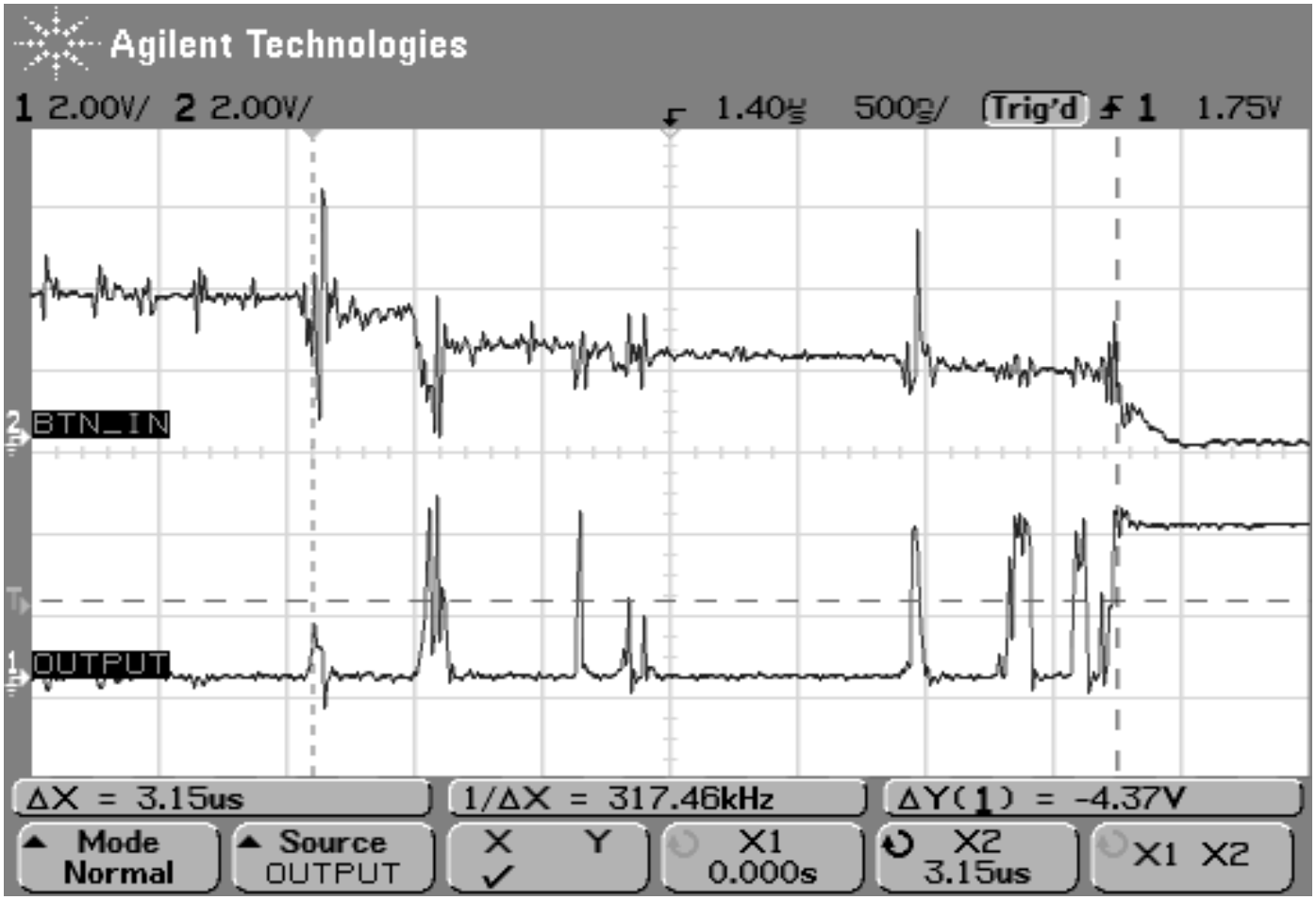
after inversion



Yet Another…



Still Yet Another…

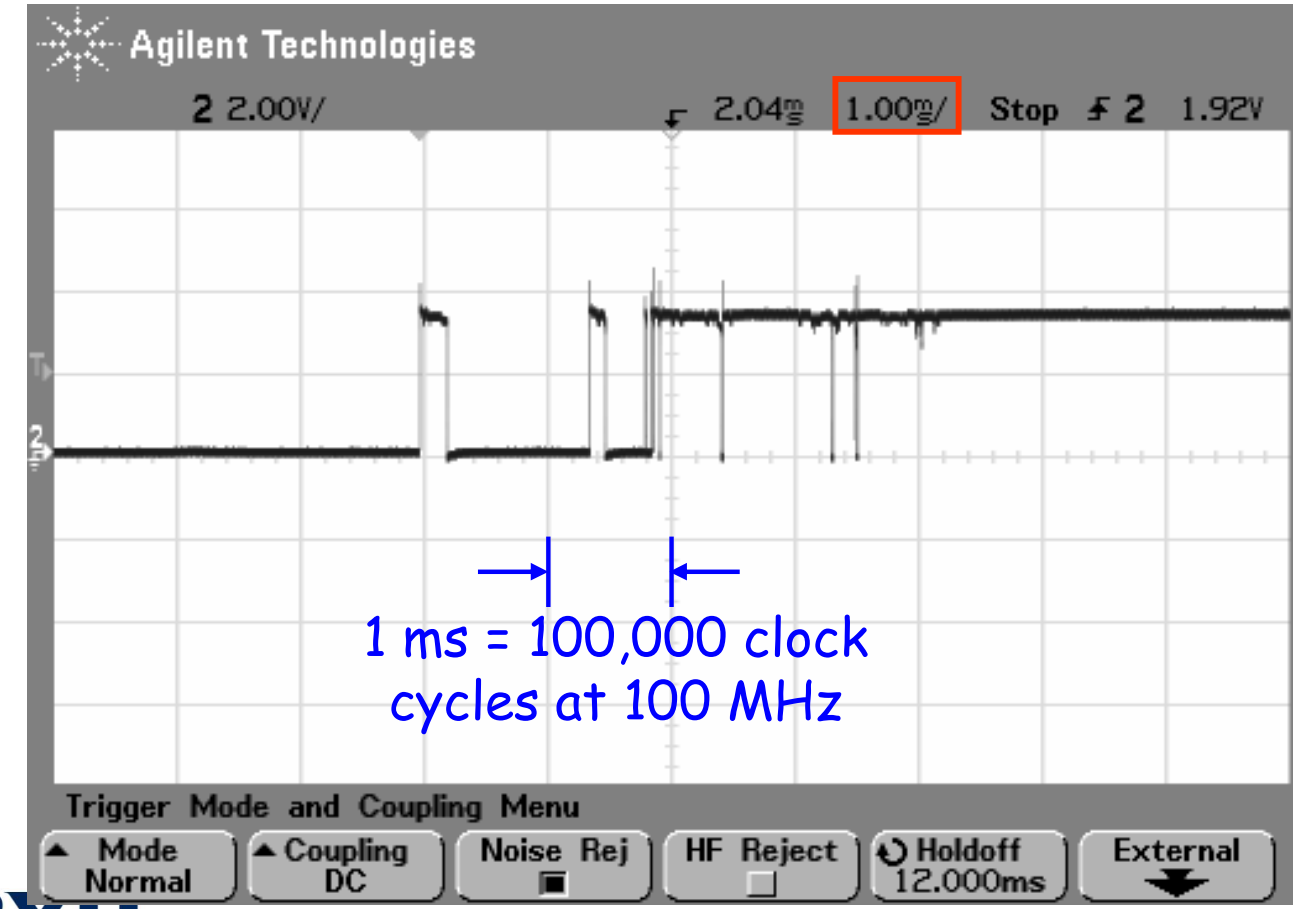


Causes

- Switches and buttons are mechanical
  - Spring loaded
    - Contacts literally bounce
  - Not an instant, once-only, on↔off change

Source of Errors

- Consider a 100 MHz system clock, which has a 10 ns period
- Each ms would be 100,000 system clock cycles
- Downstream circuitry will see every bounce as an input change!



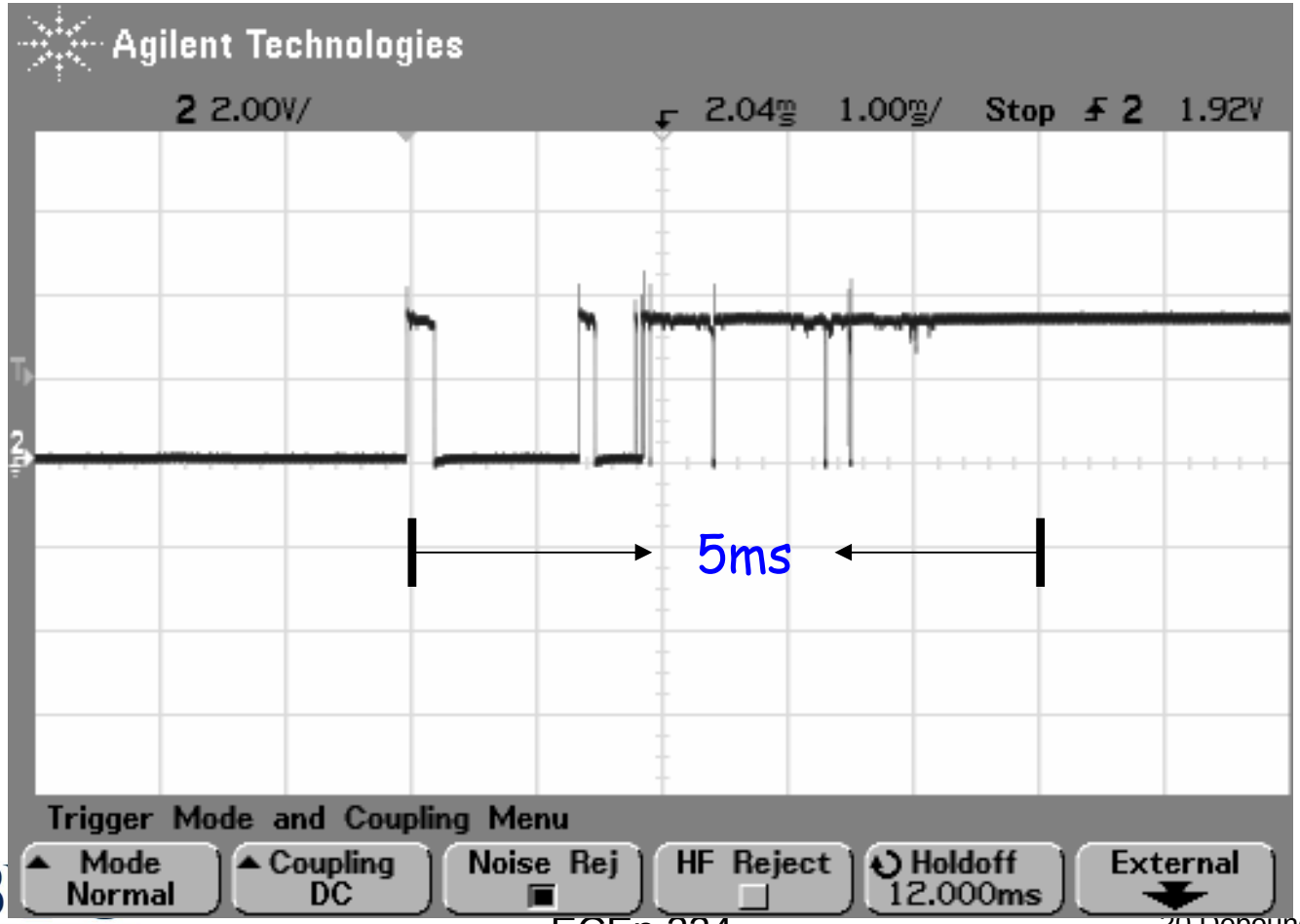
FSM-Based Solution

Solutions

- Single-output switch
  - Since all you see is bouncing value, timing-based solution can be employed
- There are other solutions but they require a different kind of switch

Timing-Based Solution

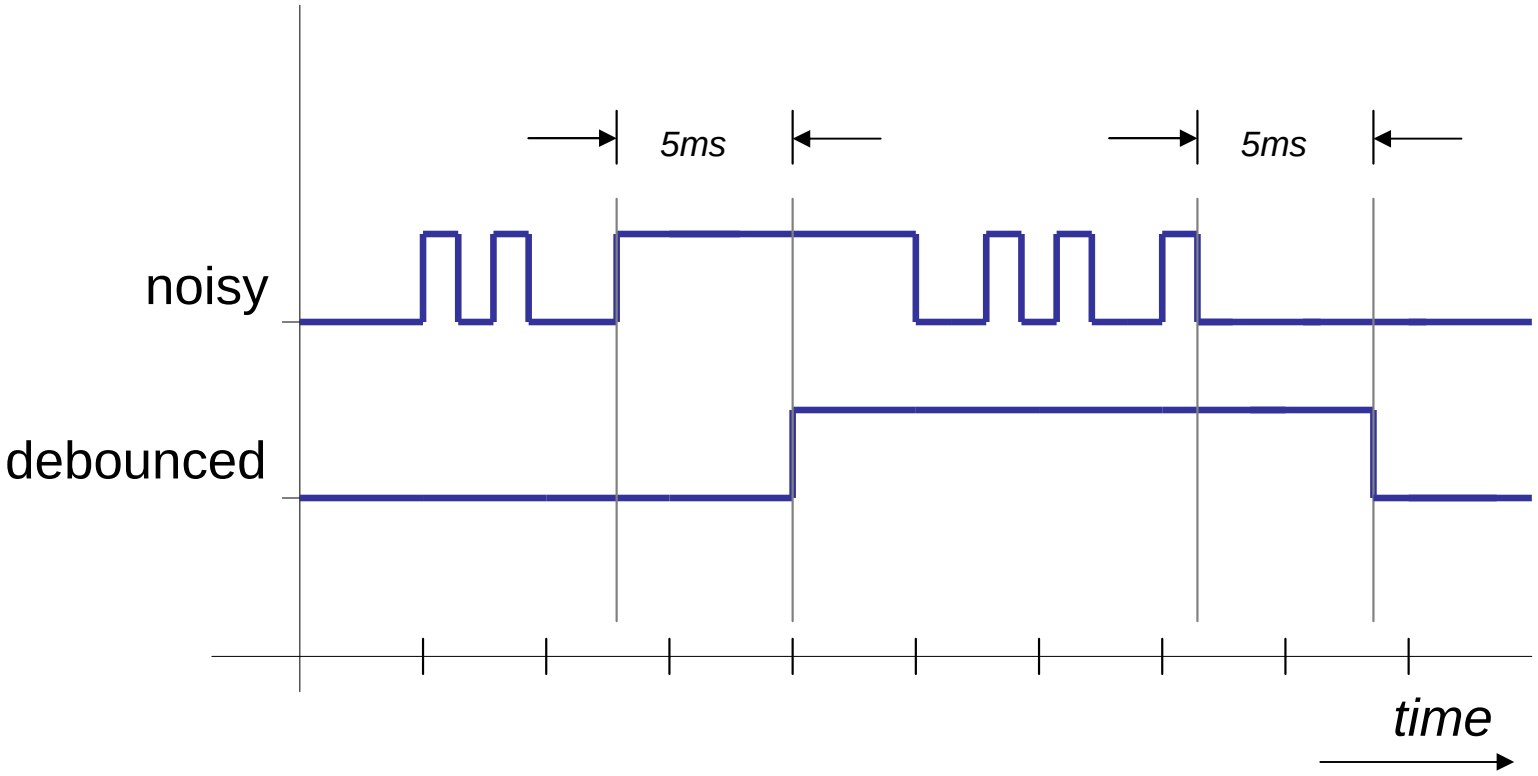
- Only declare an input change after signal has been stable for at least 5ms



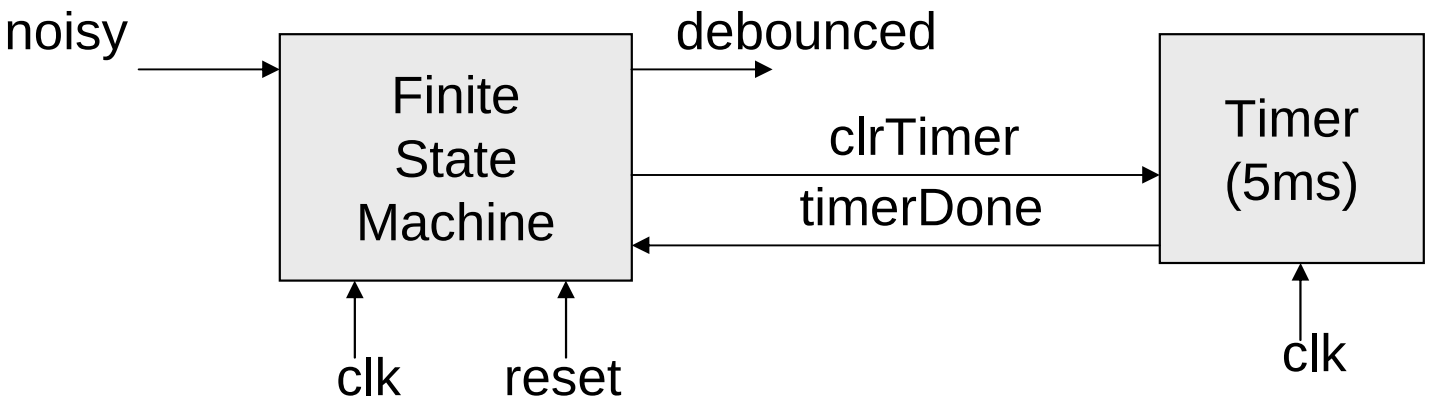
FSM Solution

- Simple enough that an FSM might not be required
  - Easy to concoct a sequential circuit to do this with a counter and a single FF
- Let's do it with an FSM
  - If solution requires only a counter and a single FF, we will find that solution

Draw a Simplified Timing Diagram



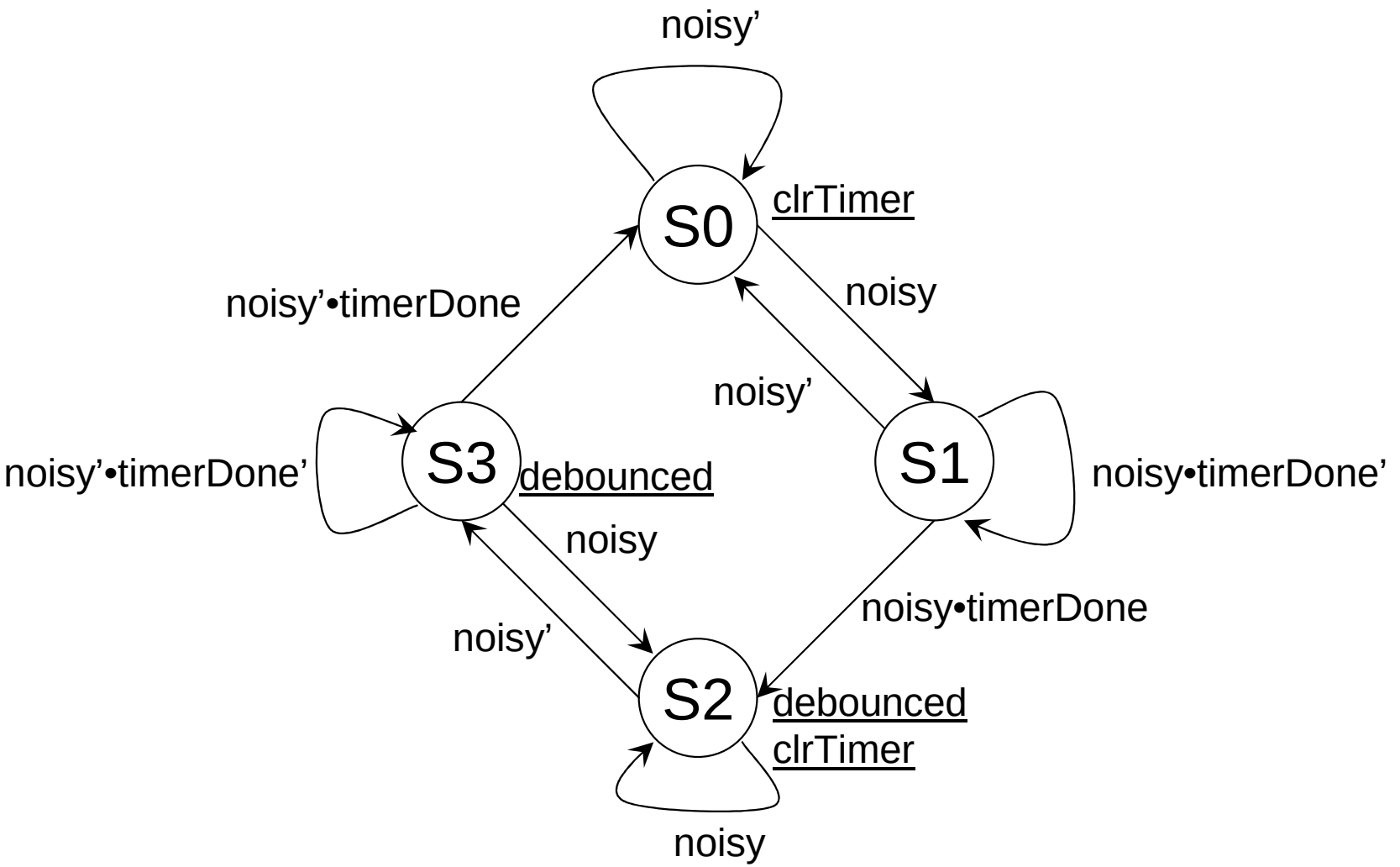
Draw a System Block Diagram



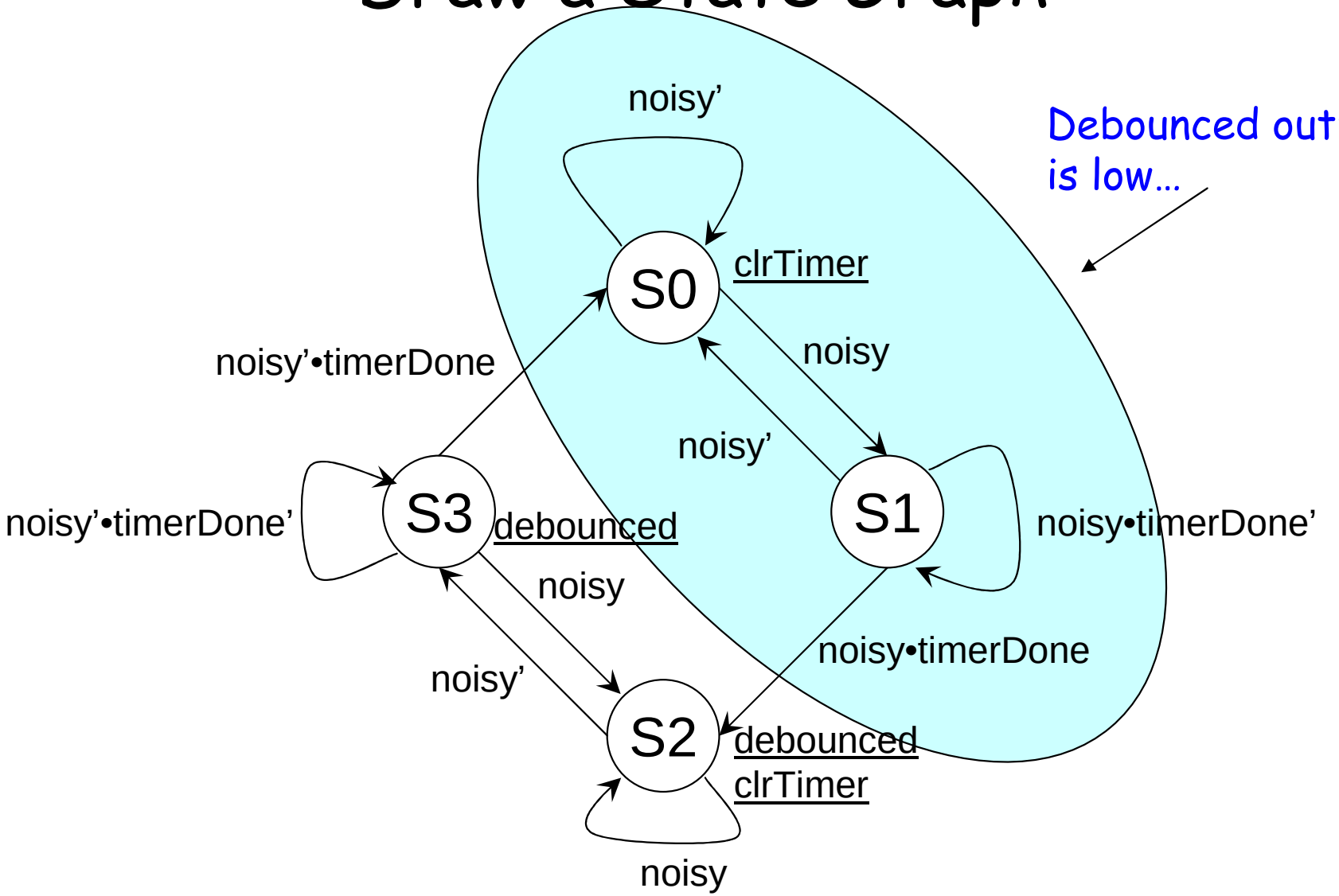
Very reminiscent of our car wash controller…

The Design of the FSM

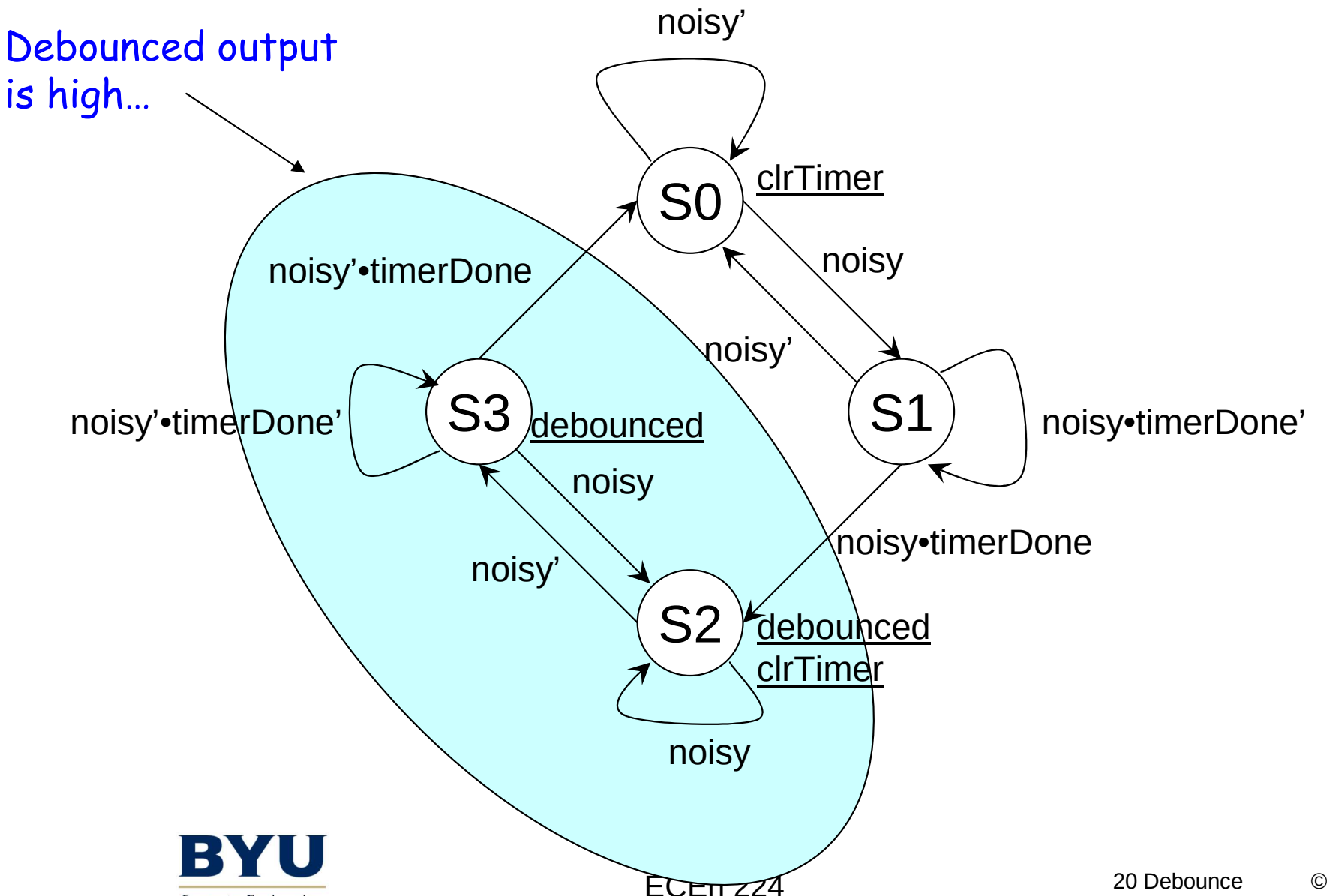
Draw a State Graph



Draw a State Graph

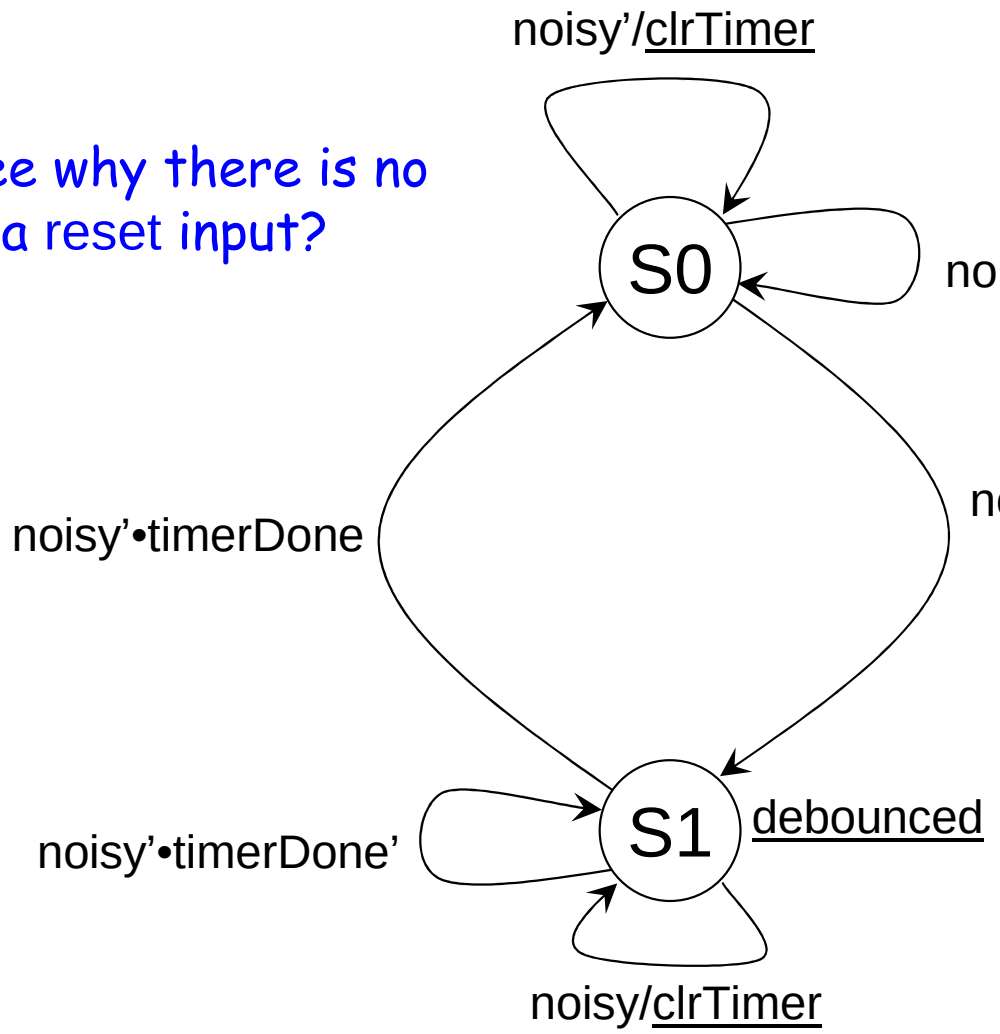


Draw a State Graph



An Improved State Graph

Do you see why there is no need for a reset input?



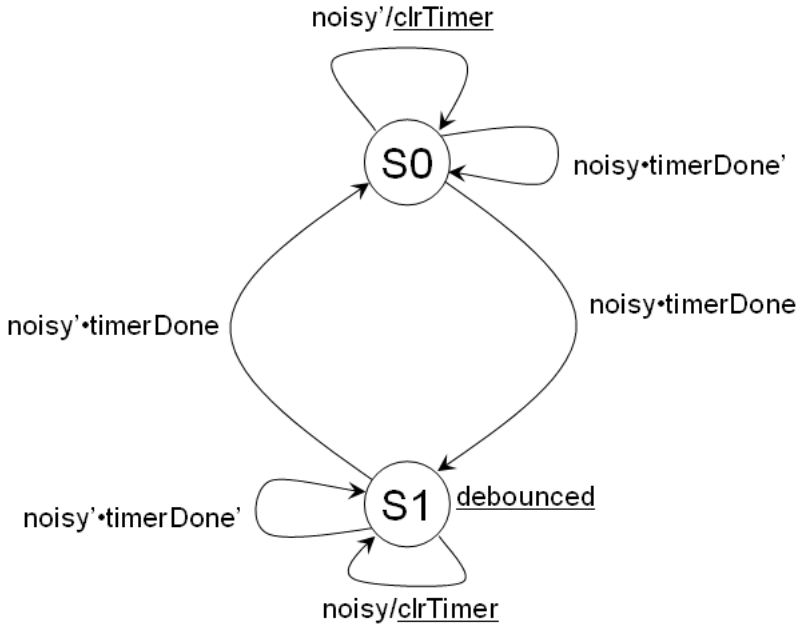
Looks like the FSM can be implemented with just a single FF

As mentioned, Mealy machines often require fewer states…

Reduce FSM to Logic

$S_0 = CS'$ $S_1 = CS$ noisy = N timerDone = T

CS \ N T	00	01	11	10
0			1	
1	1		1	1



NS = noisy•timerDone + CS•timerDone'

clrTimer = noisy'•CS' + noisy•CS

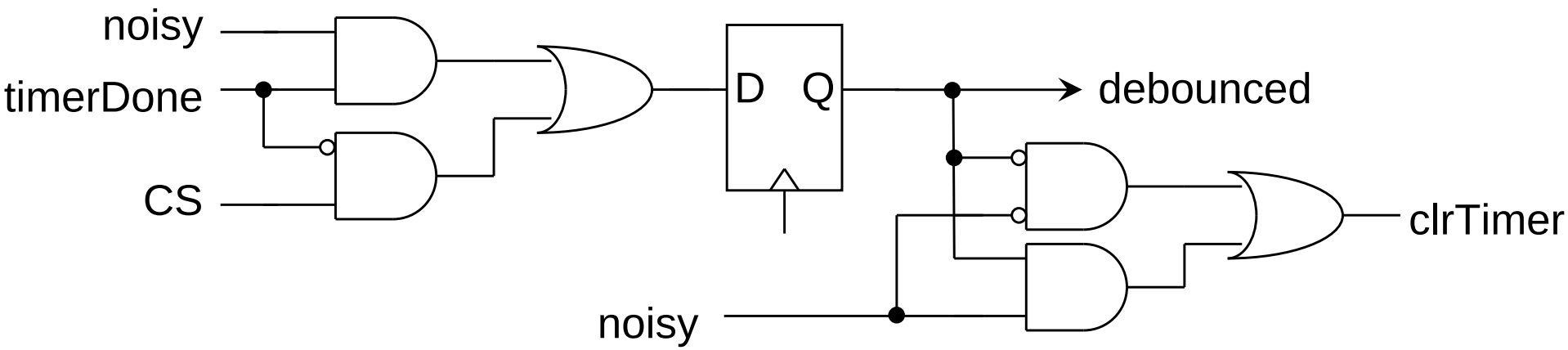
debounced = CS

Reduce FSM to Logic

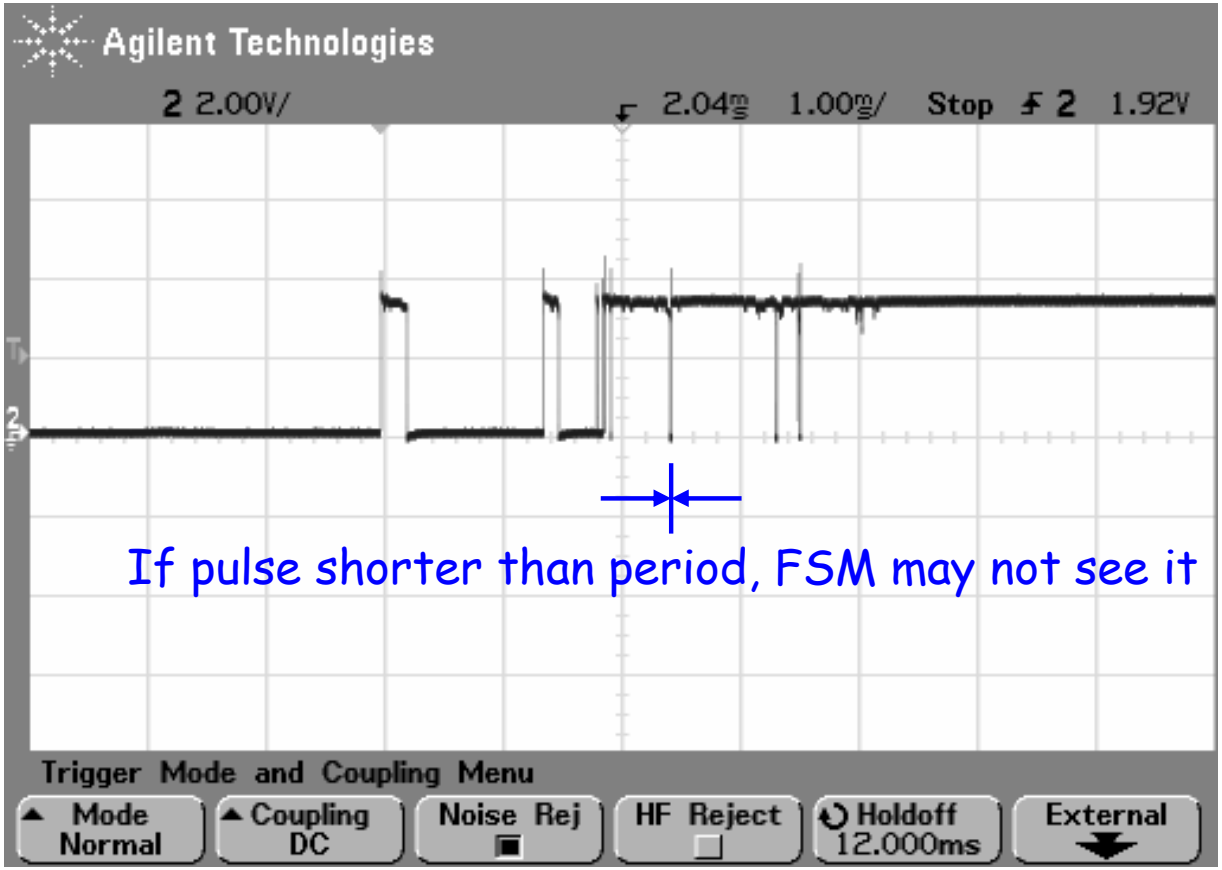
NS = noisy•timerDone + CS•timerDone'

clrTimer = noisy'•CS' + noisy•CS

debounced = CS

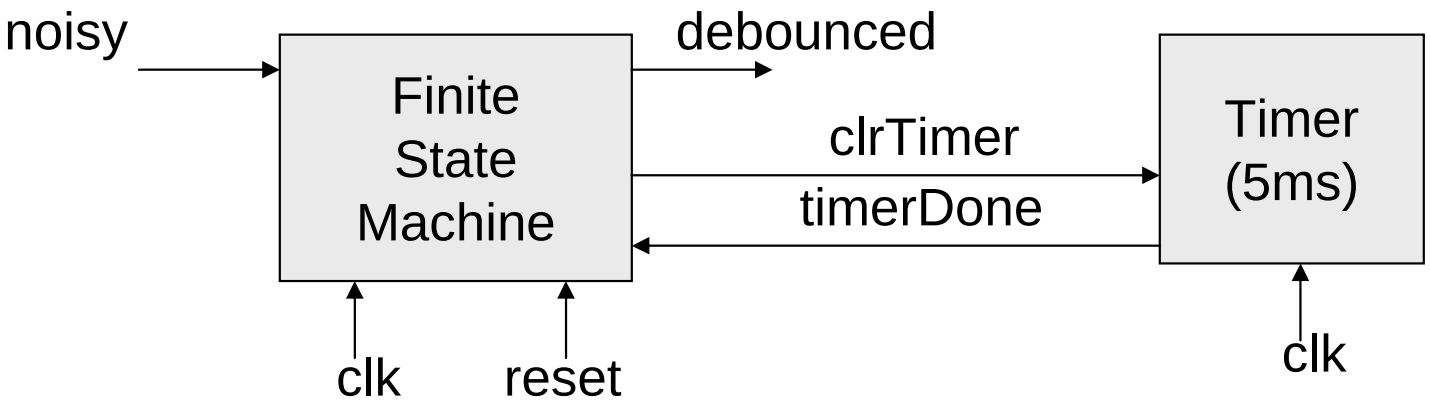


Input "noisy" is Asynchronous



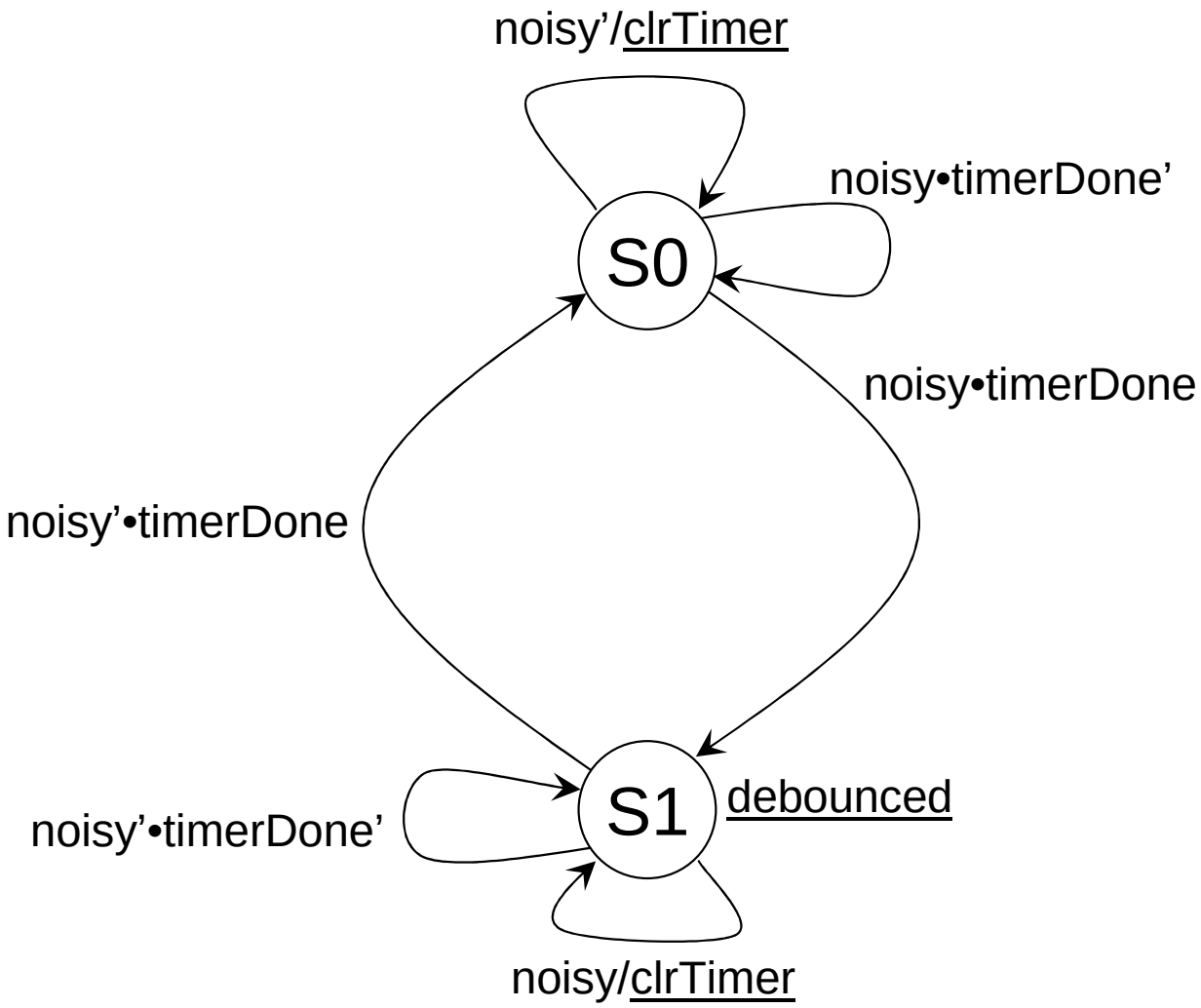
- Very small pulses may be missed by FSM
  - This is not a real problem so we will live with it

More on Asynchronous "noisy" Input



- This is the classic asynchronous input problem:
  - FSM may see input change and change state
  - Timer may *not* see input change and not clear timer
- Or vice versa
- Will this cause incorrect operation?

Asynchronous Input Problem



Look at the transitions. Will previous slide's problem cause a malfunction?

If you determine that a problem may result, that is easiest way to solve the problem?

More on Asynchronous "noisy" Input

- What about metastability?
  - Most buttons aren't pushed very often
  - Chance of metastability is very low

- We could eliminate all our asynchronous problems by adding flip flops in series
  - Avoid detailed analysis
  - Play it safe and avoid possibility of mistakes

Design of the Timer

Timer Calculations

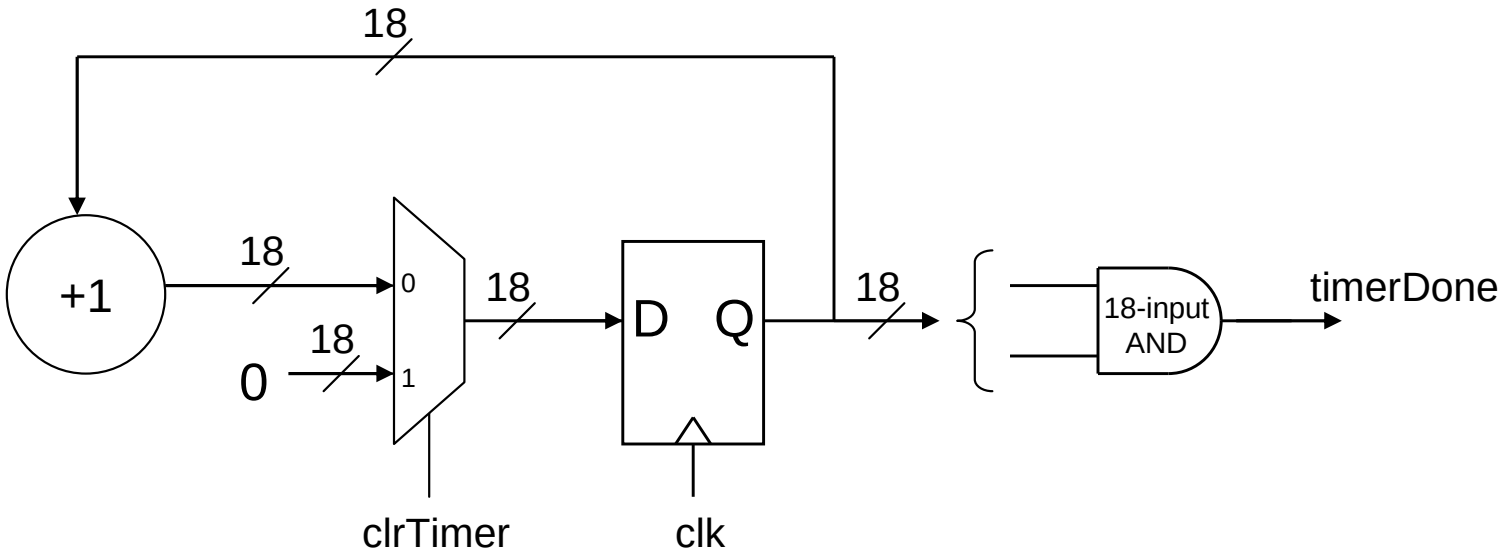
- Assume system runs at 50 MHz (20 ns period)
- 5ms/20ns = 250,000 system clock cycles
- We could design a MOD-250,000 counter
- A simple 18-bit counter will work
  - 2^{18} is a bit longer than 250,000 (262,144)
  - It is close enough to 5 ms for our purposes

Design the Timer

- 19 input state machine
  - 18 CS bits + 1 clrTimer bit
  - Very, very large truth table

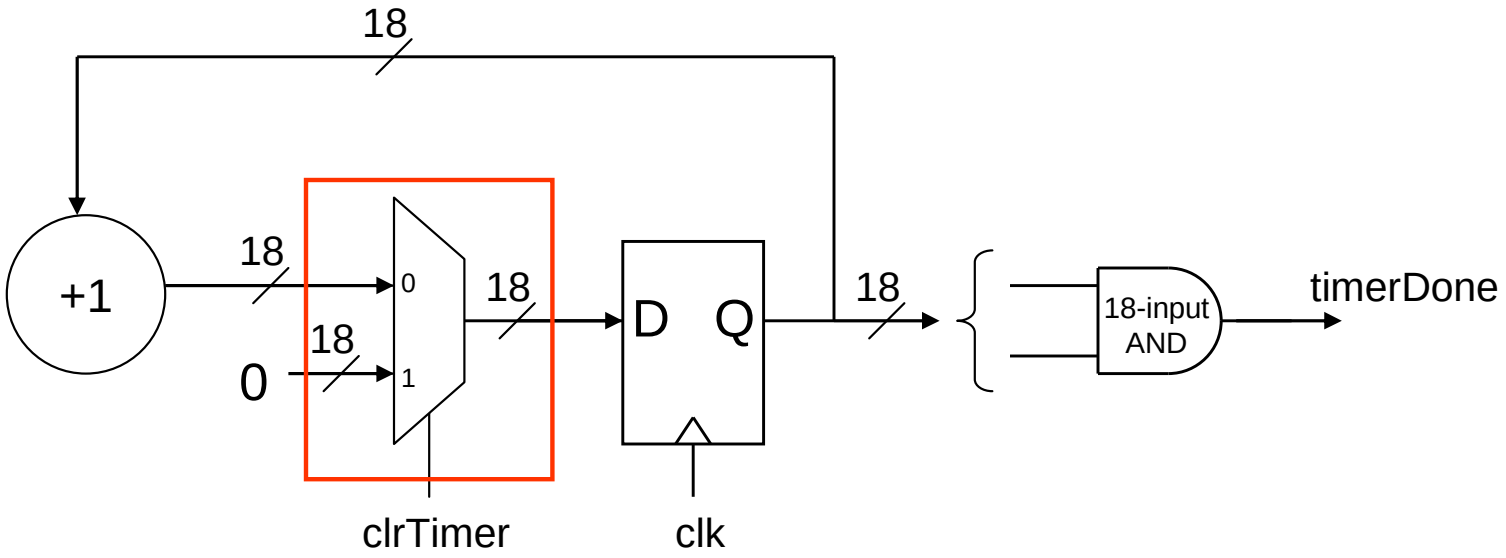
- A better approach:
  - Register that selects between CS+1 and 0
  - This is the technique of Chapter 12 (registers)

Timer Structure



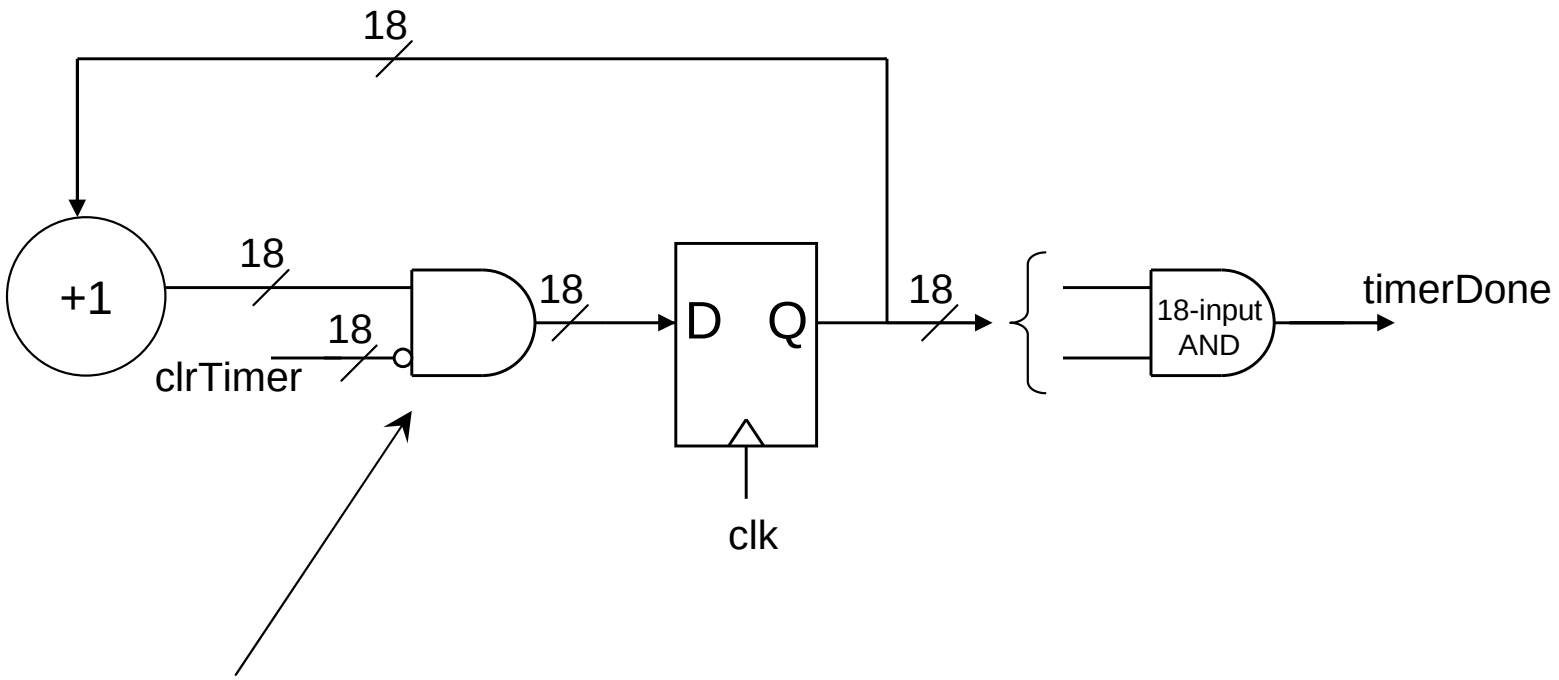
What can we do to simplify this circuit?

Timer Structure



What can we do to simplify this circuit?

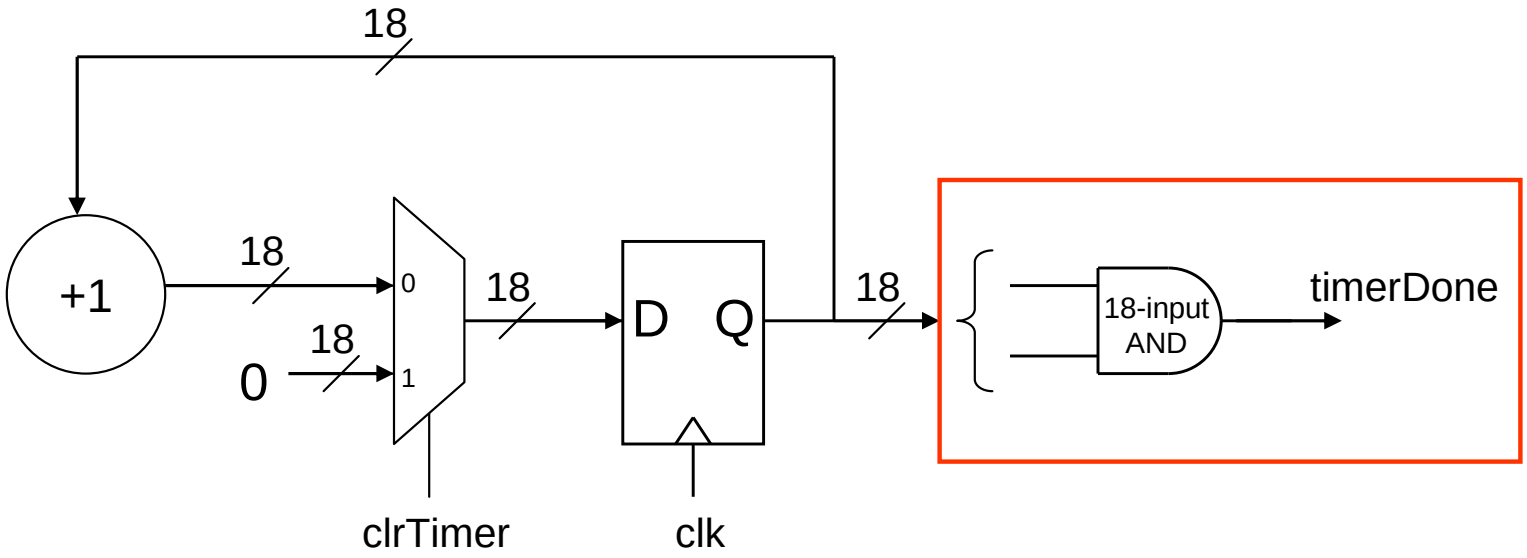
Improved Timer Structure



This is a simpler way to conditionally generate zeroes.

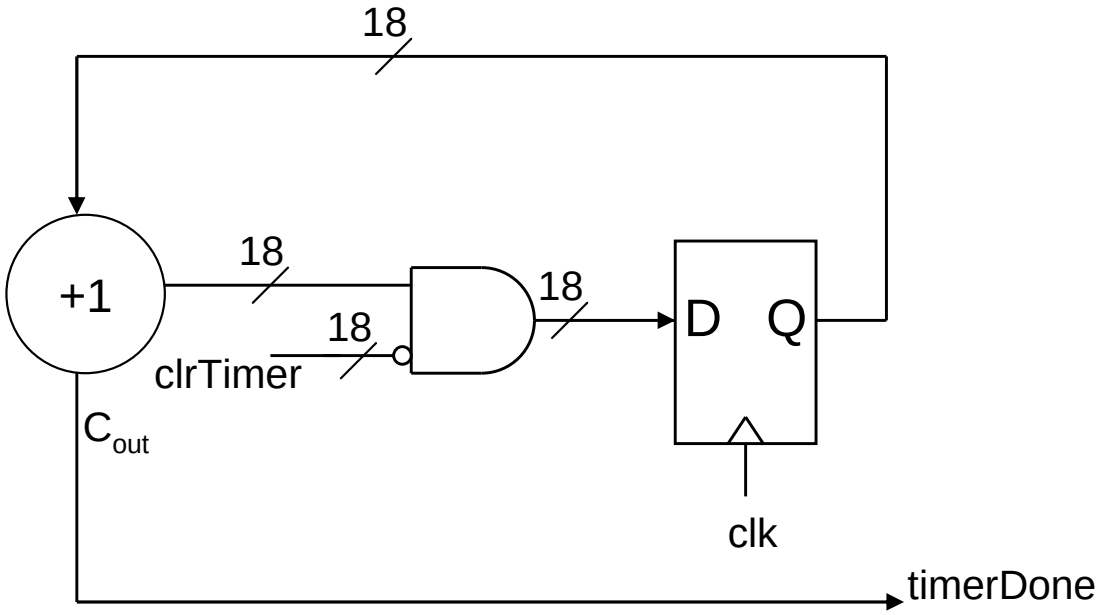
A synthesis tool likely would have generated this from Verilog or VHDL code containing a MUX

Timer Structure



What can we do to simplify this circuit?

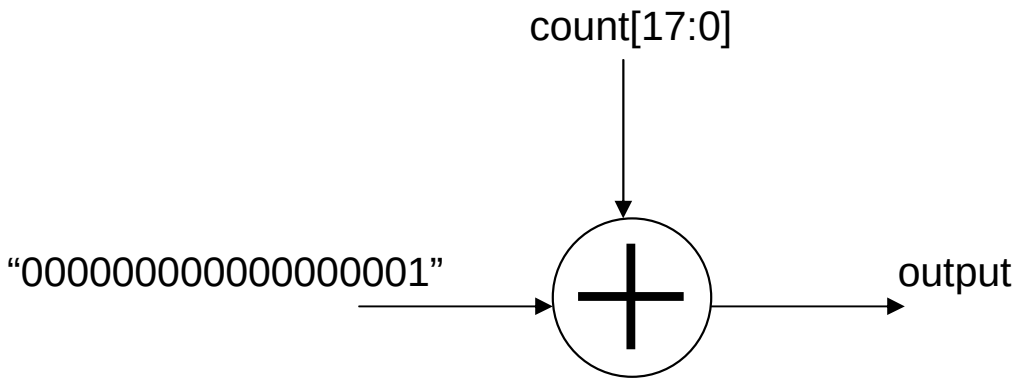
Improved Timer Structure



Use the carry out of the adder to detect rollover.

Output **timerDone** is delayed by one cycle, but this is not a problem in our system.

Building the +1 Circuit – Version #1

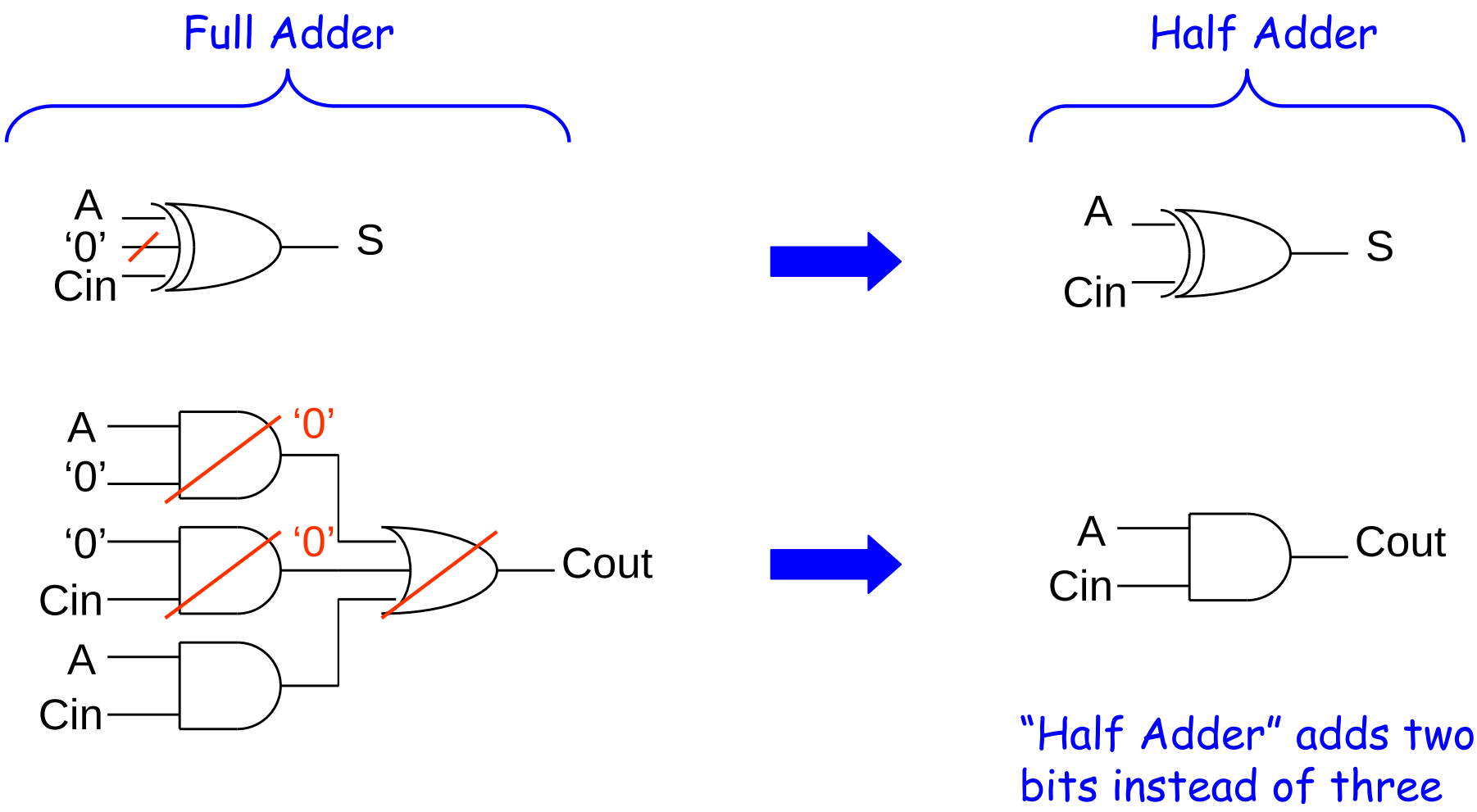


The adder could be built as outlined back in Chapter 8 using full adder blocks.

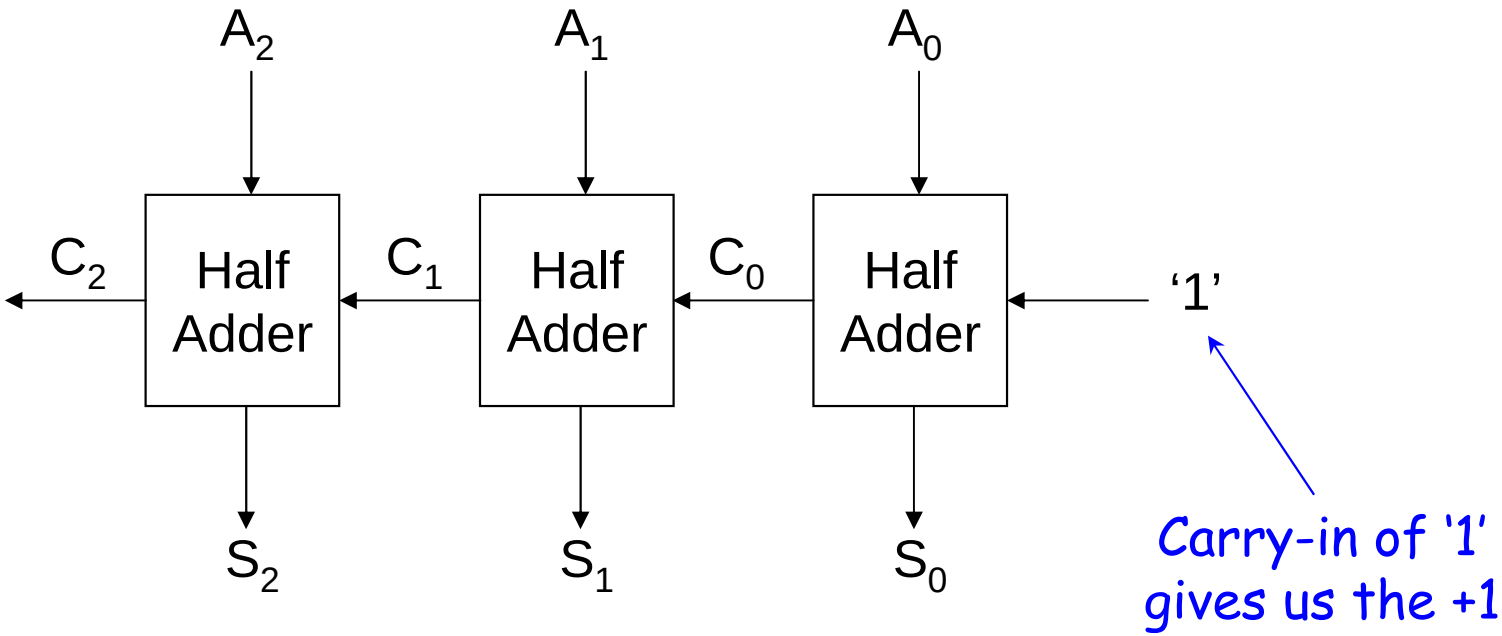
However, half the full adder inputs will be '0'. There ought to be a better way!

Hint: Any time a circuit has constant inputs (0 or 1) then the circuit can be simplified!

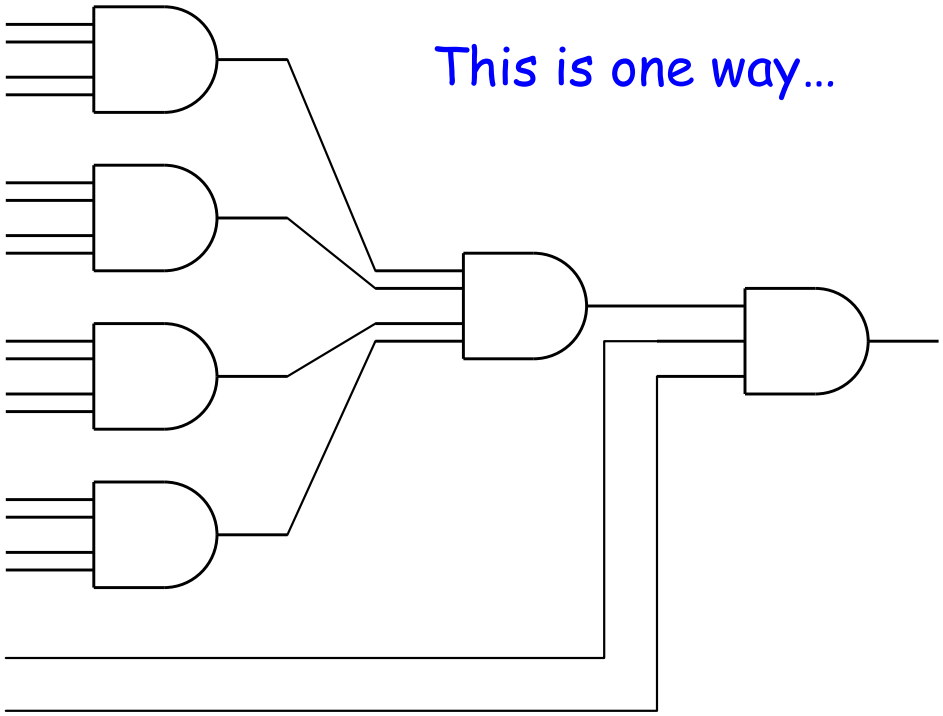
A Full-Adder with '0' Inputs



Building the +1 Circuit - Version #2



Building an 18-Bit AND



This is one way…

CAD tools are good at building structures like this from lower-level building blocks.

Just describe the AND in Verilog or VHDL and CAD tools will make a good choice.

If target technology has special structures for wide logic, CAD tools likely will use it

Debouncer Summary

- Structure is timer + FSM
- 2-state FSM makes NS logic trivial
- Asynchronous input "**noisy**" means we must be sure our system works with any input timing
  - If desired/needed, synchronize "noisy" input using one or more flip flops
- Counter too large for conventional techniques
  - Use MUX + register technique of Chapter 12
- Systems can usually be greatly simplified beyond the obvious design by using careful analysis