Test
Code
Debug
Refactor
The Cycle Repeats
Hardware Is Not An Obstacle
System Tests Make Development More Predictable

“Fibonaiku”

Fibonacci Haiku

Test 1
Code
Debug
Refactor
The Cycle Repeats
Hardware Is Not An Obstacle
System Tests Make Development More Predictable

Test
Code 1
Debug
Refactor
The Cycle Repeats
Hardware Is Not An Obstacle
System Tests Make Development More Predictable

Test
Code
Debug 2
Refactor
The Cycle Repeats
Hardware Is Not An Obstacle
System Tests Make Development More Predictable

Test
Code
Debug
Refactor 3
The Cycle Repeats
Hardware Is Not An Obstacle
System Tests Make Development More Predictable

Test
Code
Debug
Refactor
The Cycle Repeats 5
Hardware Is Not An Obstacle
System Tests Make Development More Predictable

Test
Code
Debug
Refactor
The Cycle Repeats
Hardware Is Not An Obstacle
System Tests Make Development More Predictable

Answers

Why System Testing?

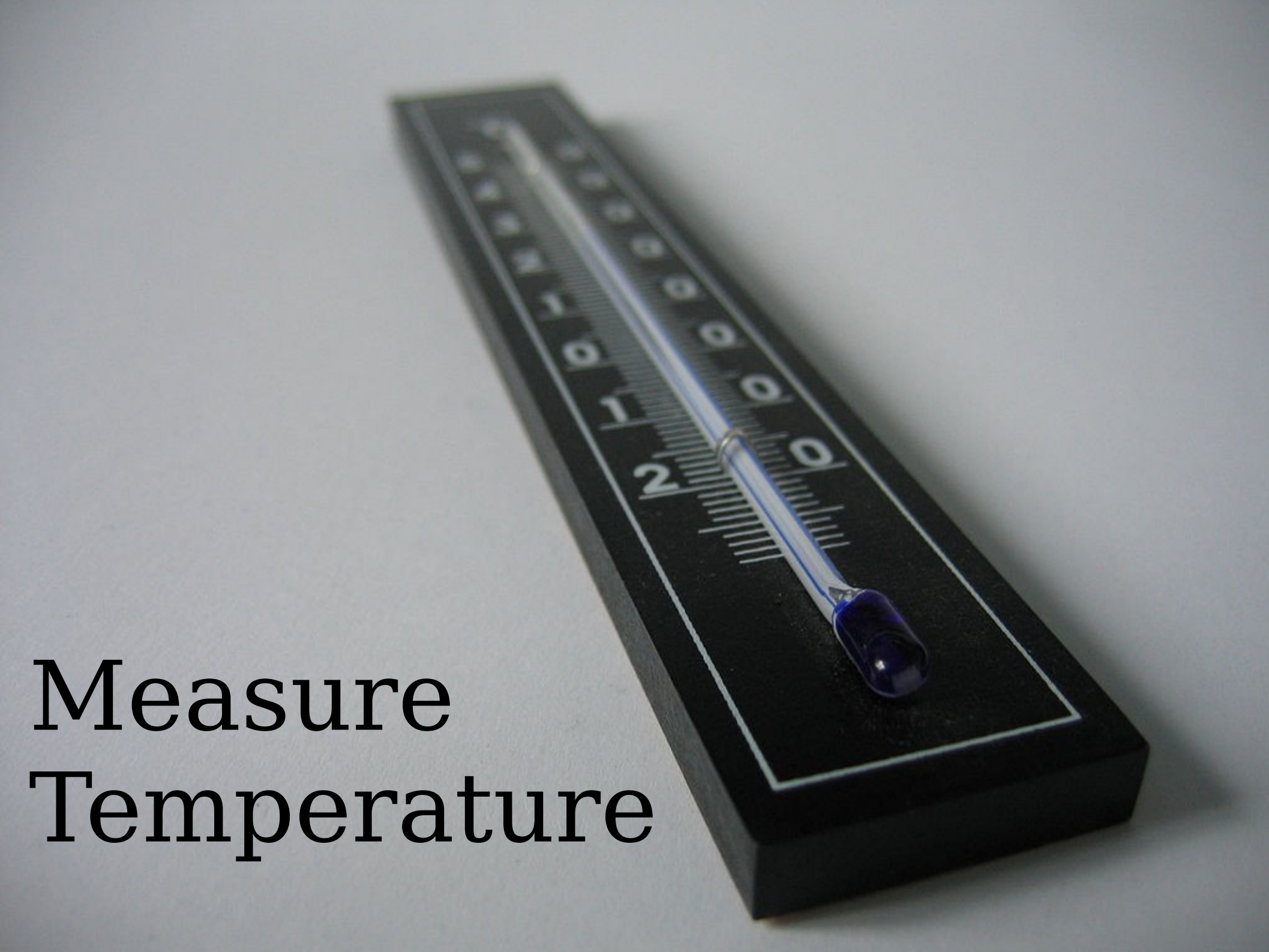
Why
System Testing?

What Is System Testing?

What Is
our product supposed
to do?

example

It Should


Measure
Temperature

It Should

Output Temp Once Per Second

It Should


Return Version
On Command
DRAFT VERSION

System Testing

NOT

Unit Testing

Ignore

When We

Ignore

We Can

Focus

On Features

We Know

When Features Are Complete

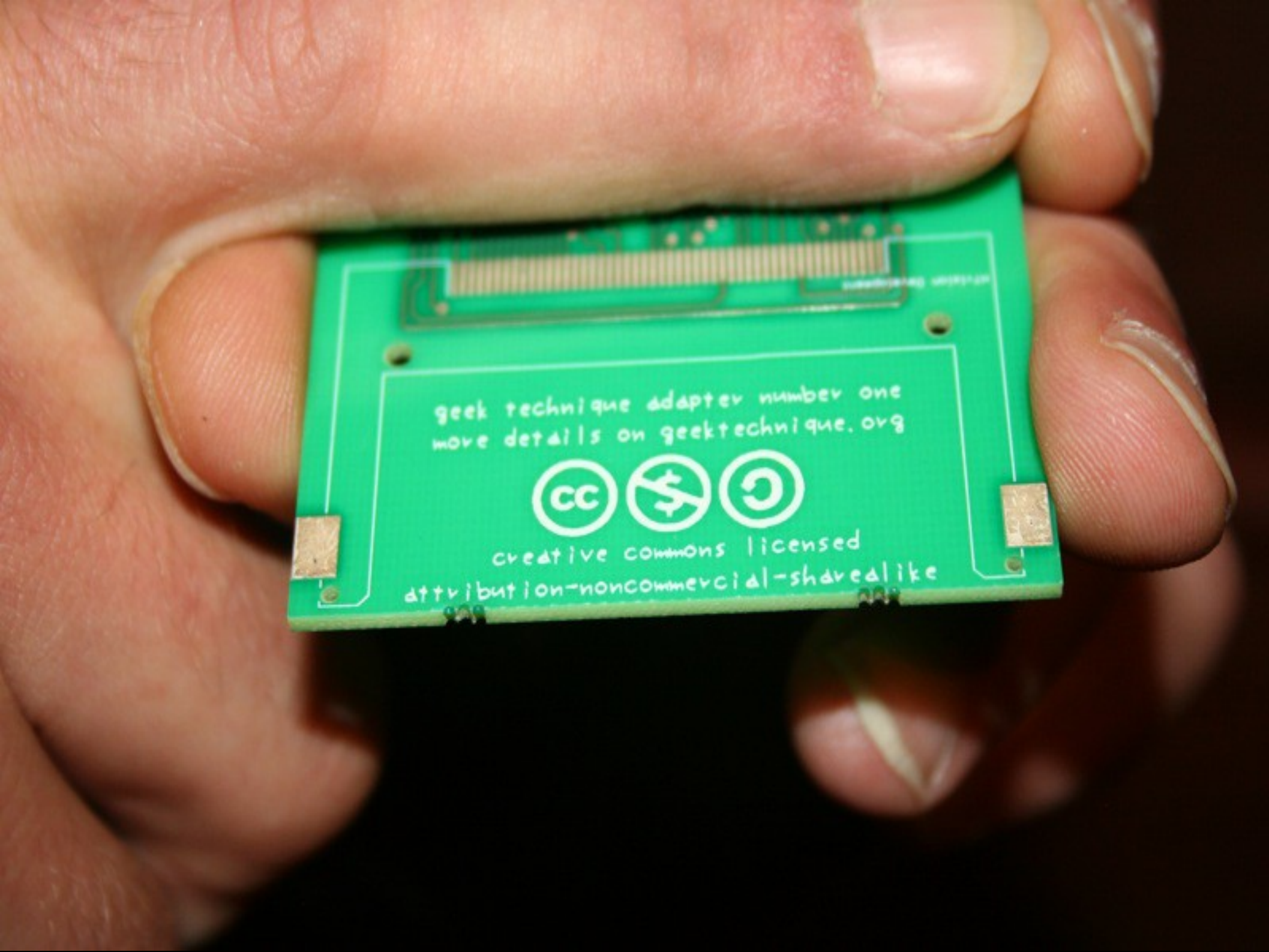
We Know

That Features Still Work

We Know

```
static inline uint32 ConvertAdcCountsToPicovolts(uint32 counts)
{
  // ADC bit weight at 10-bit resolution with 3.0V reference = 2.9296875
mV/LSB
  uint32 picovoltsPerAdcCount = 2929688;

  // Shift decimal point by 6 places to preserve accuracy in fixed-point math
  return counts * picovoltsPerAdcCount;
}

static inline uint16 ConvertPicovoltsToMillivolts(uint32 picovolts)
{
  const uint32 halfMillivoltInPicovolts = 500000;
  const uint32 picovoltsPerMillivolt = 1000000;

  // Add 0.5 mV to result so that truncation yields properly rounded result
  picovolts += halfMillivoltInPicovolts;

  // Divide appropriately to convert to millivolts
  return (uint16)(picovolts / picovoltsPerMillivolt);
}
```

and

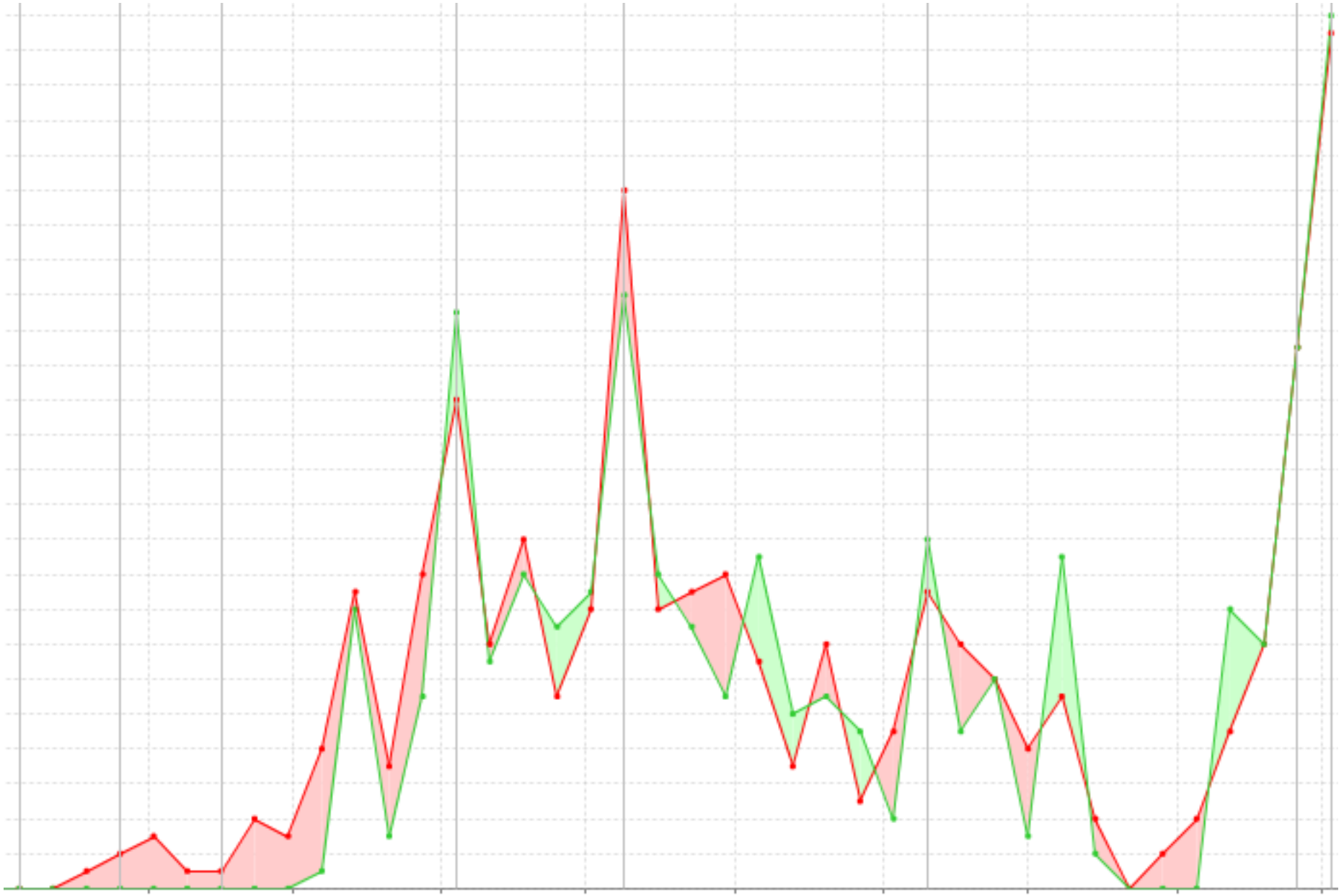

geek technique adapter number one
more details on geektechnique.org
creative commons licensed
attribution-noncommercial-sharealike

Work Together

We Even Know

Repeatability

Reliability



More Data == Good

Automate

How Often

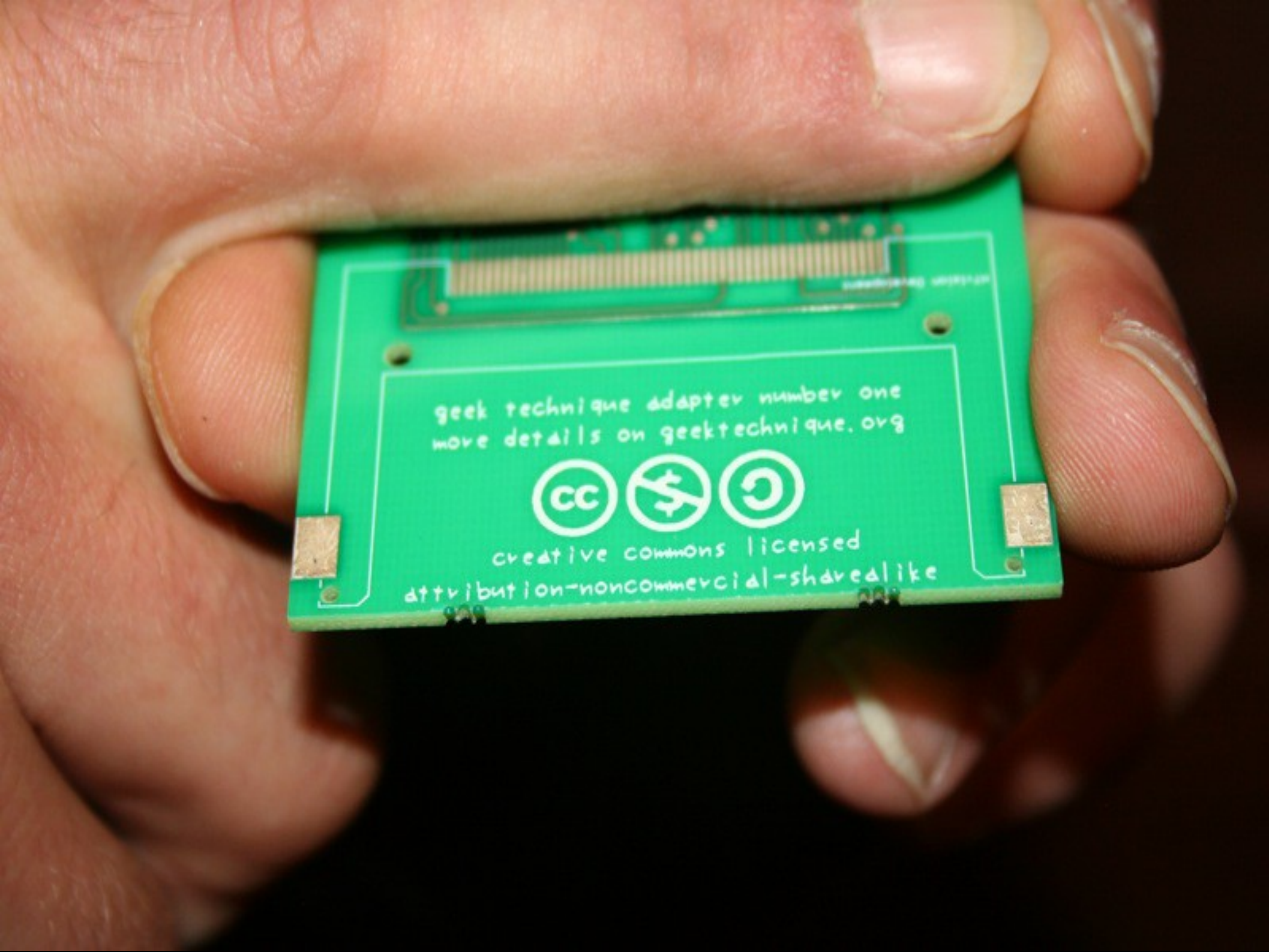
How Often

Obviously

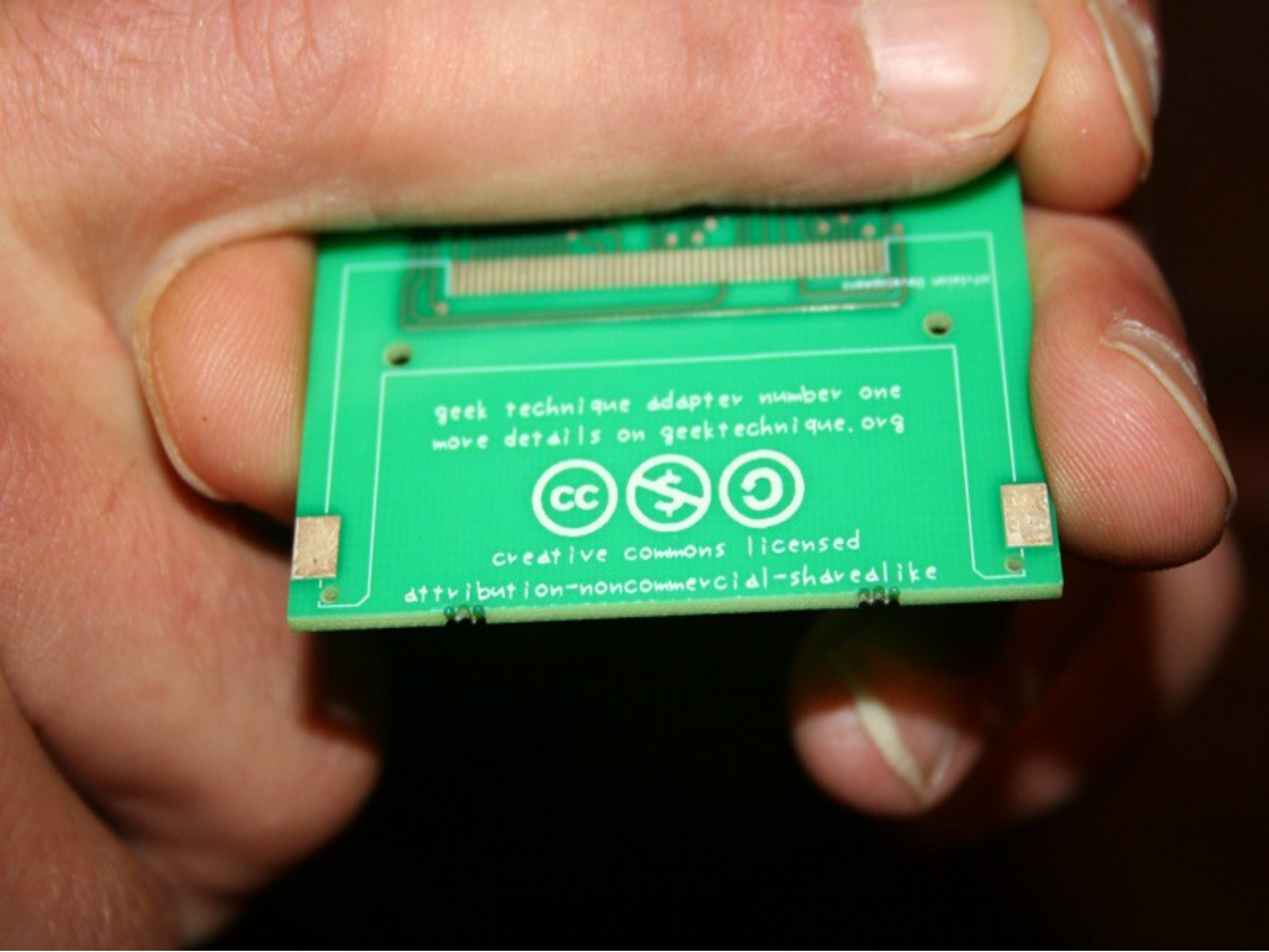
We Should Make

Do The Work

But, What About


geek technique adapter number one
more details on geektechnique.org
creative commons licensed
attribution-noncommercial-sharealike

To Test

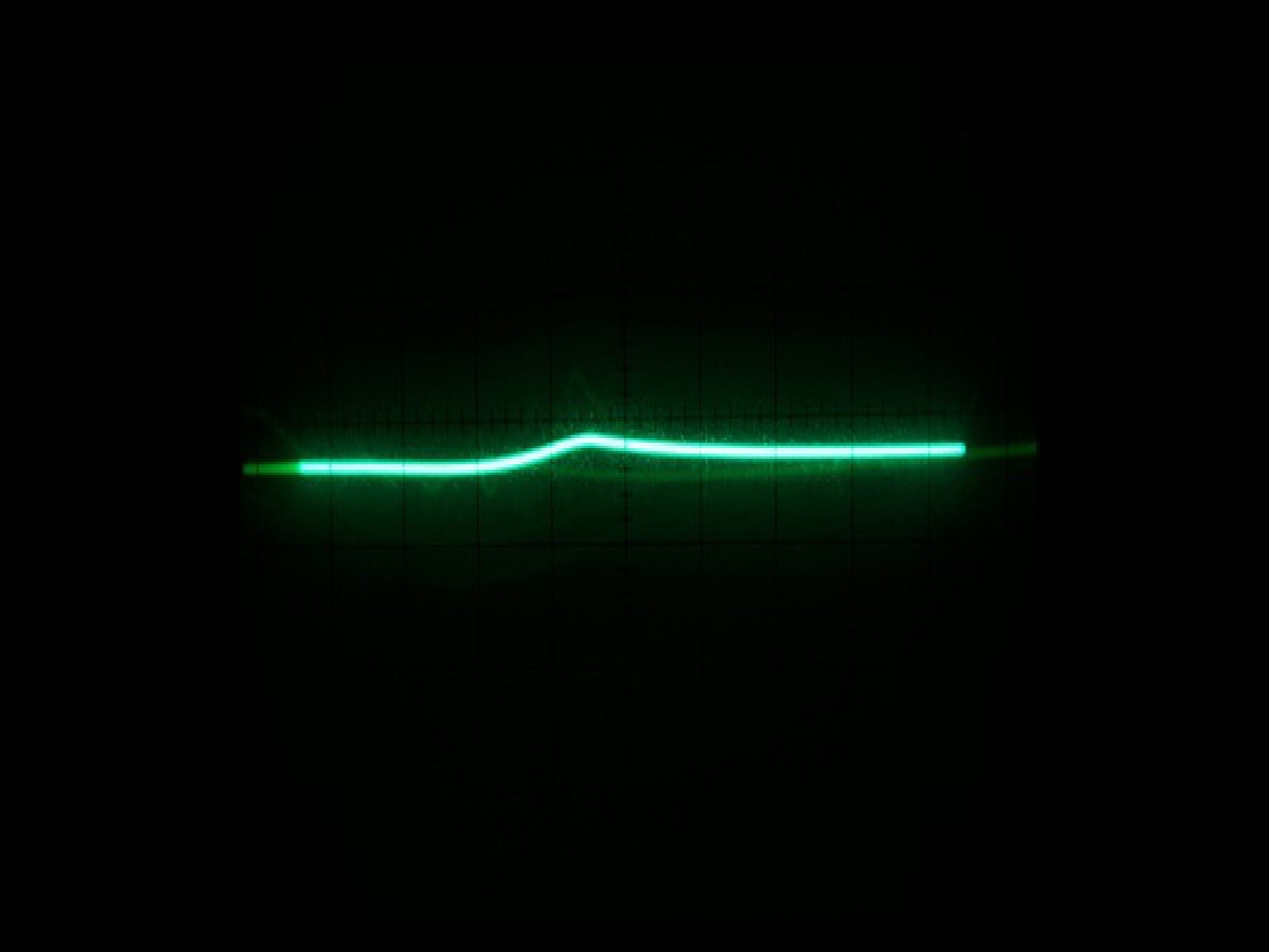

geek technique adapter number one
more details on geektechnique.org
creative commons licensed
attribution-noncommercial-sharealike

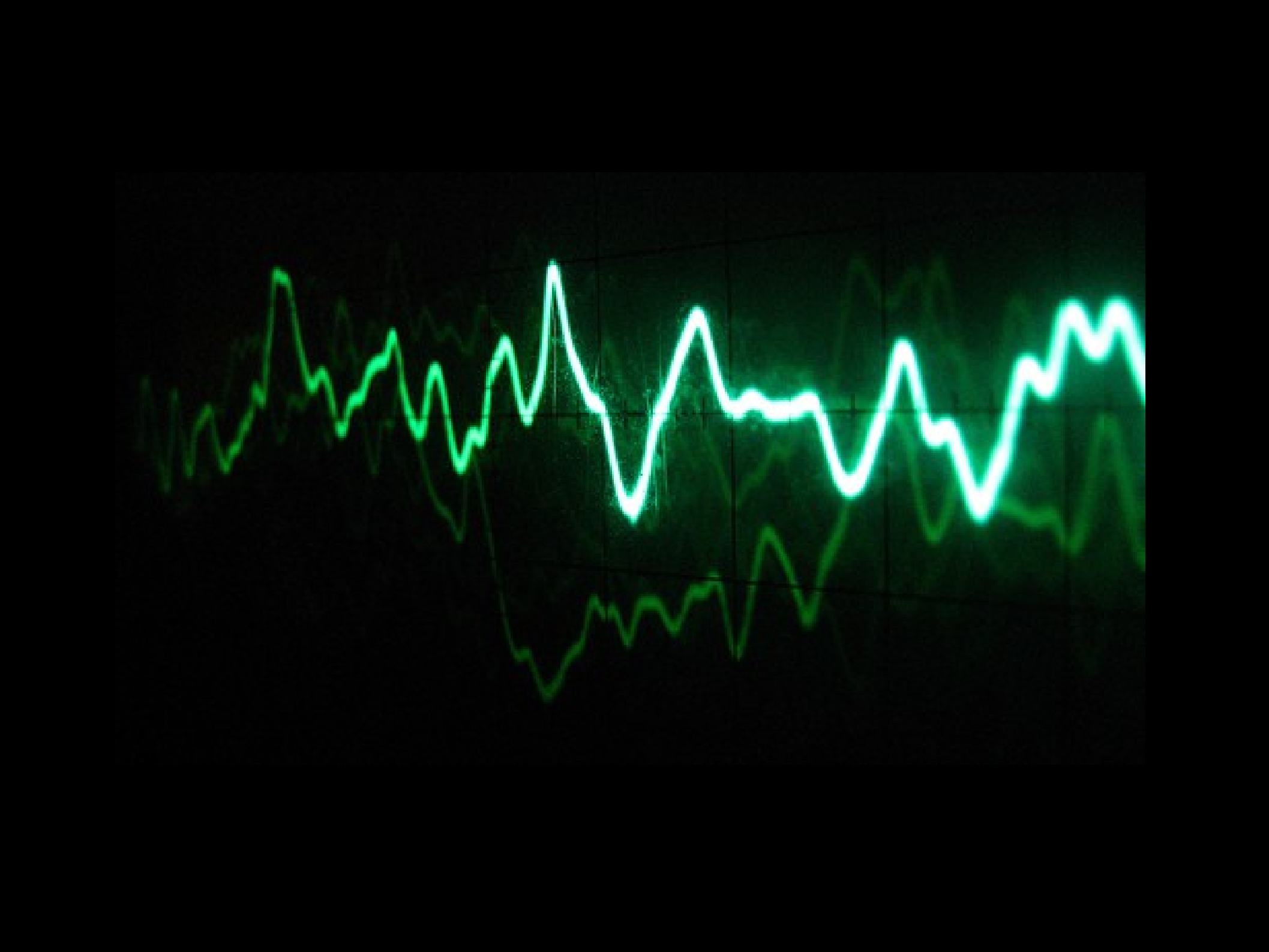
We Need To

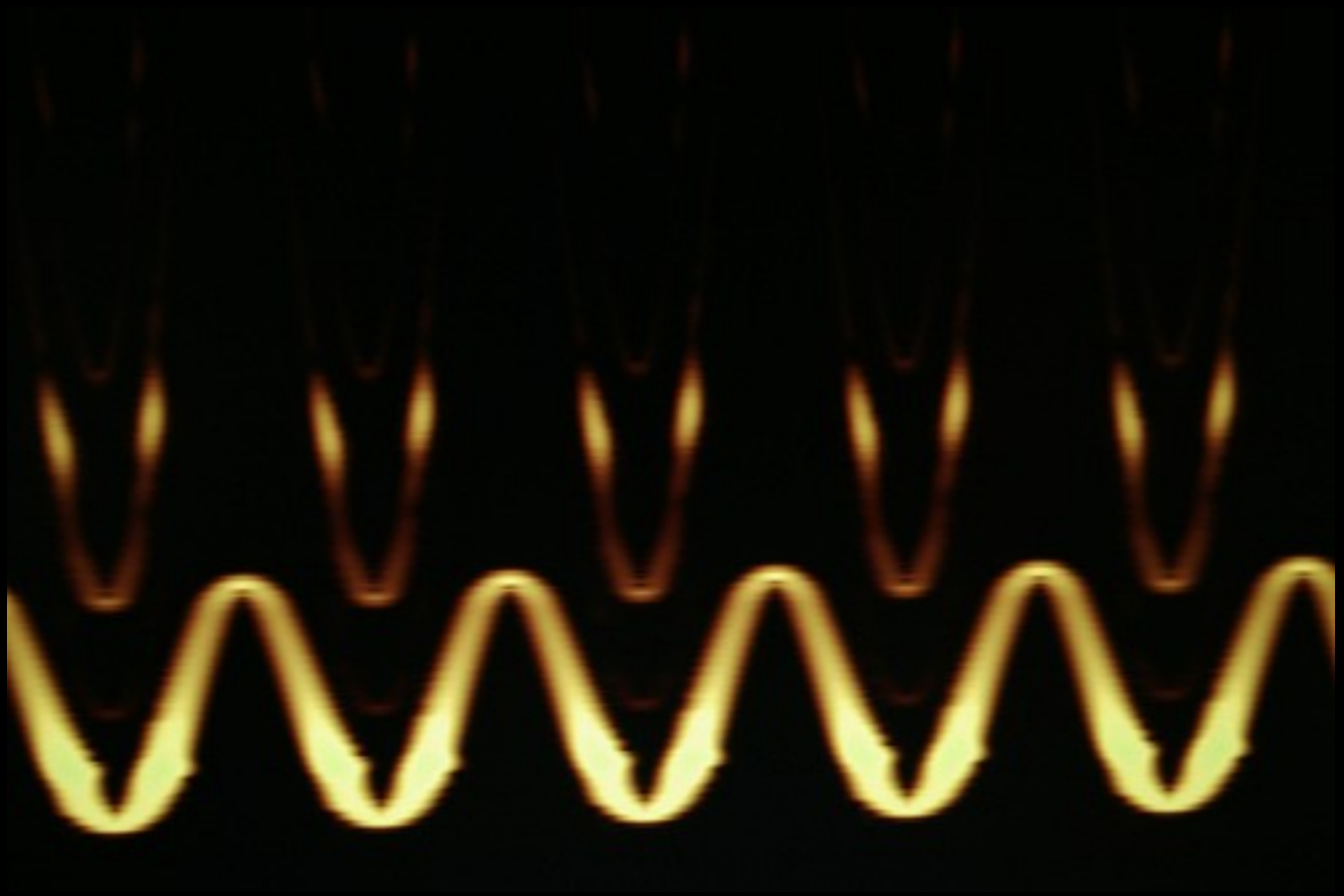
We Need To
Drive Inputs

We Need To
Drive Inputs
Measure Outputs

These Signals Could Be



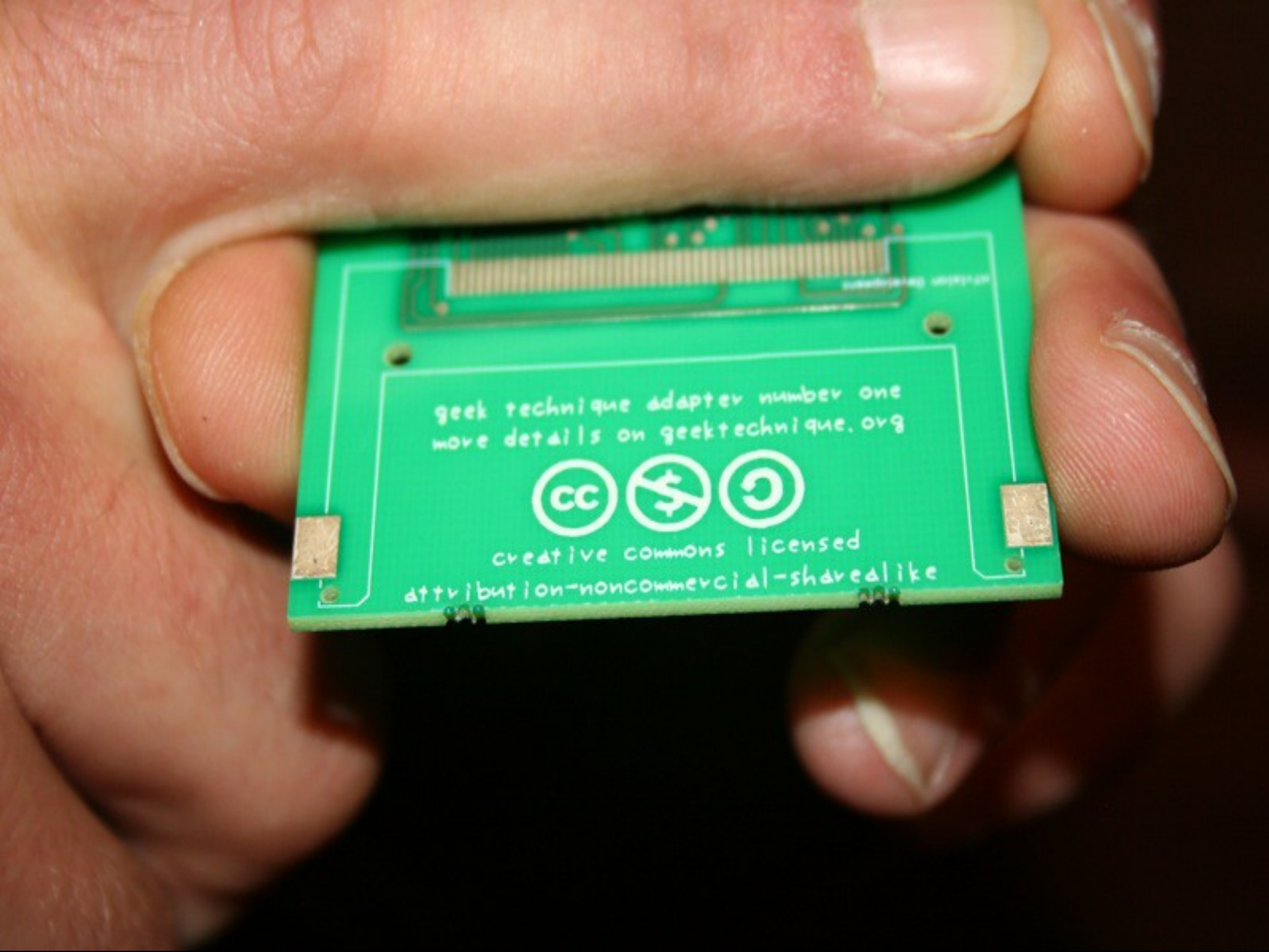


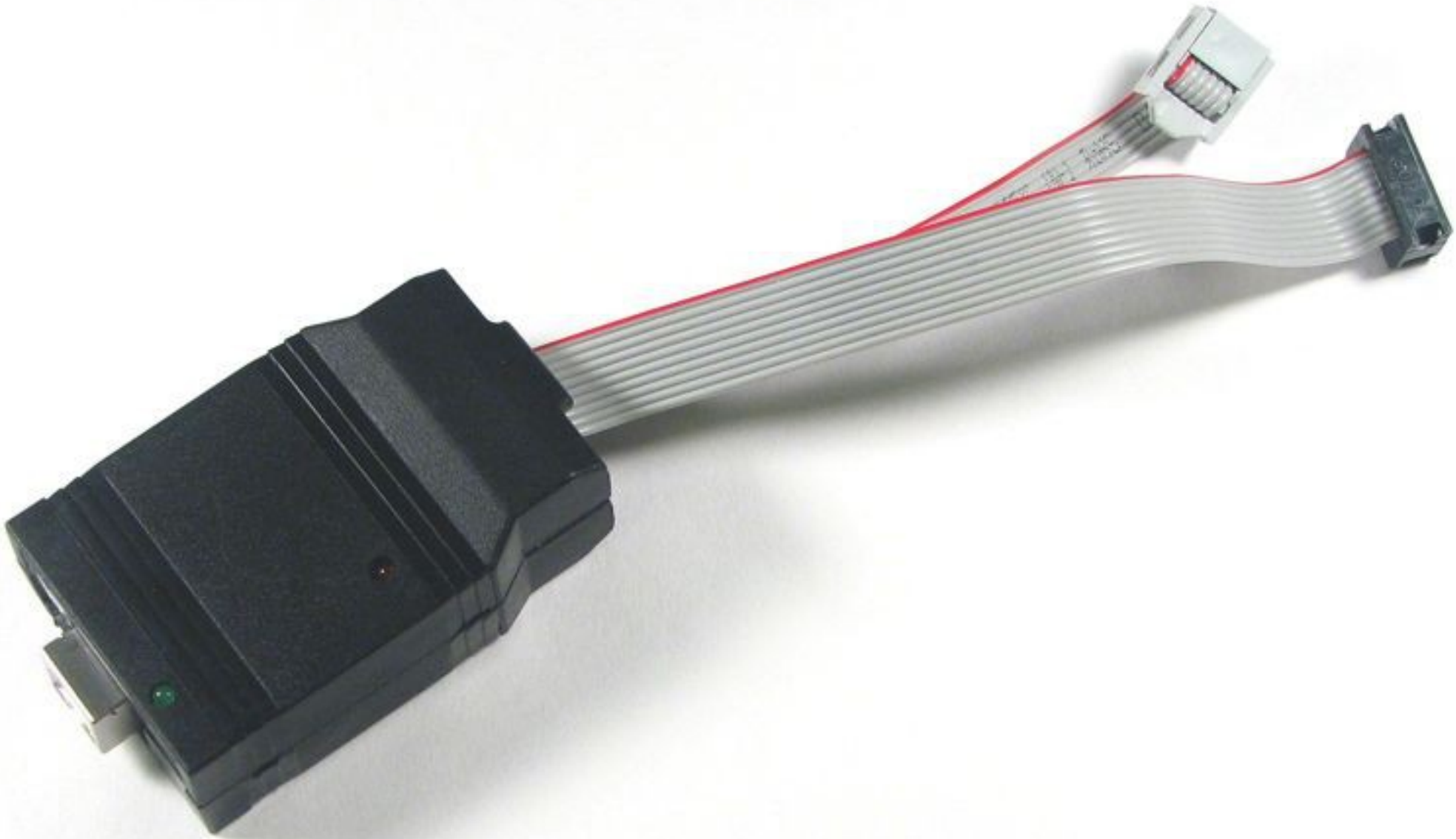


“Hardware In the Loop”


hola
DELL
Cool-Breeze

Now We Have


geek technique adapter number one
more details on geektechnique.org
creative commons licensed
attribution-noncommercial-sharealike



But We Need

Specify Tests

Language?

It Should Be

Easy to write

Simple to install

Low Cost,
Preferably Free

Any scripting language should do the trick

Maybe

Maybe

Maybe even

we chose

why?

intuitive

example

Send Serial Command

“version?”

5 times

```
5.times do
    send_serial “version?”
end
```

Maybe

Get Response?

```
text = serial_readline
```

Getting it Isn't Enough?

Check The Format

Regular Expressions

Version

v1.0.542

v1.0.542

v1.0.542

v1.0.542

```
assert_match
  /v1\.\d\.\d+/,
  text
```

v1.0.542

```
assert_match
  /v1\.\d\.\d+/,
  text
```

v1.0.542

```
assert_match
  /v1\.\d\.\d+/,
  text
```

v1.0.542

```
assert_match
  /v1\.\d\.\d+/,
  text
```

What If

v1.45.886

v1.45.886

```
assert_match
  /v1\.\d\.\d+/,
  text
```

v1.2.999665533

v1.2.999665533

```
assert_match
  /v1\.\d\.\d+/,
  text
```

test?

```
5.times do
    send_serial “version?”
    text = serial_readline
    assert_match
        /v1\.\d\.\d+/,
        text
end
```

```
5.times do
    send_serial “version?”
    text = serial_readline
    assert_match
        /v1\.\d\.\d+/,
        serial_readline
end
```

send_serial... really?

method

driver

Test Framework

Systir

System Testing In Ruby

not

System Testing Of Ruby

System Testing In Ruby

What Do Test Frameworks Do?

Manages Test Suite

instead of

```
rake test:run:verify_usart_temp
rake test:run:verify_usart_1per_sec
rake test:run:verify_correct_temp
rake test:run:verify_version
```

write

```
rake test:system
```

What else?

We Can Write Tests

Domain Specific Language

instead of

```
readval = “”
port.open(“com3”) do |p|
   begin
      while !port.eol
         readval << port.in_char
      end
   rescue
      flunk “error during read”
   end
end
assert readval =~ /\d+\.\d{2}/
```

write

```
verify_serial /\d+\.\d{2}/
```

or

verify_serial_format

What About The Interface Box?


INSTANT
KRAZY®
GLUE
BRUSH-ON


Informationsberufe

.dll's, .so's, .lib's

We could write our ruby interface

manually

or

SWIG

Which Takes

Libraries In

Wrappers Out

Which is Way Cool

But Not What This Presentation is About

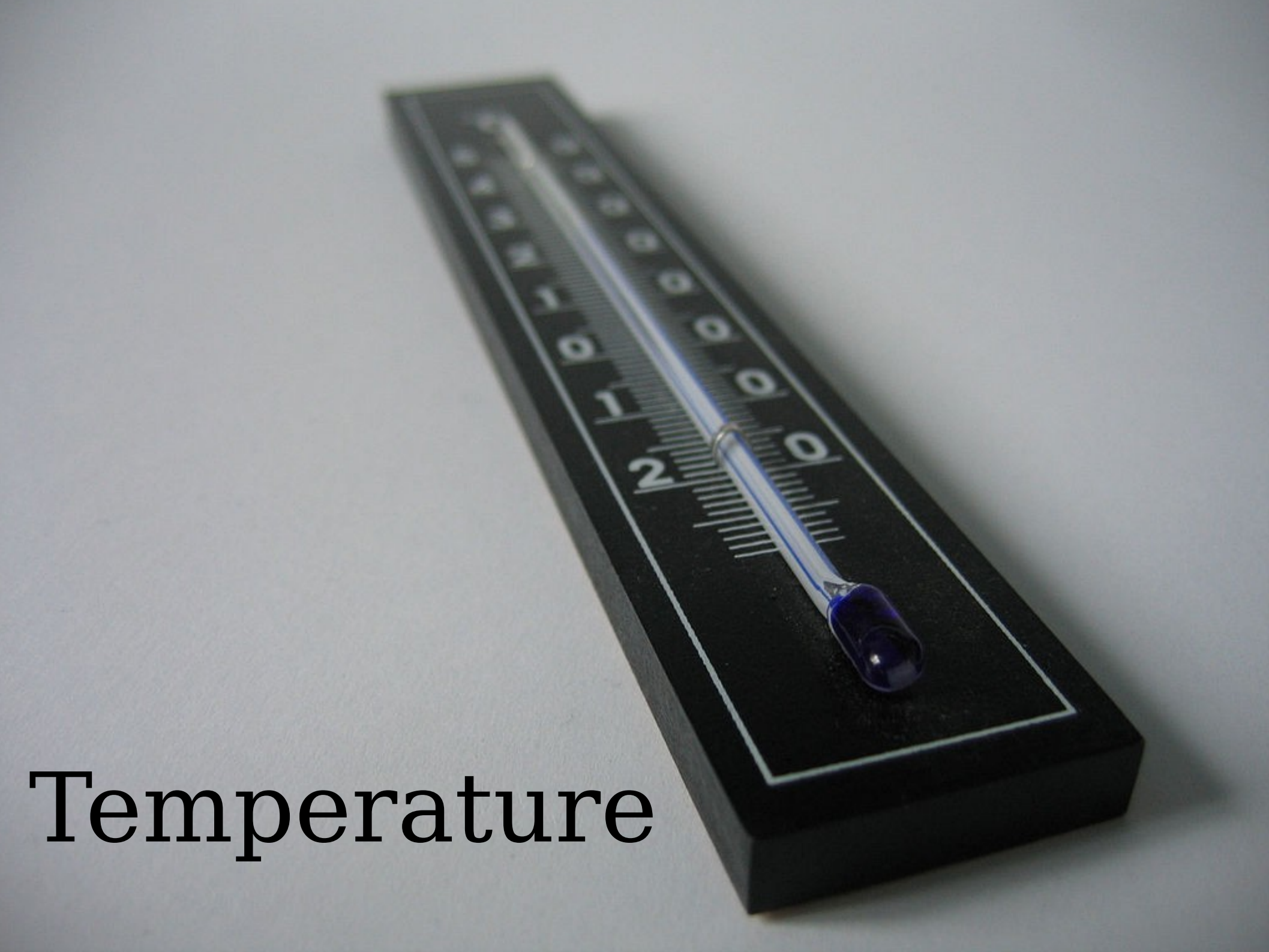
What This Presentation is About

System Testing

So


Let's Write One

First Feature:



Temperature

Will be Sent

Formatted

A Number under 1000

and

Exactly 1 Digit after the Decimal Point

Followed by a Space and then a 'C'

So First?

We need to get Data

temp = serial_readline

then, check the format

Regular Expression

Number < 1000

//

Number < 1000

/\d+/

/\d\d\d/

Number < 1000

/\d{1,3}/

decimal pt and 1 digit

/\d{1,3}/

decimal pt and 1 digit

/\d{1,3}\.\d/

a space, then a C

/\d{1,3}\.\d/

a space, then a C
/\d{1,3}\.\d\sC/

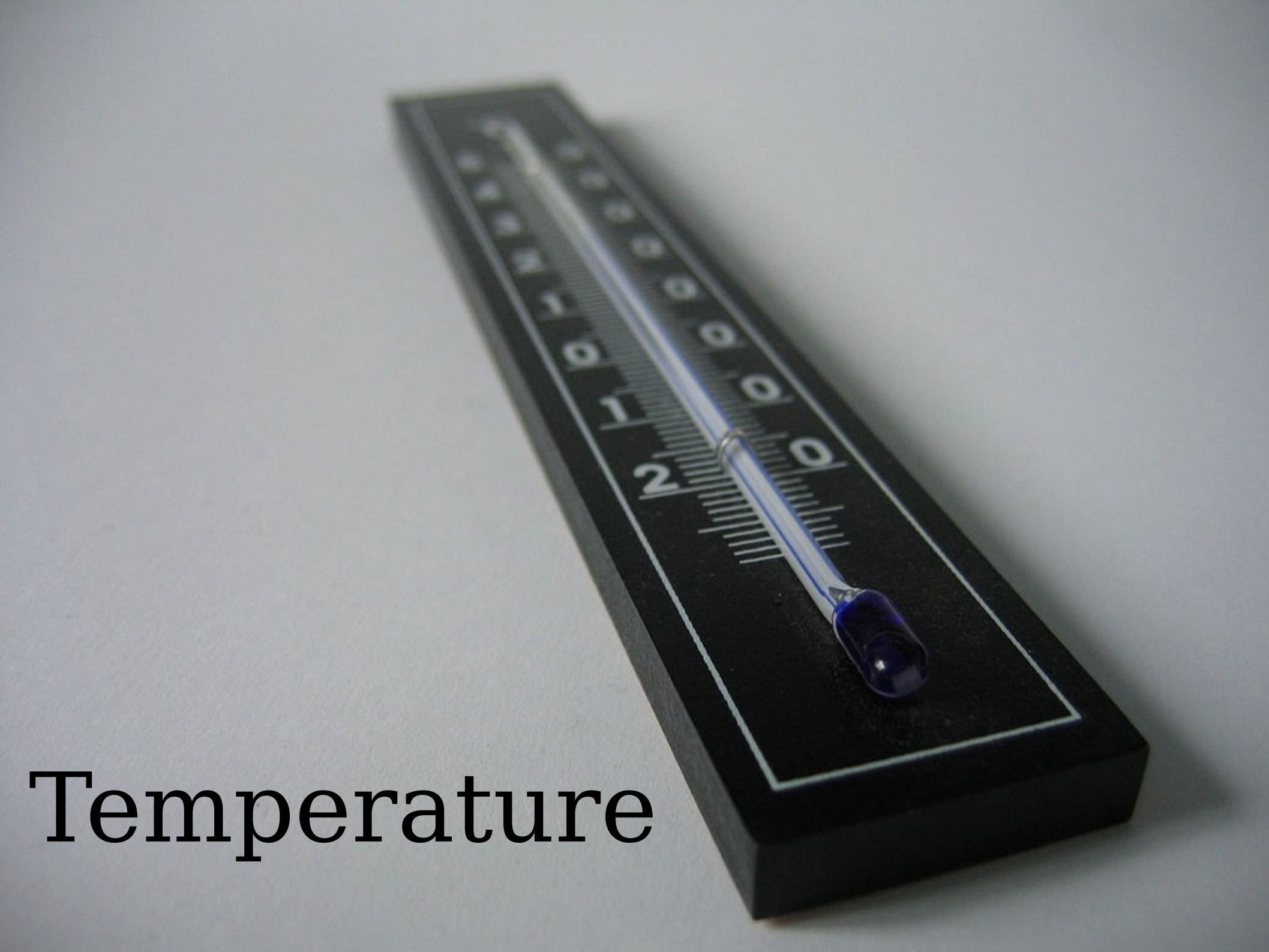
Assert that this Matches

```
temp = serial_readline
assert_match
    /\d{1,3}\.\d\sC/,
    temp
```

```
temp = serial_readline
assert_match
    /\d{1,3}\.\d\sC/,
    temp,
    “Can't You Try Harder?”
```

Does It Work?

Second Feature:


Temperature

streamed

once per second

within 100 msec

we already can read the temperature

but 1 temp per second?

Time

Time.now

Time.now
float value in seconds

```
start_time = Time.now
```

then get another
temperature line

```
text = serial_readline
```

Then grab endtime

```
stop_time = Time.now
```

make sure the timing was within spec

```
assert_in_delta
    1.0,
    (stop_time-start_time),
    0.1
```

```
assert_in_delta
    1.0,
    (stop_time-start_time),
    0.1
```

```
assert_in_delta
    1.0,
    (stop_time-start_time),
    0.1
```

```
assert_in_delta
    1.0,
    (stop_time-start_time),
    0.1
```

```
assert_in_delta
    1.0,
    (stop_time-start_time),
    0.1
```

```
assert_in_delta
    1.0,
    (stop_time-start_time),
    0.1,
    "You Fool! Time Wrong"
```

is once enough?

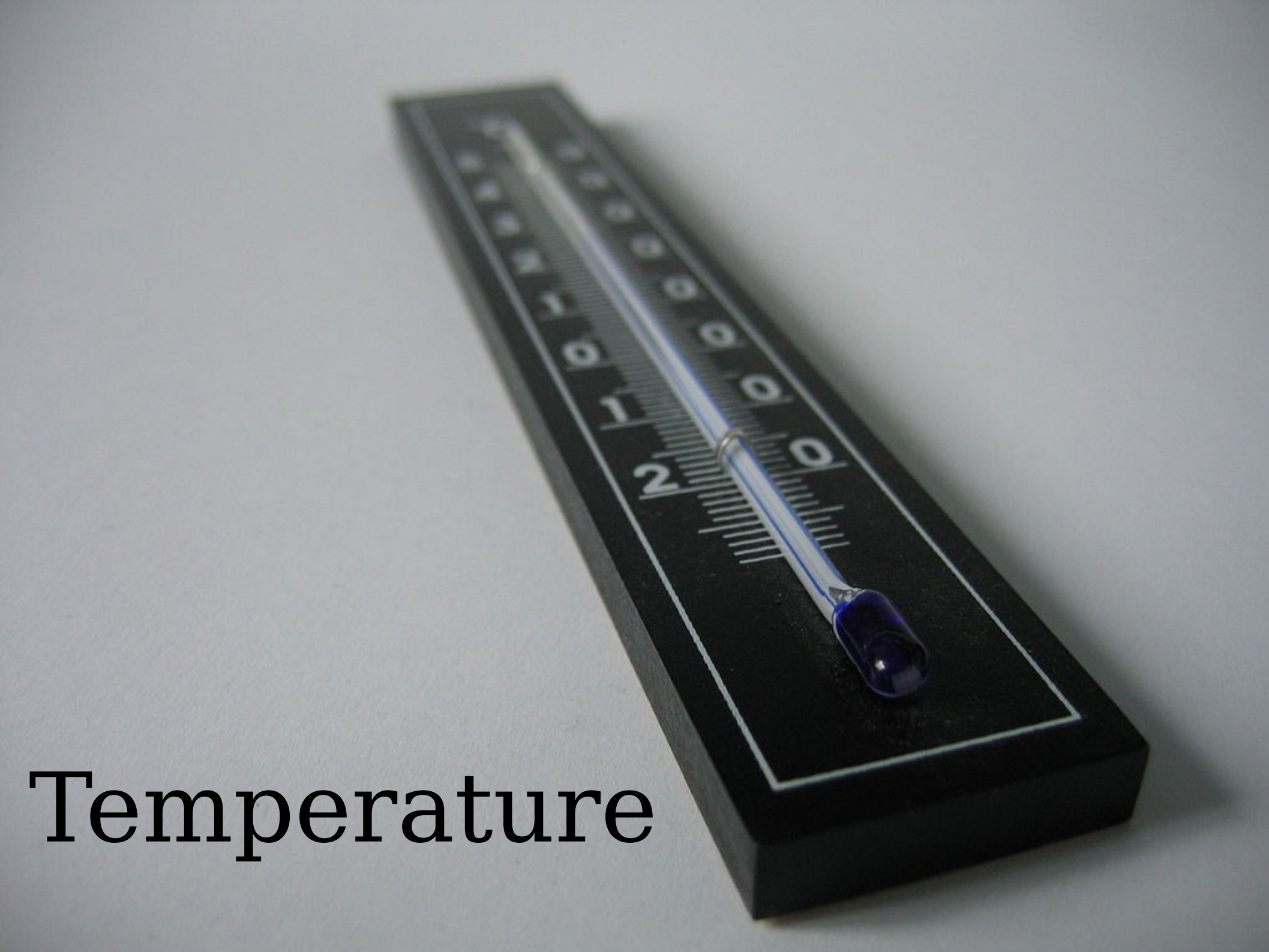
```
text = serial_readline
5.times do
   start_time = Time.now
   text = serial_readline
   stop_time = Time.now
   assert_in_delta
      1.0,
      (stop_time-start_time),
      0.1,
      “Sorry, Out Of Time”
end
```

Does It Work?

Let's Fix It

Did We Fix It?

One More Feature:


Temperature

Should Be Within

2 degrees C

Of Temperature Input

So We Need

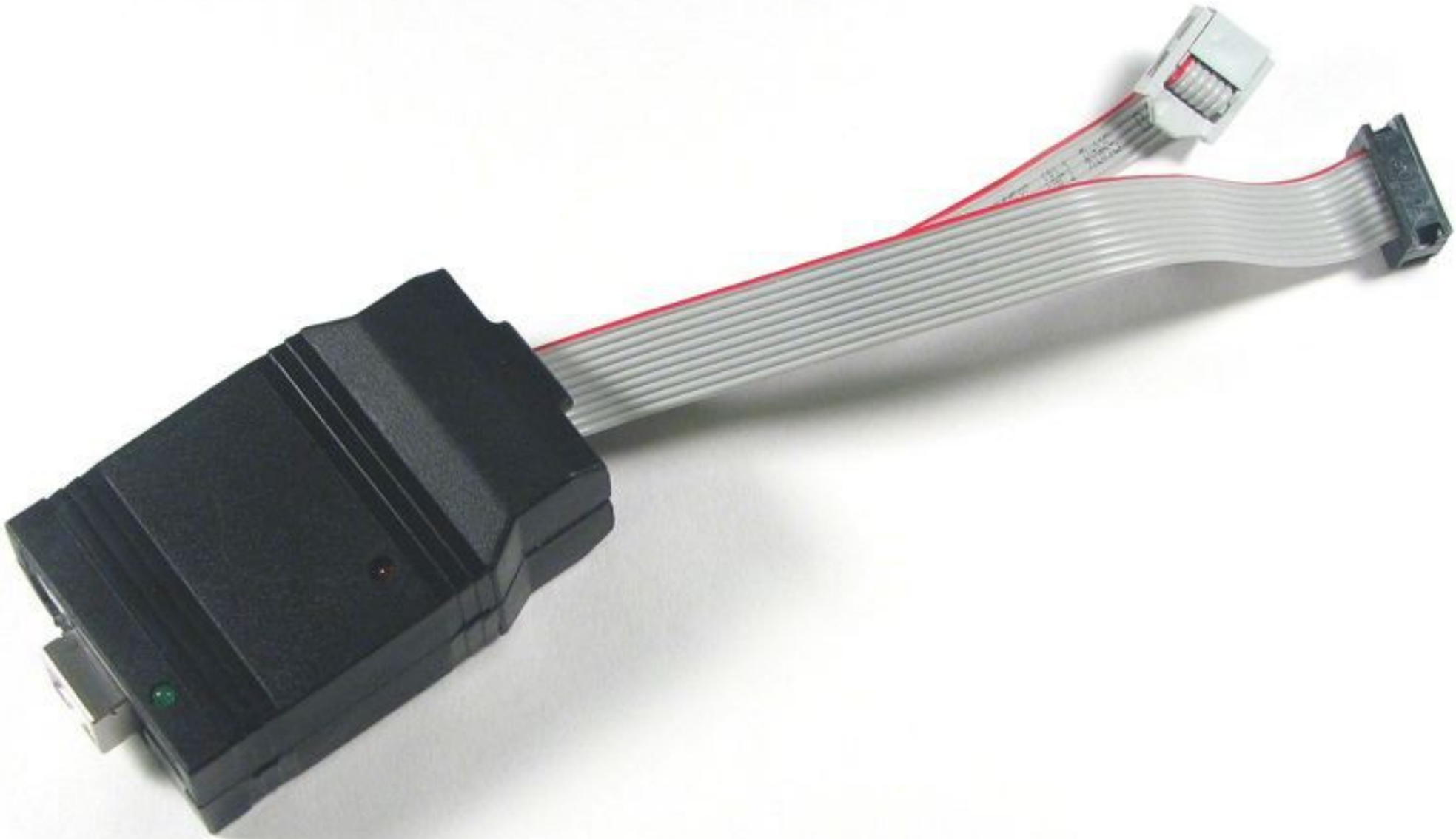

2000

Temperature Sensors Output Voltages

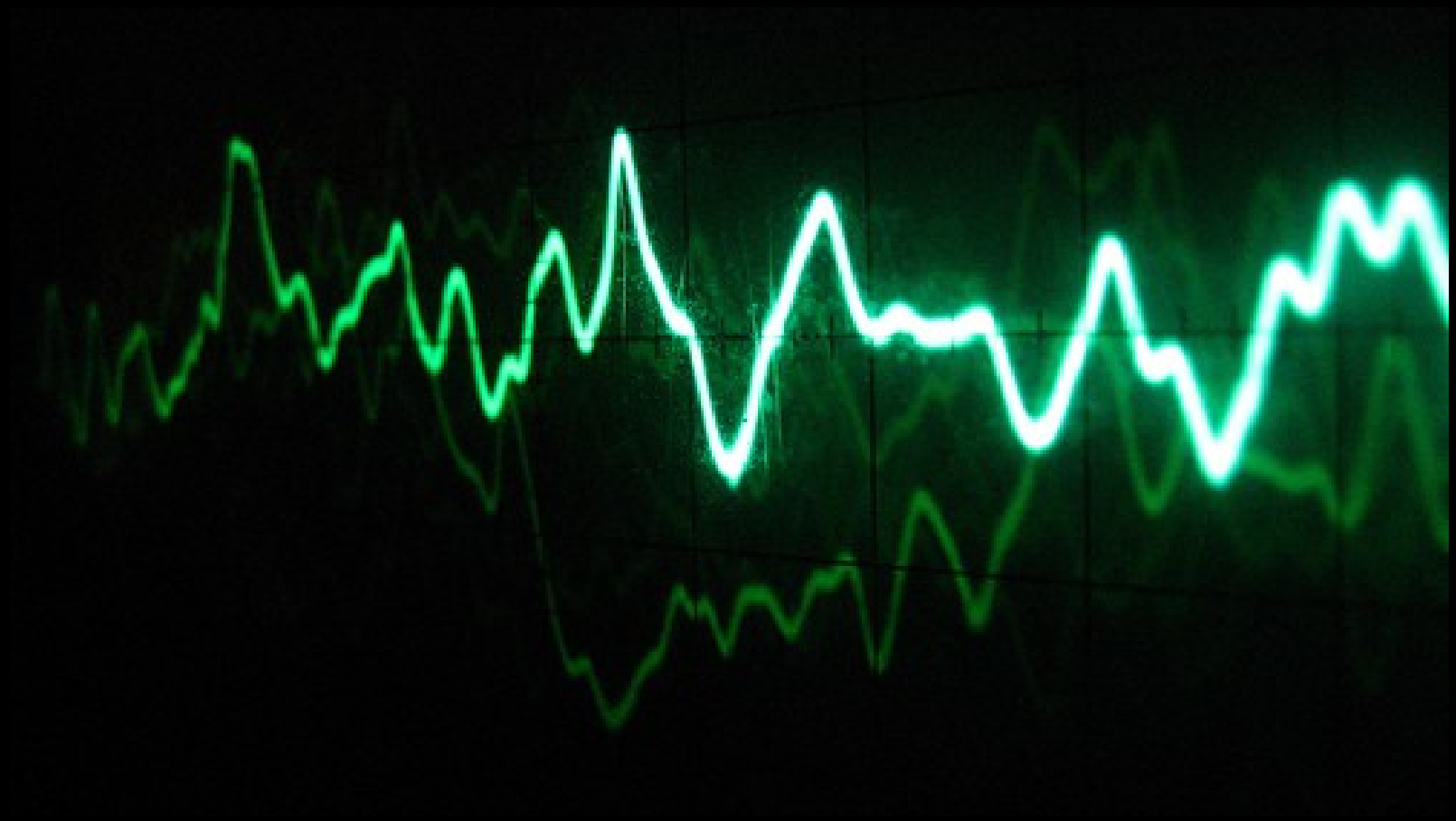
ADC only Care about Voltages

So,

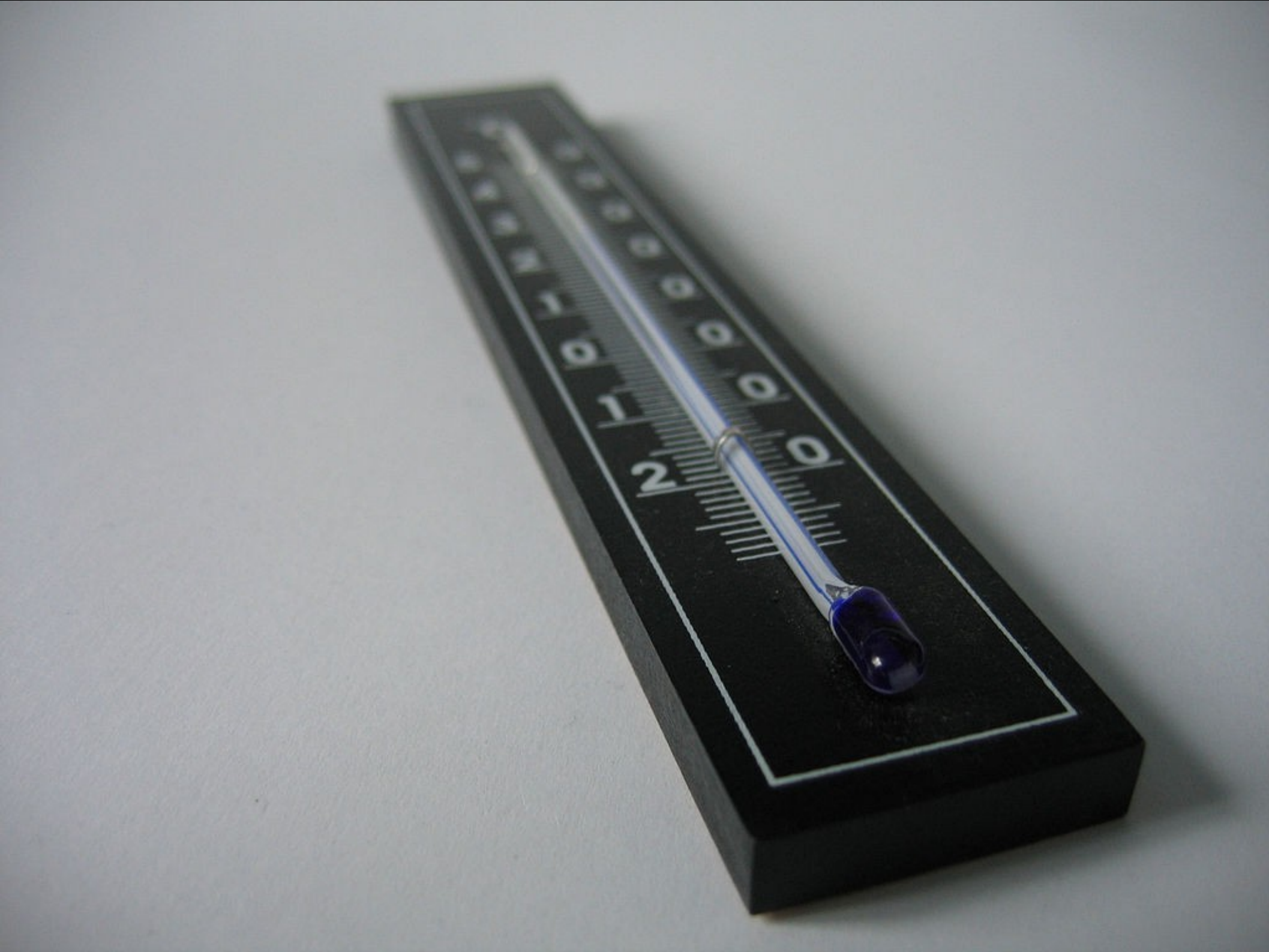
We Use



To Output



And Pretend It's



So, Grab Our Wrapper

(which isn't what this presentation is about)

Figure Out

How

Set Voltage Outputs

```
@minilab.write_analog
    pin,
    voltage
```

pin == whichever we choose

$$0 \leq \text{voltage} \leq 3$$

add a method to our driver

```
def set_temp_voltage(voltage)
    @minilab.write_analog
        TEMP_PIN, voltage
end
```

what about

$$0 \leq \text{voltage} \leq 3$$

```
def set_temp_voltage(voltage)
    raise “Idiot! Invalid Voltage”
        unless ((voltage >= 0)
            &&  (voltage <= 3))
    @minilab.write_analog
        TEMP_PIN, voltage
end
```

and settle time?

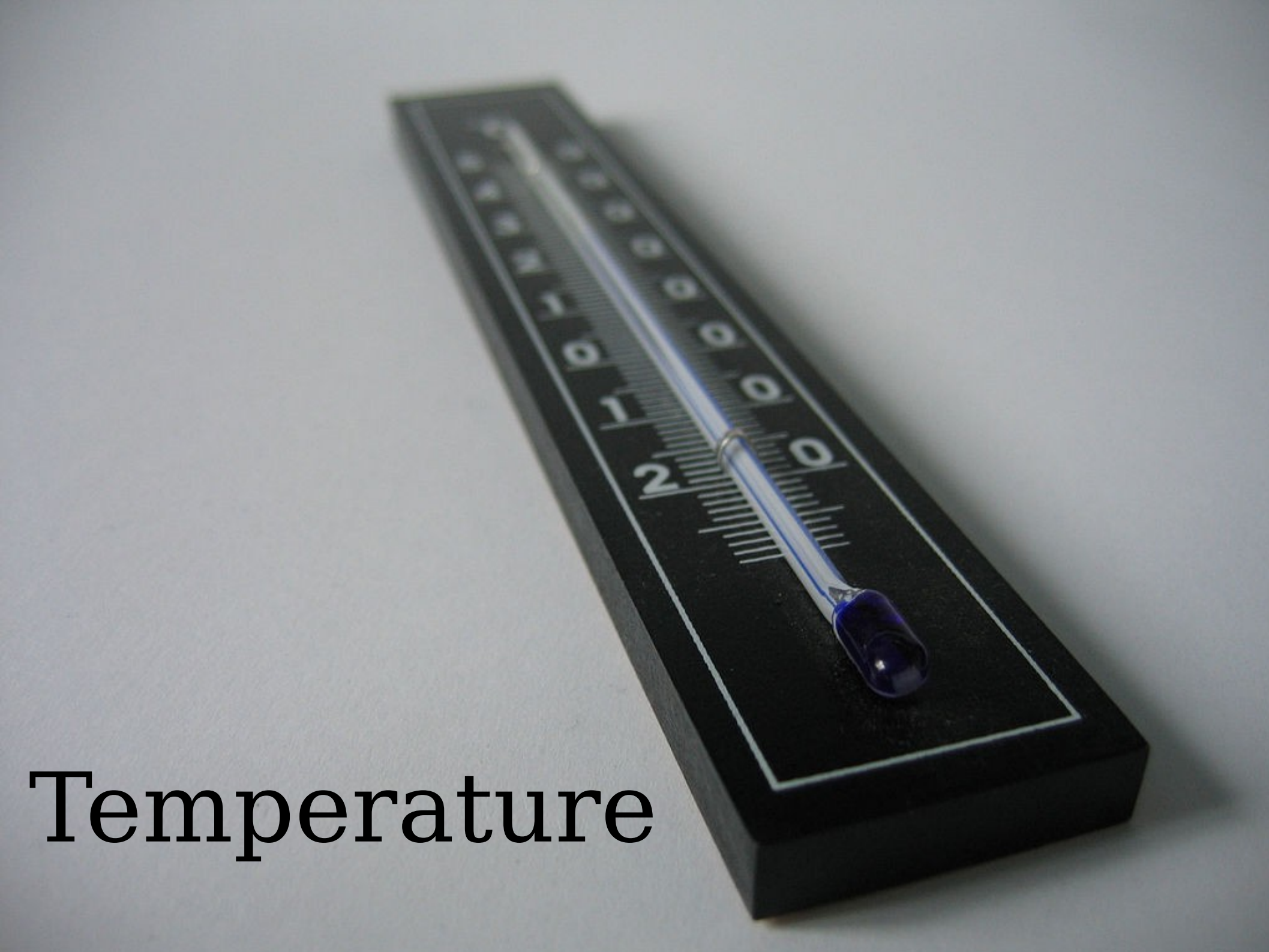
maybe 100 msec?

```
def set_temp_voltage(voltage)
    raise “Idiot! Invalid Voltage”
        unless ((voltage >= 0)
            &&  (voltage <= 3))
    @minilab.write_analog
        TEMP_PIN, voltage
    sleep 0.1
end
```


So
That's Peachy

But

Didn't We Want


Temperature

Another Driver Method

```
def set_temp_in_deg_C(temp)
   voltage =
      calc_volt_from_temp(temp)
   set_temp_voltage(voltage)
end
```

calc_volt_from_temp?

its “just math”

moving on...

can now use

set_temp_in_deg_C

To Set Temp

And

We Can Read Temp

Using serial_readline

As Text

As Text

Convert To Number

```
temp = text.to_f
```

problem?

88.2 C

88.2 C
trailing text is ignored!


So We're Ducky

```
set_temp_in_C(10.0)
text = serial_readline
temp = text.to_f
```

```
set_temp_in_C(10.0)
temp = serial_readline.to_f
```

```
set_temp_in_C(10.0)
temp = serial_readline.to_f
assert_in_delta
   10.0,
   temp,
   2.0,
   “Hey, it's not 10 degrees!”
```

should check more than one temp

we could

```
set_temp_in_C(25.0)
temp = serial_readline.to_f
assert_in_delta
   25.0,
   temp,
   2.0,
   “It's not 25 degrees.”
```

```
set_temp_in_C(35.0)
temp = serial_readline.to_f
assert_in_delta
   35.0,
   temp,
   2.0,
   “It's not 35 degrees!”
```

```
set_temp_in_C(40.0)
temp = serial_readline.to_f
assert_in_delta
   40.0,
   temp,
   2.0,
   “It's not 40.0 either!”
```

Silly?

Loop

```
setpts = [10.0, 25.0, 35.0, 40.0]
setpts.each do |setpt|
   set_temp_in_C(setpt)
   temp = serial_readline.to_f
   assert_in_delta
      setpt,
      temp,
      2.0,
      “Temp is Definitely Wrong”
end
```

What About Settle Time?

Insert Delay

And

Flush Serial Port!

```
setpts = [10.0, 25.0, 35.0, 40.0]
setpts.each do |setpt|
   set_temp_in_C(setpt)
   sleep 1.0
   serial_flush
   temp = serial_readline.to_f
   assert_in_delta
      setpt, temp, 2.0,
      “That Temp Just Ain't Right”
end
```

Does It Work?

So We've Covered

Three Features

Are We Done?

Larger Systems

Potential Problems

Interdependencies

Commands May Break Other Commands

Only The First Request Might Work

The System May Lock Up After Two Minutes

Or After 127 Commands

We Can't Predict All Problems

Is There Anything We Can Do?

Maybe

Run Tests For A Block Of Time

Like
For
Four
Hours

In Random Order


Run It
Every Night

Eventually Issues Fall Out

So We Extend Systir

Mixtir

Mixed Testing In Ruby

Differences

Tests Run Multiple Times

Tests Randomly Ordered

Suite Run For Predetermined Time

Tests Don't Start With A “Clean Slate”

Also

For Repeatability

Random Seed

Entered Manually

Rerunning Sequence

Ideally

Keep Tests Small

Instead Of

```
setpts = [10.0, 25.0, 35.0, 40.0]
setpts.each do |setpt|
   set_temp_in_C(setpt)
   sleep 1.0
   serial_flush
   temp = serial_readline.to_f
   assert_in_delta
      setpt, temp, 2.0,
      “I'm Such A Failure!”
end
```

Just Include

```
setpts = [10.0, 25.0, 35.0, 40.0]
setpts.each do |setpt|
  setpt = 10.0 + rand(30)
  set_temp_in_C(setpt)
  sleep 1.0
  serial_flush
  temp = serial_readline.to_f
  assert_in_delta
    setpt, temp, 2.0,
    “It's Scalding/Freezing!”
```

Instead Of

```
text = serial_readline
5.times do
   start_time = Time.now
   text = serial_readline
   stop_time = Time.now
   assert_in_delta
      1.0,
      (stop_time-start_time),
      0.1,
      “Yikes! The Time Was Wrong”
end
```

Just Include

```
text = serial_readline
5.times do
   start_time = Time.now
   text = serial_readline
   stop_time = Time.now
   assert_in_delta
      1.0,
      (stop_time-start_time),
      0.1,
      “Time Out Of Spec”
end
```

Does It Accomplish Anything?

We'll Use A Known Random Seed

Otherwise We Could Be Waiting A While

Does It Work?

So What Went Wrong?

Check Against Spec

Fix Test Or Code

Retry With Seed

Does It Work?

So

We've Caught A Couple Of Bugs

We've Also Prevented Future Bugs

Obviously

If You're Doing Agile

This Fits

But

System Testing

Also Useful

For All Embedded Software Developers

So That's It.

362 Slides

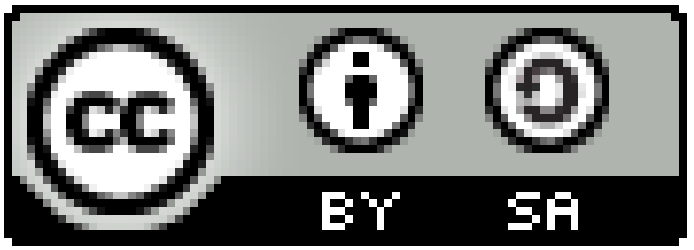
6 System Tests

Maybe Even

A New Language

In Less Than 1 Hour

Questions?



Images By (*thanks!*):

Turd – Bart Rox
Matrix Screens - “Scumfrog”
Scopes (3) – Mikael Altemark
Interface Box – Windell Oskay
Wrap - “XCtnx”
Moon - “Kevin”

Clock – Christina Castro
Mac Mini – Richard Thomas
Pearl - “Blue Heron Beauty”
Belt - “Everchanging Girl”
Wood Stove – Bob Travis
Glue – Jenny “mennyj”
Pencil - Balakov

Thermometer – Marek Papala
Version – Travis Forden
Ruby - “Afternoon Sunlight”
Statistics Graph – P. Neff
Manual Testing – Tim Conkery
UUT – Deborah Schultz
Duck – “LavenderLady”

Python – Ian Chien
HIL – Eduardo Guerrero
Traffic Lights (2) - “flrnt”
Peach – Rick Harris
Library - “Library Mistress”