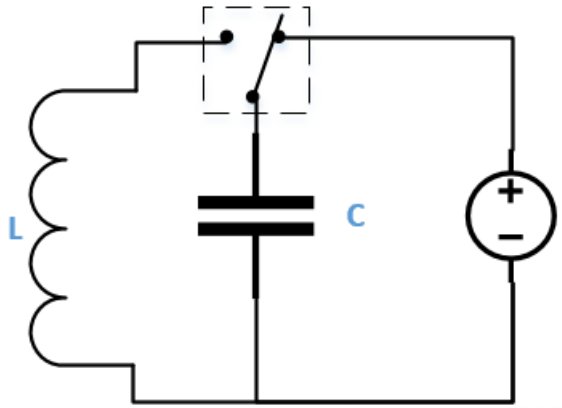
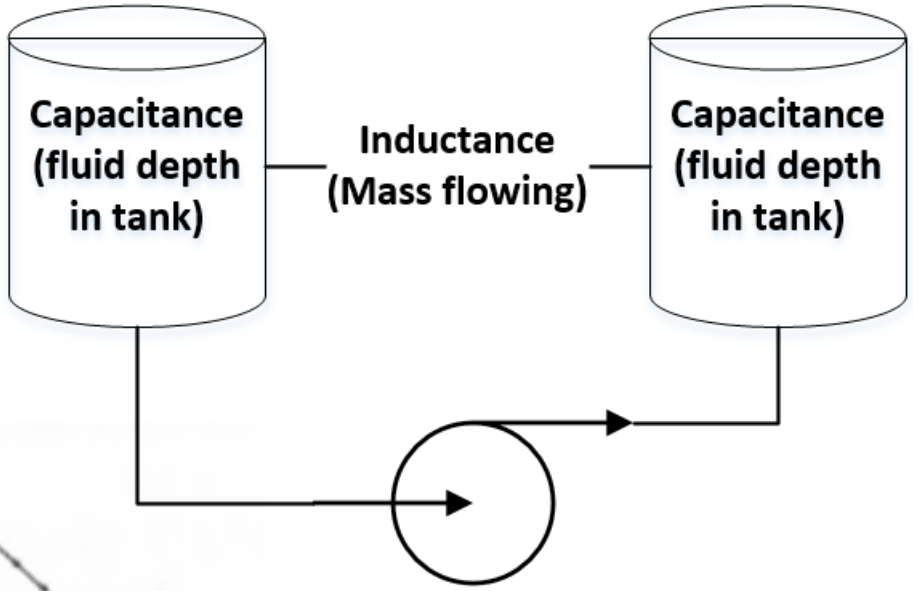
ECEN 3753: Real-Time Operating Systems

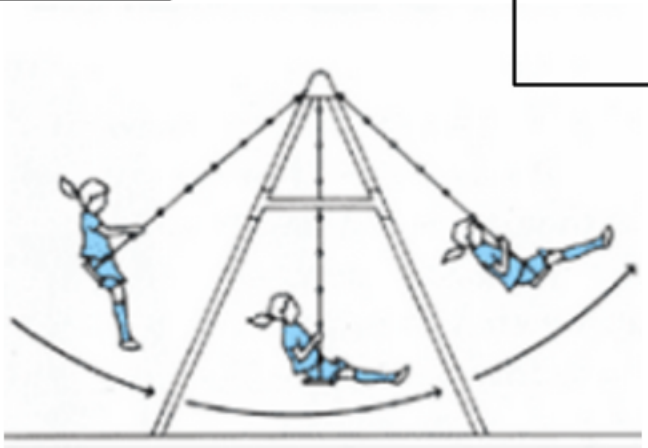
Capacitive Sensing

Capacitive Analogies

$$f = \frac{1}{2\pi\sqrt{LC}}$$

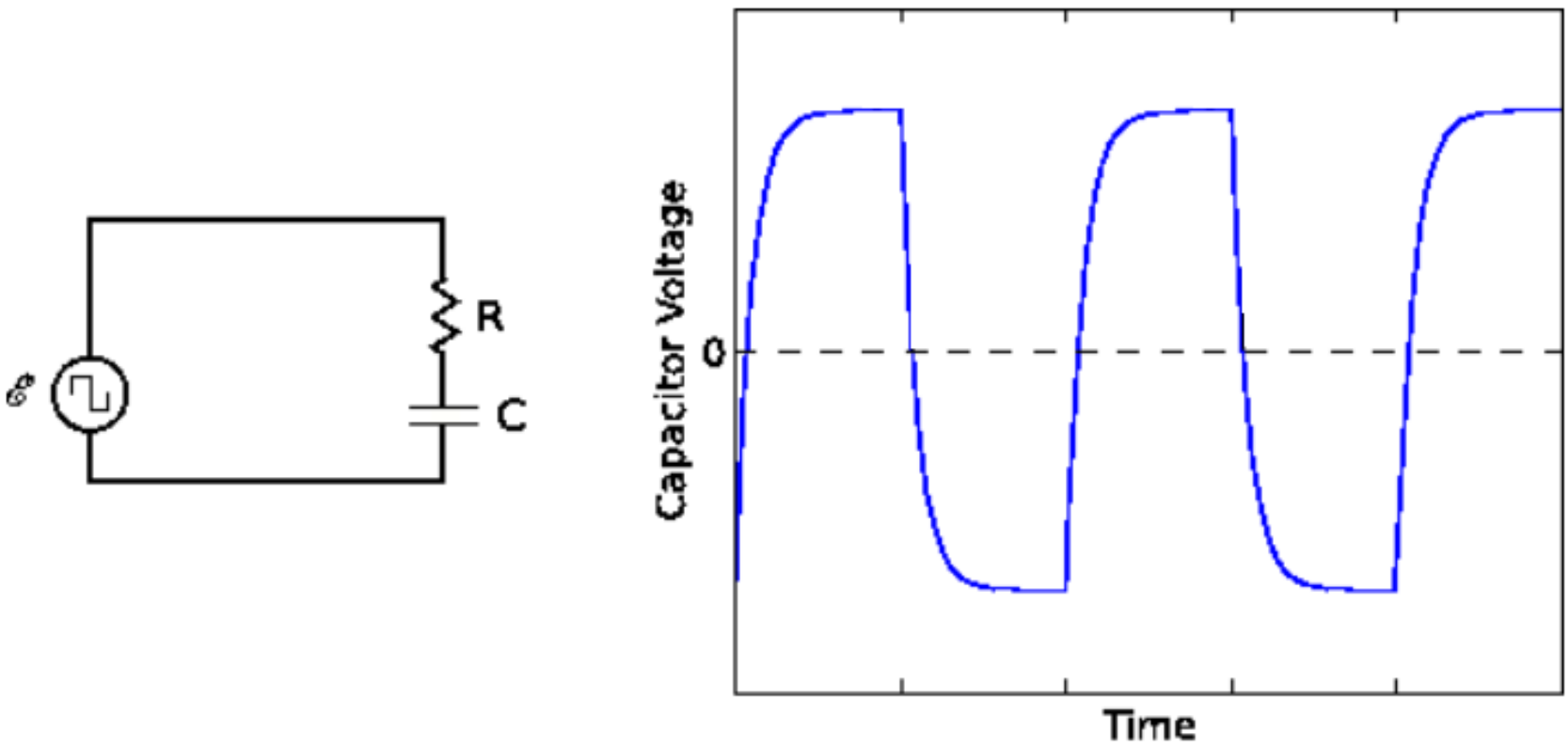






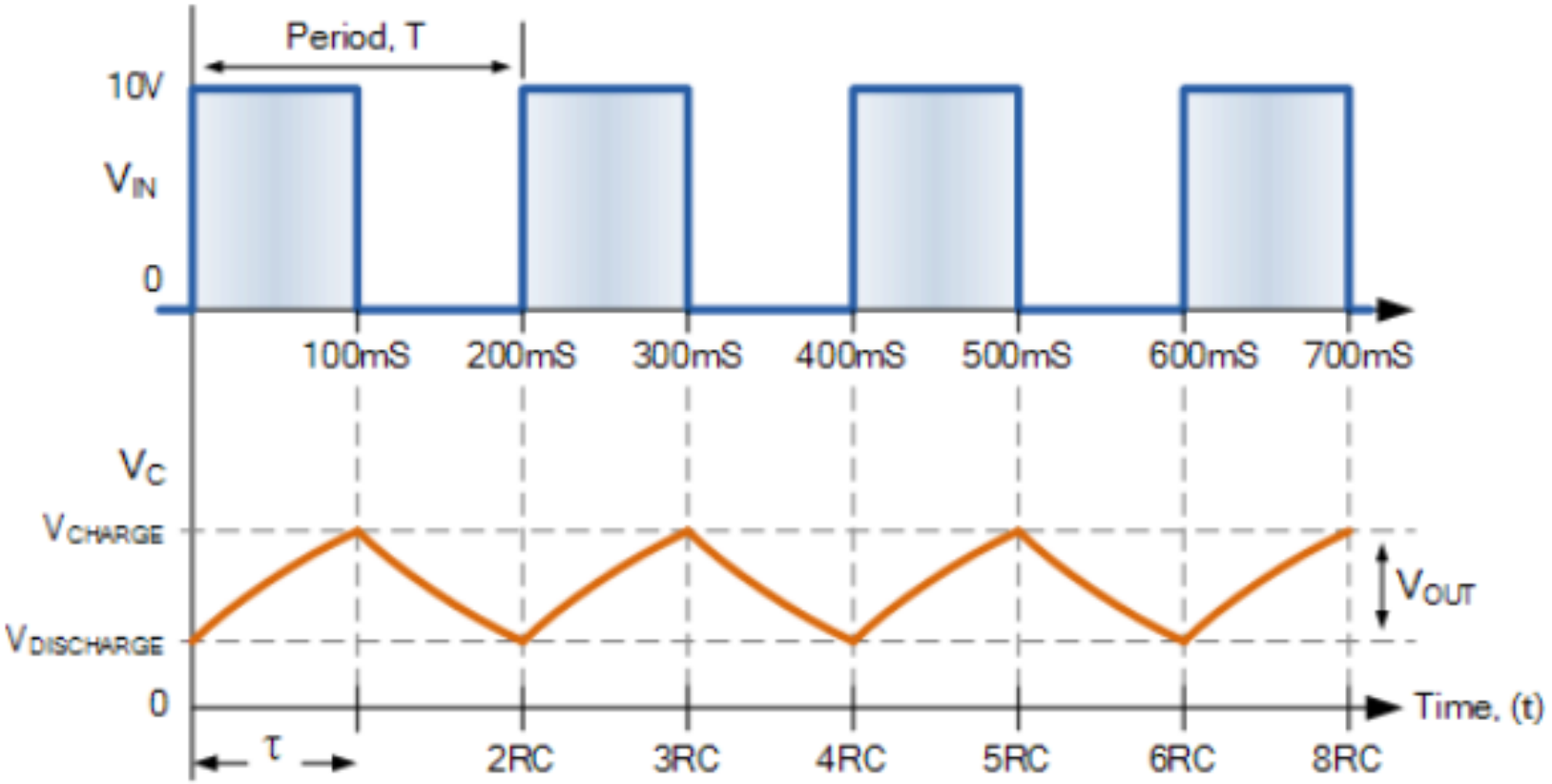
Swing picture from http://www.puremusic.com/90cc2.html

Saturating RC Oscillation



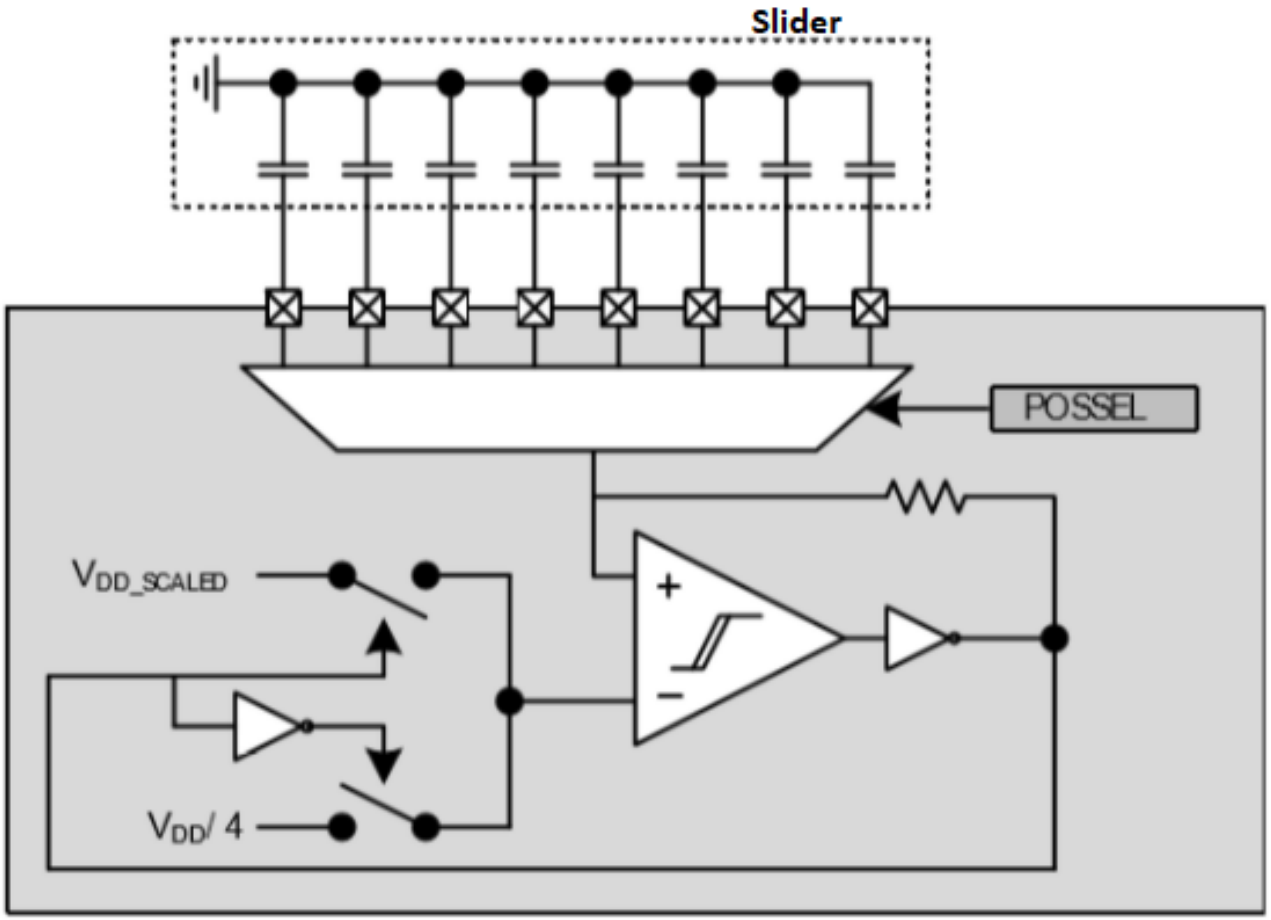
Derived from: http://webpages.ursinus.edu/tcarroll/phys112/labs/node7.html

Self-adjusting RC Oscillation



Picture from: https://www.electronics-tutorials.ws/rc/rc-integrator.html

EFM-32 CapSense



Derived from: https://www.silabs.com/community/mcu/32-bit/knowledge-base.entry.html/2015/09/18/how_can_i_implement-hcxv

Low-Power was Key for the SDK Examples

It's clear that SiLabs wanted to show off their power miserliness, and so didn't consider the concurrent use of CapSense and Micrium OS in their firmware examples.

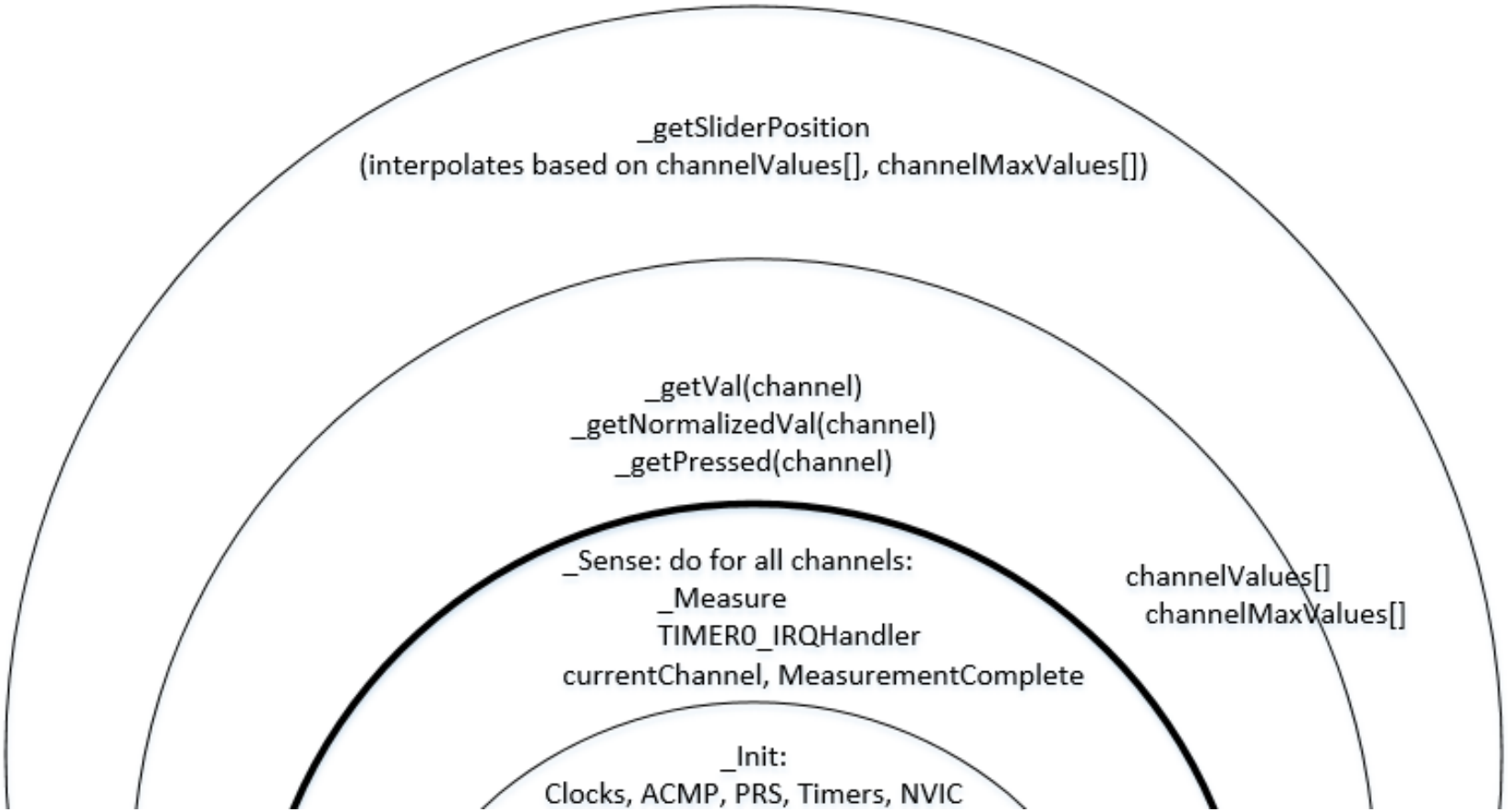
- EFM32PG12
  - Possibly only two timers exist: T0 and T1 (unclear if WideTimers reuse the 16-bit timers)
  - Can use timer interrupts to exit low-power mode
- CapSense, as written
  - T0: (SomeClock(?)/512) for time reference. Interrupt when $>$ 10 ticks
    - What is the value of this delay? Measure it in SDK example!
  - T1: counts free-running oscillator, /1024 (T1$\leftarrow$PRS[0]$\leftarrow$ACMP1.PosEdge) up to at least 64Ki counts (assuming at least a 16b register)
- Micrium OS
  - T0 is the timer used for OS_tick
  - Idle task generally goes into low power (not needed for this

When Using Micrium's Timer Services

OS_tick Resolution: If we choose a magnitude that is:

- Too small: we'll have a precise and more accurate timebase, but will potentially spend more time context-switching
- Too large: we'll either lose accuracy in our measured frequency, or need to poll to synch up to the timer.

capsense.c



CAPSENSE prefix removed from function names, for brevity.

What's with the MaxValues per channel?

In-field calibration!

- The highest frequency every seen in each channel (since power-on) is retained and used for normalization of ongoing measurements of oscillation frequencies
- 75% of max-seen on a channel is used as that channel's "touch threshold" in CAPSENSE_touch().
- Silicon Labs uses a 32b variable, in spite of appearance that the underlying timer is 16b.

SDK's Touch Project: main()

main() in this project waited 100ms, did a CAPSENSE_Sense() cycle, printed results, repeat.

10Hz-ish seems to be sufficiently-frequent capacitive sensing to provide reasonable value in this course.

Desired Compatibility with Micrium

- Set up a reasonable OS clock frequency to supplant T0's use in capsense.c
  - T0 should NOT be touched by _Measure or _Init anymore.
- Move _Sense and T0's ISR data collection into a task that waits the "long time" (e.g. 100ms) to start sequentially working through all channels (the delay from main()), waiting appropriate OS_ticks to count oscillations on each channel.
  - No more ISR.
  - Priority? (High, at least when sensing a single channel's capacitance)
    - Varying? (Esp if there is other important work to do between channel sensings.)
  - Power? (Don't care too much. Maybe even leave ACMP enabled when it's not needed, and probably avoid low-power CPU modes (force the idle task to stay OUT of low power modes).)

Investigation Grit

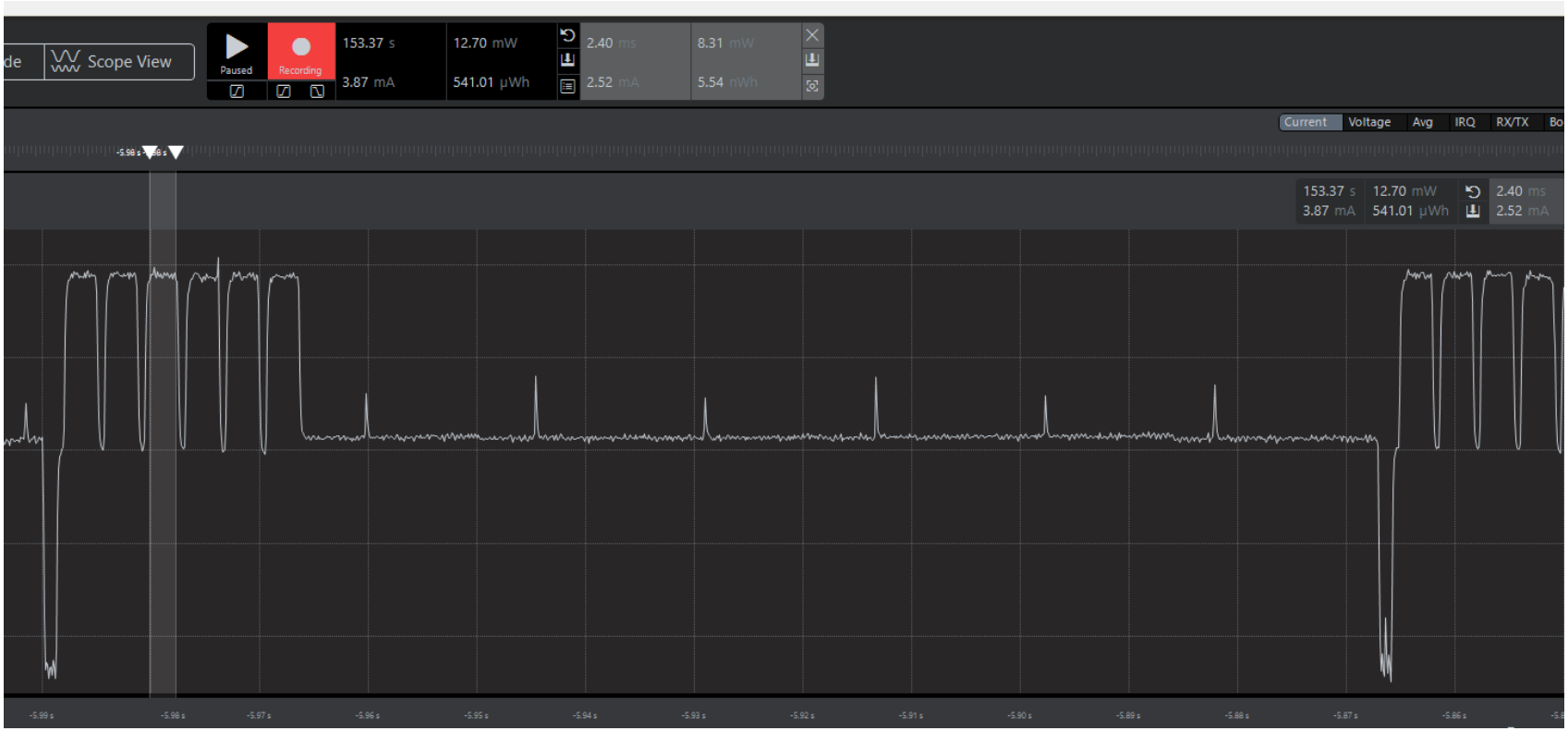
Often, our plans are thwarted by details.

SMOP = **S**mall **M**atter **O**f **P**rogramming

- Silicon Labs appears to have recently refactored their online docs (https://www.silabs.com/developers/micrium)
  - Micriμm and μC/OS docs are available at the top level now
  - Micriμm half had broken links throughout when we were looking in the week of Feb 8, 2021.
  - μC/OS half is not the same, but gave us leads.
- OS Clock appeared non-configurable in our Micriμm source code (hard-configured in our port?)
- Prescaler couldn't be reduced (it was already 1 when we peeked from the debugger)
- TopValue of T0 couldn't be hacked to 1/N of the norm (TopValue was either not used, or was MAX_UINT–which could have been assumed throughout Micriμm, so we viewed it as risky to muck with.)

Analysis and Use of OS timer

First of all, what time values are we trying to emulate?



What DID the documentation indicate?

Given that the documentation had lots of broken macros, and google search results merely pointed to the new top-level split in docs, it was challenging to find info about the OS tick.

- Micrium doc re: dynamic tick on MuCOS had an example where the OS tick rate was 100Hz
- Comment at embeddedcomputing.com (paraphrased):
  - Using uCOS as an example, on a 32-bit processor running at 300 MHz, the overhead required for the kernel to process 1000 ticks per second would likely not exceed 1 percent of the CPU's cycles. (Implies that the timer could run at 1KHz.)
- Micrium's Kernel Time Management API doc: provides a getter, but not a setter of the tick rate.
- uCOS had a constant definition that could be set to change the OS_tick rate, but this was not found in the code that come with Simplicity, so we're guessing that the port that we have may be limited to the 1ms that we measured.

Q: could we live with 1ms OS_tick?

Recall that the "counting period" for T1 counting of oscillations was about 2.5ms

- Counting a 2.5ms interval with a 1ms timer would have introduced a LOT of error (1 tick would equate to 40 percent error counting a channel's oscillation)
- What if the 24 ms became somewhat less than 100ms (if we allow summing over 10ms/channel instead of 2.5ms/channel)
  - We'd be limited to about 5 samples/s instead of 8-10, if we maintain the 100ms inter-sampling period.
  - T1 would accumulate 4x as many counts. Are there enough bits in it?
    - Debug: 85 or so for the max counts with the non-Tasking CapSense implementation. x4 would almost fit in a uint8, and we have either a uint16 or uint32 register. Variables are already uint32.

A: Apparently, Yes we can

Students reported in that their experiments were slower at providing capsense data (as expected: should be sampling about 1/2 to 1/4 as fast), but were not flaking out after a bit of time (LEDs flashing, buttons seemingly ignored as they were previously.)

Post-analysis:

- There may be a way to adjust the OS_tick. That would be preferable in some situations, so that the finger position on the slider can be measured more frequently.
- Not sure if students experimented with a Delay() function that was built-up from a non-OS example, or OSTimeDly() (which is from the OS)
- Micriμm's OSTimeTickRateHzGet() should be used whenever converting from units of time to ticks or vice-versa.
- Don't delay 100ms between channel samplings, if they're going to require nearly that long–perhaps simply always leave the new TaskingCapSense running (sleeping)
- If TaskingCapSense's priority is left low (therefore counts may vary more), add level of normalization before the Values/maxValues, to ensure those arrays are scaled by actual T1 periods. (Use OSTimeGet() around OSTimeDly() $\rightarrow$ task sleep duration)