
chip.fail

Who are we?

Thomas Roth
@stacksmashing

Josh Datko
@cryptotx

Dmitry Nedospasov
@nedos

cryptotronix

leveldown security

Special thanks: Colin O'Flynn

- Hardware blockchain expert
- CircuitCellar Author
- Builder of the ChipWhisperer

Takeaways

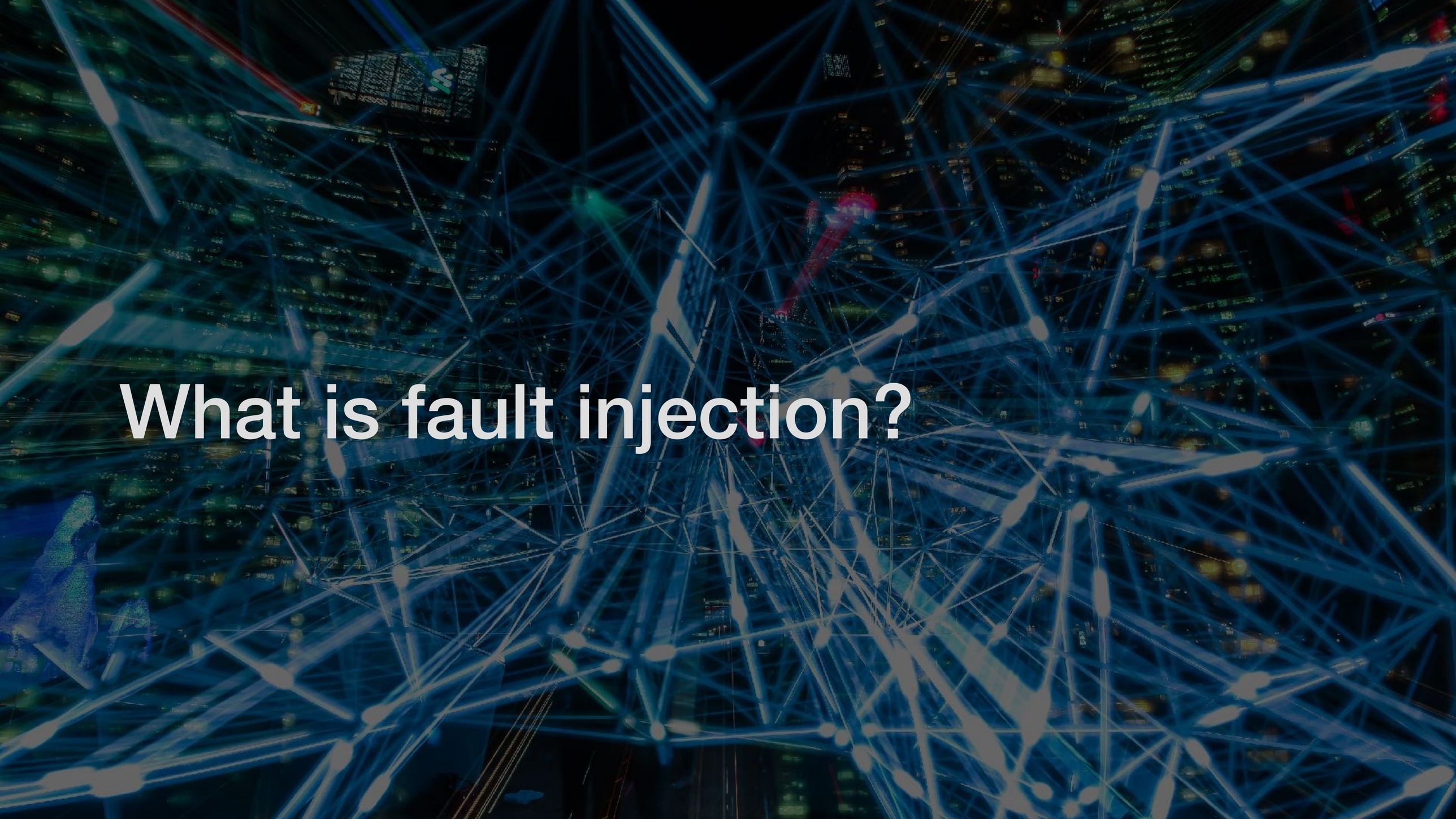
- Glitching is easier than you think
- You need to know very little about hardware (... or software) to attack a chip
- It can become part of your Software- and Hardware Development Lifecycle

Why are we here?

- Almost never see chips as part of the threat model
- Lot's of people think glitching/fault injection is magic
- We want to make glitching more accessible

Why is this getting important?

- Java methodology: Hack once break in everywhere
- Secure IoT devices are essential for a lot of use-cases
- The "secure" chips in them are not as secure as you would want them to be



What is fault injection?


What is ~~fault injection~~?
Glitching

Glitching / Fault injection

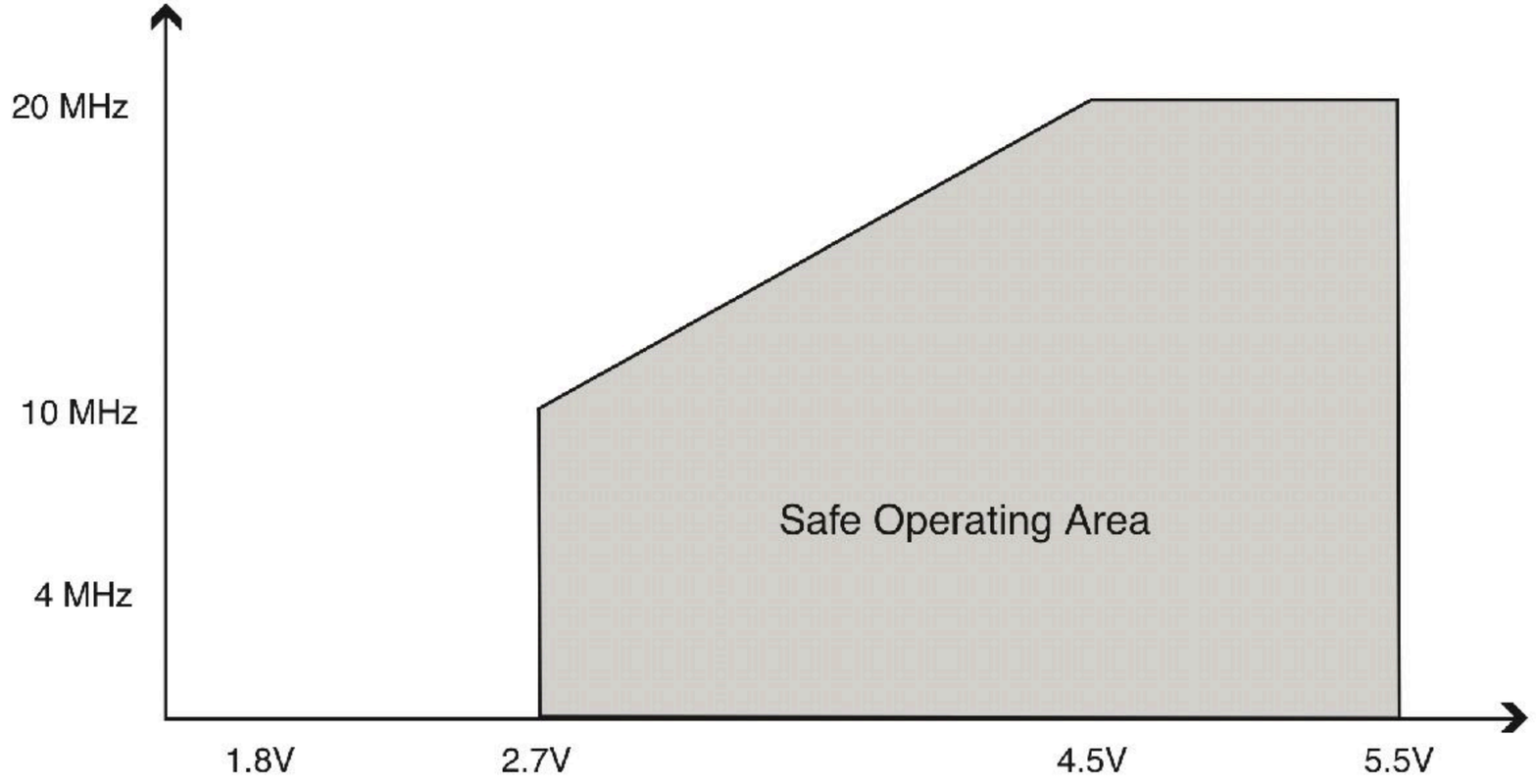
- Introduce faults into a chip
  - Cut the power for a very short amount of time
  - Change the period of the clock signal
  - Inject electro magnetic shocks (aka point a tazer at your chip)

Voltage glitching

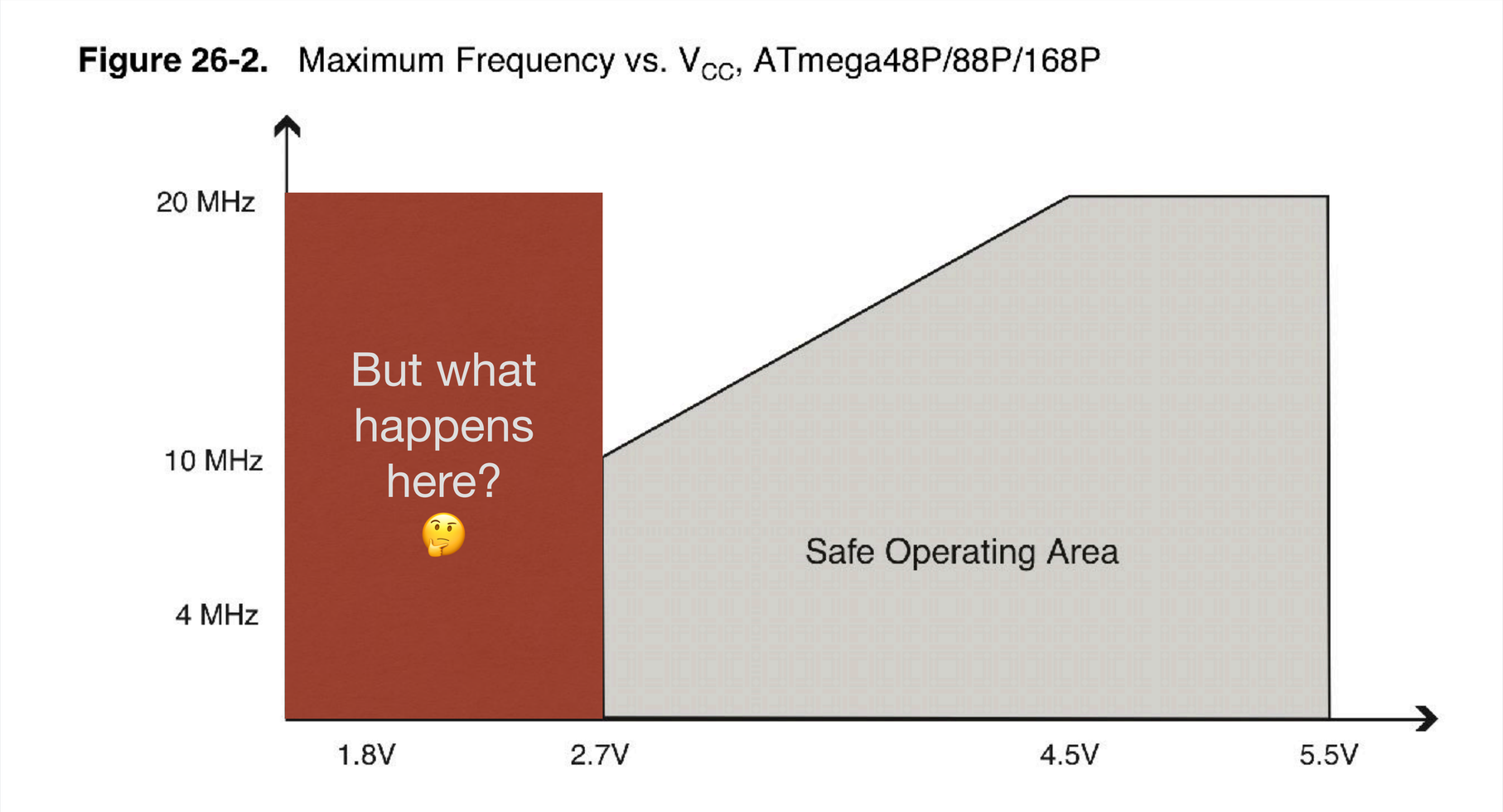
- Cut the power to the chip for a **very** short amount of time
- At a **very** precisely timed moment
- To cause undefined behaviour

Glitching / Fault injection

Figure 26-2. Maximum Frequency vs. V_{CC}, ATmega48P/88P/168P



Glitching / Fault injection

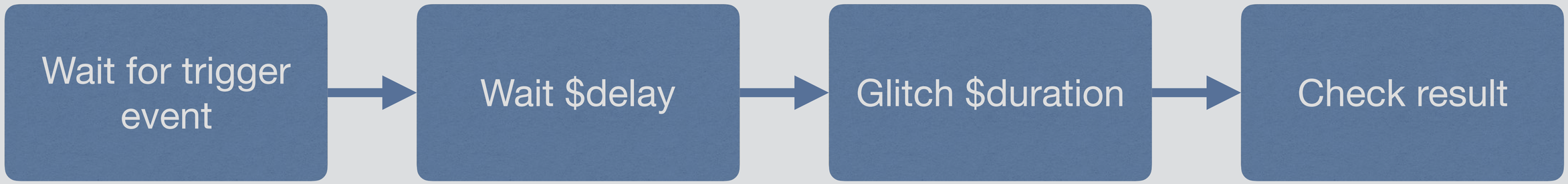


Glitching / Fault injection

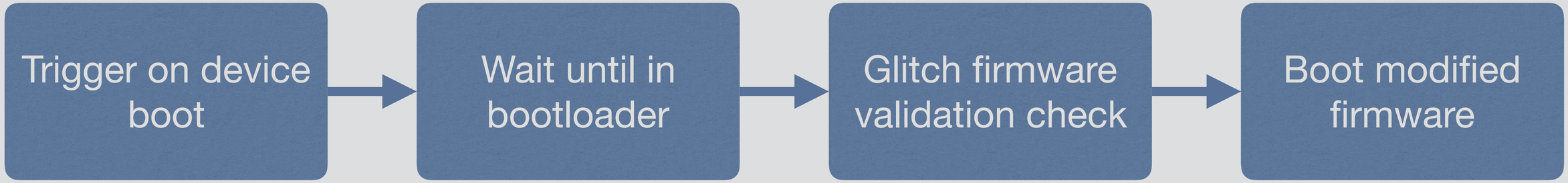
cryptotronix

leveldown security

Glitching


Wait for trigger event
Wait $delay
Glitch $duration
Check result

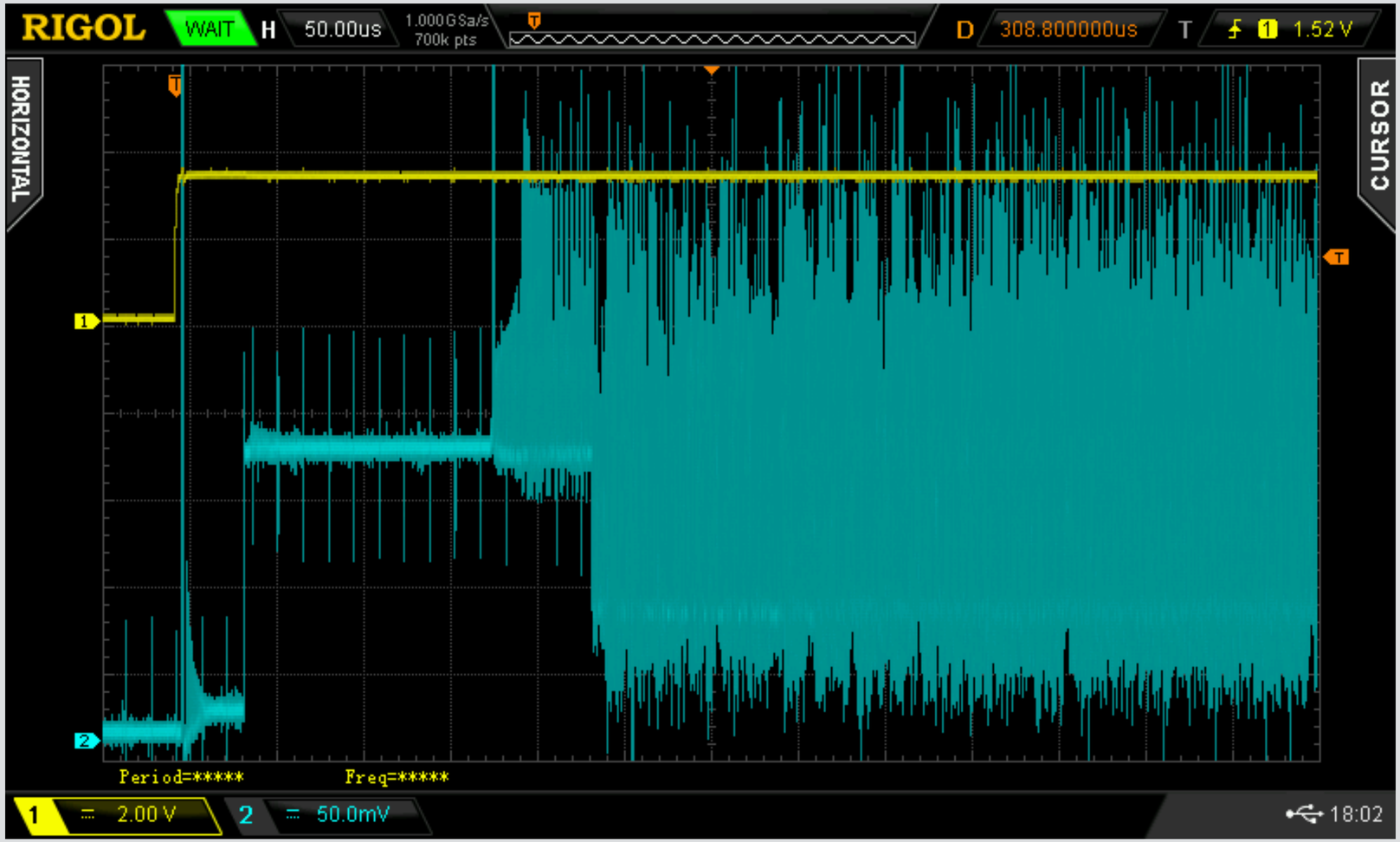
Glitching


Trigger on device boot
Wait until in bootloader
Glitch firmware validation check
Boot modified firmware

Voltage glitching: Flash reads

- Reading flash takes a lot of power
- ...We can interrupt that power
- The result gets undefined/garbled

Voltage glitching: Flash reads

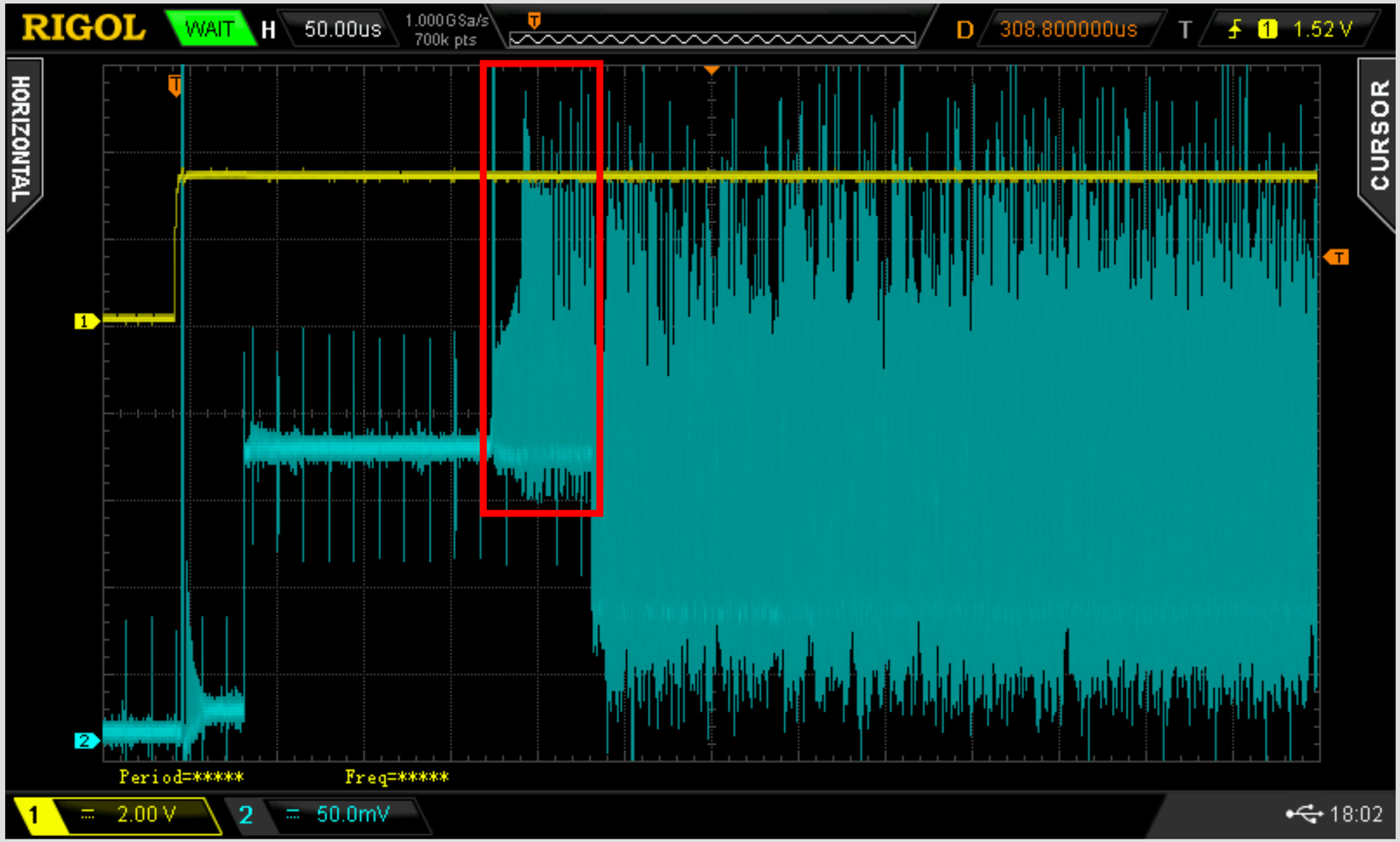


Power consumption

Voltage glitching: Flash reads

Power consumption

Voltage glitching: Flash reads



Power consumption

Voltage glitching: RAM reads

```
bool firmware_is_valid = validate_firmware();

if(firmware_is_valid)
        boot();
```

Voltage glitching: RAM reads

```
bool firmware_is_valid = validate_firmware();

if(firmware_is_valid)
        boot();
```

Voltage glitching: RAM reads

```
bool firmware_is_valid = validate_firmware();

if(firmware_is_valid)
        boot();
```

Voltage glitching: RAM reads

```
bool firmware_is_valid = validate_firmware();

if(firmware_is_valid)
        boot();
```

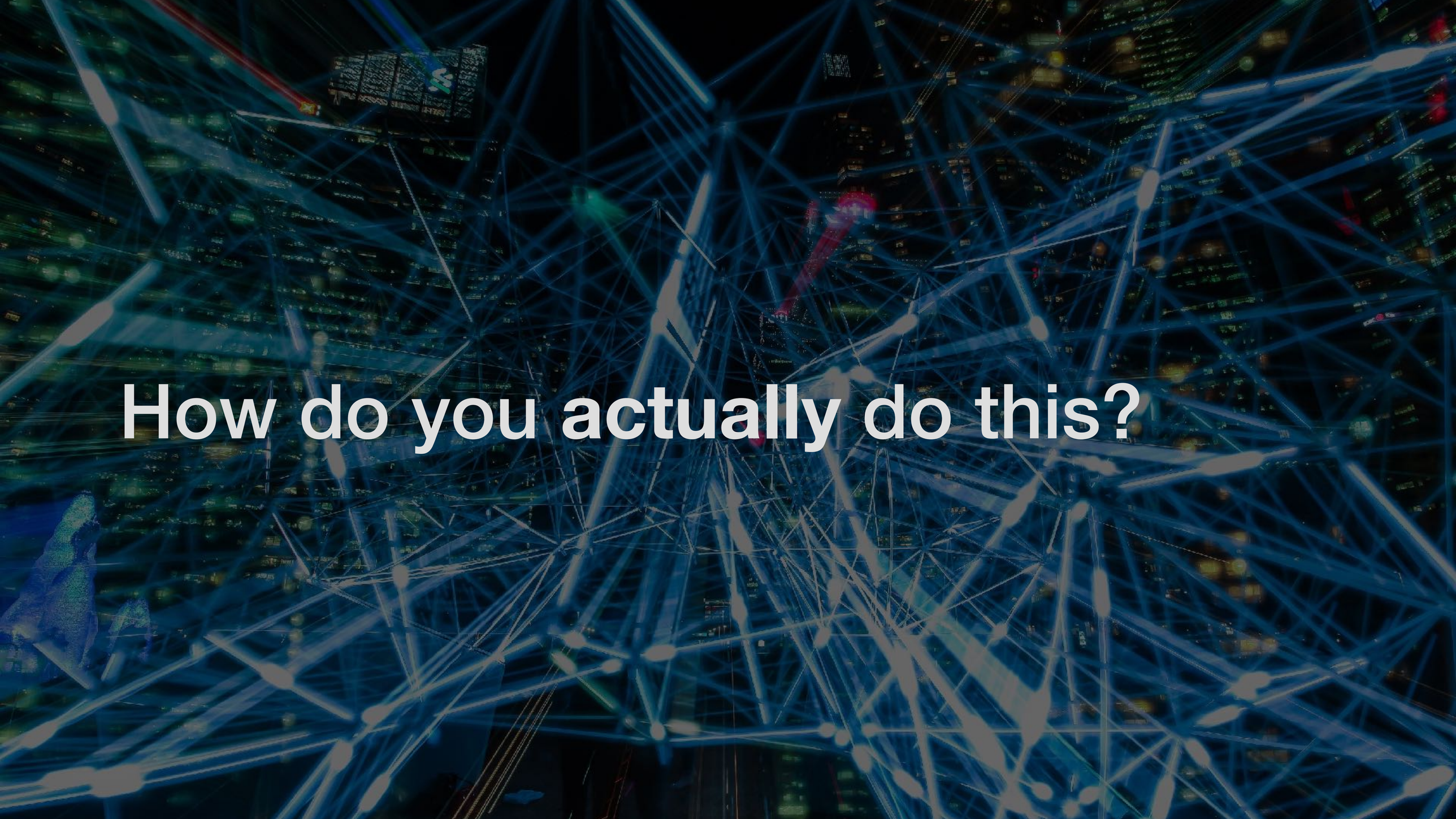
Voltage glitching: RAM reads

```
bool firmware_is_valid = validate_firmware();

if(firmware_is_valid)
        boot();
```

If we can flip a single bit
in firmware_is_valid we can boot!

Voltage glitching: RAM reads

```
bool firmware_is_valid = validate_firmware();

if(firmware_is_valid)
        boot();
```


How do you actually do this?

Three steps to success

- 1. Prepare the device
- 2. Setup a testing firmware
- 3. Hook it up & glitch!

Part 1: Preparing the device

Power domains

- Chips run on multiple voltages, i.e.:
  - Input/Outputs at 3.3V
  - CPU Core at 0.7-1.2V
  - WiFi at 1.3

Power domains

5.3.1.4 Regulator configuration examples

The voltage regulators can be configured in several ways, depending on the selected supply voltage mode (normal/high) and the regulator type option (LDO or DC/DC).

Four configuration examples are illustrated in images below.

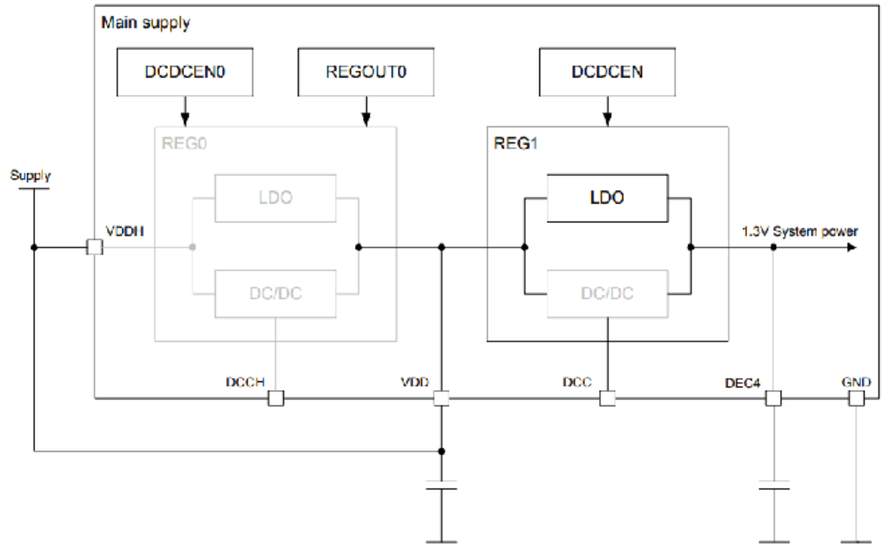
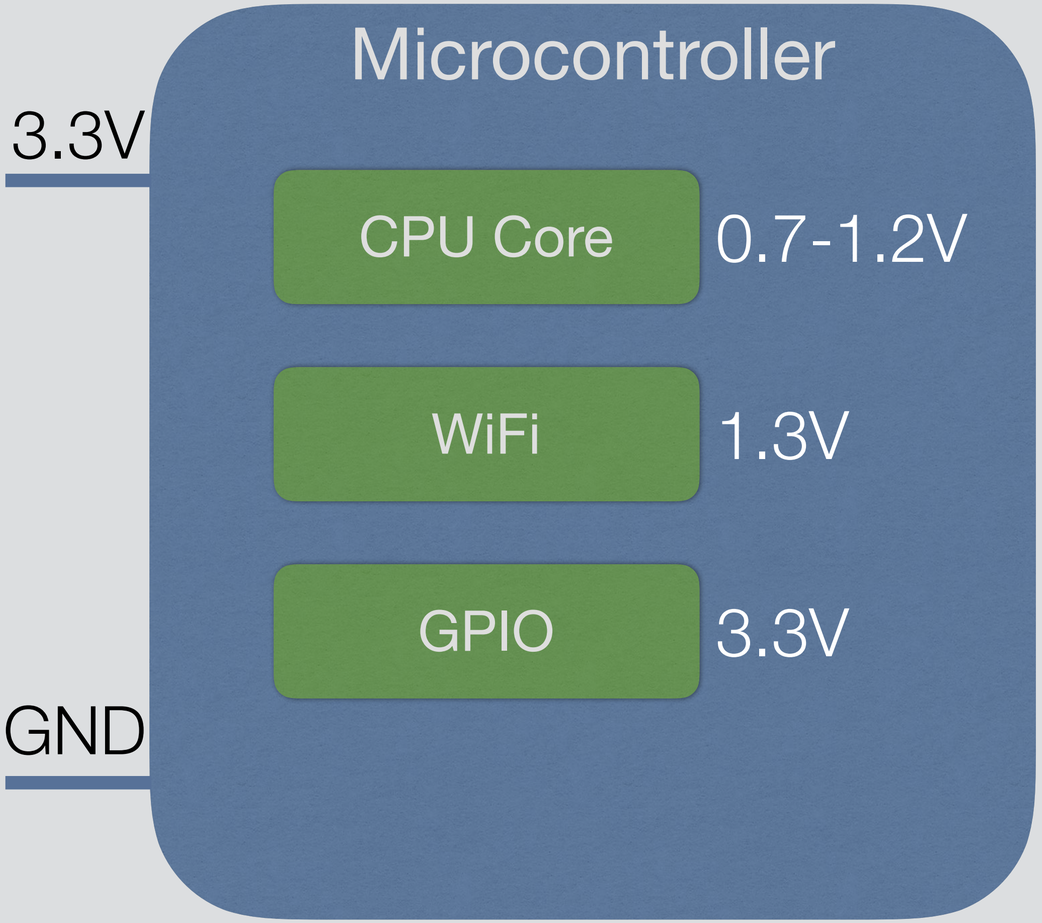
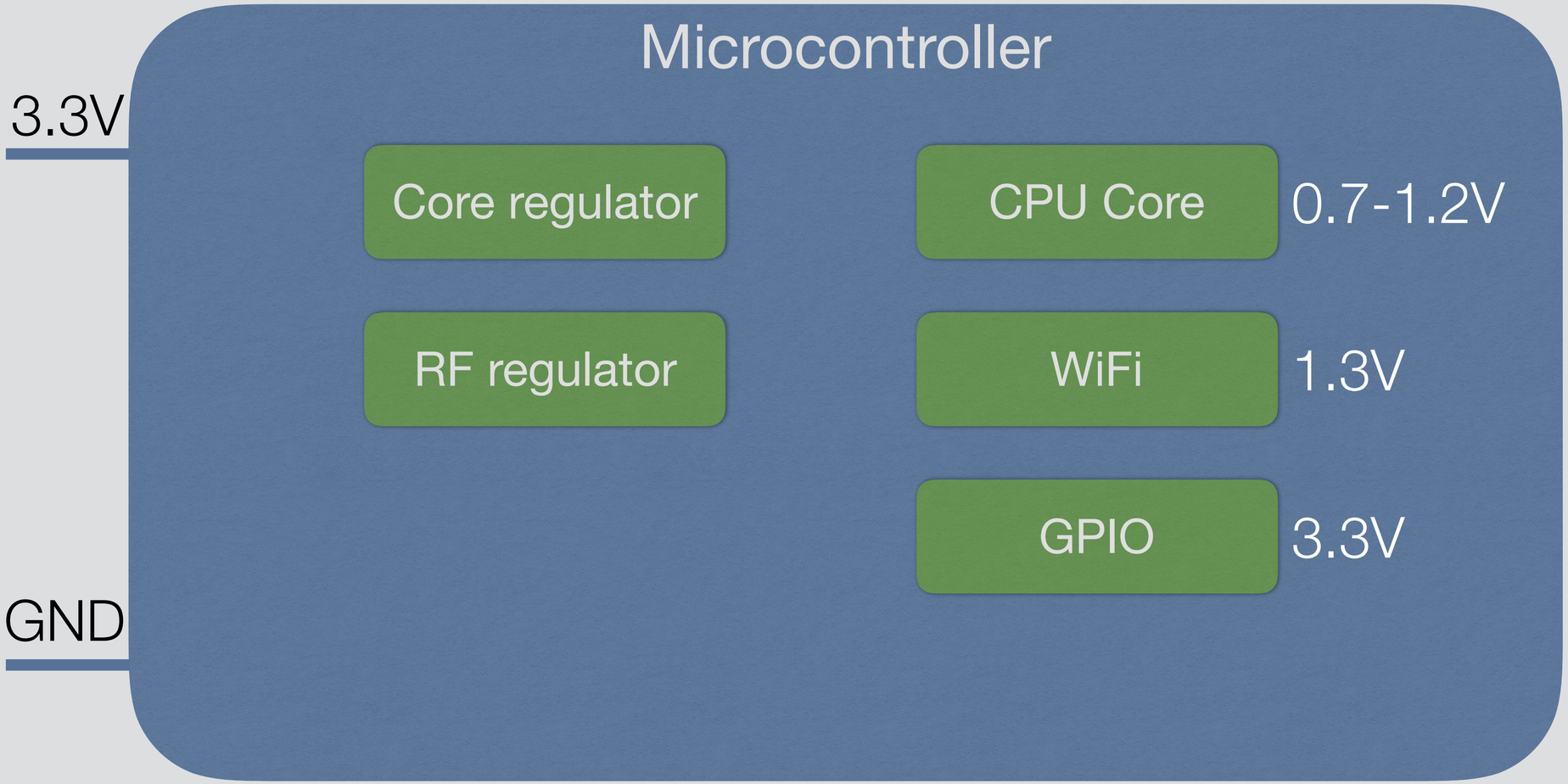


Figure 13: Normal voltage mode, LDO only

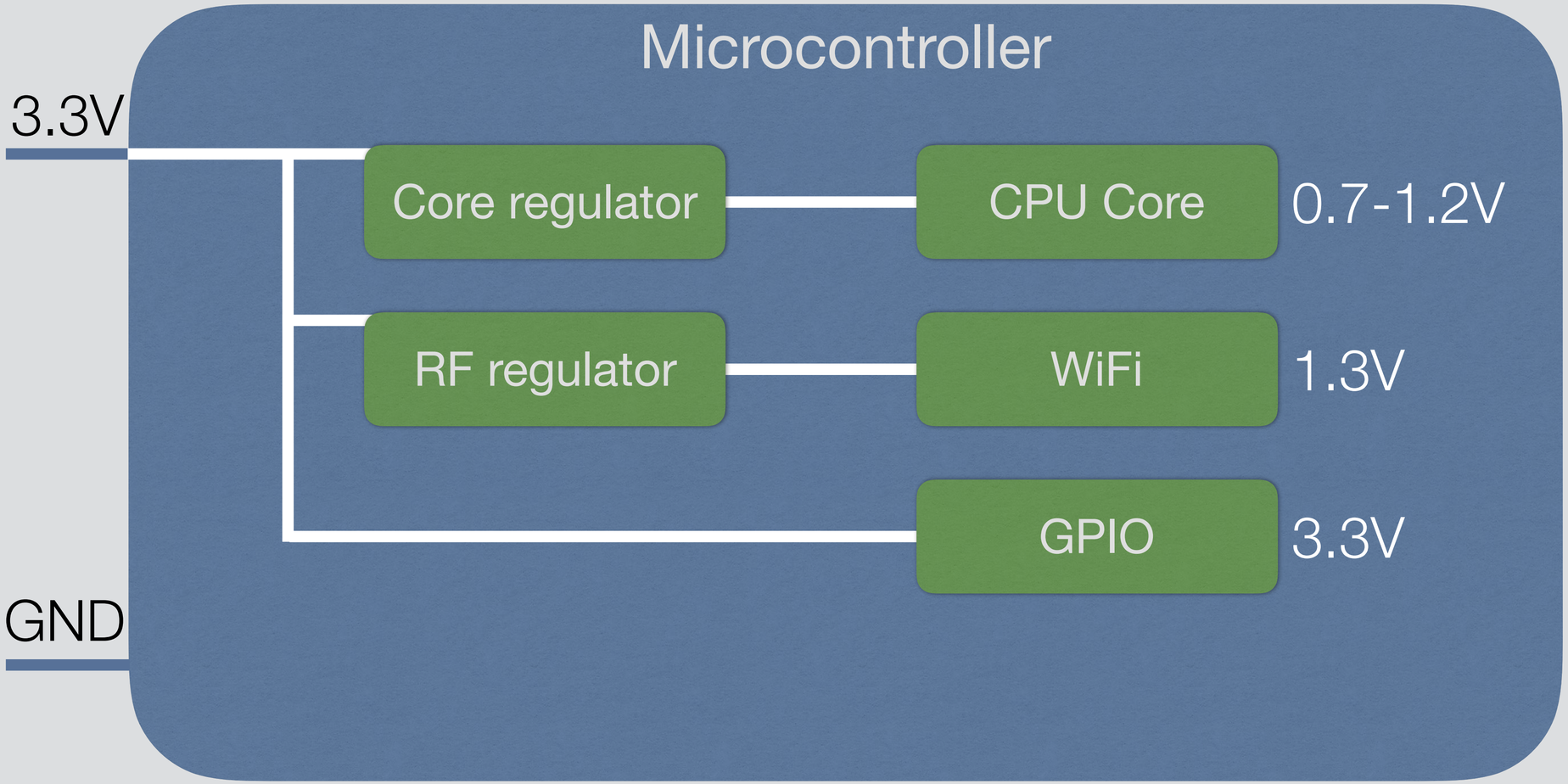
Power domains



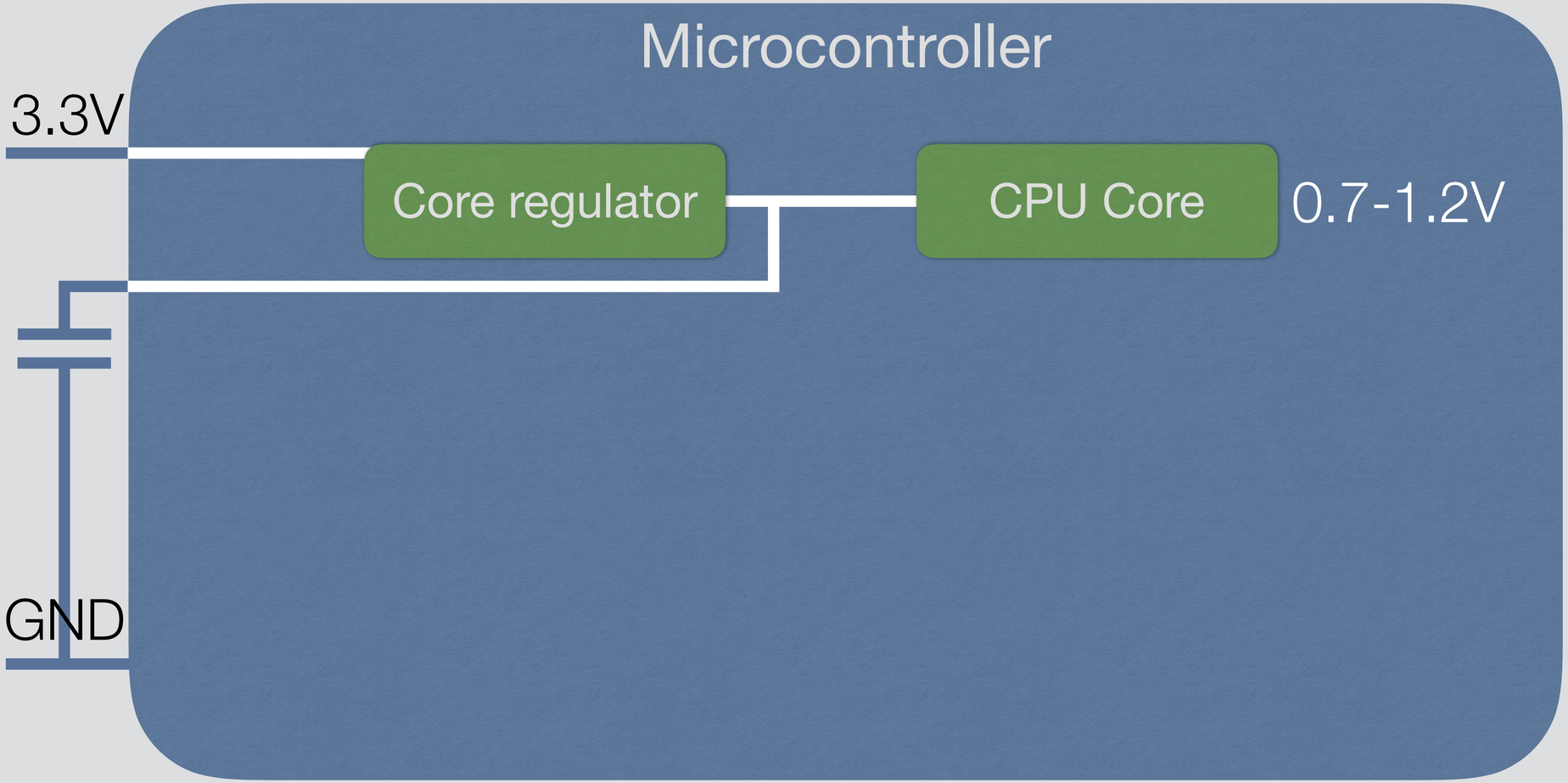
Power domains



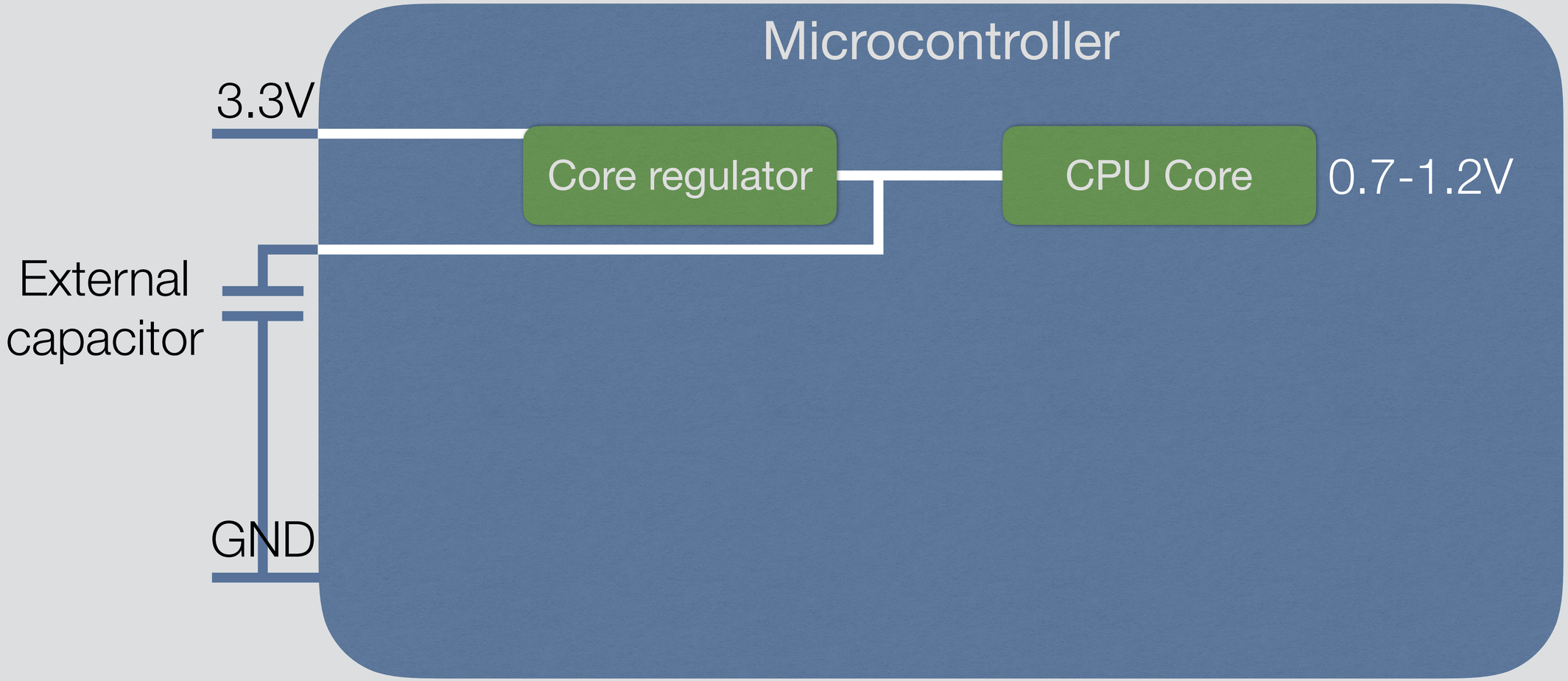
Power domains



Power domains

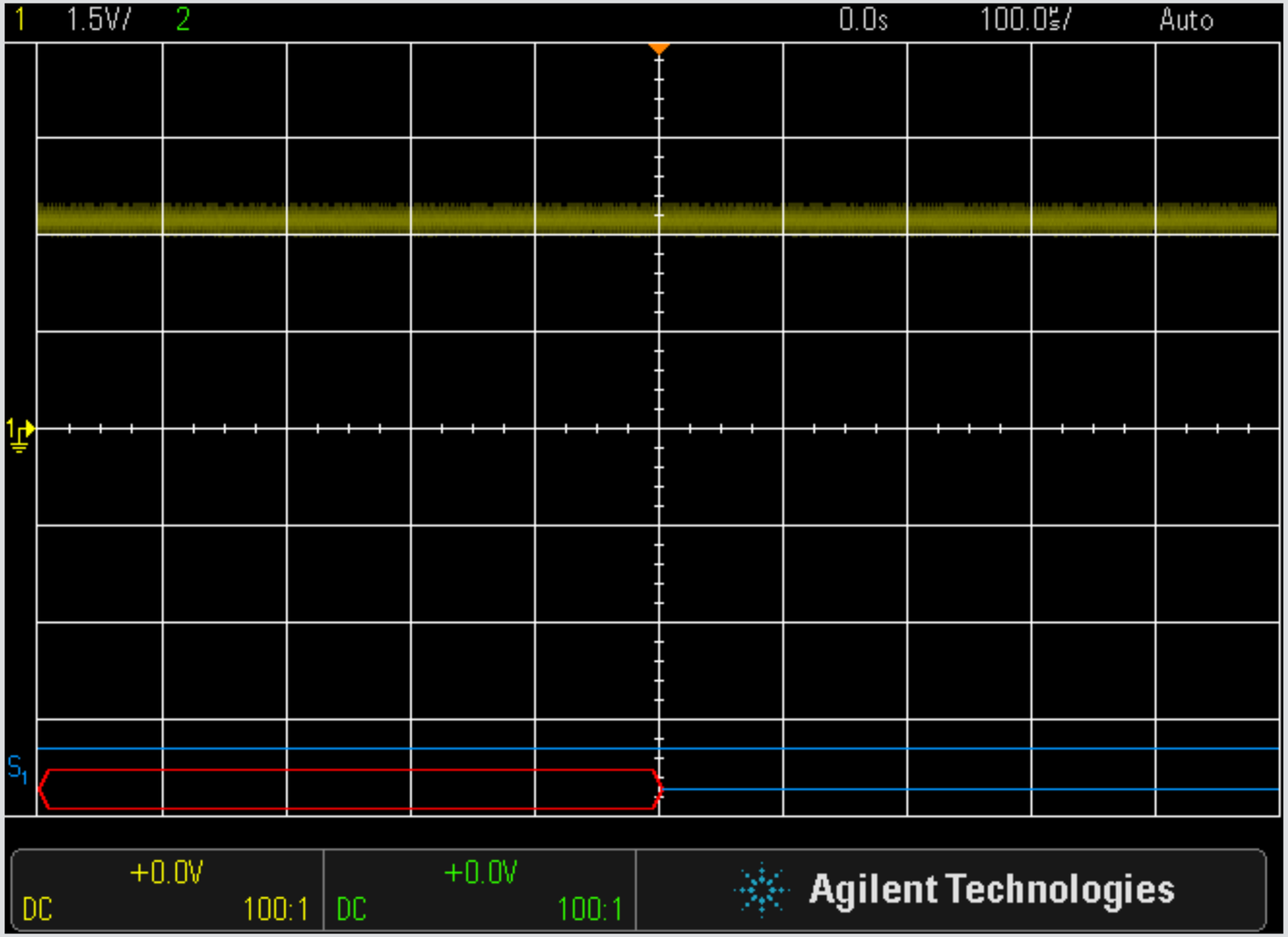


Power domains



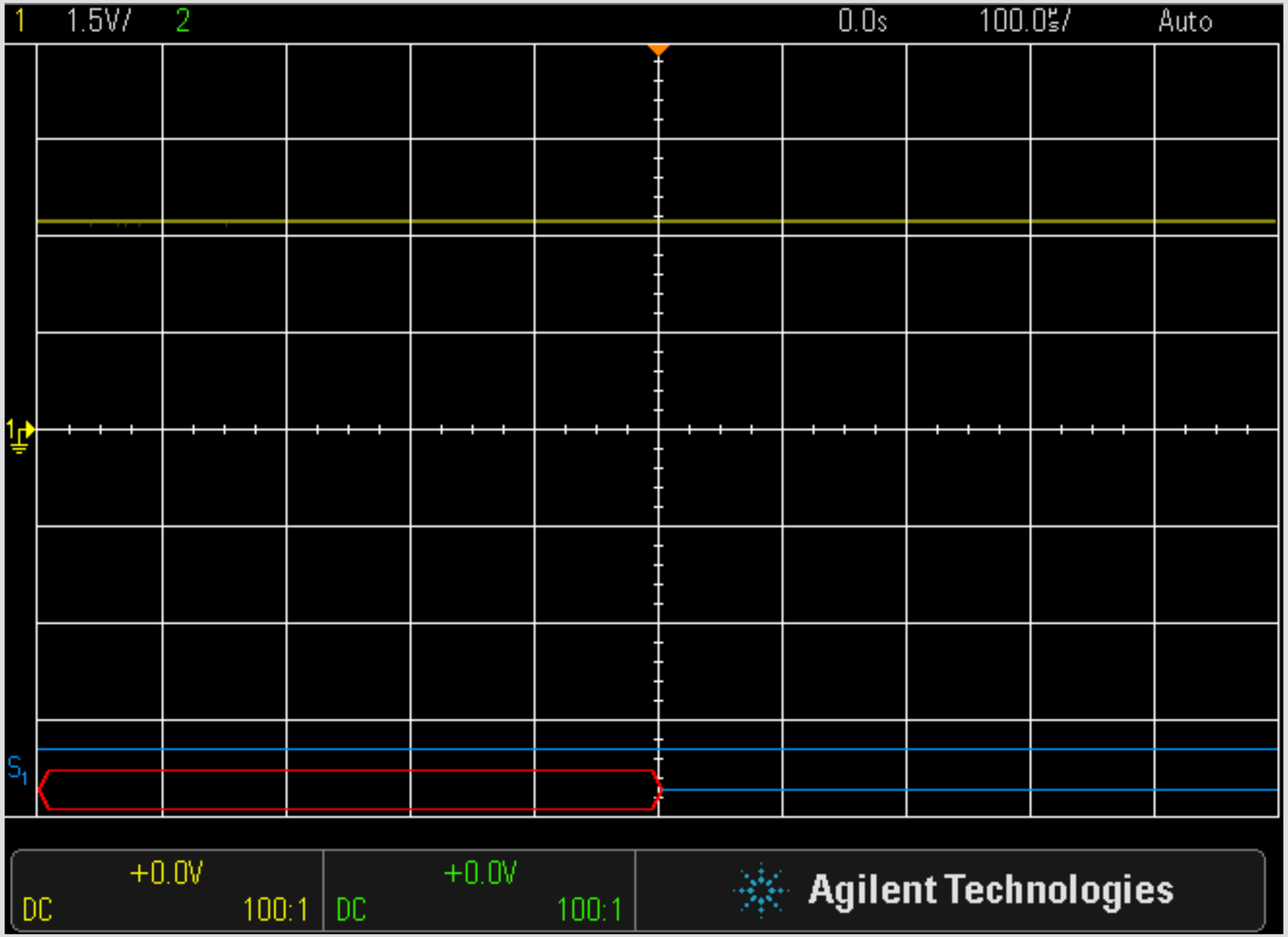
Power domains

Without bypass capacitors

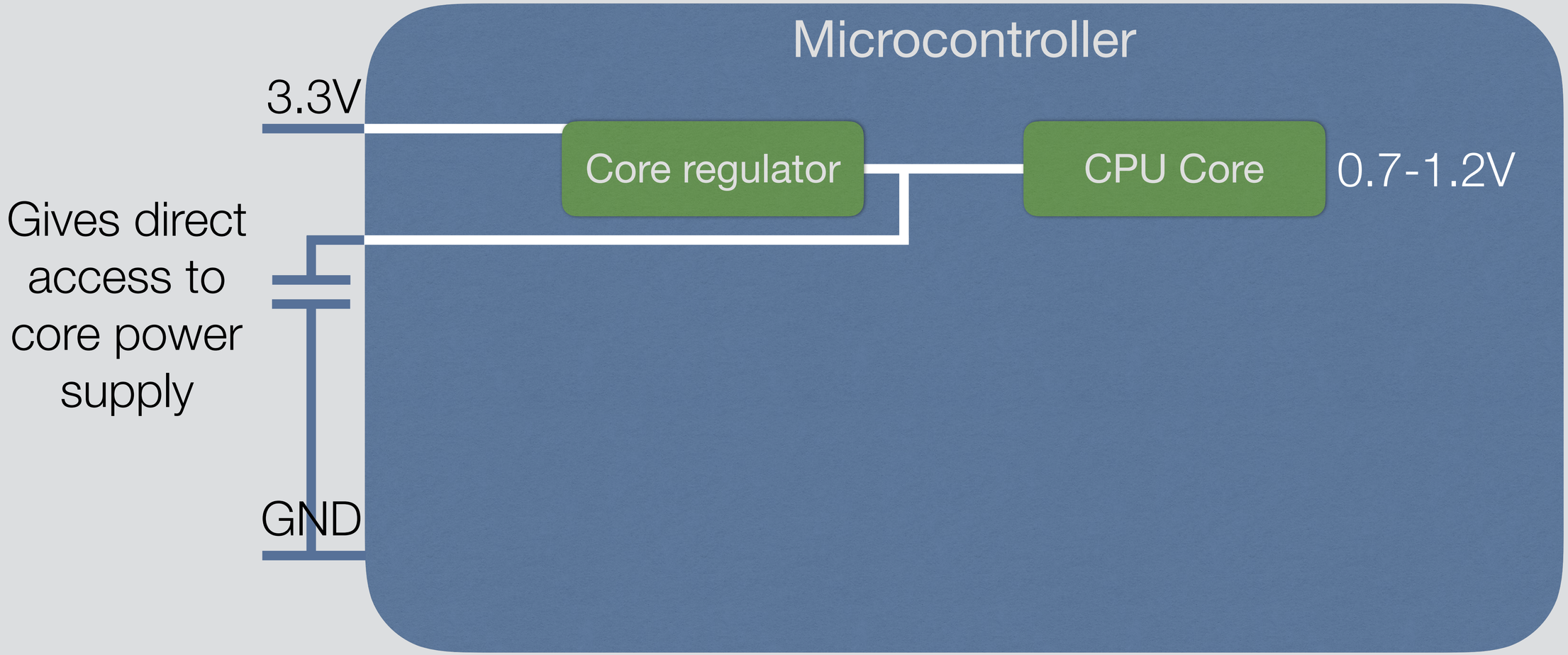


Power domains

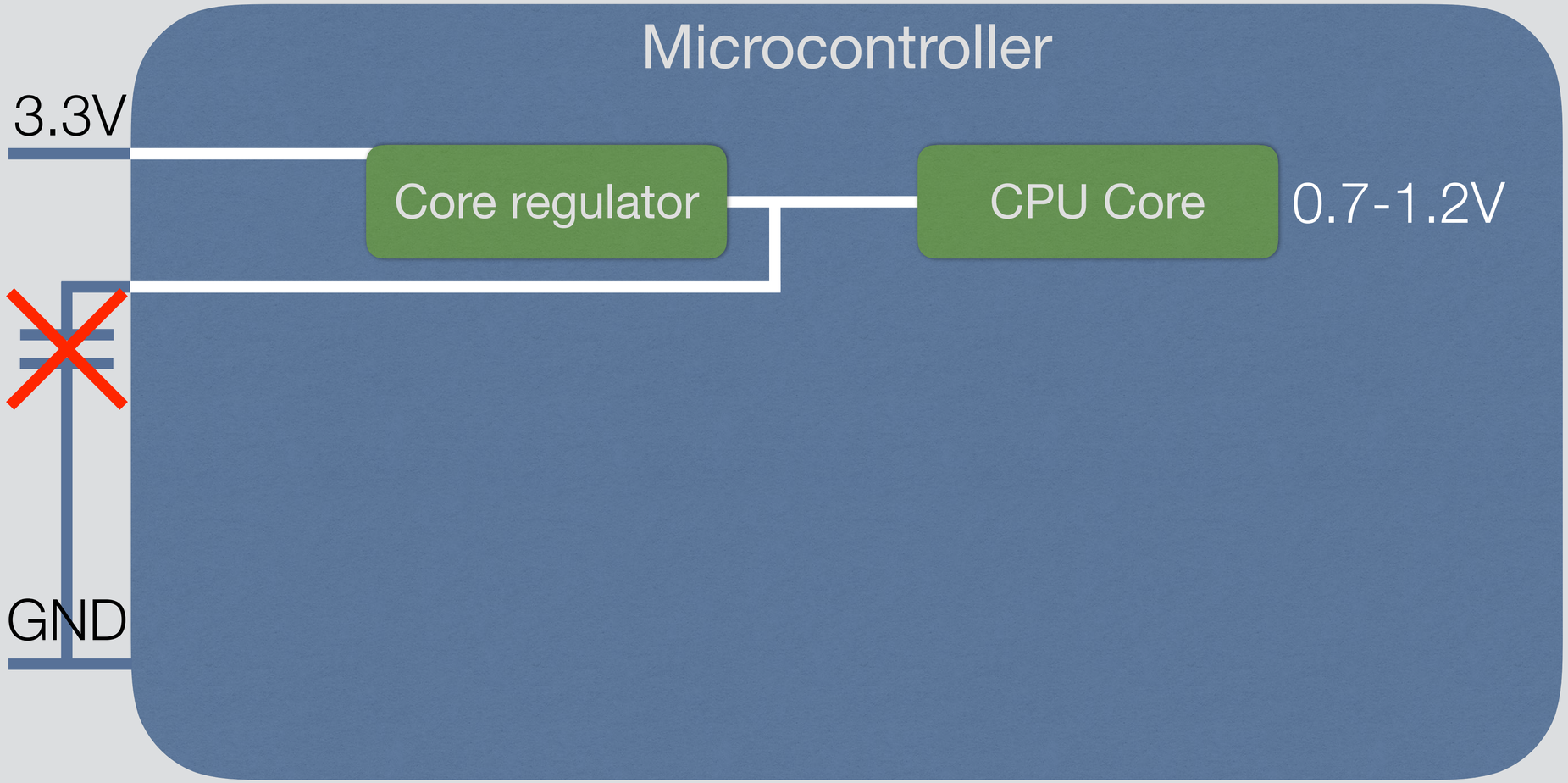
With bypass capacitors



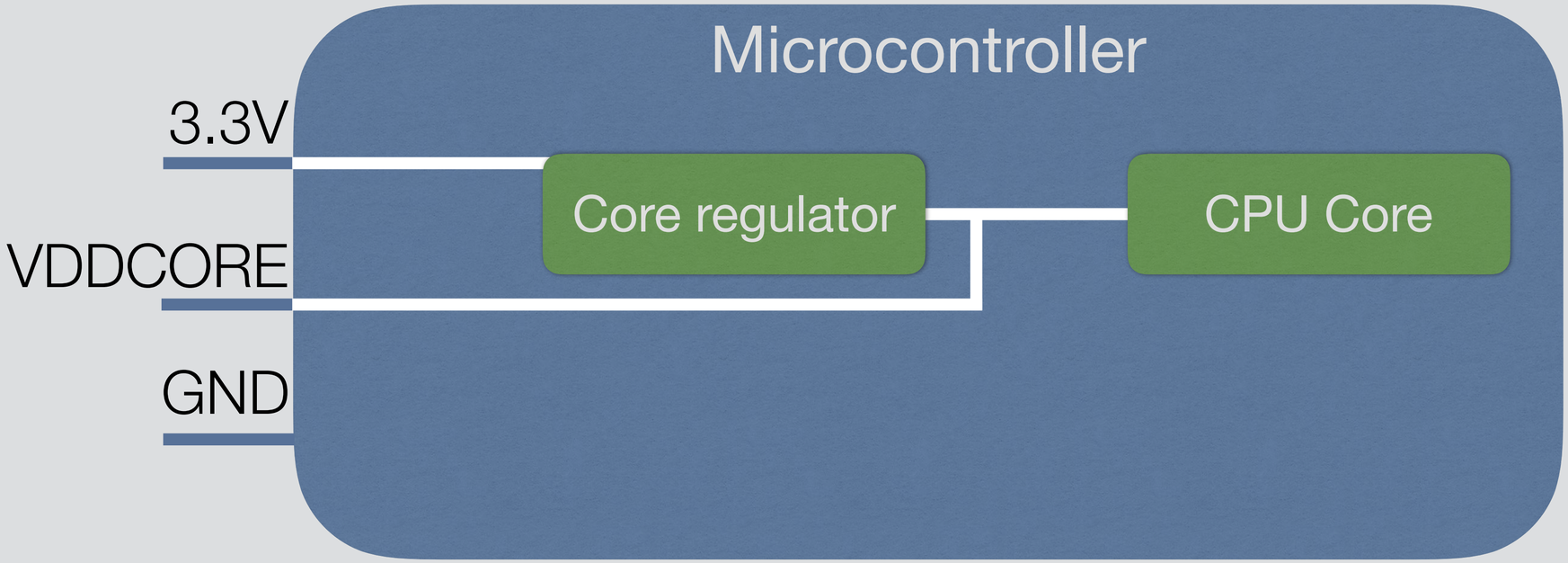
Power domains



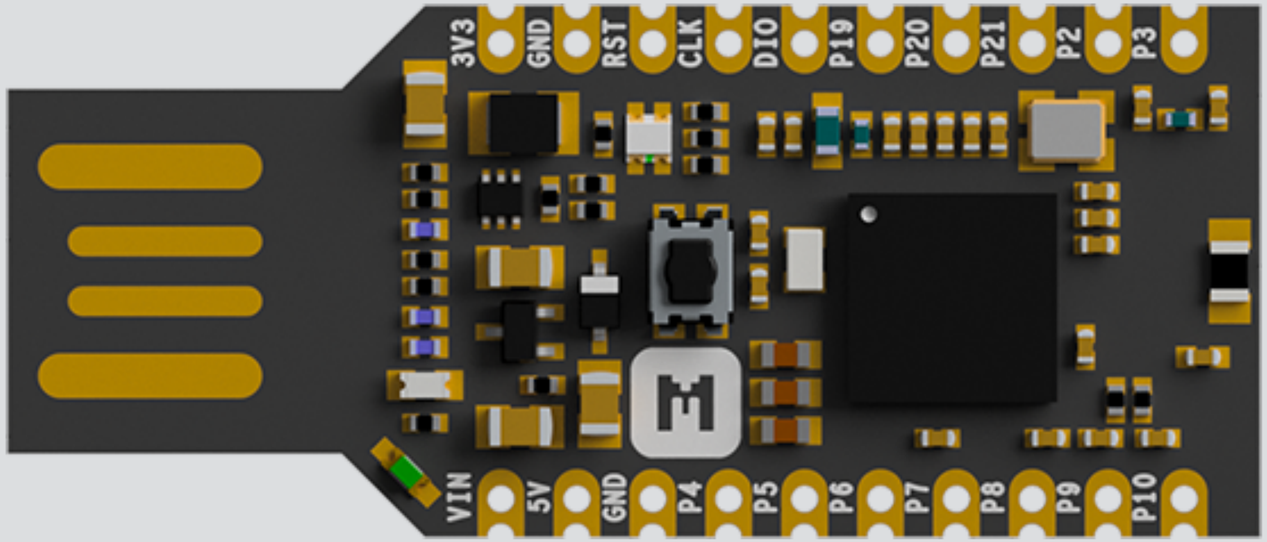
Power domains



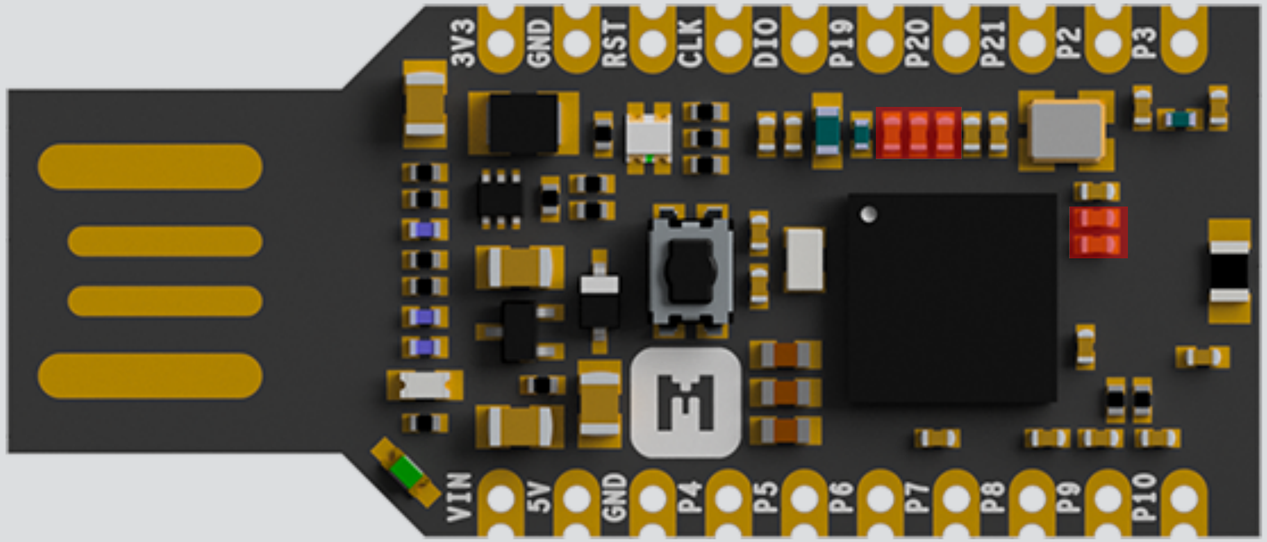
Power domains



Power domains



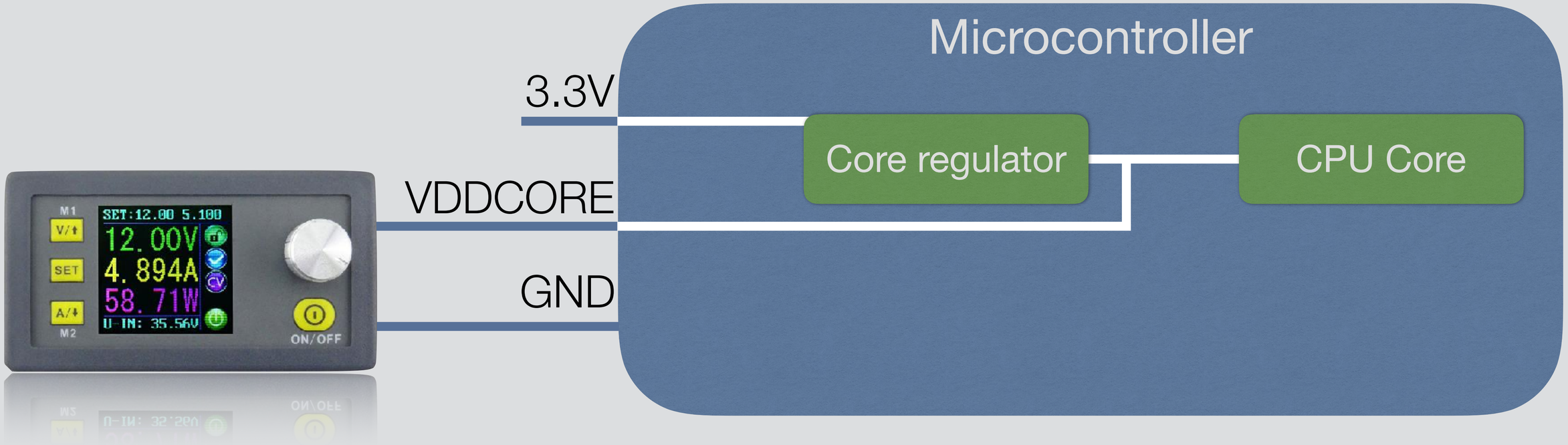
Power domains



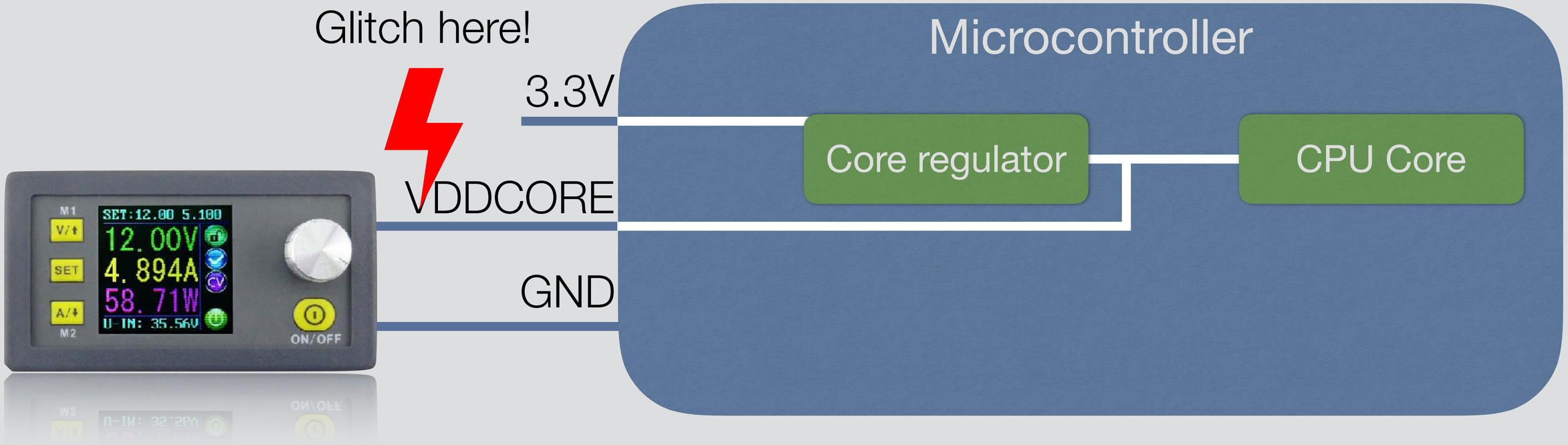
Removing capacitors: Problem...

- Depending on a lot of factors the chip might not run stable without capacitors
- Solution: Supply VCORE manually!

Power domains

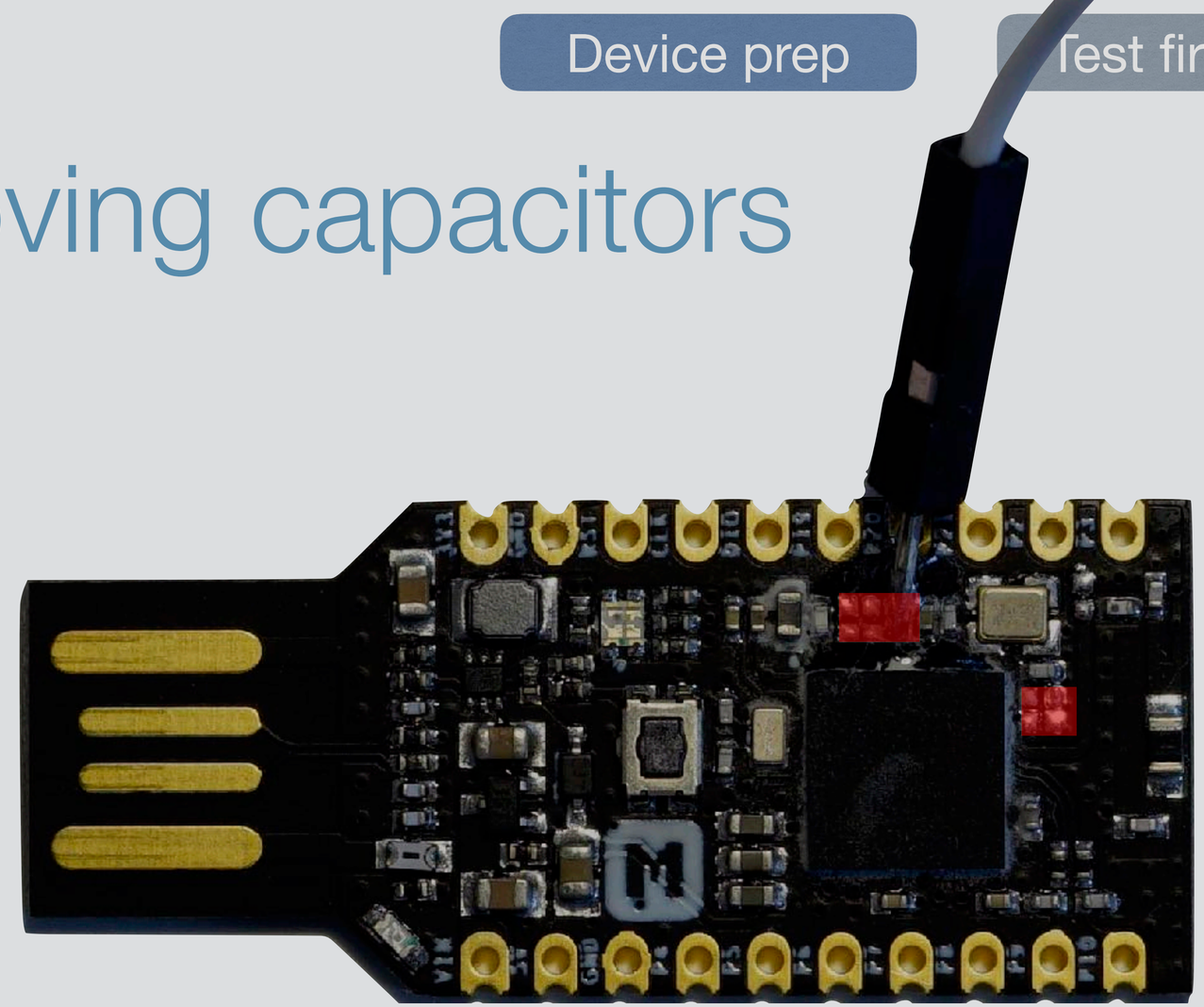


Power domains



Removing capacitors

Removing capacitors



Part 2: Building a test firmware

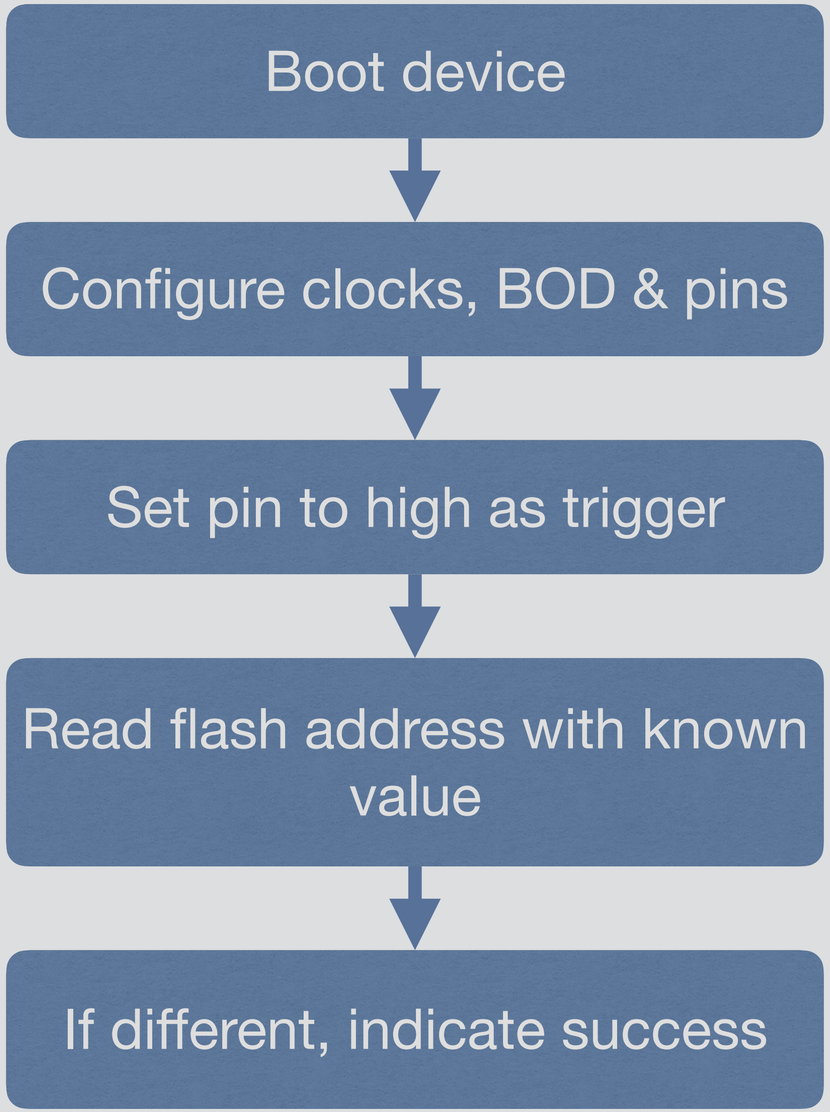
Test firmware

- Basic code to allow easy configuration of different chip features
- Should provide a good trigger
- Should provide a success indicator

Test firmware

Arduino sketch:

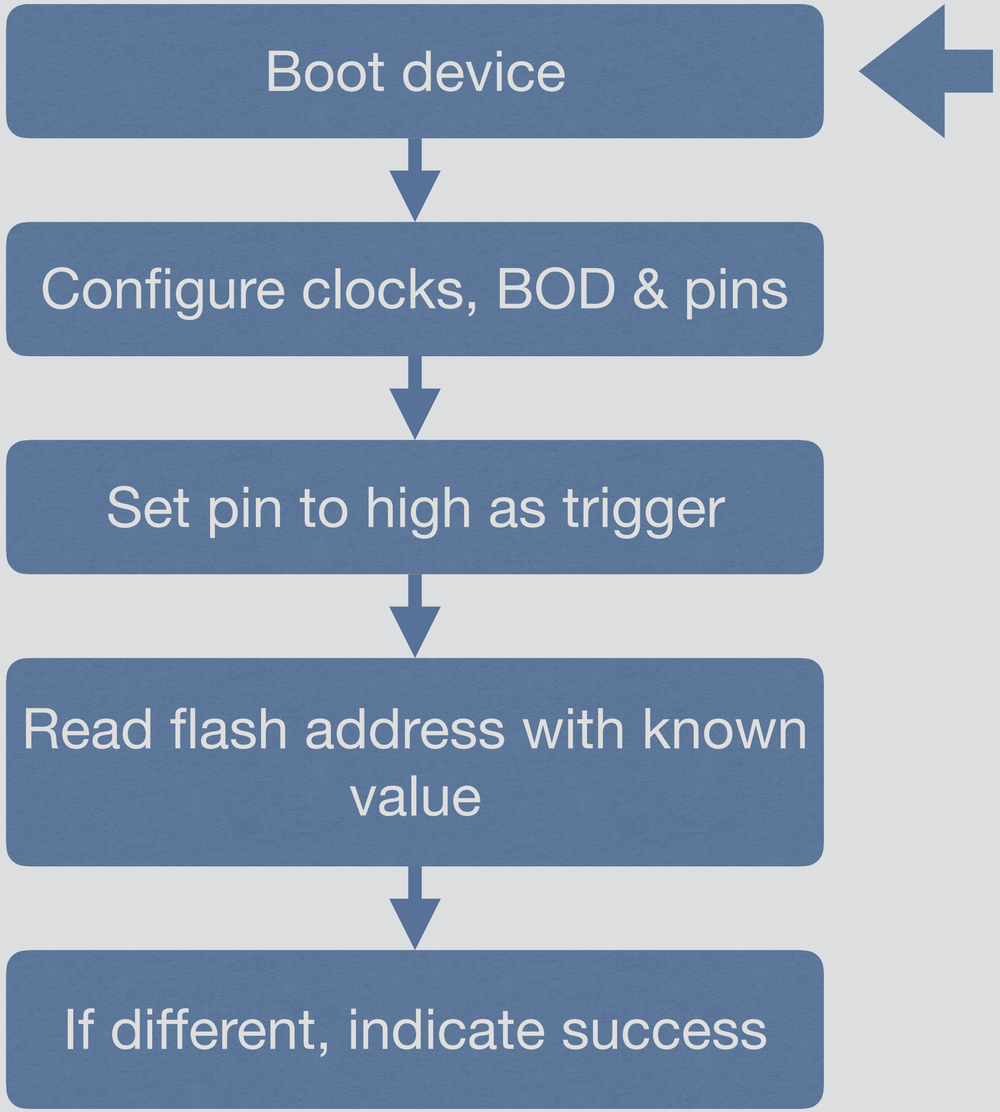
```
// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

    // Our success event
    digitalWrite(2, HIGH);

while(1);
```



Test firmware

Arduino sketch:

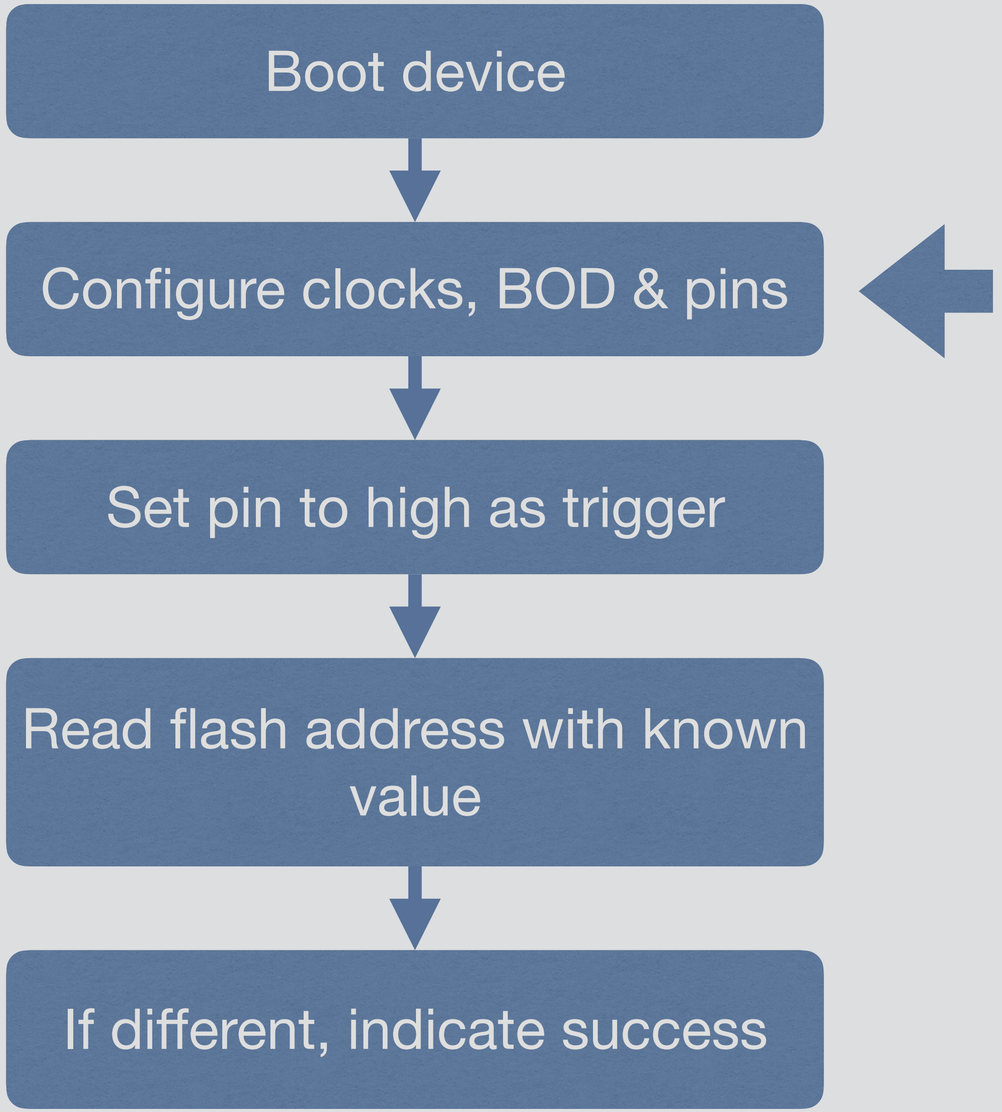
```
// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

    // Our success event
    digitalWrite(2, HIGH);

while(1);
```



Test firmware

Arduino sketch:

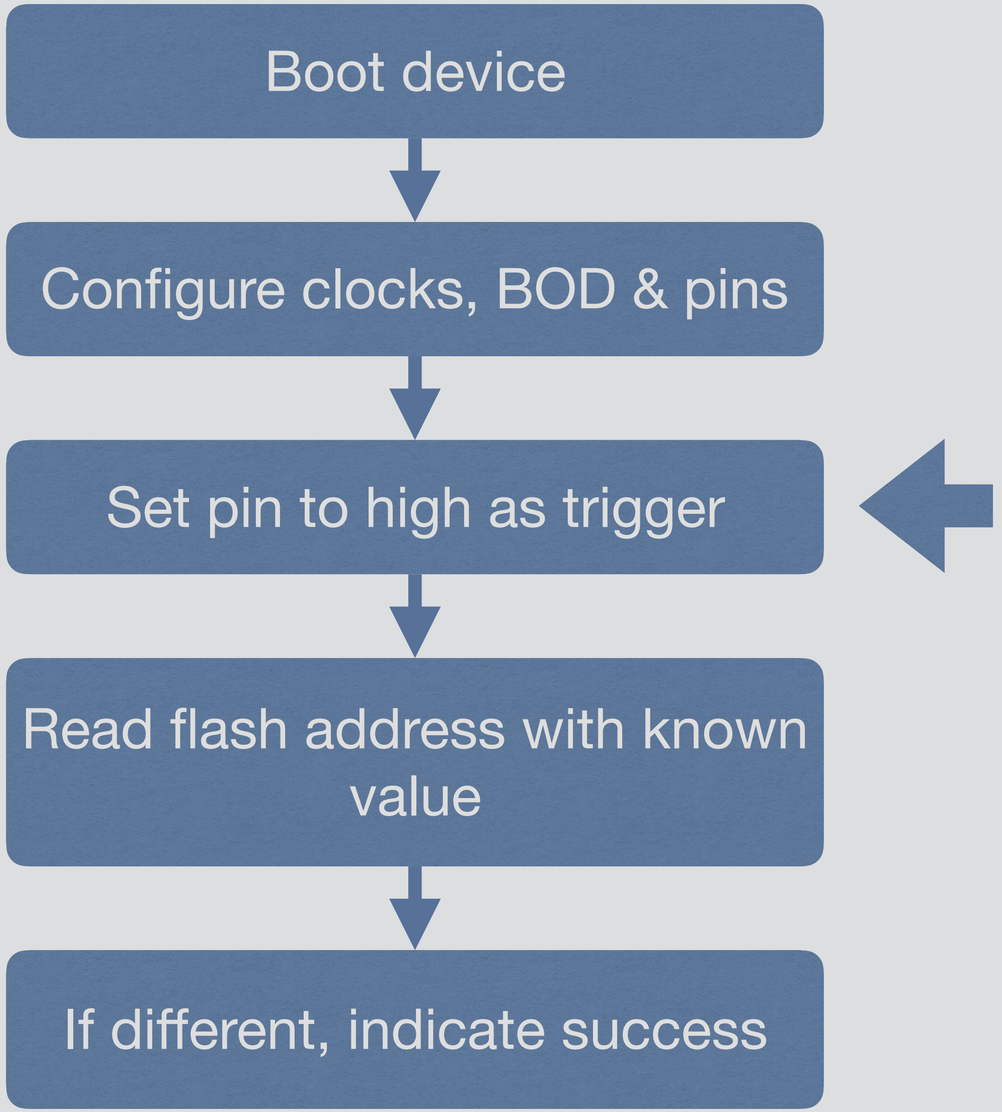
```
// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

    // Our success event
    digitalWrite(2, HIGH);

while(1);
```



Test firmware

Arduino sketch:

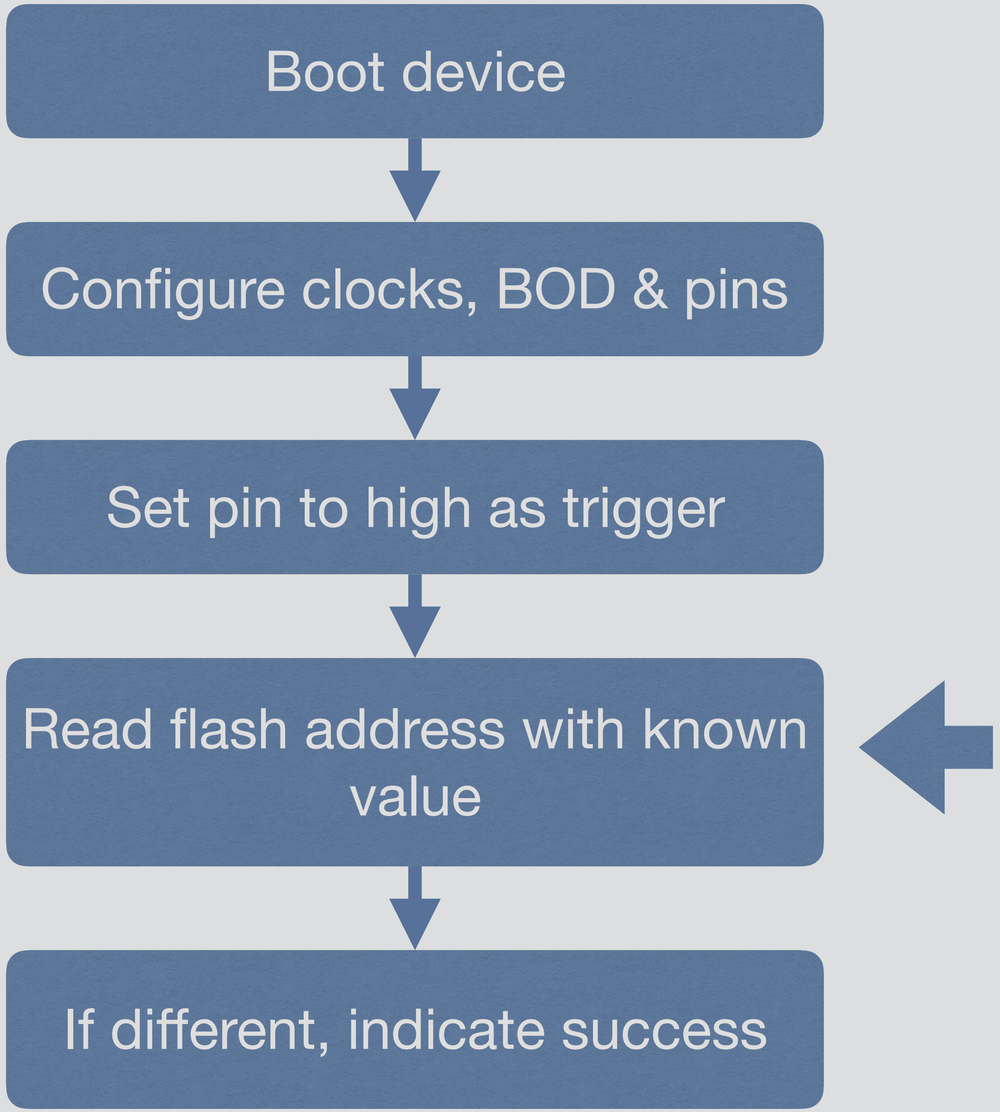
```
// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

    // Our success event
    digitalWrite(2, HIGH);

while(1);
```



Test firmware

Arduino sketch:

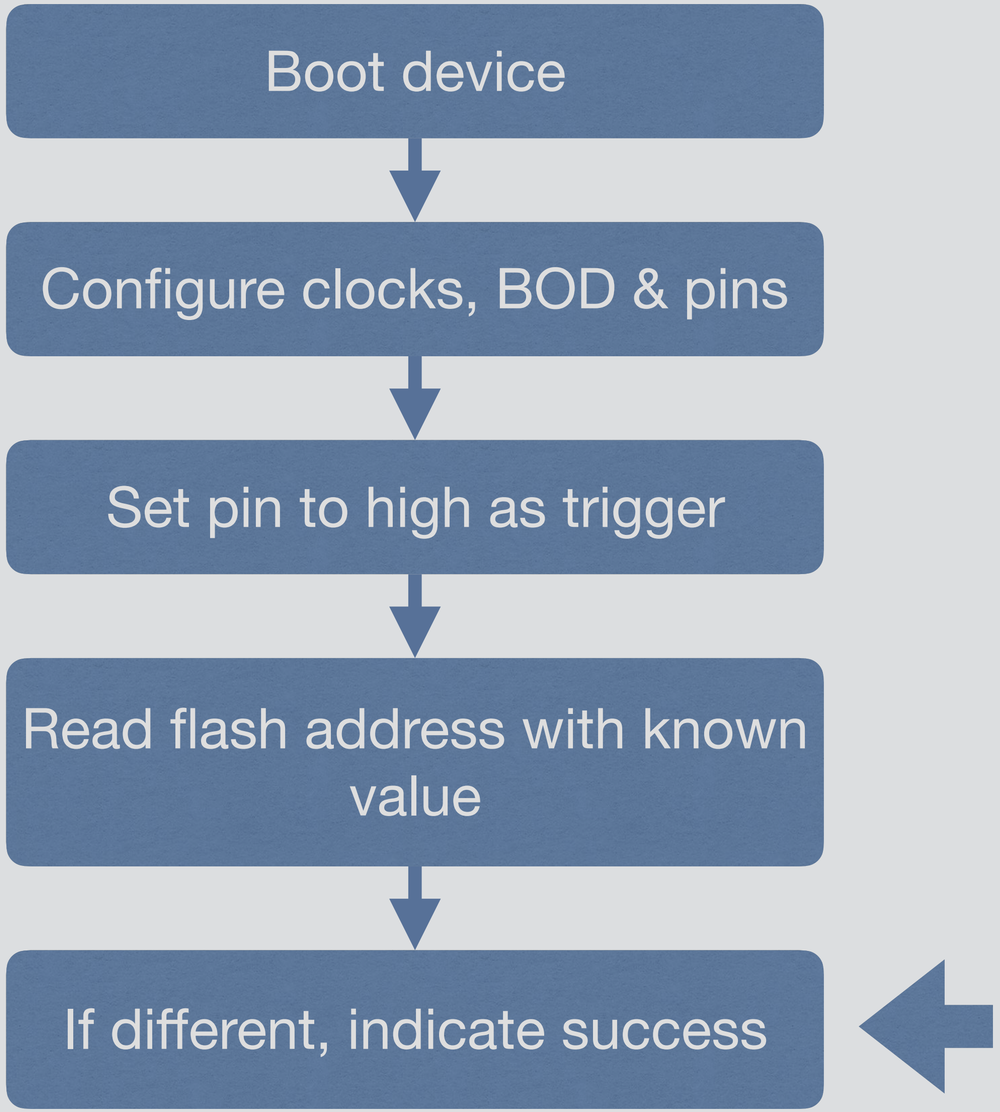
```
// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

    // Our success event
    digitalWrite(2, HIGH);

while(1);
```



Test firmware

Arduino sketch:

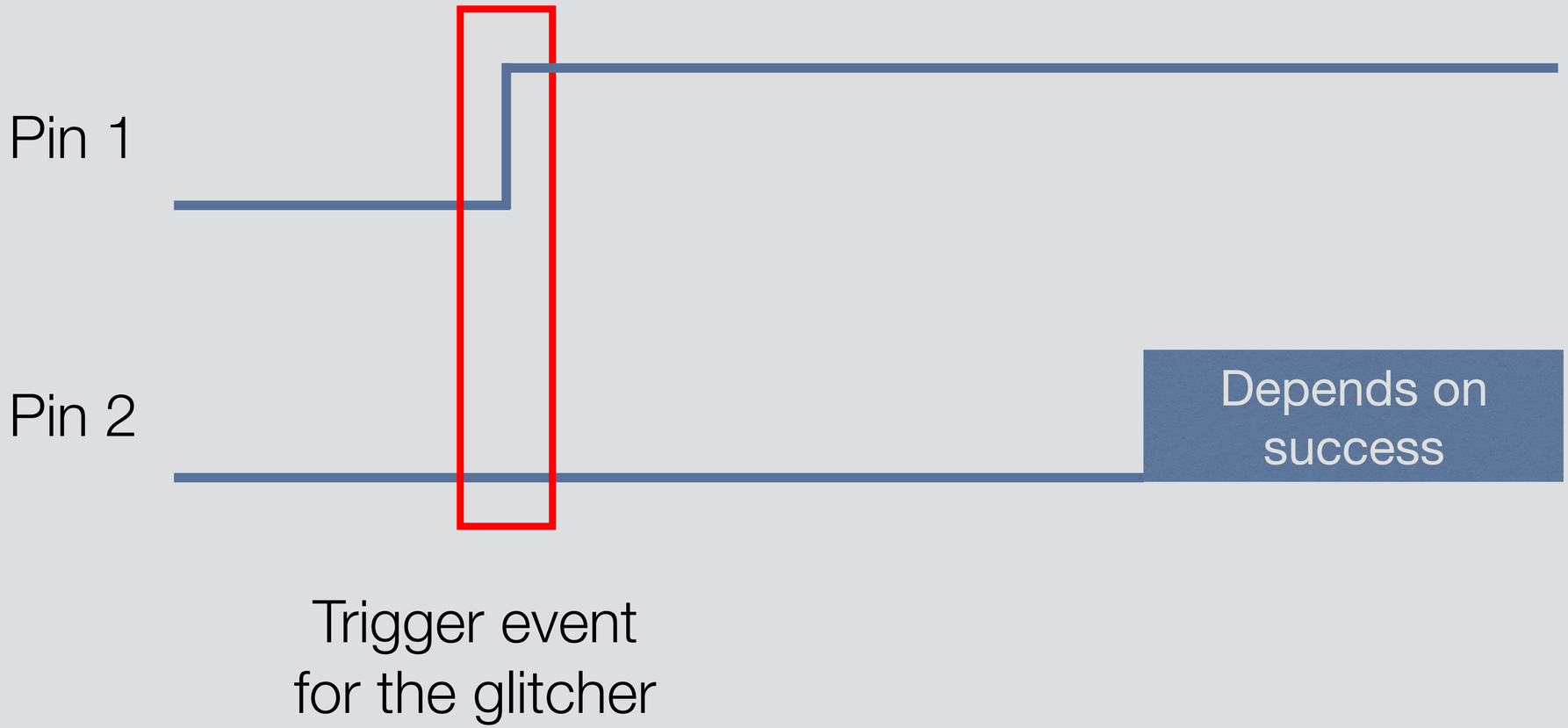
```
// Set pin 1 & 2 as output
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);

// Set pin high (used as our trigger
digitalWrite(1, HIGH);

// The read we are glitching
if(*0x0 != 0xC0FF_EEEE)

    // Our success event
    digitalWrite(2, HIGH);

while(1);
```



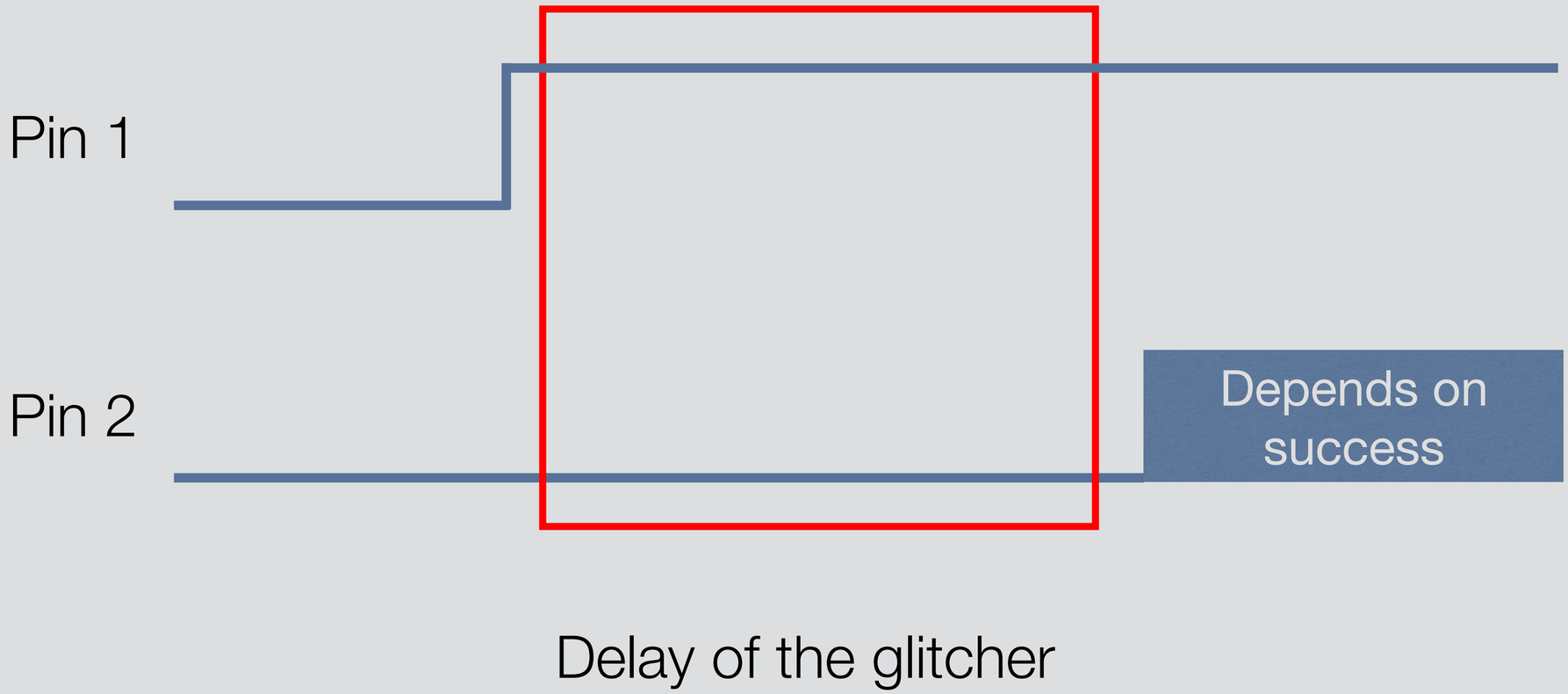
Test firmware

Test firmware

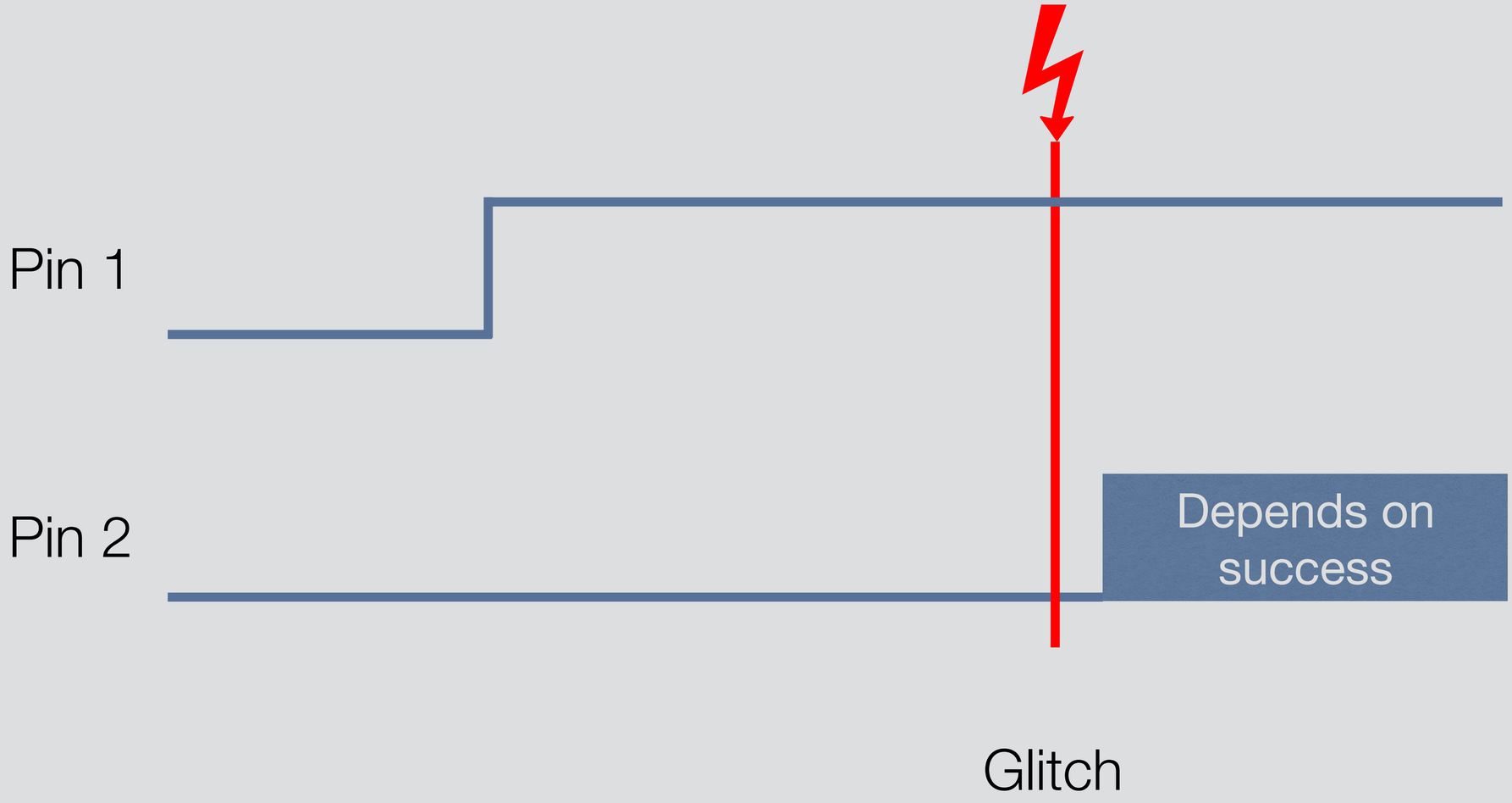
Test firmware



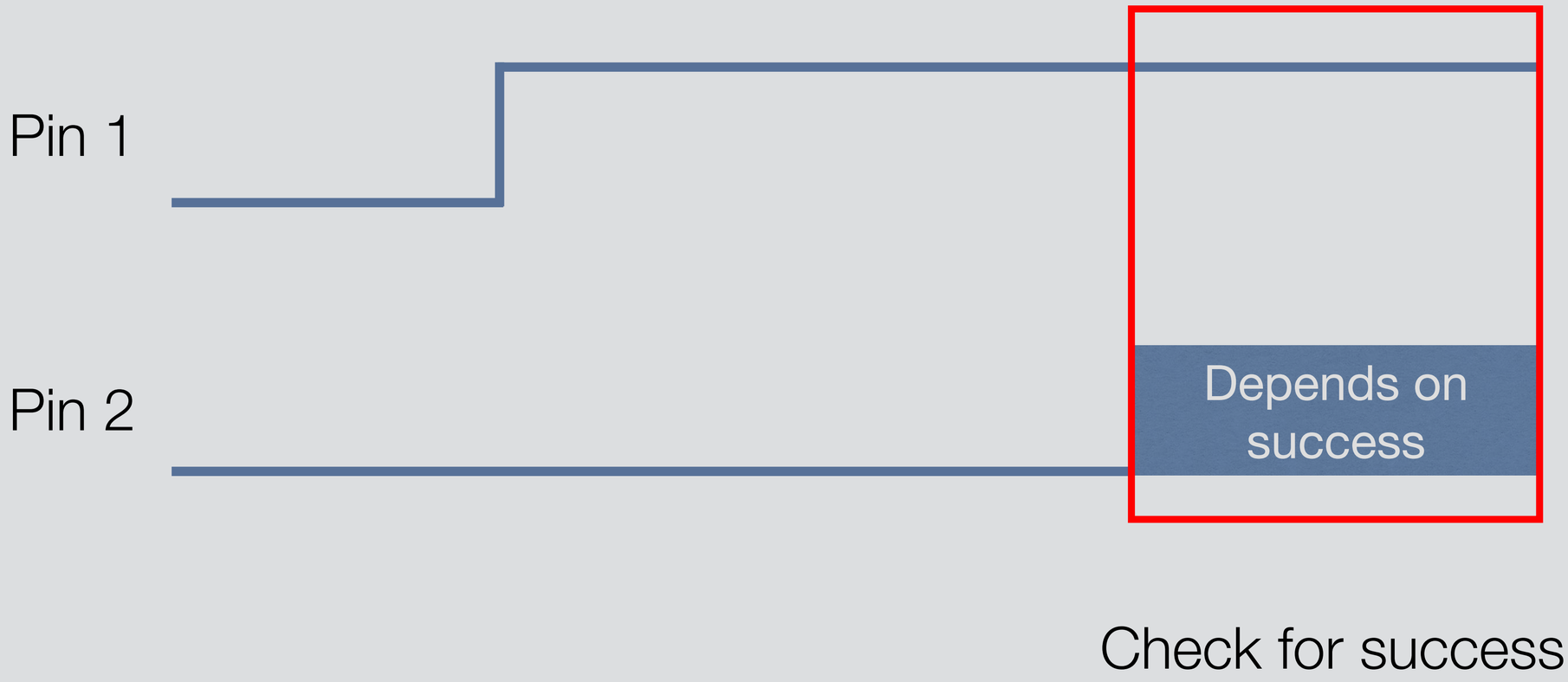
Test firmware



Test firmware



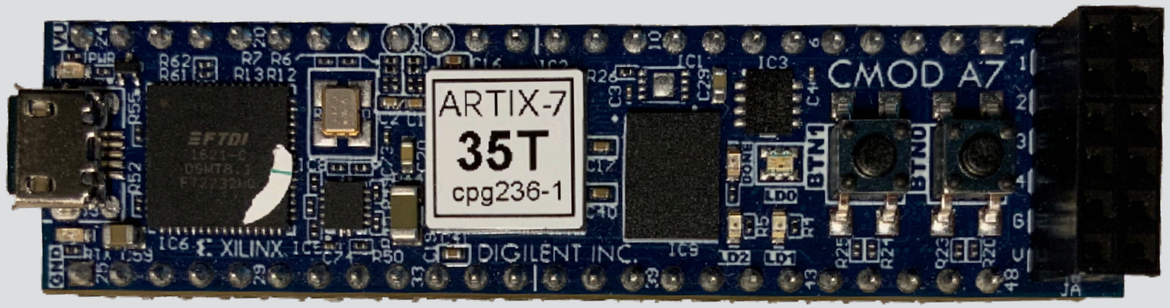
Test firmware



Part 3: The glitcher

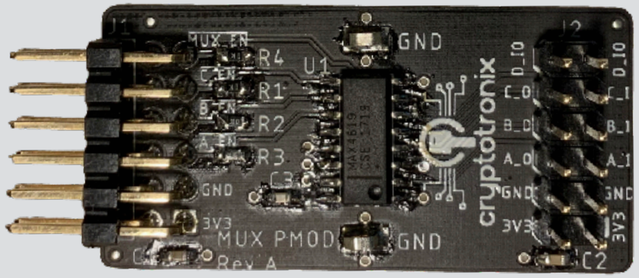
The chip.fail glitcher

Digilent Cmod A7
FPGA Board



ca. $70

MUX PMOD
Multiplexer Breakout



ca. $1.80

DPS3003
Power Supply

ca. $20

cryptotronix

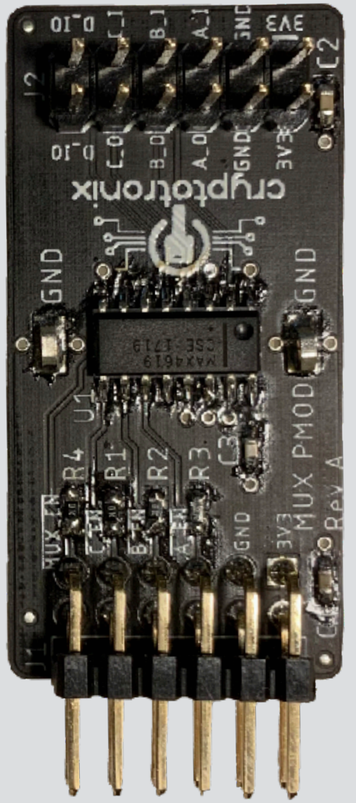
leveldown security

Digilent Cmod A7

- FPGAs provide super precise timing
- In our case, 100'000'000th of a second
- Open-source chip.fail firmware

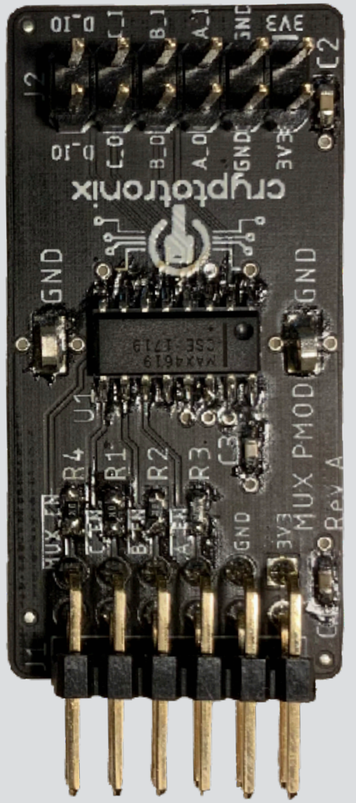
MAX PMOD

- Breakout for the Maxim MAX4619
- 3-channel analog switch
- Also available as DIP for breadboarding



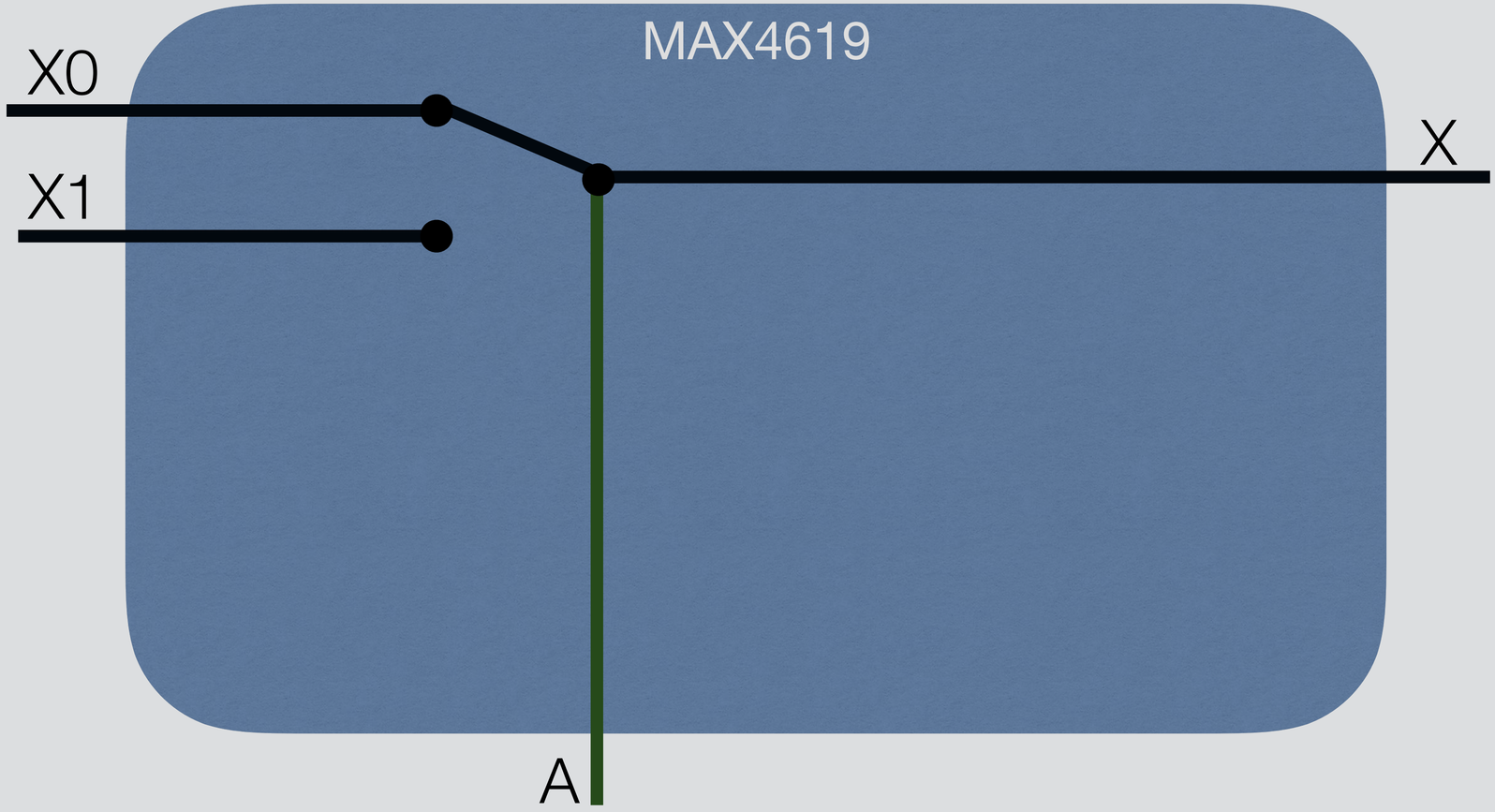
MAX PMOD

- Breakout for the Maxim MAX4619
- 3-channel analog switch
- Also available as DIP for breadboarding

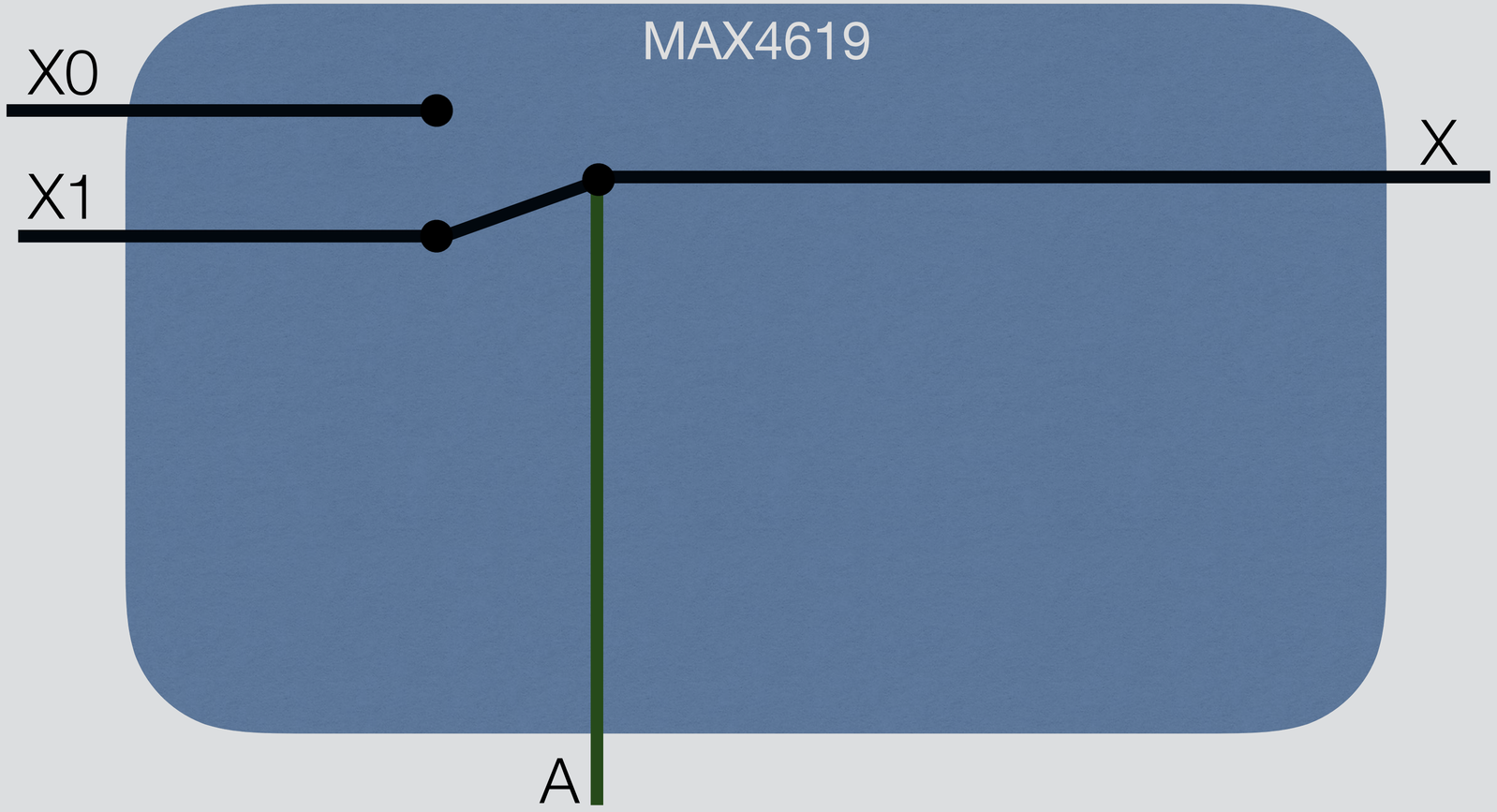


Want a PCB? Ask us!

MAX PMOD

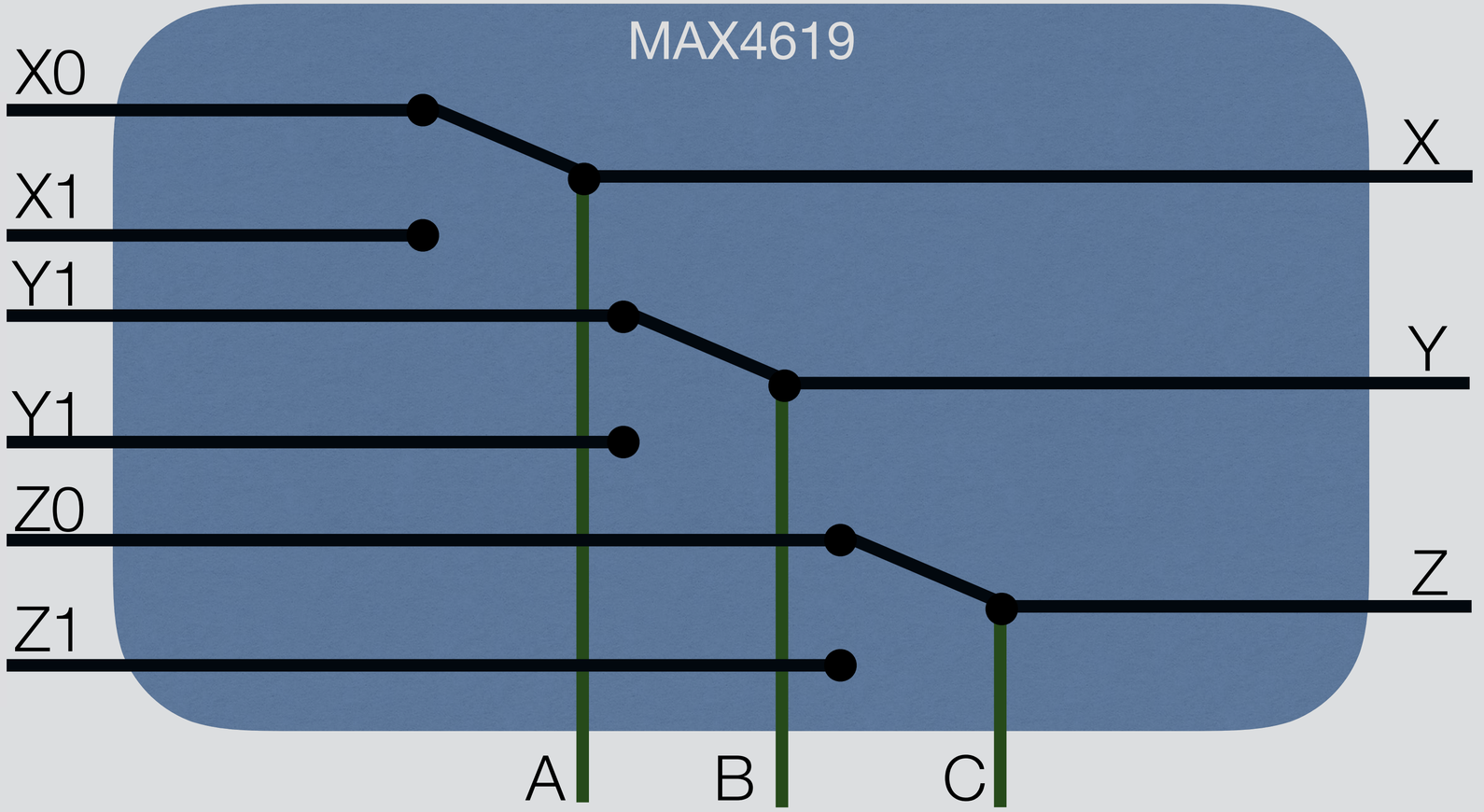


MAX PMOD



Apply power

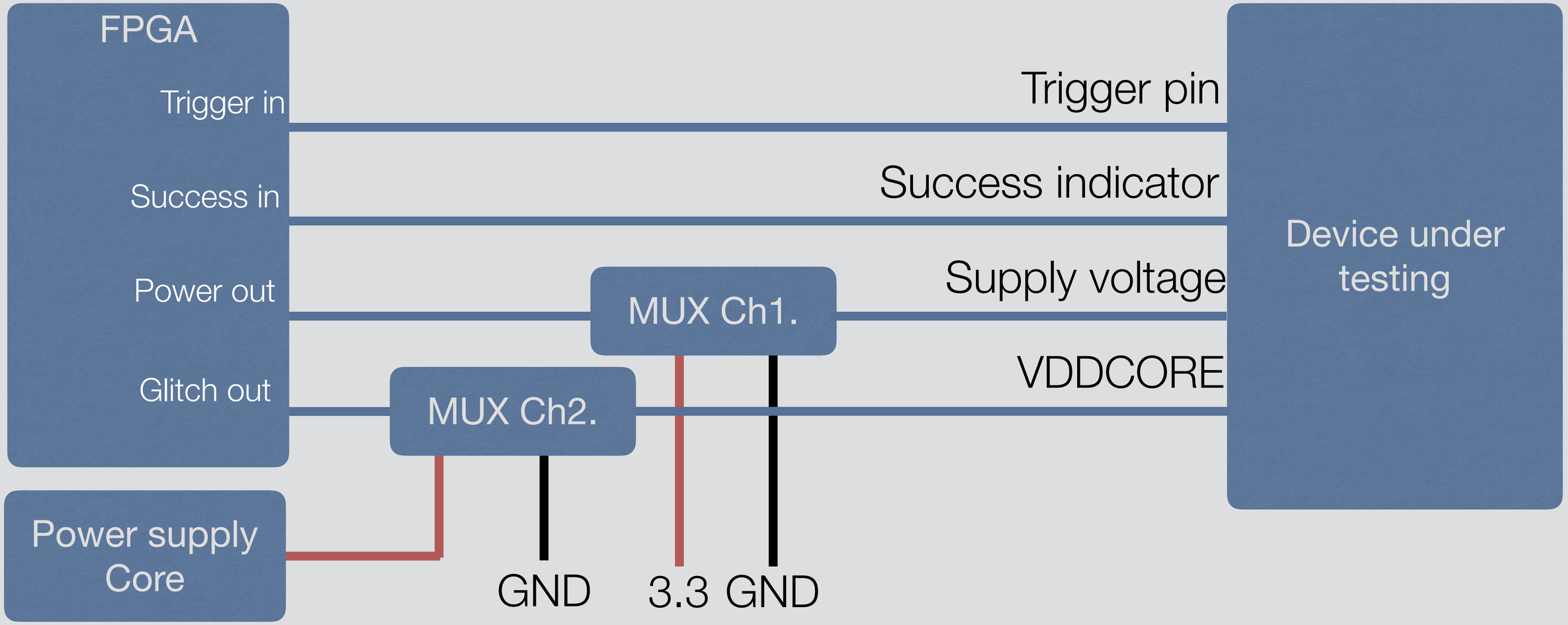
MAX PMOD



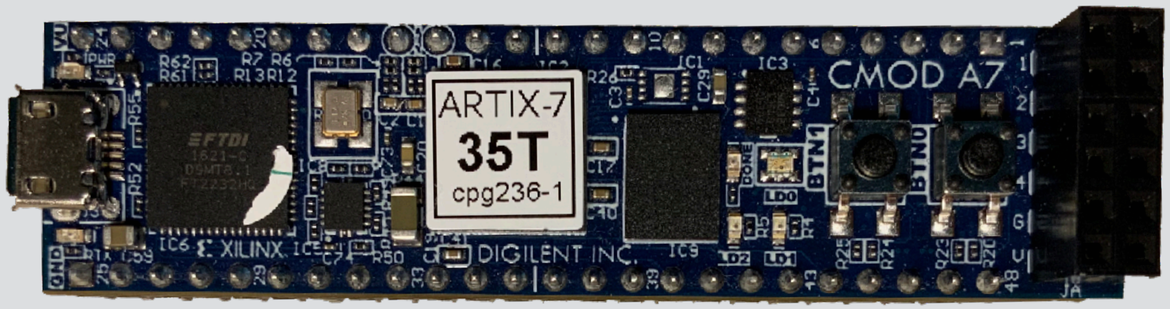
DPS3003/3005/5005

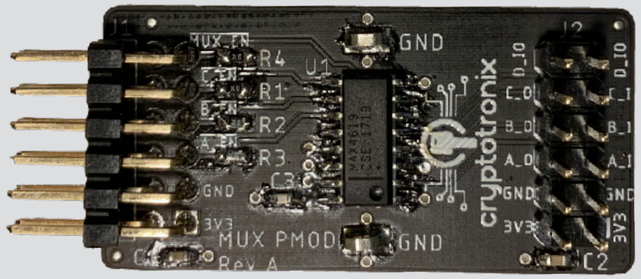
- Cheap, stable power supply
- We use it for suppling VDDCORE
- Can be controlled via UART

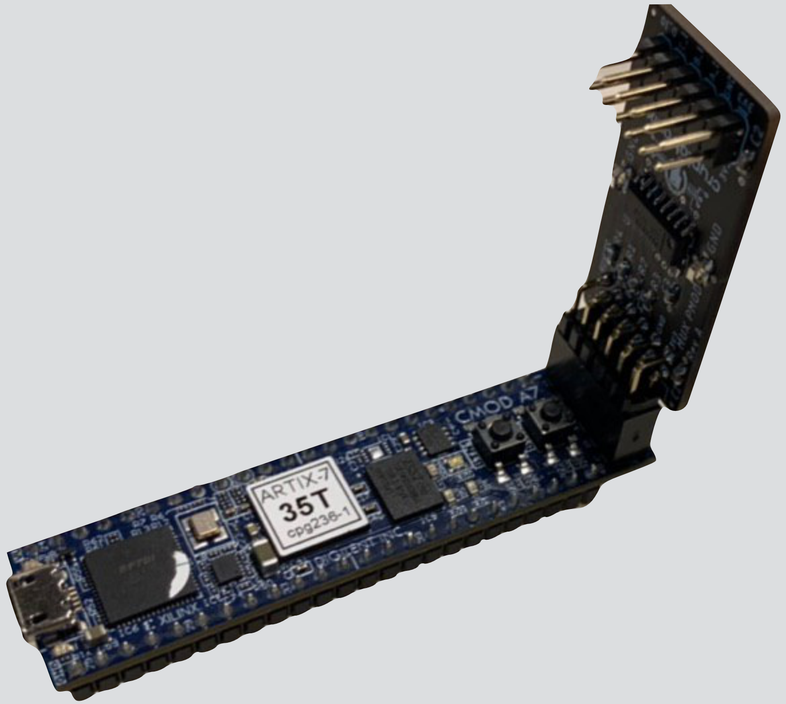
Hooking it up



The glitcher

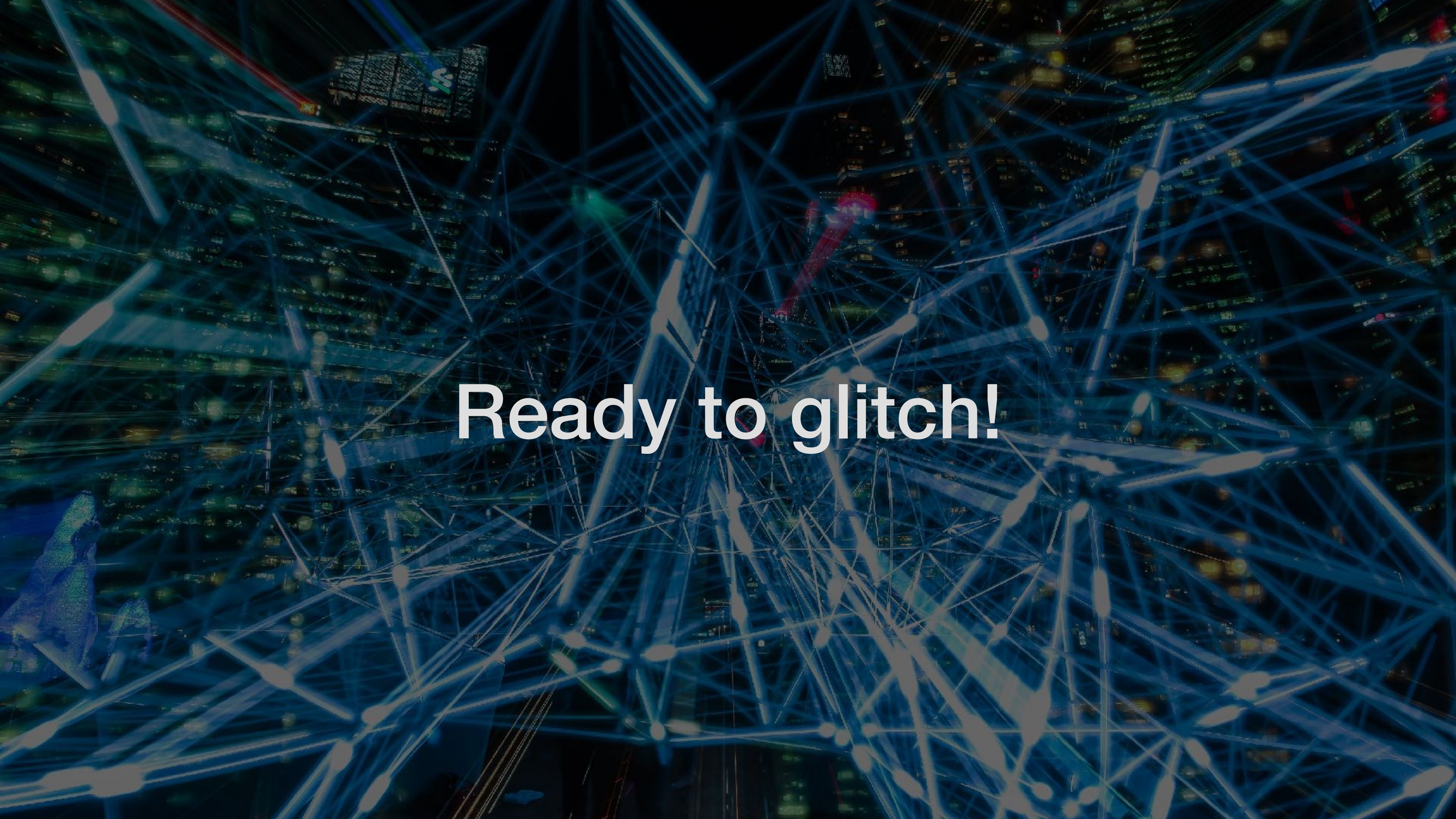






cryptotronix

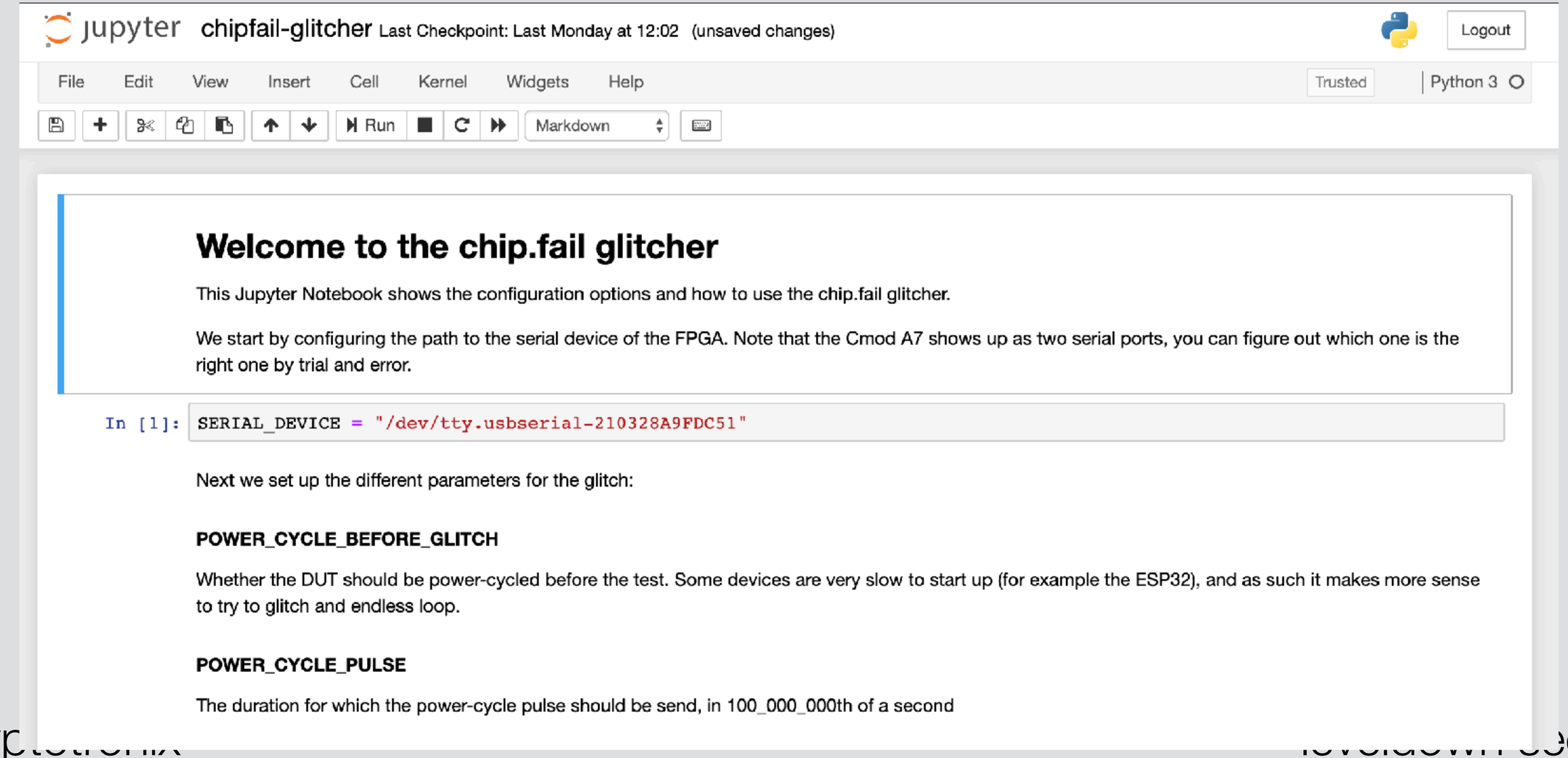
leveldown security


Ready to glitch!

FPGA Bitstream

- Provides a serial interface for setting:
  - Power reset pulse length
  - Delay timing
  - Glitch pulse length

Host control: Jupyter Notebook



Host control: Example glitcher

```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

Host control: Example glitcher

```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

Host control: Example glitcher

```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

Host control: Example glitcher

```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

Host control: Example glitcher

```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

Host control: Example glitcher

```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

Host control: Example glitcher

```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

Host control: Example glitcher

```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

Host control: Example glitcher

```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

Host control: Example glitcher

```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        time.sleep(0.05)

        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

```
Glitch pulse   : 0
Done, if you get here it means everything is working!
```

In [*]:
```
cmd_uint32(device, CMD_SET_POWER_PULSE, POWER_CYCLE_PULSE)


success = False
for delay in tnrange(DELAY_FROM, DELAY_TO):
    cmd_uint32(device, CMD_SET_DELAY, delay)
    if success:
        break
    for pulse in tnrange(PULSE_FROM, PULSE_TO, leave=False):
        cmd_uint32(device, CMD_SET_GLITCH_PULSE, pulse)
        cmd(device, CMD_GLITCH)
        # Loop until the status is == 0, aka the glitch is done.
        # This avoids having to manually time the glitch :)
        while(cmd_read_uint8(device, CMD_GET_STATE)):
            pass
        # Check whether the glitch was successful!
        gpios = cmd_read_uint8(device, CMD_READ_GPIO)
        if(gpios & 0x1):
            print("*** SUCCESS ***")
            print("Delay: " + str(delay))
            print("Pulse: " + str(pulse))
            success = True
            break
```

0% 0/50000 [00:00<?, ?it/s]

2% 2/99 [00:00<00:05, 18.13it/s]

In [48]:
```
# Show status of IOs
print(format(cmd_read_uint8(device, CMD_READ_GPIO), '#010b'))
```

```
0b11111110
```


Testing real targets

Testing real targets

- Should be commonly used in IoT devices
- Should be modern chips, not very outdated ones
- Should be from different vendors

Testing real targets

- Nordic Semiconductor nRF52840
- Espressif Systems ESP32
- Microchip/Atmel SAM L11 (Secure microcontroller)
- STMicroelectronics STM32F2

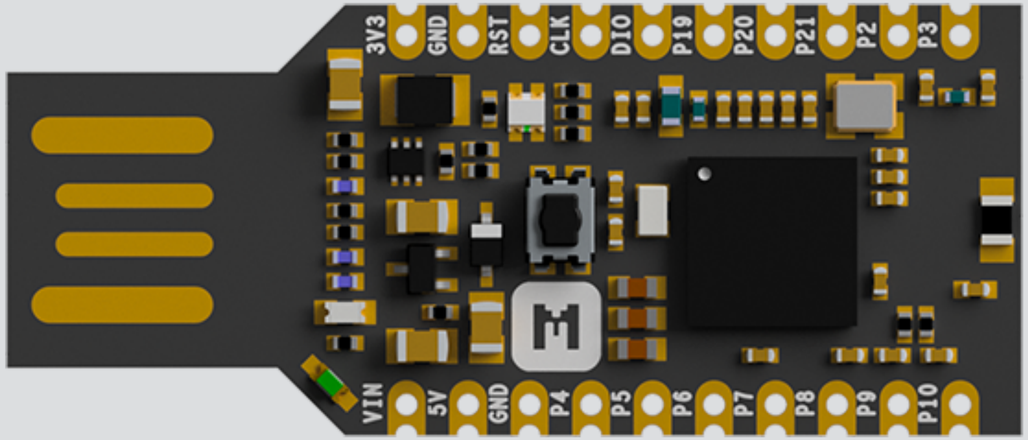
Testing real targets: Goals

- Configure the chips in real-world conditions
- Identify whether a chip is susceptible to glitching attacks
- Test the chips in-situ

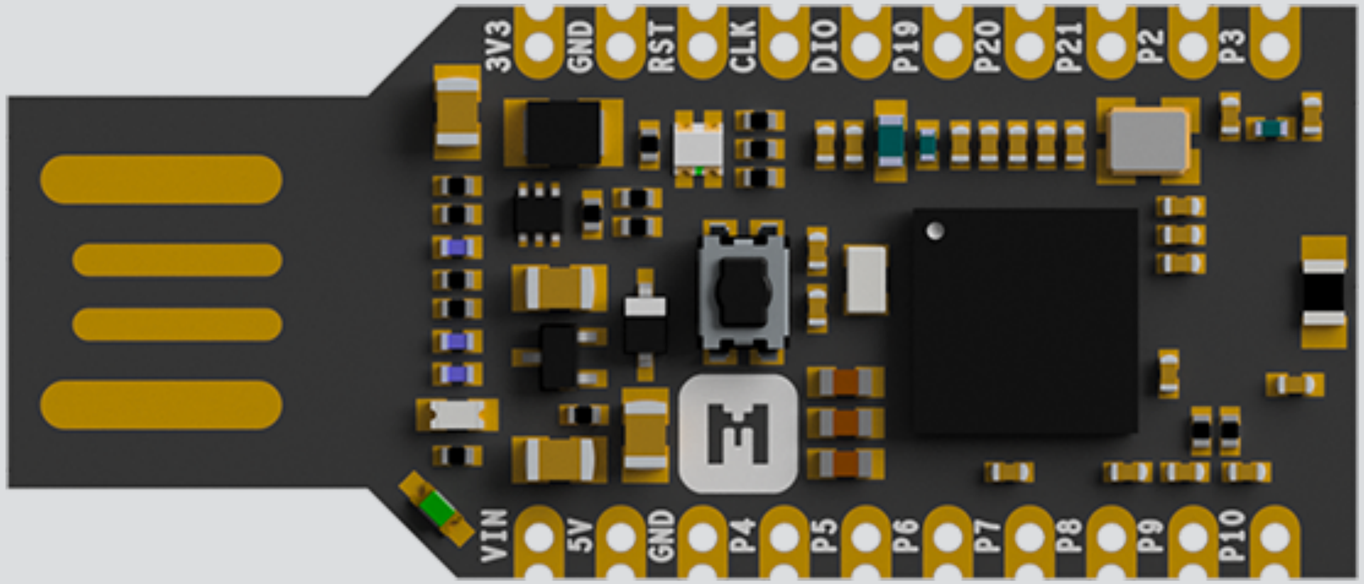

Nordic Semiconductors nRF52840

Nordic Semiconductors nRF52840

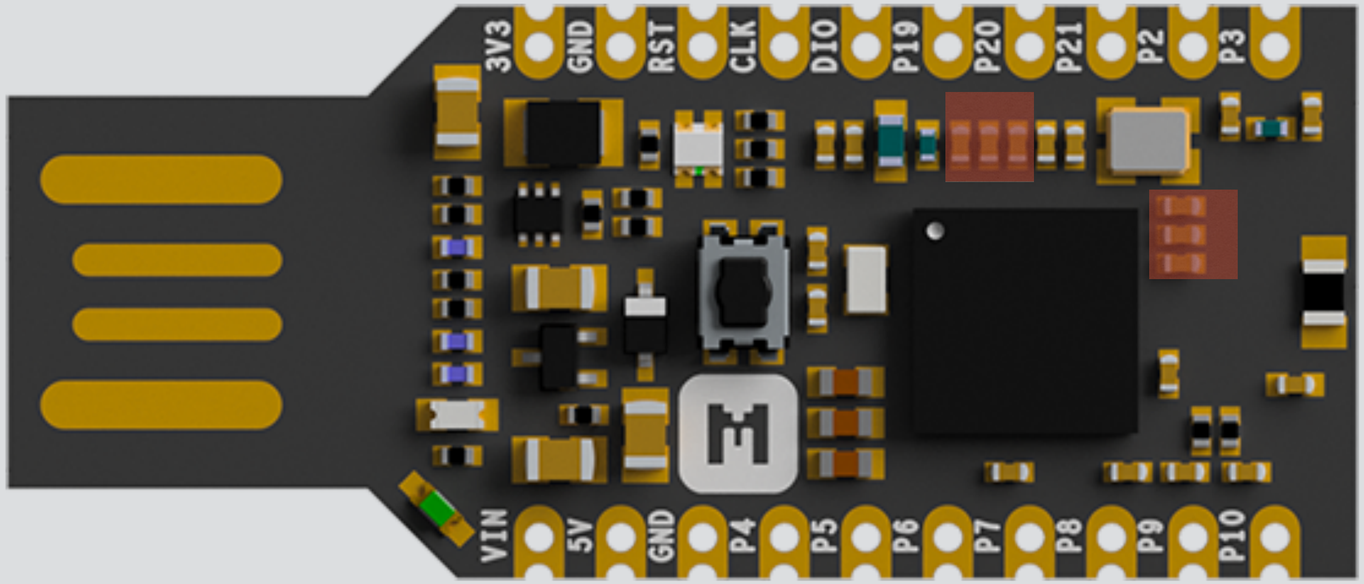
- Multi-protocol 2.4GHz SoC
- Commonly used in smaller IoT sensors
- Target:
  Makerdiary Micro Dev Kit USB Dongle



nRF52840: Target preparation



nRF52840: Target preparation



nRF52840: Target preparation

nRF52840: Test firmware

- Simple test firmware that enables a pin as trigger and reads flash
- Will be released as part of the full chip.fail release

Glitching results

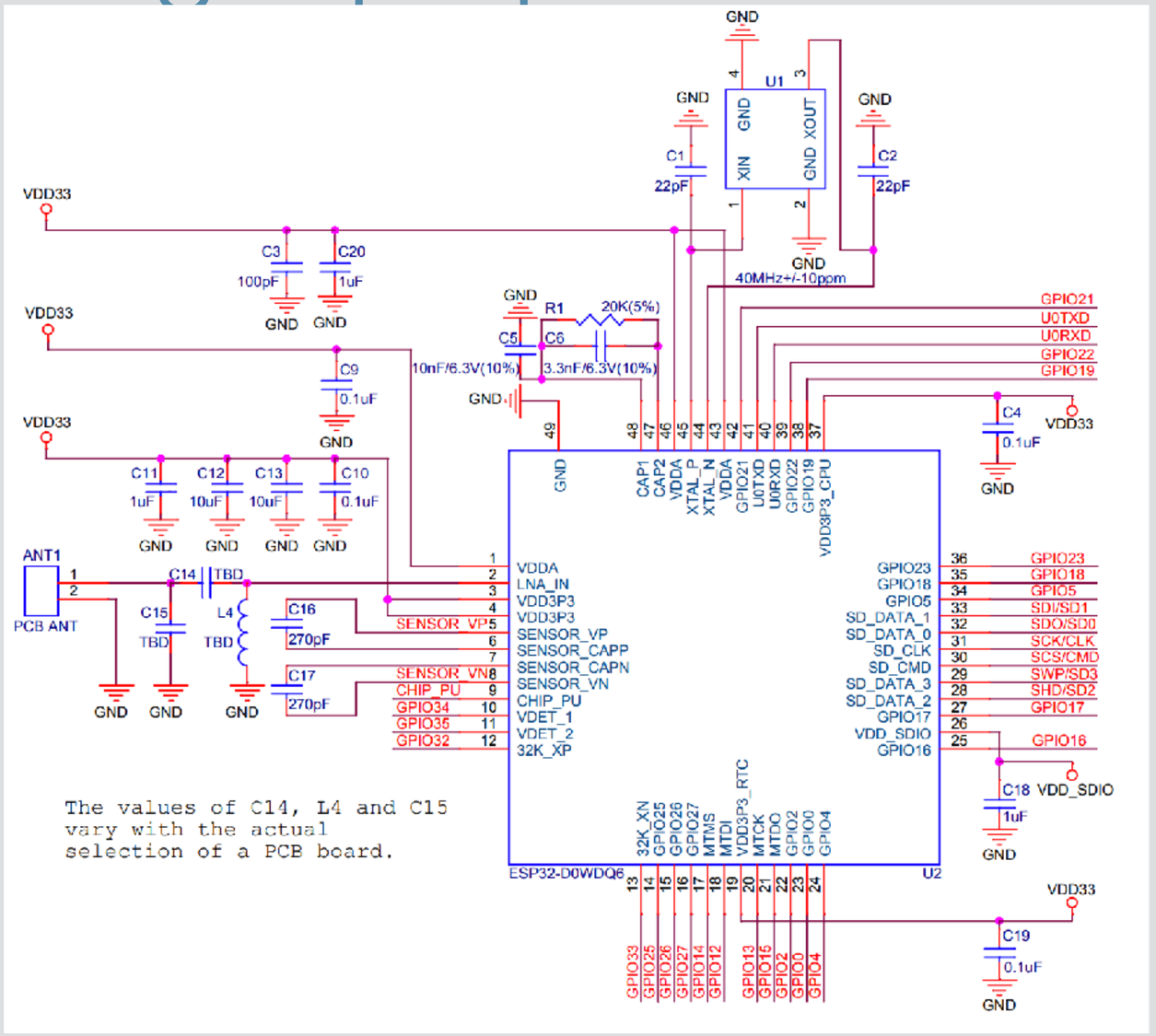
- First success after 1.5h
- Able to glitch application code
- Stable at ~100 attempts for a successful glitch


Espressif Systems ESP32

Espressif Systems ESP32

- Xtensa Architecture, 160 - 240MHz
- Bluetooth & WiFi
- Very commonly found in new IoT products

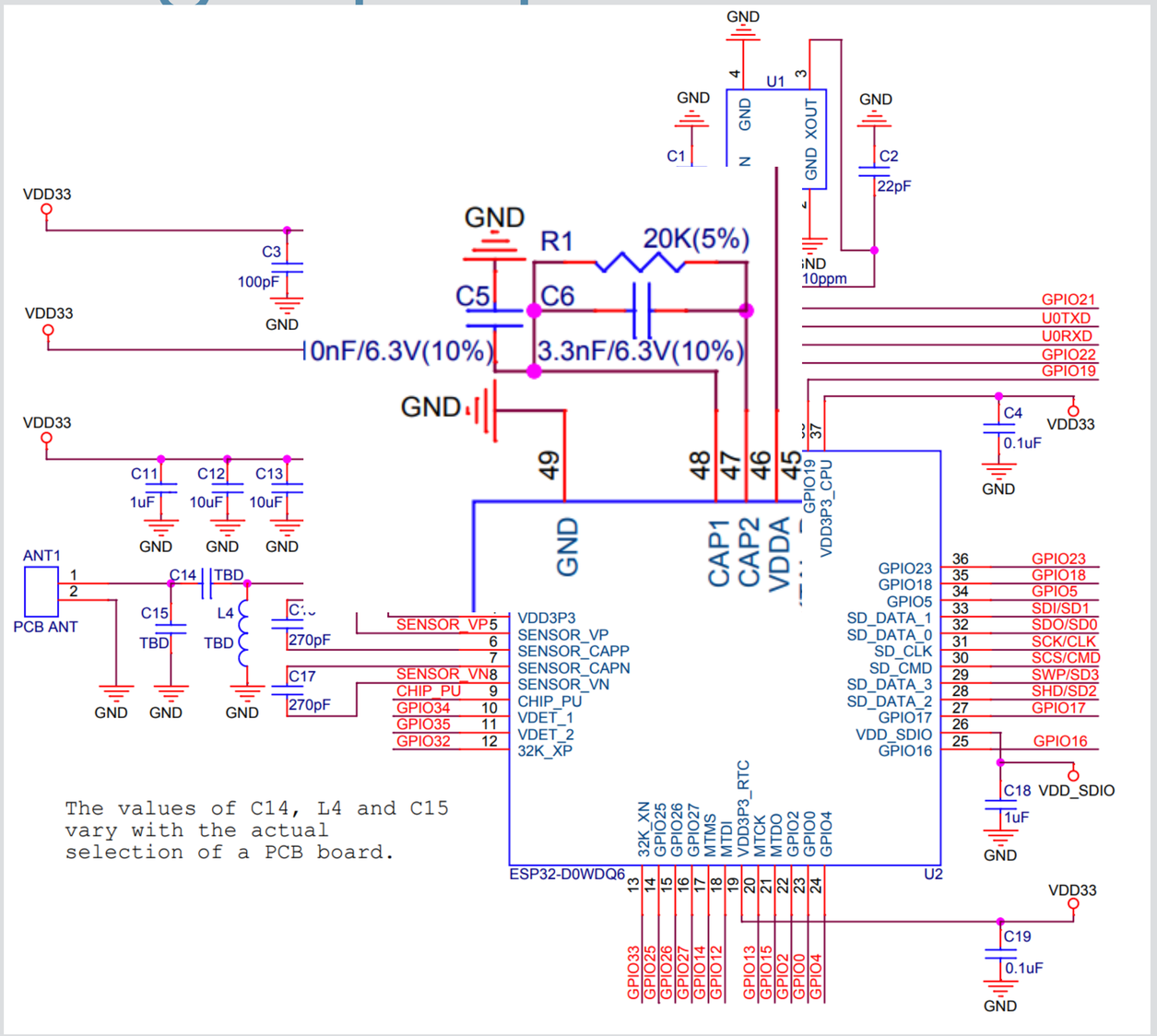
ESP32: Target preparation



cryptotronix

leveldown security

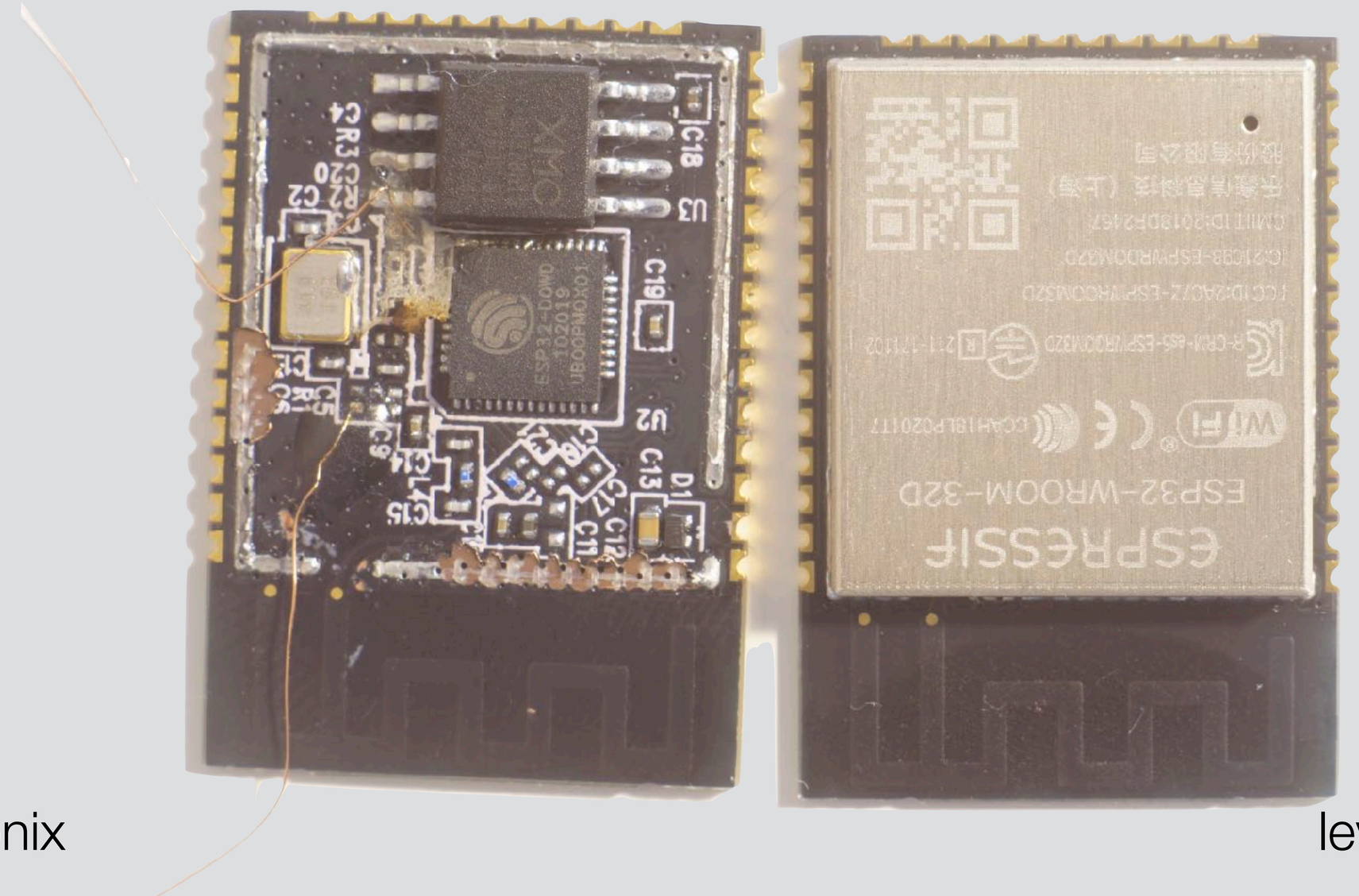
ESP32: Target preparation



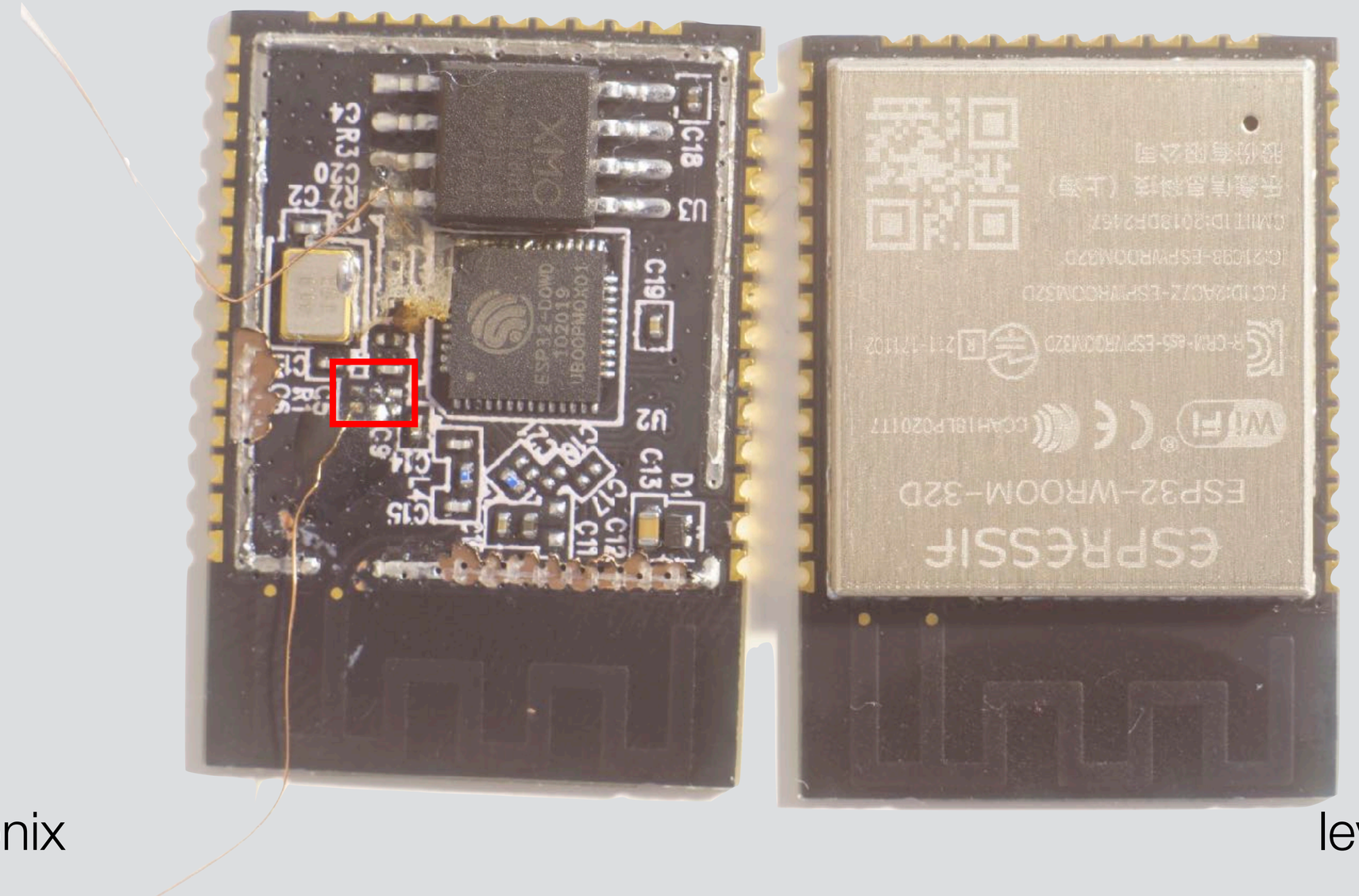
cryptotronix

leveldown security

ESP32: Target preparation



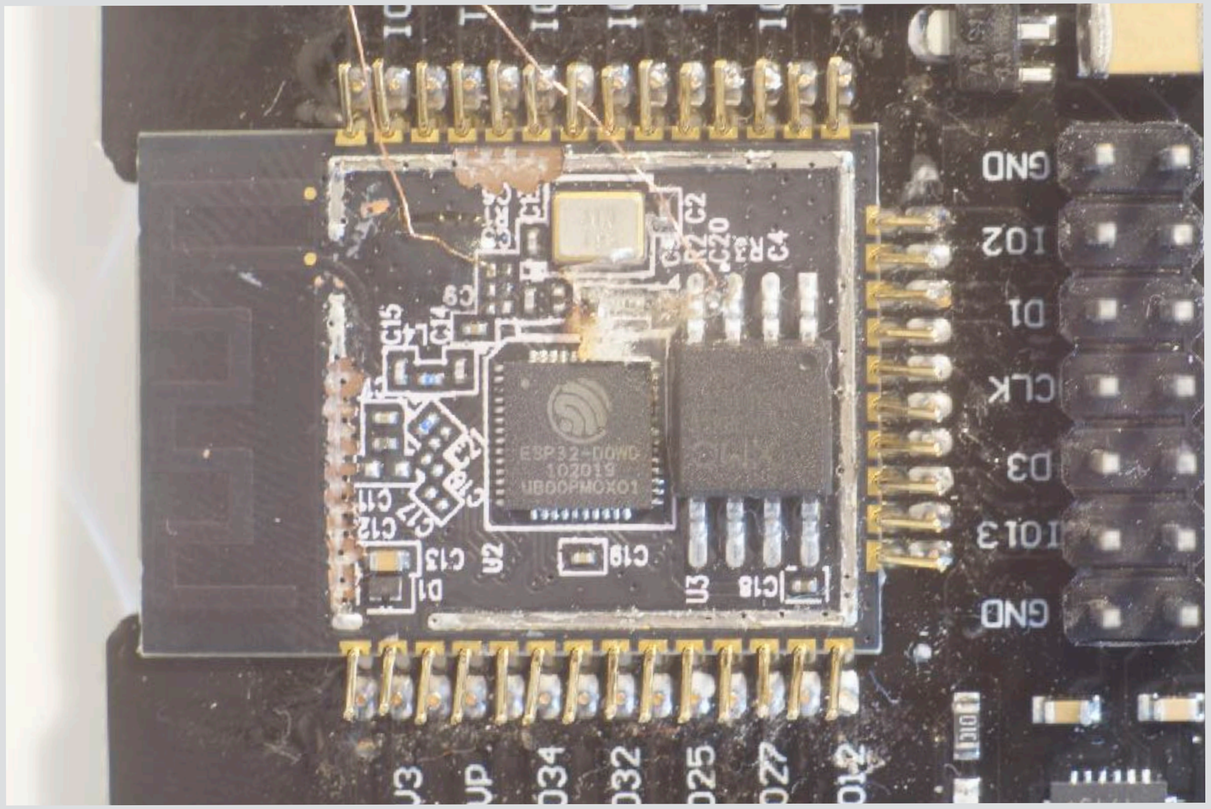
ESP32: Target preparation

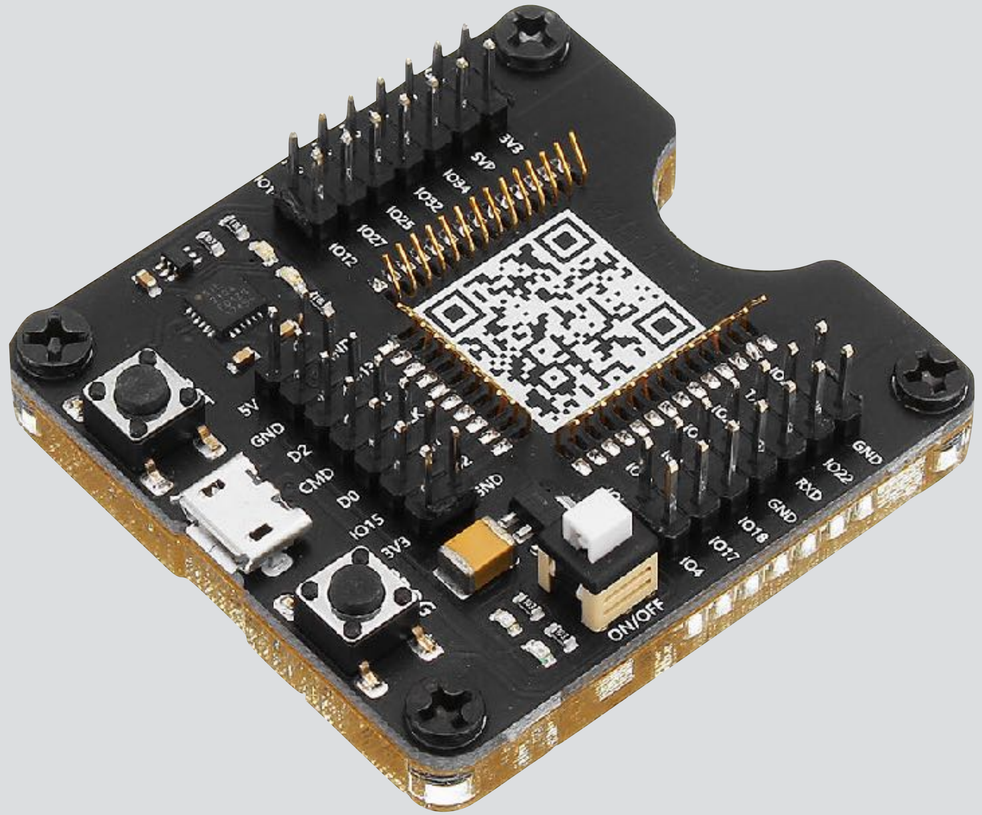


ESP32: Test firmware

- Simple test firmware written in Arduino
- Allows setting different CPU speeds etc
- Very nice for testing

ESP32: Setup





cryptotronix

leveldown security

ESP32: Glitching

- Successful after 3 hours
  - Different clock-speeds, SPI speeds etc
- Had to adjust test firmware to run in a loop - startup too slow
- Stable within ~10’000 attempts

Microchip's SAM L11

World-Class, Award-Winning SAM L10 and SAM L11 Microcontroller Family

Industry's Lowest Power 32-bit MCUs, First to Offer Chip-Level Security and Arm® TrustZone® Technology

World-Class, Award-Winning SAM L10 and SAM L11 Microcontroller Family

Industry's Lowest Power 32-bit MCUs, First to Offer Chip-Level Security and Arm® TrustZone® Technology

World-Class, Award-Winning SAM L10 and SAM L11 Microcontroller Family

Industry's Lowest Power 32-bit MCUs, First to Offer Chip-Level Security and Arm® TrustZone® Technology

World-Class, Award-Winning SAM L10 and SAM L11 Microcontroller Family

Industry's Lowest Power 32-bit MCUs, First to Offer Chip-Level Security and Arm® TrustZone® Technology

How secure is it?

6.3 Power-On Reset and Brown-Out Detectors

The SAM L10/L11 embed three features to monitor, warn and reset the device:

- A Power-on Reset (POR) on V_{DD} (VDDANA and VDDIO):
  - Monitoring is always activated, including during device startup or during any sleep modes.
  - Having V_{DD} below a fixed threshold voltage will reset the whole device.

  Note: Refer to 46.11.2 Power-On Reset (POR) Characteristics for the rising and falling threshold voltages.
- A Brown-out Detector (BOD33) on V_{DD} (VDDANA and VDDIO):
  - The BOD33 can monitor VDD continuously (continuous mode) or periodically (sampled mode) with a programmable sample frequency in active mode as in any sleep modes.
  - A programmable threshold loaded from the NVM User Row is used to trigger an interrupt and/or reset the whole device.
- A Brown-out Detector (BOD12) on VDDCORE.
  Note: BOD12 is calibrated in production and its calibration parameters are stored in the NVM User Row. These data must not be changed to ensure correct device behavior.

cryptotronix

leveldown security

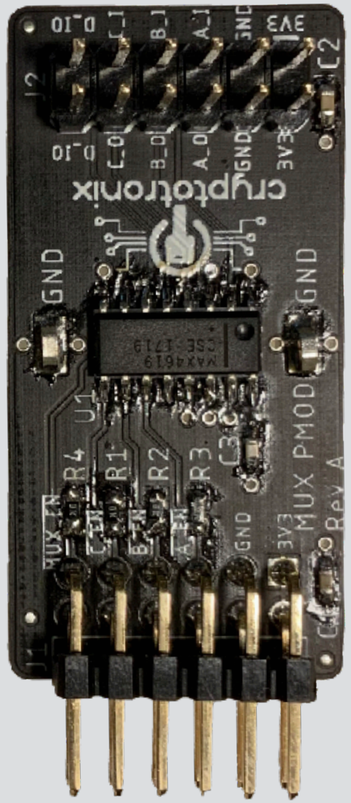
Can we glitch it?

Glitching the SAM L11

- Super susceptible to voltage glitching
- Success after literally 5 minutes
- Bypass of the secure reference boot loader - disclosure in progress


Just HOW easy is it to glitch?

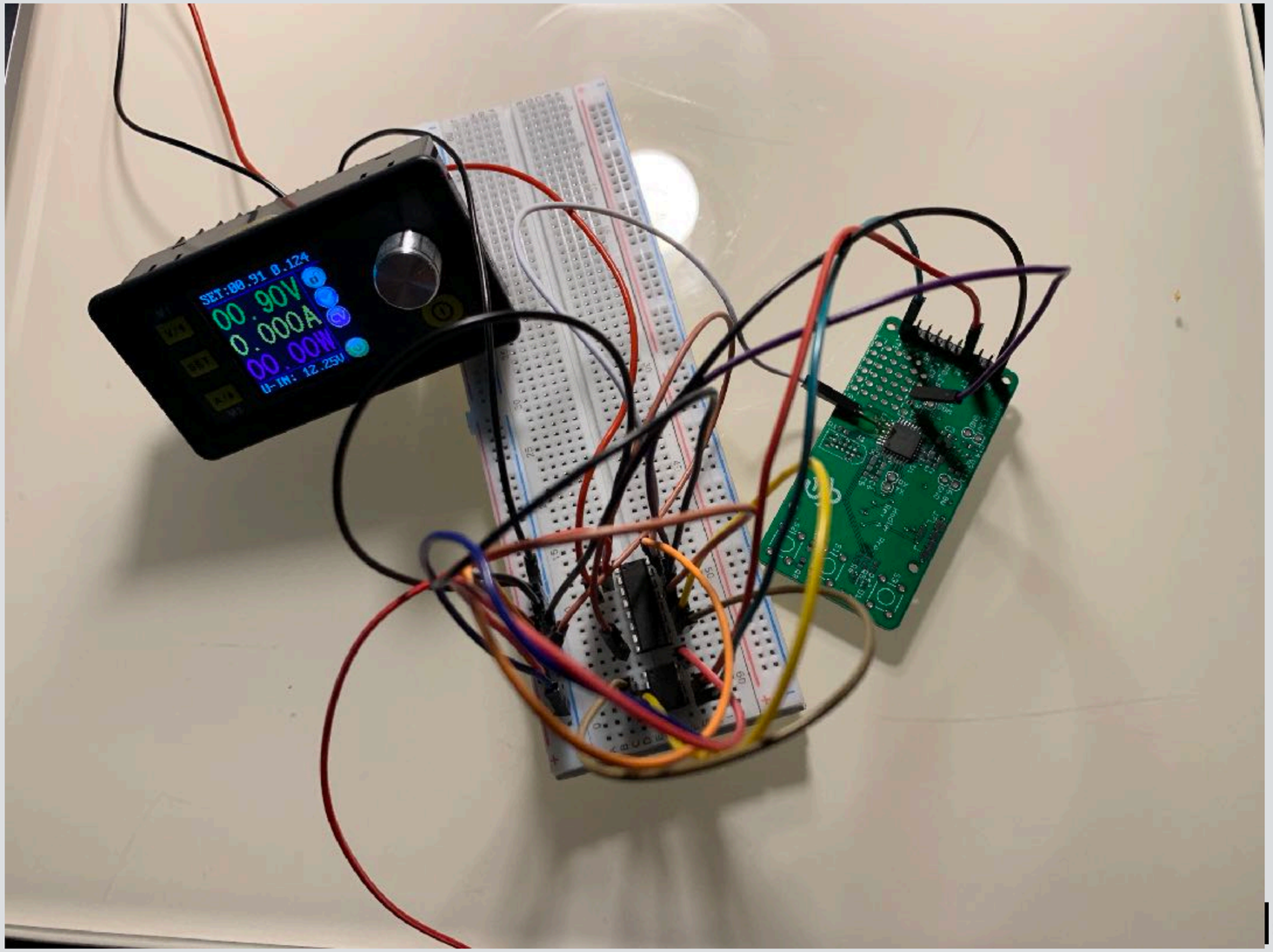
The 5$ Glitcher...



cryptotronix

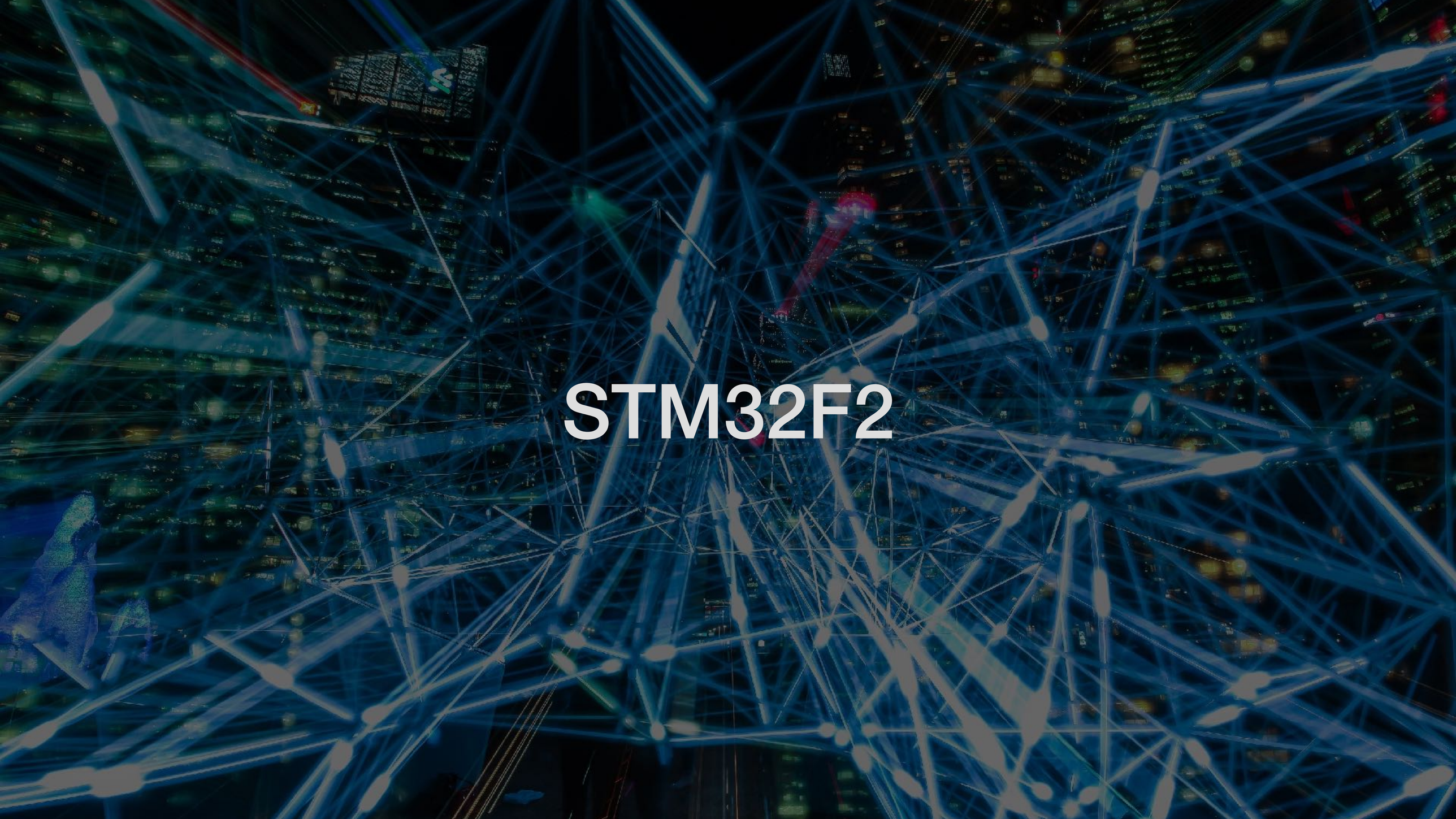
leveldown security

Glitching the SAM L11.. For $5



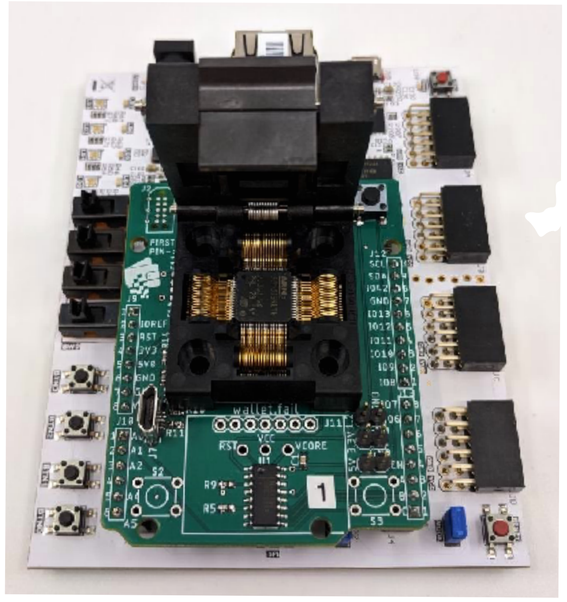
cryptotronix

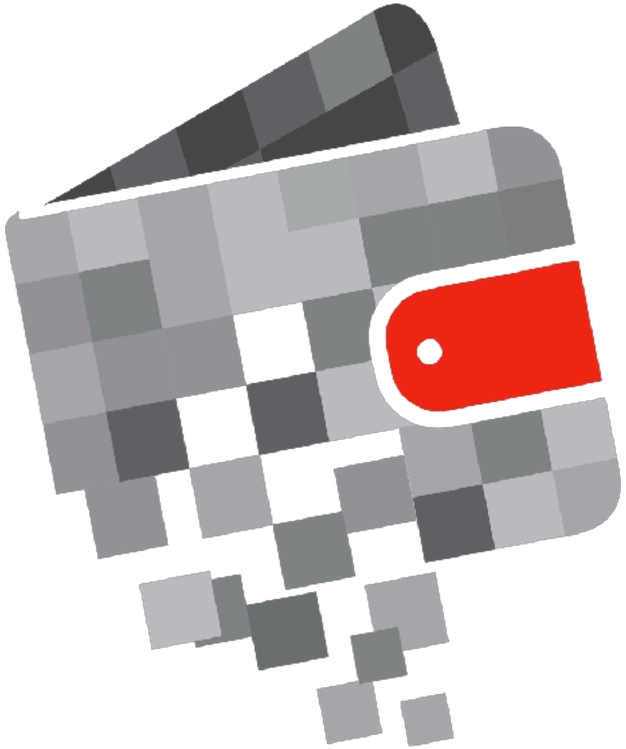
leveldown security

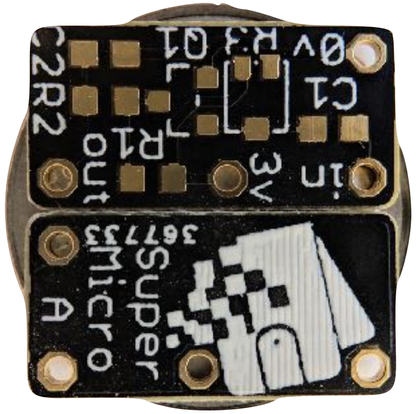

STM32F2

STM32F2

The million dollar microcontroller

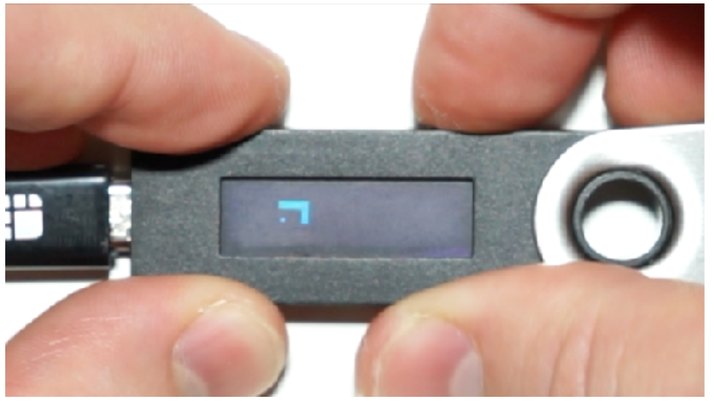






WALLET.FAIL

Poof goes your crypto…

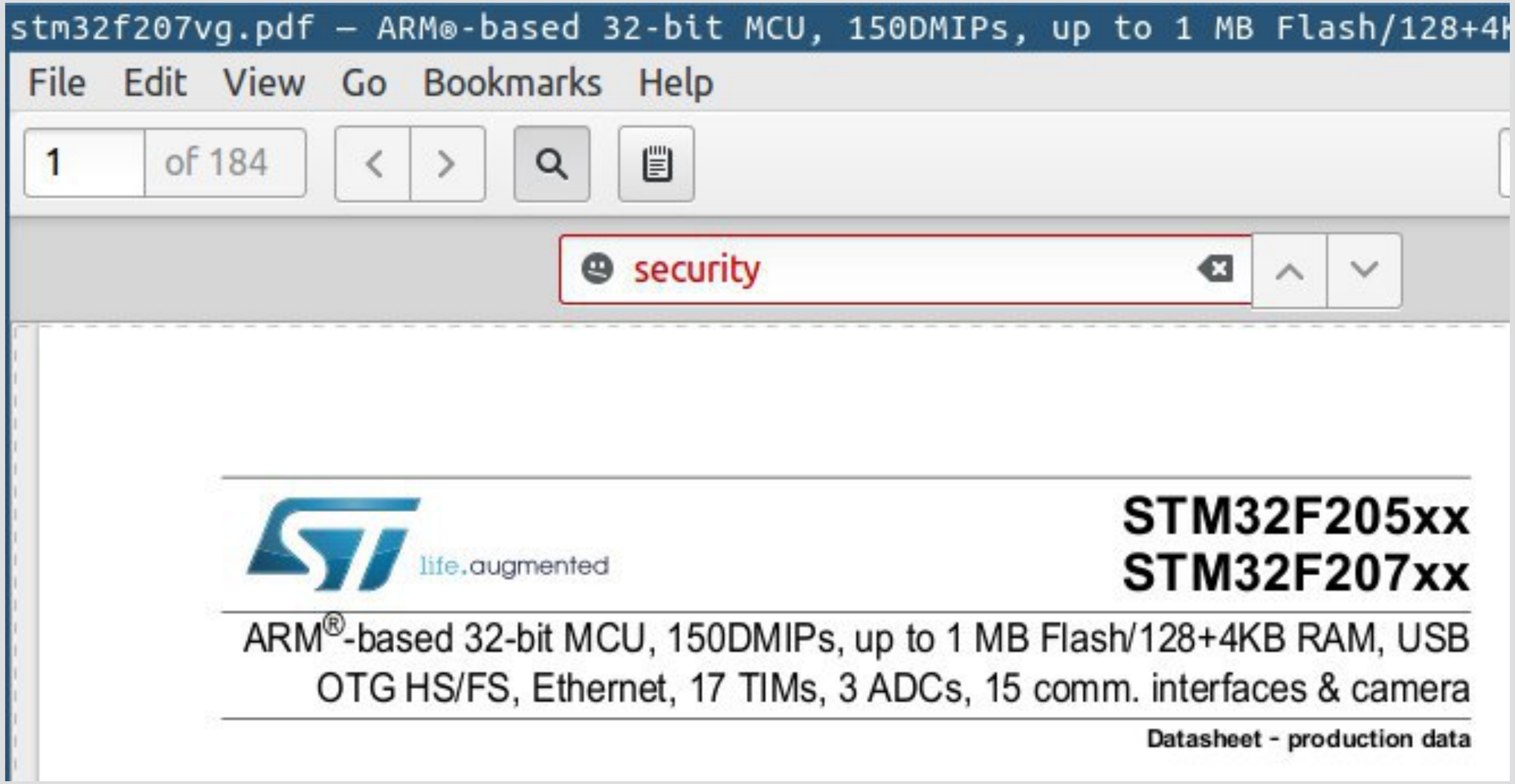


Previous work

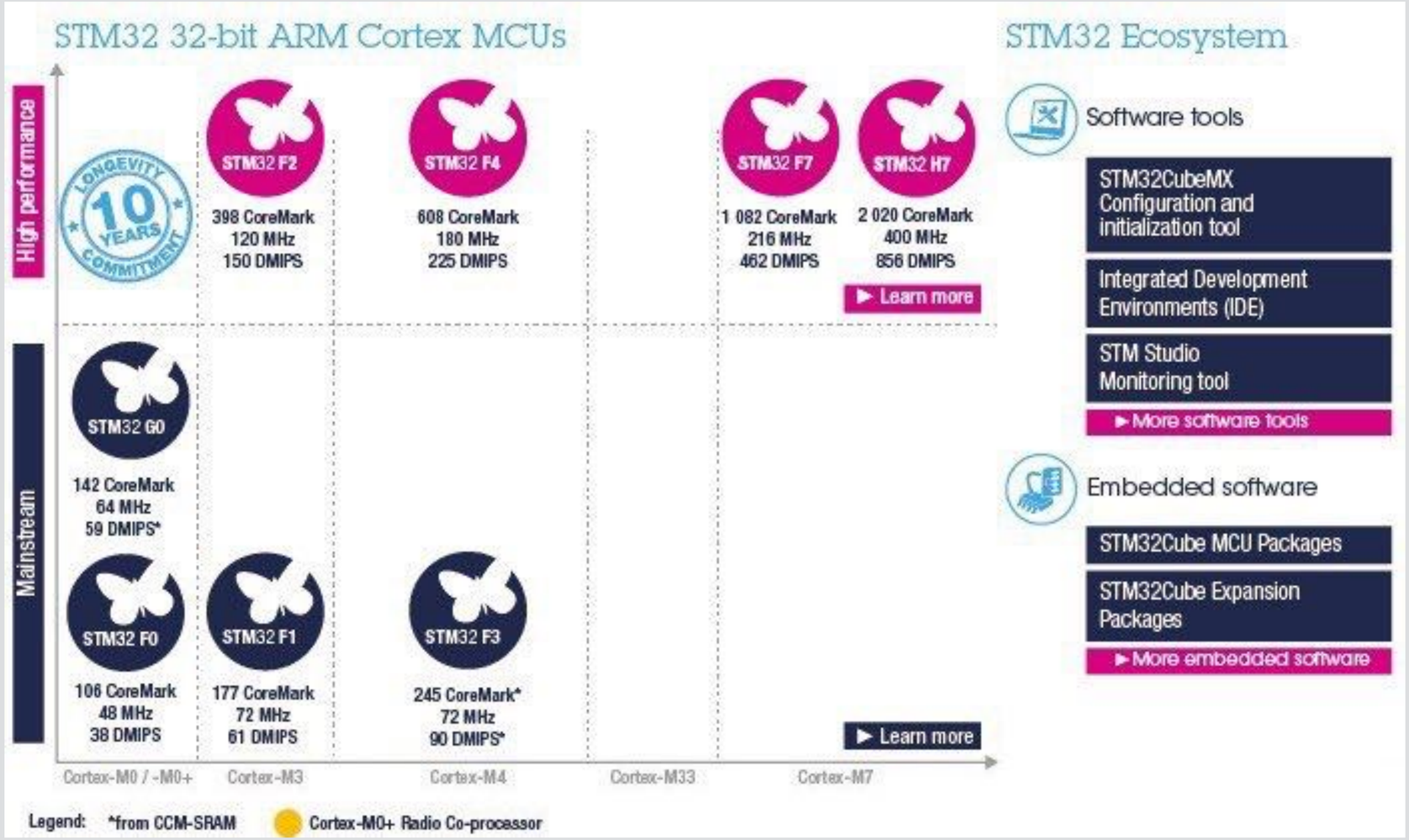
- STM32F0: JTAG bug + RDP2 -> RDP1 Downgrade
  "*Shedding too much Light on a Microcontroller's Firmware Protection*"
  https://www.usenix.org/system/files/conference/woot17/woot17-paper-obermaier.pdf
- STM32F1/STM32F3: Shaping the Glitch
  https://tches.iacr.org/index.php/TCHES/article/download/7390/6562/
- How to apply it all? Read:
  "*Verifying Code Readout Protection Claims*"
  http://circuitcellar.com/cc-blog/verifying-code-readout-protection-claims/

STM32F2

- General purpose microcontroller
- ARM Cortex-M3
- Used particularly often in Bitcoin/Crypto wallets....


stm32f207vg.pdf – ARM®-based 32-bit MCU, 150DMIPs, up to 1 MB Flash/128+4K
File Edit View Go Bookmarks Help
1
of 184
security
life.augmented
STM32F205xx
STM32F207xx
ARM®-based 32-bit MCU, 150DMIPs, up to 1 MB Flash/128+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera
Datasheet - production data

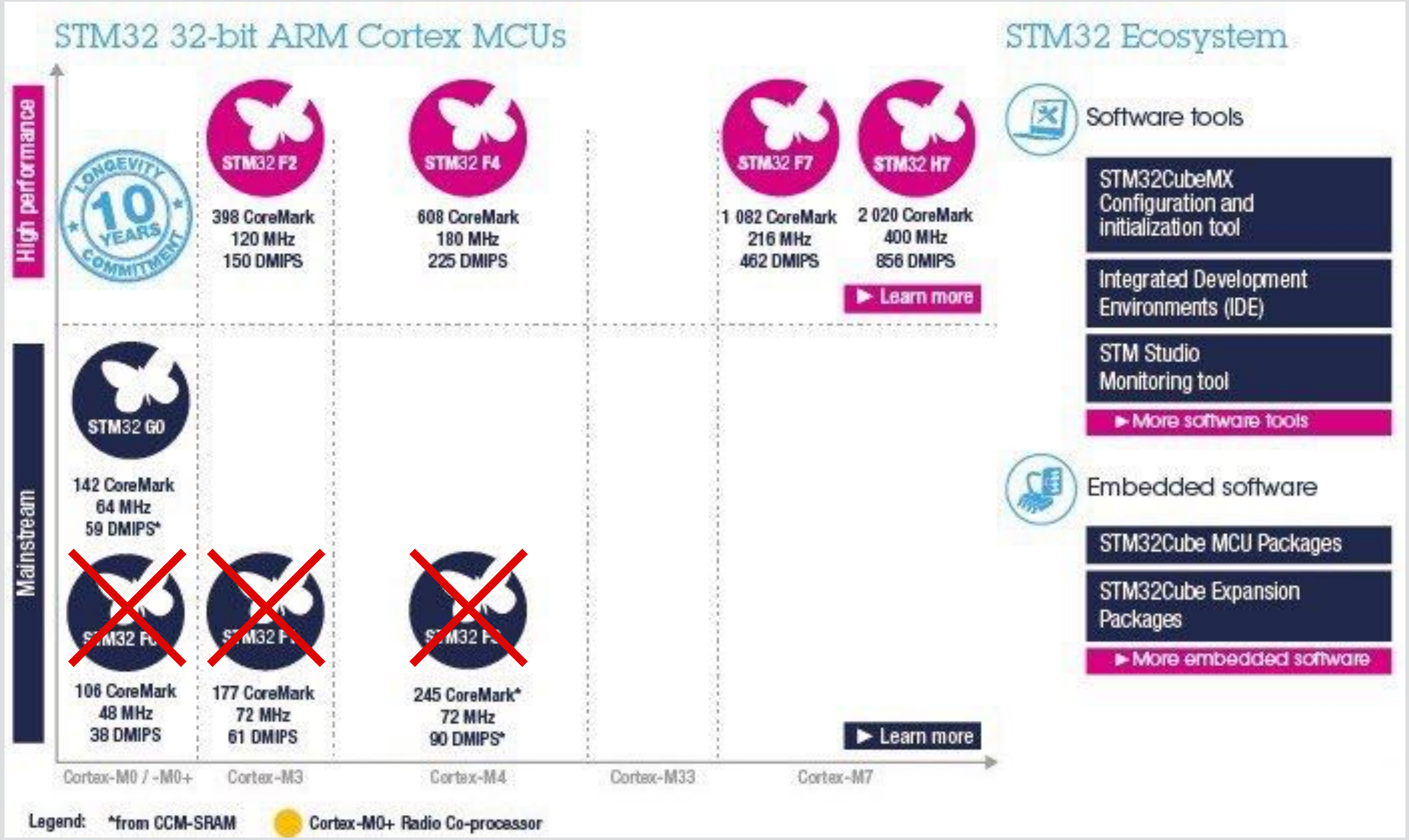
STM32F2



cryptotronix

leveldown security

STM32F2



cryptotronix

leveldown security

STM32F2

cryptotronix

leveldown security

STM32 Read-out Protection (RDP)

RDP	Value	Behavior
RDP0	0xAA	Full Access
RDP1	NOT 0xAA or 0xCC	Ability to read RAM + Regs, no flash access, no single stepping
RDP2	0xCC	No access

STM32 Read-out Protection (RDP)

RDP	Value	Behavior
RDP0	0xAA	Full Access
RDP1	NOT 0xAA or 0xCC	Ability to read RAM + Regs, no flash access, no single stepping
RDP2	0xCC	No access

Dumping the bootrom

- The STM32F2 bootrom is readable
- Dumped using openocd, reverse engineered
- Trying to find the RDP2 (0xCC) check

RDP	Value
RDP0	0xAA
RDP1	NOT 0xAA or 0xCC
RDP2	0xCC

Dumping the bootrom

- The STM32F2 bootrom is readable
- Dumped using openocd, reverse engineered
- Trying to find the RDP2 (0xCC) check

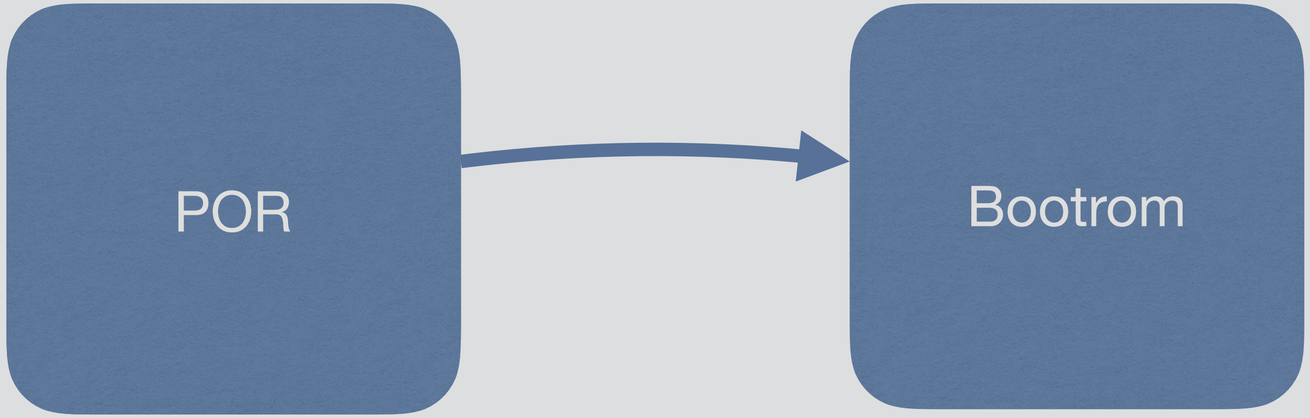
No check found, all checks only
for RDP0 (0xAA)

RDP	Value
RDP0	0xAA
RDP1	NOT 0xAA or 0xCC
RDP2	0xCC


Let's apply our methodology

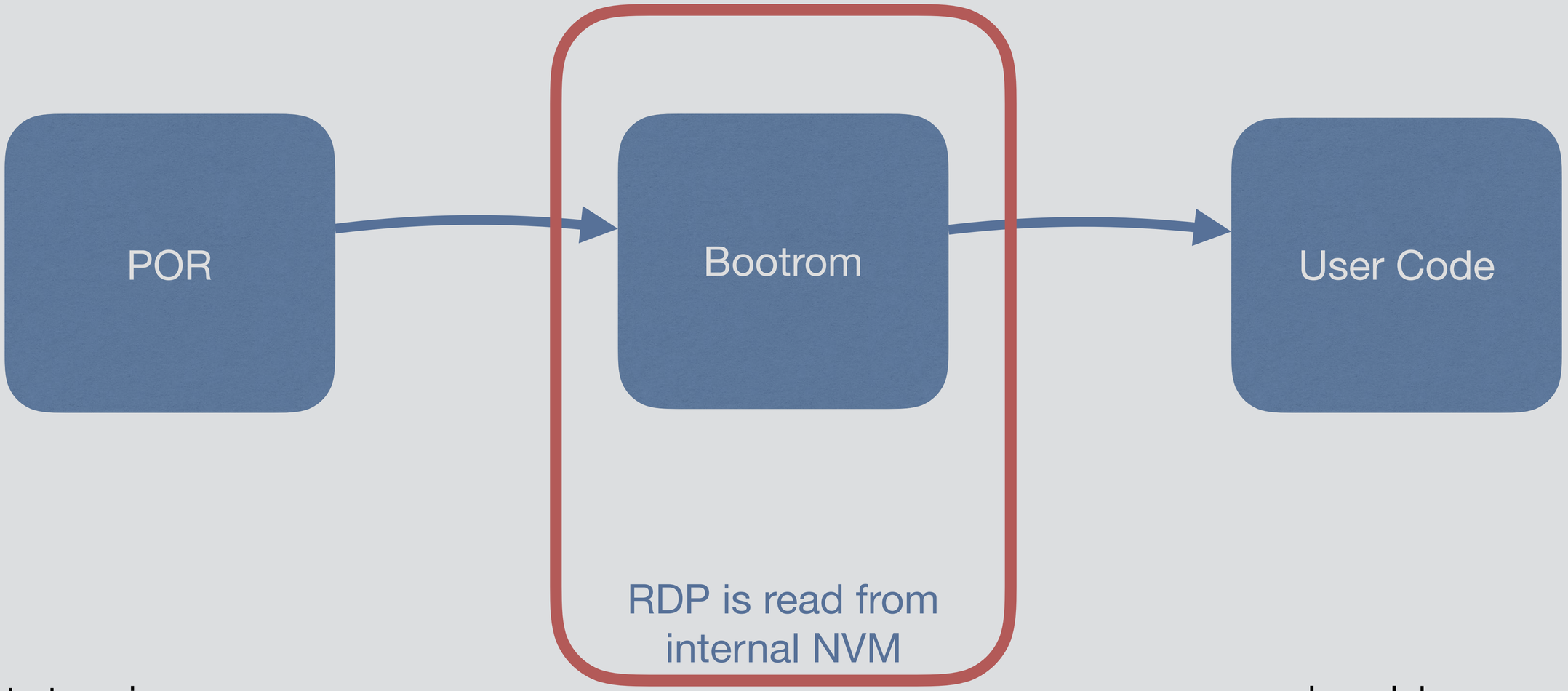
Bootrom Glitching

Bootrom Glitching

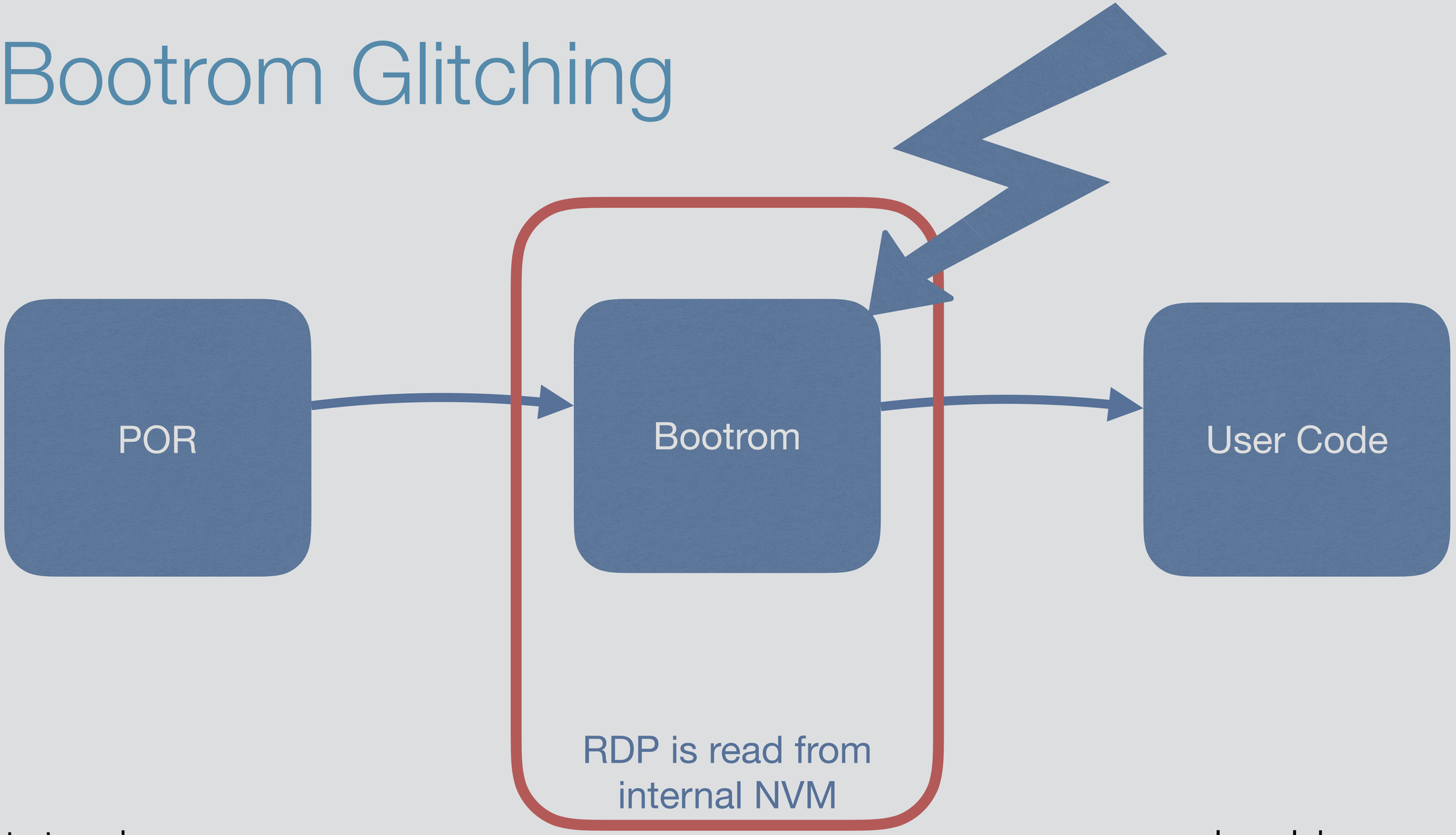


Bootrom Glitching

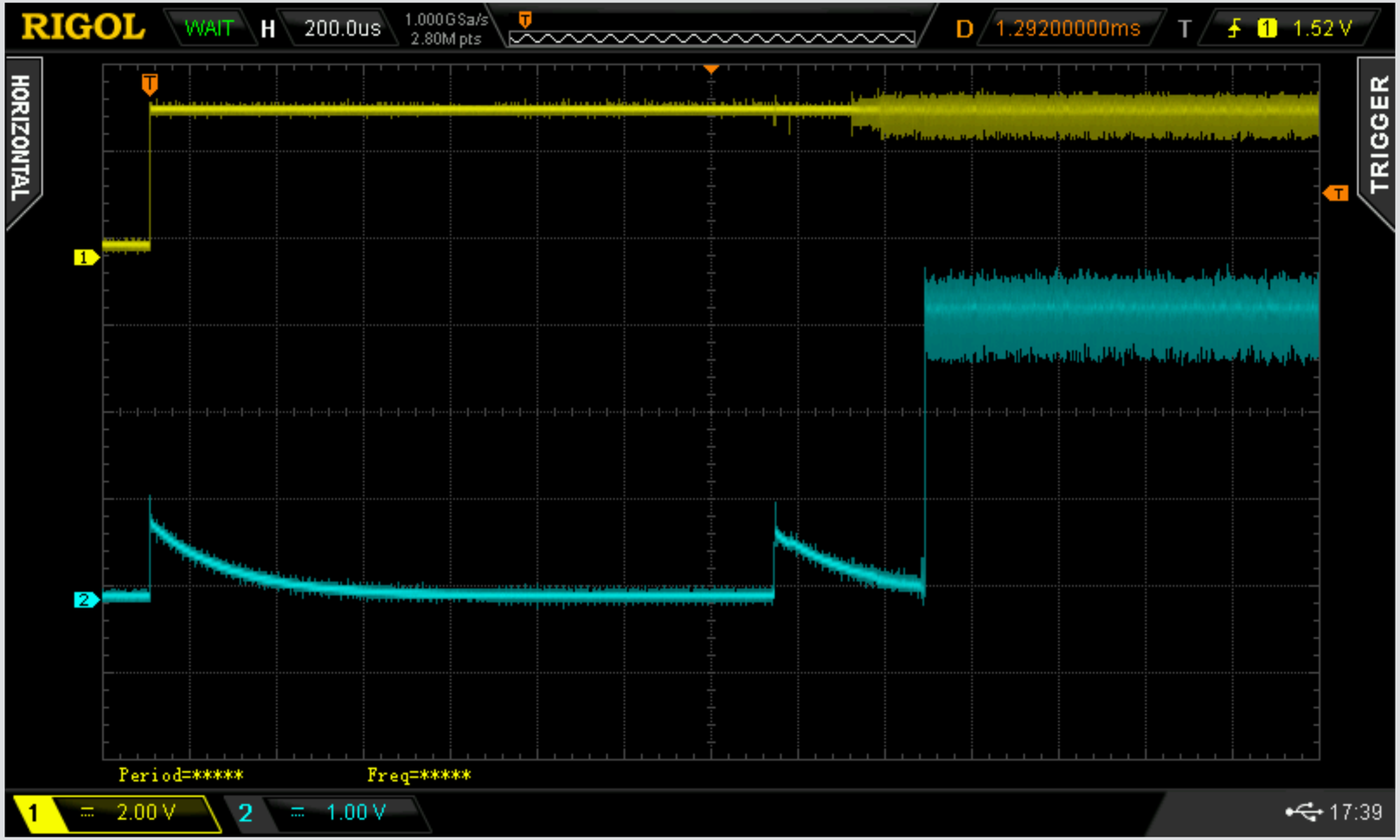
Bootrom Glitching



Bootrom Glitching



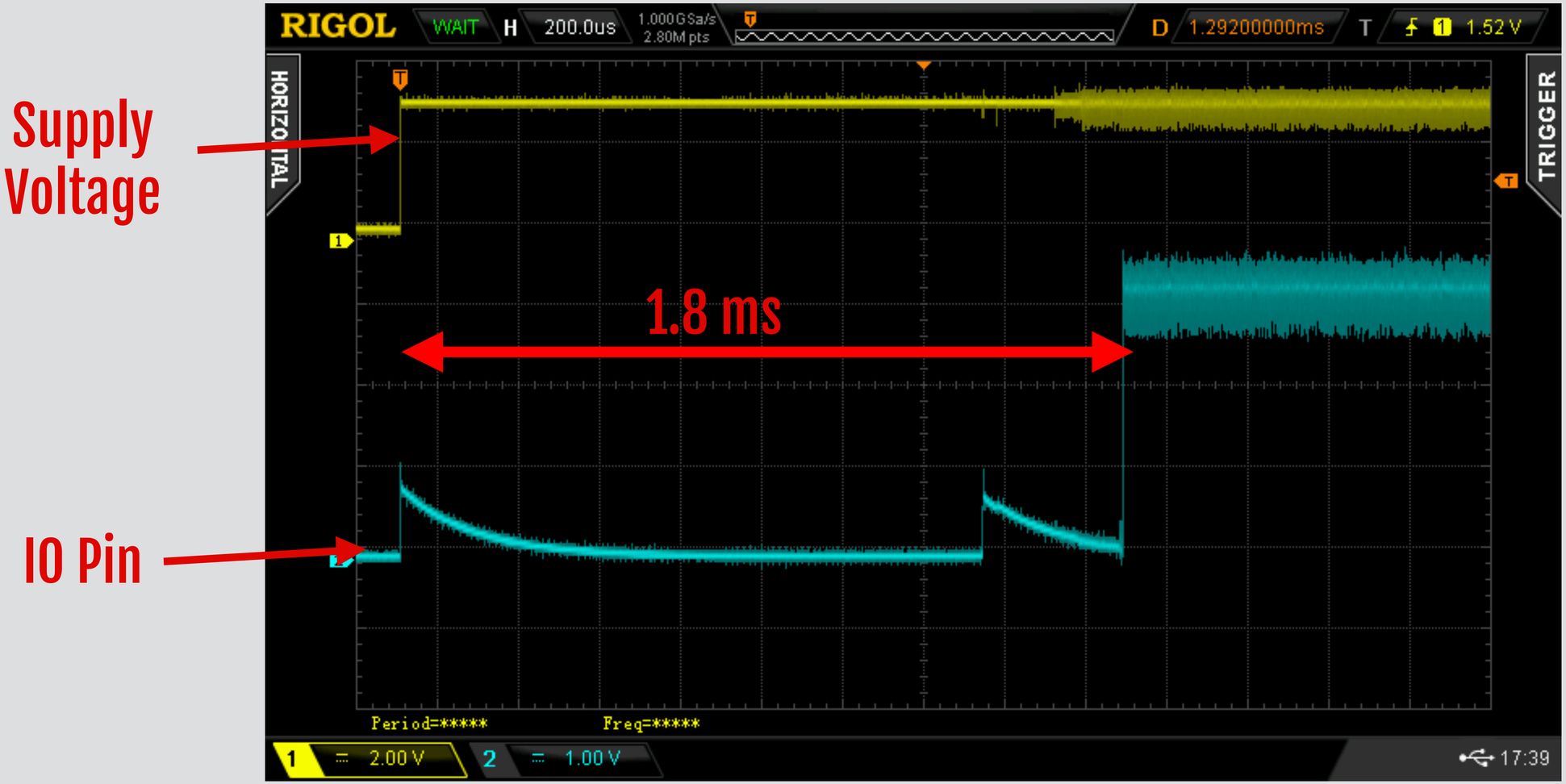
STM32F2 Boot process (1.8ms)



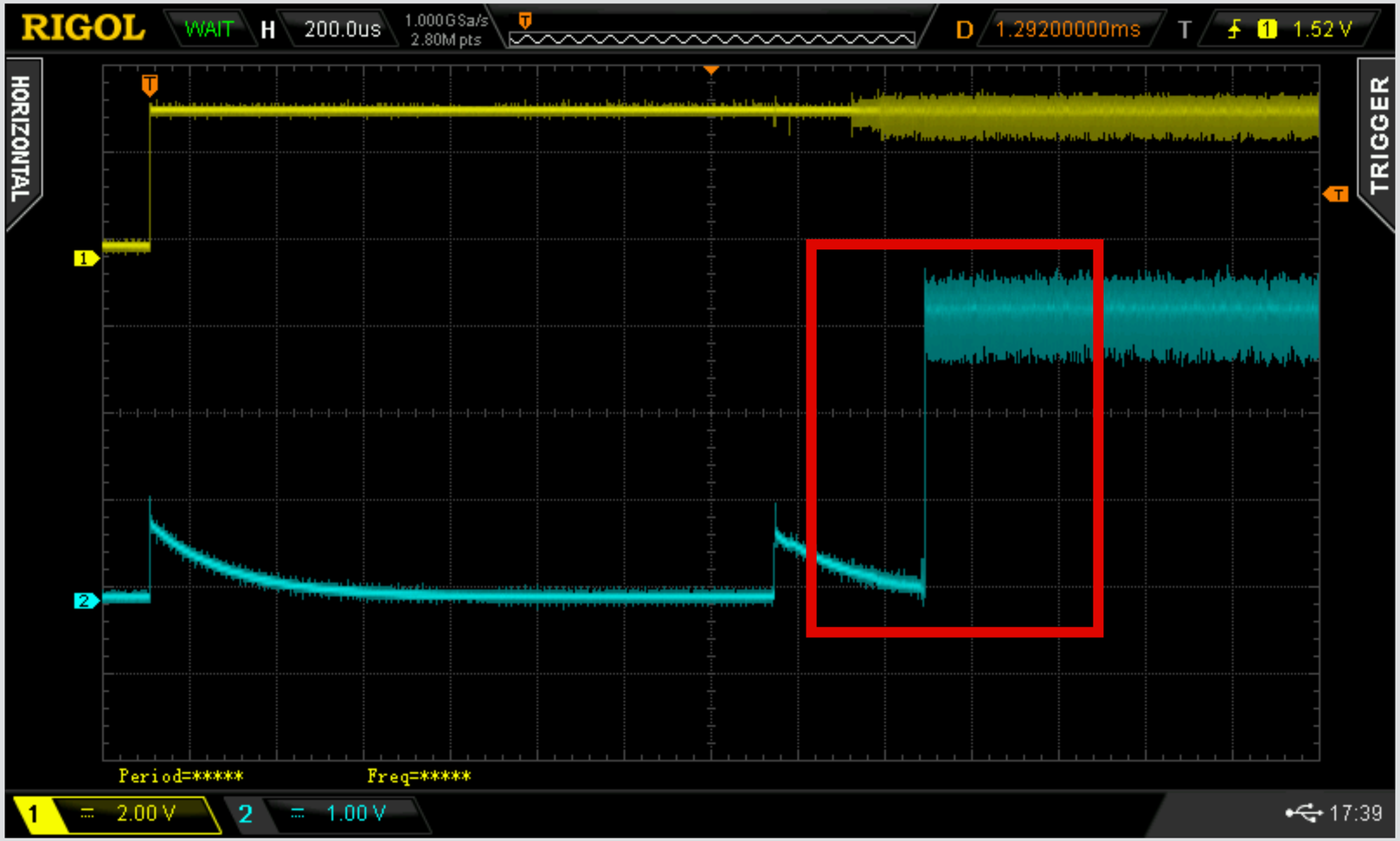
cryptotronix

leveldown security

STM32F2 Boot process (1.8ms)



STM32F2 Boot process (1.8ms)



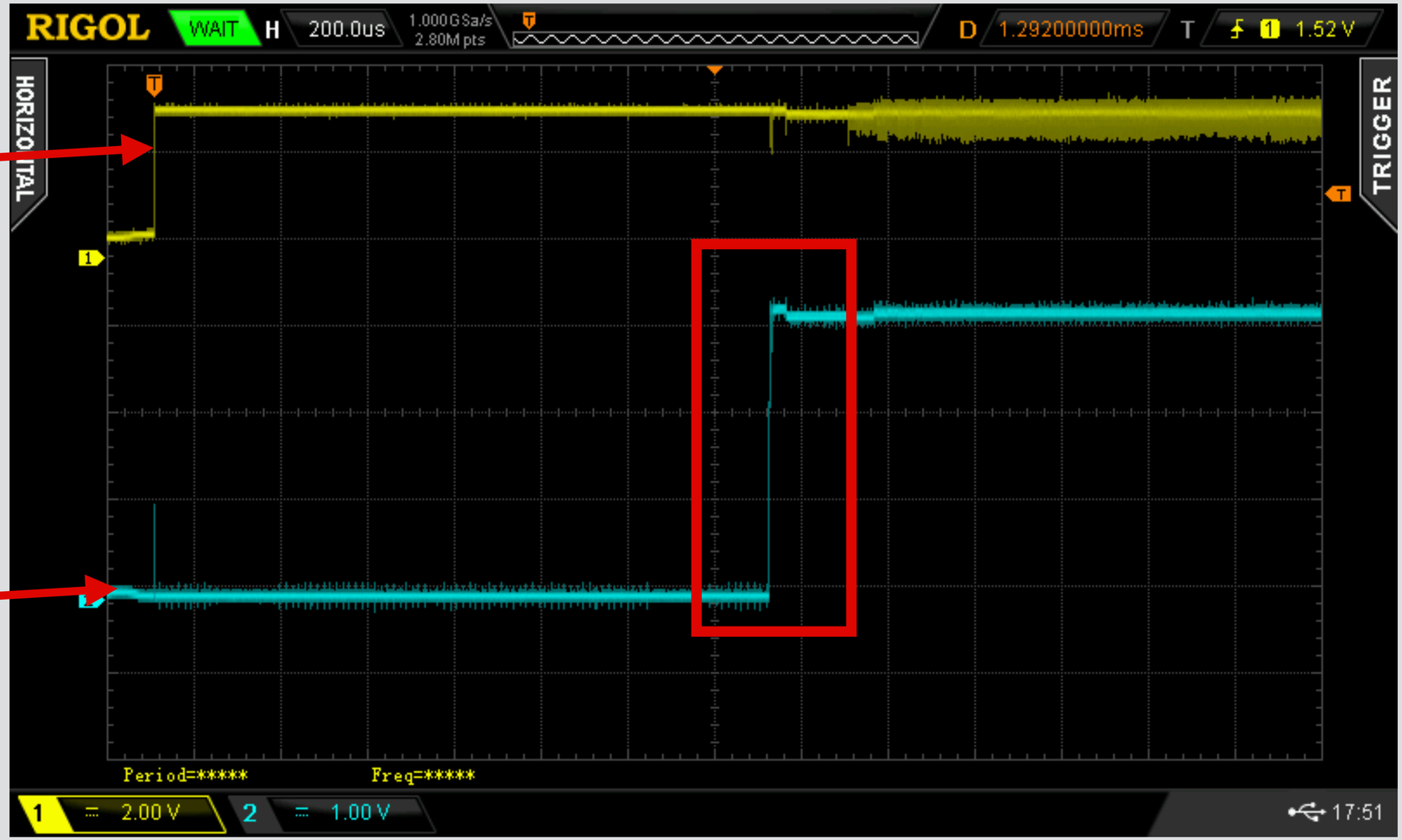
cryptotronix

leveldown security

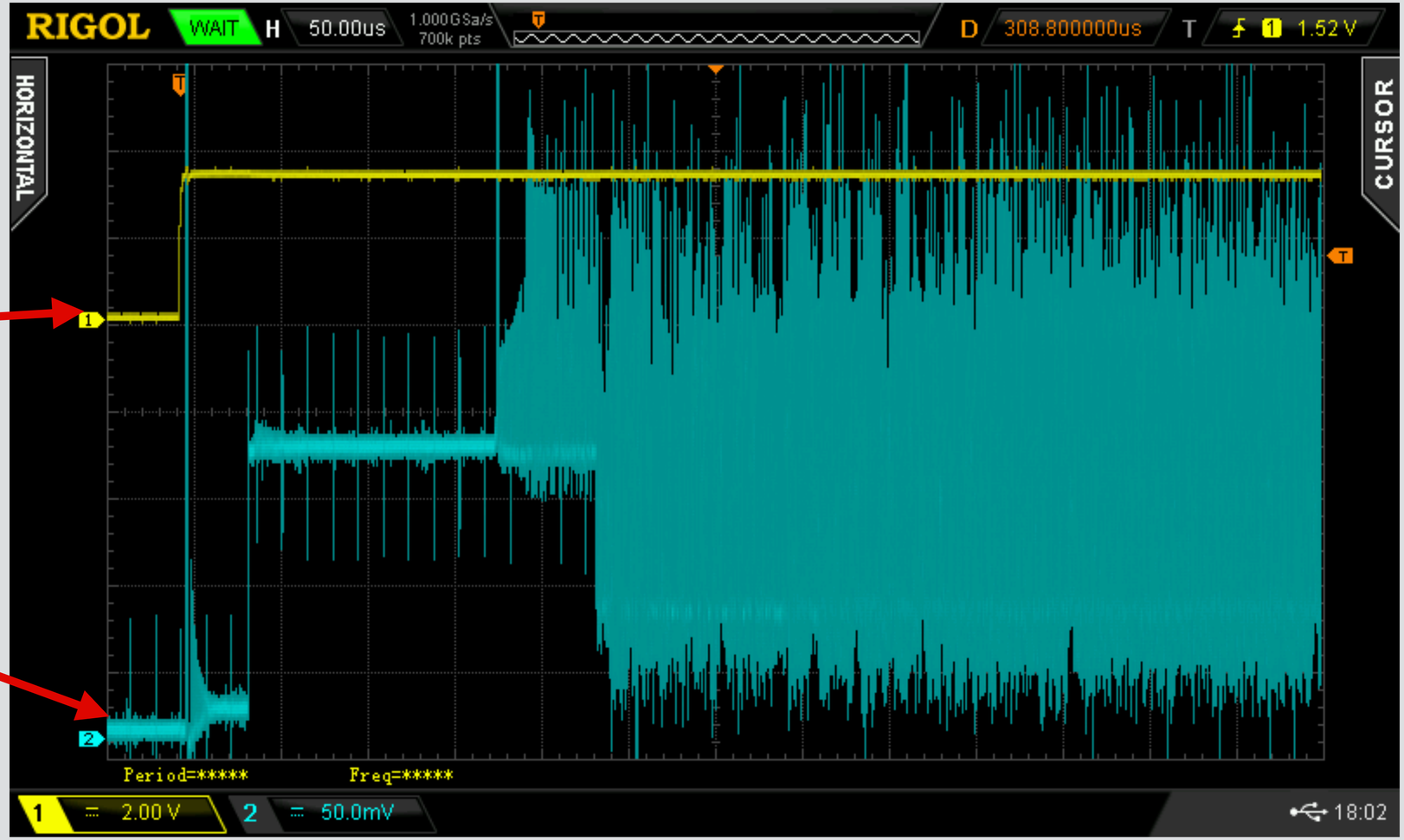
STM32F2 Boot process (1.4ms)

Supply Voltage

Reset pin



Power consumption after reset (200µs)



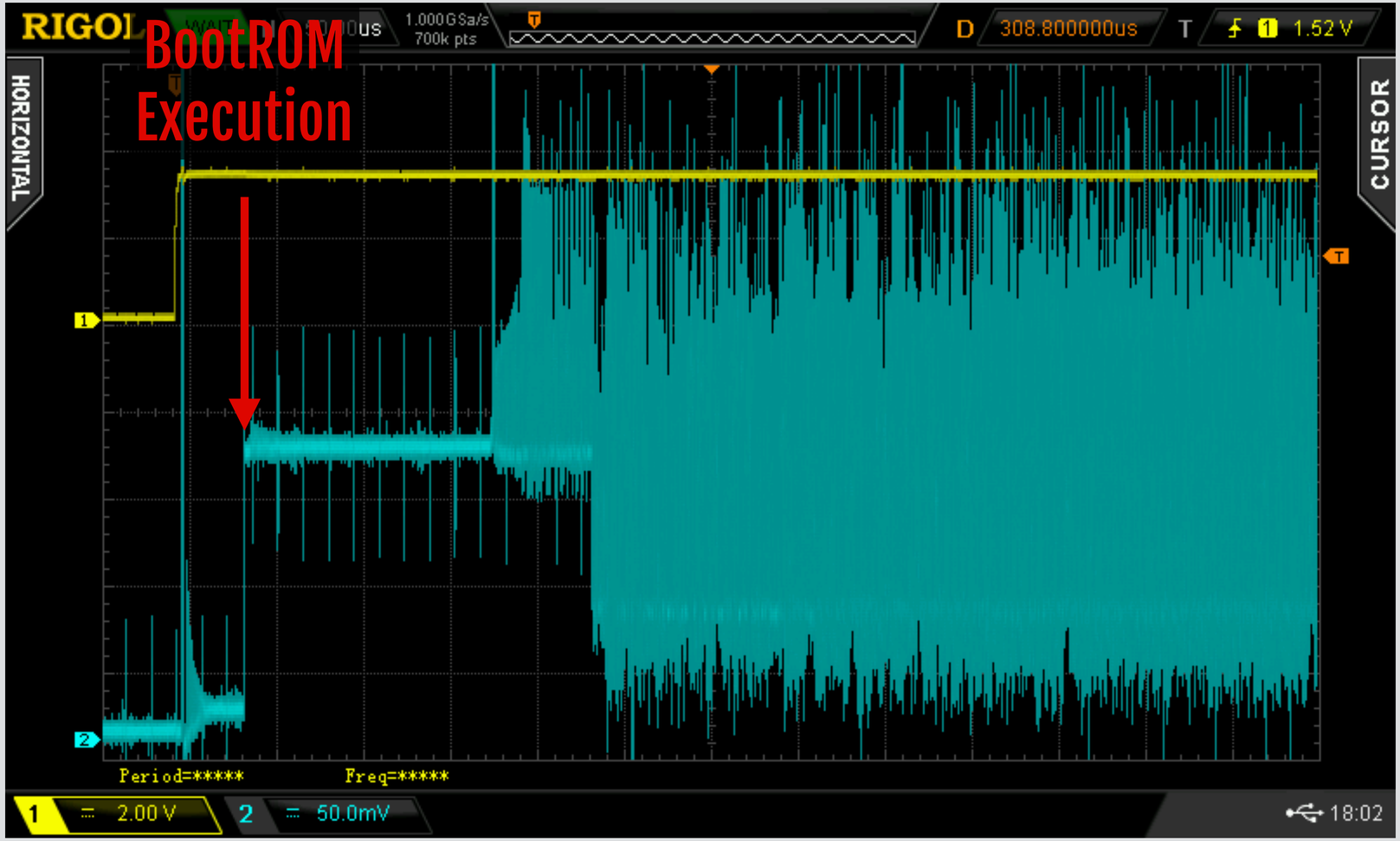
Reset

Power consumption

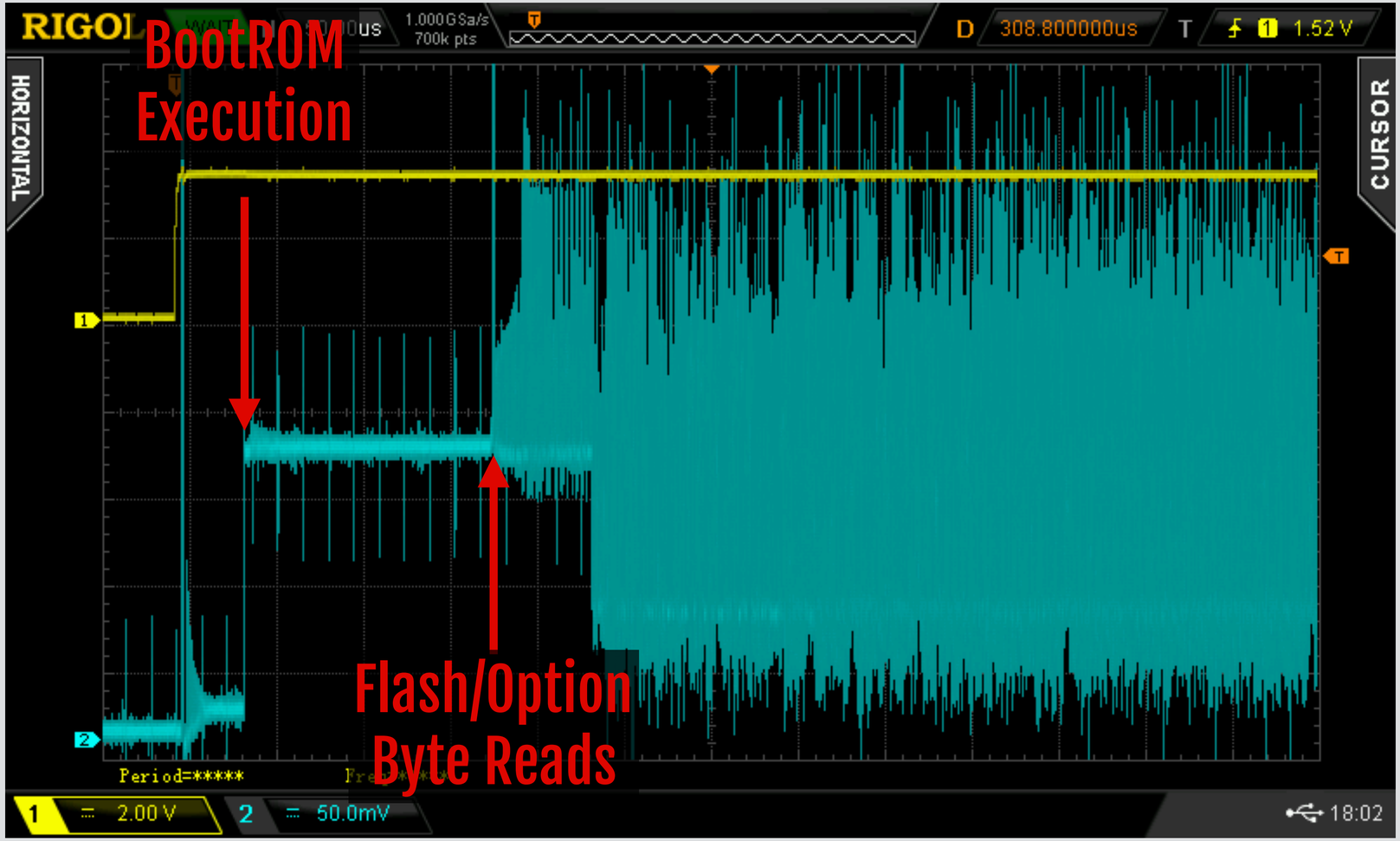
cryptotronix

leveldown security

Power consumption after reset (200µs)



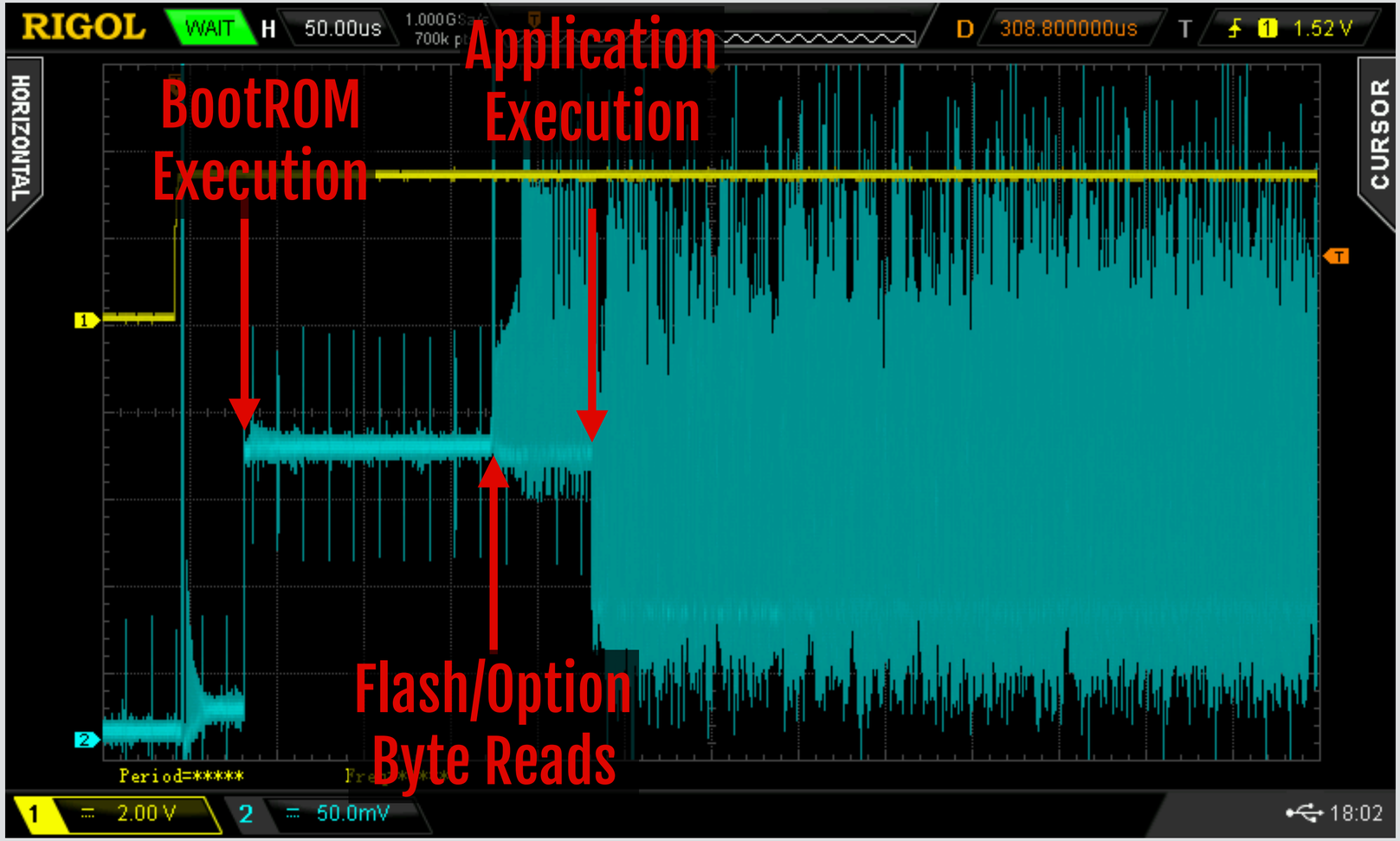
Power consumption after reset (200µs)



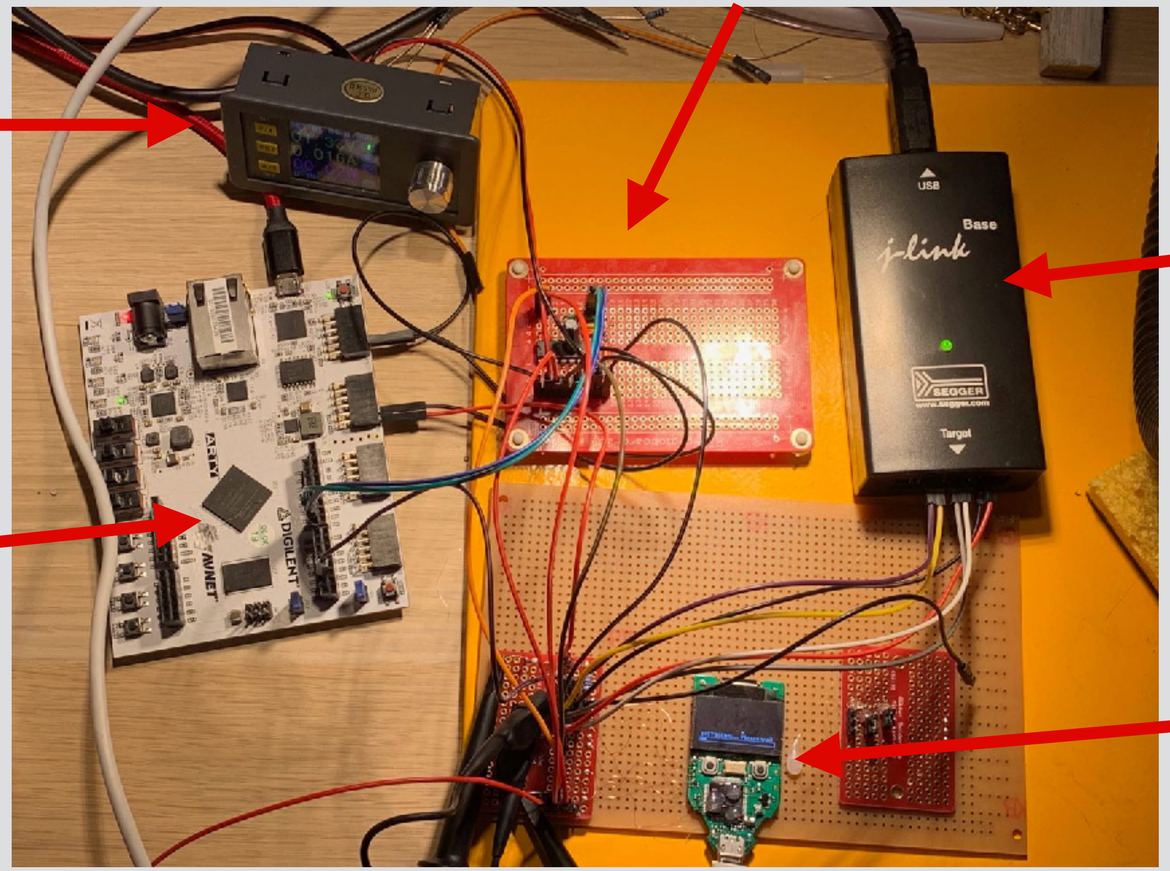
cryptotronix

leveldown security

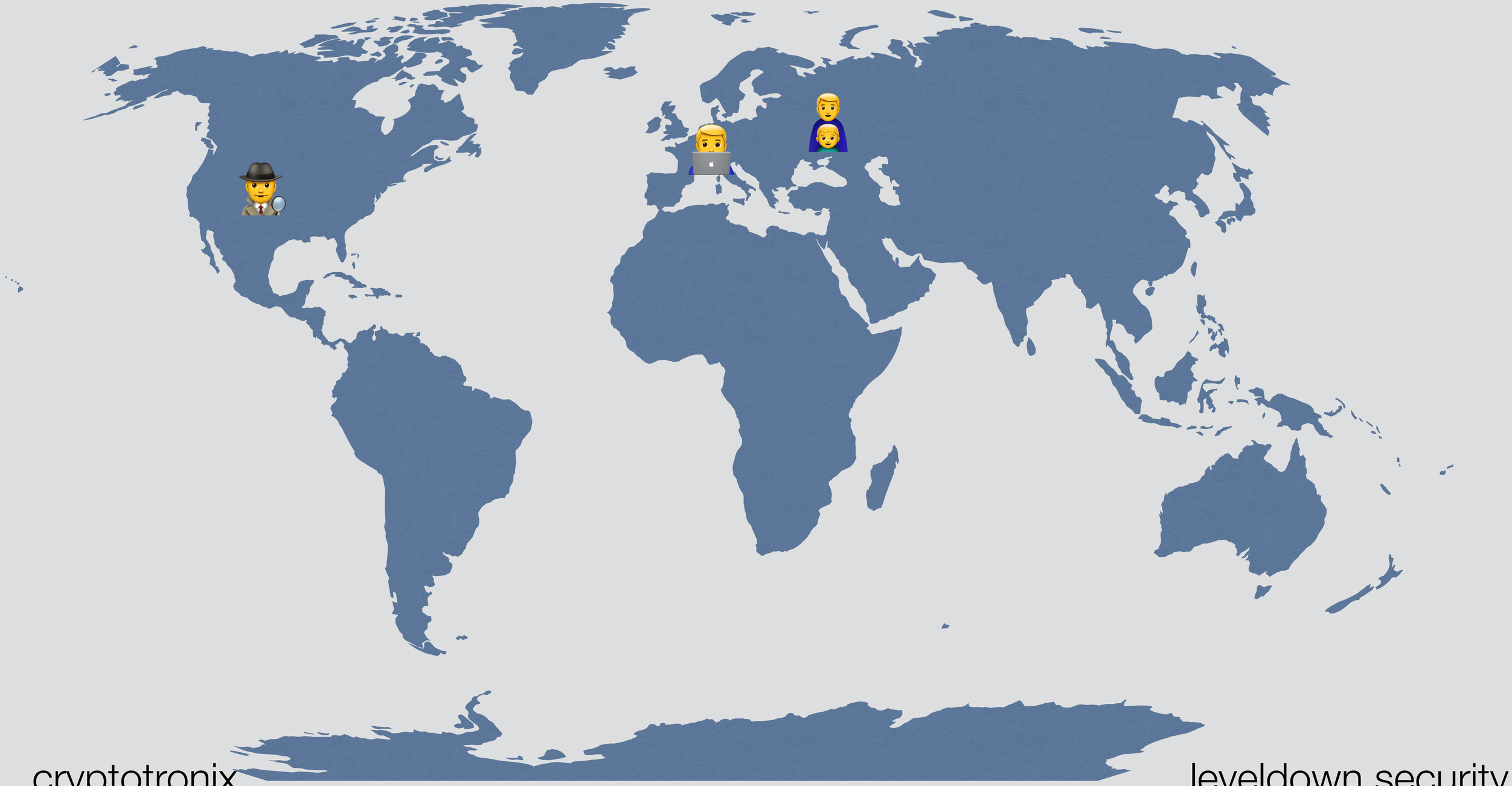
Power consumption after reset (200µs)



Glitching the Trezor One




RIGOL
T'D
H
2.000us
1.000GSa/s
28.0k pts
D
177.240000us
T
928mV
HORIZONTAL
A->X= 165.0us
A->Y= 1.640 V
B->X= 165.0us
B->Y= 3.200 V
ΔX= 0.000 s
1/ΔX= ****
ΔY= 1.560 V
TRIGGER
Type
Pulse
Source
CH2
When
Upper Limit
17.2ms
Lower Limit
855us
Sweep
Normal
Setting
1
1.00 V
2
2.00 V
19:13


cryptotronix
leveldown security

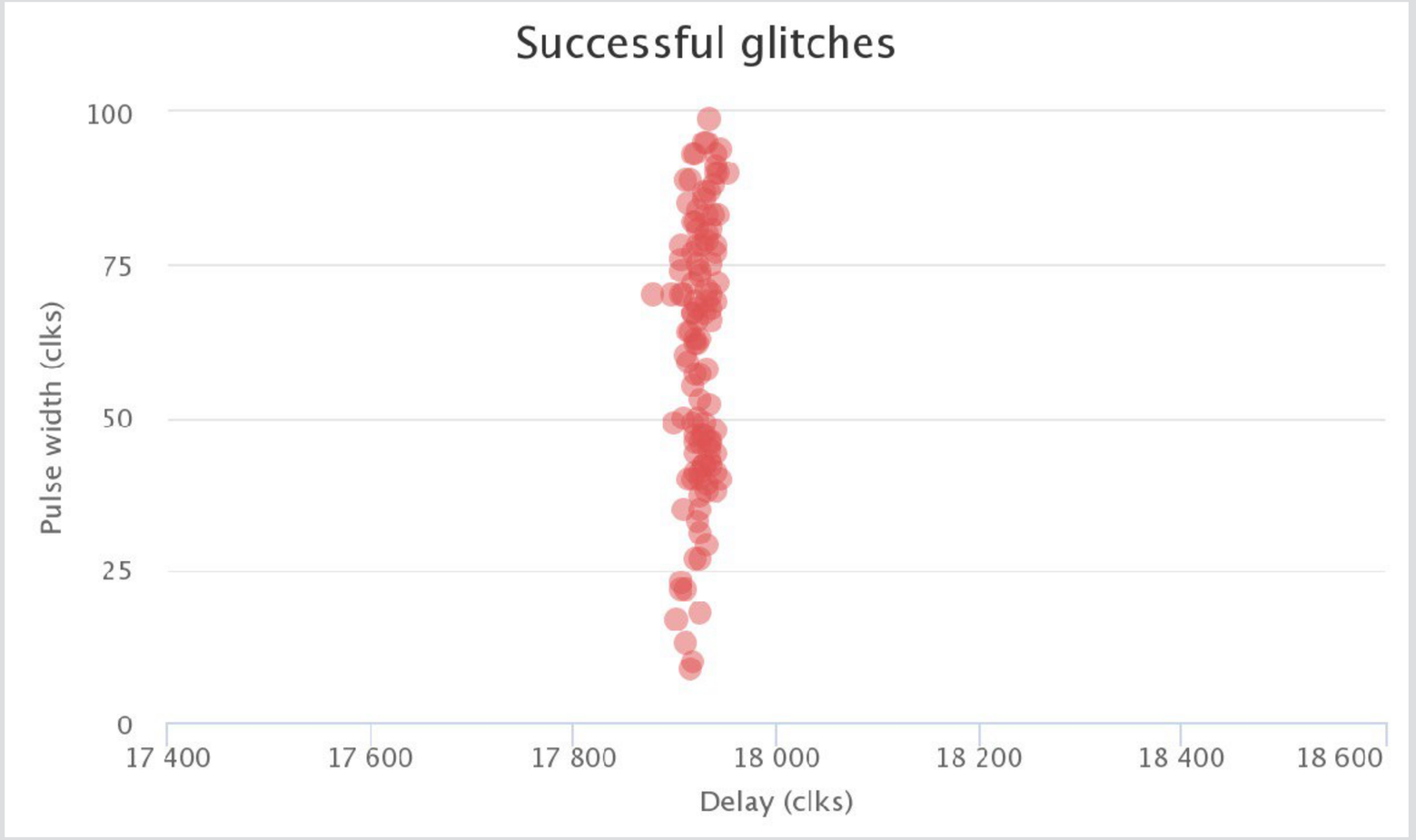
Eventually... Success!

```
student@ubuntu:/mnt/hgfs/git/stm-glitcher/python$ openocd -f openocd_swd.cfg -c
"dump_image SRAM.bin 0x20000000 131072" -c "exit"
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 1000 kHz
adapter_nsrst_delay: 100
none separate
cortex_m reset_config sysresetreq
Info : No device selected, using first device.
Info : J-Link V10 compiled Oct 26 2018 12:04:17
Info : Hardware version: 10.10
Info : VTarget = 3.341 V
Info : clock speed 1000 kHz
Info : SWD DPIDR 0x2ba01477
```

Parameters

- After mitigations have been implemented in Trezor we are happy to release our glitching parameters:
  - Delay: ~17900
  - Pulse: 50
- Our Trezor Glitcher is now available on Github!

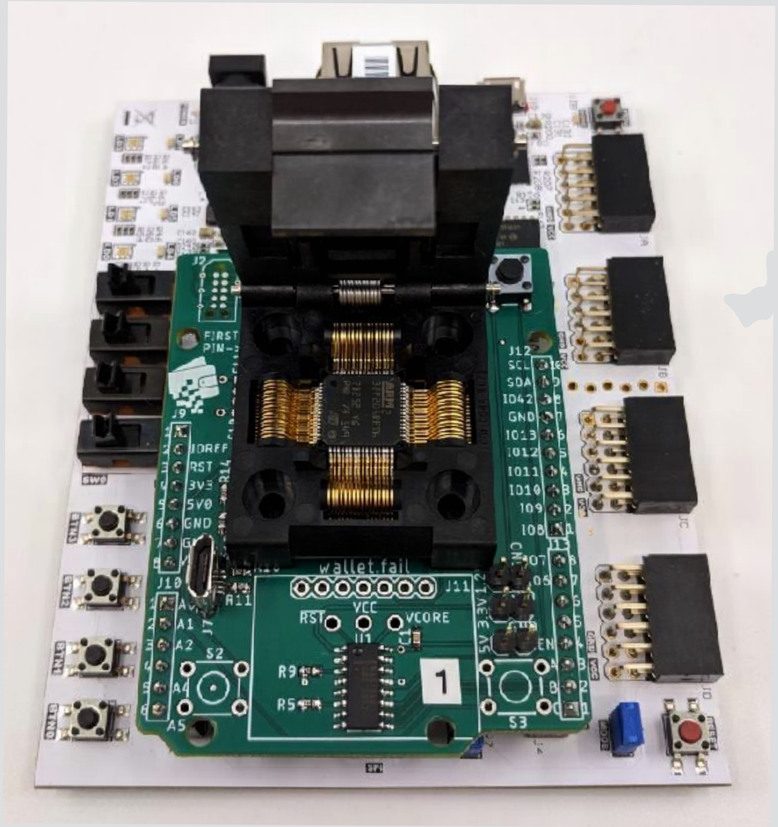
How stable is the glitch?



Dumping the money!

- RDP2 -> RD1 allows for SRAM access
- Secrets get copied to SRAM. Bitcoin secrets = money.
- See wallet.fail for more details

The STM32F2 Glitcher



cryptotronix

leveldown security

Options for defense

- Choose a component with glitch monitors (Brown out detector != glitch protection)
- Use active tamper
- Test your design for susceptibility (on dev kits)
- Write glitch-resistant code

Conclusion

- All chips we looked at were trivially glitchable
- Glitching can be done on the cheap - and on the very cheap
- Just because a chip is glitchable does not equal exploit

Releases on chip.fail

- Verilog for chip.fail glitter
- PCBs (schematic/gerbers) for MUX PMOD and Arty Glitcher
- All the target test firmware

Thanks!

- chip.fail - going live later this week
- You can reach us on
  - cryptotronix.com
  - leveldown.de