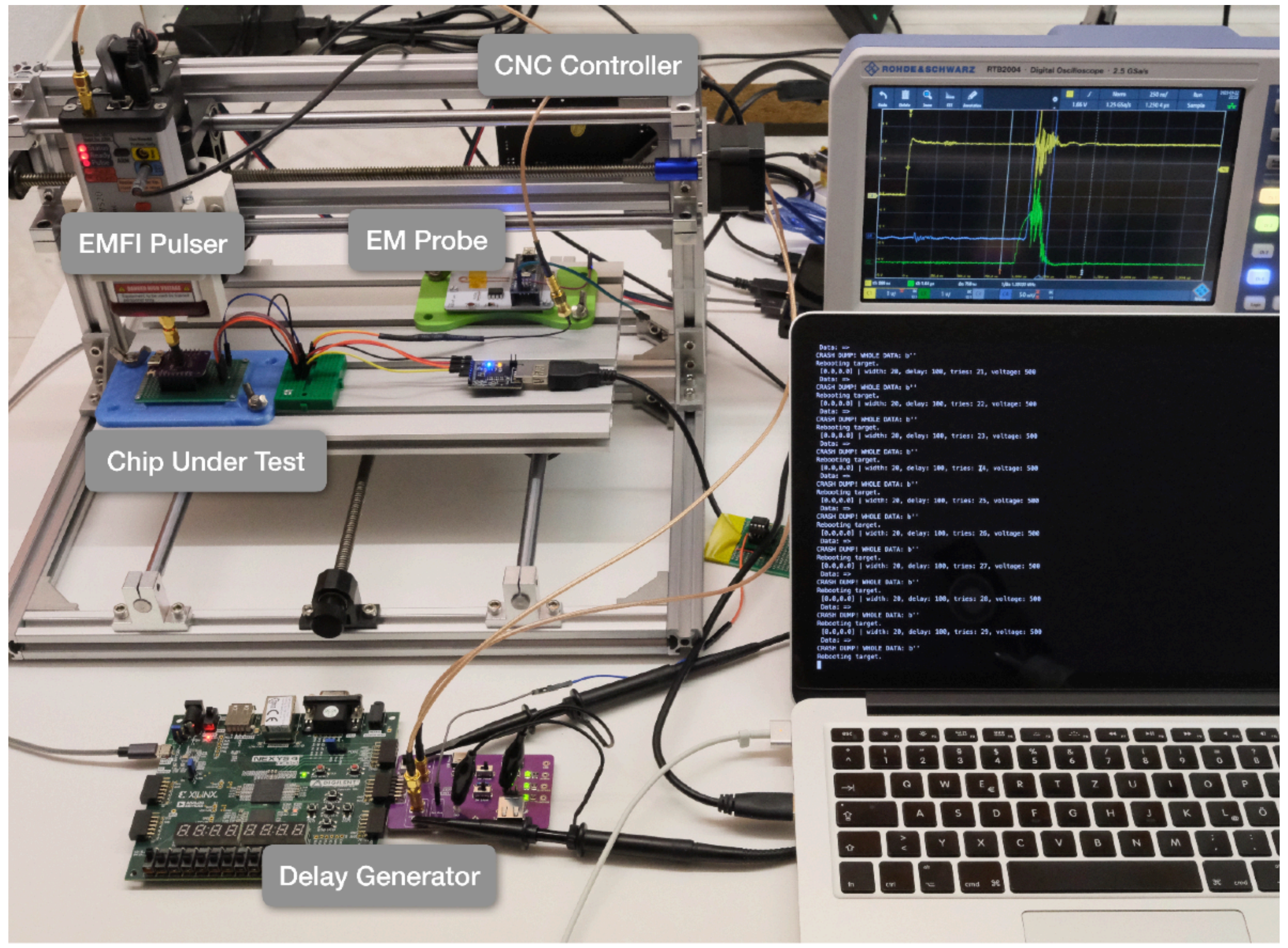
Affordable* EMFI Attacks Against Modern IoT Chips



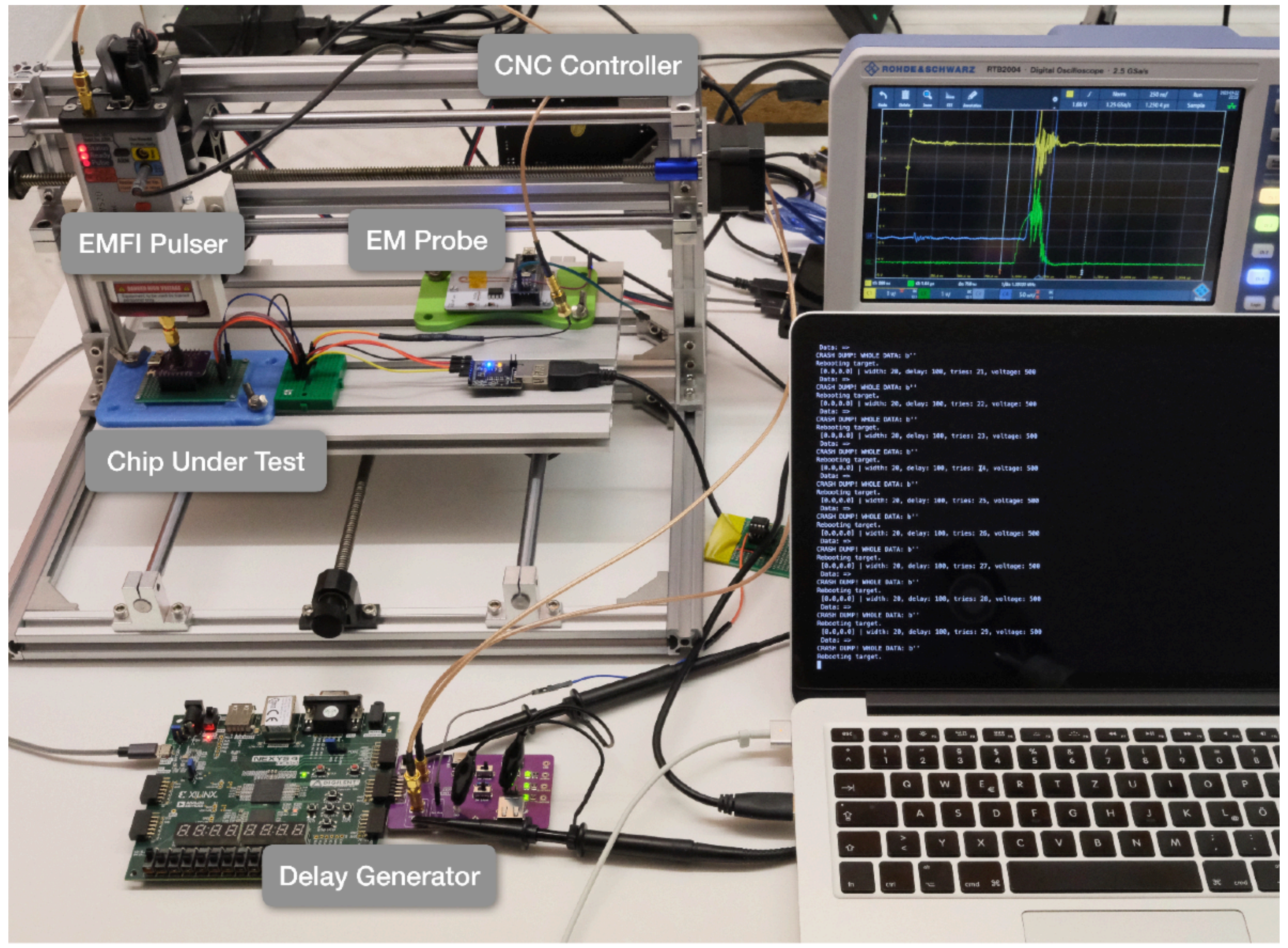
Davide Toldo
Secure Mobile Networking Lab - SEEMOO
Technical University of Darmstadt, Germany

SECURE MOBILE NETWORKING

TECHNISCHE UNIVERSITÄT DARMSTADT

Affordable* EMFI Attacks Against Modern IoT Chips

* and open-source

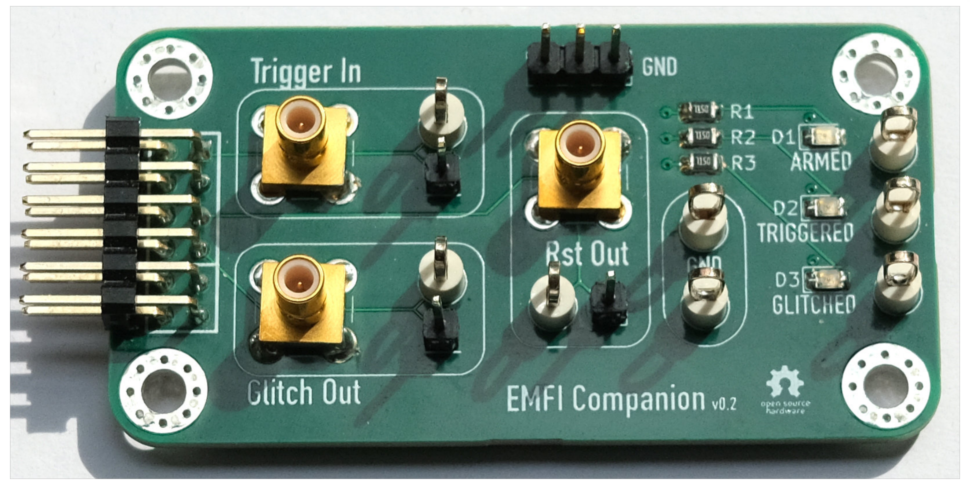


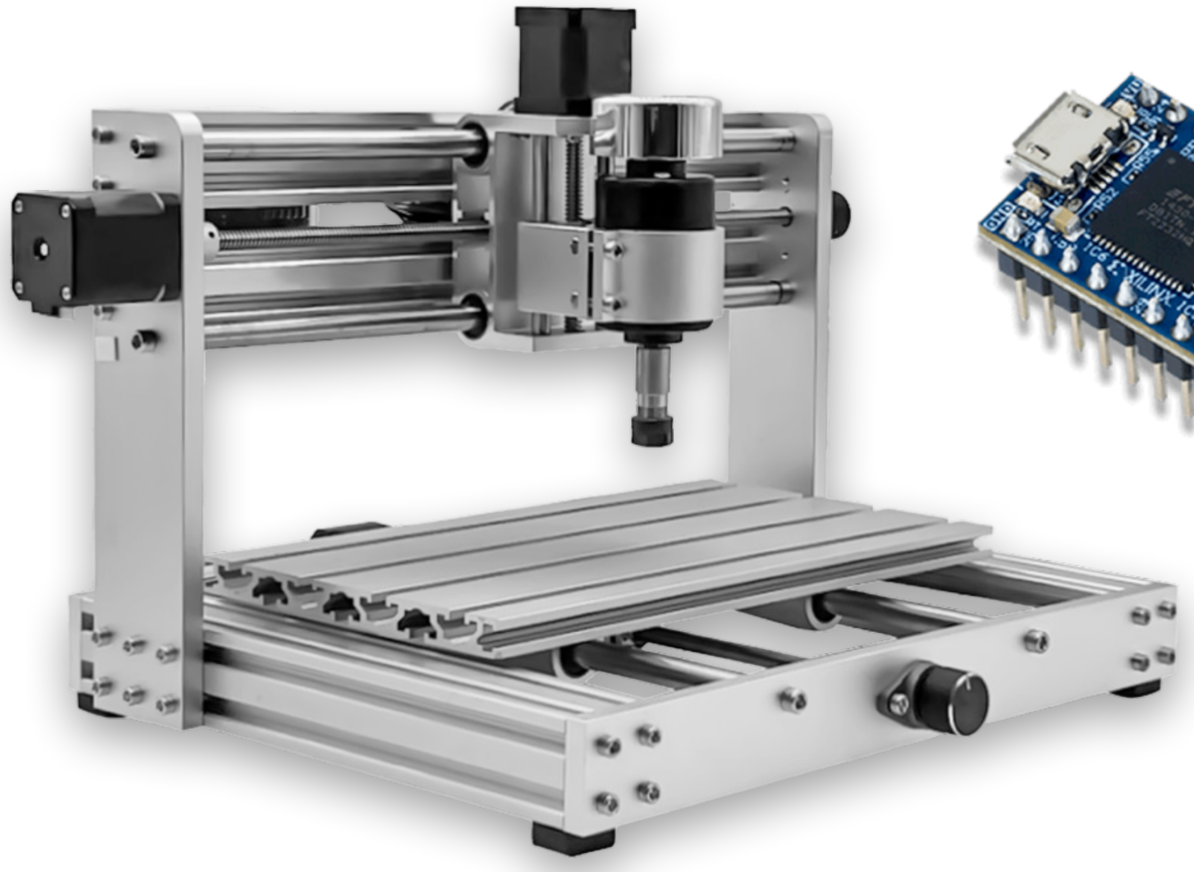
Davide Toldo
Secure Mobile Networking Lab - SEEMOO
Technical University of Darmstadt, Germany

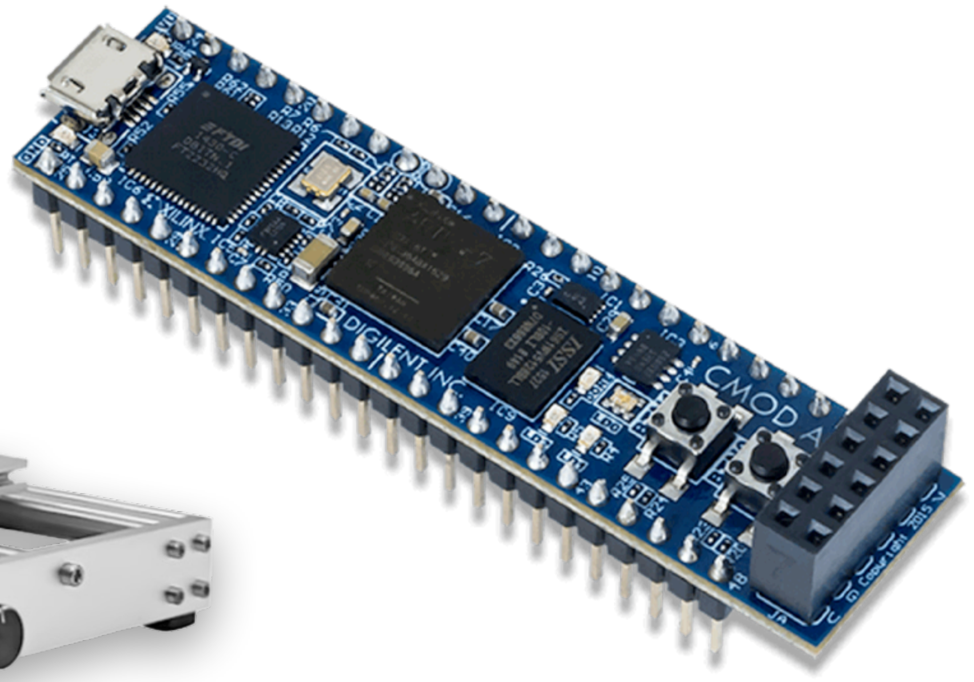
SECURE MOBILE NETWORKING

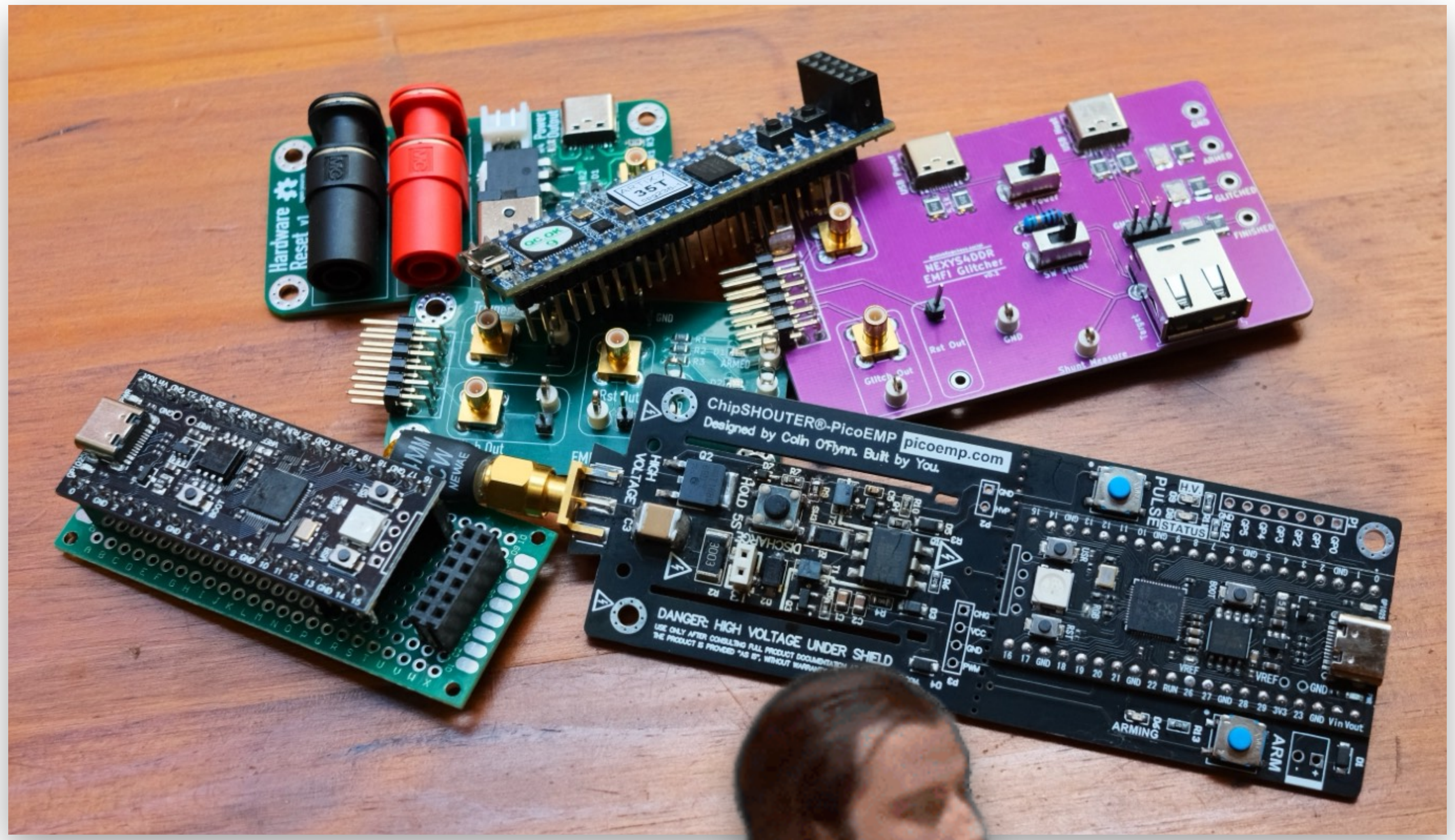
TECHNISCHE UNIVERSITÄT DARMSTADT

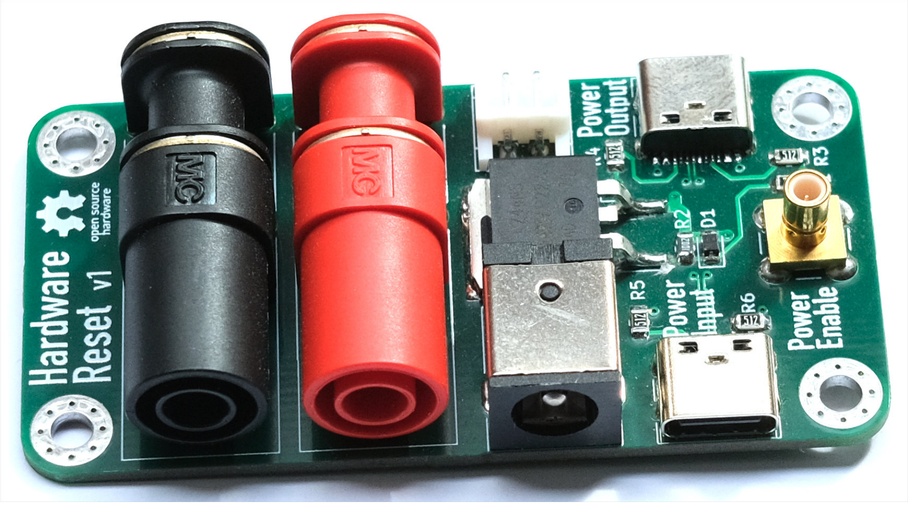
What are we actually doing here?

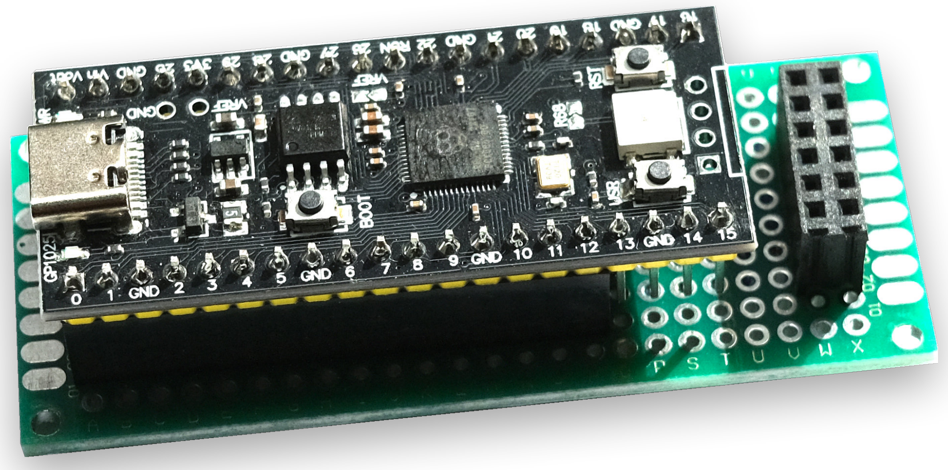


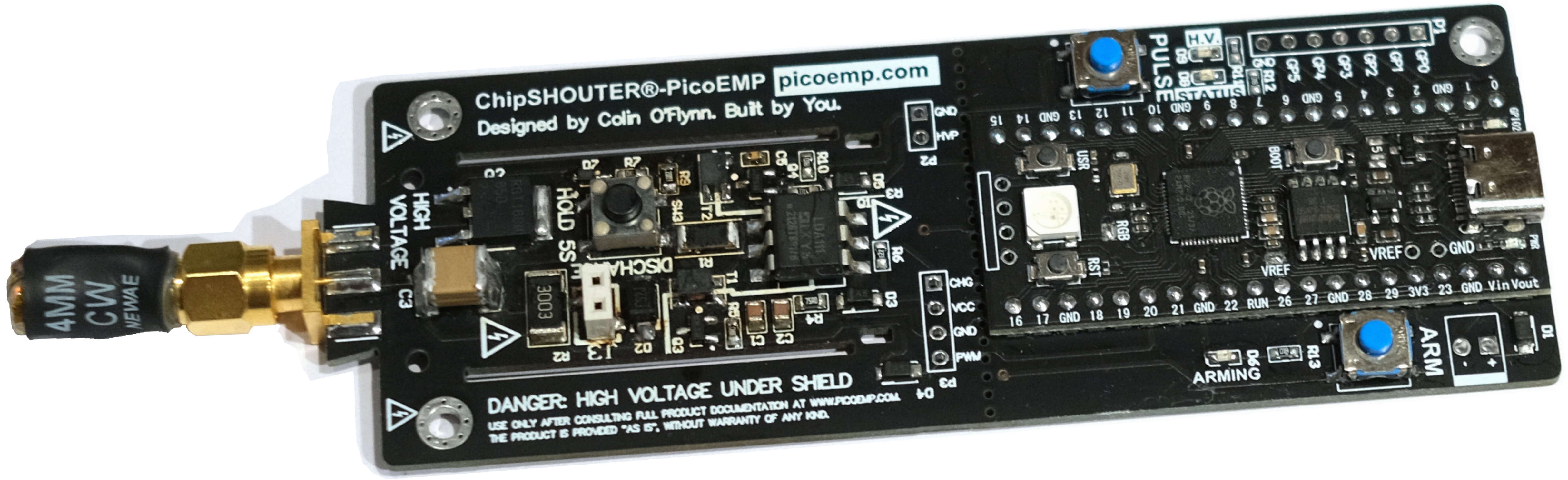


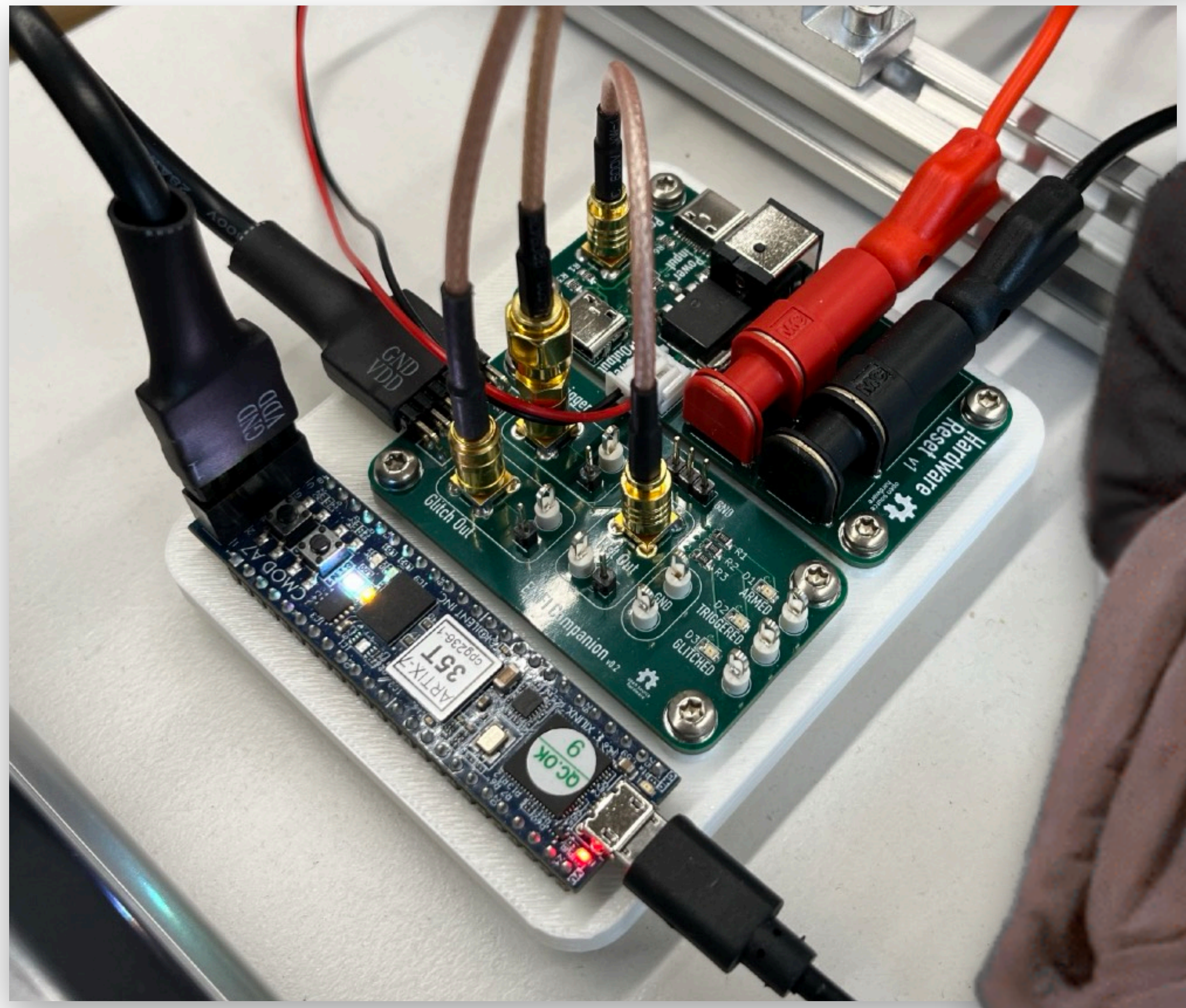






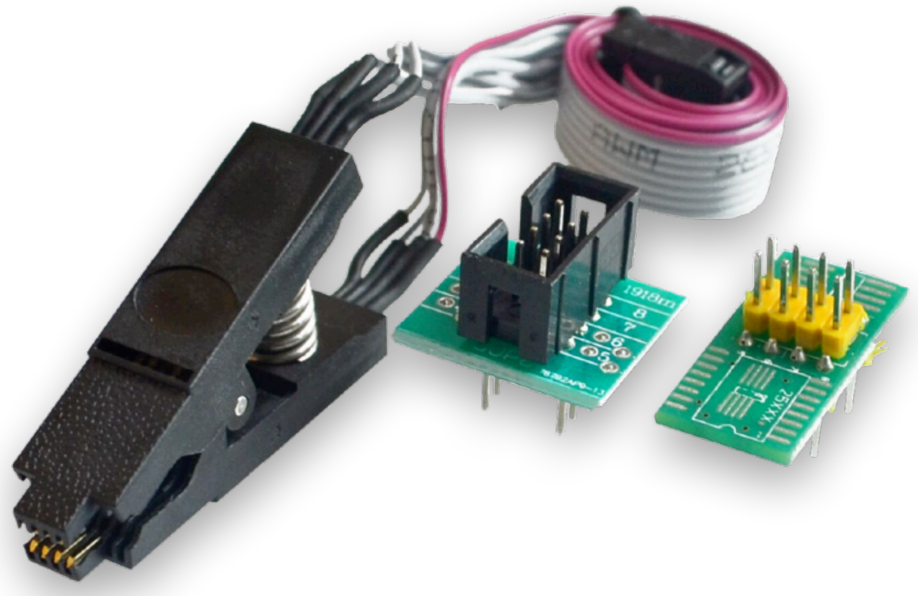






Motivation

- Modern security features prevent simple hardware attacks, such as:
  - Extract, modify and reflash firmware: Tasmota & similar FOSS alternative firmwares for embedded devices or custom-made ones
  - Get full access to devices you own (root shell, debug access, ...)
  - Performing (security) research on embedded devices when such levels of access are not available
- High entry barrier towards defeating these new security features



https://opencircuit.shop/product/ic-test-clip-soic-8-pin

https://commons.wikimedia.org/wiki/File:Segger_J-Link_PRO.jpg

EMFI = 🧲

Changing execution path through magnetic fields

```
static bool debug_enable = false
void setup() {
  // check if debugging enabled
  if (debug_enable) {
    enable_debugging()
  }
}
void loop() {}
```

EMFI = 🧲

Changing execution path through magnetic fields

```
static bool debug_enable = true
void setup() {
  // check if debugging enabled
  if (debug_enable) {
    enable_debugging()
  }
}
void loop() {}
```

EMFI = 🧲

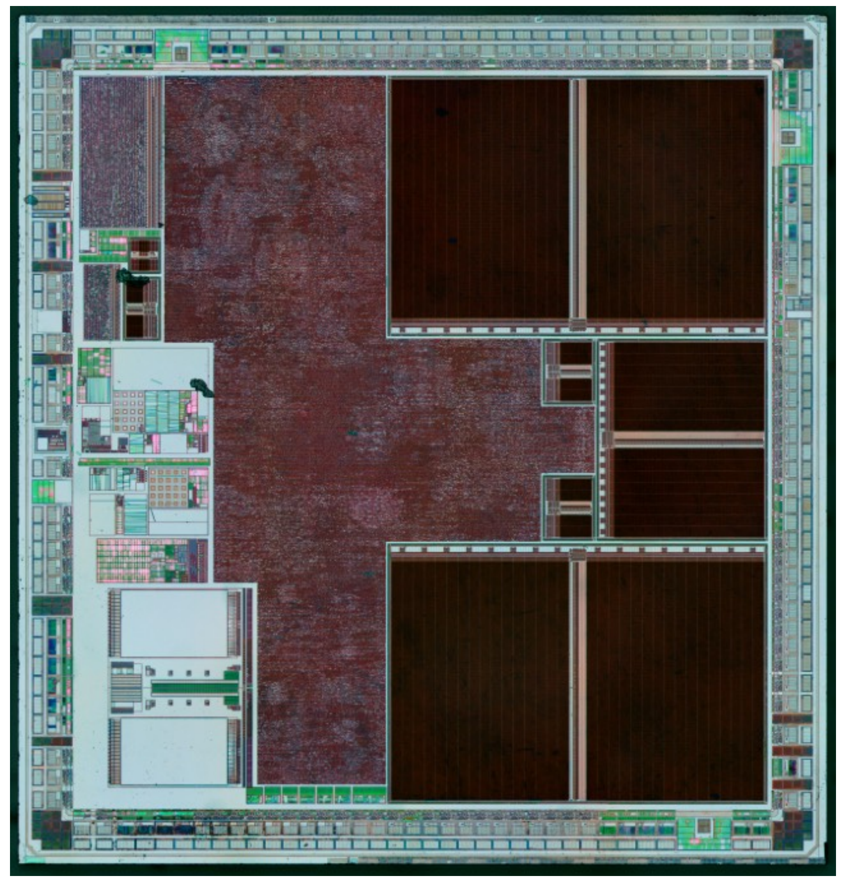
Changing execution path through magnetic fields

```
static bool debug_enable = true
void setup() {
  // check if debugging enabled
  if !debug_enable) {
    enable_debugging()
  }
}
void loop() {}
```

EMFI = 🧲

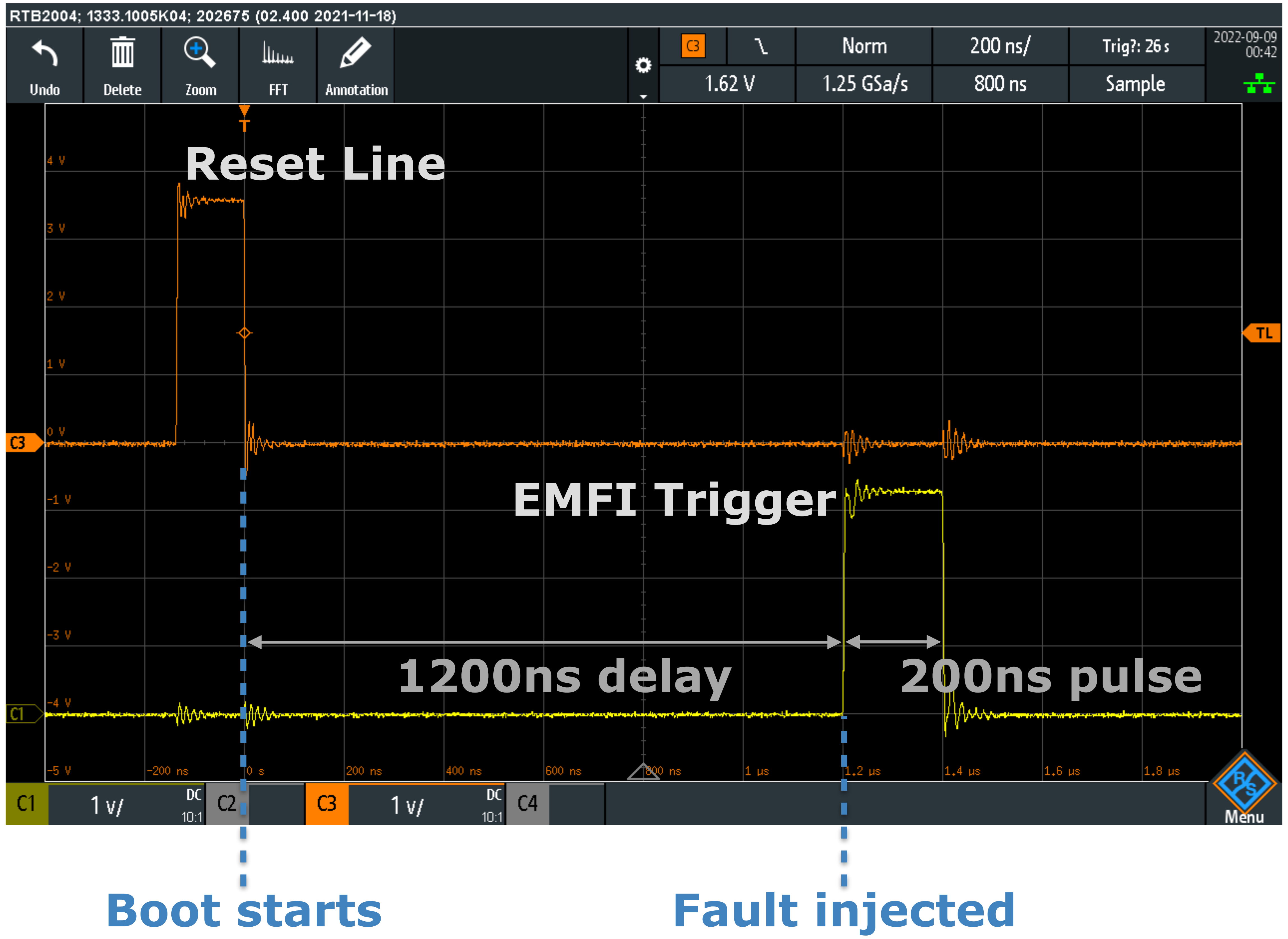
- Injecting "faults" directly into an IC can force it to behave differently and give us the access we need.
- Physical FI: affecting chip's internal behavior through external conditions.
- EMFI: electromagnetic pulses on SoC's / memory → induce currents
  → affect transistors
  → change execution path

EMFI Setup Requirements

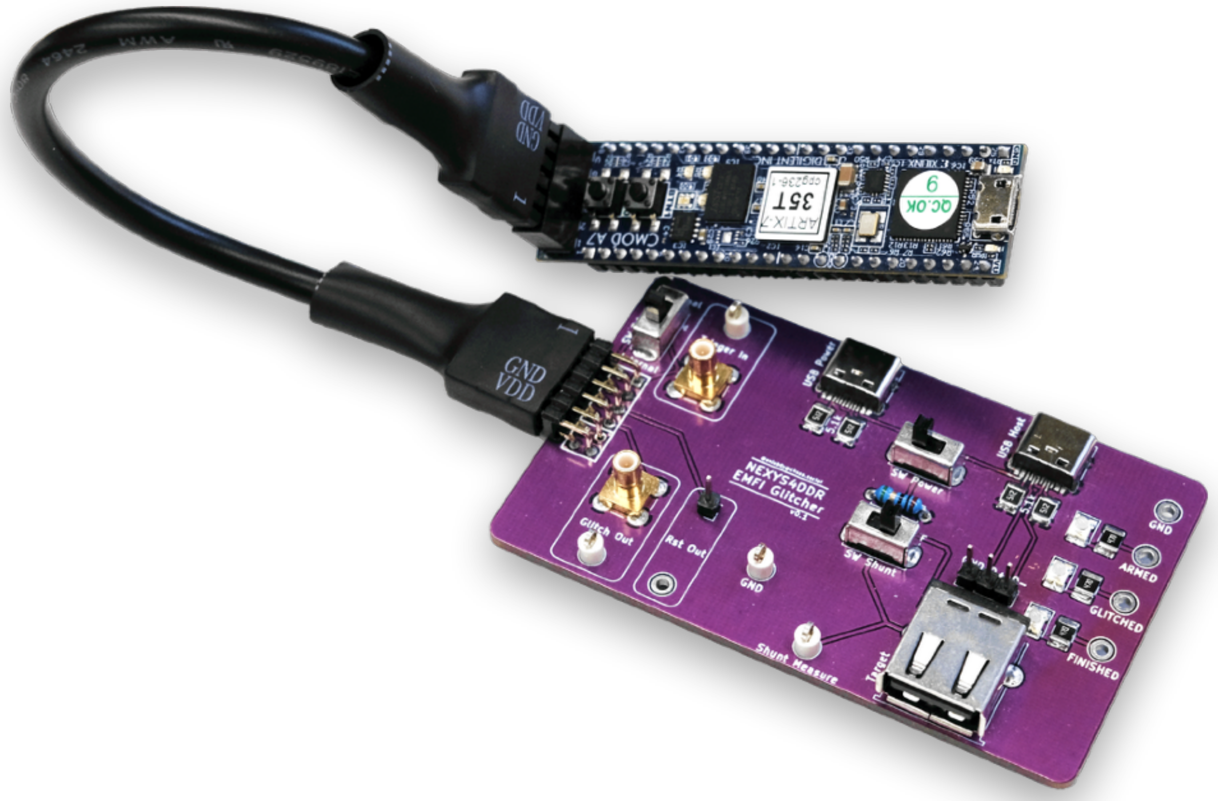


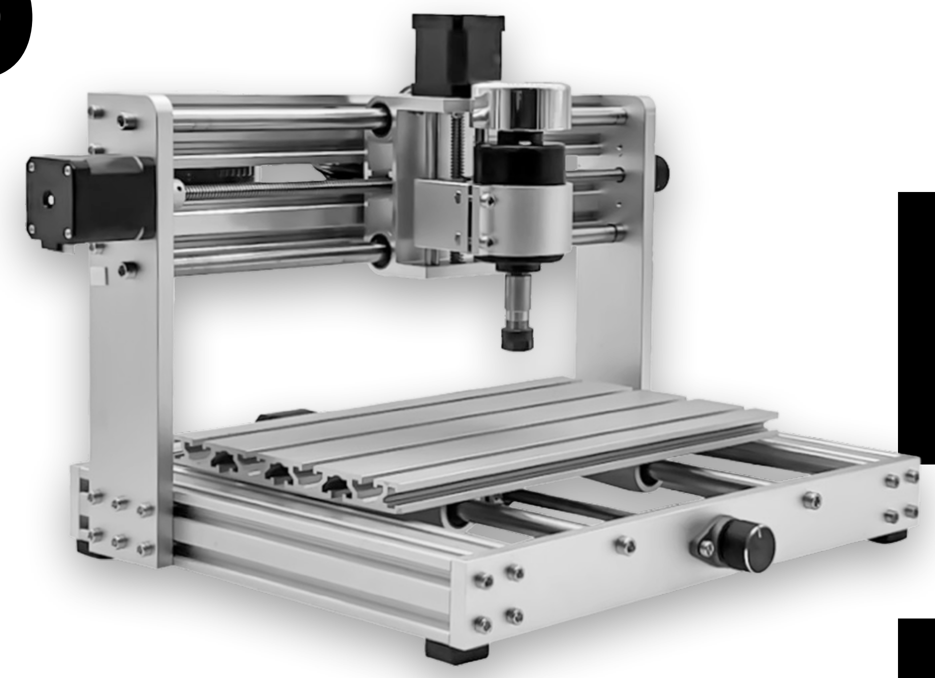
https://commons.wikimedia.org/wiki/File:GD32F103CBT6-Si-HD.jpg

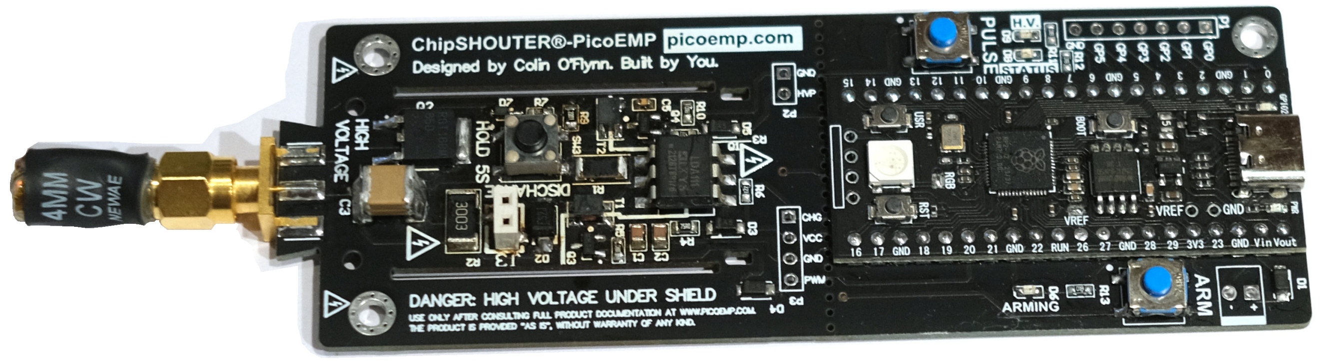
- Location & timing essential: fault exactly at the desired instruction and SoC area
- Code, binary and side channel analysis help discover timing for potential fault
- FPGA: 400MHz = 2.5ns steps

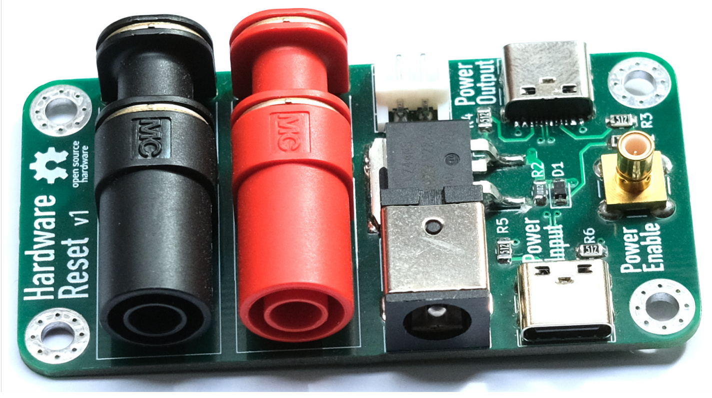


EMFI Setup

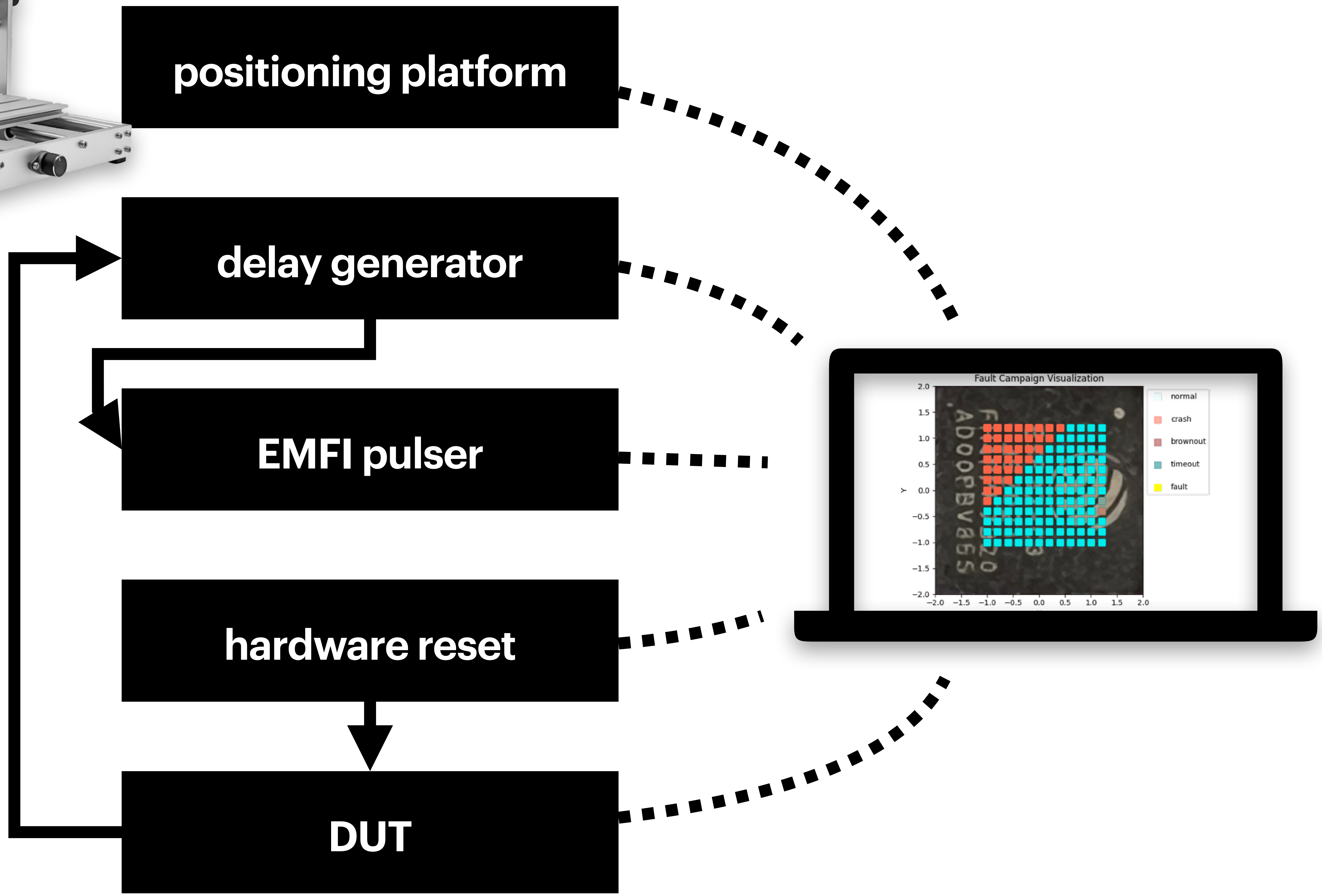





ChipSHOUTER®-PicoEMP picoemp.com
Designed by Colin O'Flynn. Built by You.
DANGER: HIGH VOLTAGE UNDER SHIELD
ARM


Hardware Reset

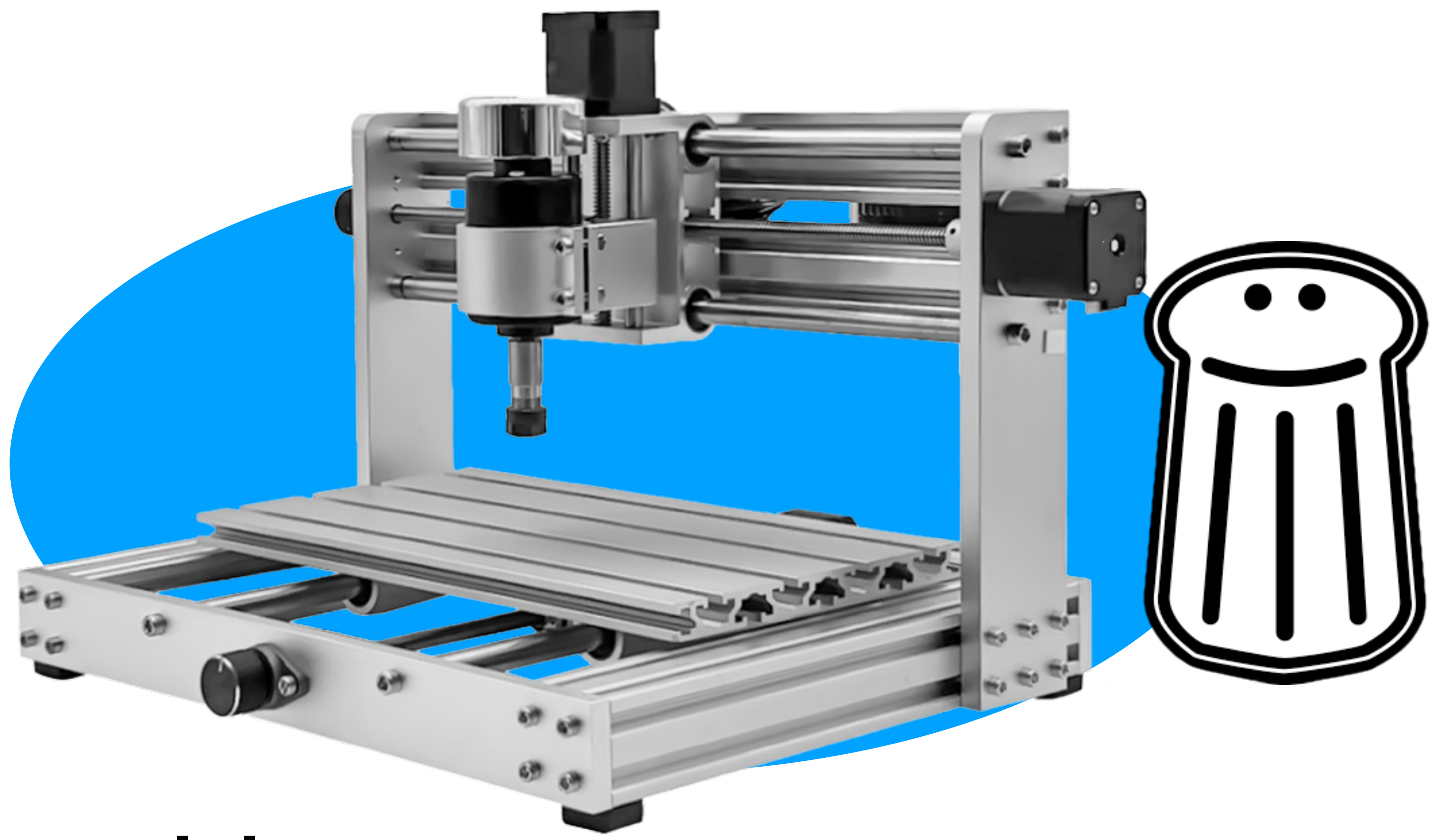

ESP32-C3
C3 Mini


positioning platform
delay generator
EMFI pulser
hardware reset
DUT
Fault Campaign Visualization
normal
crash
brownout
timeout
fault

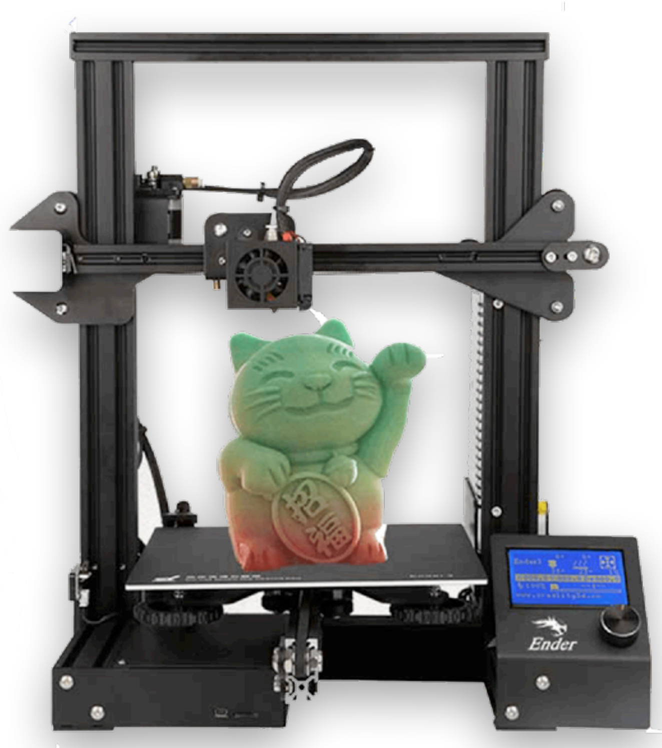
EMFI Setup

Positioning Platform

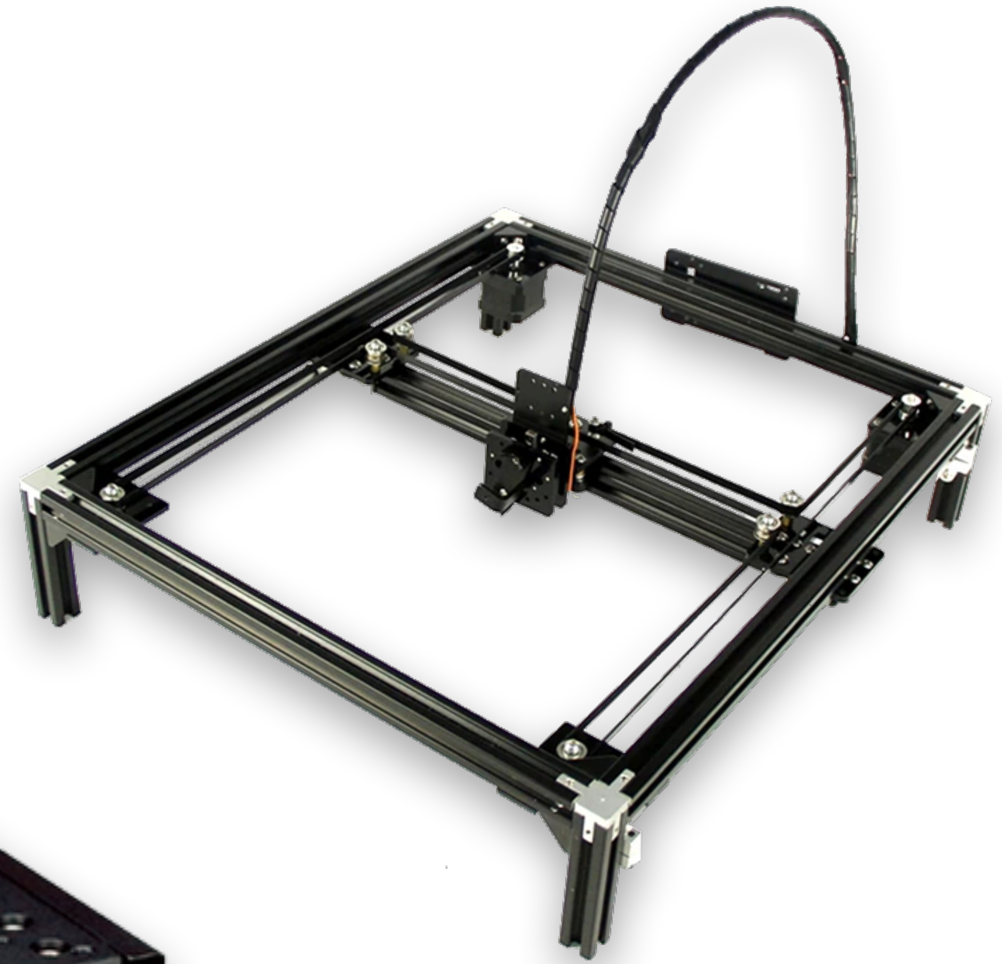
- CNCs or 3D printers can be used interchangeably due to GCODE
- Both are available for very low cost, come with motor controllers and everything needed
- Lead screws have approx. 10x more backlash → if budget allows, use belts
- Motorized XY stages offer small benefit for the price and IoT target



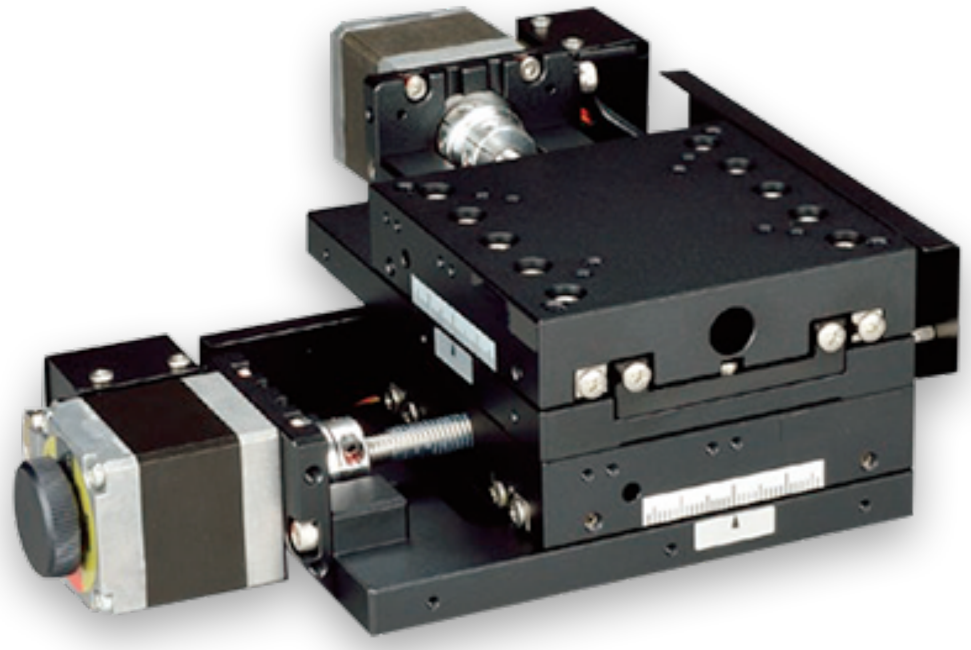
https://aliexpress.com



https://aliexpress.com



https://aliexpress.com

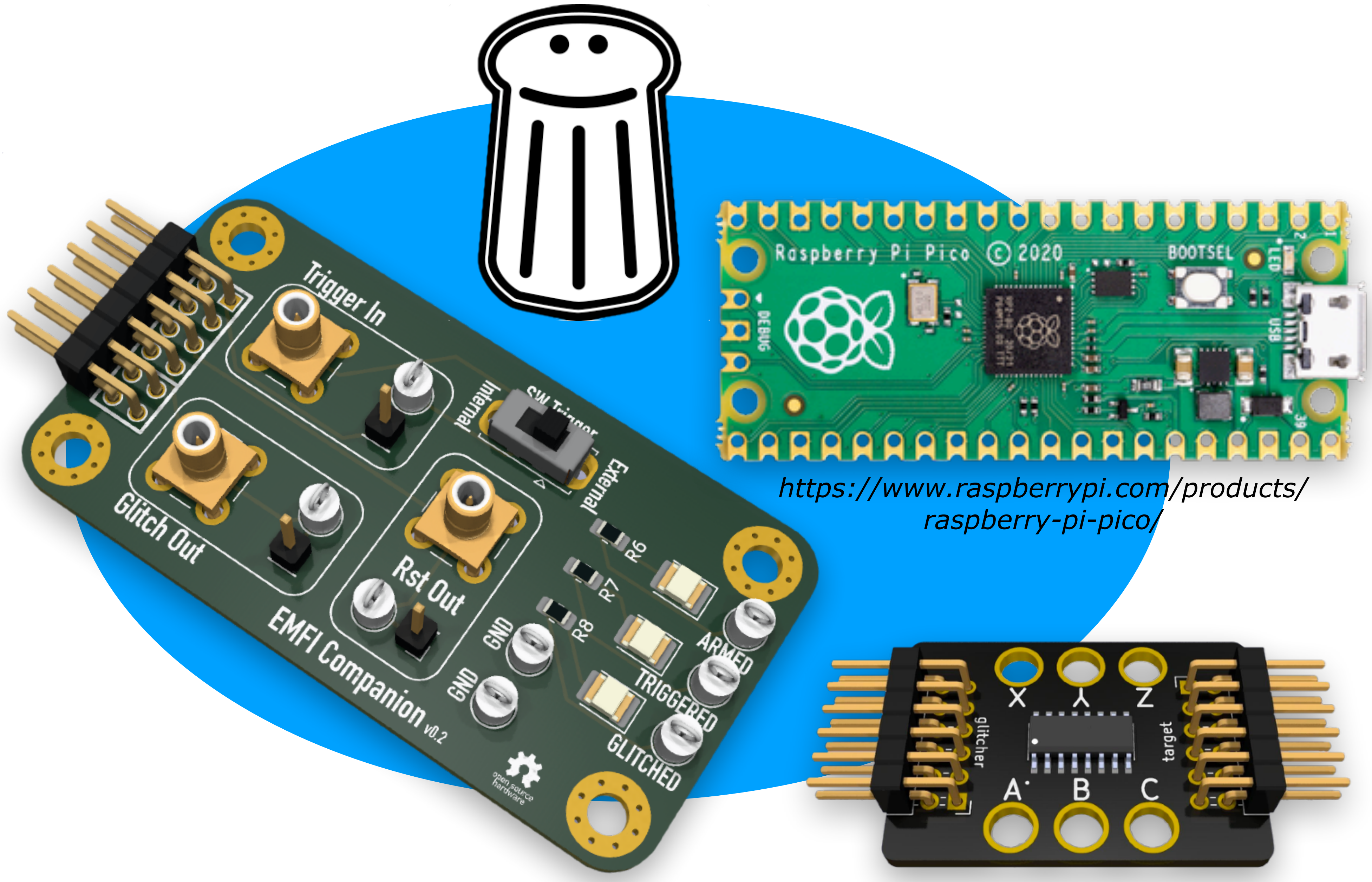


https://www.thk.com

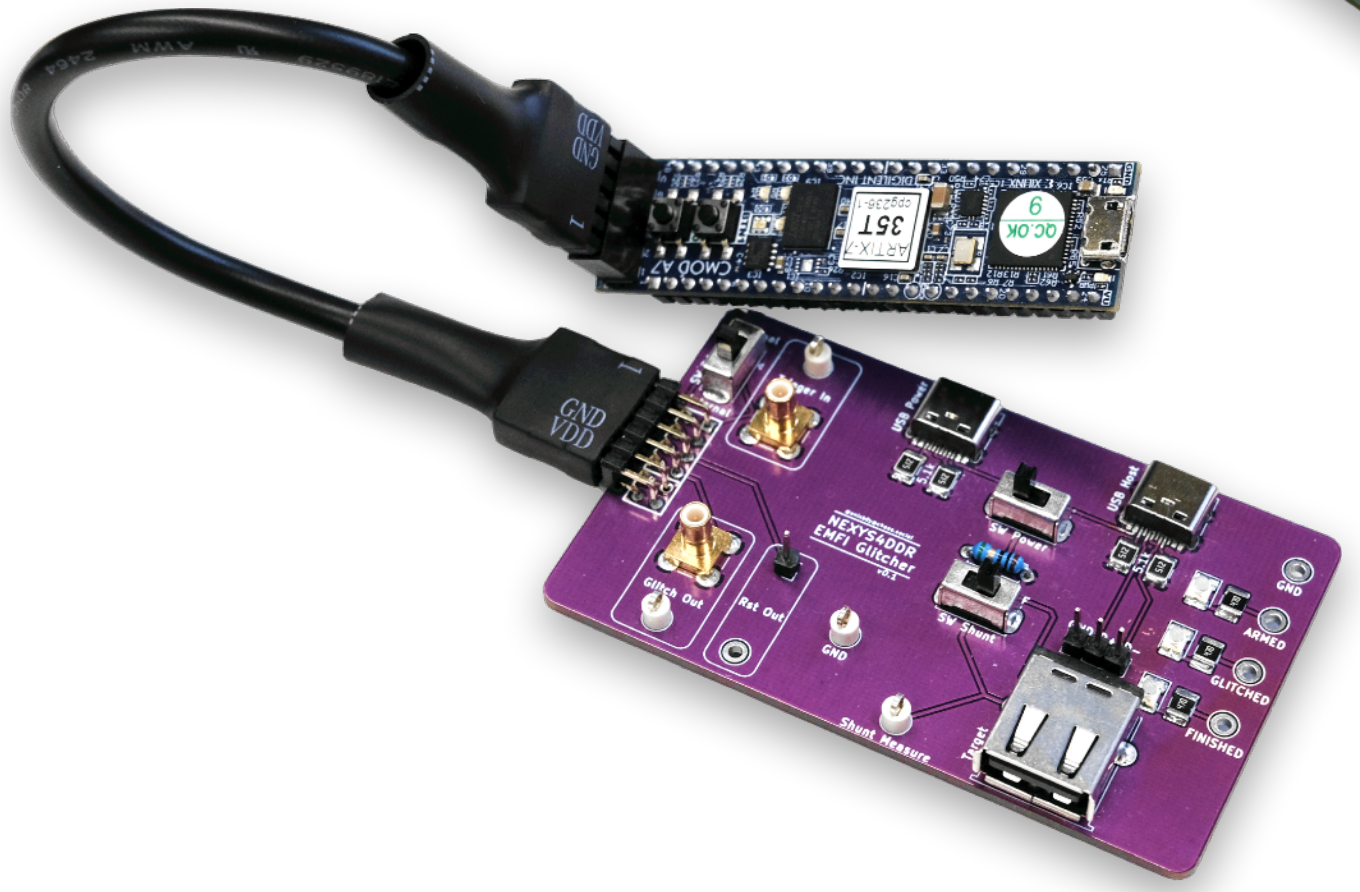
EMFI Setup

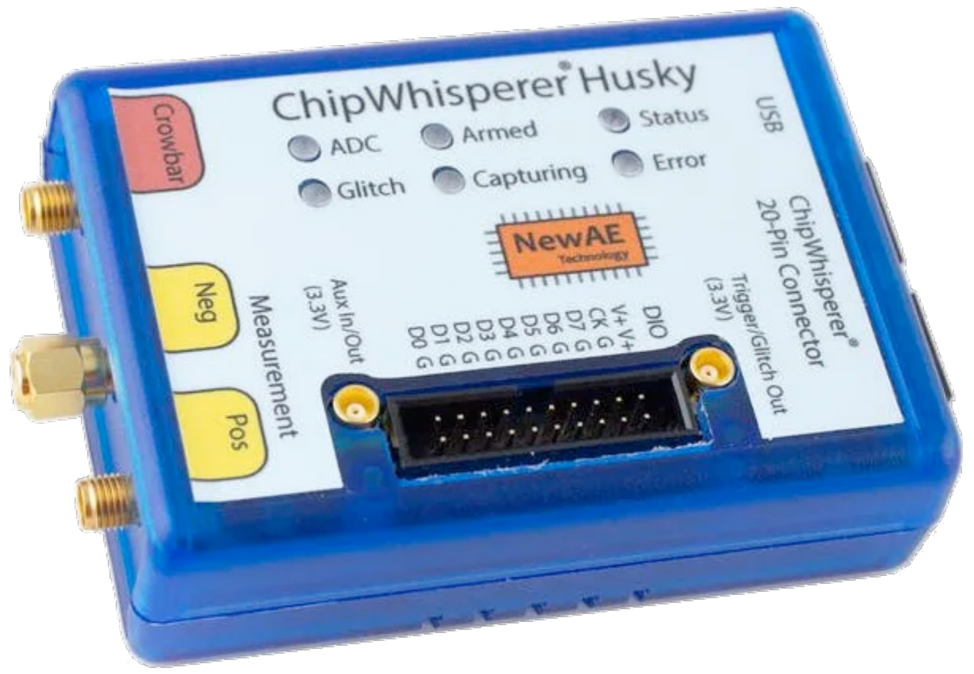
Delay Generator

- Raspberry Pi Pico: custom firmware by me (NEW!! 🎉)
- FPGA: chip.fail FOSS bitstream by @stacksmashing
- ChipWhisperer by @colinoflynn



https://www.raspberrypi.com/products/raspberry-pi-pico/

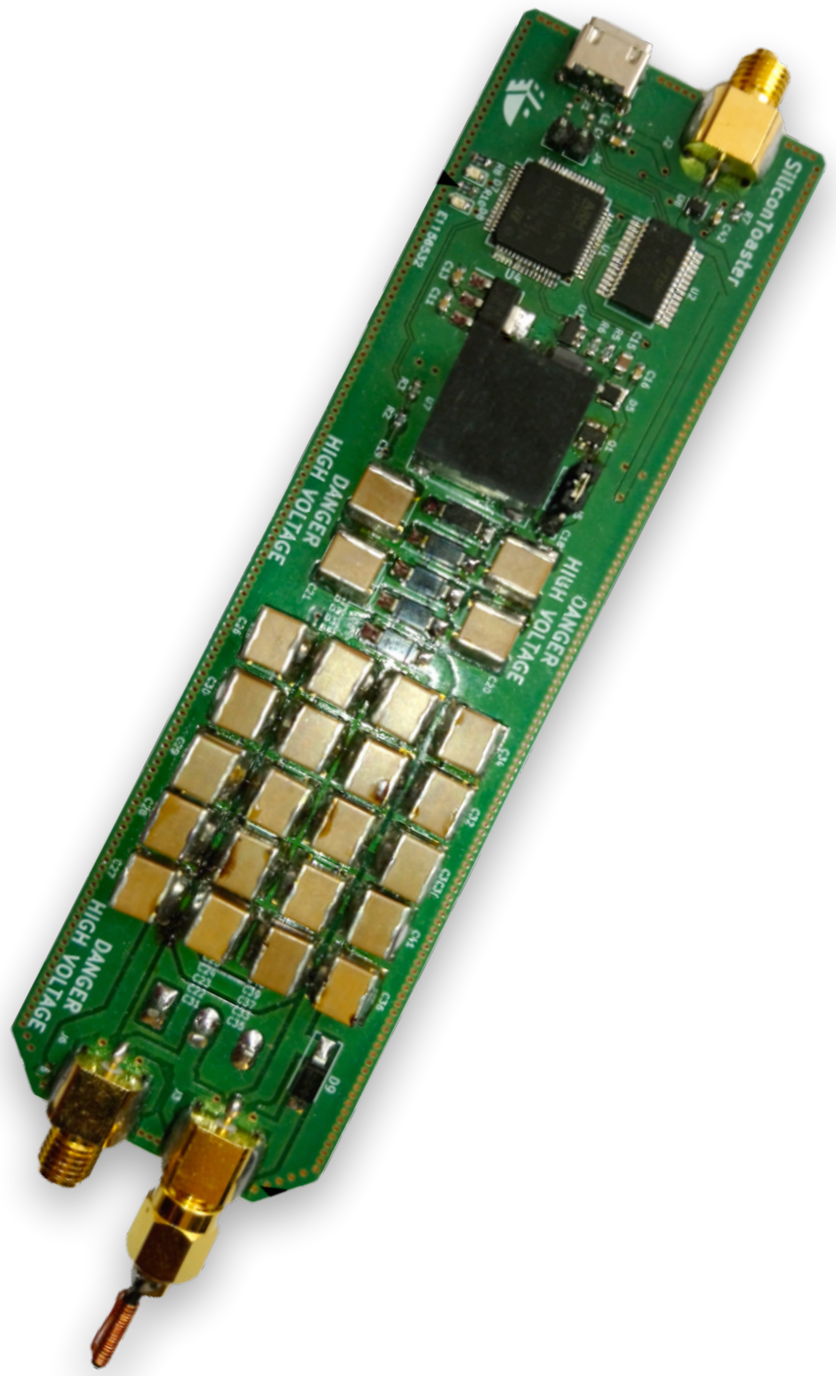




https://www.newae.com/products/NAE-CWHUSKY

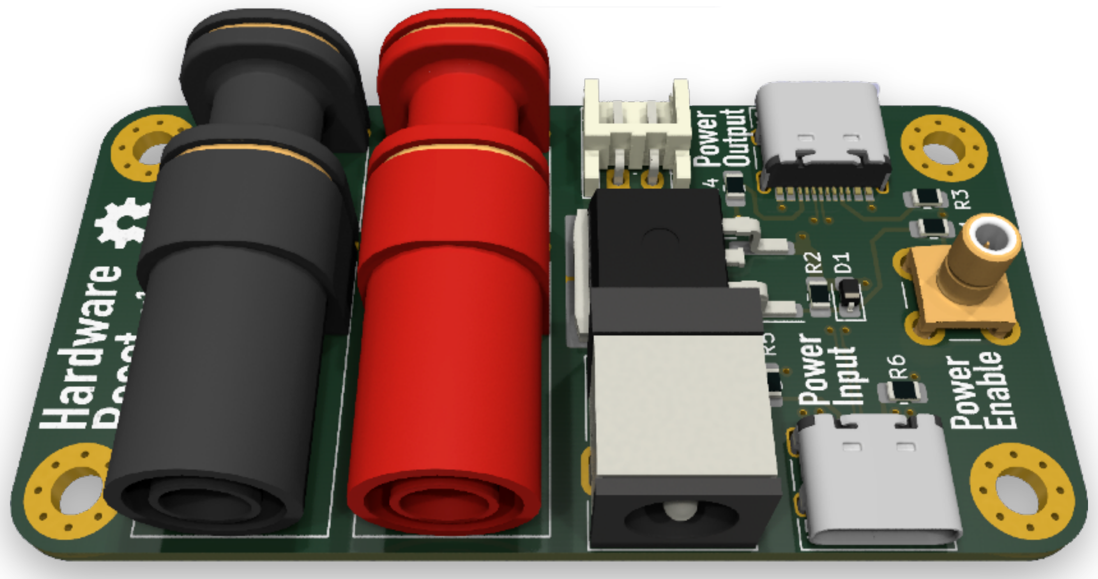
EMFI Setup
EMFI Pulser

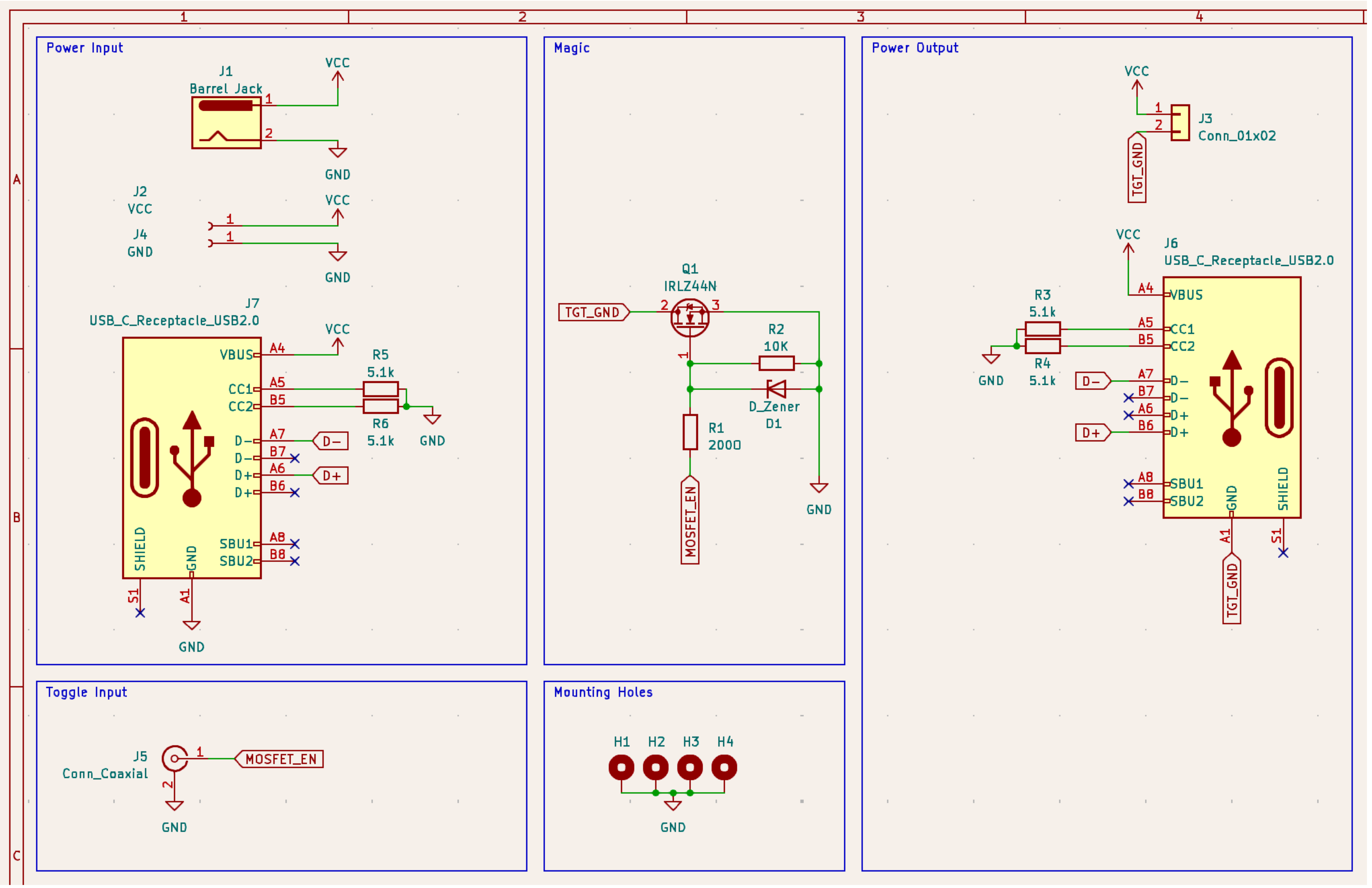
- ChipSHOUTER by @colinoflynn
- PicoEMP by @colinoflynn, @stacksmashing et al.
- SiliconToaster by Ledger



EMFI Setup

Hardware Reset

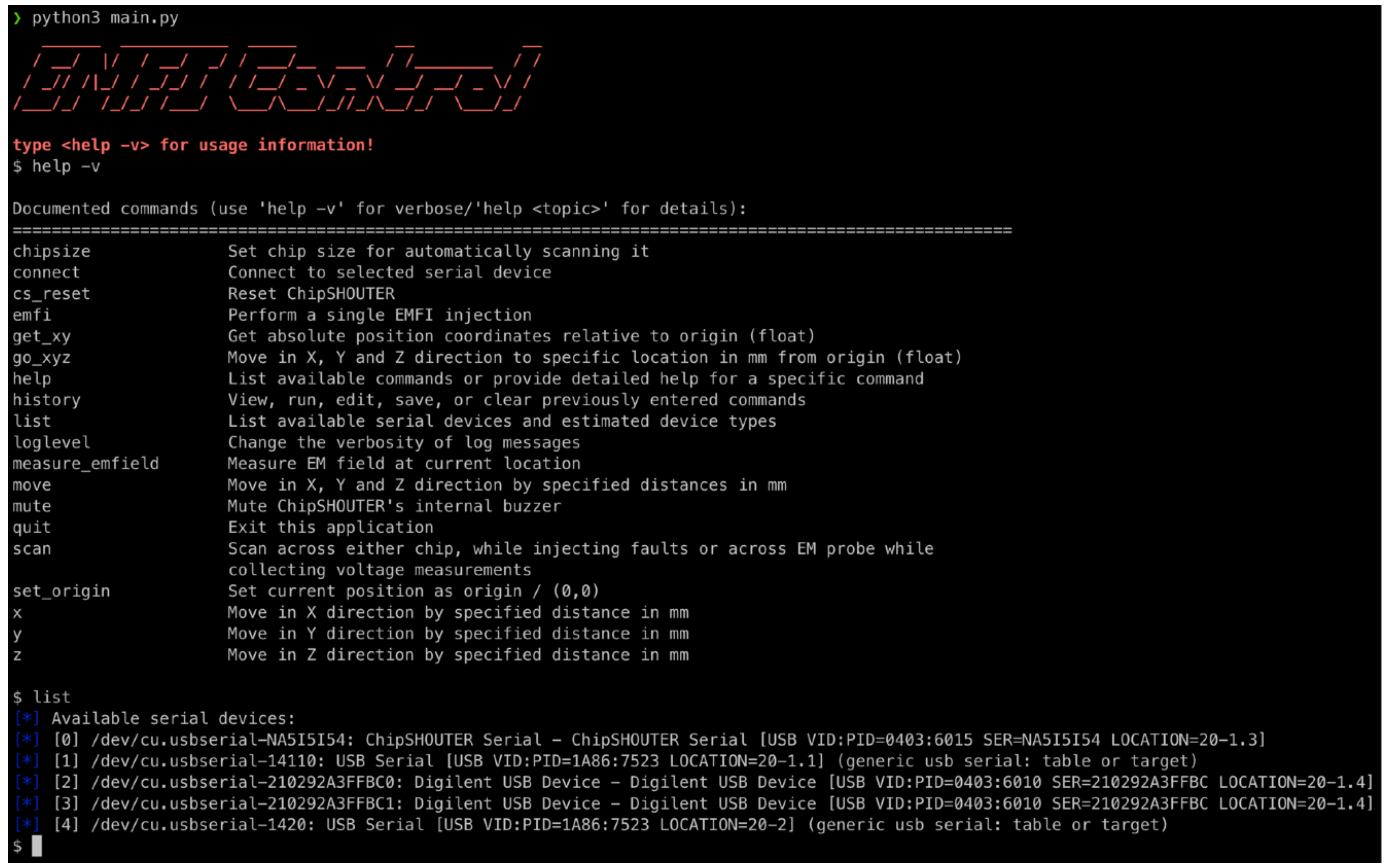




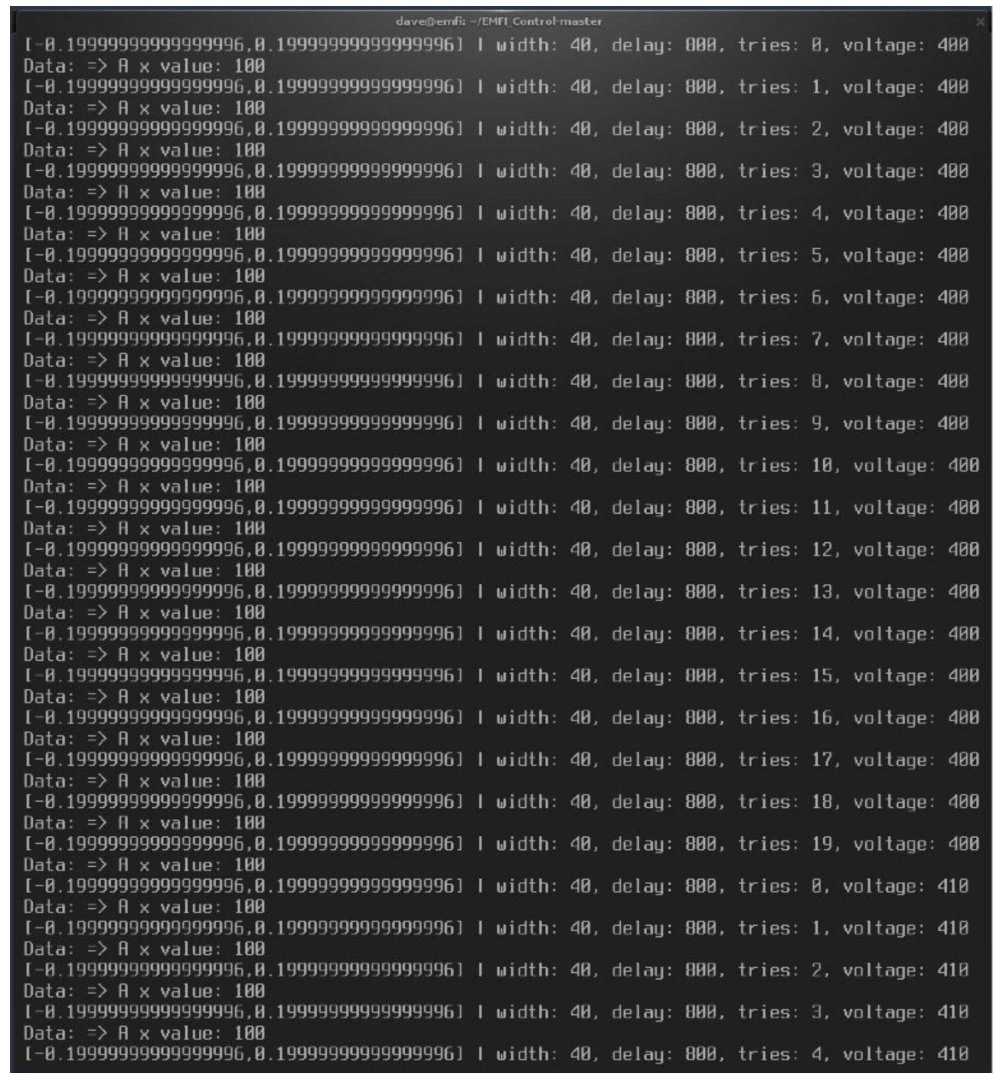

```
# Prepare hardware
platform.move(0,0)

glitcher.arm()

delay.set_delay(100)
delay.set_width(100)
delay.arm()

# Reset target
hw_reset.reset()

# Wait for result
res = target.read()
```

EMFI Setup

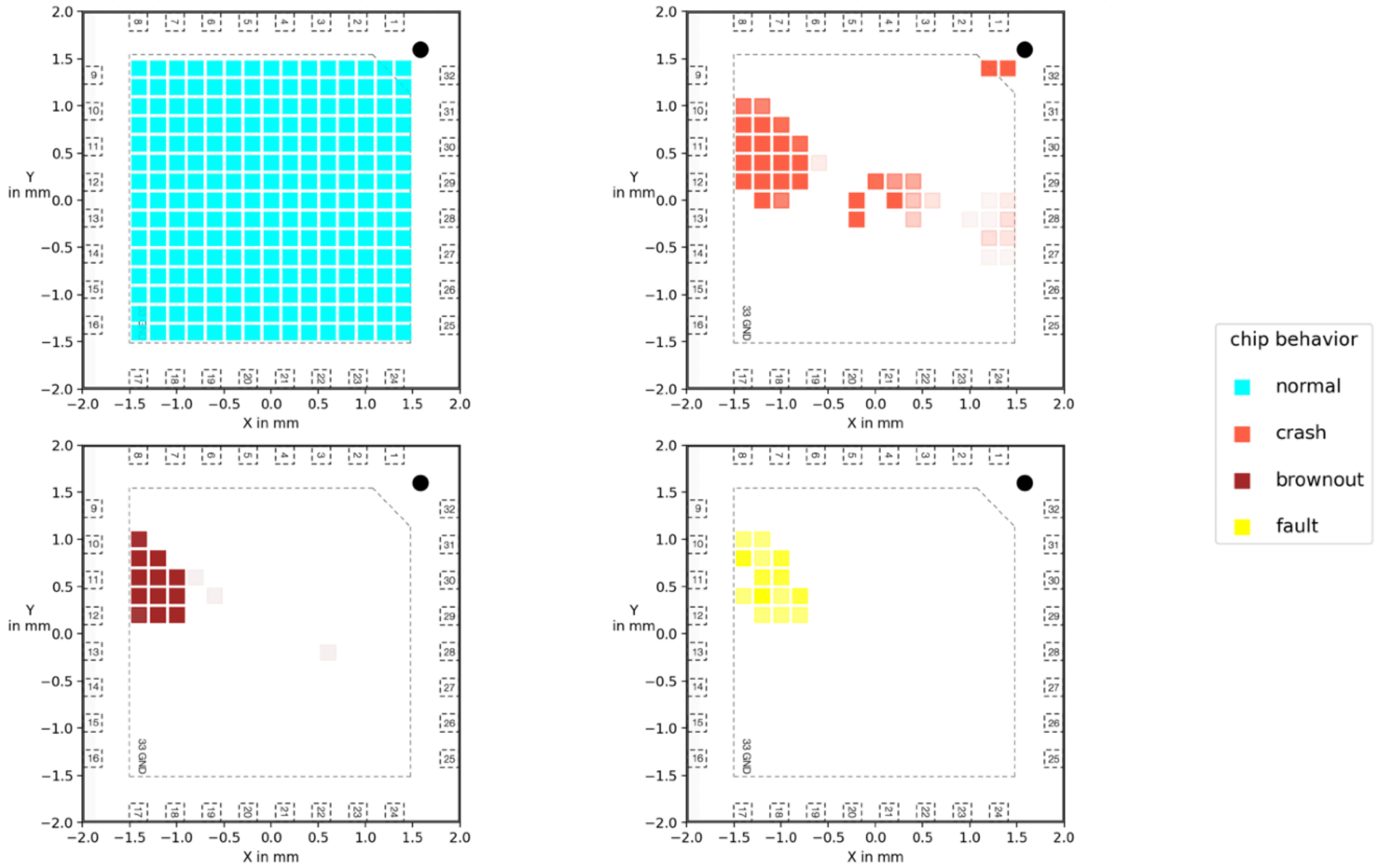
Custom Software



EMFIControl

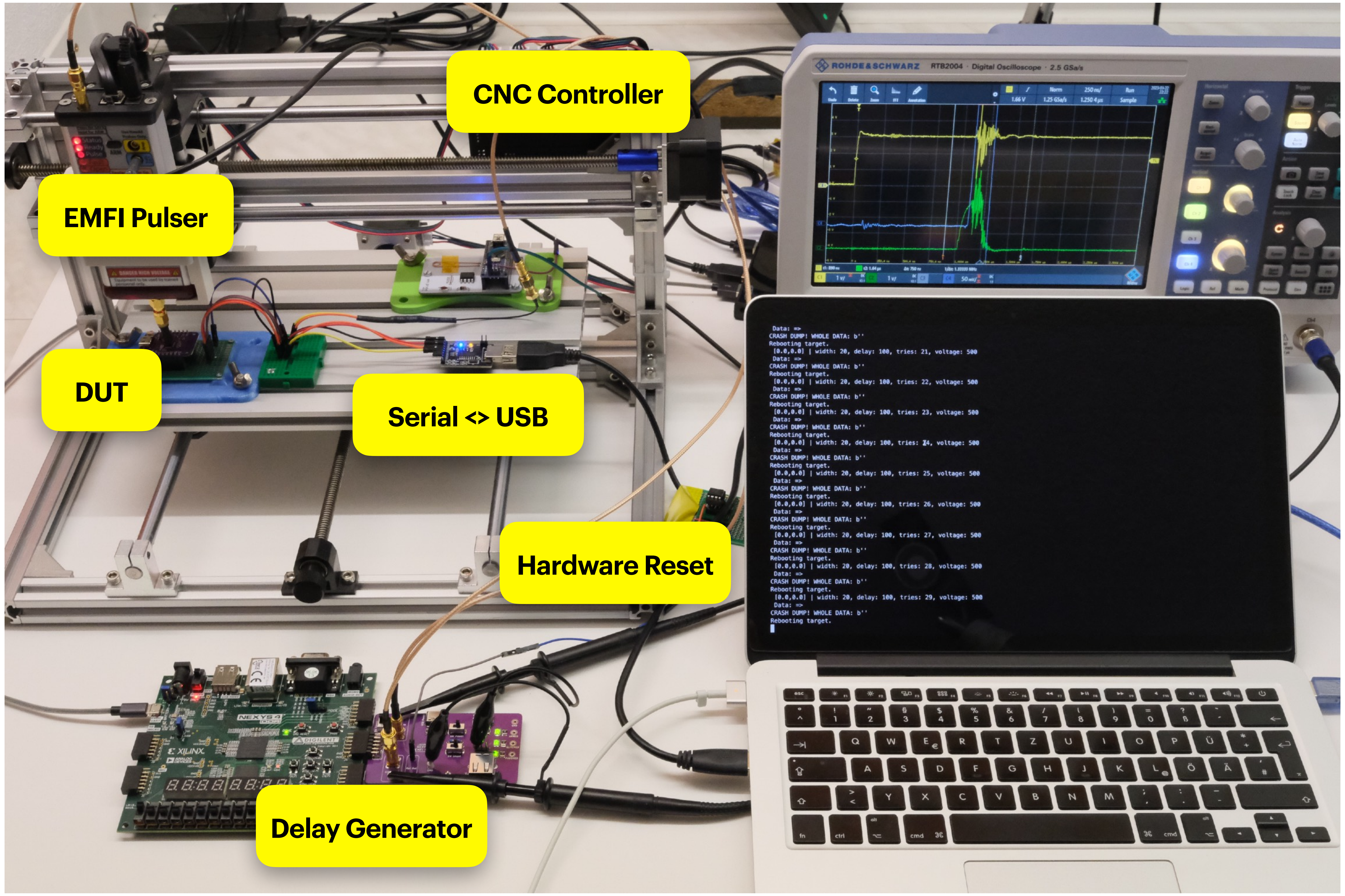


Fault campaign

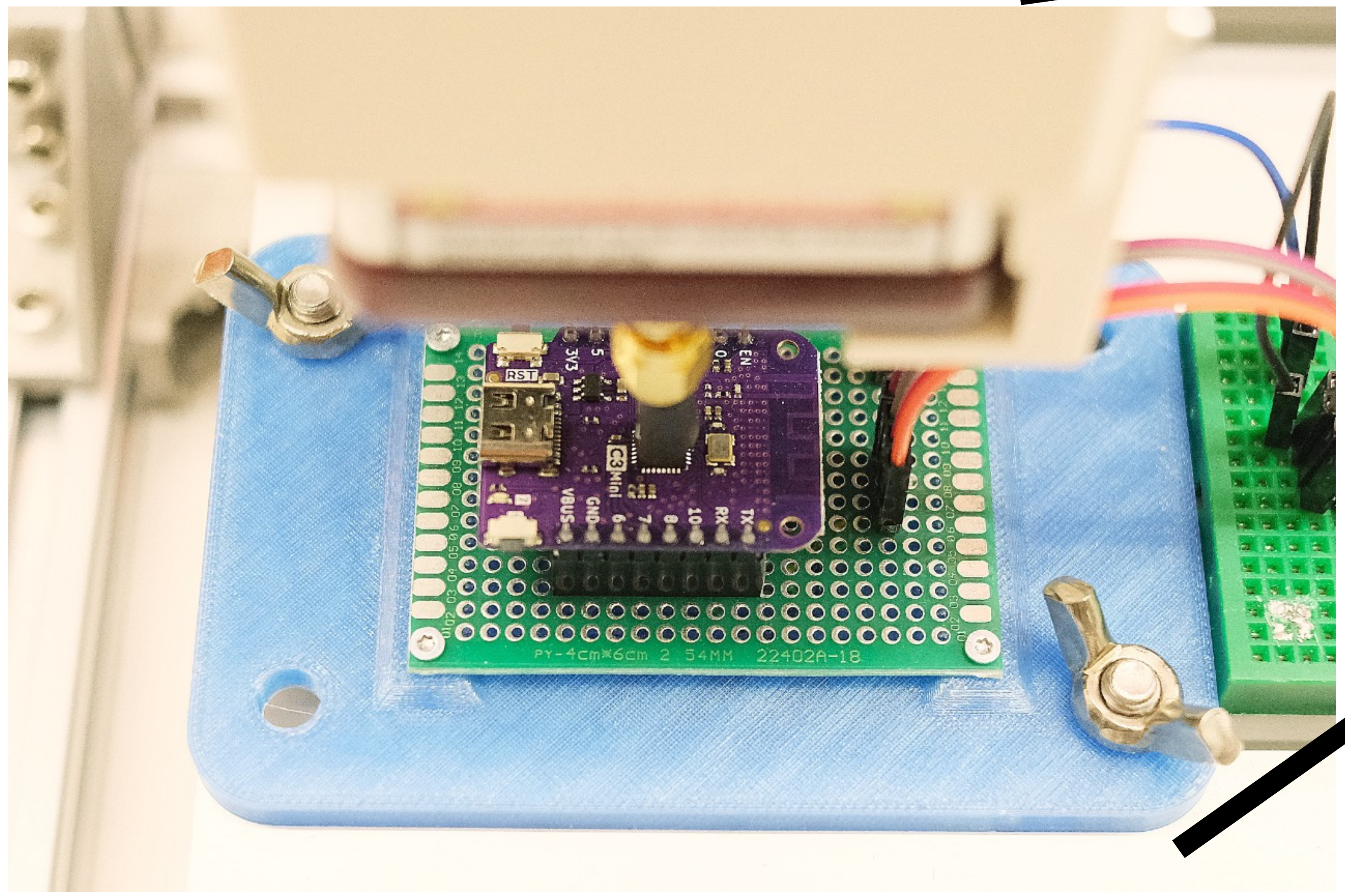


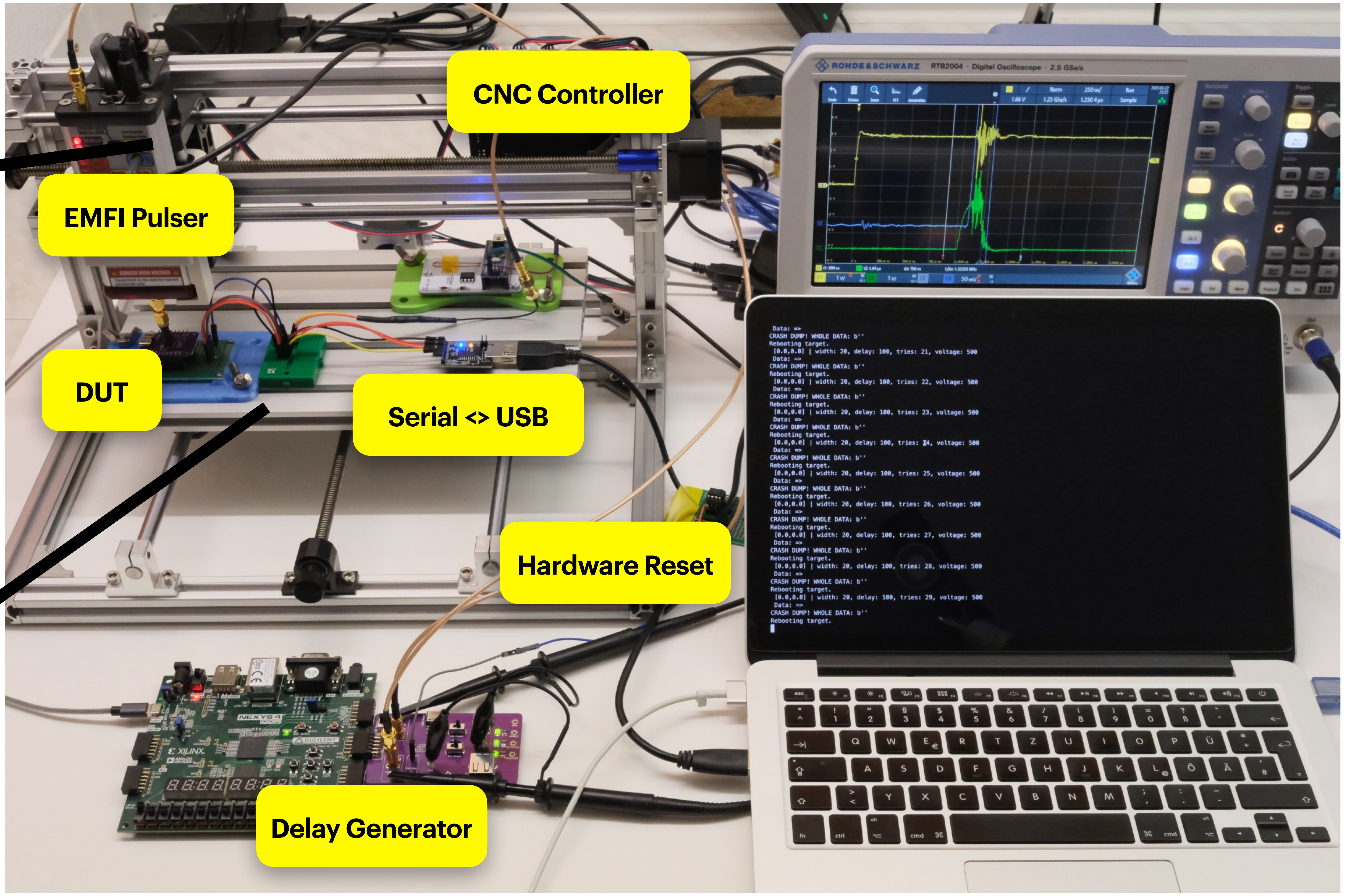
Result visualization

EMFI Setup

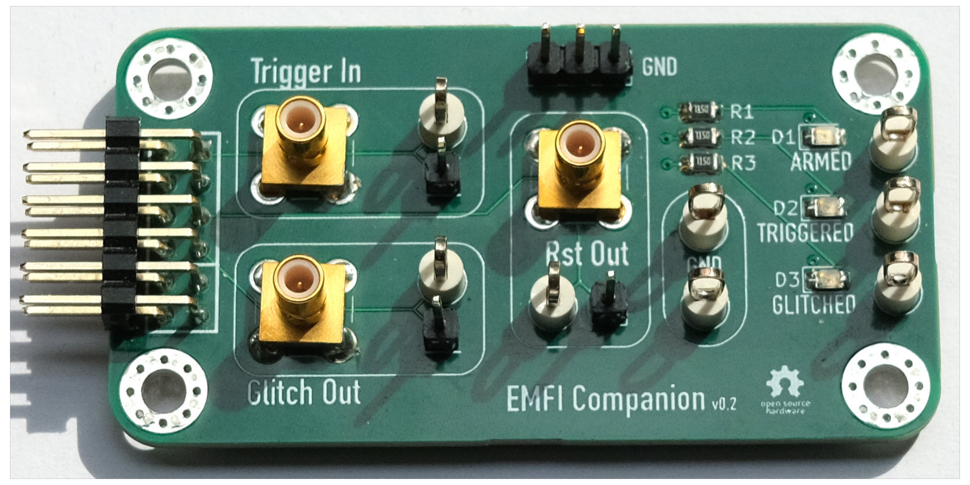

CNC Controller
EMFI Pulser
DUT
Serial <> USB
Hardware Reset
Delay Generator

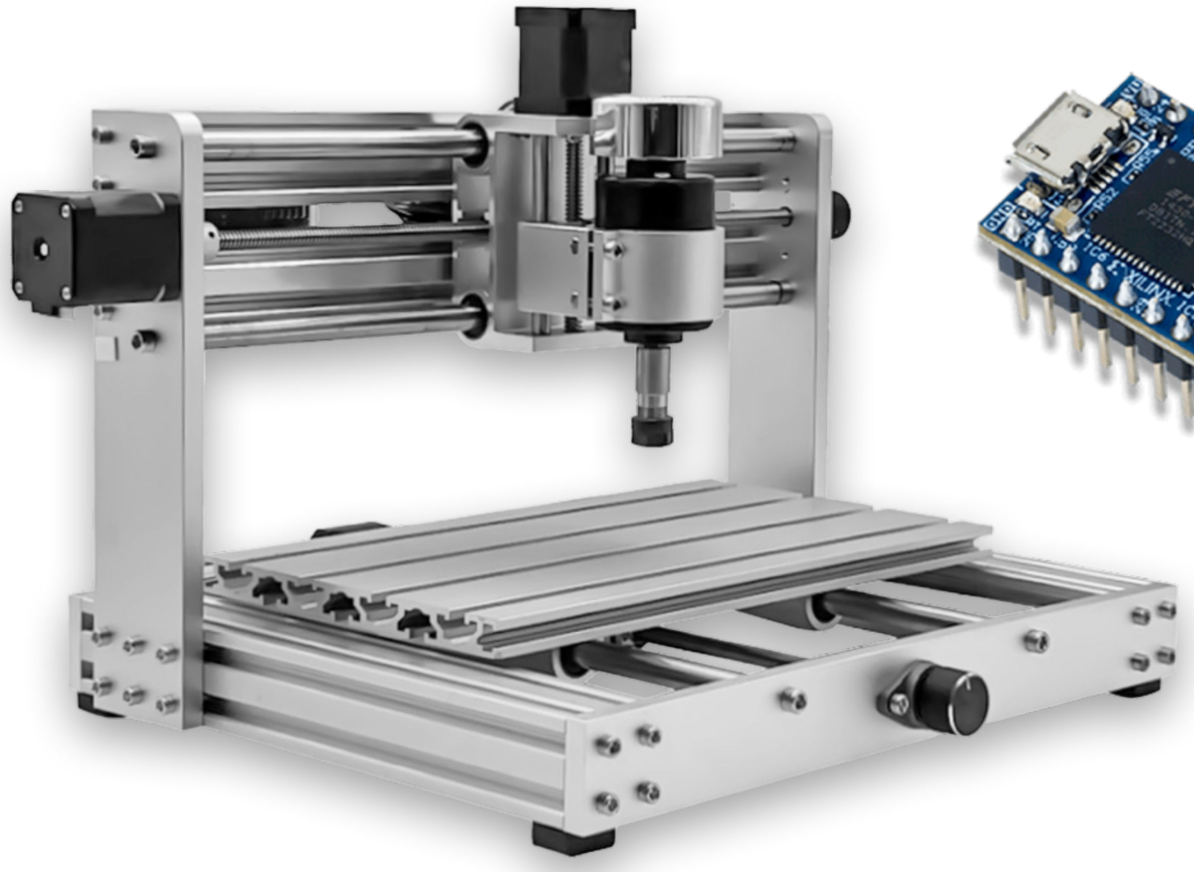
EMFI Setup

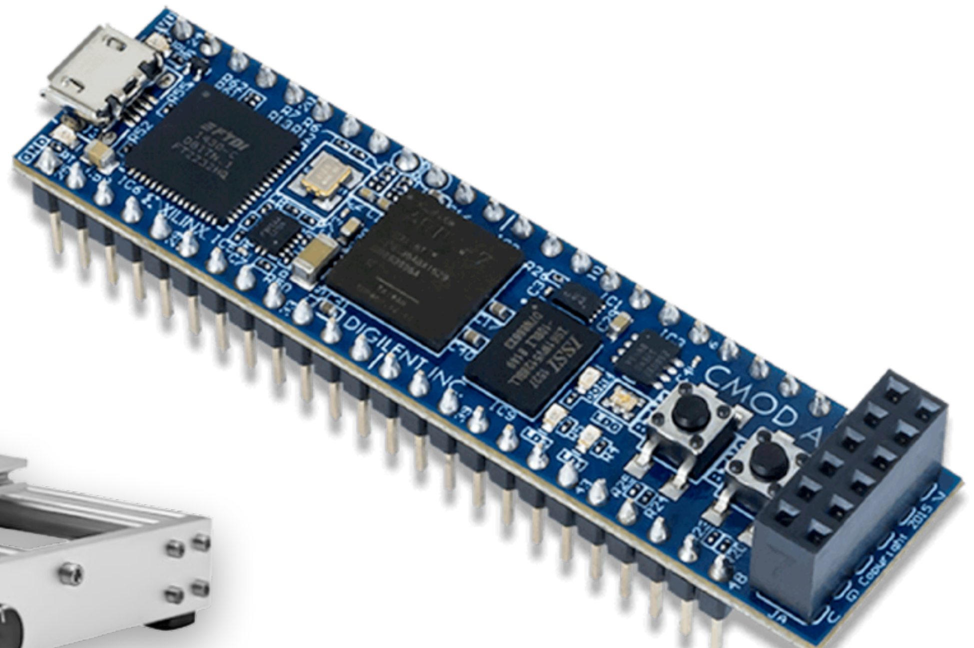


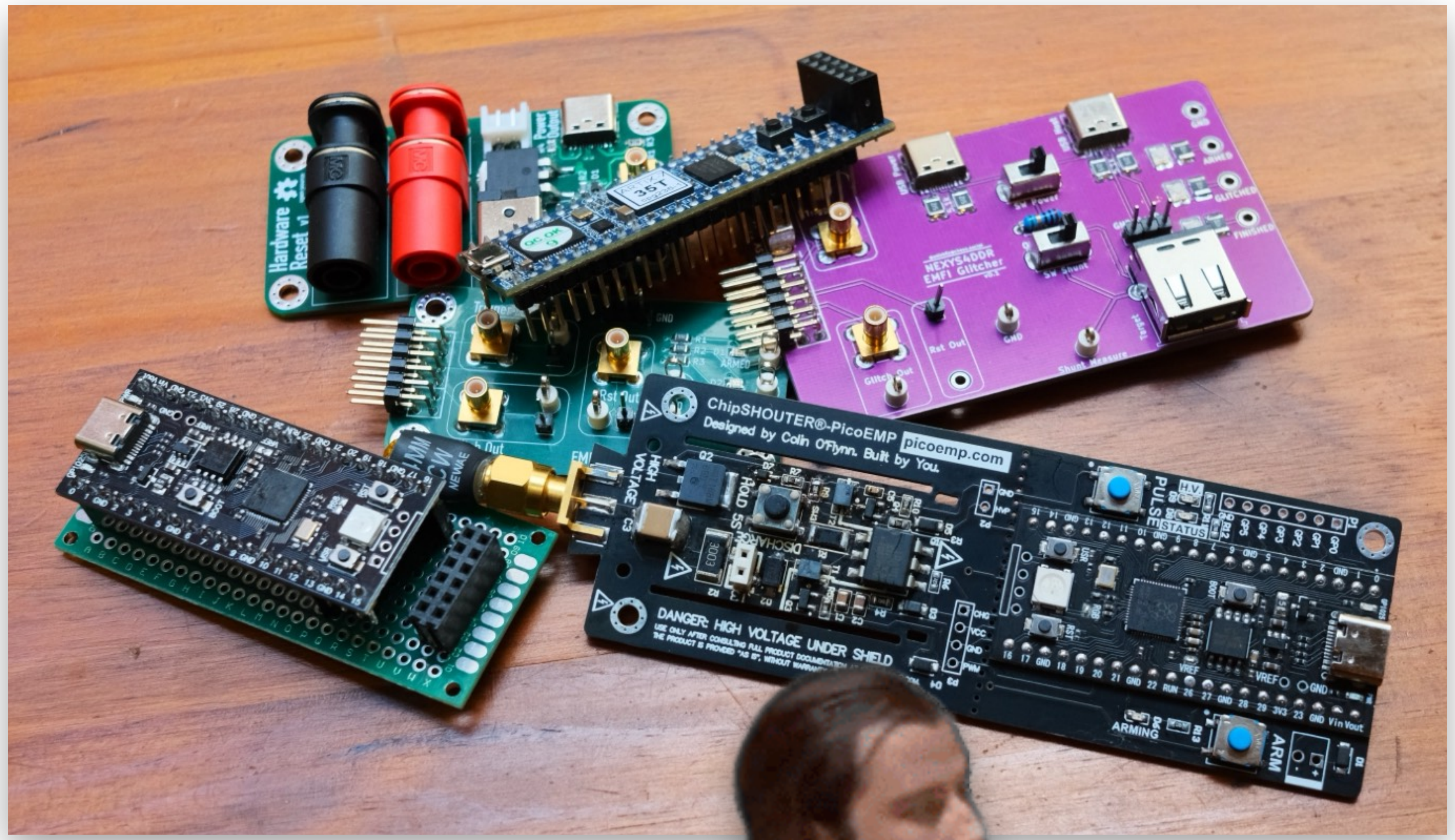

CNC Controller
EMFI Pulser
DUT
Serial <> USB
Hardware Reset
Delay Generator

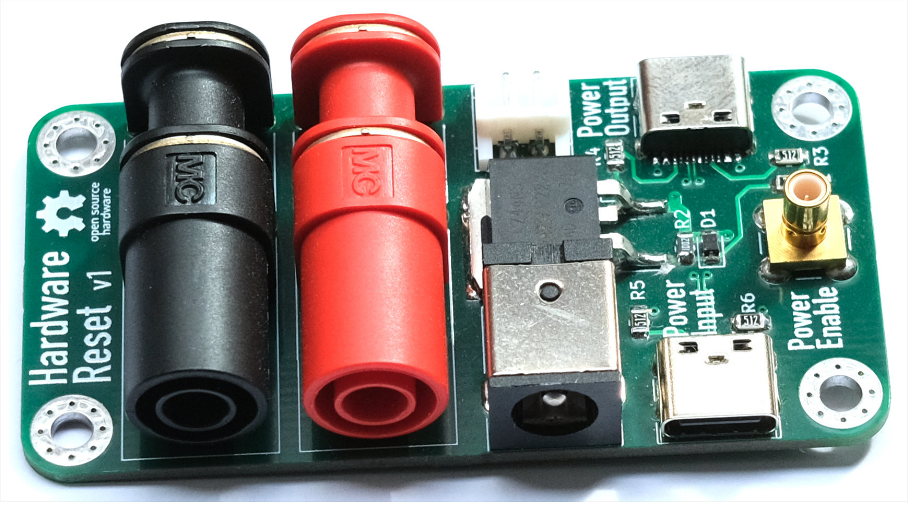
What are we actually doing here?

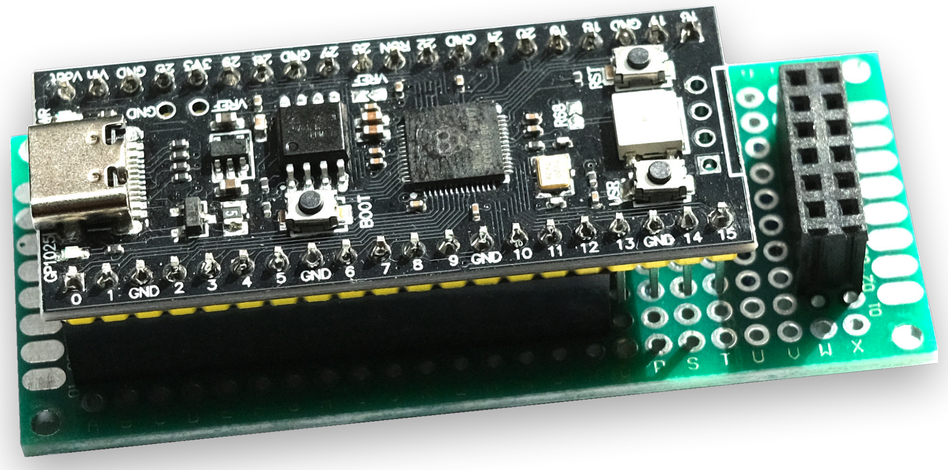


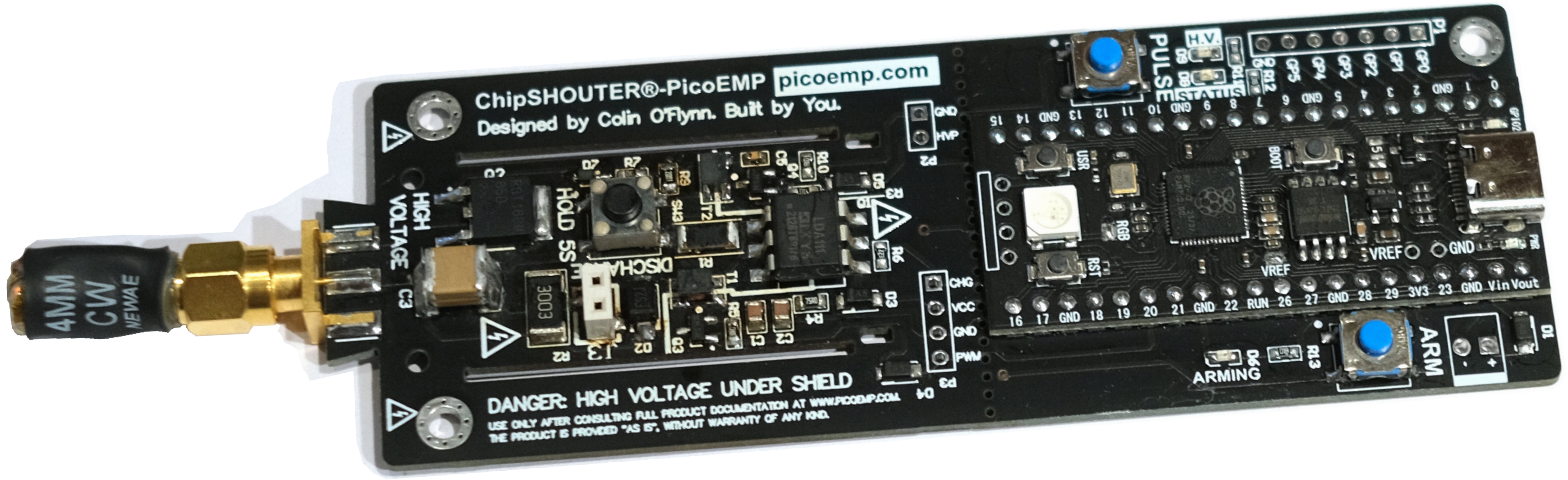


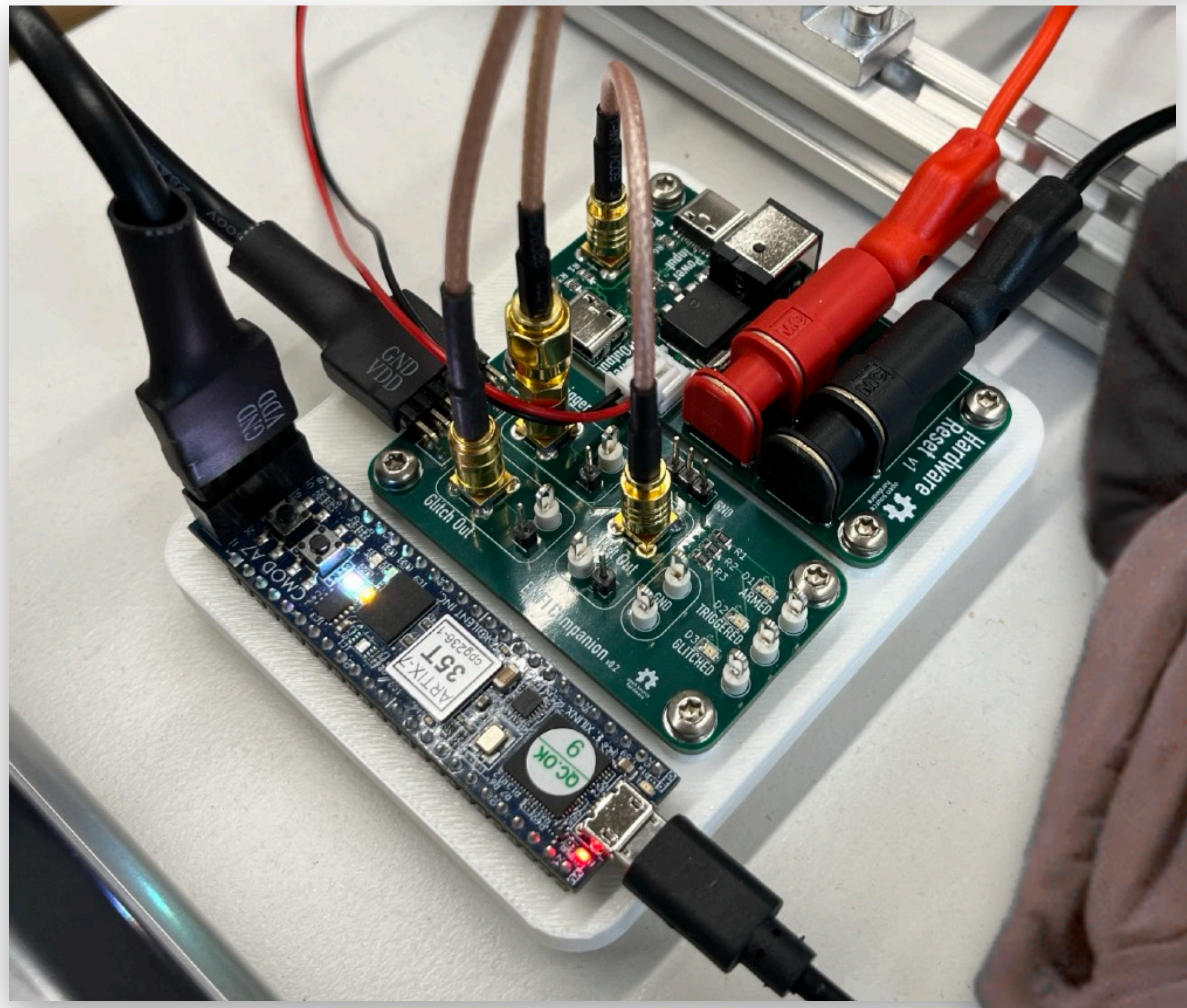






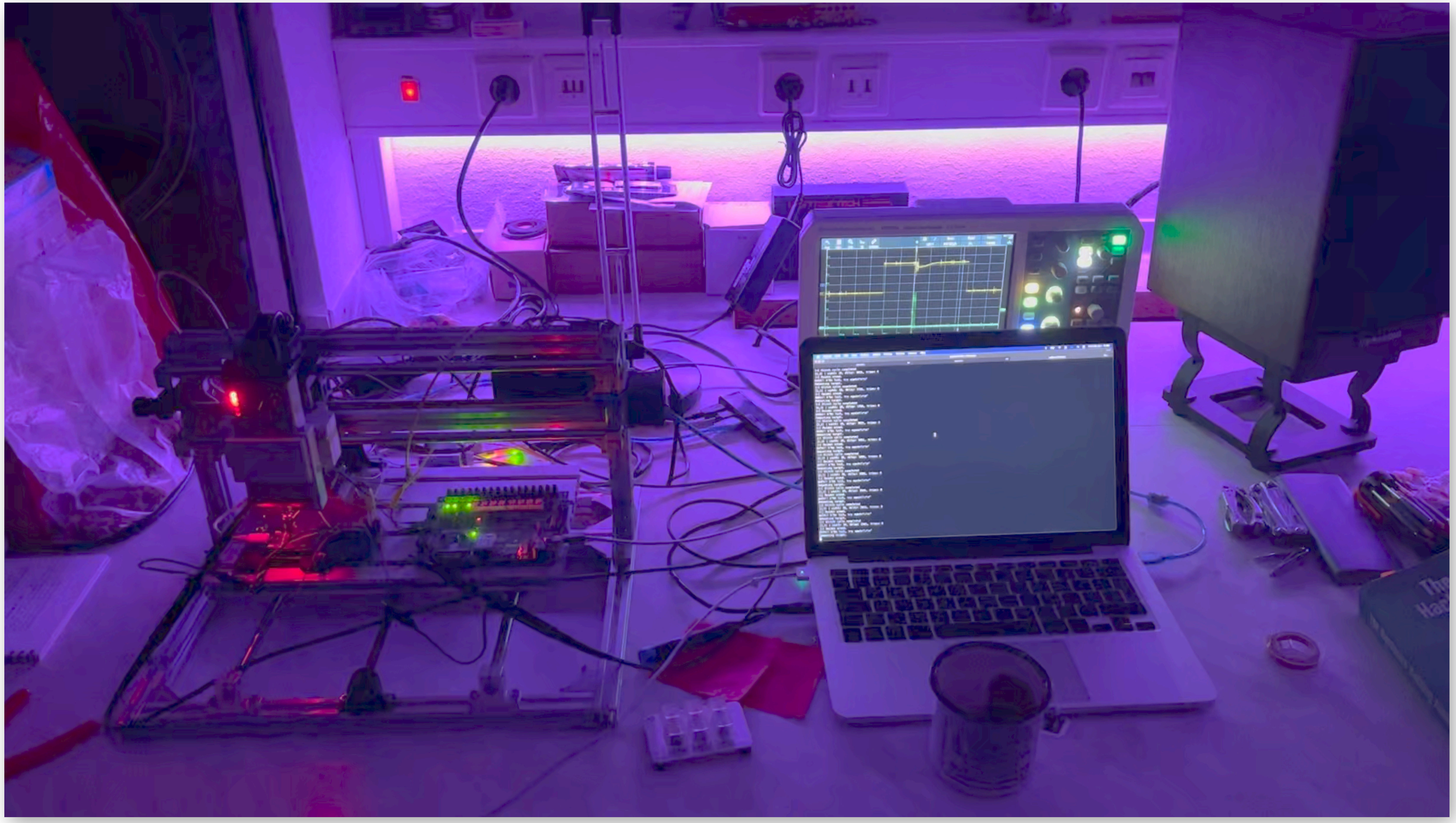






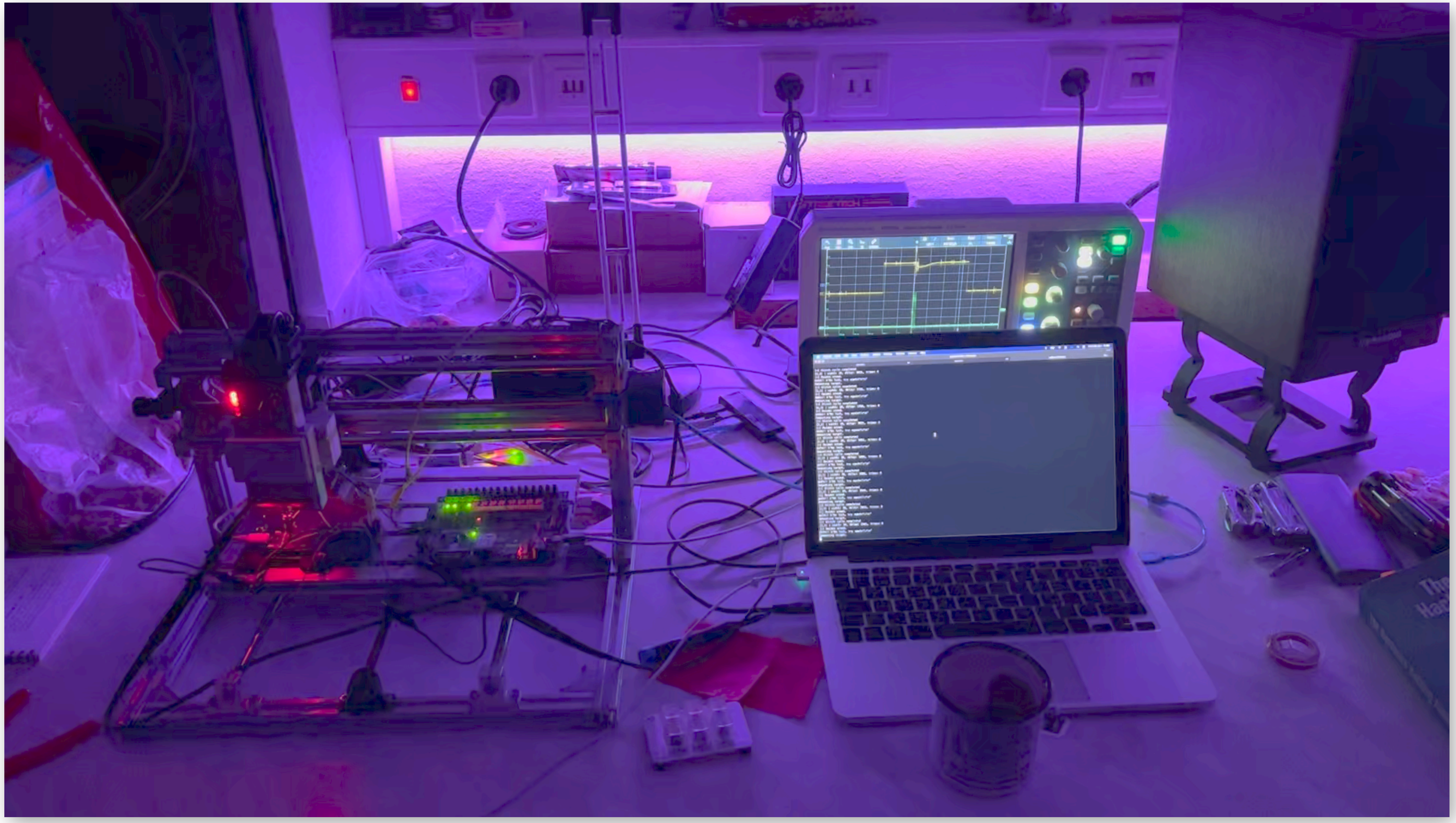
Results

IoT chips with "physical security features"



Results

IoT chips with "physical security features"



Results

IoT chips with "physical security features"

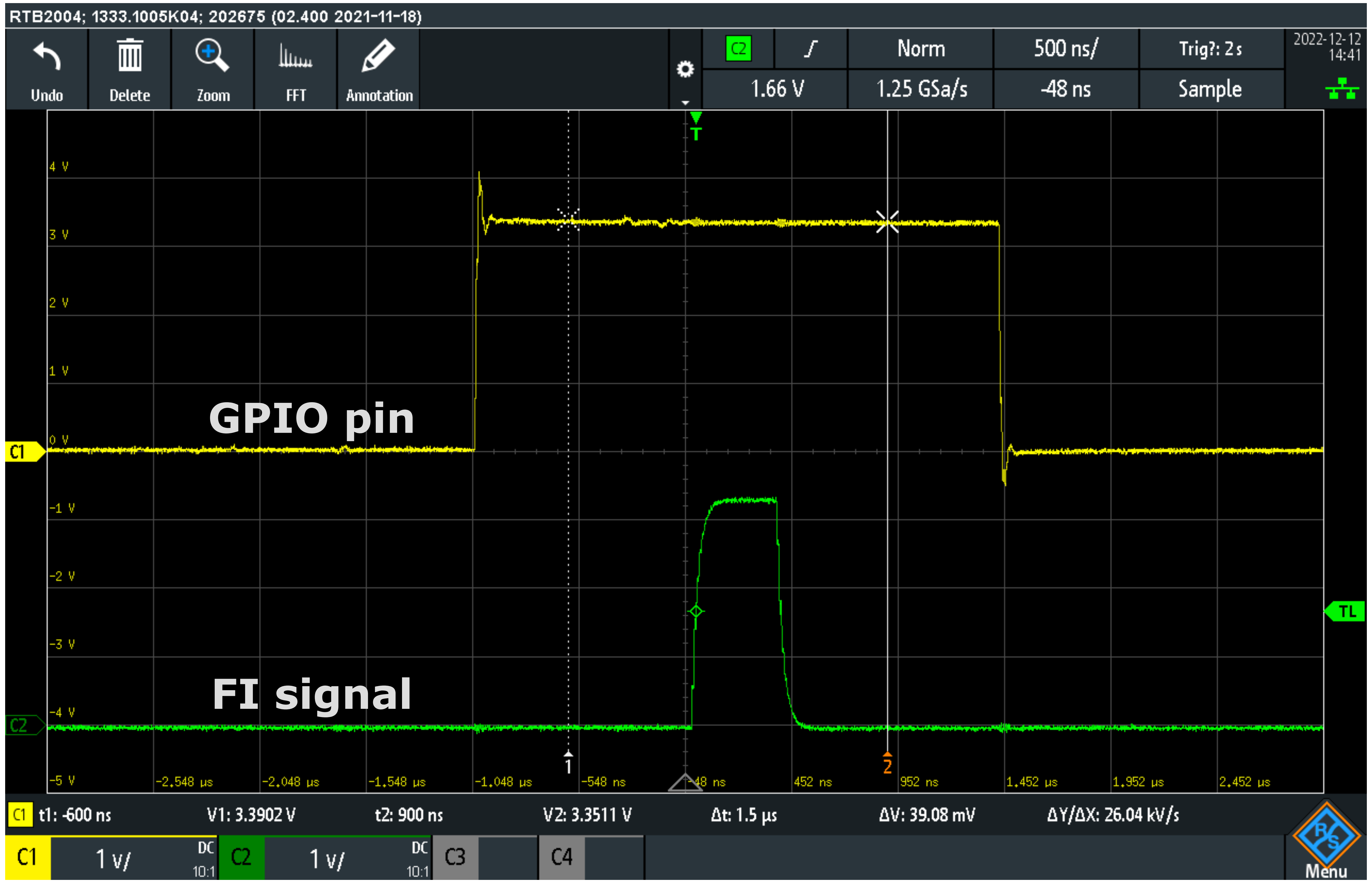
3.10 Physical Security Features

- Transparent off-package flash encryption (AES-XTS algorithm) with software inaccessible key prevents unauthorized readout of your application code or data.
- Secure boot feature uses a hardware root of trust to ensure only signed firmware (with RSA-PSS signature) can be booted.
- HMAC module can use a software inaccessible MAC key to generate MAC signatures for identity verification and other purposes.
- Digital Signature module can use a software inaccessible secure key to generate RSA signatures for identity verification.
- World Controller provides two running environments for software. All hardware and software resources are sorted to two groups, and placed in either secure or general world. The secure world cannot be accessed by hardware in the general world, thus establishing a security boundary.

Results

IoT chips with "physical security features"

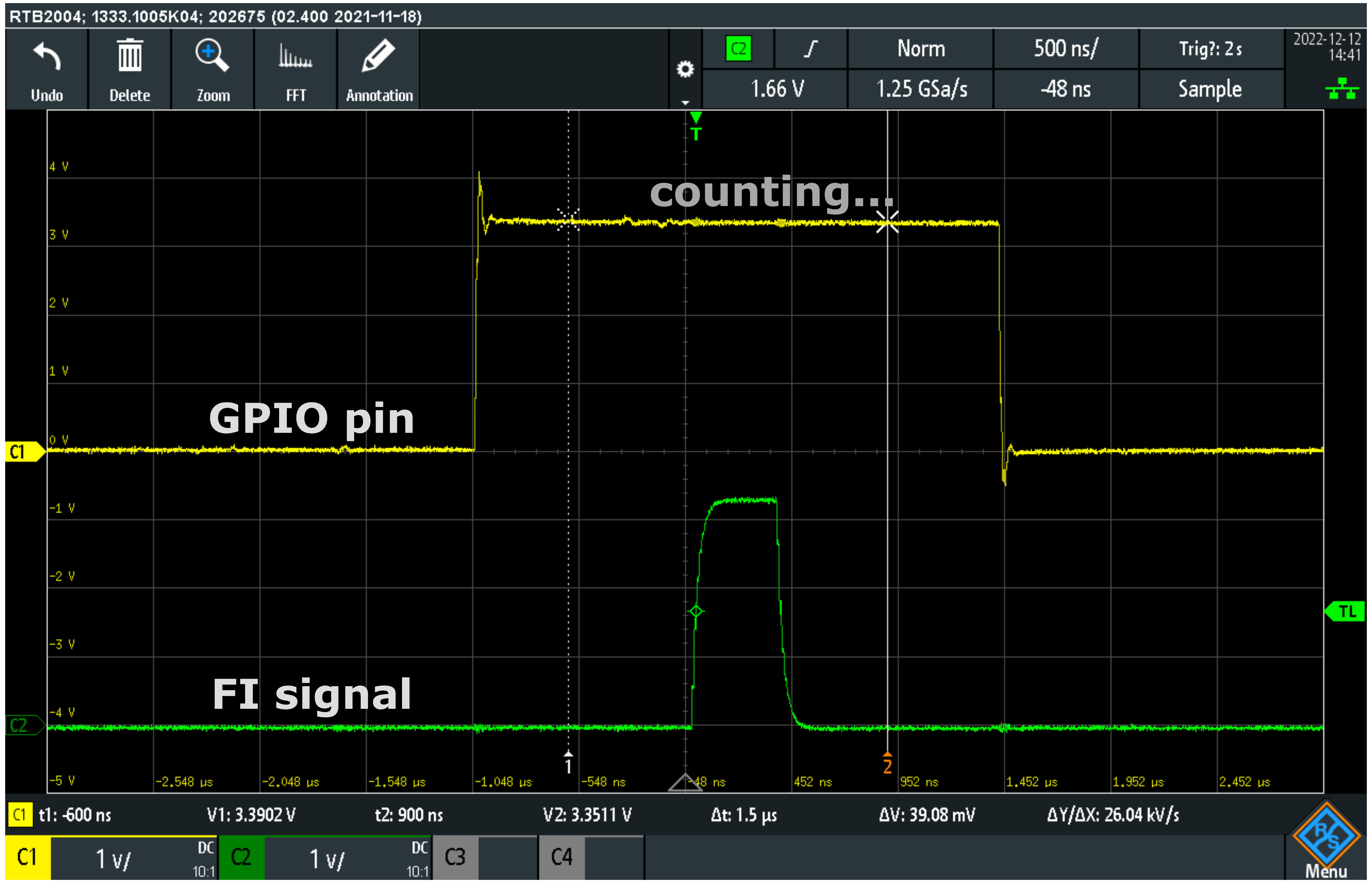
- Loop test:
  1. Trigger GPIO pin high
  2. Count from 0 to 255
  3. Trigger GPIO pin low
  4. Check the result



Results

IoT chips with "physical security features"

- Loop test:
  1. Trigger GPIO pin high
  2. Count from 0 to 255
  3. Trigger GPIO pin low
  4. Check the result



Results

IoT chips with "physical security features"

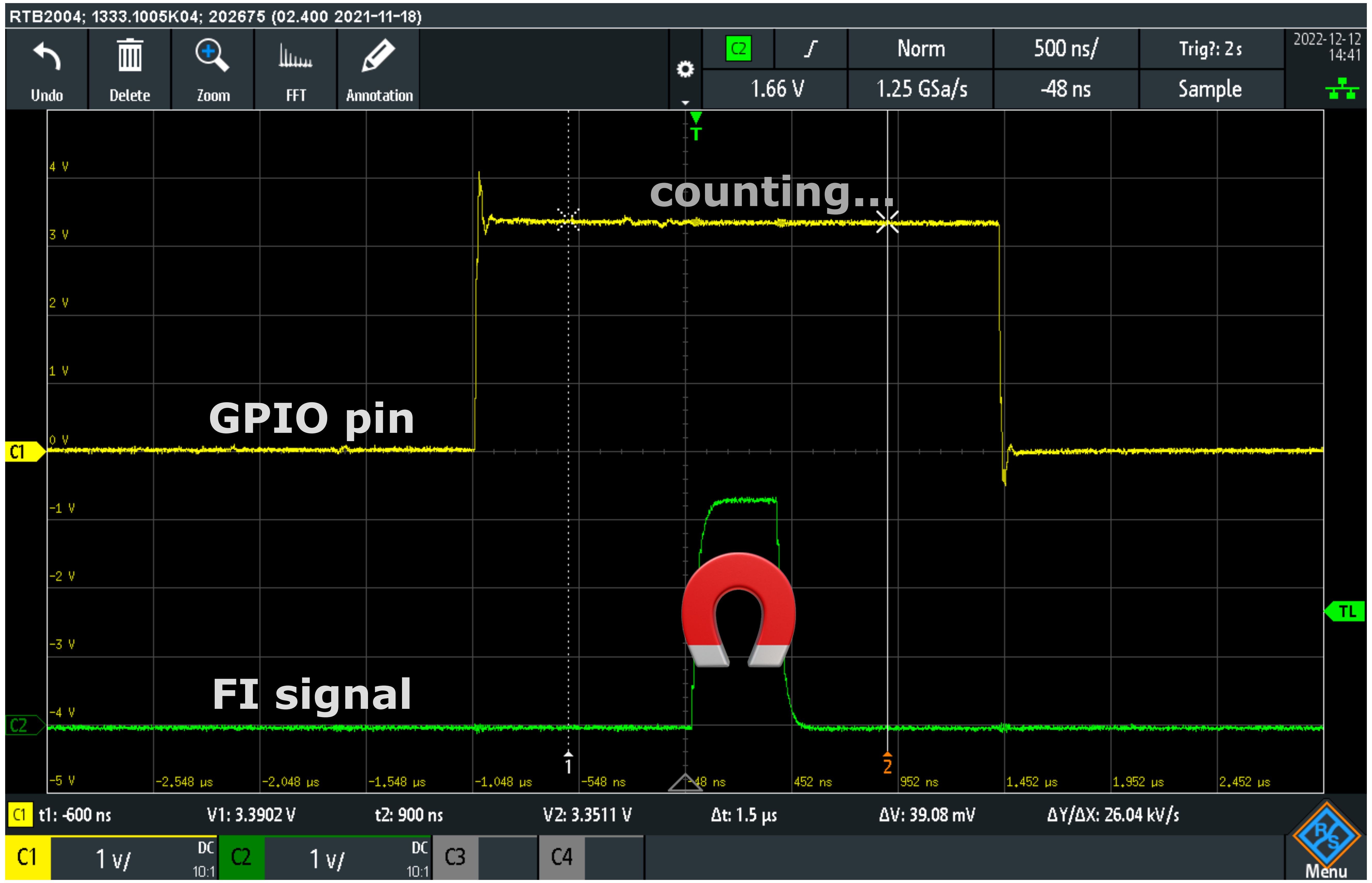
- Loop test:

  1. Trigger GPIO pin high

  2. Count from 0 to 255

  3. Trigger GPIO pin low

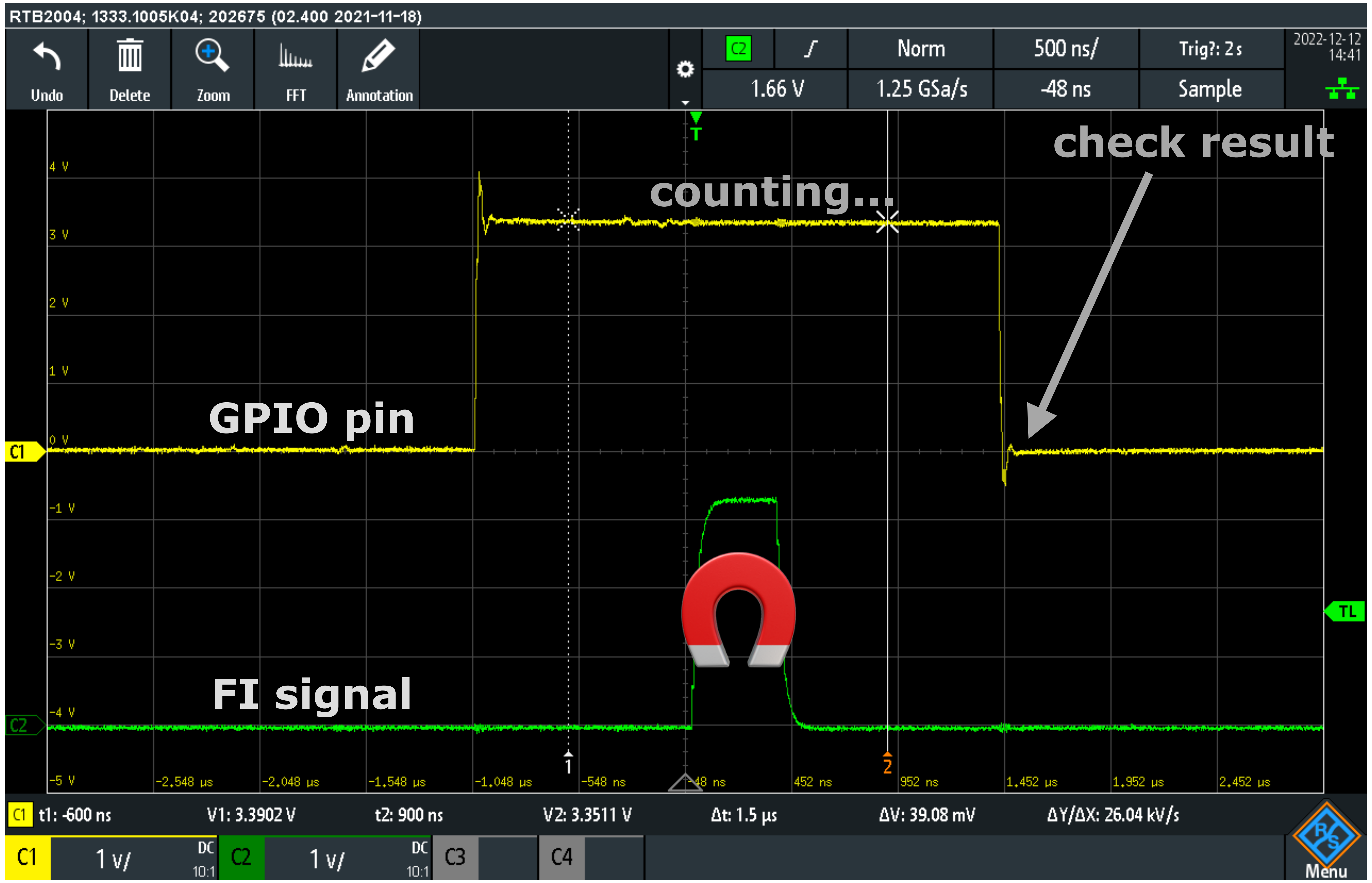
  4. Check the result



Results

IoT chips with "physical security features"

- Loop test:
  1. Trigger GPIO pin high
  2. Count from 0 to 255
  3. Trigger GPIO pin low
  4. Check the result

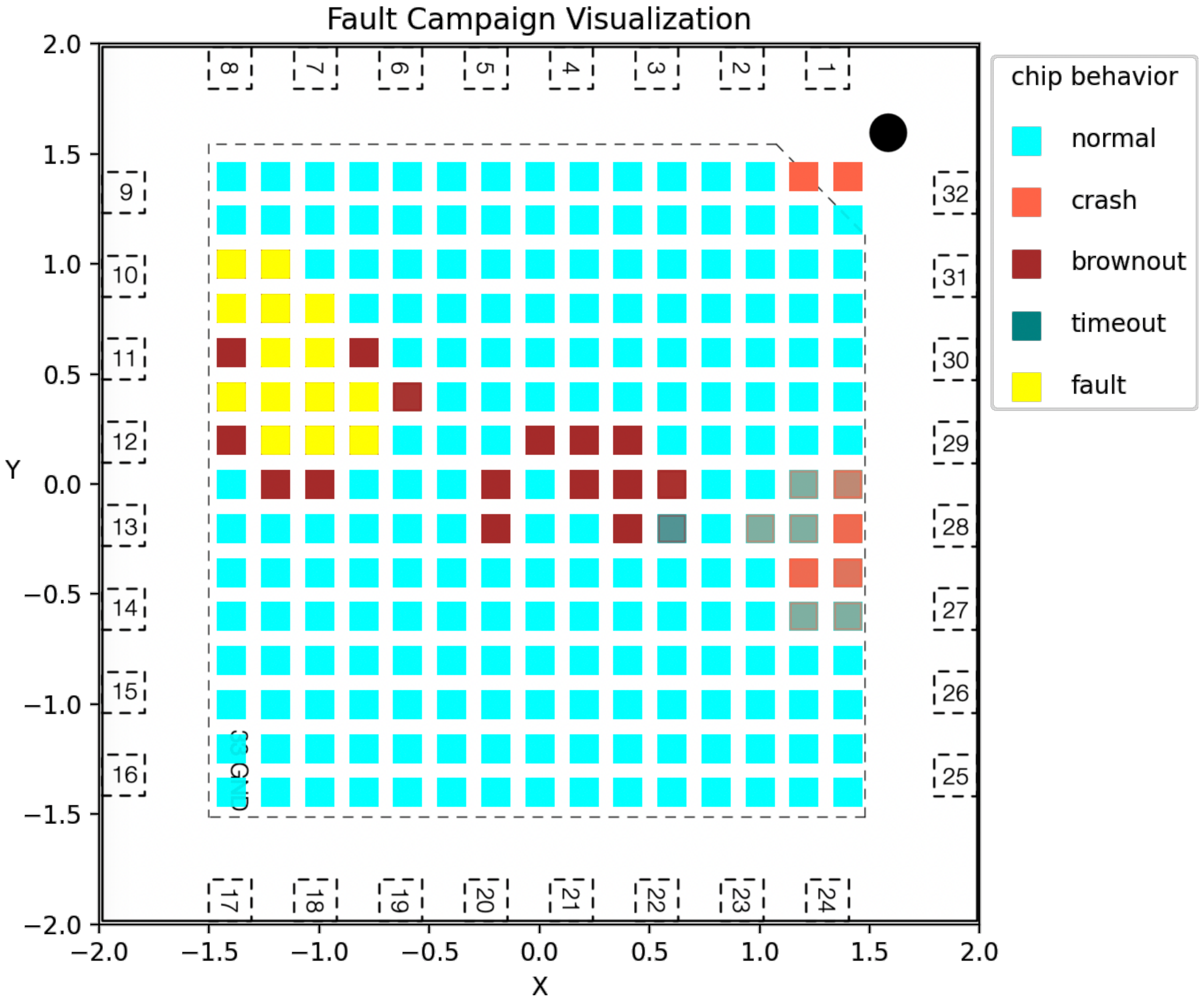


Fault Campaign

Visualization of chip behavior under faults

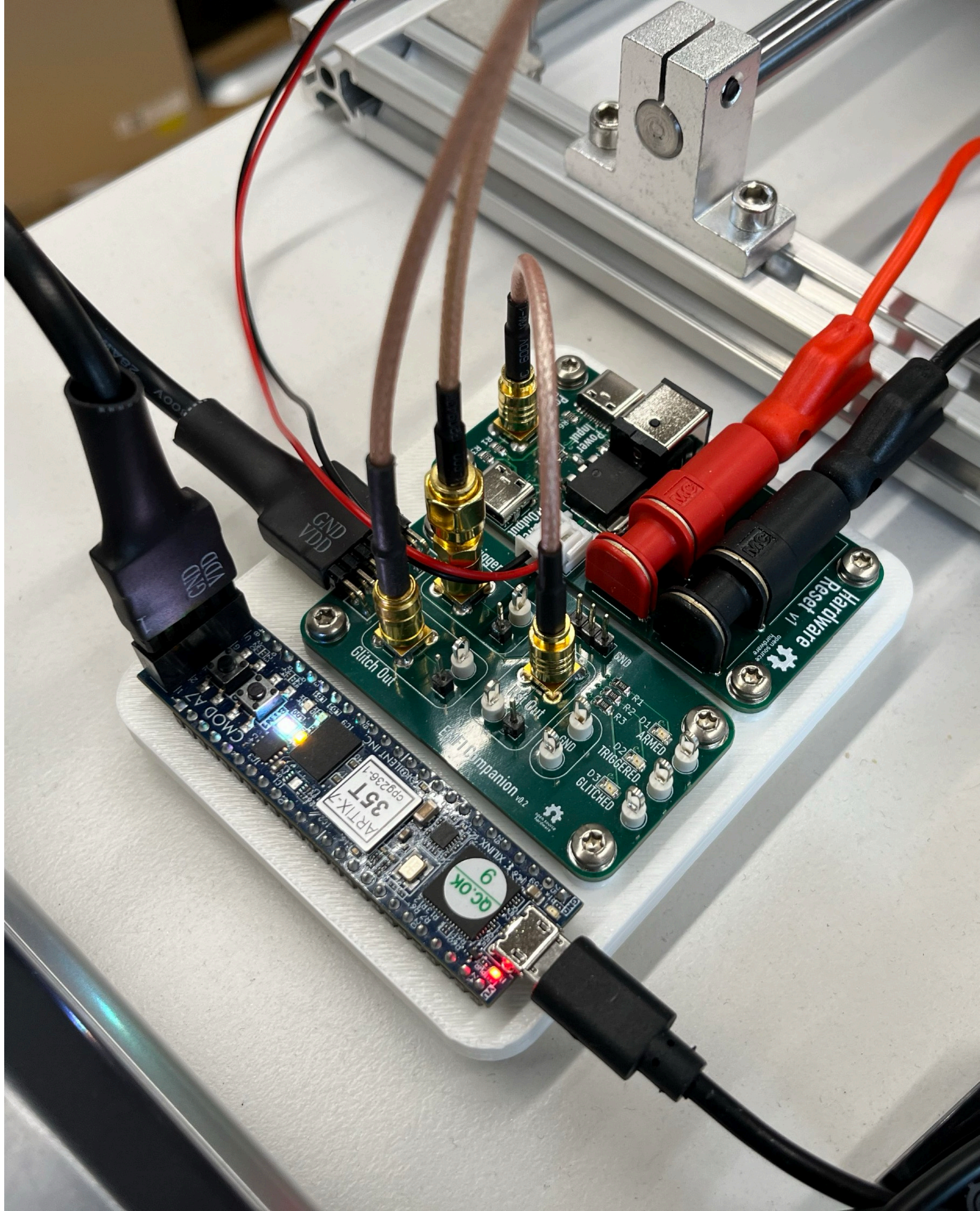
- Successful faults, i.e. instruction skips in the top right corner
- Could be applied to any app-level code, bootloader, custom security measures, ... using the right timings

try_num	x	y	voltage	delay	width	normal	fault	brownout	timeout	crash	data
4	-1.0	1.0	470	100	20	FALSE	FALSE	FALSE	FALSE	TRUE	b''
5	-1.0	1.0	470	100	20	FALSE	FALSE	TRUE	FALSE	FALSE	b'dESP-ROM:esp32c3-api1-2(
6	-1.0	1.0	470	100	20	TRUE	FALSE	FALSE	FALSE	FALSE	b'No luck, try again! 100||\r\n'
7	-1.0	1.0	470	100	20	FALSE	FALSE	FALSE	FALSE	TRUE	b''
8	-1.0	1.0	470	100	20	FALSE	FALSE	TRUE	FALSE	FALSE	b'dESP-ROM:esp32c3-api1-2(
9	-1.0	1.0	470	100	20	TRUE	FALSE	FALSE	FALSE	FALSE	b'No luck, try again! 100||\r\n'
0	-1.0	1.0	480	100	20	FALSE	TRUE	FALSE	FALSE	FALSE	b'Glitch! 99||\r\n'
1	-1.0	1.0	480	100	20	FALSE	FALSE	TRUE	FALSE	FALSE	b'dESP-ROM:esp32c3-api1-2(
2	-1.0	1.0	480	100	20	FALSE	FALSE	FALSE	FALSE	TRUE	b''



Conclusion

- Low-cost, FOSS / OSHW setup
  - ~150€ X-Y stage
  - ~50-2000€ EMFI pulser
  - ~10€ delay generator
- Can be improved with little extra cost
  - 3D printer (belts)
  - Higher voltage pulser



Q & A

github.com/unixb0y/EMFI-Resources

@unixb0y@chaos.social

@unixb0y

dtoldo@seemoo.de