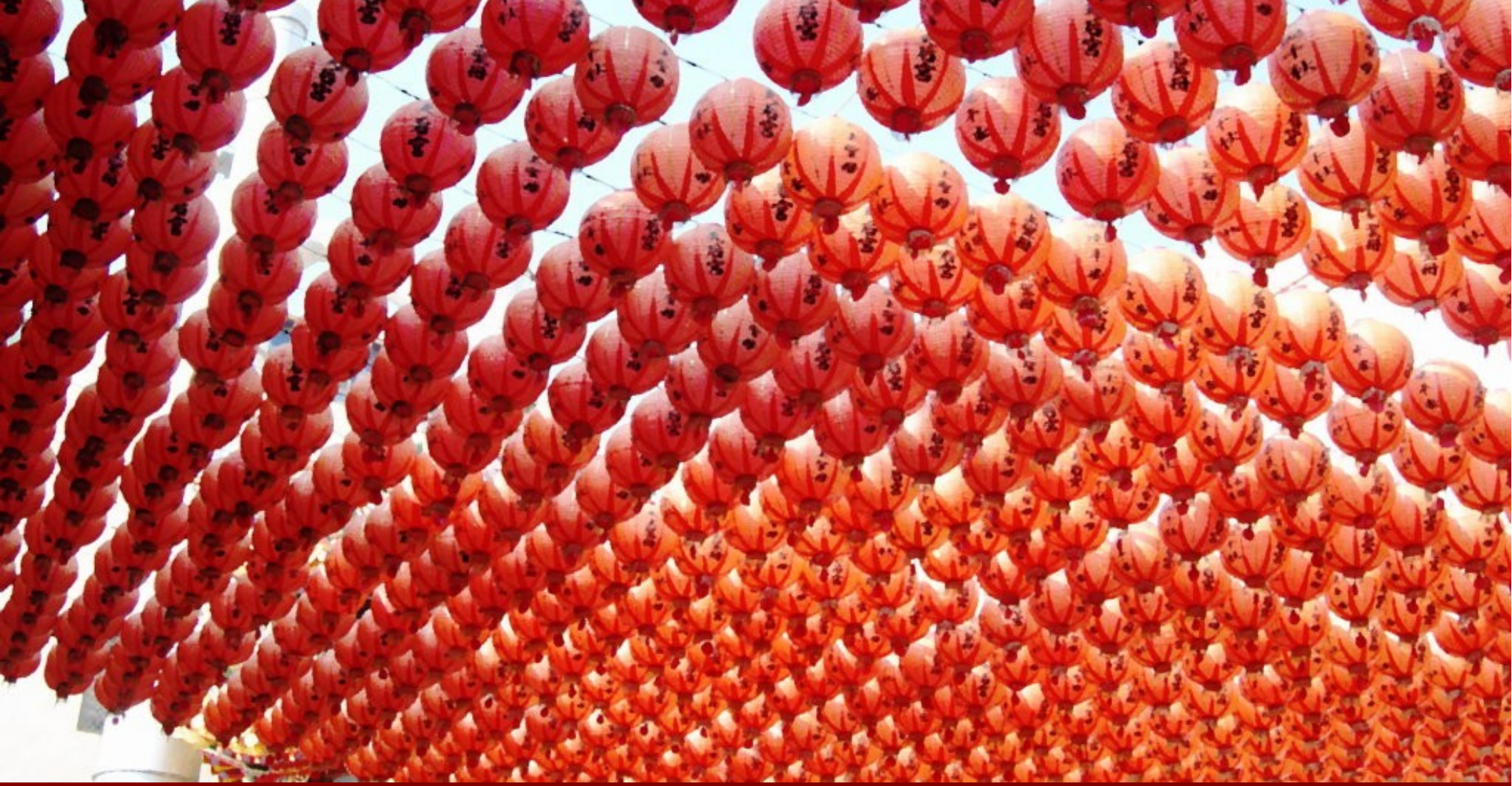


Adventures with Particle

Pete Soper, Apex Proto Factory

TriEmbed September 13, 2021

Introduction

- Did embedded development from the 70s to mid 80s and at various times afterward at a hobby level: looped back to it after leaving Sun Microsystems Java Engineering Group a little before Oracle bought Sun
- Owner, Apex Proto Factory (2015)
  - Chartered to facilitate embedded prototype development
  - Some hardware development as well as firmware

Argon: Nordic nRF52840+ESP32

A Better Alternative to noInterrupts()/interrupts()

- What is an interrupt?
- Why are noInterrupts()/interrupts() needed?
- Basic limitations of noInterrupts() and interrupts()
- A better alternative with Particle DeviceOS that could be ported to Arduino or elsewhere

What is an Interrupt?

- *Like* a function call that can be called almost any time
- To use interrupts in our programs safely we have to control what is “almost”
  - Any time must not include the time spent manipulating data that your interrupt code will also manipulate if this can cause incorrect results
- What happens if “almost” is “too much”?
  - Program misbehavior that can happen when you can’t afford it
  - Misbehavior can be hard to find

Recipe for program misbehavior

- Code inside “begin” or “loop” or a non-interrupt function:

  A = A

  (“Interrupt function” call happens here)

  + 1; // this not finished yet

  B = A;
- Code inside the interrupt function:

  A = A + 1;

  B = A;
- The “old” value of A gets one added to it and the sum overwrites int value, so B ends up one less than it should be

The fix

Make the code outside the interrupt function look like this:

```
noInterrupts();
A = A + 1;
B = A;
Interrupts();
```

The two statements are now “atomic” and cannot be subdivided in time by an intervening interrupt. Variable B can no longer have missing increments because of any interrupt (Arduino) or pin interrupt (Particle)

Limitations

- Interrupts() and noInterrupts paint your code with a brush that is either black or white.
  - What if interrupts were already disabled? Your code might be unable to know that, and it can cause chaos to call interrupts() before handling is complete (e.g. some **other** interrupt can now happen that the “system” is not ready to cope with)
  - On Particle noInterrupts() only means “no ***PIN*** interrupts” (my potential mis-adventure!)

Better alternative:AtomicSection

- A C++ class in Particle DeviceOS with the following semantics:
  - When an AtomicSection object is created the current interrupt state is saved and “almost all” interrupts are disabled: more than pin ones (only excludes specials like non-maskable and “fault” interrupts)
  - When an AtomicSection object is “-un-created” aka “destroyed” the previous interrupt state is restored

AtomicSection Examples

```
void func() {
  AtomicSection lock;
  A = A + 1;
  B = A;
}
or:
<some code>
{AtomicSection lock;
  A = A + 1;
  B = A;
}
<some more code>
```

Huh? How did that happen?

<hidden call of AtomicSection constructor>

A = A + 1;

B = A;

<hidden call of AtomicSection object destructor>

- The “scope” of the object is the lifetime of the function or standalone “block” of code inside the curly braces, so the interrupts are disabled (if not already) and restored to previous state exactly where you want it to be

The AtomicSection source code

```
class AtomicSection {
        int prev;
public:
        AtomicSection() { // constructor
                prev = HAL_disable_irq();
        }
        ~AtomicSection() { // destructor
                HAL_enable_irq(prev);
        }
};
```

- It should be straight forward to make versions for non-Particle platforms

Advantages

- Don’t care if inside an interrupt handler or not
- If some other interrupt such as a clock interrupt causing a thread switch could create the corruption scenario, now it cannot because clock interrupts are held off along with GPIO pin interrupts

Debugging beyond printing

- Your program was running, you made some changes and now it misbehaves badly
  - e.g. no output of any kind
- Sometimes you can think about your changes and the light goes on and you know how to fix the code
- Other times you can’t see anything wrong, so you need to inspect the program state while it runs
- Time for some print statements!

A different approach: gdb, the Gnu DeBugger

- Interactive tool to:
  - Start program execution
  - Stop at (almost) arbitrary places
  - Inspect current values of variables
  - Single step program one statement at a time
  - See the set of function calls that got the code to where it is
  - Trap references to variables and show exactly what changed them
  - Execute an arbitrary function at any time
  - And much much more

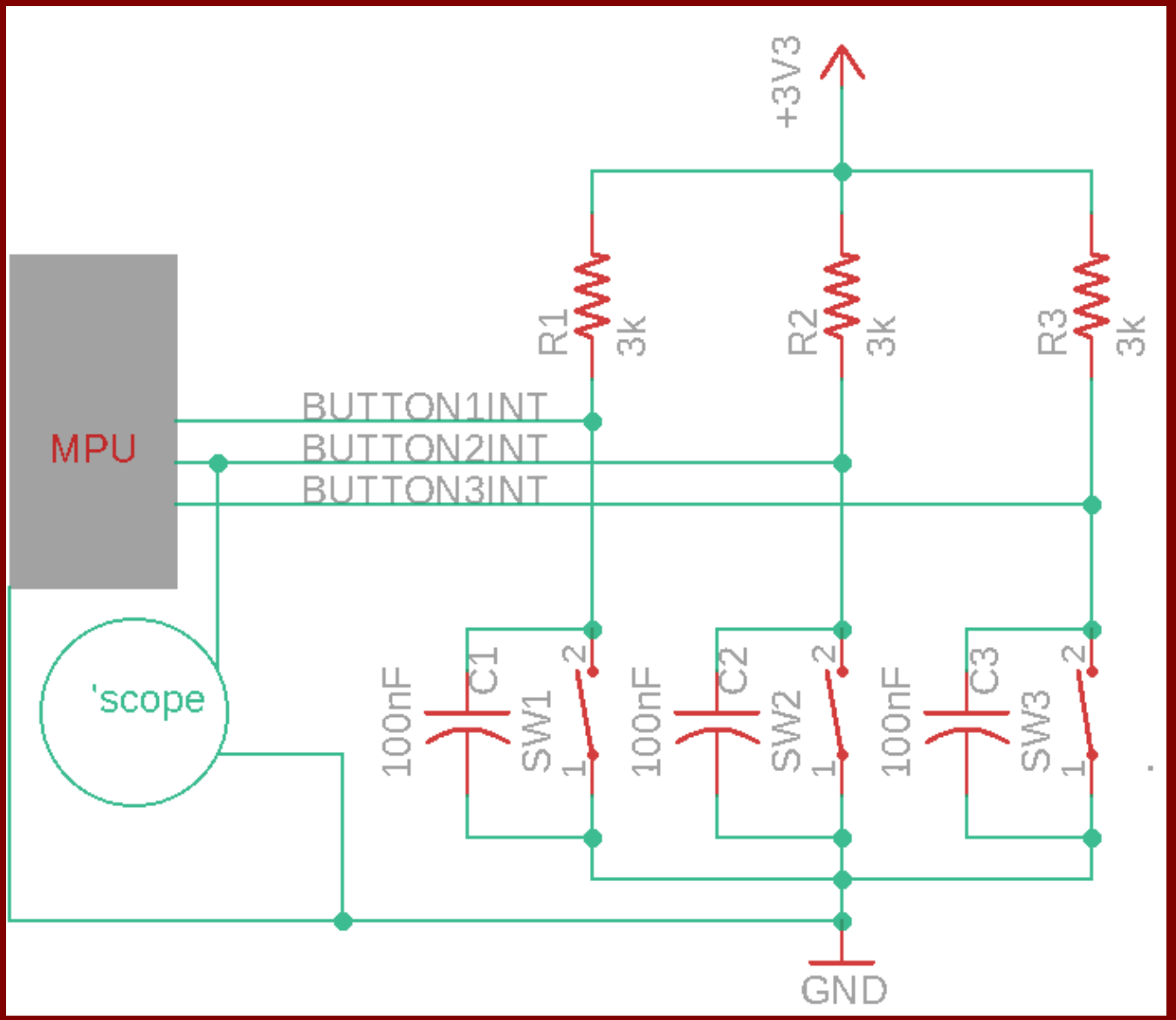
Adventure: gdb along side Particle Workbench

- Integrated into Particle Workbench
  - GUI for aforementioned operations
  - Pulls strings on gdb to accomplish actions
- Works with Windows, Mac OS, and some flavors of Linux
  - But not mine (Ubuntu Mate)
  - Had to “route around the problem”
  - Launch gdb from command line “along side” workbench, the latter being strictly for builds and flashing

gdb demo: device driver dive

(Demo of Particle Workbench, gdb “along side it”)

Adventure: phantom interrupts

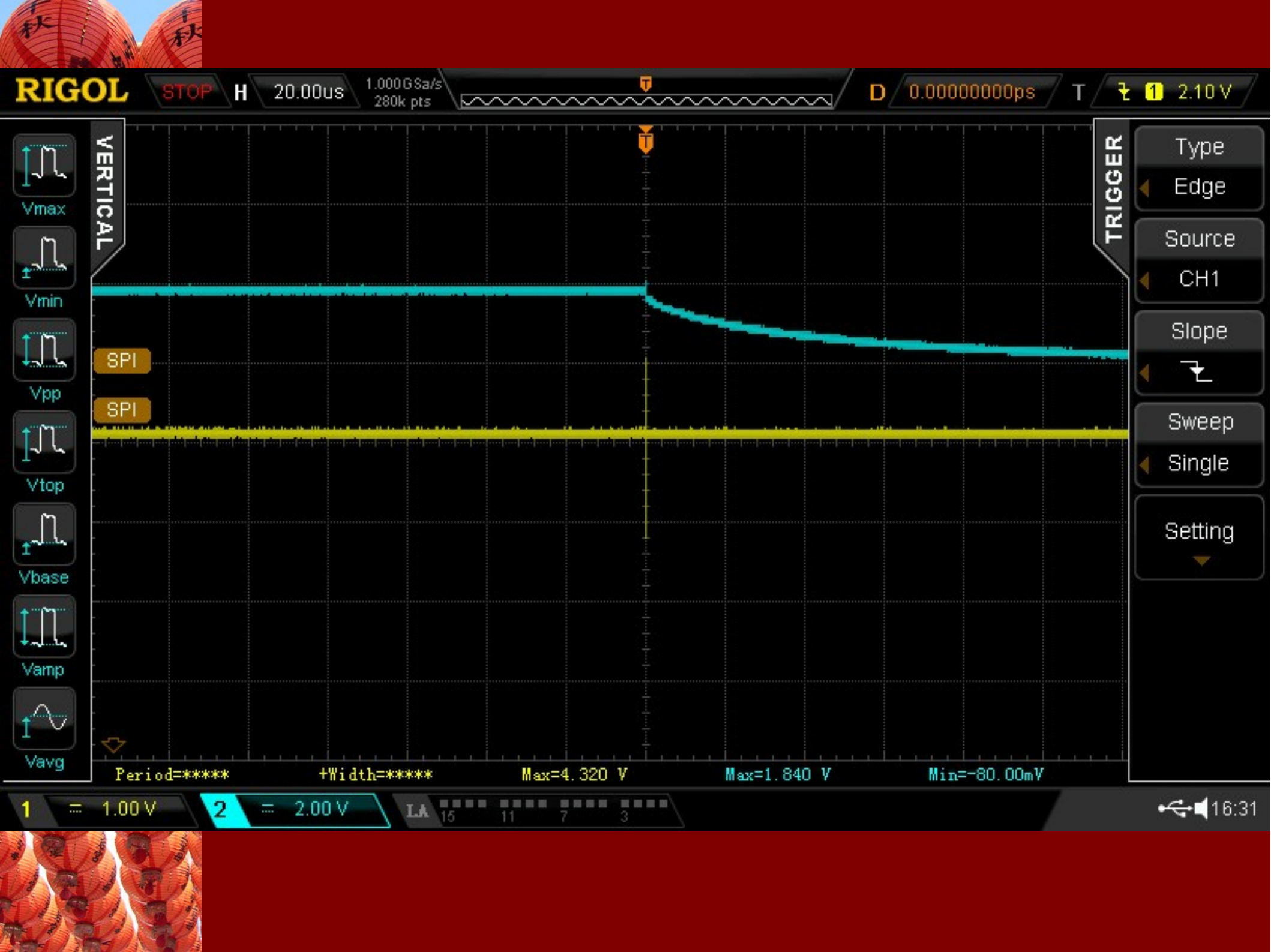


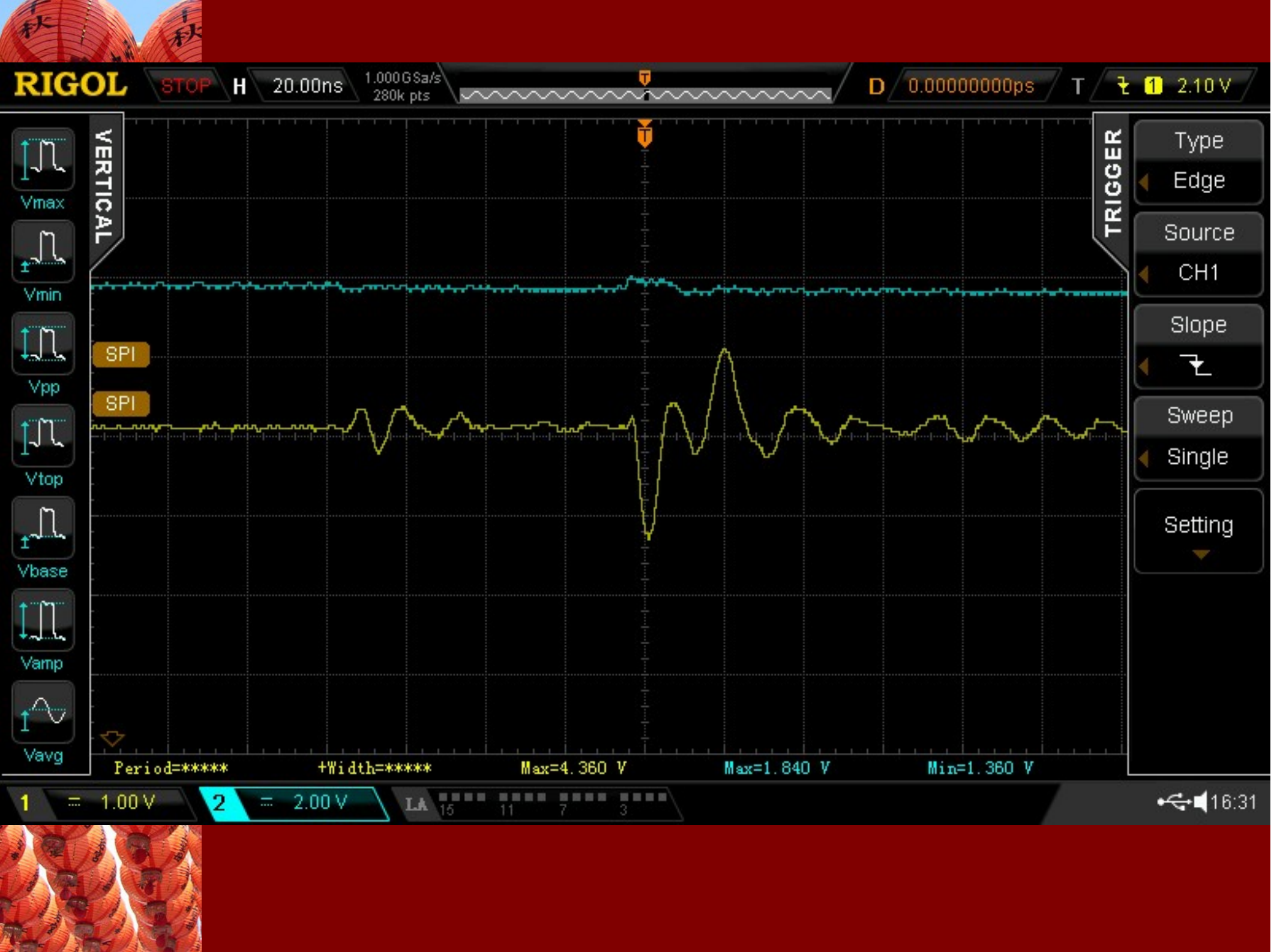
Context

- Three SPST pushbutton, momentary contact, normally off switches
- Each switch has a 100nf cap across it and the cap is in series with a 3k ohm resister tied to a 3.3 volt supply and the bottom of the switch is tied to ground
- Pressing a switch shorts it to ground, releasing the switch charges the capacitor and the output rises to 3.3 volts
- About 10 inches of wire between switches and MPU interrupt pins

The mystery: phantom interrupts

- Pressing switch randomly causes the switch two interrupt to trigger
- Same for the switch three trigger, but less frequently
- In the next slide the blue trace is switch one signal falling to ground with a press with each horizontal division being 20usec
- The yellow trace is the switch two interrupt signal close to the MPU
- The slide following is 20nsec/division


RIGOL
STOP
H 20.00us
1.000GSa/s
280k pts
D 0.00000000ps
T 1 2.10V
VERTICAL
TRIGGER
Vmax
Vmin
Vpp
Vtop
Vbase
Vamp
Vavg
SPI
SPI
Type
Edge
Source
CH1
Slope
Sweep
Single
Setting
Period=*****
+Width=*****
Max=4.320 V
Max=1.840 V
Min=-80.00mV
1 1.00V
2 2.00V
LA 15 11 7 3
16:31


RIGOL
STOP
H
20.00ns
1.000GSa/s
280k pts
D
0.00000000ps
T
1
2.10V
VERTICAL
Vmax
Vmin
Vpp
Vtop
Vbase
Vamp
Vavg
SPI
SPI
TRIGGER
Type
Edge
Source
CH1
Slope
Sweep
Single
Setting
Period=*****
+Width=*****
Max=4.360 V
Max=1.840 V
Min=1.360 V
1
1.00V
2
2.00V
LA
15
11
7
3
16:31

How can this be? Inductance?

- In the meantime I removed the capacitors and used software debouncing

More info

- Particle: https://particle.io
- gdb: https://en.wikipedia.org/wiki/GNU_Debugger
- Sensor chip: https://www.nxp.com/docs/en/data-sheet/FXOS8700CQ.pdf
- Note: this chip is “end of life” and cannot be purchases after 12/21 :-(
- But many similar chips are out there

1.01 9/13/2021