Debouncing a Switch

A Design Example
Background and Motivation
When you throw a switch (button or two-pole switch)...

• It often bounces...
Another switch...
Yet Another...
Still Yet Another...
Causes

• Mechanical switch
  - Not an instant, once-only make-or-break
  - Spring loaded – contacts literally bounce
Source of Errors

- 20ns clock period is very short compared to bouncing...
- 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 Timing Diagram

noisy

debounced

5ms 5ms

time

ECE 238L
Draw a System Block Diagram

Very reminiscent of our car wash controller...
The Design of the FSM
Draw a State Graph

S0
- noisy
- noisy' • timerDone

S1
- noisy • timerDone'

S2
- noisy
- noisy' • timerDone

S3
- noisy' • timerDone'
- debounced

noisy' • timerDone'
Debounce

Draw a State Graph

Debounced output is low...
Debounced output is high...

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...

ECE 238L
Reduce FSM to Logic

\[ NS = CS' \cdot \text{noisy} \cdot \text{timerDone} + CS \cdot \text{noisy} + CS \cdot \text{noisy}' \cdot \text{timerDone}' \]

\[ S_0 = CS \quad S_1 = CS \quad \text{noisy} = N \quad \text{timerDone} = T \]

\[ \text{clrTimer} = \text{noisy}' \cdot CS' + \text{noisy} \cdot CS \]

\[ \text{debounced} = CS \]
Reduce FSM to Logic

\[ NS = \text{noisy} \cdot \text{timerDone} + \text{CS} \cdot \text{timerDone}' \]

\[ \text{clrTimer} = \text{noisy}' \cdot \text{CS}' + \text{noisy} \cdot \text{CS} \]

\[ \text{debounced} = \text{CS} \]

This is smaller than one-hot implementation

In addition, the one-hot would require a reset input to get it to state \( S0 \)
noisy is an Asynchronous Input

- Signal noisy is asynchronous
  - No restrictions on pulse widths
- We will live with this possibility...
More on Asynchronous Input

Classical async input handling problem:
1. FSM may see noisy change and change state
2. Timer may not see clrTimer that results

Or, the other way around may occur...

Will this cause incorrect operation?
Asynch Input Problem

Look at the transitions - will previous slide’s problem cause a malfunction??

If you determine that a problem may result, what is easiest way to solve the problem?
Design of the Timer
Timer Calculations

• Assume system runs at 50MHz (20ns period)
• 5ms/20ns = 250,000
• An 18-bit counter will work...
• $2^{18}$ is a bit longer than 250,000 (262,144)
  - But is close enough to 5ms for our purposes
Timer Structure

• 19 inputs: 18 CS bits + 1 clrTimer bit
  - Very, very large truth table

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

\[ +1 \]

\[ \text{clrTimer} \]

\[ \text{clk} \]

\[ \text{D} \rightarrow \text{Q} \]

\[ \text{18-input AND} \]

\[ \text{timerDone} \]
Improved Timer Structure

This is a simpler way to conditionally generate zeroes. A synthesizer likely would have generated this from Verilog or VHDL code containing a MUX.
The adder would 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...
A Full-Adder with '0' Inputs

Full Adder

A
‘0’
Cin

A
‘0’
Cin

A
‘0’
Cin

A
Cin

‘0’

A
Cin

Cout

S

A
Cin

Cout

S
A Half-Adder

Called a half-adder

Will add 2 bits together and generate sum and carry.
Building the +1 Circuit - Version #2

Half-adders add A's and carries '1' on right end is the +1
Building an 18-Bit AND

This is one way...

Synthesizers are good at building structures like this from lower-level building blocks. Just write an 18-bit AND in your Verilog or VHDL code...

If circuit has special structures for wide logic, synthesizer likely will use it (carry/cascade logic in an FPGA is an example)
Debouncer Summary

- Structure is timer + FSM
- 2-state FSM makes NS logic trivial
- Asynchronous input makes it possible (but unlikely) to miss a glitch on input noisy
  - If desired, synchronize noisy with a FF
- Counter too large for conventional techniques
  - Use MUX+register techniques of Chapter 12
- NOTE: FSM technique resulted in FF+counter mentioned previously...