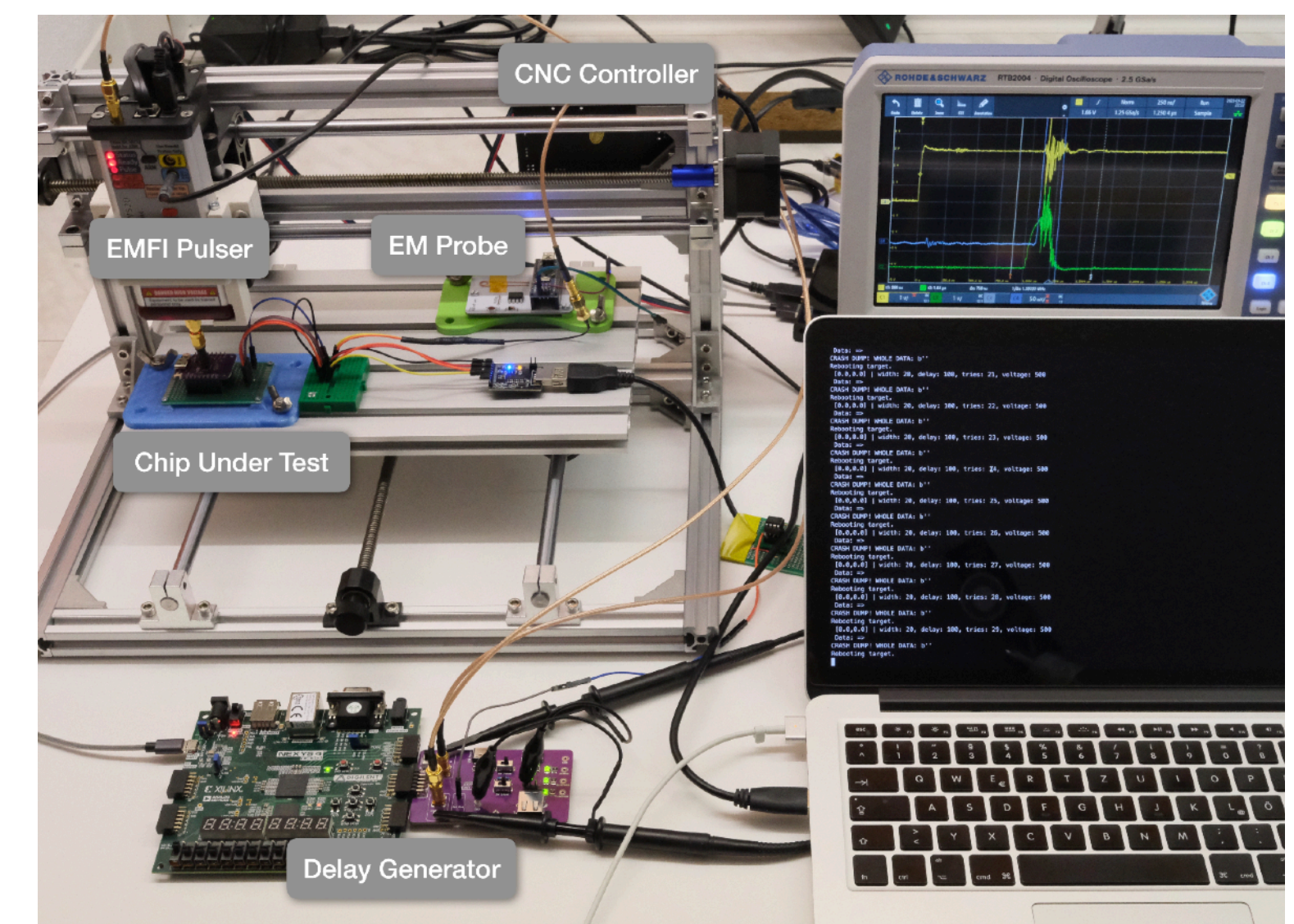


# Affordable\* EMFI Attacks Against Modern IoT Chips

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

**SEEMOO**  
SECURE MOBILE NETWORKING



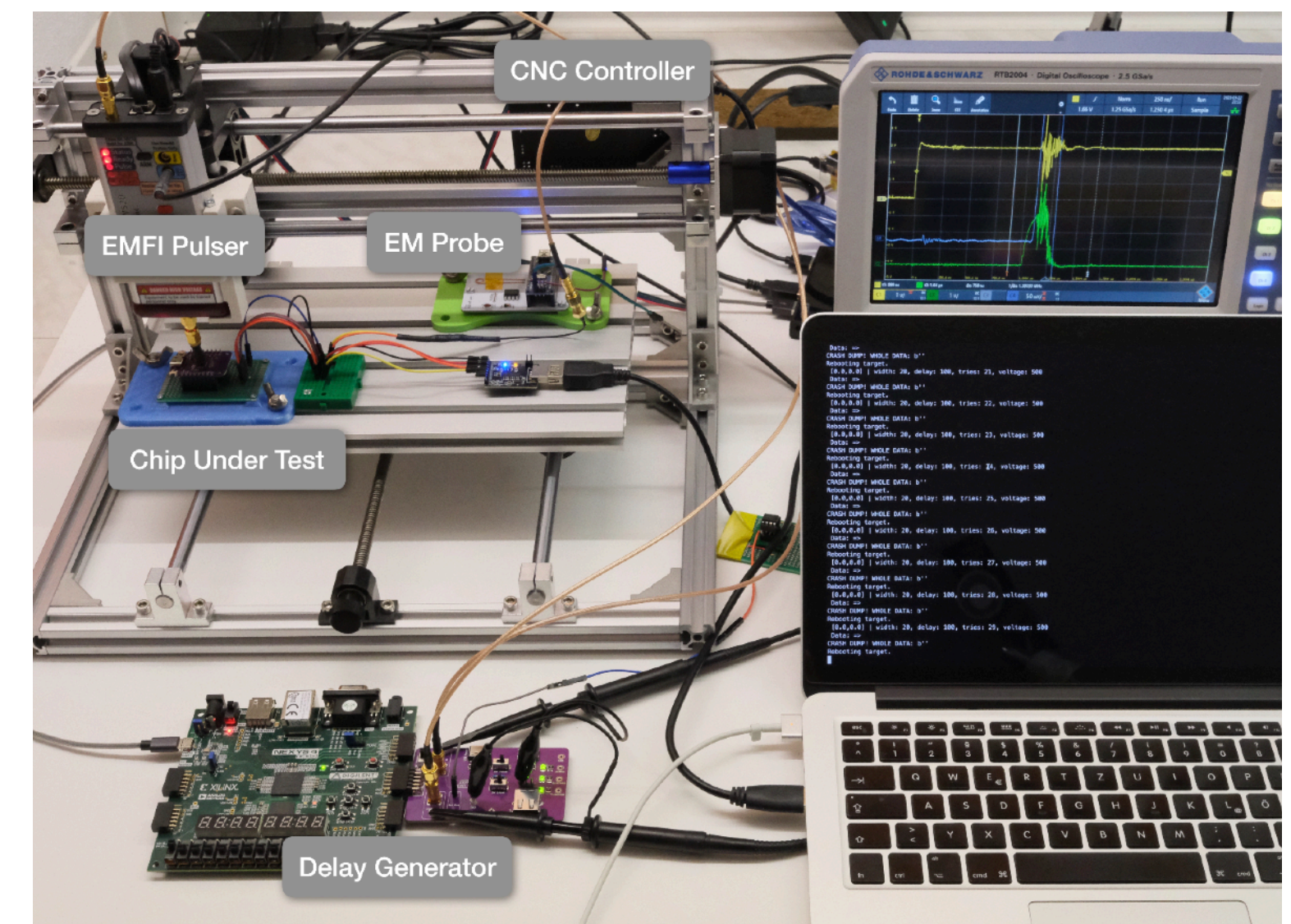


# Affordable\* EMFI Attacks Against Modern IoT Chips

\* and open-source

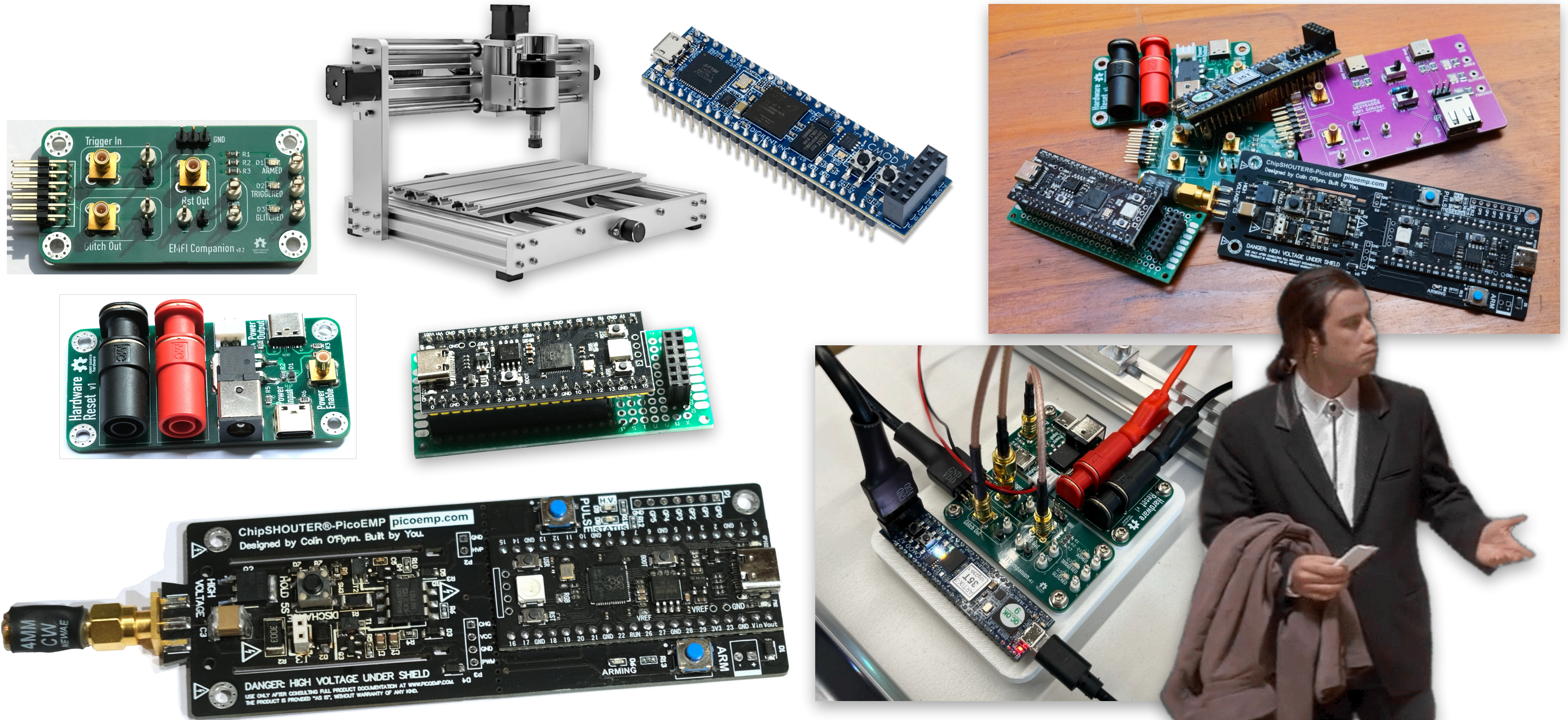
Daive Toldo  
Secure Mobile Networking Lab - SEEMOO  
Technical University of Darmstadt, Germany

**SEEMOO**  
SECURE MOBILE NETWORKING





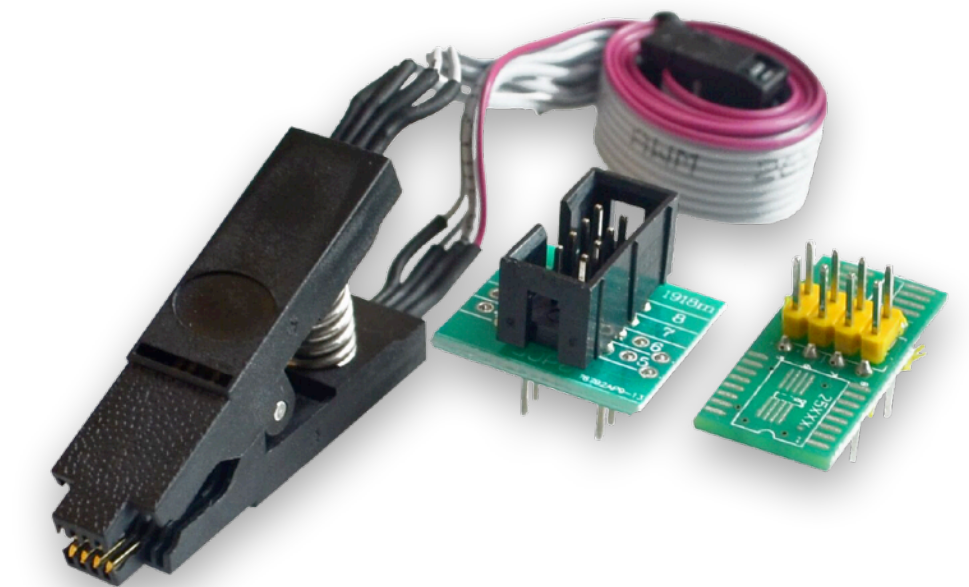
# What are we actually doing here?





# 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](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



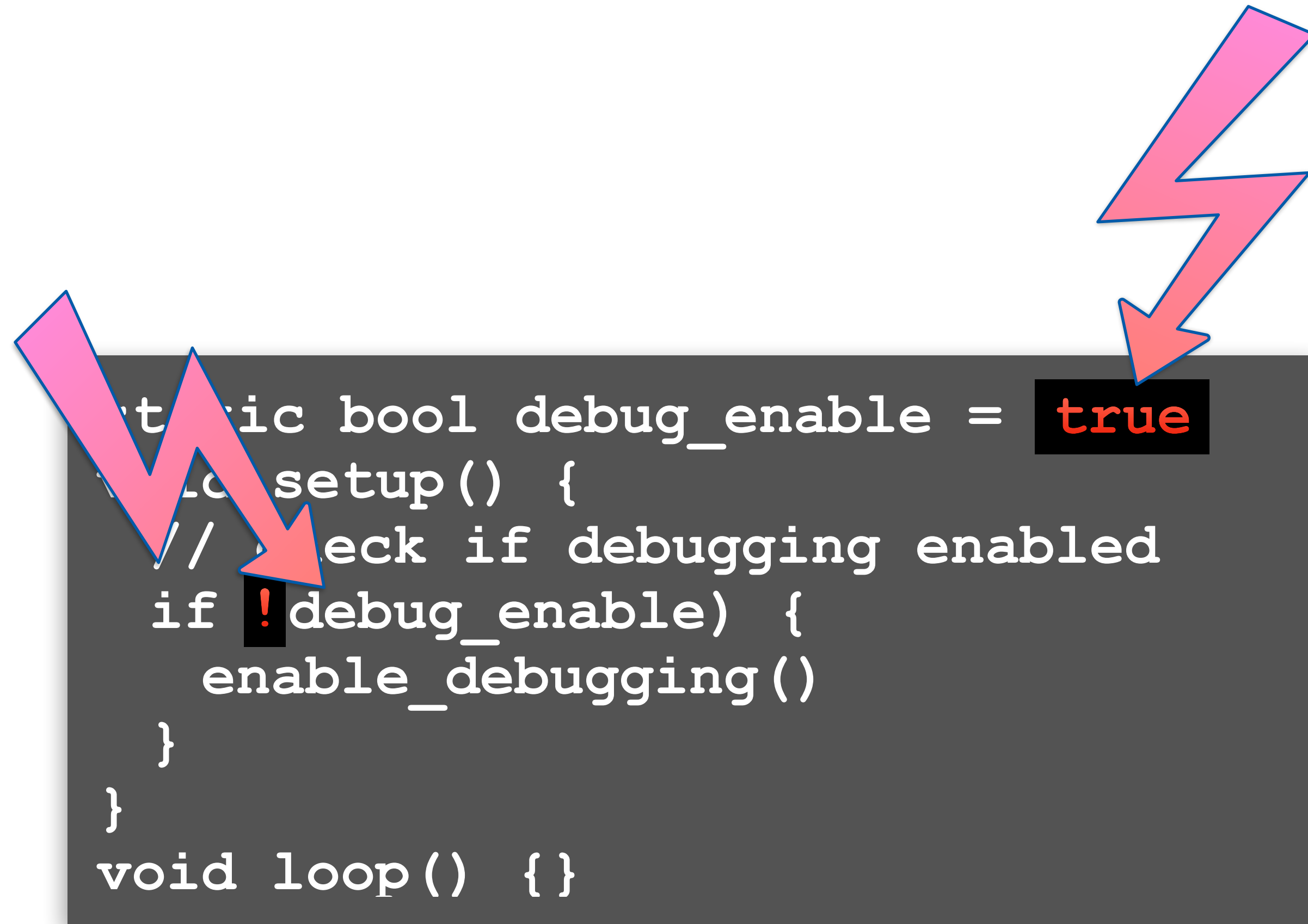
```
static bool debug_enable = true
void setup() {
  // check if debugging enabled
  if (debug_enable) {
    enable_debugging()
  }
}
void loop() {}
```



# 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() {}
```



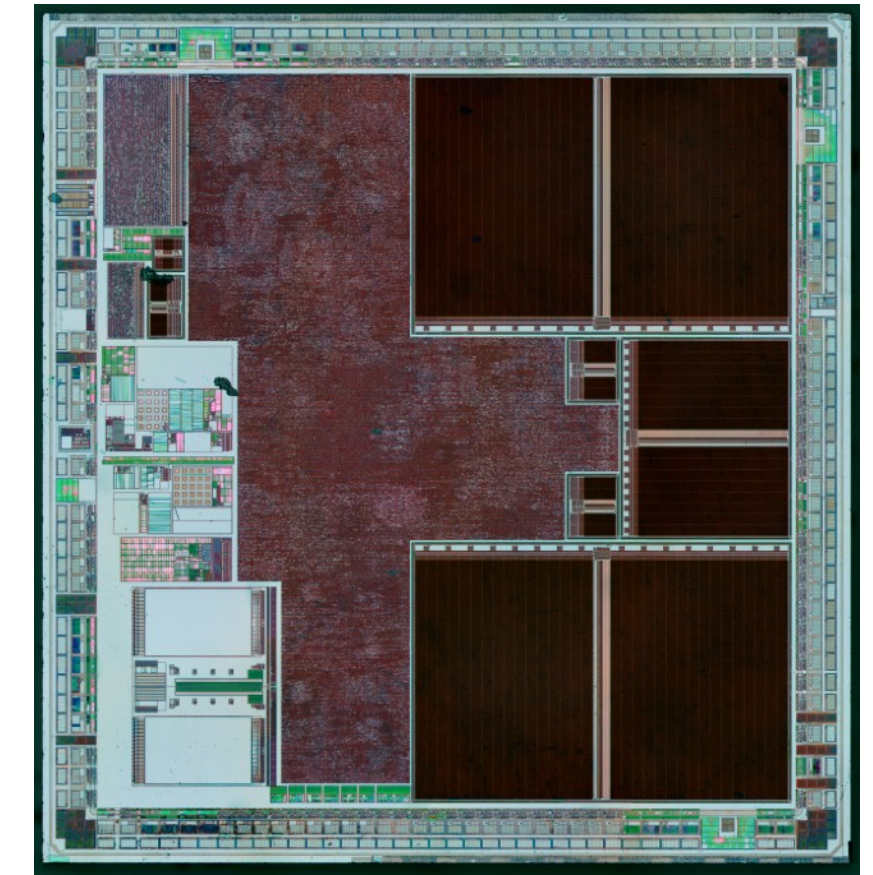


# EMFI =

- 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

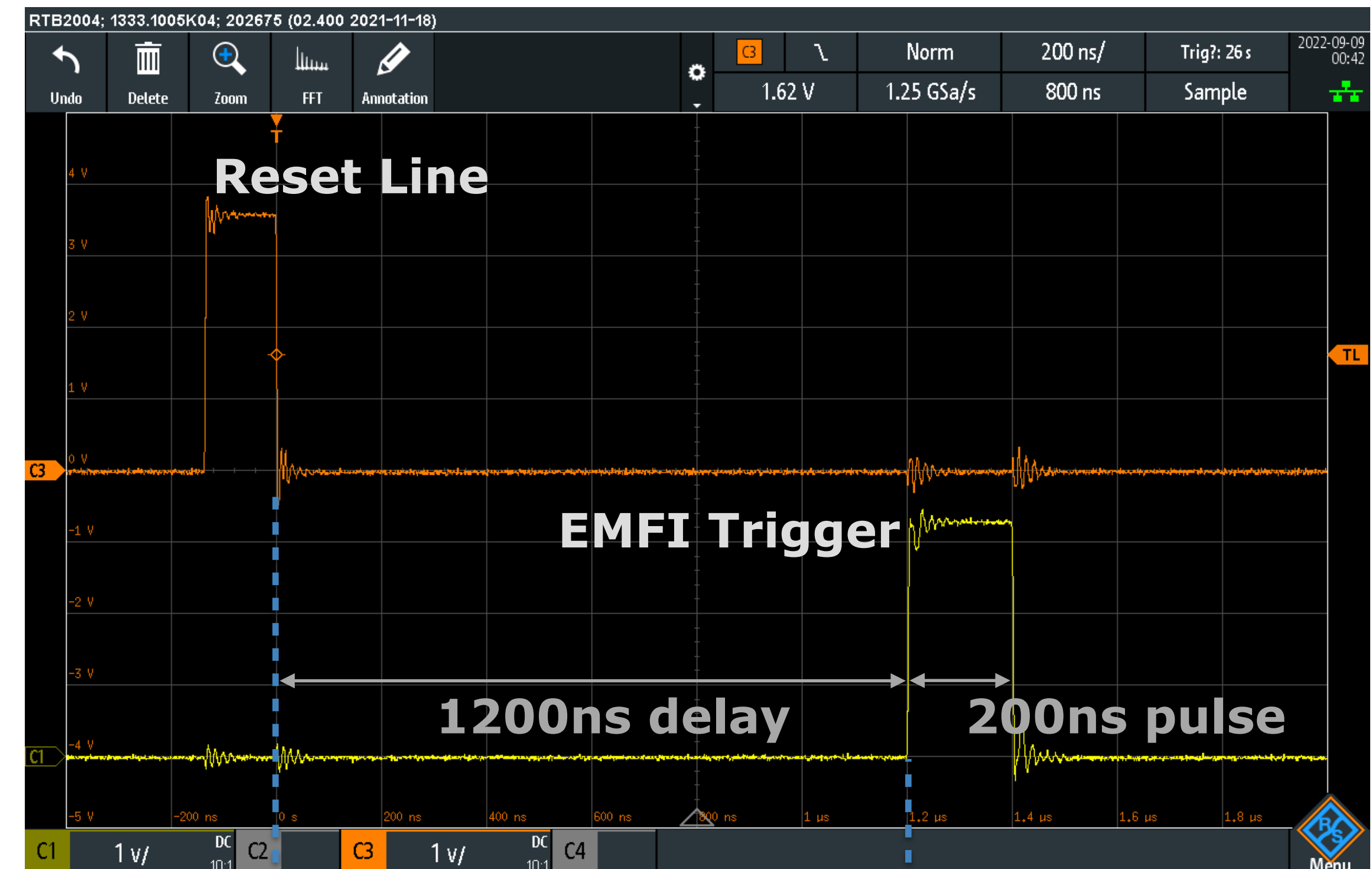


# EMFI Setup Requirements



<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

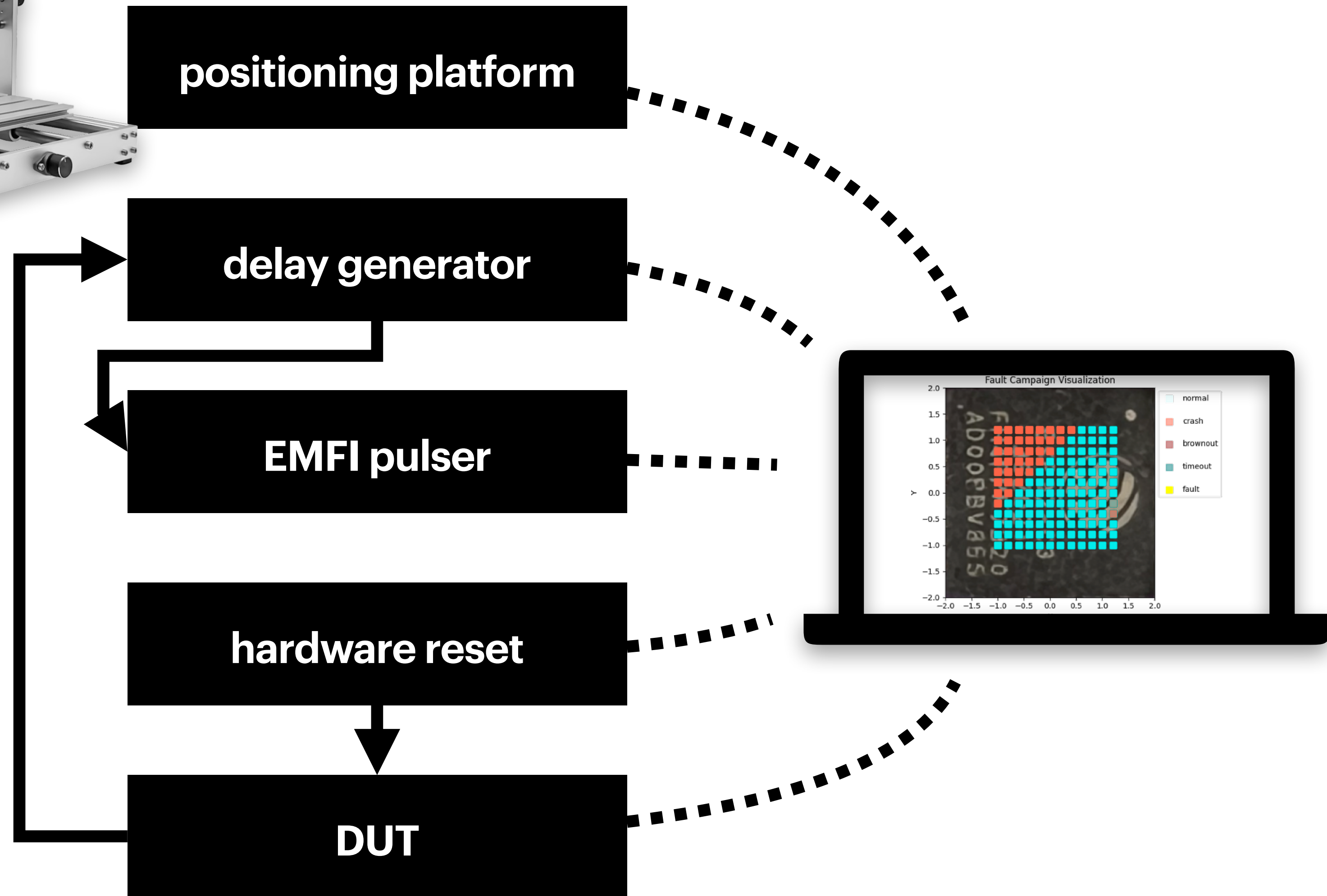
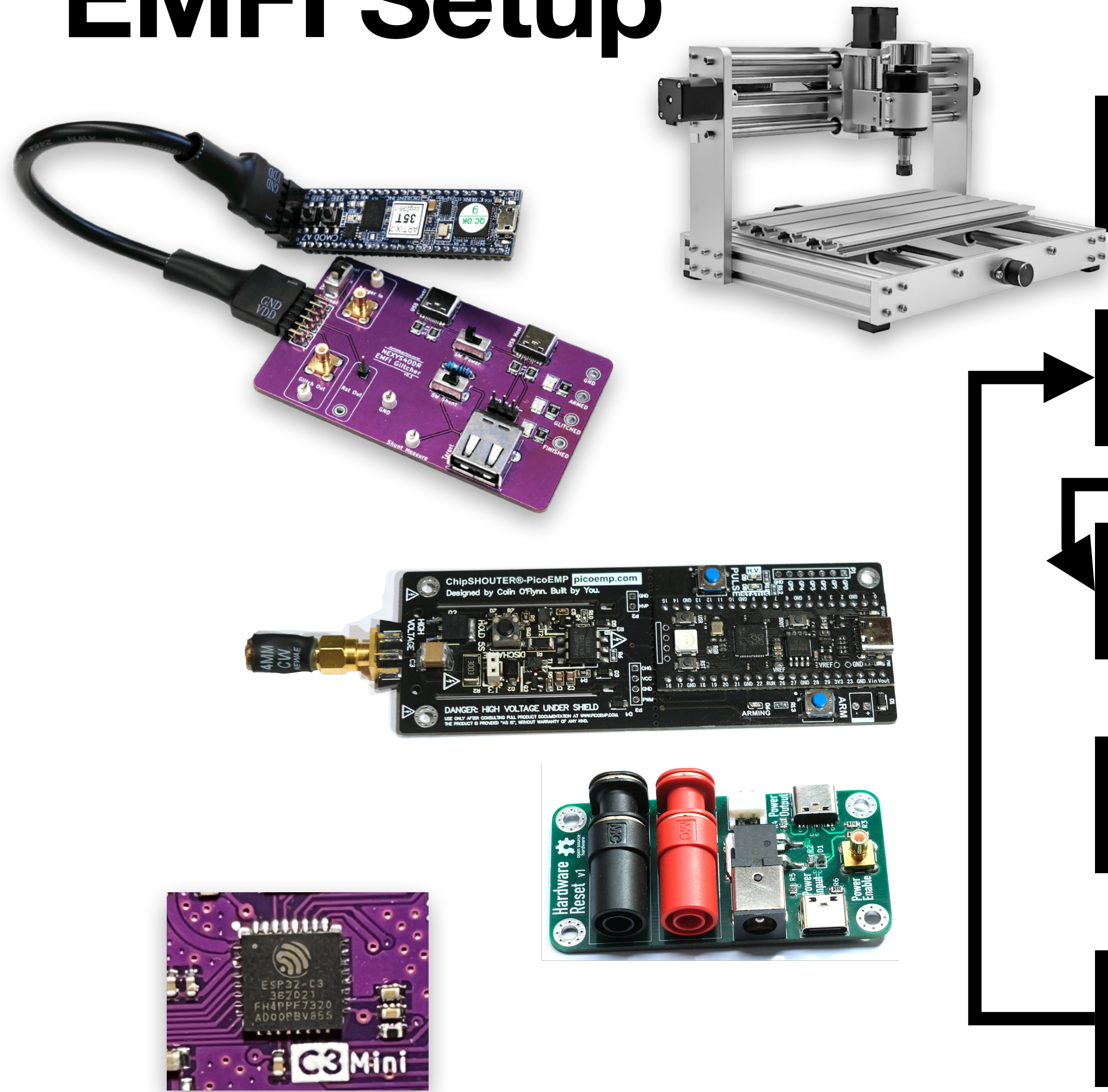


Boot starts

Fault injected



# EMFI Setup

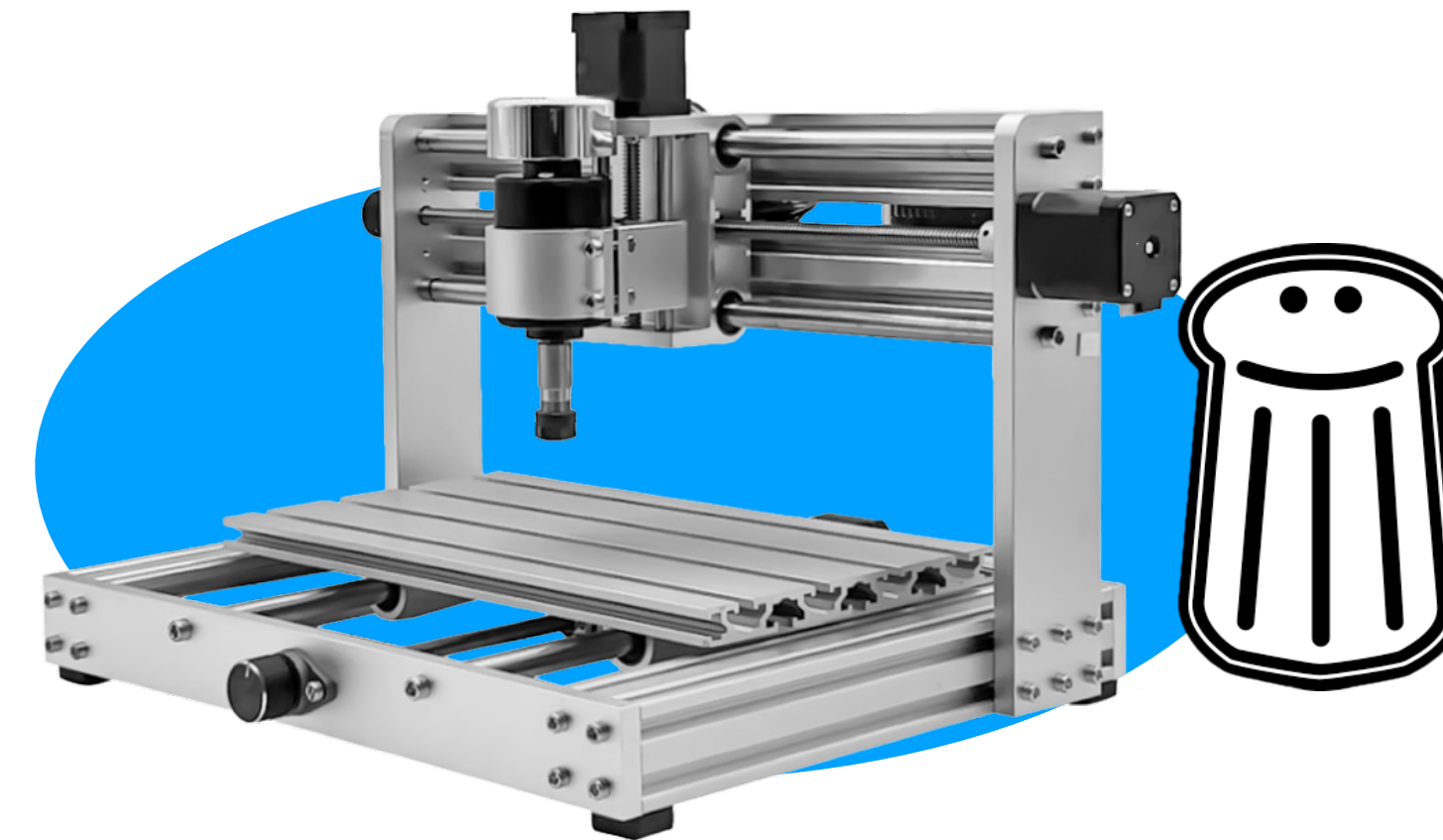




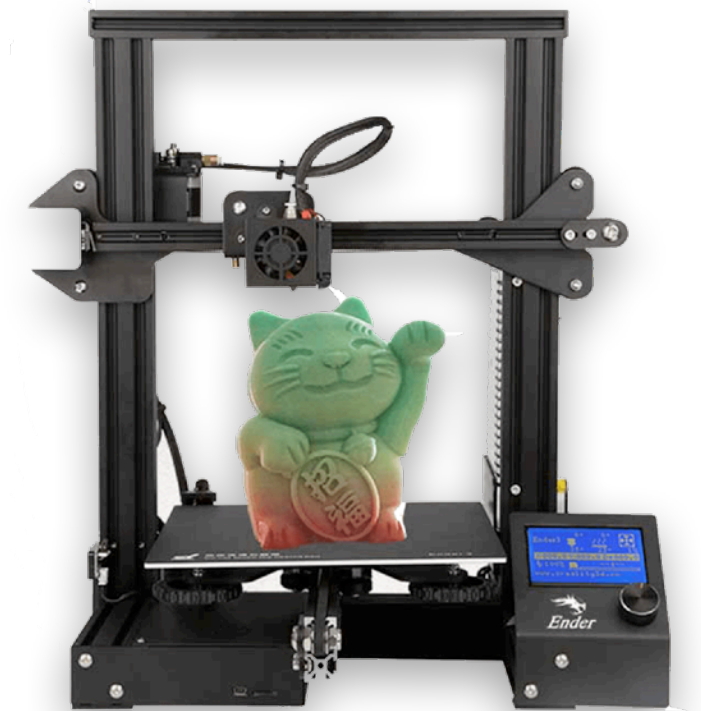
# EMFI Setup

## Positioning Platform

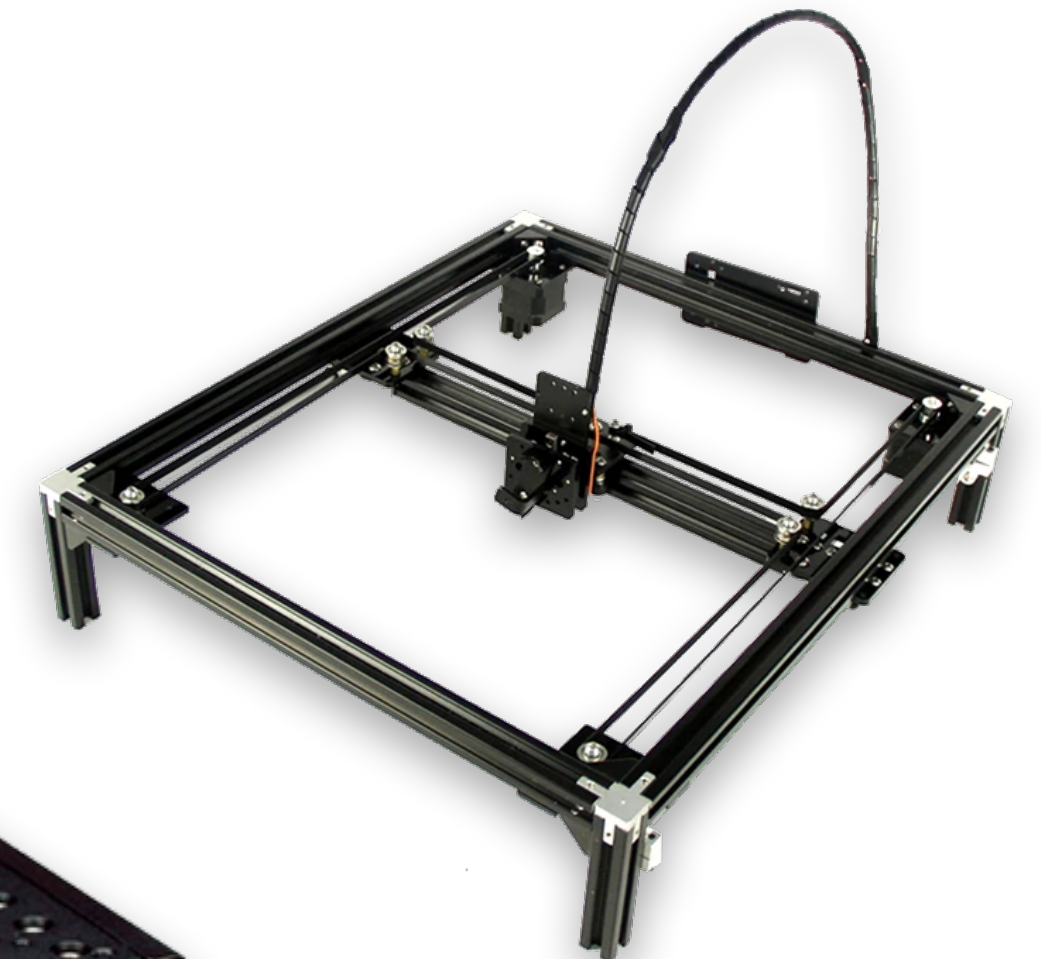
- CNCs or 3D printers can be used interchangeably due to GCODE
- Both are available for very low cost, come with motor controllers and everything needed
- Lead screws have approx. 10x more backlash → if budget allows, use belts
- Motorized XY stages offer small benefit for the price and IoT target



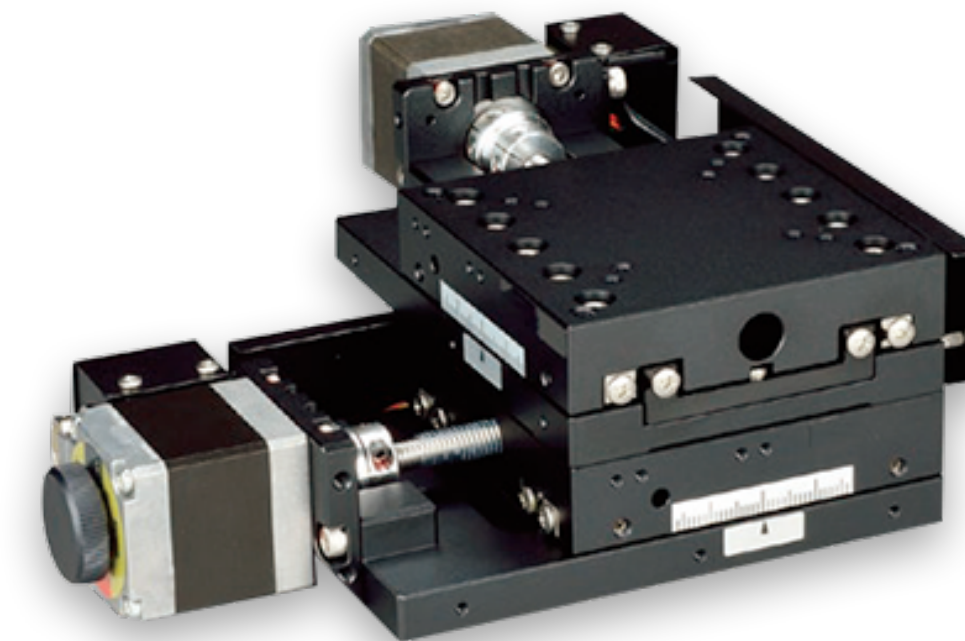
<https://aliexpress.com>



<https://aliexpress.com>



<https://aliexpress.com>



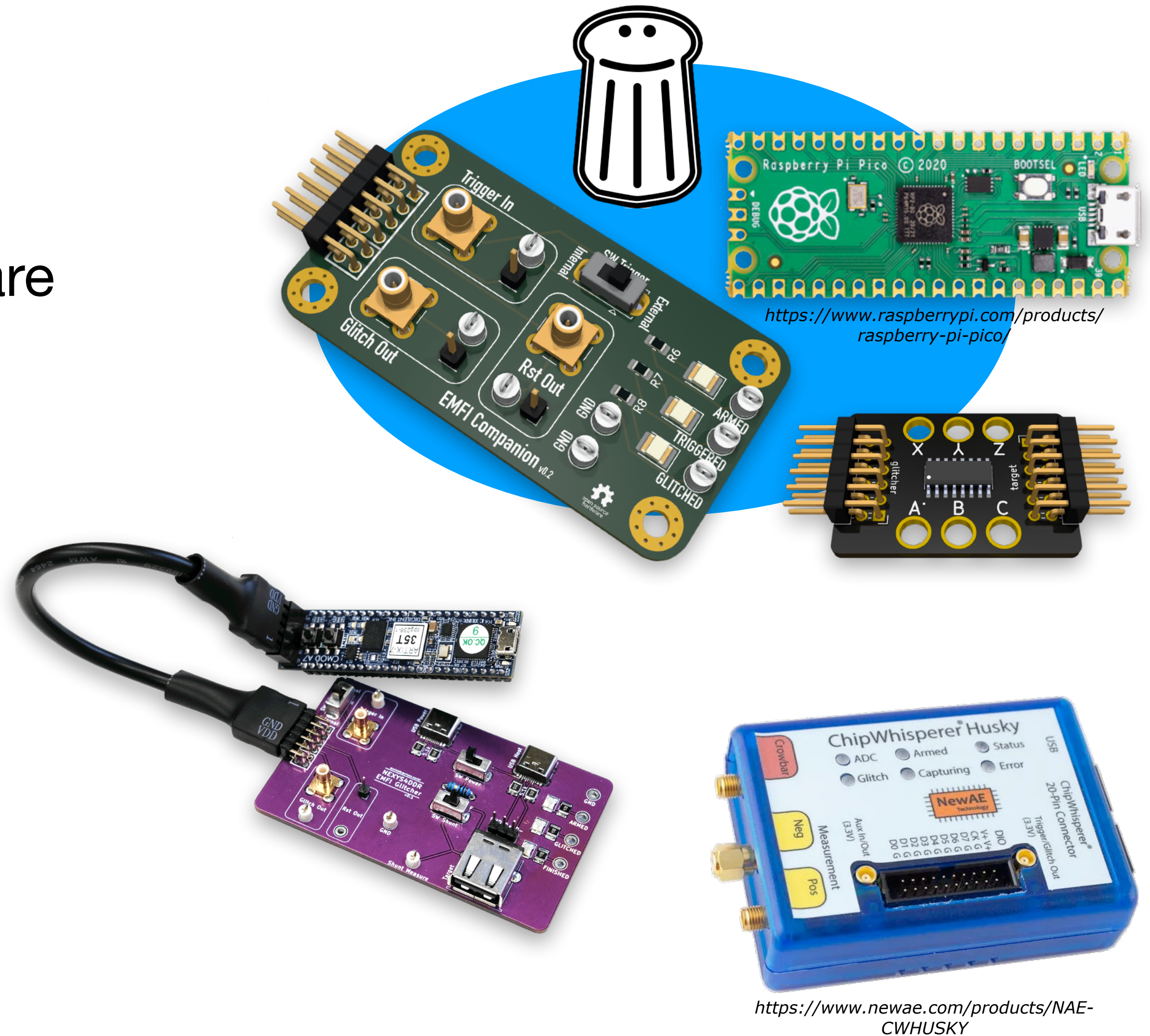
<https://www.thk.com>



# EMFI Setup

## Delay Generator

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

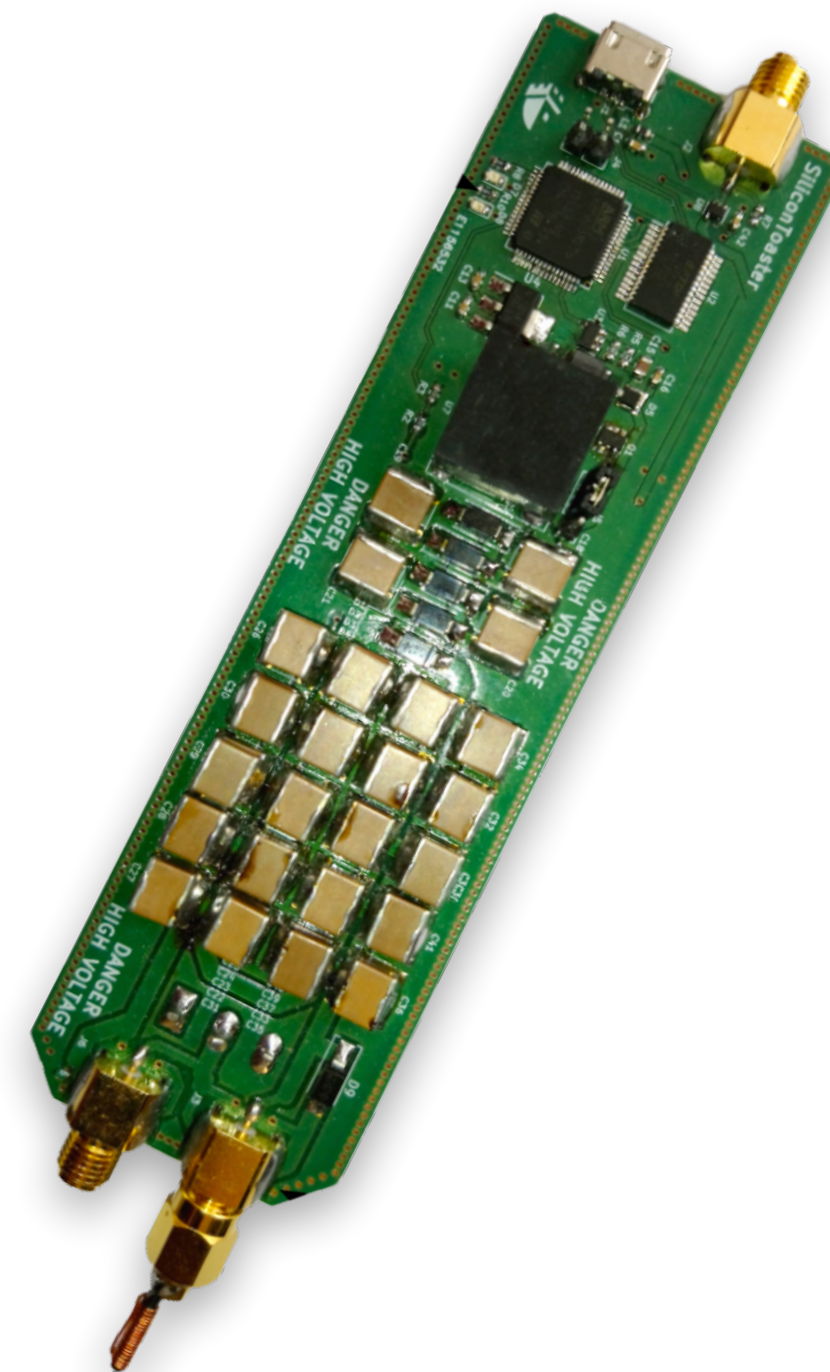




# EMFI Setup

## EMFI Pulser

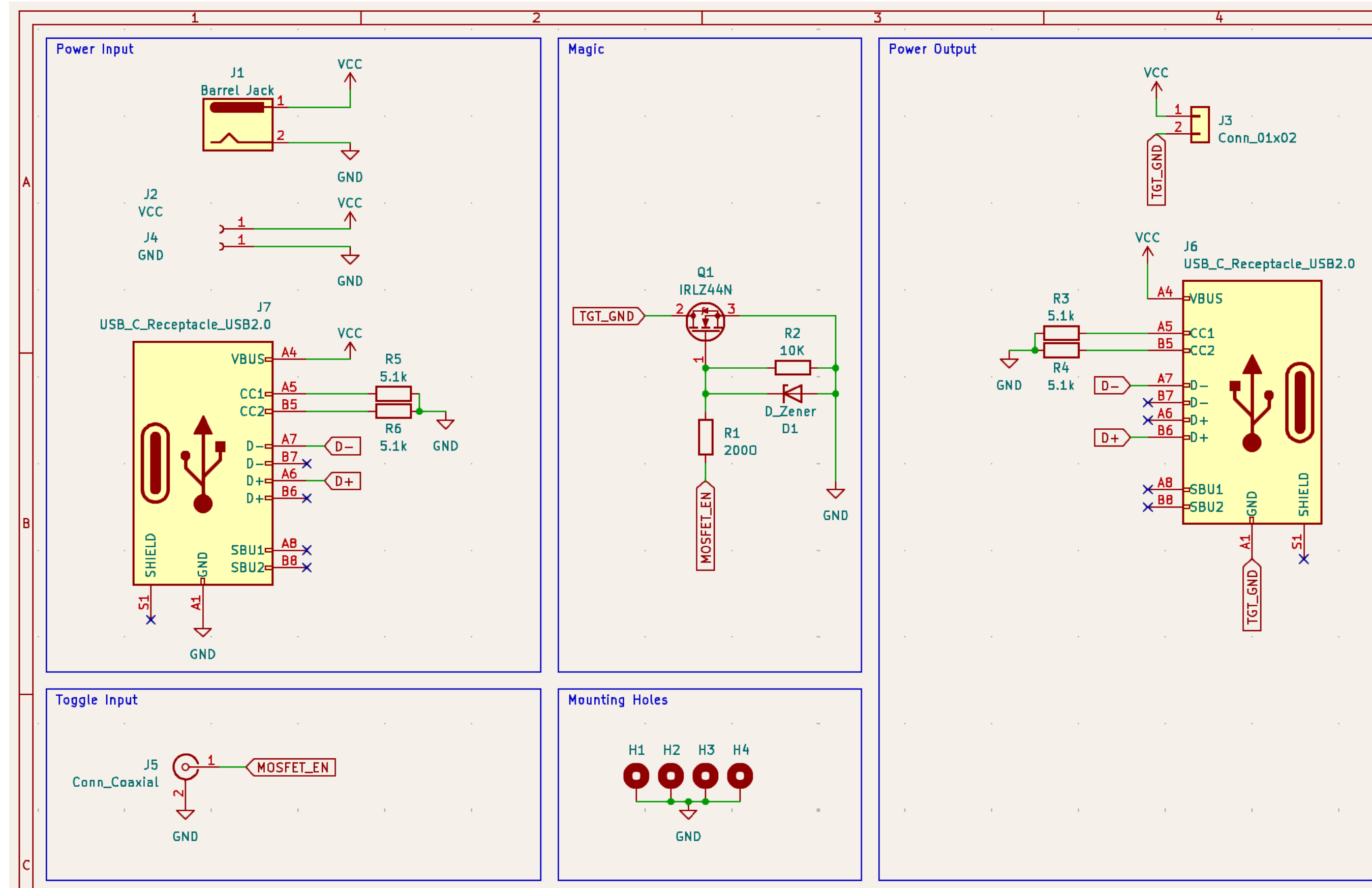
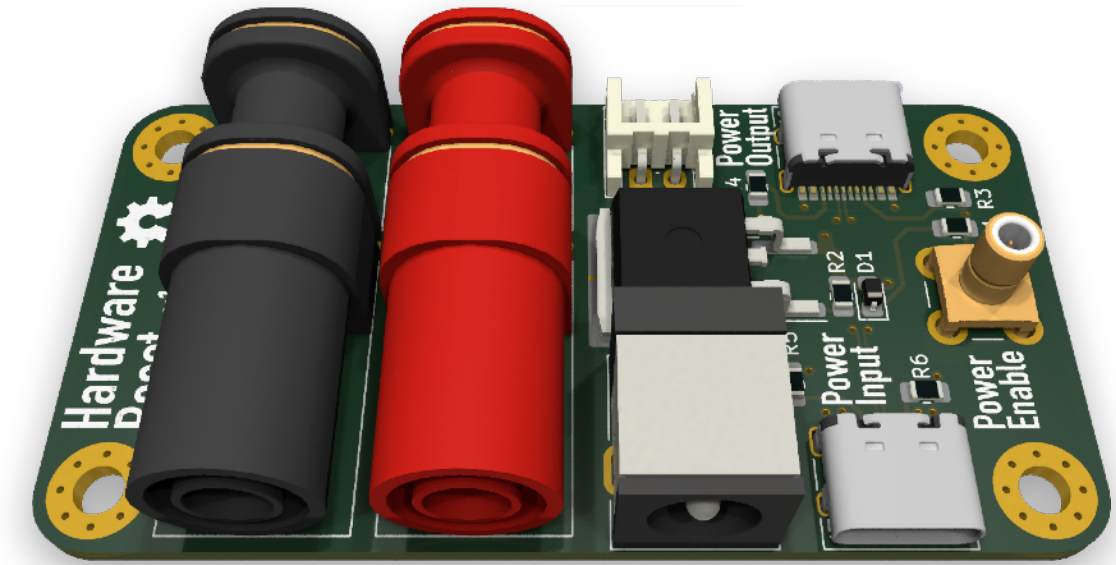
- ChipSHOUTER by @colinoflynn
- PicoEMP by @colinoflynn, @stacksmashing et al.
- SiliconToaster by Ledger





# 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

```
python3 main.py
ChipSHOUTER

type <help -v> for usage information!
$ help -v

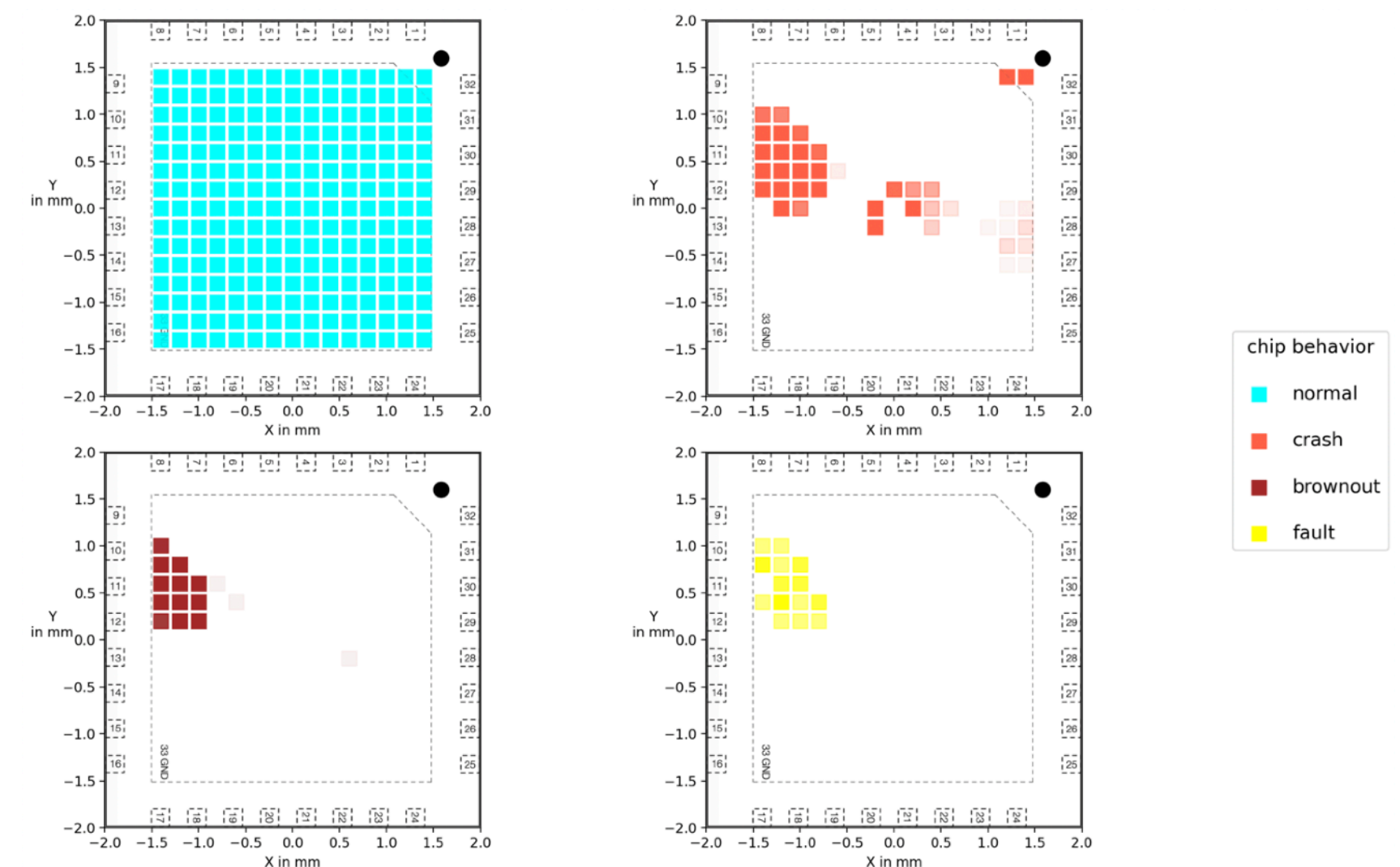
Documented commands (use 'help -v' for verbose/'help <topic>' for details):
=====
chipsize      Set chip size for automatically scanning it
connect       Connect to selected serial device
cs_reset      Reset ChipSHOUTER
emfi          Perform a single EMFI injection
get_xy        Get absolute position coordinates relative to origin (float)
go_xyz        Move in X, Y and Z direction to specific location in mm from origin (float)
help          List available commands or provide detailed help for a specific command
history       View, run, edit, save, or clear previously entered commands
list          List available serial devices and estimated device types
loglevel      Change the verbosity of log messages
measure_emfield Measure EM field at current location
move          Move in X, Y and Z direction by specified distances in mm
mute          Mute ChipSHOUTER's internal buzzer
quit          Exit this application
scan          Scan across either chip, while injecting faults or across EM probe while
              collecting voltage measurements
set_origin    Set current position as origin / (0,0)
x             Move in X direction by specified distance in mm
y             Move in Y direction by specified distance in mm
z             Move in Z direction by specified distance in mm

$ list
[*] Available serial devices:
[*] [0] /dev/cu.usbserial-NA5I5I54: ChipSHOUTER Serial - ChipSHOUTER Serial [USB VID:PID=0403:6015 SER=NA5I5I54 LOCATION=20-1.3]
[*] [1] /dev/cu.usbserial-14110: USB Serial [USB VID:PID=1A86:7523 LOCATION=20-1.1] (generic usb serial: table or target)
[*] [2] /dev/cu.usbserial-210292A3FFBC0: Diligent USB Device - Diligent USB Device [USB VID:PID=0403:6010 SER=210292A3FFBC LOCATION=20-1.4]
[*] [3] /dev/cu.usbserial-210292A3FFBC1: Diligent USB Device - Diligent USB Device [USB VID:PID=0403:6010 SER=210292A3FFBC LOCATION=20-1.4]
[*] [4] /dev/cu.usbserial-1420: USB Serial [USB VID:PID=1A86:7523 LOCATION=20-2] (generic usb serial: table or target)
[*] $
```

EMFIControl

```
ChipSHOUTER - EMFI Control-master
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 8, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 1, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 2, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 3, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 4, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 5, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 6, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 7, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 8, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 9, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 10, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 11, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 12, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 13, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 14, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 15, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 16, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 17, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 18, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 19, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 20, voltage: 400
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 1, voltage: 410
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 2, voltage: 410
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 3, voltage: 410
Data: => R x value: 100
I-0.19999999999999996.0.19999999999999996 I width: 40, delay: 800, tries: 4, voltage: 410
```

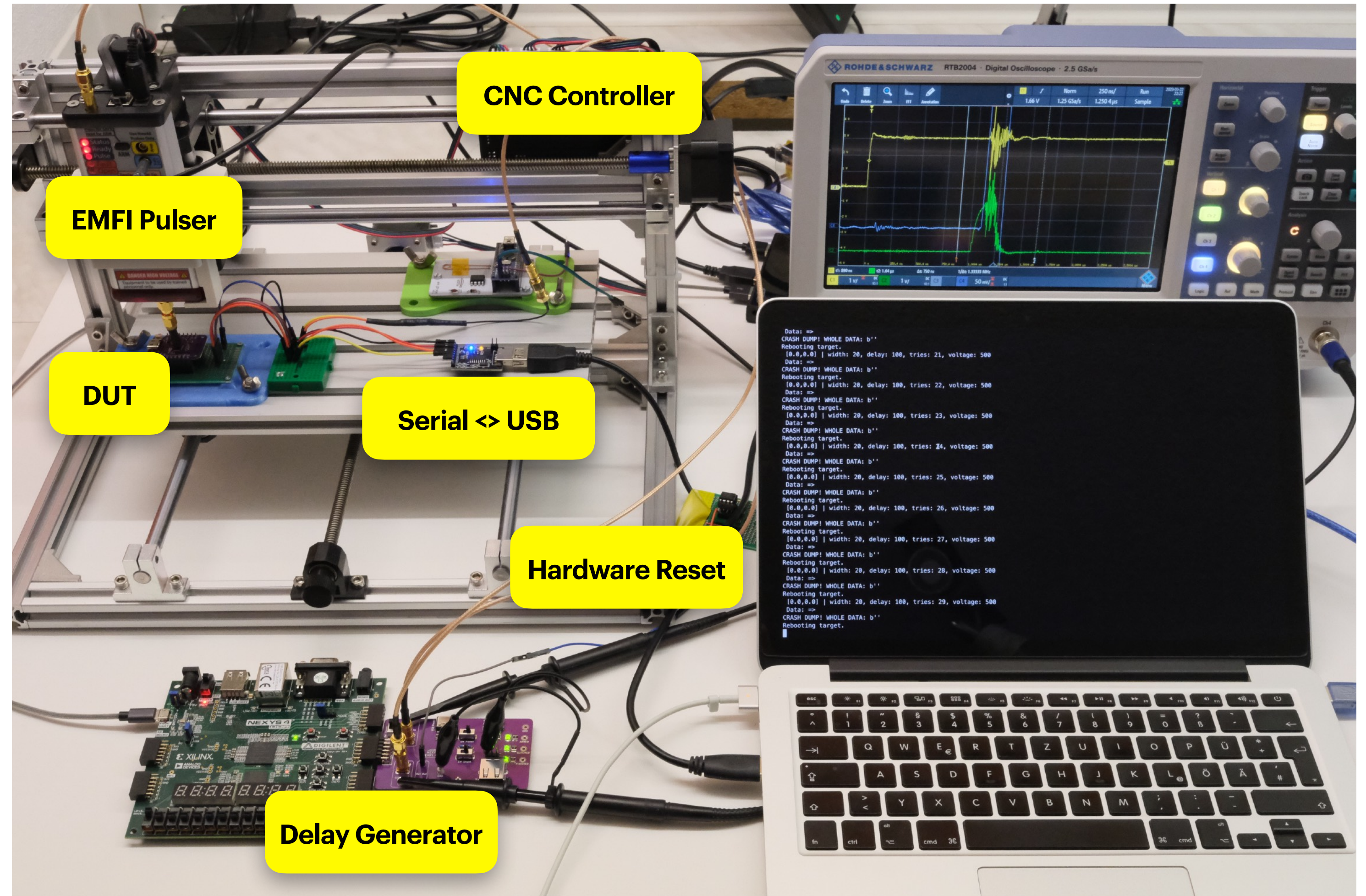
Fault campaign



Result visualization

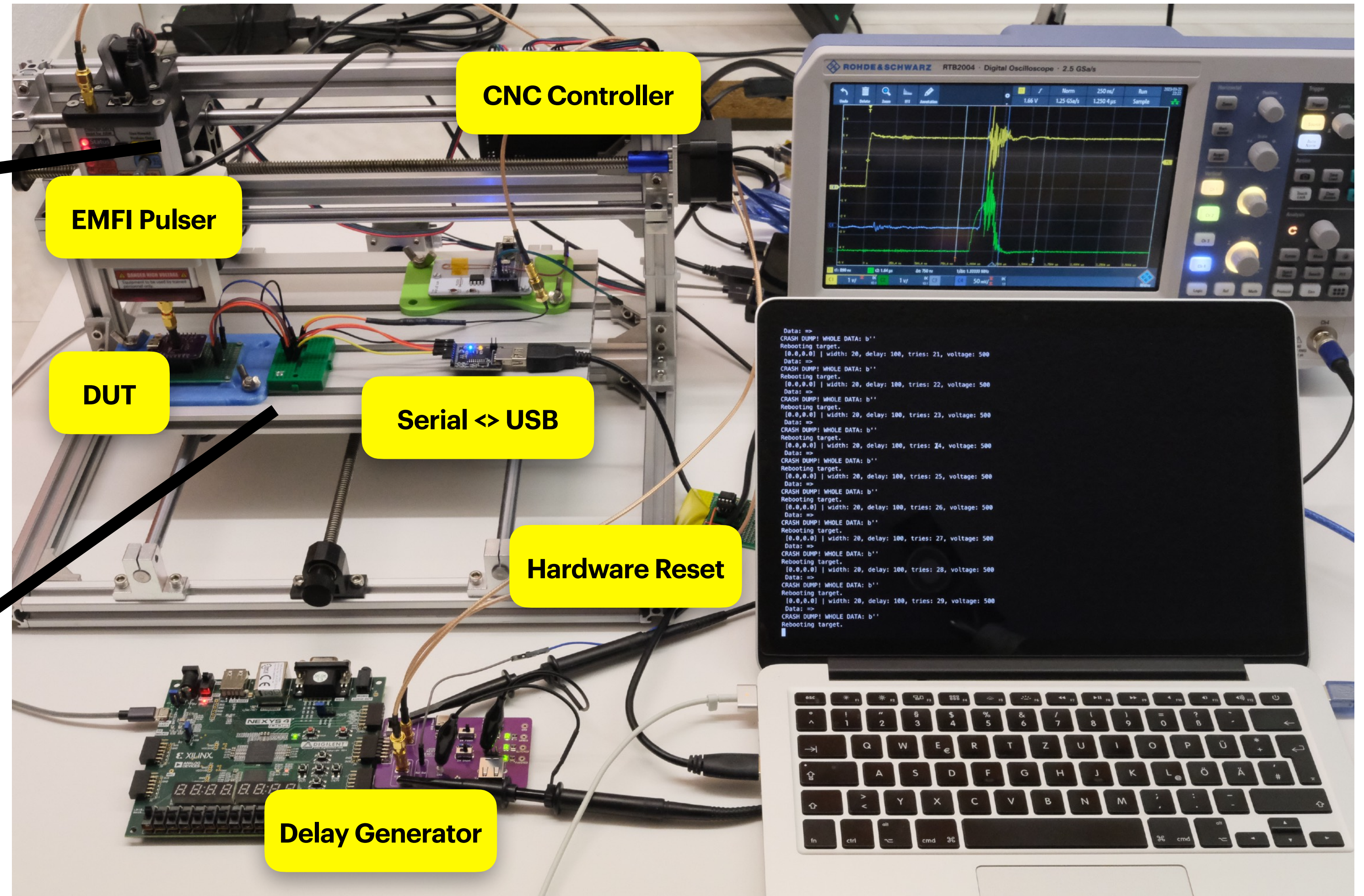
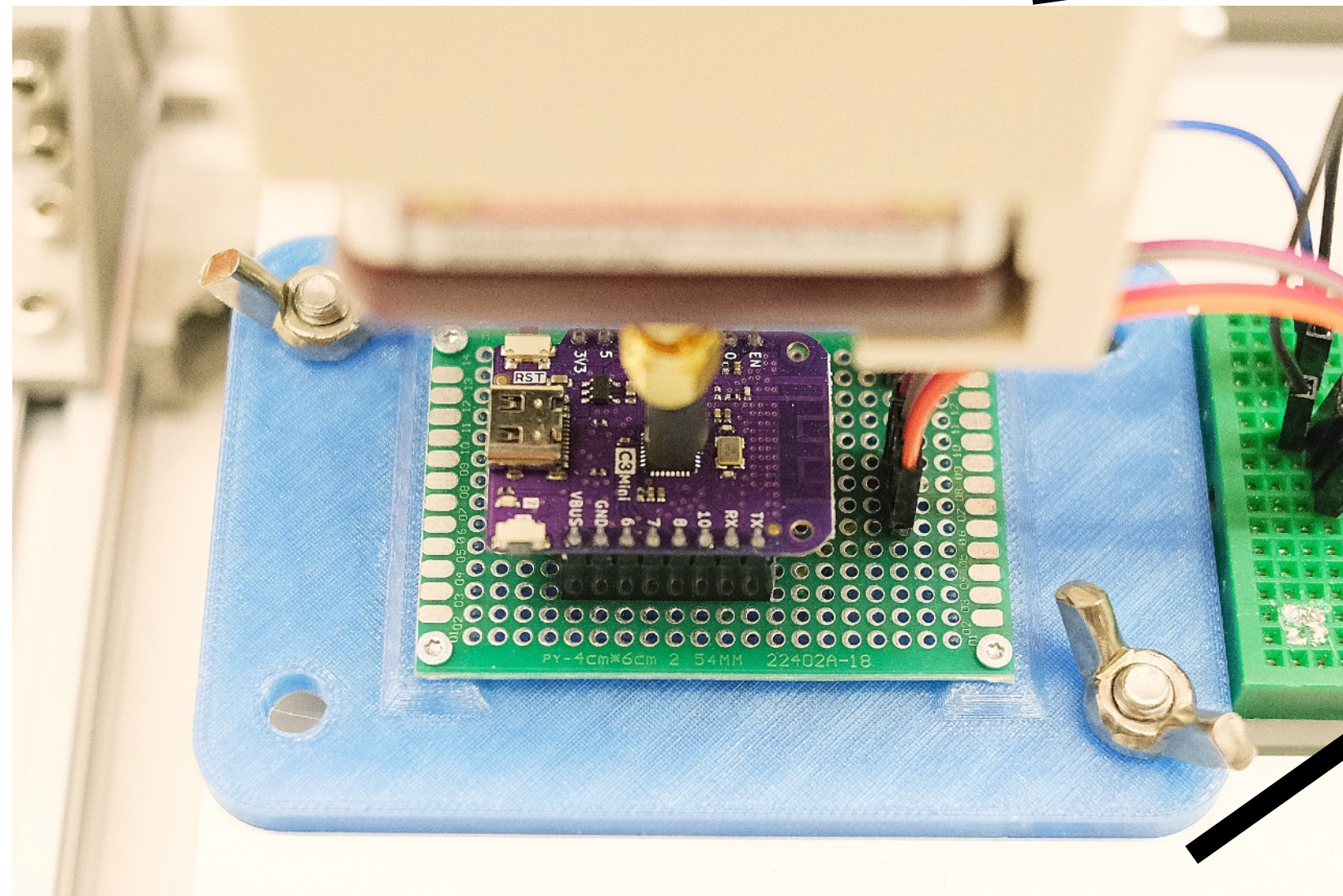


# EMFI Setup



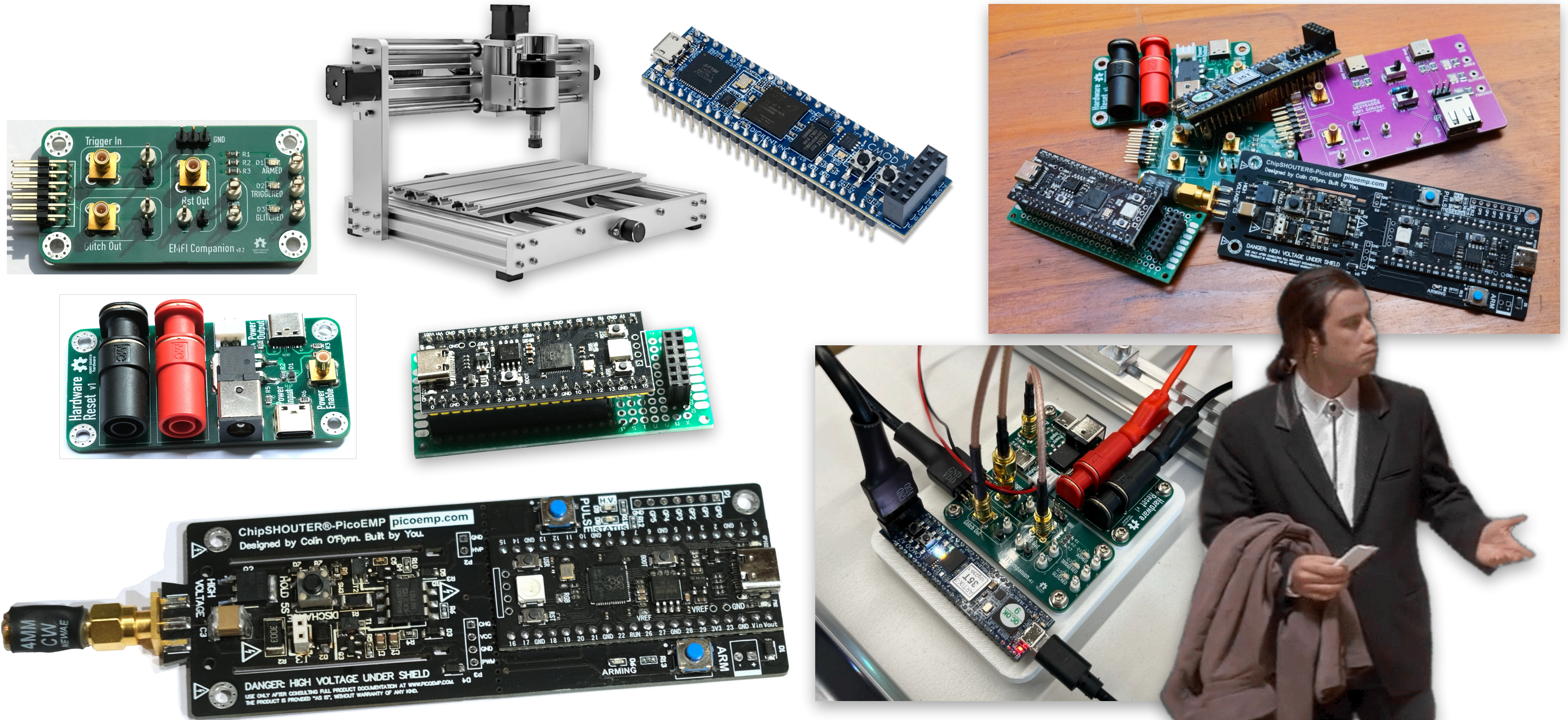


# EMFI Setup





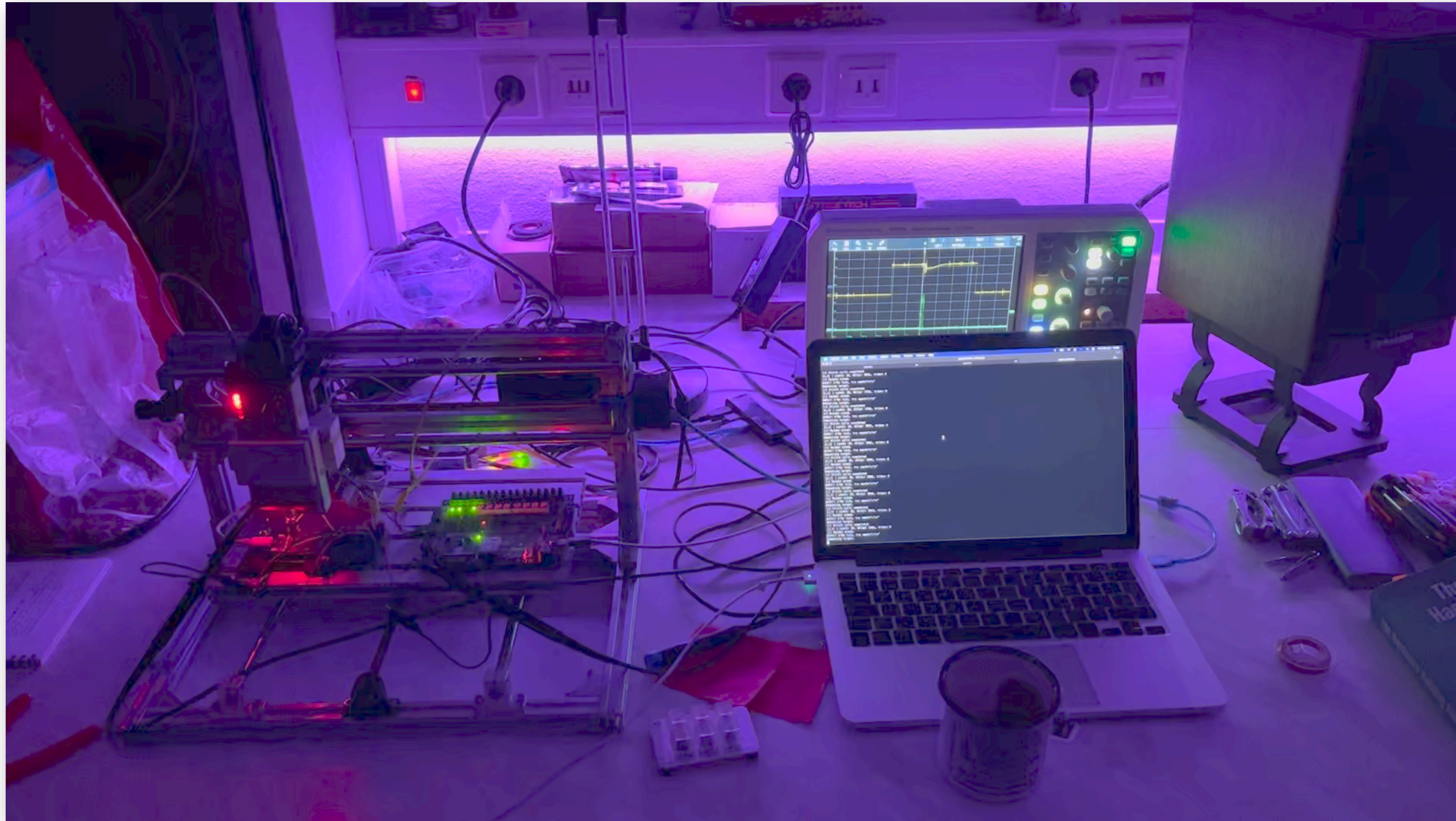
# What are we actually doing here?





# Results

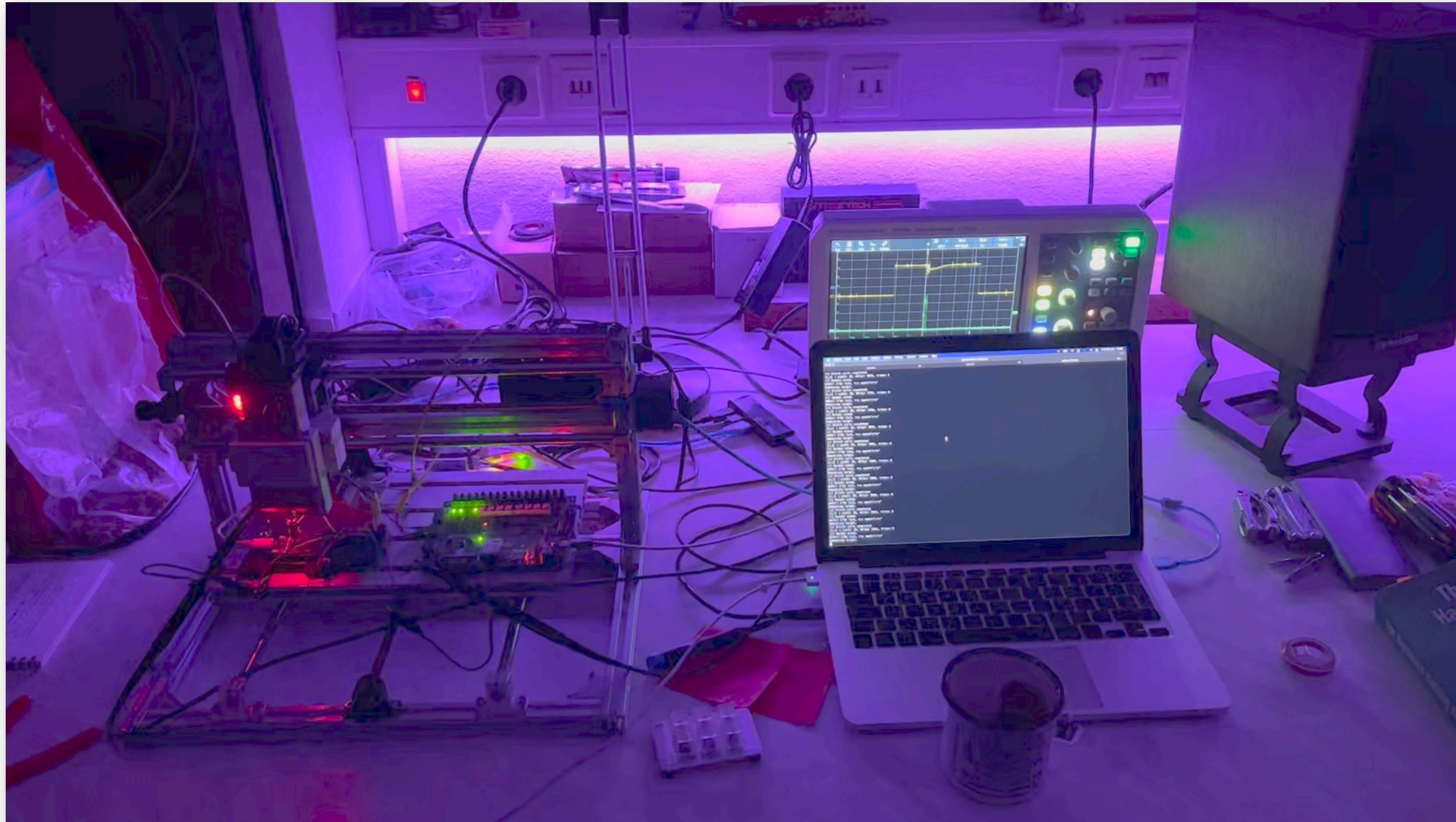
IoT chips with "physical security features"





# Results

IoT chips with "physical security features"





# Results

## IoT chips with "physical security features"

3.6	Bluetooth LE	38
3.6.1	Bluetooth LE Radio and PHY	38
3.6.2	Bluetooth LE Link Layer Controller	38
3.7	Power Management	
3.8	Timers	
3.8.1	General Purpose Timers	
3.8.2	System Timer	
3.8.3	Watchdog Timers	
3.9	Cryptographic Hardware Accelerators	
3.10	Physical Security Features	
3.11	Peripheral Pin Configurations	

### 3.10 Physical Security Features

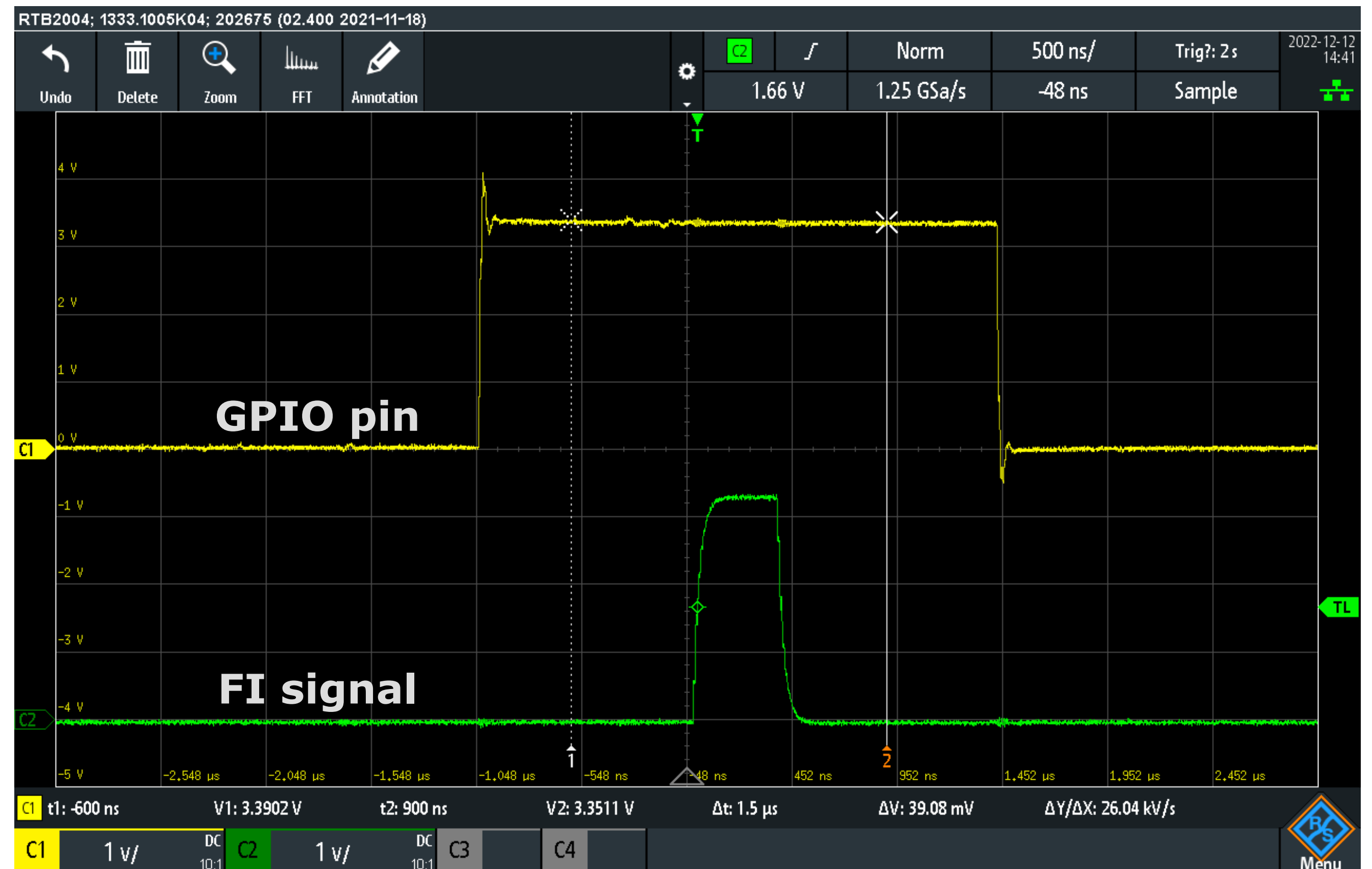
- Transparent off-package flash encryption (AES-XTS algorithm) with software inaccessible key prevents unauthorized readout of your application code or data.
- Secure boot feature uses a hardware root of trust to ensure only signed firmware (with RSA-PSS signature) can be booted.
- HMAC module can use 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

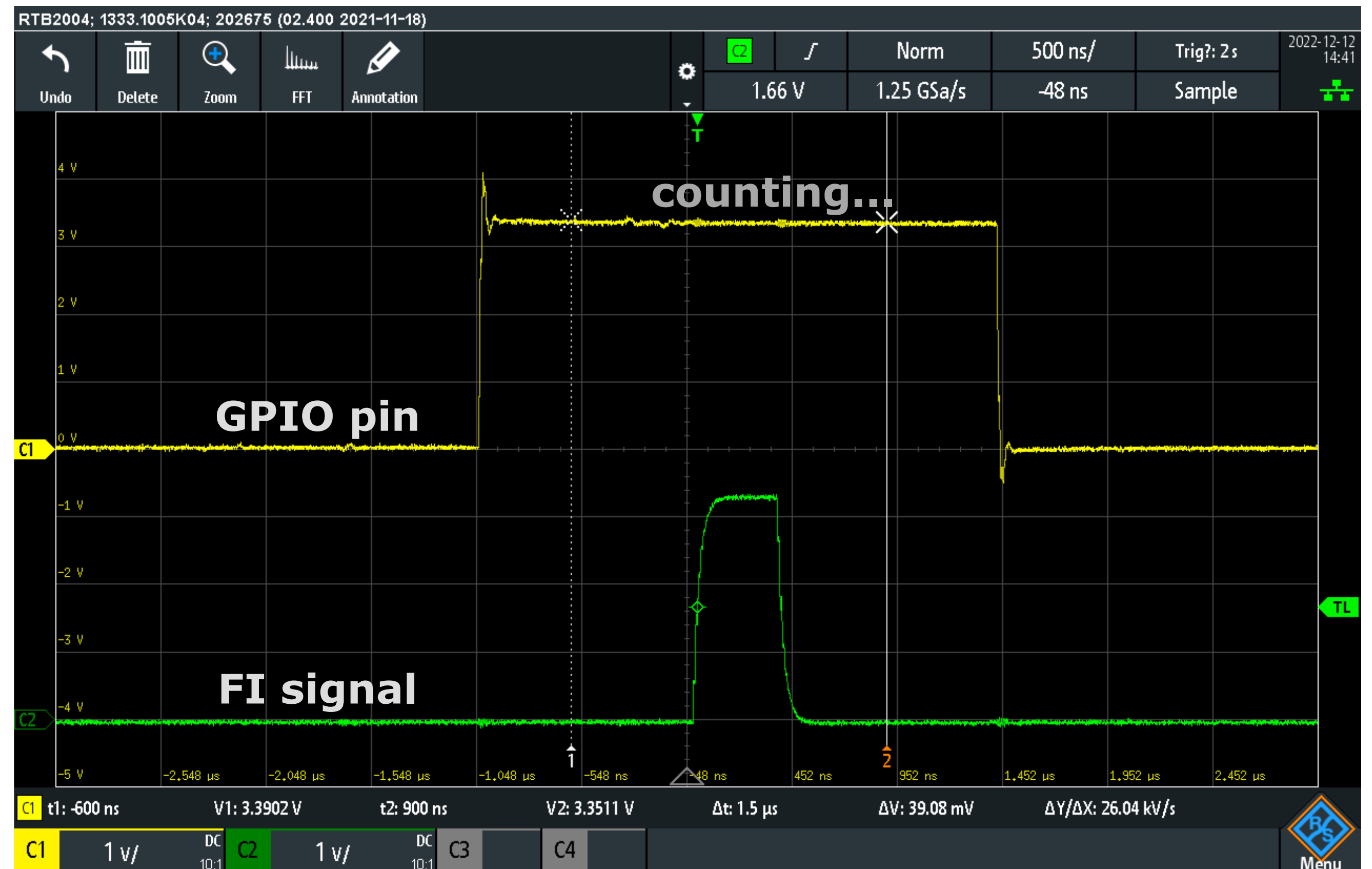




# Results

## 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

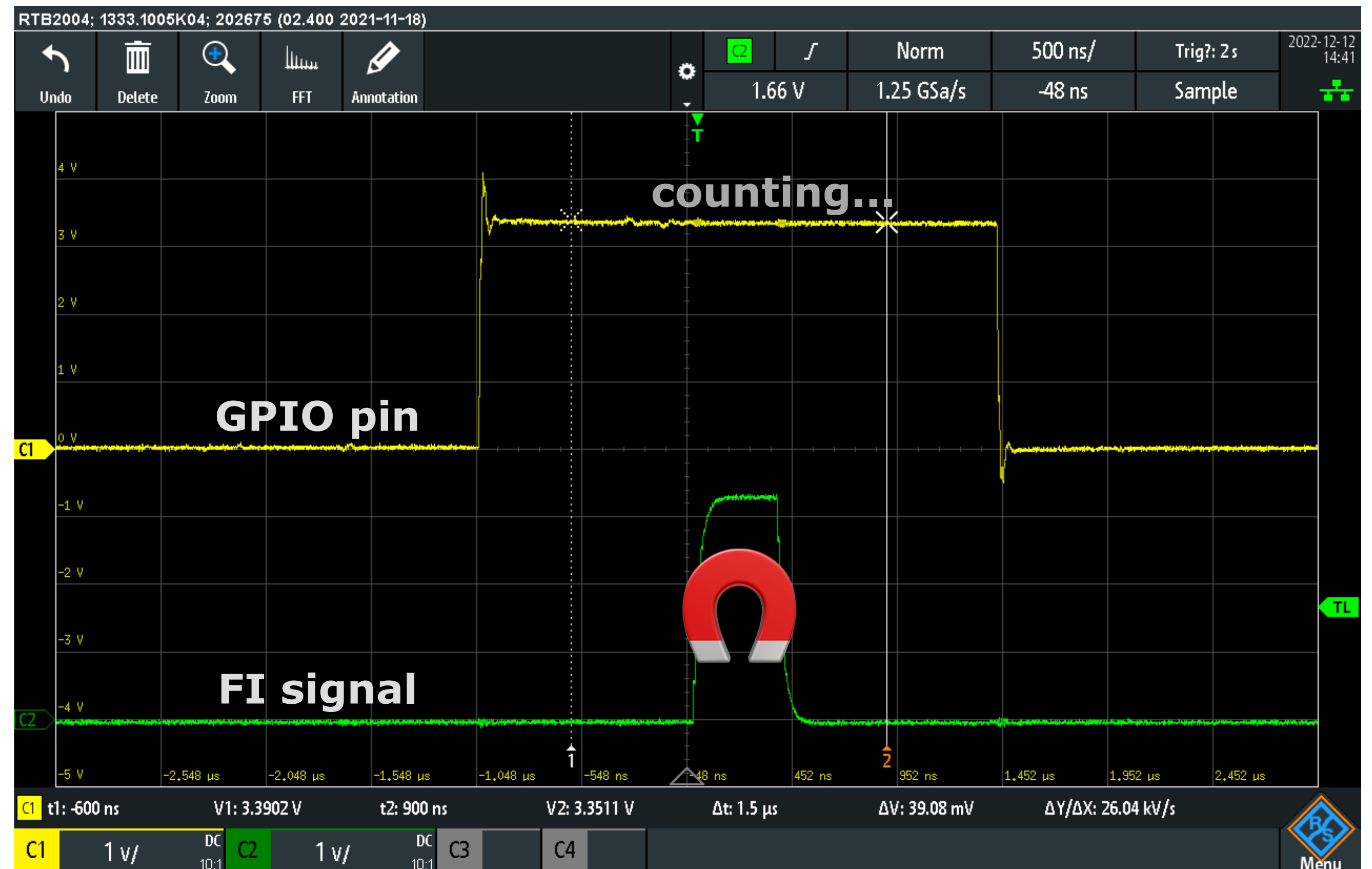




# Results

## 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

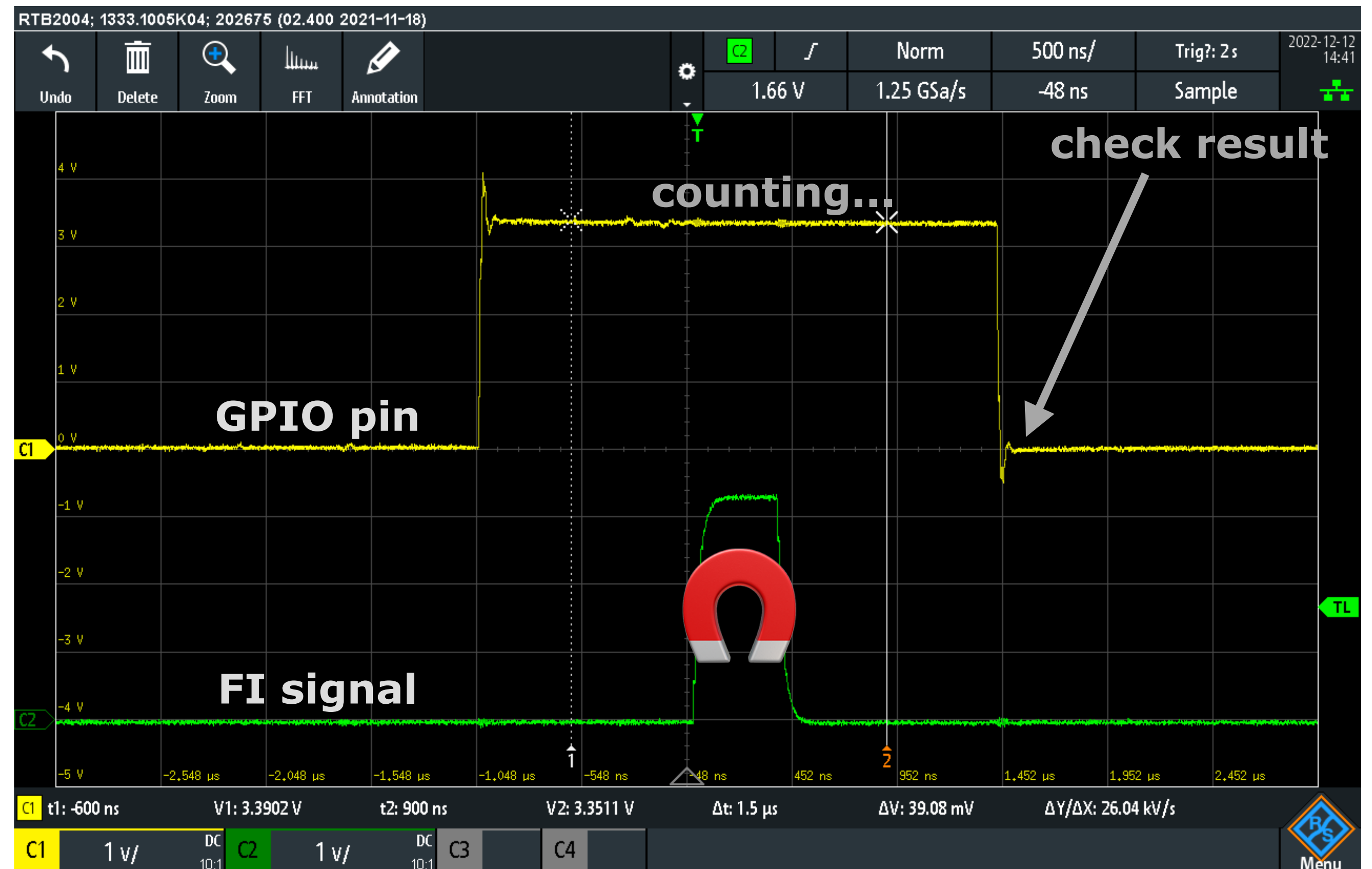




# Results

## 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



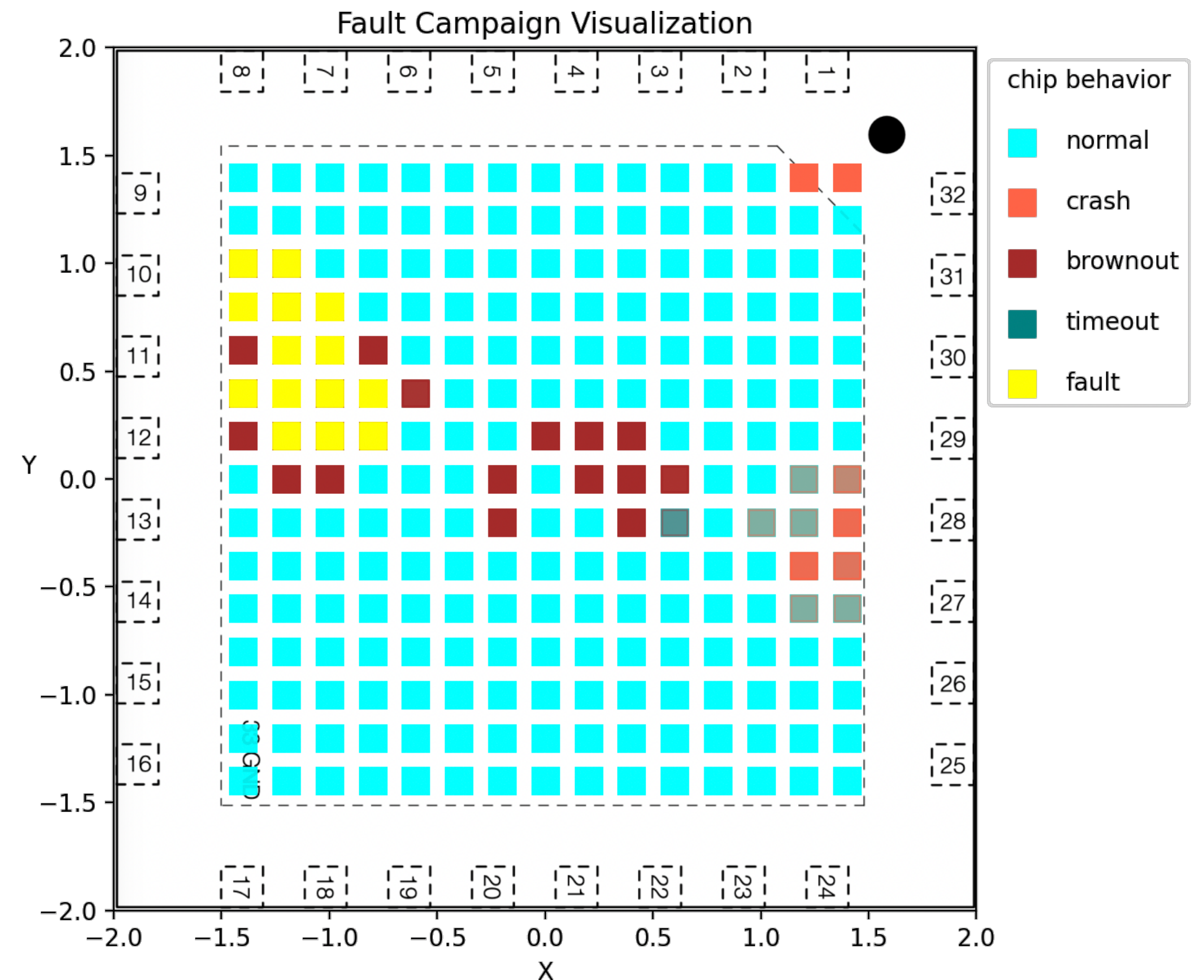


# Fault Campaign

## 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

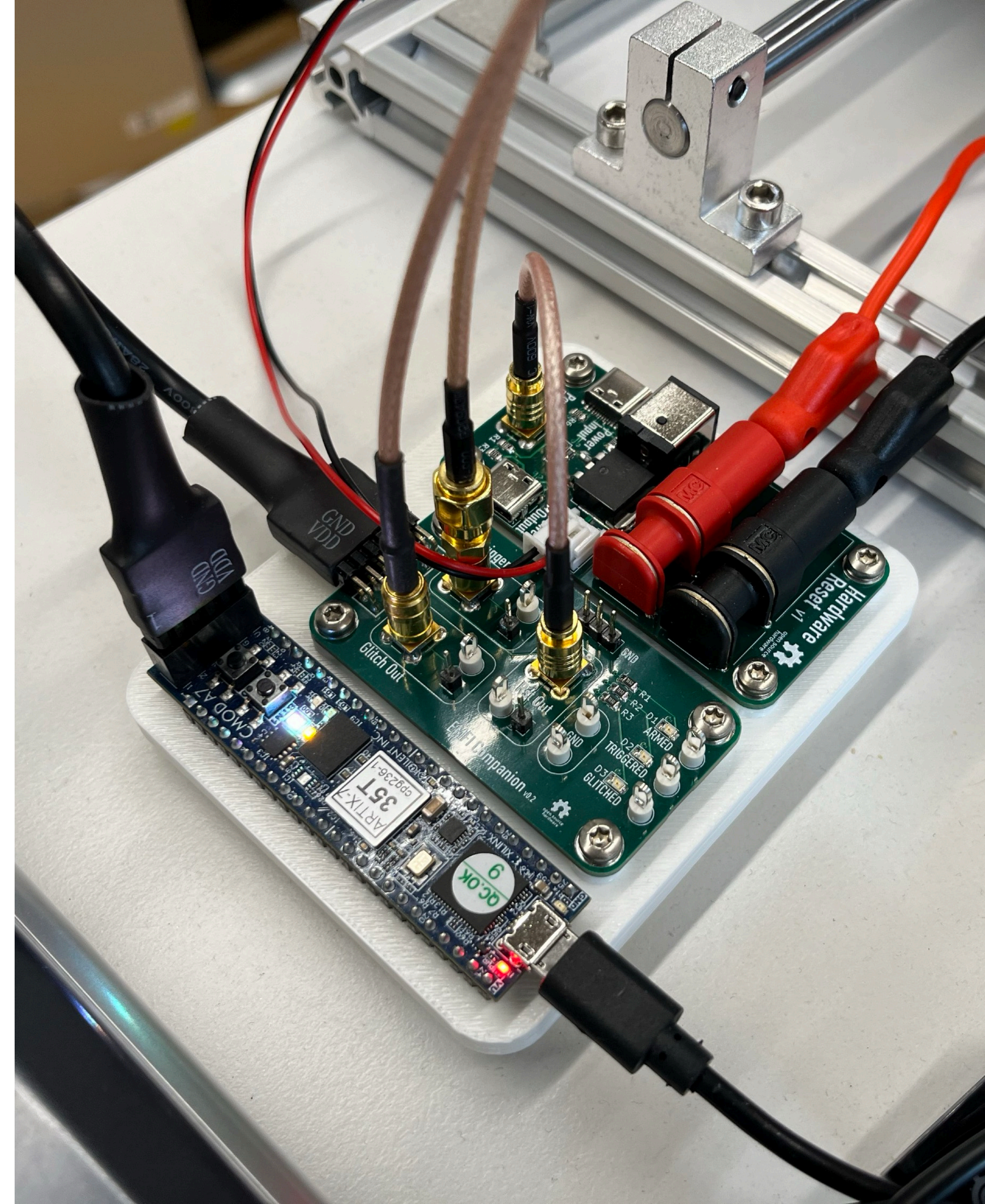
try_num	x	y	voltage	delay	width	normal	fault	brownout	timeout	crash	data
0	-1.0	1.0	470	100	20	TRUE	FALSE	FALSE	FALSE	FALSE	b'No luck, try again! 100 \r\n'
4	-1.0	1.0	470	100	20	FALSE	FALSE	FALSE	FALSE	TRUE	b''
5	-1.0	1.0	470	100	20	FALSE	FALSE	TRUE	FALSE	FALSE	b'dESP-ROM:esp32c3-api1-20
6	-1.0	1.0	470	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-api1-20
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-api1-20
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](https://github.com/unixb0y/EMFI-Resources)

 [@unixb0y@chaos.social](https://chaos.social/@unixb0y)

 [@unixb0y](https://twitter.com/unixb0y)

 [dtoldo@seemoo.de](mailto:dtoldo@seemoo.de)

