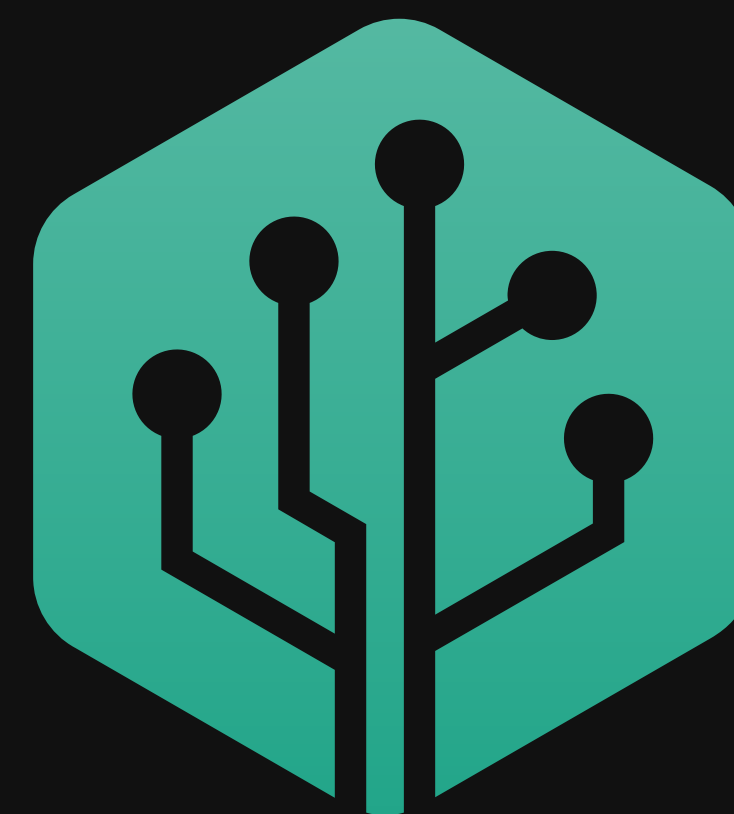


ACE up the Sleeve

Hacking into Apple's new USB-C controller



whoami

- Thomas Roth aka stacksmashing
- Security researcher - Hardware & Firmware
- Co-founder at hextree.io
- Twitter: @ghidraninja YouTube: @stacksmashing

Thanks

- Siguza
- Oly (Thunderbolt Patcher)
- AsahiLinux Team
- Carlo Maragno
- Jiska, Fabian & Caro
- Carlo Maragno
- Marc Zyngier (maz)
- T8012 Dev Team
 - aunali1
 - h0m3us3r
 - mrarm
 - Rick Mark

The backstory...





The obvious stuff

Charging

USB

Video & Audio



The obvious stuff

Charging

USB

Video & Audio



The cool stuff

JTAG

UART

SDQ



Tamarin Cable

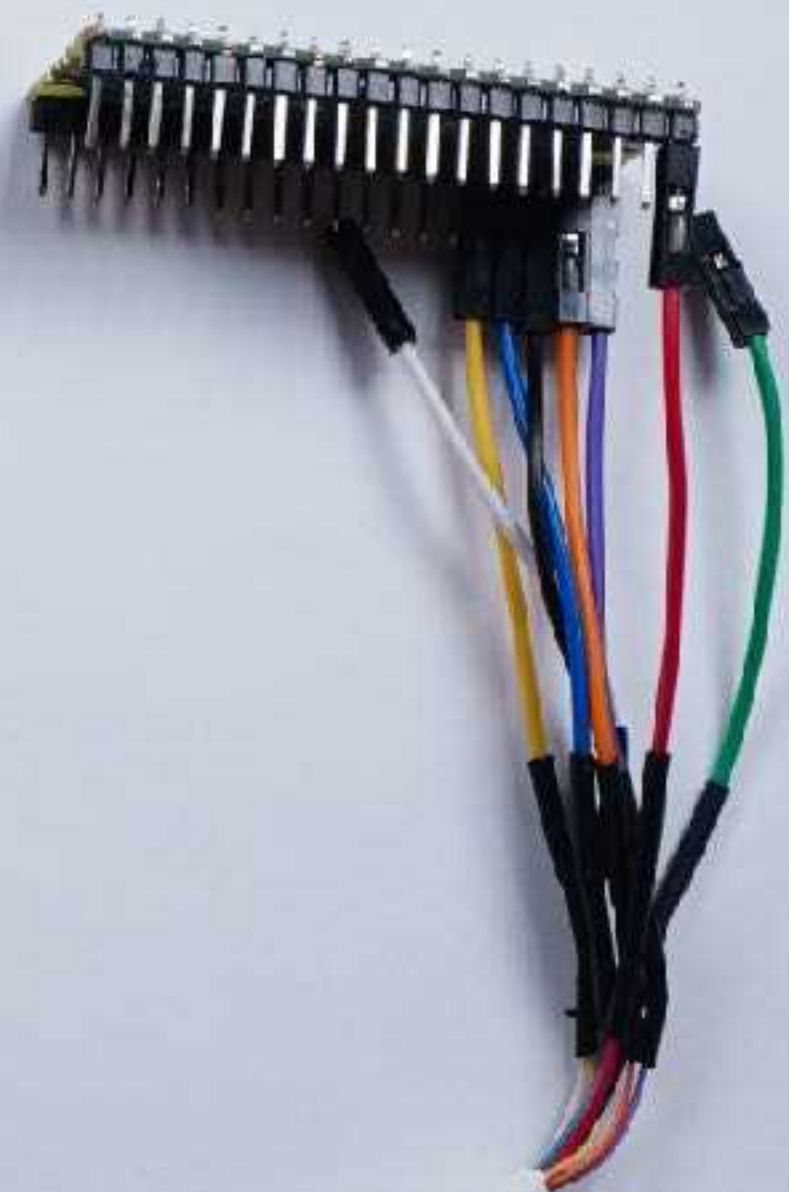


The cool stuff

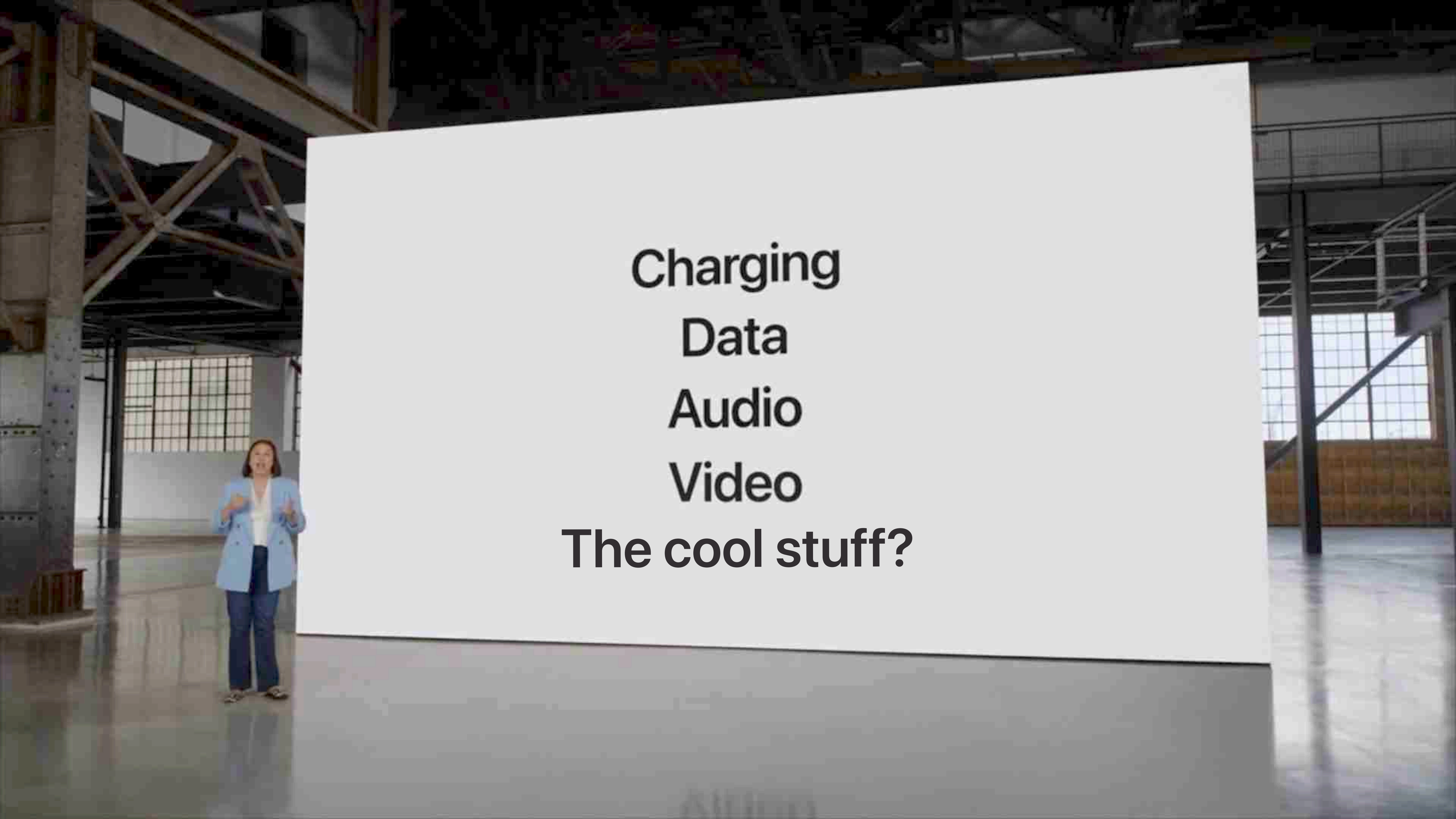
JTAG

UART

SDQ



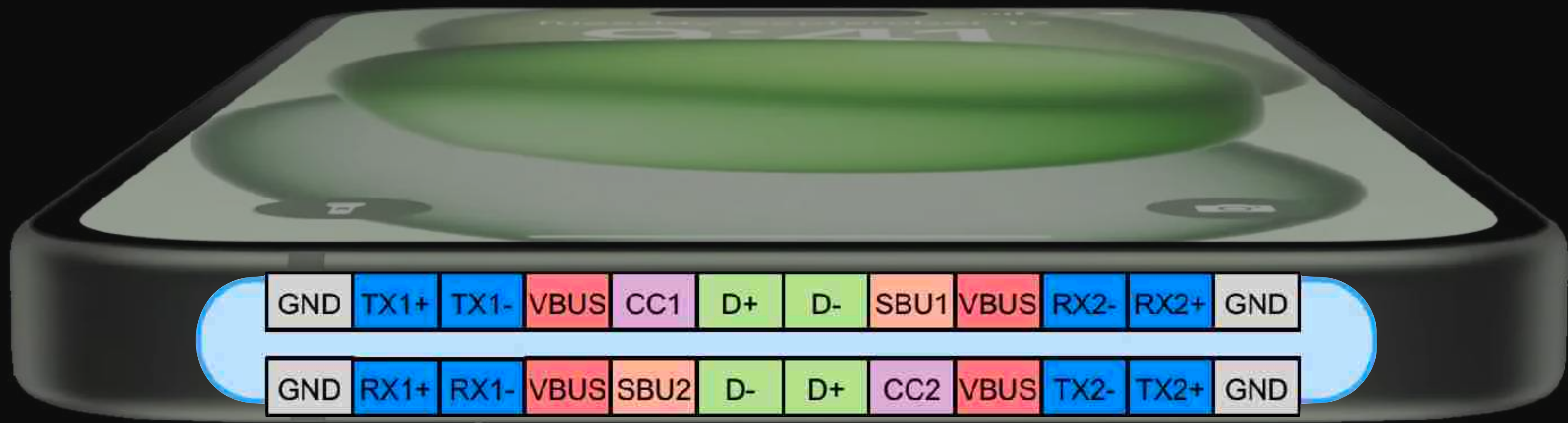


A woman with dark hair, wearing a light blue blazer over a white top and dark blue trousers, stands to the left of a large white screen. She is gesturing with her hands as if presenting. The screen is positioned in a large, modern industrial space with high ceilings, exposed steel beams, and large windows in the background. The floor is highly reflective, showing the woman and the screen. The text on the screen is centered and reads:

**Charging
Data
Audio
Video
The cool stuff?**

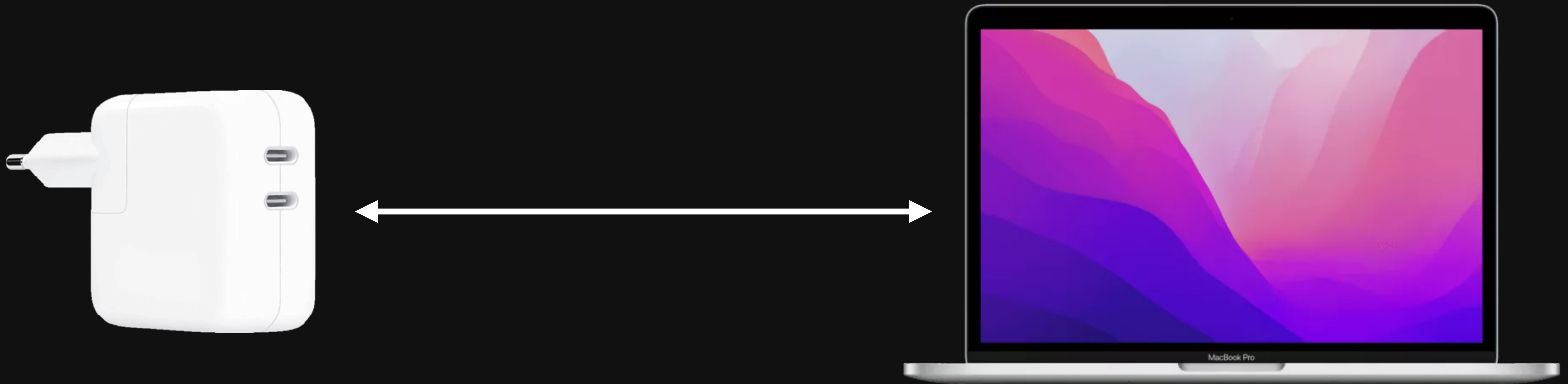




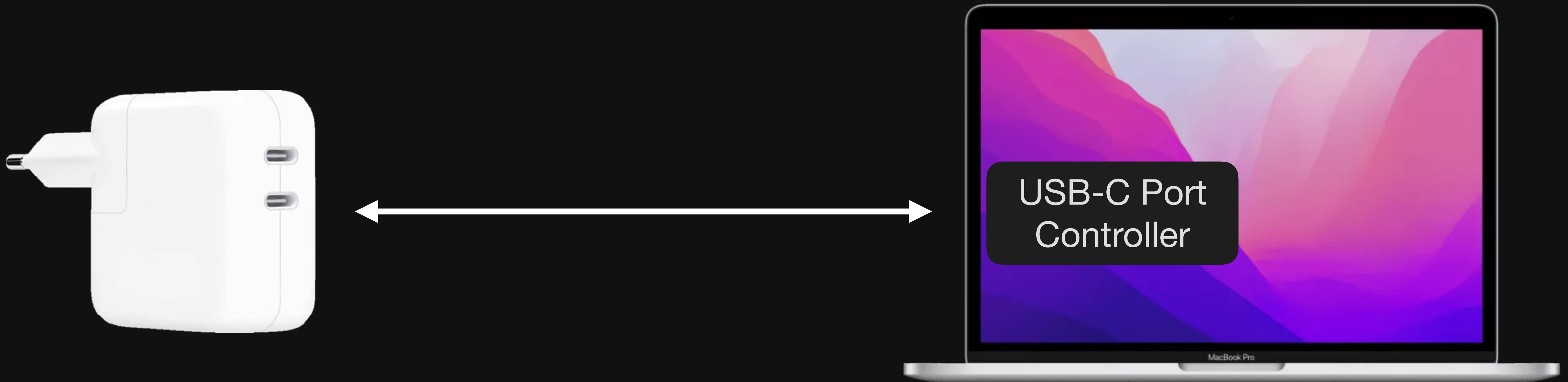




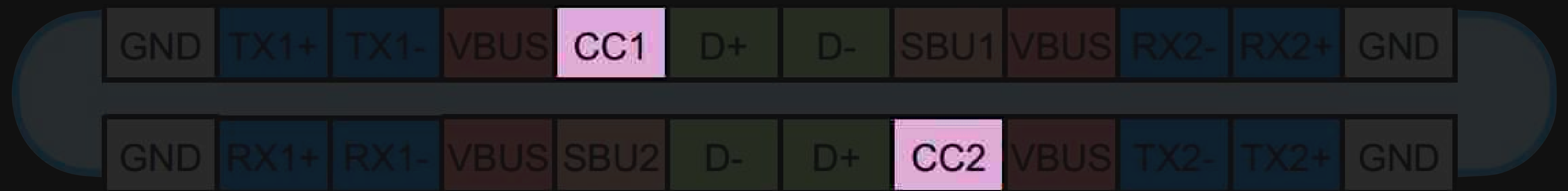
USB-PD Negotiation



USB-PD Negotiation

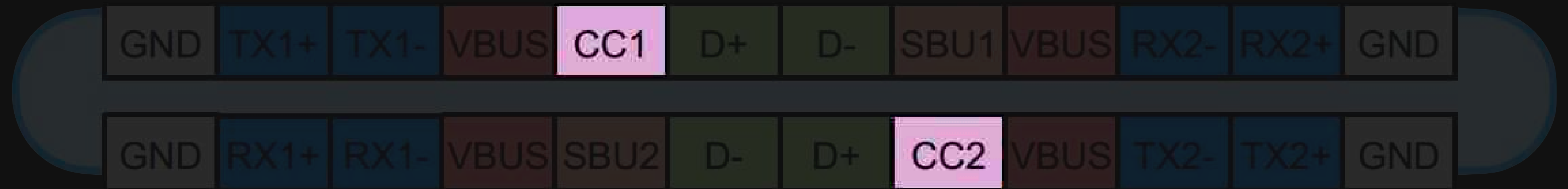






Configuration Channel

All handled by the USB-C Port (Micro)controller



Configuration Channel

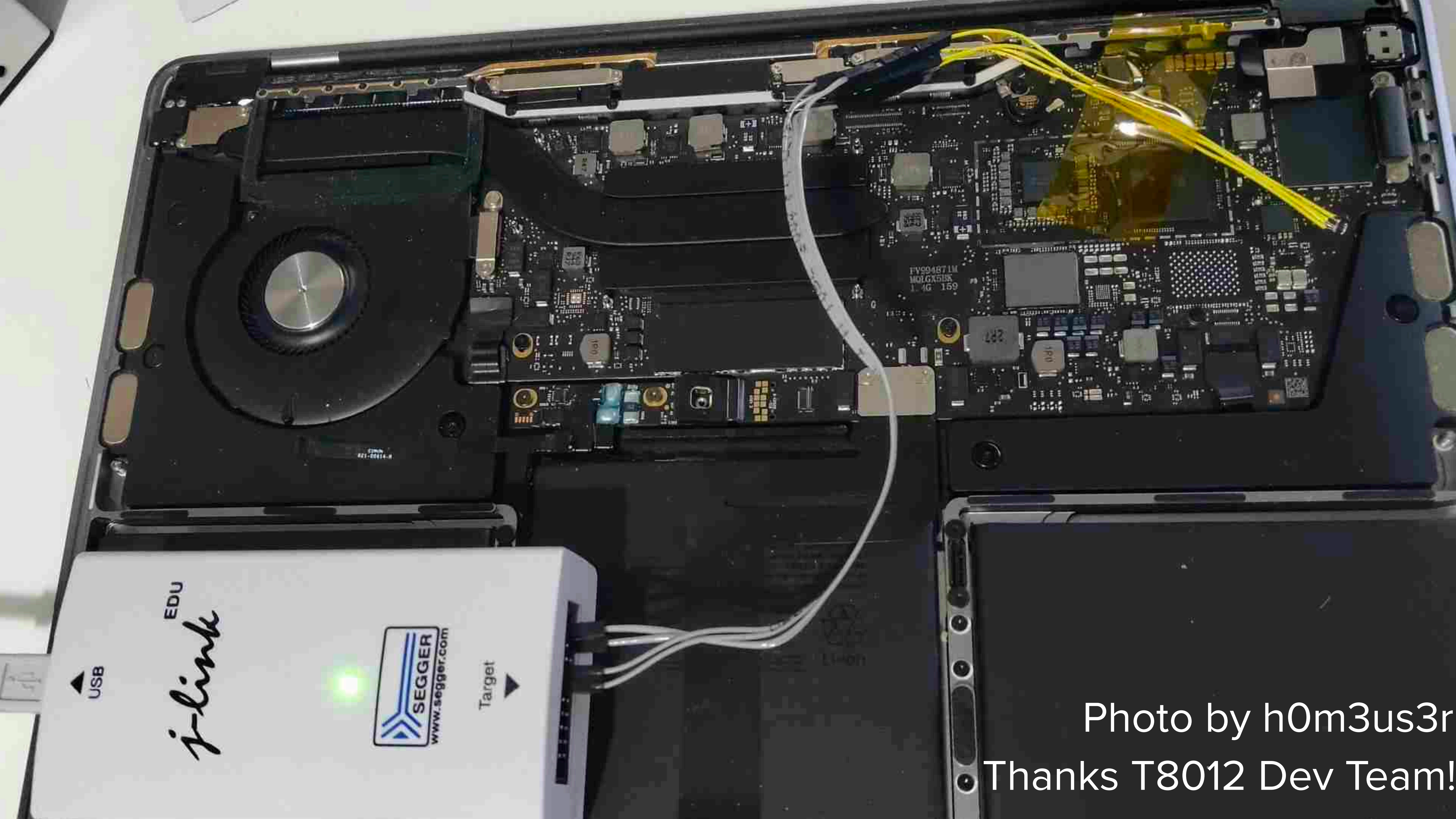


Photo by h0m3us3r
Thanks T8012 Dev Team!

web.archive.org

https://blog.t8012.dev/ace-part-1/

Go

SEP 2020

OCT 23 2021

NOV 2022

About this capture

The T2 Development Blog

Home Blog Contact Store

Subscribe

USB-PD

ACE: Apple Type-C Port Controller Secrets | Part 1

Exposing Apple's Vendor Defined Messaging protocol, AppleVDM, built on USB-PD.

mrarm, Aun-Ali Zaidi

Dec 30, 2020 · 16 min read

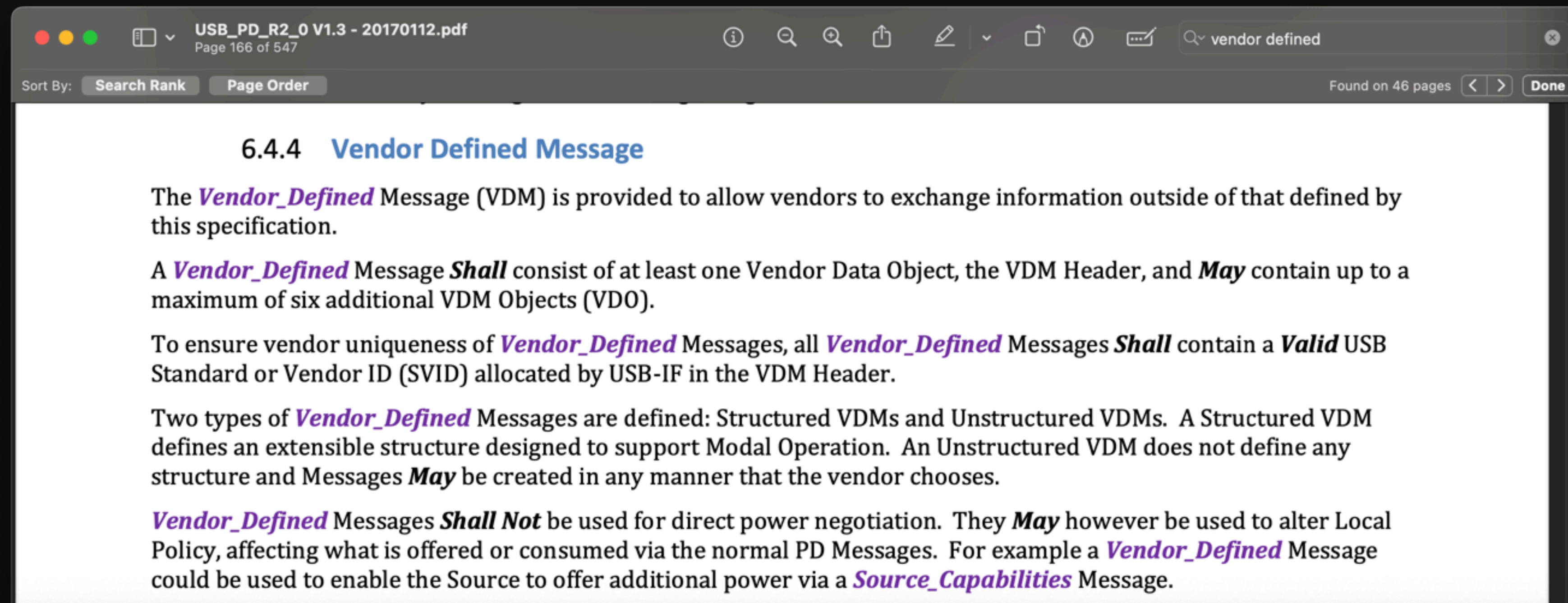
VDM

VDM

Vendor Defined Messages

VDM

Vendor Defined Messages



Protocol Summary

With the code above available, it is possible to finally sum up how this protocol works. All messages must come from `SOP'DBG` or `SOP' 'DBG`.

AppleVDM `0x10` : Get Action List

Input: `{ 0x5AC8010 }`

Reply: Shorts encoded using VDM (first short in high 16 bytes, second in low 16 bytes). Zero terminated.

Example reply VDOs: `05020702 06060206 01060105 02030103`
`00000000`

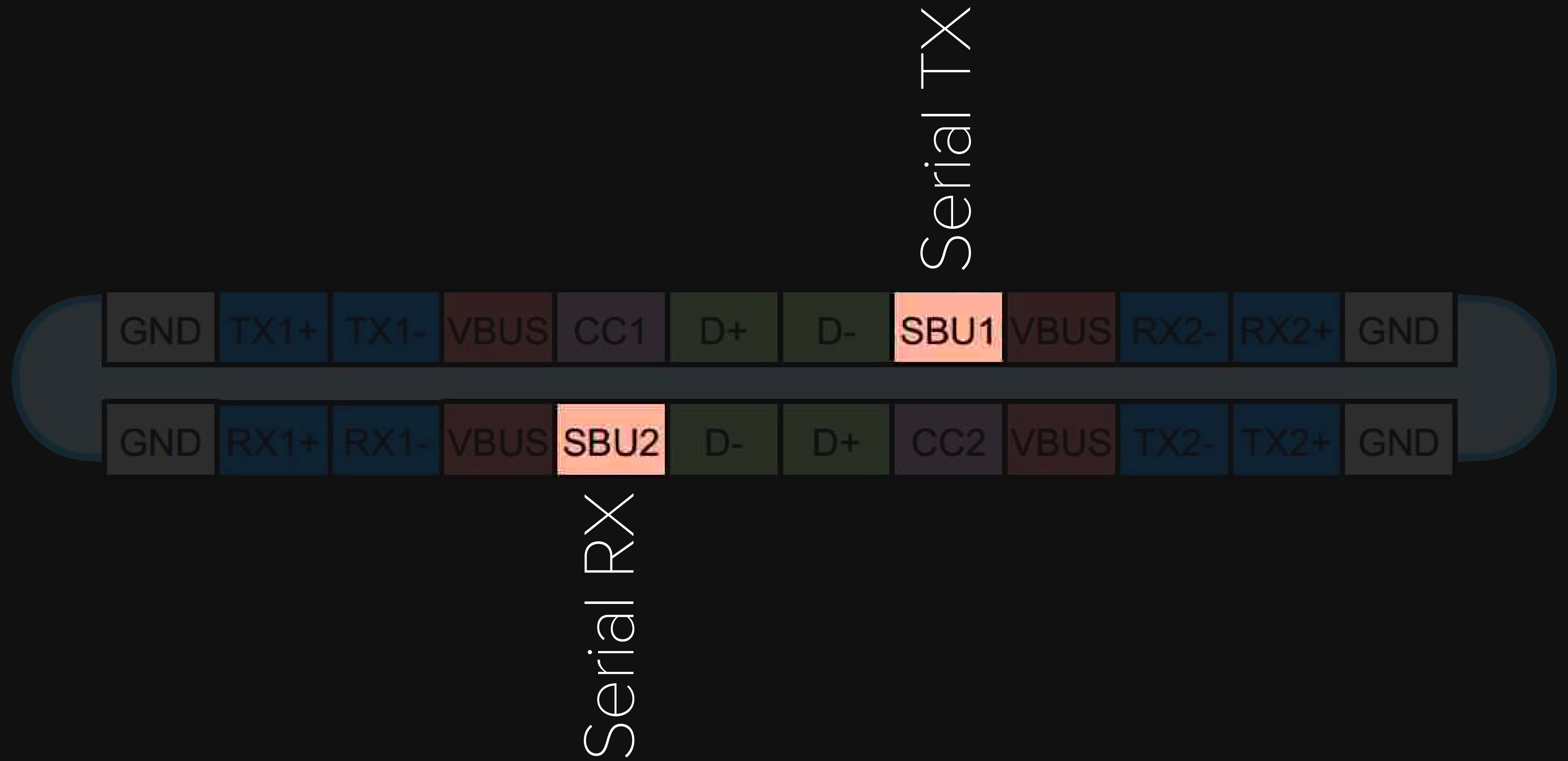
AppleVDM `0x11` : Get Action Info

Parameters:

* `uint16_t ActionId` - specifies the action, taken from 0x10 reply

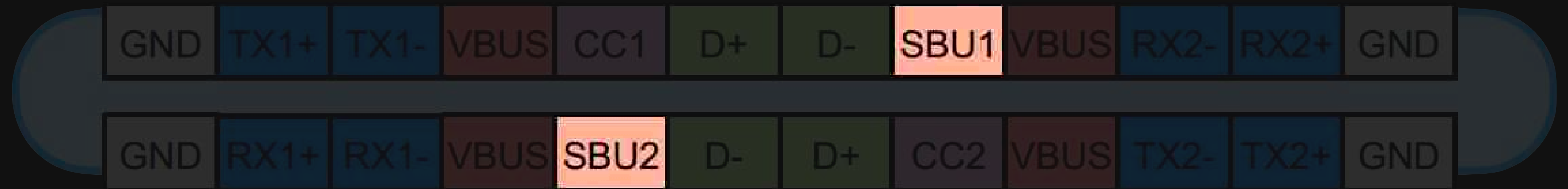
Input: `{ 0x5AC8011, ActionId }`



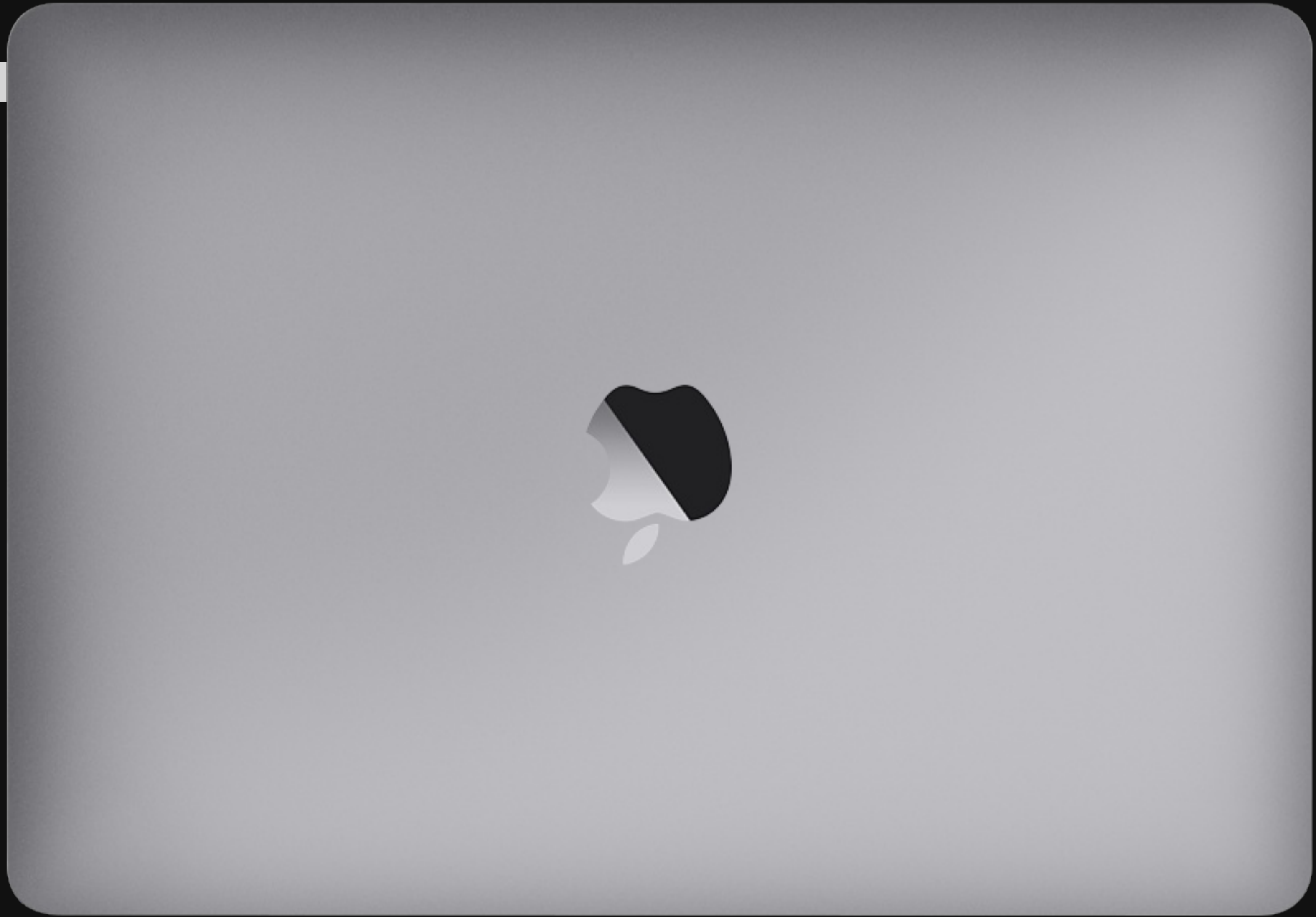


VDM action 0x306

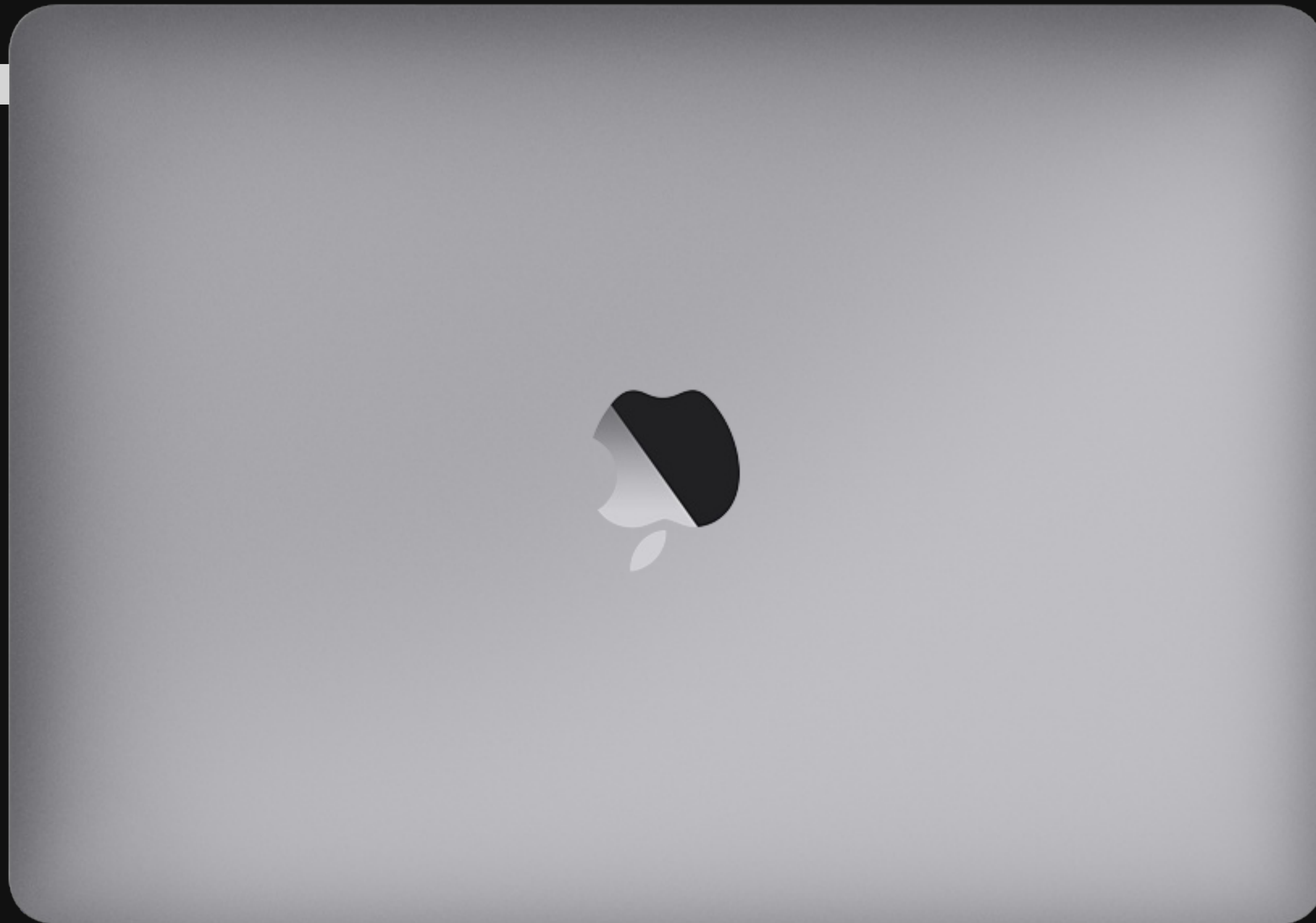
Serial TX



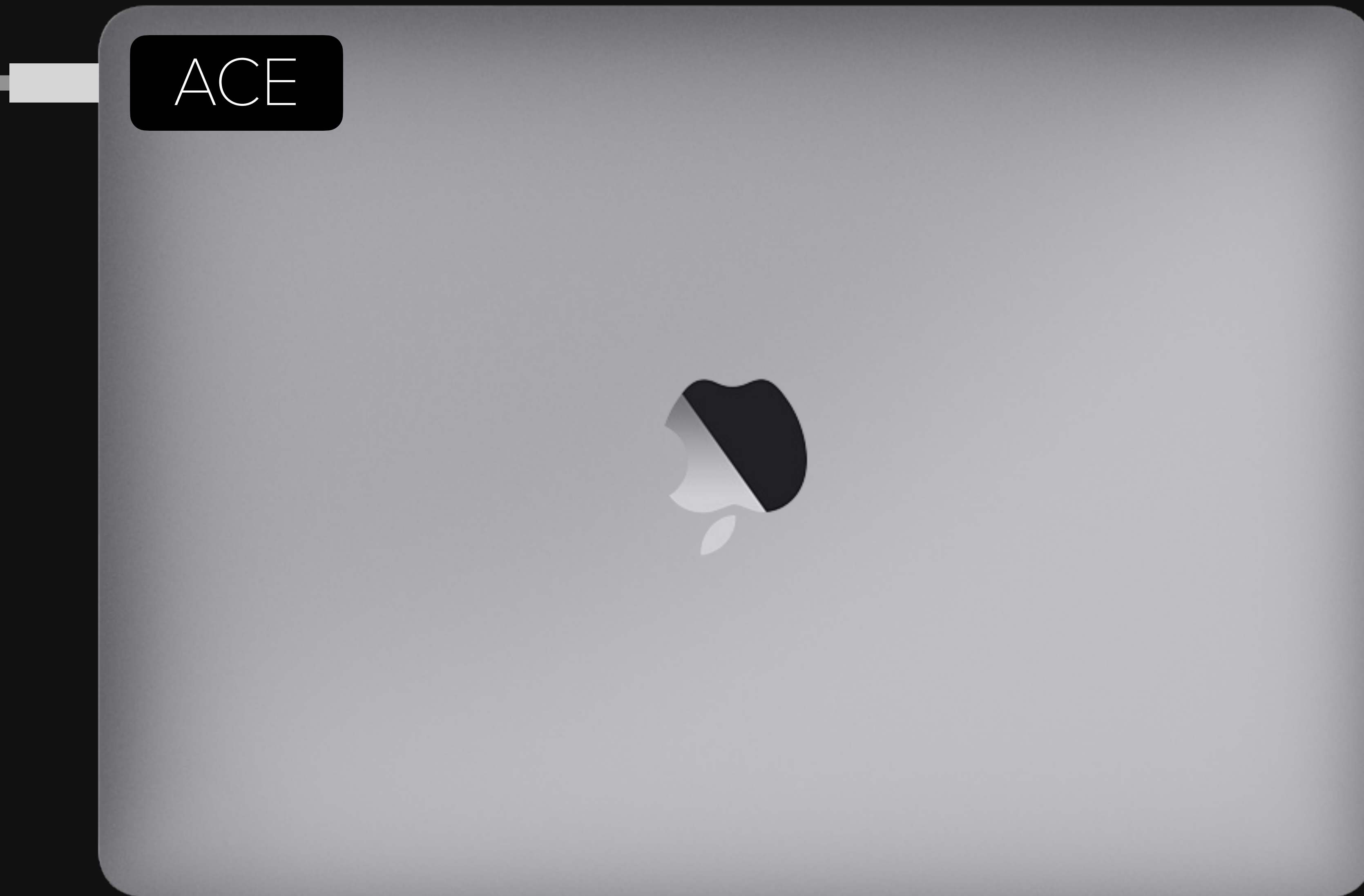
Serial RX



Type-C Port Controller



Type-C Port Controller



Type-C Port Controller

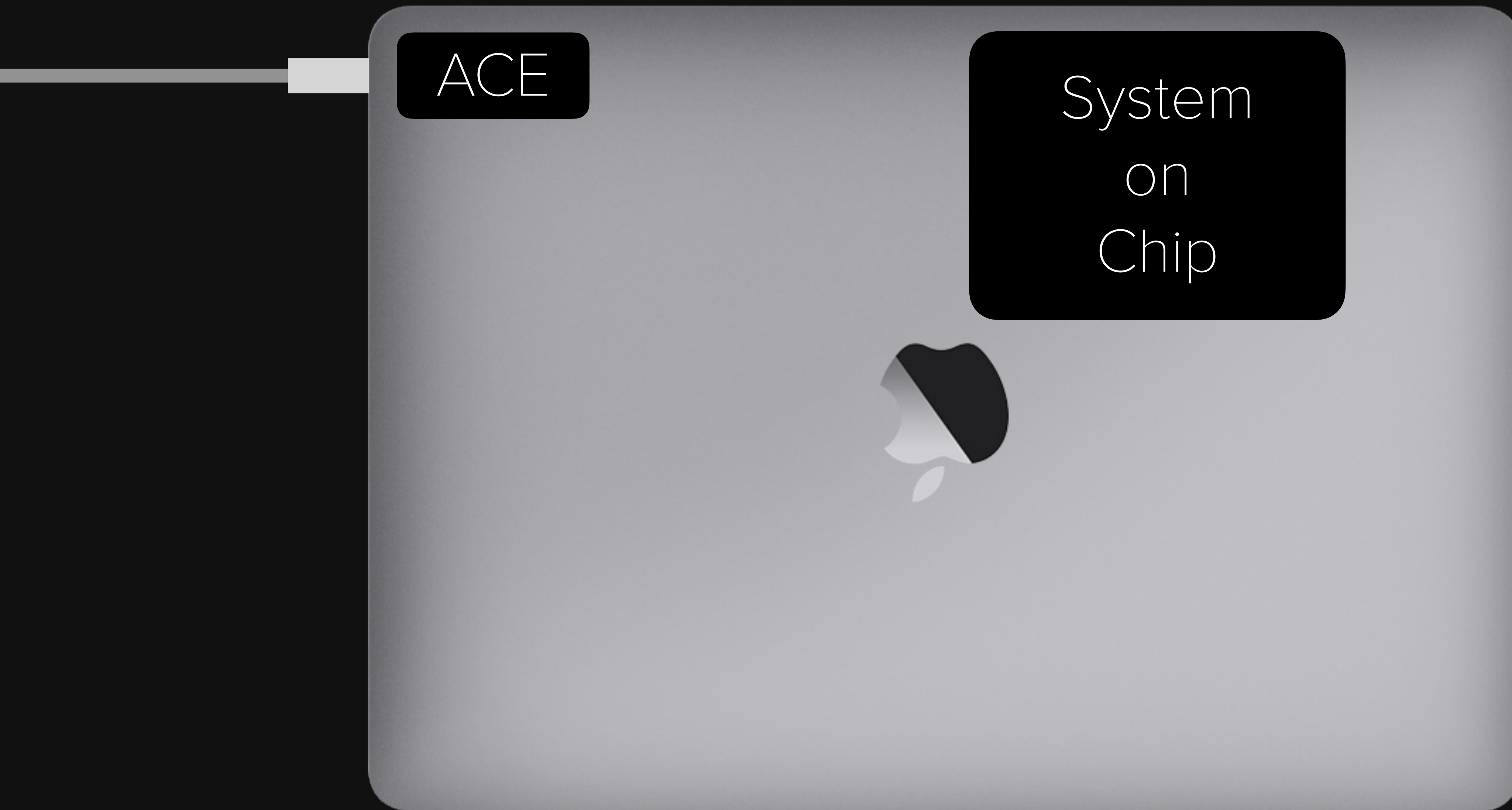
ACE

ACE2

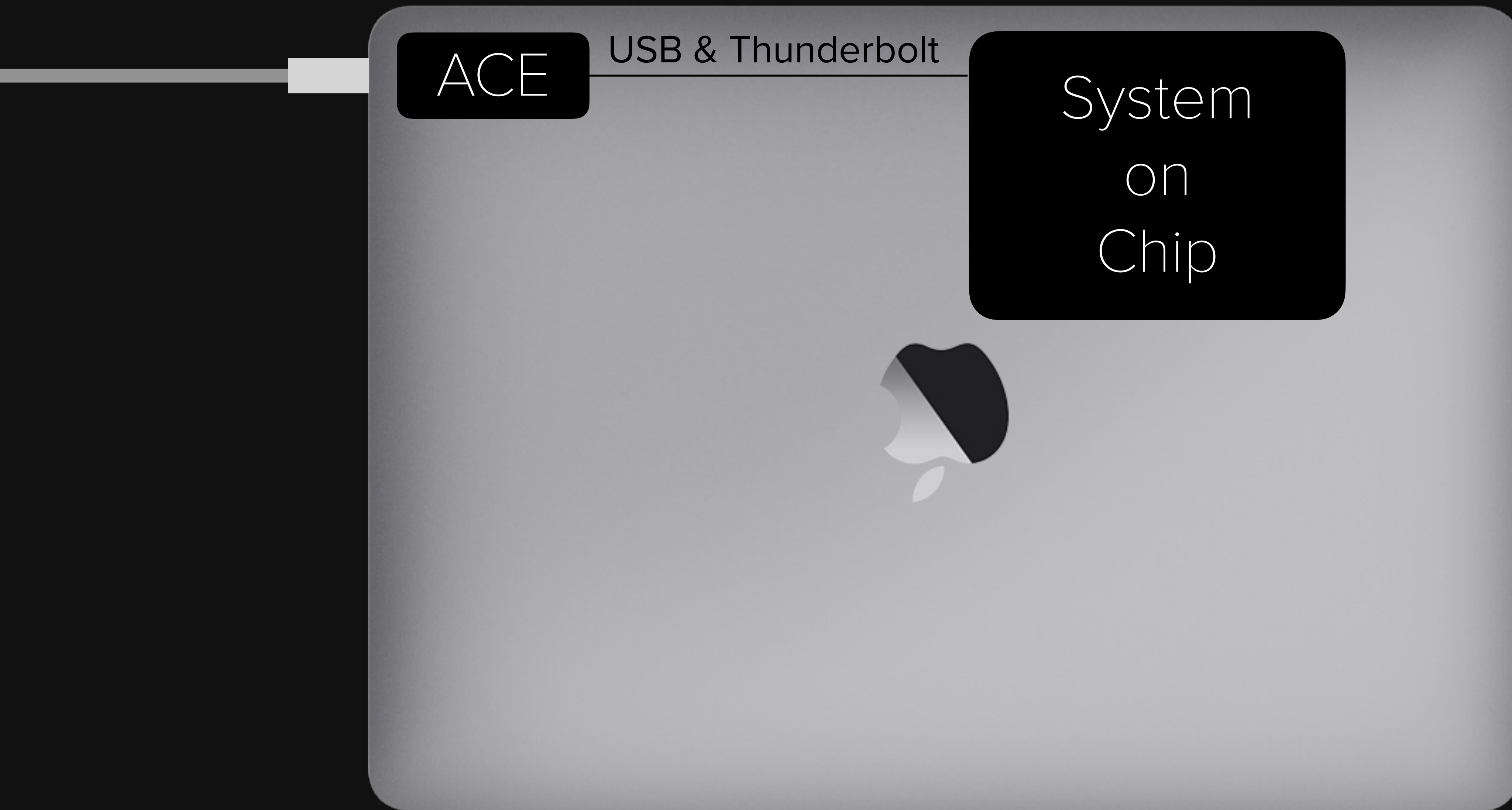
ACE3



Type-C Port Controller



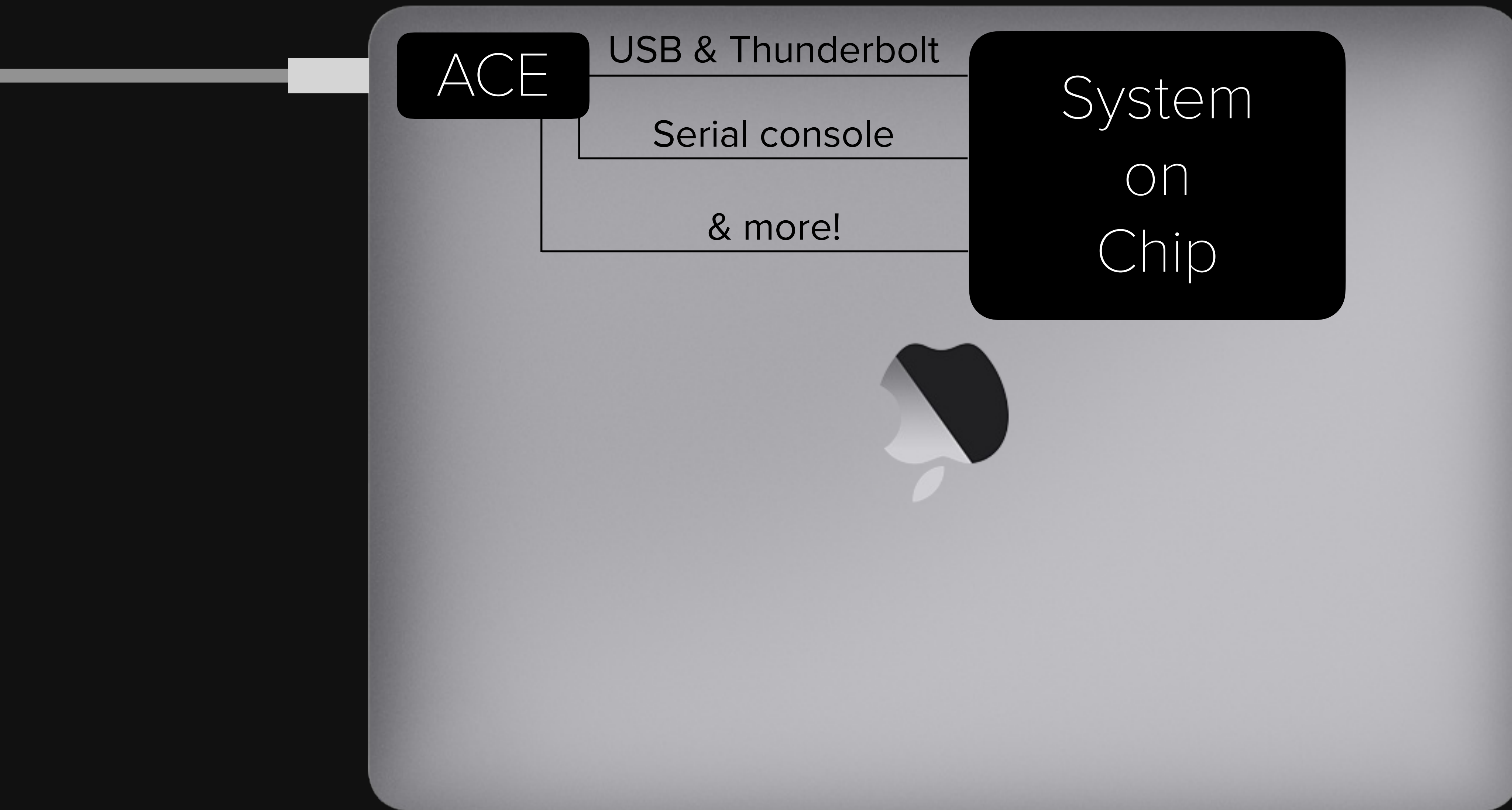
Type-C Port Controller



Type-C Port Controller



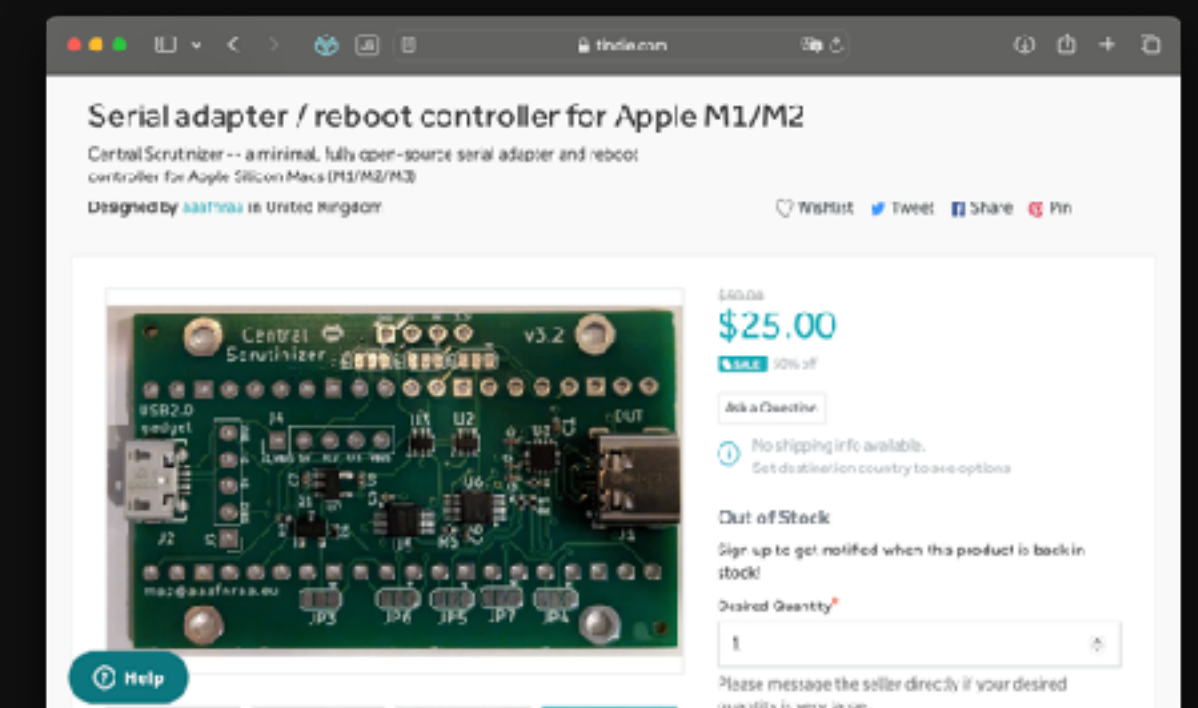
Type-C Port Controller



But how can we send VDM?

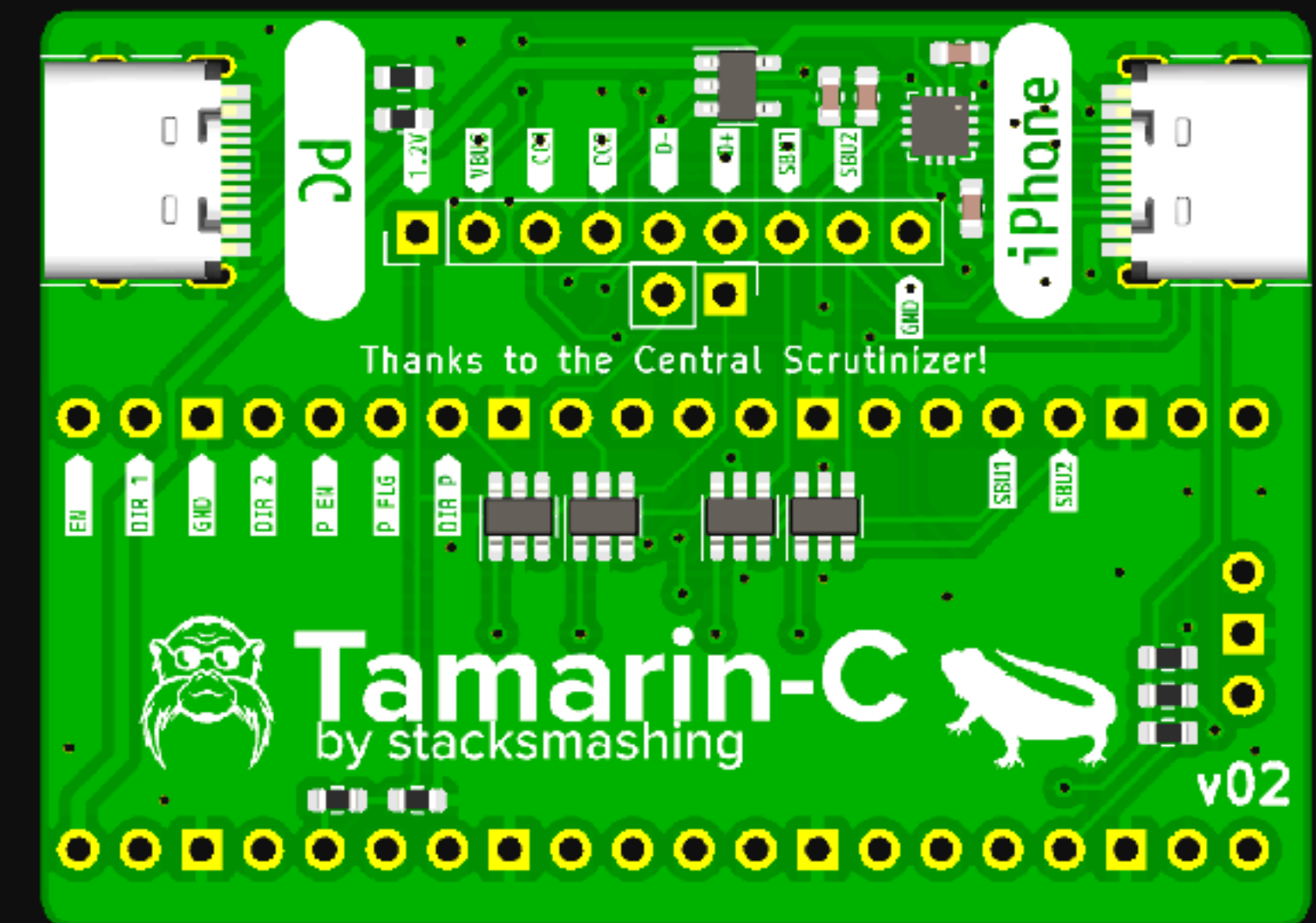
How can we send VDM?

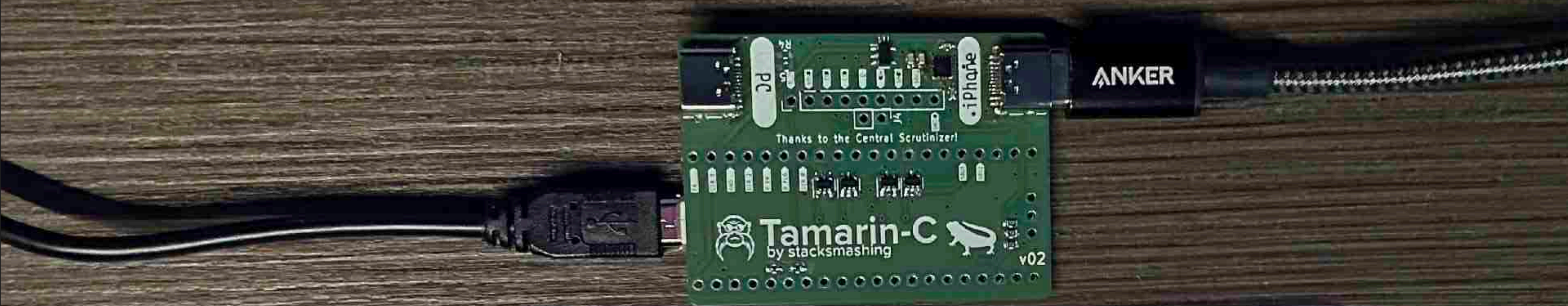
- macvdmtool
Back-left port of MacBook Pro to get serial etc
- Central Scrutinizer
Hardware tool to get serial console on MacBook

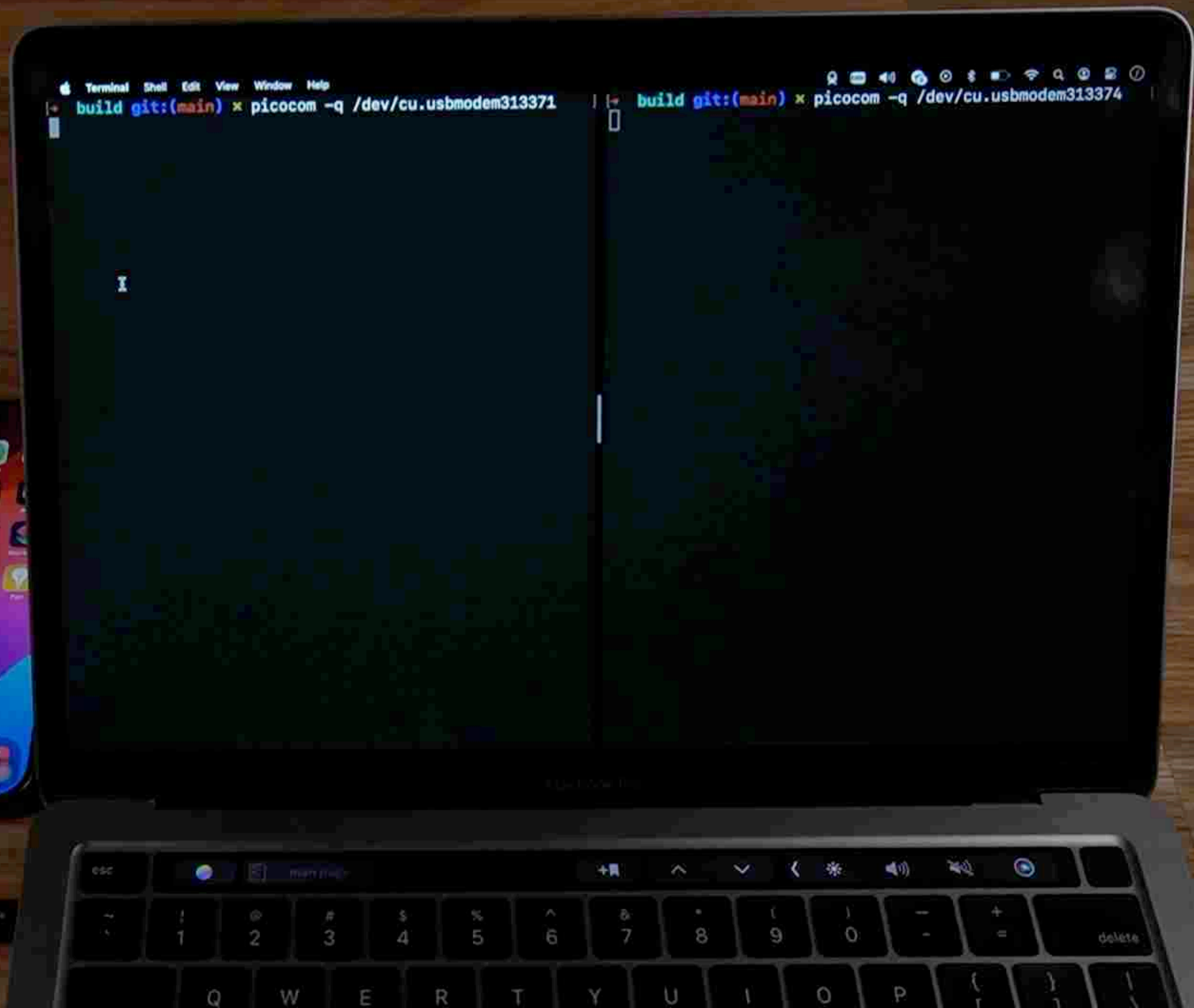


Tamarin-C

- Allows bi-directional access to internal busses
- JTAG probe integrated
- Discovered SPMI on iPhone 15 & M3 Pro/Max



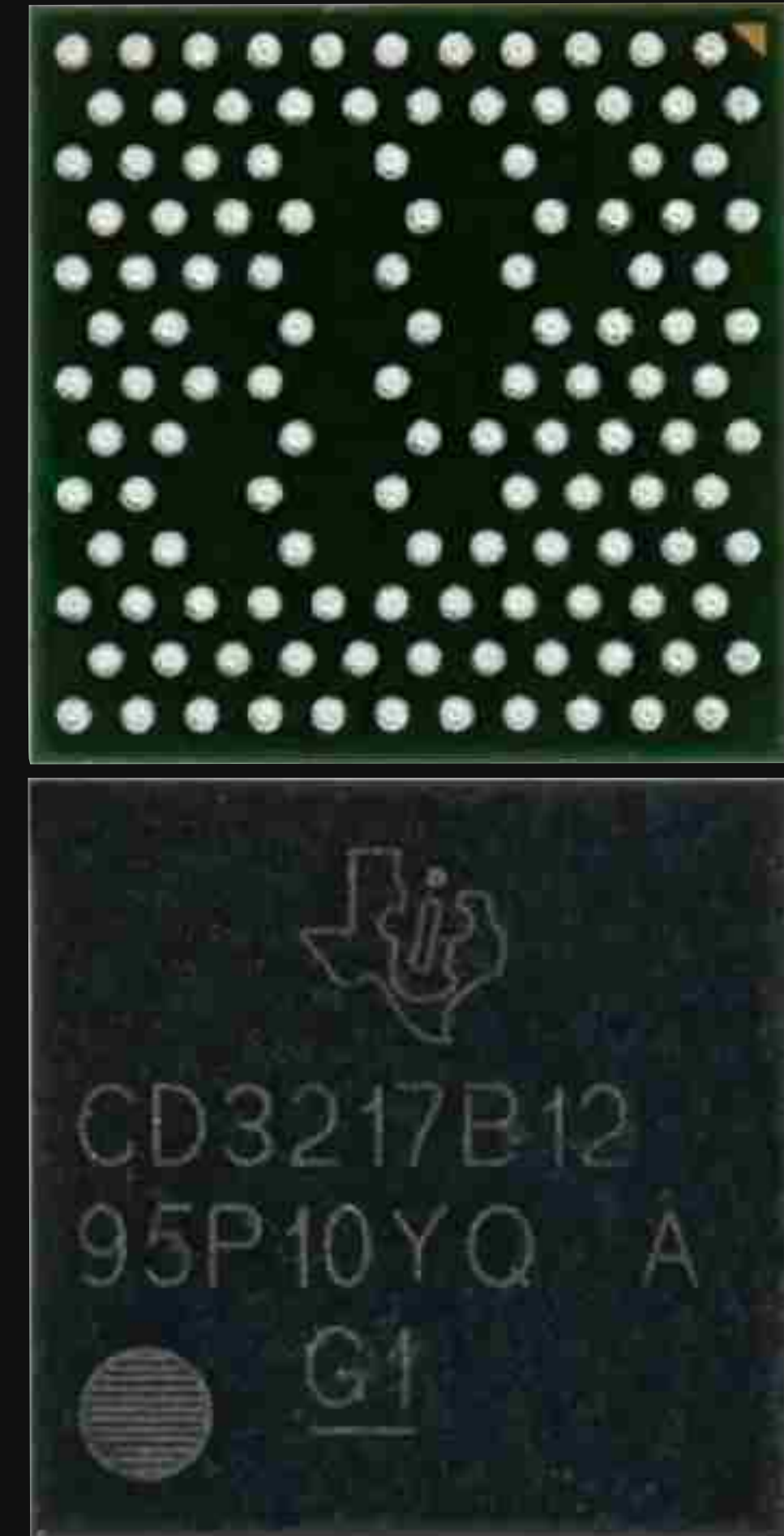




The ACE2

ACE2

- CD3217 - USB-C / PD Controller
- Arm-based and connected via I2C
- Found on MacBooks starting with the T2





TPS65986

SLVSD13C – OCTOBER 2015 – REVISED AUGUST 2016

TPS65986 USB Type-C and USB PD Controller and Power Switch

1 Features

- **USB Power-Delivery (PD) Controller**
 - Mode Configuration for Source (Host), Sink (Device), or Source-Sink
 - Bi-Phase Marked Encoding/Decoding (BMC)
 - Physical Layer (PHY) Protocol
 - Policy Engine
 - Configurable at Boot and Host-Controlled
- **USB Type-C Specification Compliant**
 - Detect USB Cable-Plug Attach
 - Cable Orientation and Role Detection
 - Assign CC and VCONN Pins
 - Advertise Default, 1.5 A, or 3 A for Type-C Power
- **Port Power Switch**
 - 5-V, 3-A Switch to VBUS for Type-C Power
 - 5-V to 20-V, 3-A Bidirectional Switch to or from VBUS for USB PD Power
 - 5-V, 600-mA Switches for VCONN
 - Overcurrent Limiter, Overvoltage Protector
 - Slew-Rate Control
 - Hard Reset Support
- **Port Data Termination**
 - USB 2.0 Low-Speed Endpoint
- **Power Management**
 - Power Supply from 3.3 V or VBUS Source
 - 3.3-V LDO Output for Dead Battery Support
- **BGA MicroStar Junior Package**
 - 0.5-mm Pitch
 - Through-Hole Via Compatible for All Pins

3 Description

The TPS65986 device is a stand-alone USB Type-C and power delivery (PD) controller providing cable plug and orientation detection at the USB Type-C connector. Upon cable detection, the TPS65986 device communicates on the CC wire using the USB PD protocol. When cable detection and USB PD negotiation are complete, the TPS65986 device enables the appropriate power path and configures alternate mode settings for (optional) external multiplexers.

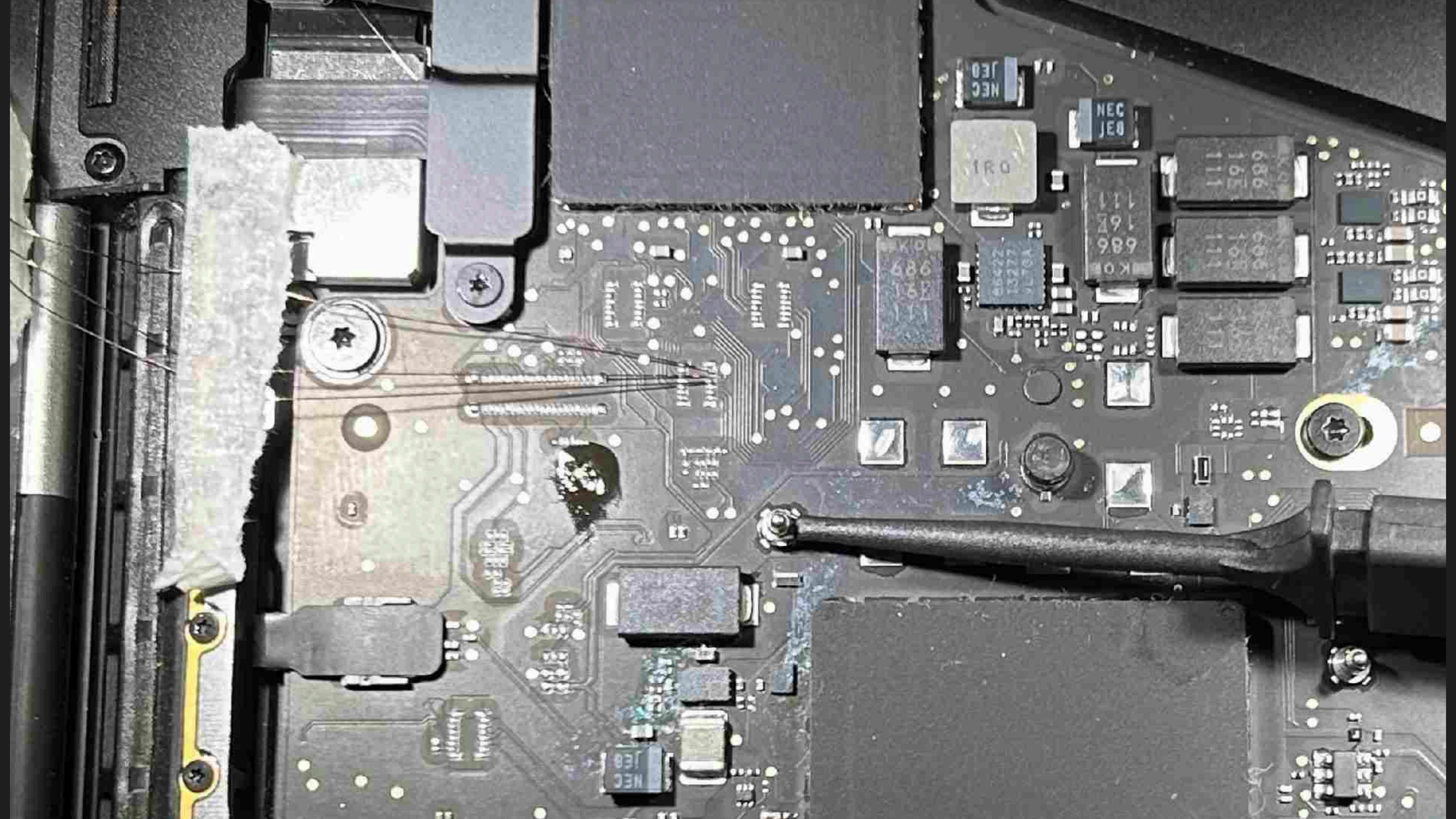
The mixed-signal front end on the CC pins advertises default (900 mA), 1.5 A, or 3 A for Type-C power sources, detects a plug event and determines the USB Type-C cable orientation, and autonomously negotiates USB PD contracts by adhering to the specified bi-phase marked coding (BMC) and physical layer (PHY) protocol.

The port power switch passes up to 3 A downstream at 5 V for legacy and Type-C USB power. An additional bi-directional switch path provides USB PD power up to 3 A at a maximum of 20 V as either a source (host), sink (device), or source-sink.

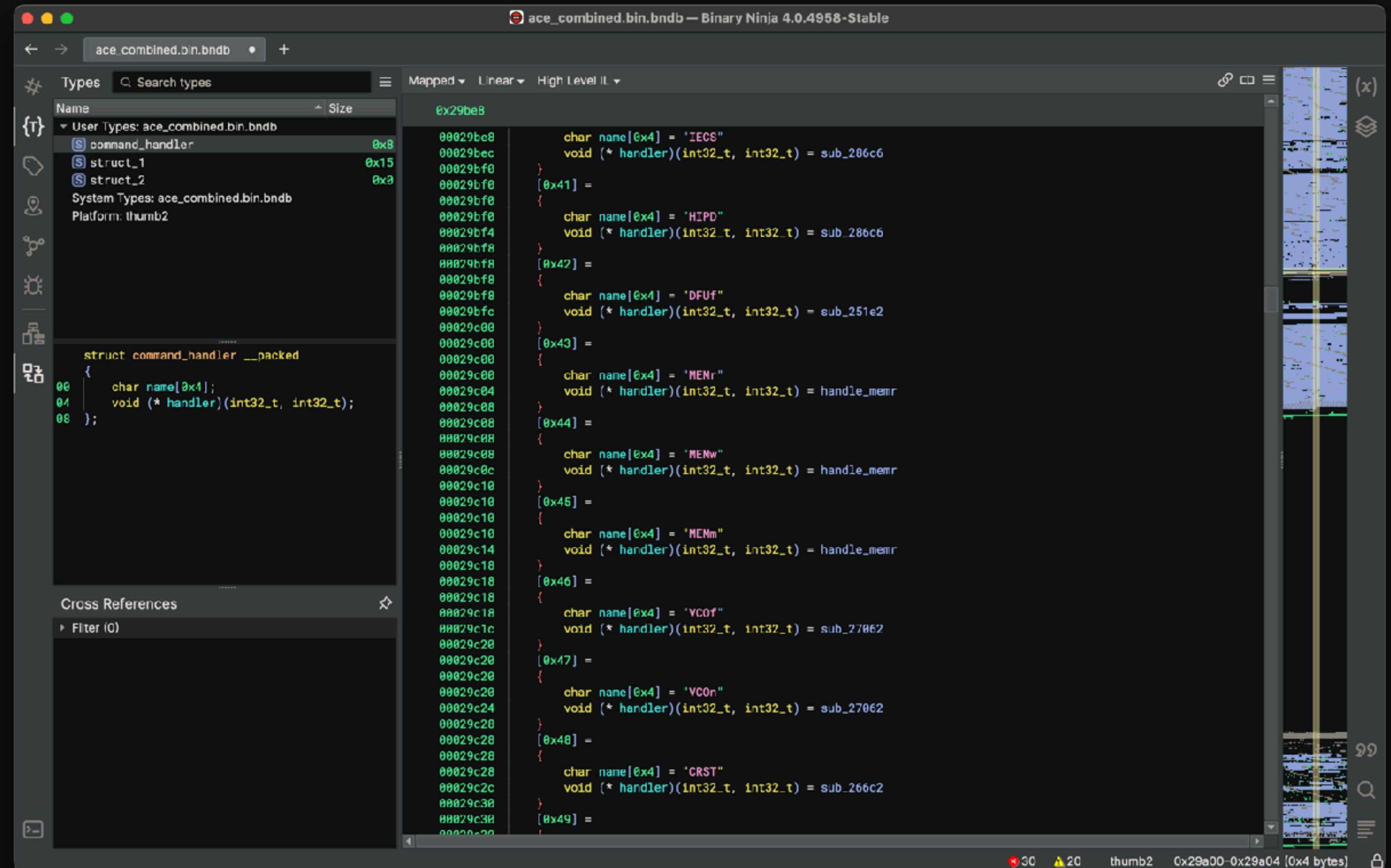
The TPS65986 device is also an upstream-facing port (UFP), downstream-facing port (DFP), or dual-role port for data. The port data termination passes data to or from the top or bottom D+/D- signal pair to the USB 2.0 low-speed endpoint. The power management circuitry uses a 3.3-V power supply inside the system and also uses VBUS to start up and negotiate power from a dead battery or no battery condition.

Device Information⁽¹⁾

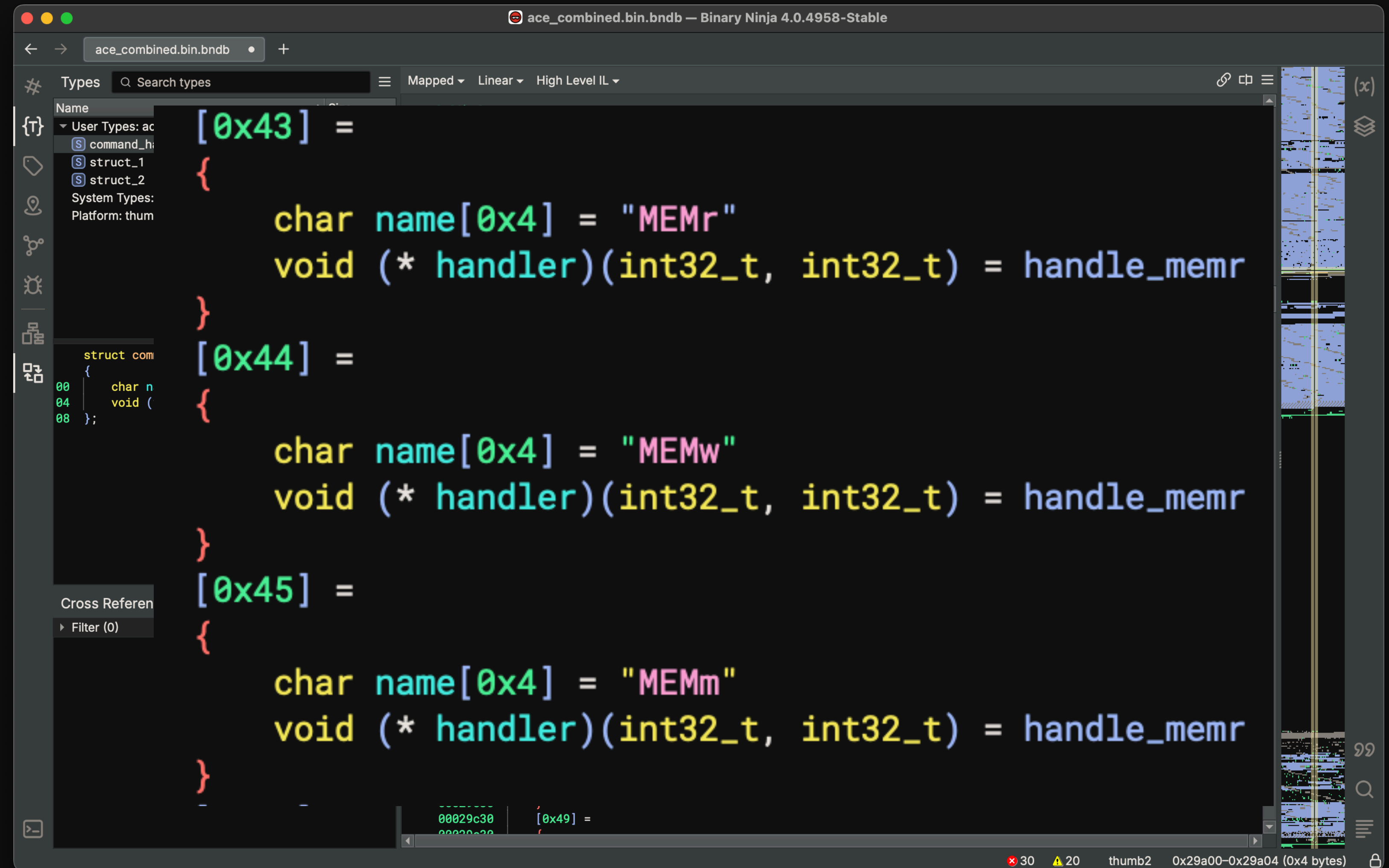
PART NUMBER	PACKAGE	BODY SIZE (NOM)
-------------	---------	-----------------



- Identified command-handler
- Contains "privileged" commands
- MEMr/MEMw/MEMm



- Identified command-handler
- Contains "privileged" commands
- MEMr/MEMw/MEMm



```
ace_combined.bin.bndb — Binary Ninja 4.0.4958-Stable

Types Search types Mapped Linear High Level IL

Name
User Types: ac
[S] command_h
[S] struct_1
[S] struct_2
System Types:
Platform: thum

struct com
{
  char n
  void (
};

[0x43] =
{
  char name[0x4] = "MEMr"
  void (* handler)(int32_t, int32_t) = handle_memr
}

[0x44] =
{
  char name[0x4] = "MEMw"
  void (* handler)(int32_t, int32_t) = handle_memr
}

[0x45] =
{
  char name[0x4] = "MEMm"
  void (* handler)(int32_t, int32_t) = handle_memr
}

[0x49] =
{
  char name[0x4] = "MEMm"
  void (* handler)(int32_t, int32_t) = handle_memr
}

00029c30 00029c30 [0x49] =
```


Apple HPM Bus



- Internal interface to talk to I2C
- Used by HPMDiagnose & co
- Can be used to communicate with ACE

FourCC Commands

- Different registers that can be written/read to/from
- 4-digit integer/ASCII commands
 - Commands in register 9, Status in 3
- acetool - Tool to communicate with ACE

```
→ acecomm git:(main) x sudo ./acetool IOService:/AppleARMP
10F00000/AppleT810xIO/i2c0@35010000/AppleS5L8940XI2CControl
sManager@6B/AppleHPMbusController/hpm1/AppleHPMARMi2C GAID
Status: APP
Running command: GAID - Data: 0
Executing command
Status: BOOT
→ acecomm git:(main) x
```

FourCC Commands

```
→ acecomm git:(main) ✖ sudo ./acetool IOService:/Apple  
10F00000/AppleT810xIO/i2c0@35010000/AppleS5L8940XI2CCo  
sManager@6B/AppleHPMBusController/hpm1/AppleHPMARMi2C  
Status: APP  
Running command: GAID – Data: 0  
Executing command  
Status: BOOT  
→ acecomm git:(main) ✖ █
```





SPI Flash



SPI Flash



Send firmware
via UART



ACE2: SPI Flash



SPI Flash

- Does not contain full firmware
- Contains "patches" for the ROM
- Makes reversing... annoying

```
int32_t sub_2275a(int32_t arg1, void* arg2)
```

0002275a	int32_t var_18 = r3
00022760	char* r5 = *(arg2 + 4)
00022762	void* r6 = nullptr
00022766	void* r4 = nullptr
0002276e	data_200443c5 = (zx.d(*r5) << 0x1e u>> 0x1e).b
00022776	*r5 = (zx.d(*r5) << 0x1c u>> 0x1c).b
00022778	r5[1] = 0
0002277c	sub_293b8(0)
00022788	data_10002008 = data_10002008 & 0xffffffffbf
00022792	int32_t r2_2 = data_1000048c
00022796	uint32_t r0_5 = r2_2 << 0x10 u>> 0x18
0002279c	uint32_t r7 = zx.d(r2_2.b)
0002279e	uint32_t r1_2 = zx.d(data_200443c5)

```
int32_t sub_2275a(int32_t arg1, void* arg2)
```

```
0002275a | int32_t var_18 = r3
00022760 | char* r5 = *(arg2 + 4)
00022762 | void* r6 = nullptr
00022766 | void* r4 = nullptr
0002276e | data_200443c5 = (zx.d(*r5) << 0x1e u>> 0x1e).b
00022776 | *r5 = (zx.d(*r5) << 0x1c u>> 0x1c).b
00022778 | r5[1] = 0
0002277c | sub_293b8(0)
00022788 | data_10002008 = data_10002008 & 0xffffffffbf
00022792 | int32_t r2_2 = data_1000048c
00022796 | uint32_t r0_5 = r2_2 << 0x10 u>> 0x18
0002279c | uint32_t r7 = zx.d(r2_2.b)
0002279e | uint32_t r1_2 = zx.d(data_200443c5)
```

```
000293b8 | int32_t sub_293b8(int32_t arg1)
```

```
000293c8 | int32_t r0_1
000293c8 | int32_t r1_1
000293c8 | int32_t r2
000293c8 | int32_t r3
000293c8 | r0_1, r1_1, r2, r3 = data_20041894(arg1)
00023606 | return sub_e0(r0_1, r1_1, r2, r3)
```

```
int32_t sub_2275a(int32_t arg1, void* arg2)
```

```
0002275a  int32_t var_18 = r3
00022760  char* r5 = *(arg2 + 4)
00022762  void* r6 = nullptr
00022766  void* r4 = nullptr
0002276e  data_200443c5 = (zx.d(*r5) << 0x1e u>> 0x1e).b
00022776  *r5 = (zx.d(*r5) << 0x1c u>> 0x1c).b
00022778  r5[1] = 0
0002277c  sub_293b8(0)
00022788  data_10002008 = data_10002008 & 0xffffffffbf
00022792  int32_t r2_2 = data_1000048c
00022796  uint32_t r0_5 = r2_2 << 0x10 u>> 0x18
0002279c  uint32_t r7 = zx.d(r2_2.b)
0002279e  uint32_t r1_2 = zx.d(data_200443c5)
```

```
000293b8  int32_t sub_293b8(int32_t arg1)
```

```
000293c8  int32_t r0_1
000293c8  int32_t r1_1
000293c8  int32_t r2
000293c8  int32_t r3
000293c8  r0_1, r1_1, r2, r3 = data_20041894(arg1)
00023606  return sub_e0(r0_1, r1_1, r2, r3)
```

```
2004188c  void* data_2004188c = sub_21f22
20041890  void* data_20041890 = sub_21f40
20041894  void* data_20041894 = sub_220c0
20041898  void* data_20041898 = sub_22146
2004189c  void* data_2004189c = sub_22186
```



```
int32_t sub_2275a(int32_t arg1, void* arg2)
```

```
0002275a  int32_t var_18 = r3
00022760  char* r5 = *(arg2 + 4)
00022762  void* r6 = nullptr
00022766  void* r4 = nullptr
0002276e  data_200443c5 = (zx.d(*r5) << 0x1e u>> 0x1e).b
00022776  *r5 = (zx.d(*r5) << 0x1c u>> 0x1c).b
00022778  r5[1] = 0
0002277c  sub_293b8(0)
00022788  data_10002008 = data_10002008 & 0xffffffffbf
00022792  int32_t r2_2 = data_1000048c
00022796  uint32_t r0_5 = r2_2 << 0x10 u>> 0x18
0002279c  uint32_t r7 = zx.d(r2_2.b)
0002279e  uint32_t r1_2 = zx.d(data_200443c5)
```

```
000293b8  int32_t sub_293b8(int32_t arg1)
```

```
000293c8  int32_t r0_1
000293c8  int32_t r1_1
000293c8  int32_t r2
000293c8  int32_t r3
000293c8  r0_1, r1_1, r2, r3 = data_20041894(arg1)
00023606  return sub_e0(r0_1, r1_1, r2, r3)
```

Loaded from flash
into RAM

```
2004188c  void* data_2004188c = sub_21f22
20041890  void* data_20041890 = sub_21f40
20041894  void* data_20041894 = sub_220c0
20041898  void* data_20041898 = sub_22146
2004189c  void* data_2004189c = sub_22186
```

```
int32_t sub_2275a(int32_t arg1, void* arg2)
```

```
0002275a  int32_t var_18 = r3
00022760  char* r5 = *(arg2 + 4)
00022762  void* r6 = nullptr
00022766  void* r4 = nullptr
0002276e  data_200443c5 = (zx.d(*r5) << 0x1e u>> 0x1e).b
00022776  *r5 = (zx.d(*r5) << 0x1c u>> 0x1c).b
00022778  r5[1] = 0
0002277c  sub_293b8(0)
00022788  data_10002008 = data_10002008 & 0xffffffffbf
00022792  int32_t r2_2 = data_1000048c
00022796  uint32_t r0_5 = r2_2 << 0x10 u>> 0x18
0002279c  uint32_t r7 = zx.d(r2_2.b)
0002279e  uint32_t r1_2 = zx.d(data_200443c5)
```

```
000293b8  int32_t sub_293b8(int32_t arg1)
```

```
000293c8  int32_t r0_1
000293c8  int32_t r1_1
000293c8  int32_t r2
000293c8  int32_t r3
000293c8  r0_1, r1_1, r2, r3 = data_20041894(arg1)
00023606  return sub_e0(r0_1, r1_1, r2, r3)
```

```
2004188c  void* data_2004188c = sub_21f22
20041890  void* data_20041890 = sub_21f40
20041894  void* data_20041894 = sub_220c0
20041898  void* data_20041898 = sub_22146
2004189c  void* data_2004189c = sub_22186
```



```
int32_t sub_2275a(int32_t arg1, void* arg2)
```

```
0002275a    int32_t var_18 = r3
00022760    char* r5 = *(arg2 + 4)
00022762    void* r6 = nullptr
00022766    void* r4 = nullptr
0002276e    data_200443c5 = (zx.d(*r5) << 0x1e u>> 0x1e).b
00022776    *r5 = (zx.d(*r5) << 0x1c u>> 0x1c).b
00022778    r5[1] = 0
0002277c    sub_293b8(0)
00022788    data_10002008 = data_10002008 & 0xffffffffbf
00022792    int32_t r2_2 = data_1000048c
00022796    uint32_t r0_5 = r2_2 << 0x10 u>> 0x18
0002279c    uint32_t r7 = zx.d(r2_2.b)
0002279e    uint32_t r1_2 = zx.d(data_200443c5)
```

```
000293b8    int32_t sub_293b8(int32_t arg1)
```

```
000293c8    int32_t r0_1
000293c8    int32_t r1_1
000293c8    int32_t r2
000293c8    int32_t r3
000293c8    r0_1, r1_1, r2, r3 = data_20041894(arg1)
00023606    return sub_e0(r0_1, r1_1, r2, r3)
```

```
000220c0    void sub_220c0(int32_t arg1)
```

```
000220ce    if (arg1 == 0)
000220ee        *0x40050018 = 1
000220fa        data_10002008 = data_10002008 | 0x40
00022102        data_10002004 = 0xeabe0001
0002210c        while (data_10002000 << 0x1d s>= 0)
0002210c            nop
00022110        return
000220d2    if (arg1 == 1)
00022112        *0x40050018 = 1
0002211e        data_10002008 = data_10002008 | 0x40
00022126        data_10002004 = 0xeabe0001
00022130        while (data_10002000 << 0x1d s>= 0)
00022130            nop
00022136        data_10002004 = 0xeabe0002
00022140        while (data_10002000 << 0x1c s>= 0)
00022140            nop
00022144        return
000220d6    if (arg1 == 2)
000220da        data_10002004 = 0xeabe0000
000220e4        *0x40050018 = 0
```

```
2004188c    void* data_2004188c = sub_21f22
20041890    void* data_20041890 = sub_21f40
20041894    void* data_20041894 = sub_220c0
20041898    void* data_20041898 = sub_22146
2004189c    void* data_2004189c = sub_22186
```

```
int32_t sub_2275a(int32_t arg1, void* arg2)
```

```
0002275a    int32_t var_18 = r3
00022760    char* r5 = *(arg2 + 4)
00022762    void* r6 = nullptr
00022766    void* r4 = nullptr
0002276e    data_200443c5 = (zx.d(*r5) << 0x1e u>> 0x1e).b
00022776    *r5 = (zx.d(*r5) << 0x1c u>> 0x1c).b
00022778    r5[1] = 0
0002277c    sub_293b8(0)
00022788    data_10002008 = data_10002008 & 0xffffffffbf
00022792    int32_t r2_2 = data_1000048c
00022796    uint32_t r0_5 = r2_2 << 0x10 u>> 0x18
0002279c    uint32_t r7 = zx.d(r2_2.b)
0002279e    uint32_t r1_2 = zx.d(data_200443c5)
```

```
000293b8    int32_t sub_293b8(int32_t arg1)
```

```
000293c8    int32_t r0_1
000293c8    int32_t r1_1
000293c8    int32_t r2
000293c8    int32_t r3
000293c8    r0_1, r1_1, r2, r3 = data_20041894(arg1)
00023606    return sub_e0(r0_1, r1_1, r2, r3)
```

```
000220c0    void sub_220c0(int32_t arg1)
```

```
000220ce    if (arg1 == 0)
000220ee        *0x40050018 = 1
000220fa        data_10002008 = data_10002008 | 0x40
00022102        data_10002004 = 0xeabe0001
0002210c        while (data_10002000 << 0x1d s>= 0)
0002210c            nop
00022110            return
000220d2    if (arg1 == 1)
00022112        *0x40050018 = 1
0002211e        data_10002008 = data_10002008 | 0x40
00022126        data_10002004 = 0xeabe0001
00022130        while (data_10002000 << 0x1d s>= 0)
00022130            nop
00022136        data_10002004 = 0xeabe0002
00022140        while (data_10002000 << 0x1c s>= 0)
00022140            nop
00022144            return
000220d6    if (arg1 == 2)
000220da        data_10002004 = 0xeabe0000
000220e4        *0x40050018 = 0
```

```
2004188c    void* data_2004188c = sub_21f22
20041890    void* data_20041890 = sub_21f40
20041894    void* data_20041894 = sub_220c0
20041898    void* data_20041898 = sub_22146
2004189c    void* data_2004189c = sub_22186
```

Can be in ROM or
RAM (if patched)


```
int32_t sub_2275a(int32_t arg1, void* arg2)
```

```
0002275a    int32_t var_18 = r3
00022760    char* r5 = *(arg2 + 4)
00022762    void* r6 = nullptr
00022766    void* r4 = nullptr
0002276e    data_200443c5 = (zx.d(*r5) << 0x1e u>> 0x1e).b
00022776    *r5 = (zx.d(*r5) << 0x1c u>> 0x1c).b
00022778    r5[1] = 0
0002277c    sub_293b8(0)
00022788    data_10002008 = data_10002008 & 0xffffffffbf
00022792    int32_t r2_2 = data_1000048c
00022796    uint32_t r0_5 = r2_2 << 0x10 u>> 0x18
0002279c    uint32_t r7 = zx.d(r2_2.b)
0002279e    uint32_t r1_2 = zx.d(data_200443c5)
```

```
000293b8    int32_t sub_293b8(int32_t arg1)
```

```
000293c8    int32_t r0_1
000293c8    int32_t r1_1
000293c8    int32_t r2
000293c8    int32_t r3
000293c8    r0_1, r1_1, r2, r3 = data_20041894(arg1)
00023606    return sub_e0(r0_1, r1_1, r2, r3)
```

```
000220c0    void sub_220c0(int32_t arg1)
```

```
000220ce    if (arg1 == 0)
000220ee        *0x40050018 = 1
000220fa        data_10002008 = data_10002008 | 0x40
00022102        data_10002004 = 0xeabe0001
0002210c        while (data_10002000 << 0x1d s>= 0)
0002210c            nop
00022110        return
000220d2    if (arg1 == 1)
00022112        *0x40050018 = 1
0002211e        data_10002008 = data_10002008 | 0x40
00022126        data_10002004 = 0xeabe0001
00022130        while (data_10002000 << 0x1d s>= 0)
00022130            nop
00022136        data_10002004 = 0xeabe0002
00022140        while (data_10002000 << 0x1c s>= 0)
00022140            nop
00022144        return
000220d6    if (arg1 == 2)
000220da        data_10002004 = 0xeabe0000
000220e4        *0x40050018 = 0
```

```
2004188c    void* data_2004188c = sub_21f22
20041890    void* data_20041890 = sub_21f40
20041894    void* data_20041894 = sub_220c0
20041898    void* data_20041898 = sub_22146
2004189c    void* data_2004189c = sub_22186
```

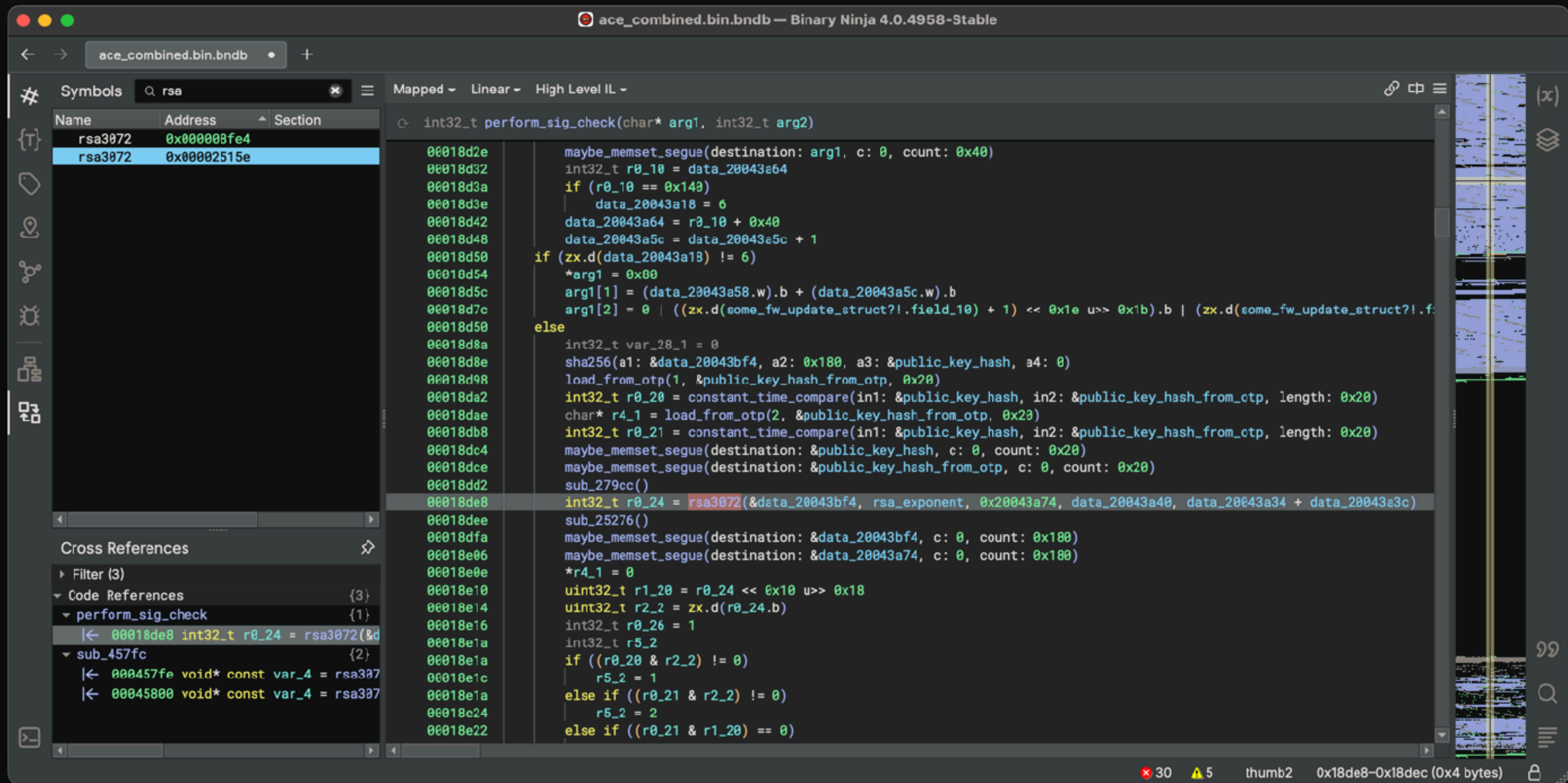




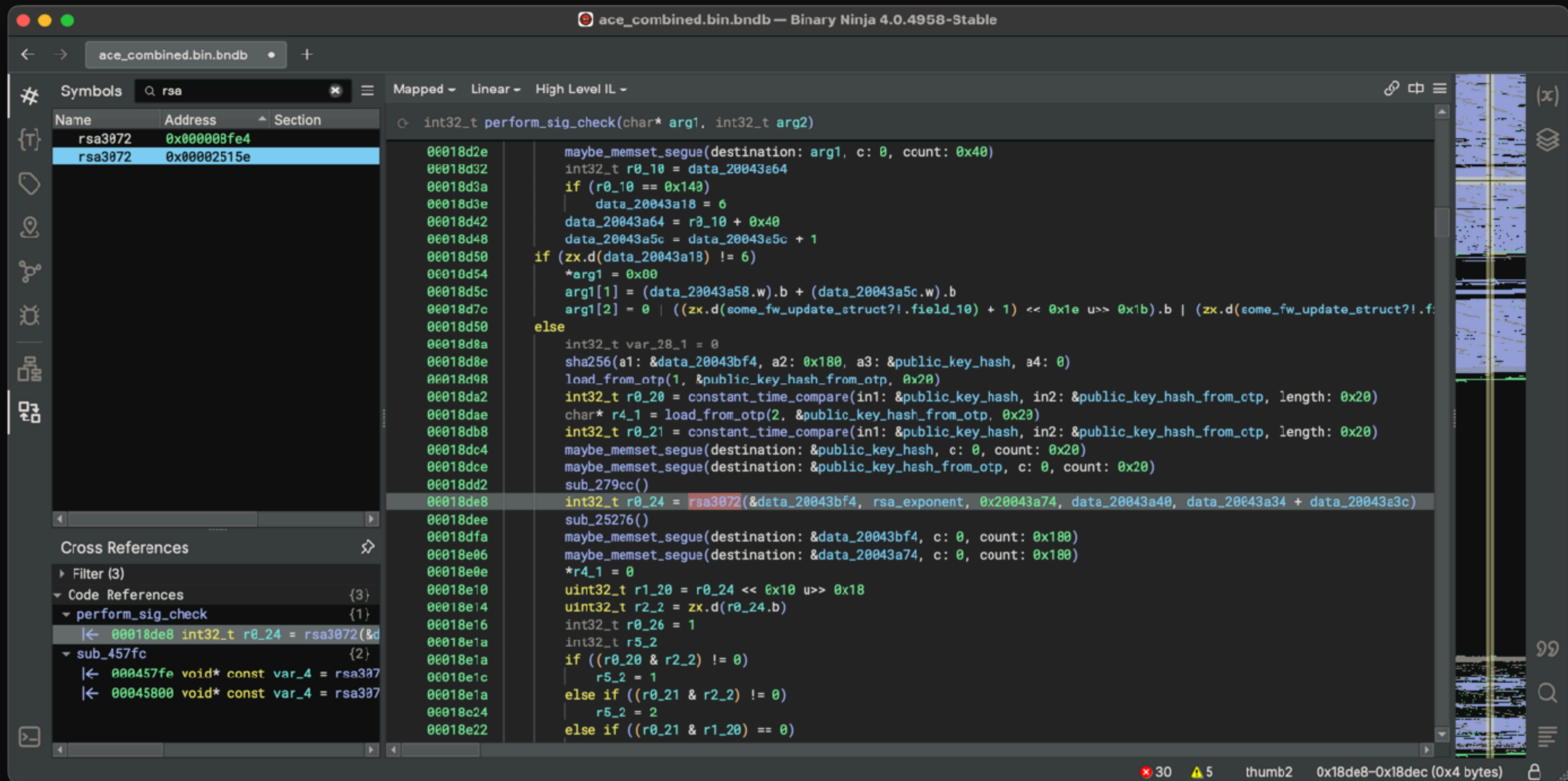
```
→ ~ sudo usbcfwflasher --verbose
2024-05-08 00:19:13.897 usbcfwflasher[66217:21870978]
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978]
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978] Hardware Present:
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978]
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978]             RID: 0
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978]             UUID: F21A3208-4151-1994-C34D-9FB099F8FB81
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978]             Name: USB-C_HPM,28
2024-05-08 00:19:13.899 usbcfwflasher[66217:21870978]             Version: 002.170.00.15
2024-05-08 00:19:13.912 usbcfwflasher[66217:21870978]             OTP Key Hash: 0F C3 8B 26
2024-05-08 00:19:13.912 usbcfwflasher[66217:21870978]
2024-05-08 00:19:13.912 usbcfwflasher[66217:21870978]
2024-05-08 00:19:13.916 usbcfwflasher[66217:21870978] Updates to be done:
2024-05-08 00:19:13.916 usbcfwflasher[66217:21870978] ---
2024-05-08 00:19:13.916 usbcfwflasher[66217:21870978] {
    0 = {
        RID = 0;
        options = {
        };
    };
}
```

```
→ ~ sudo usbcfwflasher --verbose
2024-05-08 00:19:13.897 usbcfwflasher[66217:21870978]
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978]
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978] Hardware Present:
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978]
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978]             RID: 0
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978]             UUID: F21A3208-4151-1994-C34D-9FB099F8FB81
2024-05-08 00:19:13.898 usbcfwflasher[66217:21870978]             Name: USB-C_HPM,28
2024-05-08 00:19:13.899 usbcfwflasher[66217:21870978]             Version: 002.170.00.15
2024-05-08 00:19:13.912 usbcfwflasher[66217:21870978]             OTP Key Hash: 0F C3 8B 26
2024-05-08 00:19:13.912 usbcfwflasher[66217:21870978]
2024-05-08 00:19:13.912 usbcfwflasher[66217:21870978]
2024-05-08 00:19:13.916 usbcfwflasher[66217:21870978] Updates to be done:
2024-05-08 00:19:13.916 usbcfwflasher[66217:21870978] ---
2024-05-08 00:19:13.916 usbcfwflasher[66217:21870978] {
    0 = {
        RID = 0;
        options = {
        };
    };
}
```

Updates are protected....



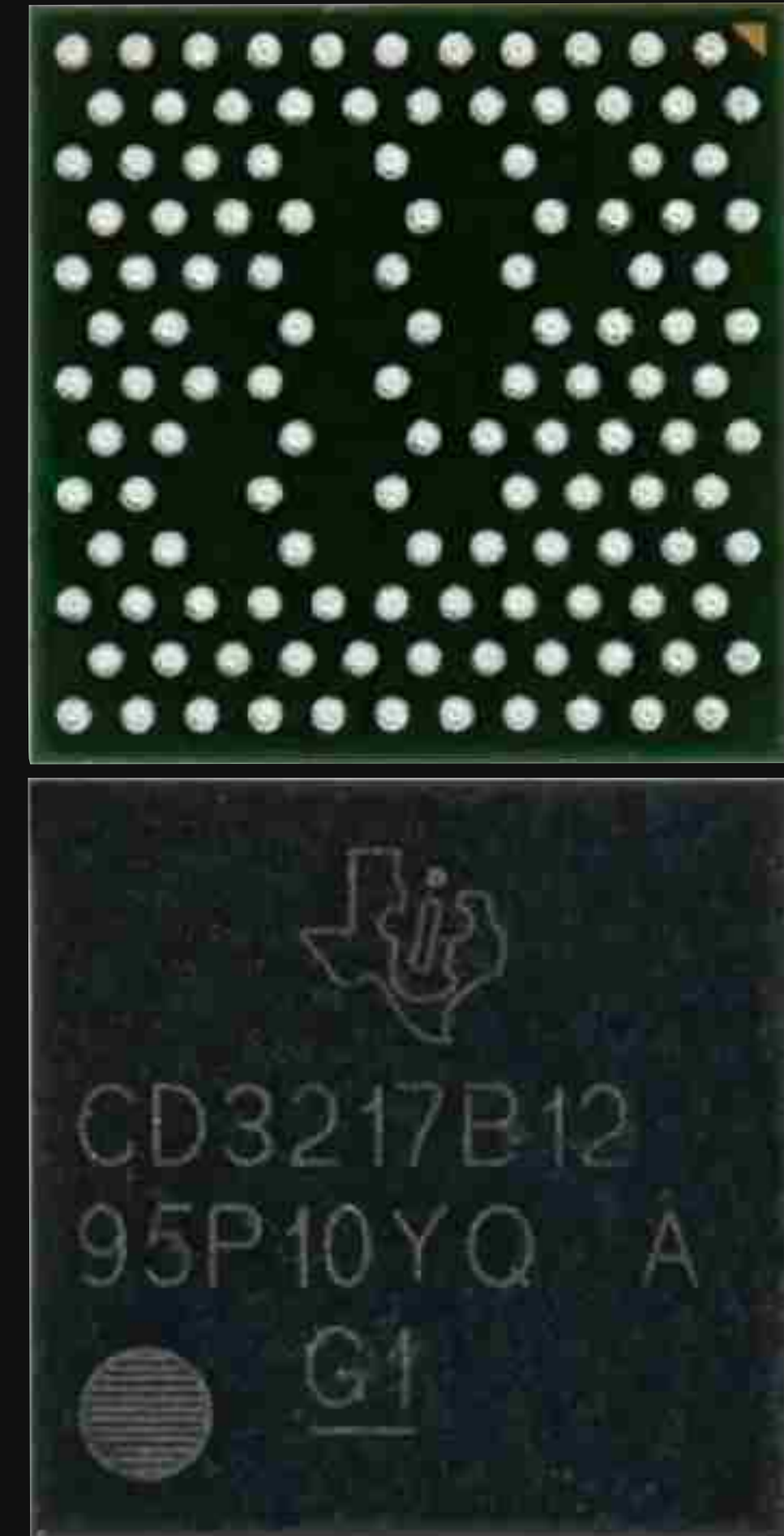
...with RSA3072



(But flash contents are not!)

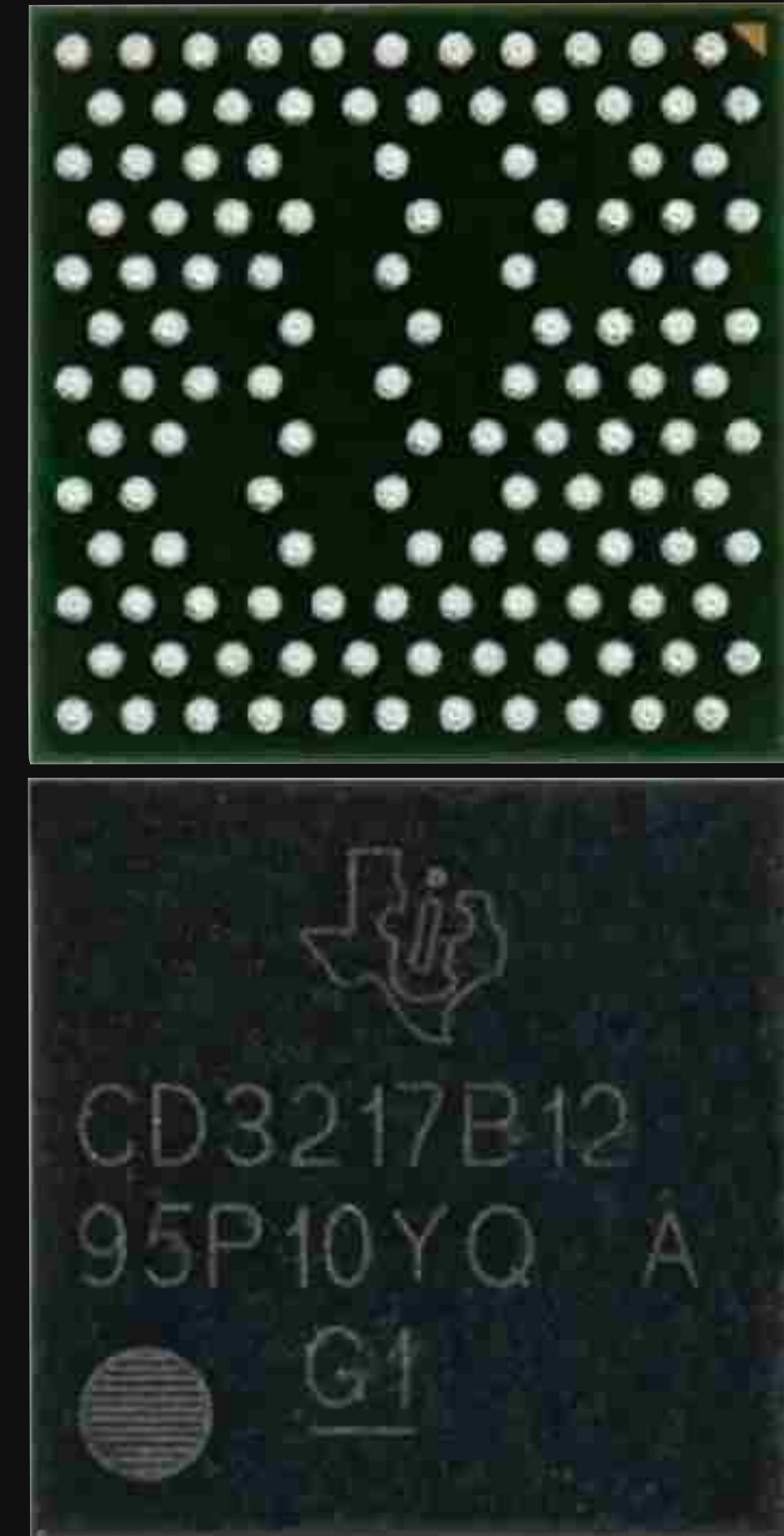
Still found an attack!

- Mix of software and hardware
- Abusable with root
- Survives full restores...



But...

- ACE2 is on its way out
- It doesn't do a whole lot
- iPhone 15 (Pro) uses successor



The ACE3

ACE3

- Texas Instruments SN25A12
- **Zero** public information
- Used in iPhone 15 and MacBook Pro M3 Pro & Max



ACE3

USB 3.1 Bus:

Host Controller Driver: AppleT8103USBXHCI

Apple Device (Port DFU Mode):

Product ID: 0xf014

Vendor ID: 0x05ac (Apple Inc.)

Version: 0.00

Serial Number: SDOM:01 CPID:0022 BDID:

Manufacturer: Apple Inc.

Location ID: 0x00100000

- Runs a full USB stack ("Port DFU")
- Has access to some internal busses
- Interesting potential...

ACE3

- Doesn't use usbcfwflasher (libAce3Updater.dylib)
- Different upgrade mechanism on iPhone 15 vs. MacBook Pro
- Updates are personalized 🥲



Sooooooooo... I ordered a MacBook M3 Max

... and my vuln doesn't work :(

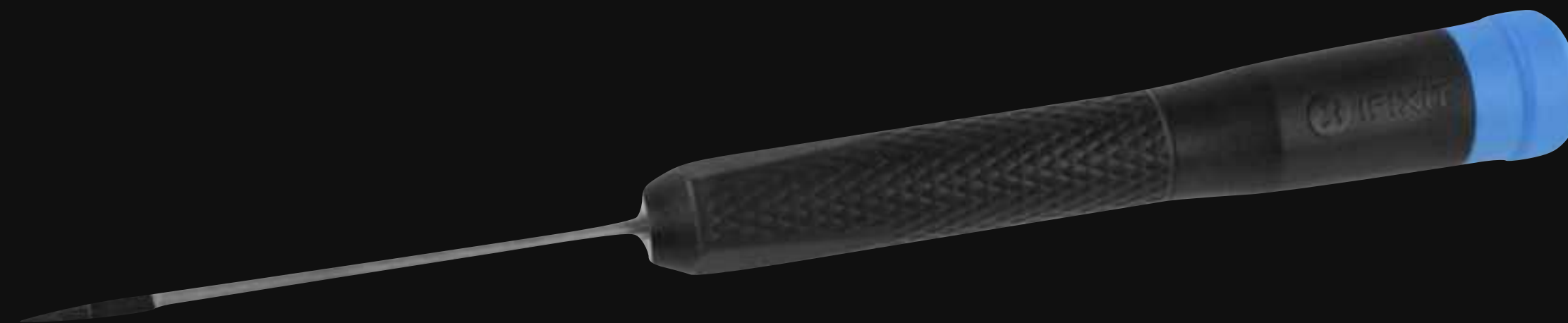
Send it back...

Send it back...

or...

Send it back...

or...



MacBook Pro

Downloaded by [Kainat K. Guliyeva] at 09:00 01 June 2015















ACE3

SPI Flash

ACE3



SPI Flash

ACE3

Debug
connectors





Debug port seems disabled 🥲

Debug port seems disabled 🥲
...so let's dump the flash!



M3Flashdump_Verified.bin.bndb — Binary Ninja 4.0.4958-Stable

←

→

M3Flashdump_...ied.bin.bndb

+

#

{T}

Cross References

Filter (0)

Mapped

Hex

00004000

03 00 e0 ac 00 40 20 00-00 28 00 00 40 00 00 00

.....@ ..(.@...

00004010

c0 a8 00 00 7f 02 00 00-7f ab 00 00 7f 07 00 00

.....

00004020

be b2 00 00 3d 9c 08 19-ff ff ff ff ff ff ff

....=.....

00004030

ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

.....

00004040

dc a7 00 00 00 00 72 00-f4 1e 05 20 f4 1e 05 20

.....r.....

00004050

68 1f 05 20 25 77 04 20-05 b1 0c 01 00 00 20 00

h.. %w.

00004060

7d 33 00 34 00 40 20 00-c0 a8 00 00 18 77 04 20

}3.4.@w.

00004070

00 00 00 00 00 00 00 00-00 00 00 7e 06 fc e8

.....~...

00004080

10 b5 40 21 26 48 08 f0-b3 ff 10 bd f0 b5 ff b0

..@!&H.....

00004090

ff b0 01 21 23 48 95 b0-c9 04 01 60 22 49 c9 7b

...!#H....."I.{

000040a0

09 07 02 d4 01 21 09 05-01 60 20 4c 00 20 a0 63

.....!...` L. .c

000040b0

5b 21 0e 20 08 f0 a2 ff-1c 48 80 38 c0 6a 1c 49

[!H.8.j.I

000040c0

08 40 03 21 c9 03 88 42-03 d1 e0 6b 10 21 08 43

.@.!....B...k!.C

000040d0

e0 63 18 48 41 6a 08 20-01 43 16 48 24 30 08 f0

.c.HAj. .C.H\$0..

000040e0

93 ff 01 ae 14 4a 15 49-30 46 08 f0 93 ff 14 4f

.....J.I0F.....0

000040f0

00 24 06 20 25 46 45 43-a9 19 88 1c 08 f0 90 ff

.\$. %FEC.....

00004100

00 90 70 5b 04 22 c0 19-69 46 08 f0 8f ff 64 1c

..p[."...iF....d.

00004110

b6 2c ee d3 ff f7 b4 ff-7f b0 7f b0 15 b0 f0 bd

.....

00004120

6a 01 04 20 80 e1 00 e0-54 23 04 20 c0 00 09 40

j..T#. ...@

00004130

80 80 01 00 00 41 09 40-44 04 00 00 20 1a 05 20

.....A.@D... ..

00004140

a8 09 04 20 10 b5 08 f0-77 ff 13 4c 20 78 40 09

...w..L x@.

00004150

01 28 20 d1 a0 78 c0 06-13 d4 10 48 00 79 40 07

.(..x.....H.y@.

00004160

0f d4 c5 21 0e 20 08 f0-49 ff 0d 48 40 68 40 06

...! . ..I..H@h@.

00004170

c0 0f 06 d0 a0 78 10 21-08 43 a0 70 c8 05 09 49

.....x.!.C.p...I

00004180

08 63 09 48 00 78 c0 07-05 d0 08 48 80 7e 01 28

.c.H.x.....H.~.(

00004190

01 d1 08 f0 37 fa 10 bd-54 23 04 20 24 23 04 20

....7...T#. \$#.

000041a0

00 00 06 40 c0 01 09 40-f8 1e 05 20 b4 30 04 20

...@...@.... .0.

000041b0

ff b5 85 b0 0c 46 05 00-1f d0 00 20 07 99 ce 07

.....F.....

000041c0

89 07 c9 0f 03 91 07 99-f6 0f c9 09 b0 43 88 43

.....C.C

000041d0

02 90 08 f0 37 ff 02 46-01 20 00 90 01 2c 0e d0

....7..F.,

000041e0

02 2c 0c d0 00 20 00 2e-2d d0 00 21 00 28 0e d0

.,... ..-...!.(

000041f0

01 2c 06 d0 02 2c 07 d0-08 e0 01 20 de e7 01 20

.,...,.....

00004200

f1 e7 01 2a 06 d0 01 e0-02 2a 03 d0 00 21 13 2d

...*.....*...!.-

00004210

02 d0 18 e0 01 21 fa e7-fe 4a 12 7f 12 07 03 d4

.....!...J.....

00004220

fd 4a 12 78 00 2a 59 d1-00 28 0c d0 06 22 51 43

.J.x.*Y..(.."qc

00004230

f8 4a a0 3a 89 18 c0 31-09 78 03 29 03 d1 f7 49

.J.:...1.x.)...I

00004240

09 78 00 29 4a d1 f6 4f-00 28 0d d0 b8 79 01 07

.x.)J..0.(...y..

00004250

80 07 89 0f 80 0f 08 f0-fb fe 21 46 28 46 08 f0

.....!F(F..

00004260

fd fe 01 21 48 40 00 90-38 79 41 07 28 d5 80 07

...!H@..ByA.(...

00004270

80 0f 01 28 24 d9 01 2d-05 d0 02 2d 03 d0 03 2d

...(\$..-...-...-

30

21

thumb2

0x422d

nextree.io

M3Flashdump_Verified.bin.bndb — Binary Ninja 4.0.4958-Stable

M3Flashdump_...led.bin.bndb +

Types Search types Mapped Linear High Level IL

User Types: M3Flashdump_Verif...

Name	Size
ace3_binary_header	0x40
ace3_firmware_header	0x28
struct_1	0x0

System Types: M3Flashdump_Verif...

Platform: thumb2

struct ace3_binary_header __packed

```
{
00  uint32_t binary_size;
04  uint32_t u[0x7];
20  uint32_t binary_crc;
24  uint32_t version;
28  uint32_t boot_config_offset;
2c  uint32_t u2[0x4];
3c  uint32_t header_crc;
40 };
```

Cross References

Filter (1)

Data References {1}

00004040 struct ace3_binary_header d

0x4000

```
00004000 struct ace3_firmware_header data_4000 =
00004000 {
00004000     uint32_t ace_id = 0xace00003
00004004     uint32_t ace_version = 0x204000
00004008     uint32_t unknown1 = 0x2800
0000400c     uint32_t ace_binary_start_relative = 0x40
00004010     uint32_t boot_config = 0xa8c0
00004014     uint32_t boot_config_size = 0x27f
00004018     uint32_t im4m_offset = 0xab7f
0000401c     uint32_t im4m_size = 0x77f
00004020     uint32_t ace_binary_size = 0xb2be
00004024     uint32_t ace_binary_crc = 0x19089c3d
00004028 }

00004028 ff ff ff ff ff ff ff ff .....
00004030 ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff .....

00004040 struct ace3_binary_header data_4040 =
00004040 {
00004040     uint32_t binary_size = 0xa7dc
00004044     uint32_t u[0x7] =
00004044     {
00004044         [0x0] = 0x00720000
00004048         [0x1] = 0x20051ef4
0000404c         [0x2] = 0x20051ef4
00004050         [0x3] = 0x20051f68
00004054         [0x4] = 0x20047725
00004058         [0x5] = 0x010cb105
0000405c         [0x6] = 0x00200000
00004060     }
00004060     uint32_t binary_crc = 0x3400337d
00004064     uint32_t version = 0x204000
00004068     uint32_t boot_config_offset = 0xa8c0
0000406c     uint32_t u2[0x4] =
0000406c     {
0000406c         [0x0] = 0x20047718
00004070         [0x1] = 0x00000000
00004074         [0x2] = 0x00000000
00004078         [0x3] = 0x00000000
0000407c     }
0000407c     uint32_t header_crc = 0xe8fc067e
00004080 }
```

30 12 thumb2 0x40c8-0x40ca (0x2 bytes)

M3Flashdump_Verified.bin.bndb — Binary Ninja 4.0.4958-Stable

M3Flashdump_...led.bin.bndb +

Types Search types Mapped Linear High Level IL

User Types: M3Flashdump_Verif...

Name	Size
ace3_binary_header	0x40
ace3_firmware_header	0x28
struct_1	0x0

System Types: M3Flashdump_Verif...

Platform: thumb2

struct ace3_binary_header __packed

```
00 uint32_t binary_size;
04 uint32_t u[0x7];
20 uint32_t binary_crc;
24 uint32_t version;
28 uint32_t boot_config_offset;
2c uint32_t u2[0x4];
3c uint32_t header_crc;
40 };
```

Cross References

Filter (1)

Data References {1}

00004040 struct ace3_binary_header d

0x4000

```
00004000 struct ace3_firmware_header data_4000 =
00004000 {
00004000     uint32_t ace_id = 0xace00003
00004004     uint32_t ace_version = 0x204000
00004008     uint32_t unknown1 = 0x2800
0000400c     uint32_t ace_binary_start_relative = 0x40
00004010     uint32_t boot_config = 0xa8c0
00004014     uint32_t boot_config_size = 0x27f
00004018     uint32_t im4m_offset = 0xab7f
0000401c     uint32_t im4m_size = 0x77f
00004020     uint32_t ace_binary_size = 0xb2be
00004024     uint32_t ace_binary_crc = 0x19089c3d
00004028 }

00004028 ff ff ff ff ff ff ff ff .....
00004030 ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff .....

00004040 struct ace3_binary_header data_4040 =
00004040 {
00004040     uint32_t binary_size = 0xa7dc
00004044     uint32_t u[0x7] =
00004044     {
00004044         [0x0] = 0x00720000
00004048         [0x1] = 0x20051ef4
0000404c         [0x2] = 0x20051ef4
00004050         [0x3] = 0x20051f68
00004054         [0x4] = 0x20047725
00004058         [0x5] = 0x010cb105
0000405c         [0x6] = 0x00200000
00004060     }
00004060     uint32_t binary_crc = 0x3400337d
00004064     uint32_t version = 0x204000
00004068     uint32_t boot_config_offset = 0xa8c0
0000406c     uint32_t u2[0x4] =
0000406c     {
0000406c         [0x0] = 0x20047718
00004070         [0x1] = 0x00000000
00004074         [0x2] = 0x00000000
00004078         [0x3] = 0x00000000
0000407c     }
0000407c     uint32_t header_crc = 0xe8fc067e
00004080 }
```

30 12 thumb2 0x40c8-0x40ca (0x2 bytes)

Can't get it to boot a modified firmware...

Either I'm bad at reversing ...

Either I'm bad at reversing ...
... or they are good at engineering

Either I'm bad at reversing ...
... or they are good at engineering
(or both)

What I tried...

- Software vulnerability
- Physical SWD access
- Modifying & switching flash contents
- Fuzzing
- ...



Sooooo

- Completely documented chip
- No firmware
- Only some simple commands



Sooooo

- Completely documented chip
- No firmware
- Only some simple commands



Time to give up?



Fault Injection

Fault Injection...

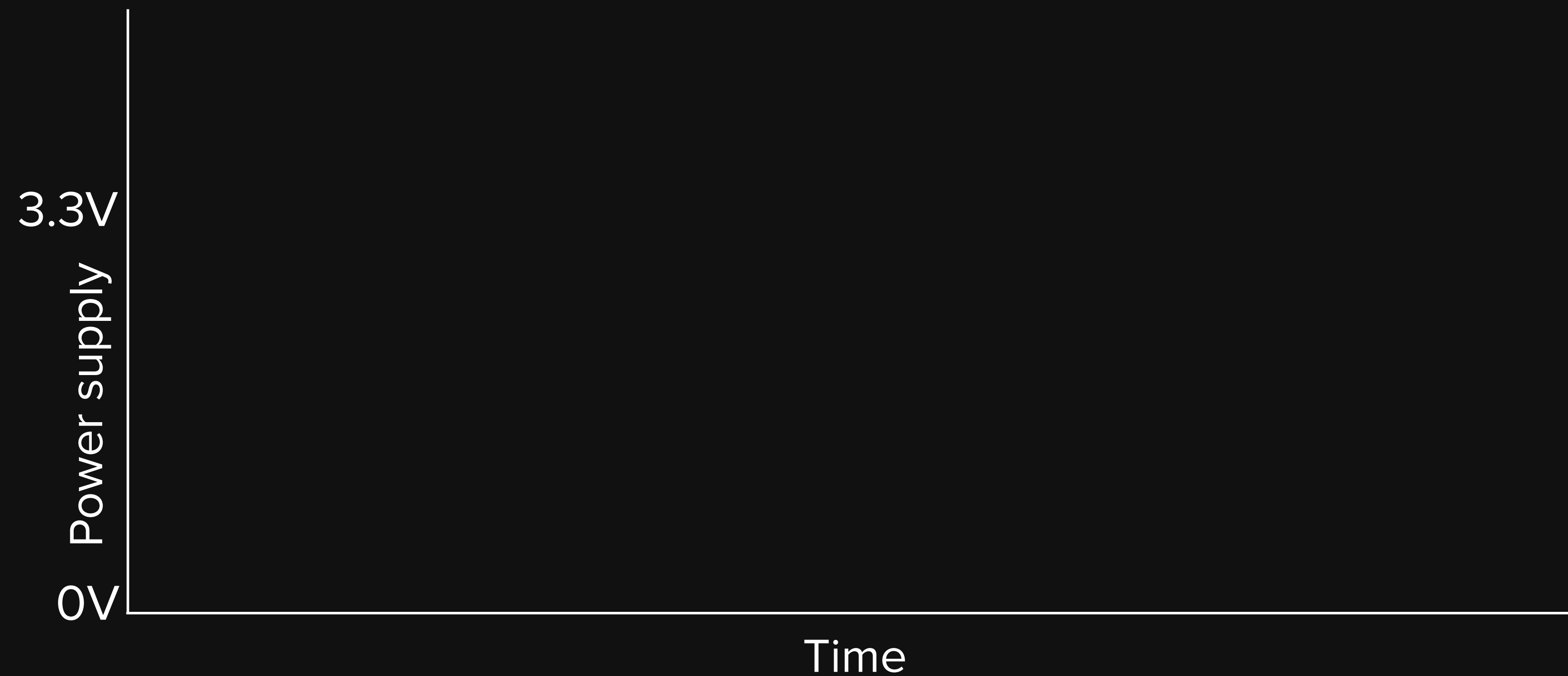
- Introduce **faults** into the chip
- Allows to modify the behavior of the running software
- Voltage, Laser, BBI, Electro-Magnetic...

Fault Injection...

- Introduce **faults** into the chip
- Allows to modify the behavior of the running software
- Voltage, Laser, BBI, Electro-Magnetic...

Often bricks chips...

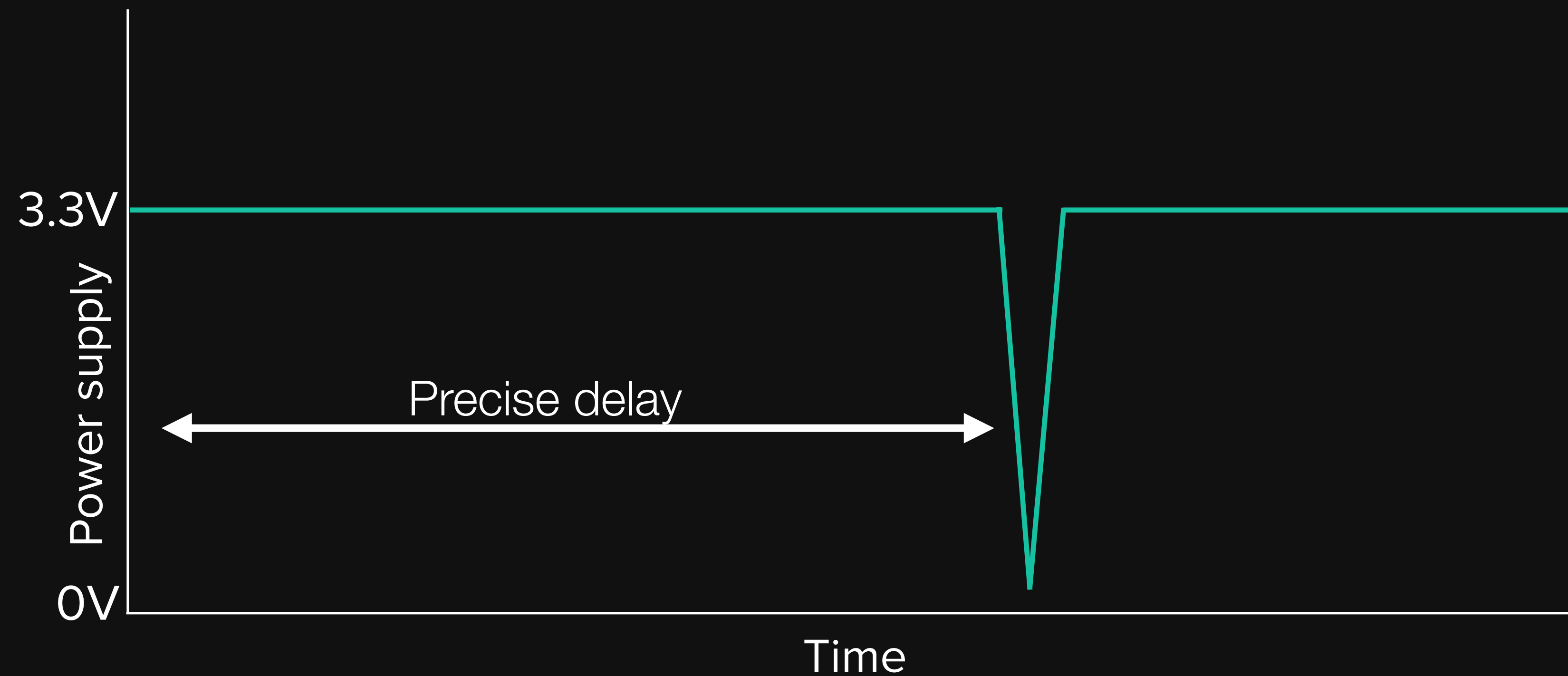
Voltage Fault Injection



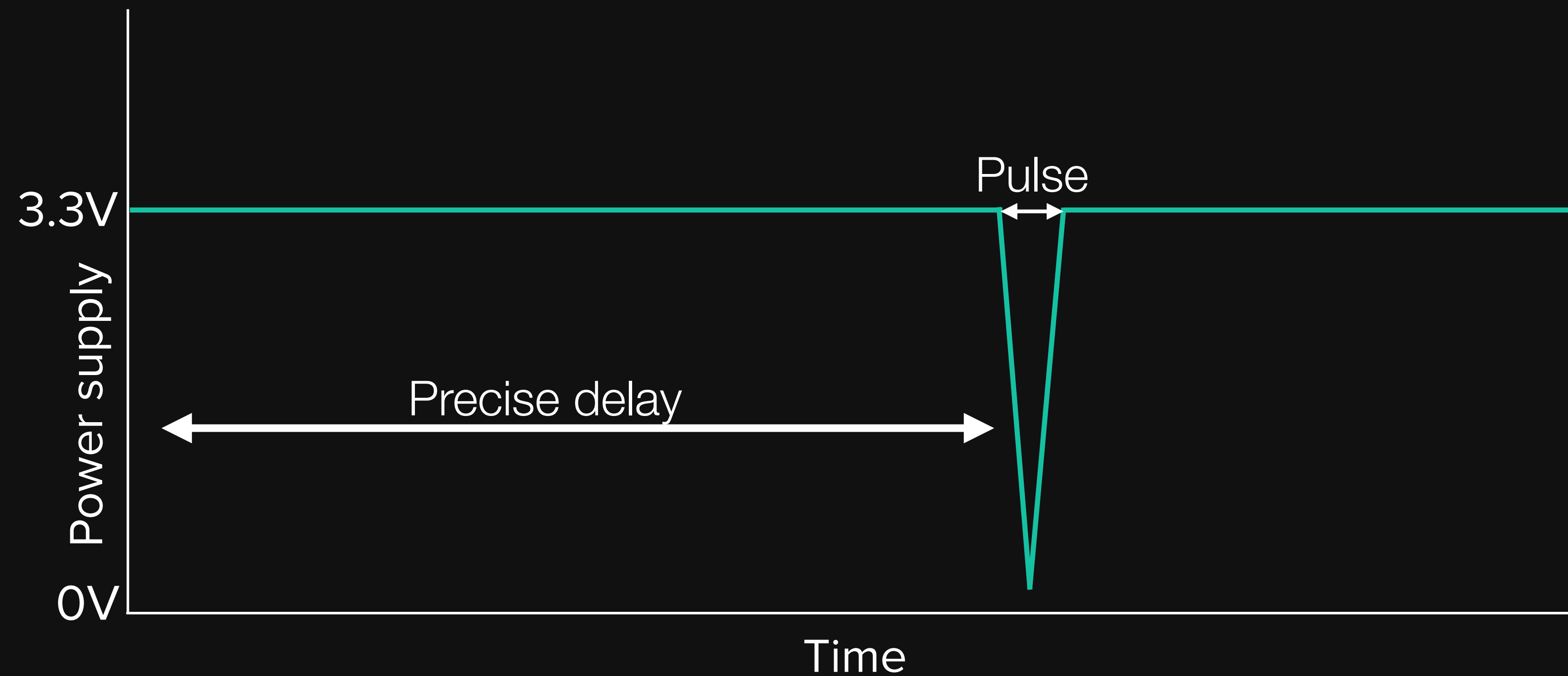
Voltage Fault Injection



Voltage Fault Injection



Voltage Fault Injection



Voltage Fault Injection

- Pretty much requires soldering
 - Removal of capacitors
 - Best performed on potential bypass capacitors
 - More difficult with shared voltage rails

EMFI

Electro-Magnetic Fault-Injection

Electro-Magnetic Fault-Injection

- Create high-voltage pulse into a coil
- This lets us inject current into a very precise location on the chip
- Skip instructions, change register values, etcpp

Electro-Magnetic Fault-Injection

- Create high-voltage pulse into a coil
- This lets us inject current into a very precise location on the chip
- Skip instructions, change register values, etcpp

No target prep necessary!

EMFI



EMFI



EMFI



EMFI



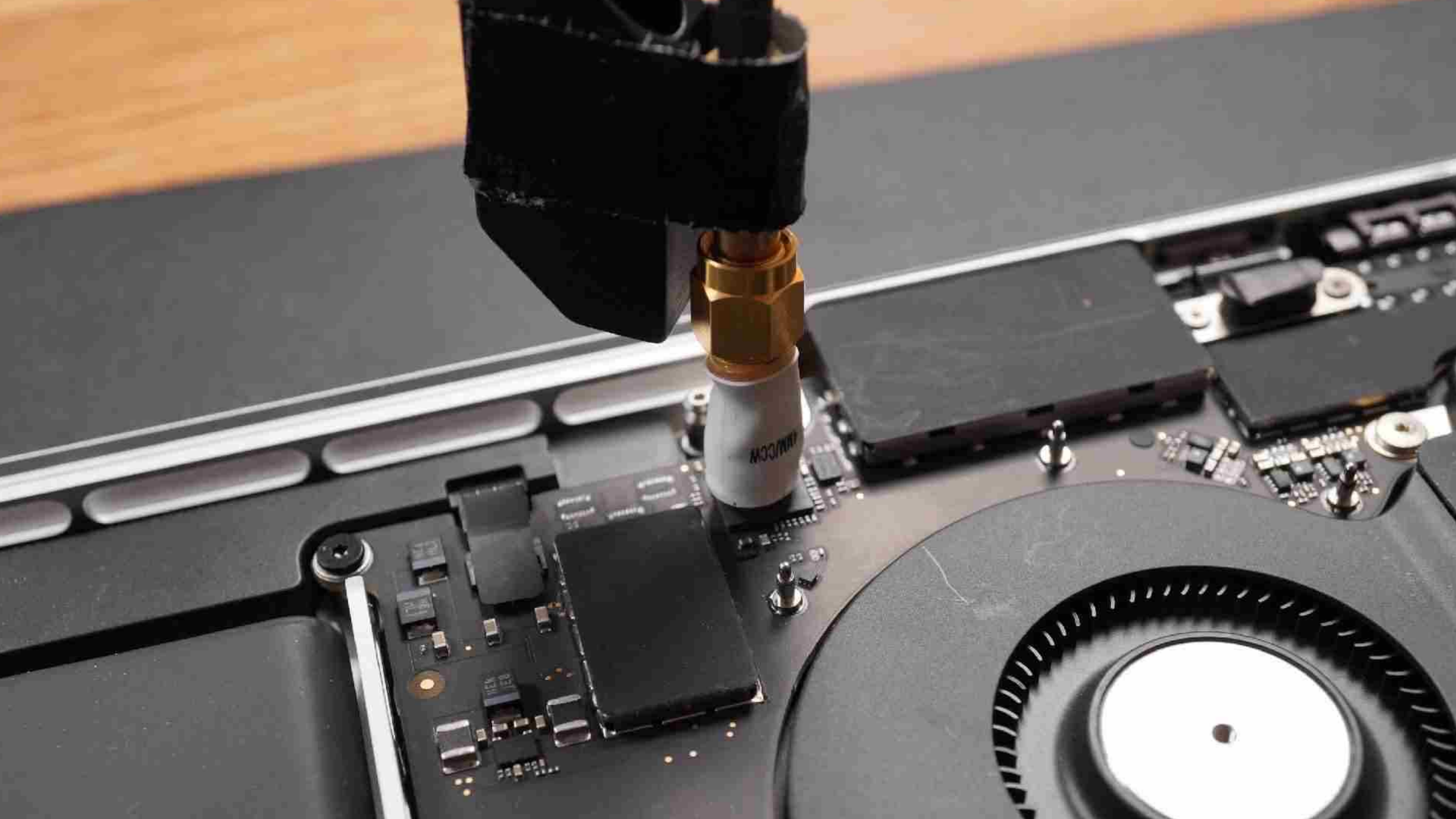
Chip
Hacked!

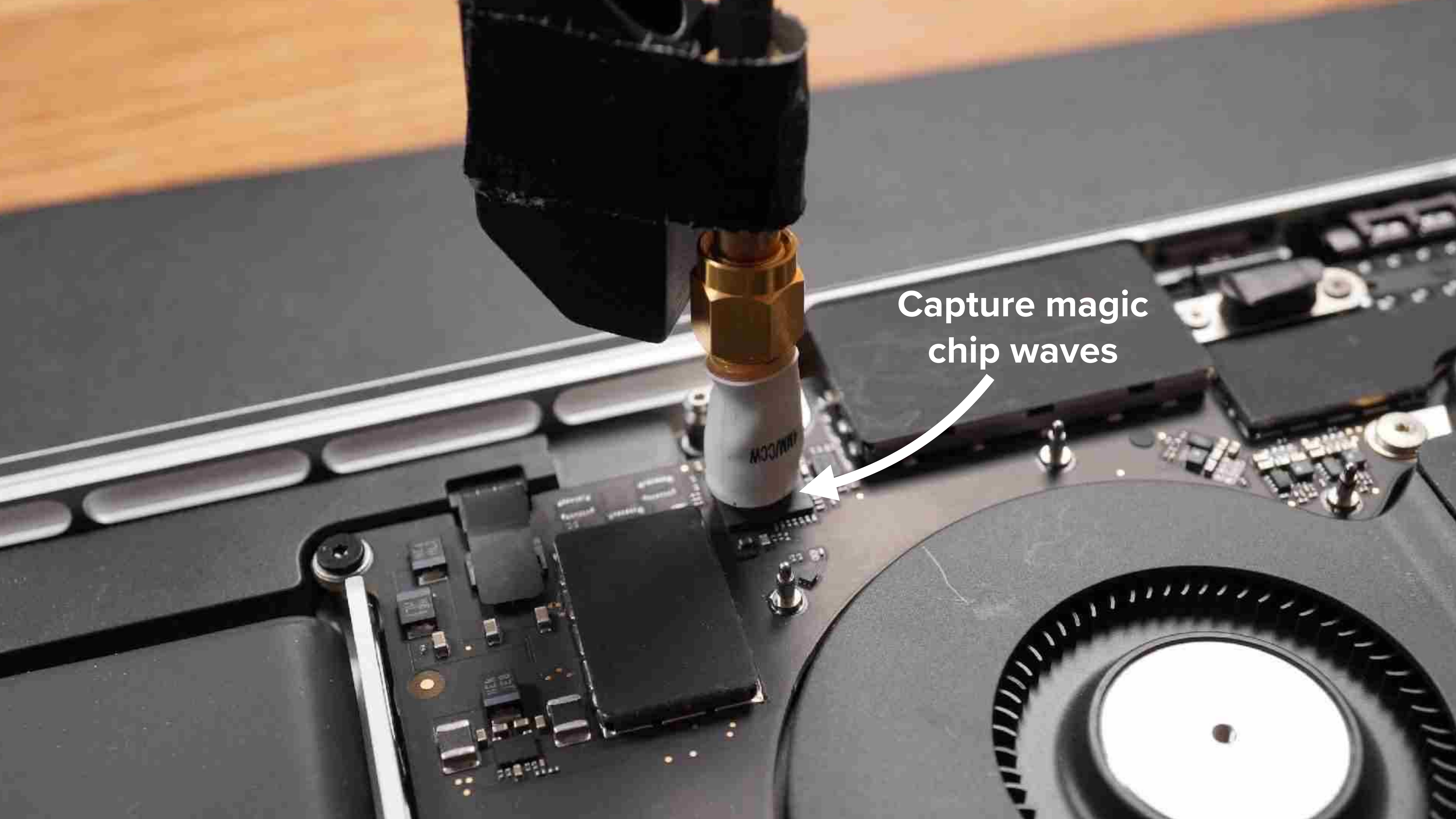
But we need precise timing on when
to inject our glitch...



Side Channels

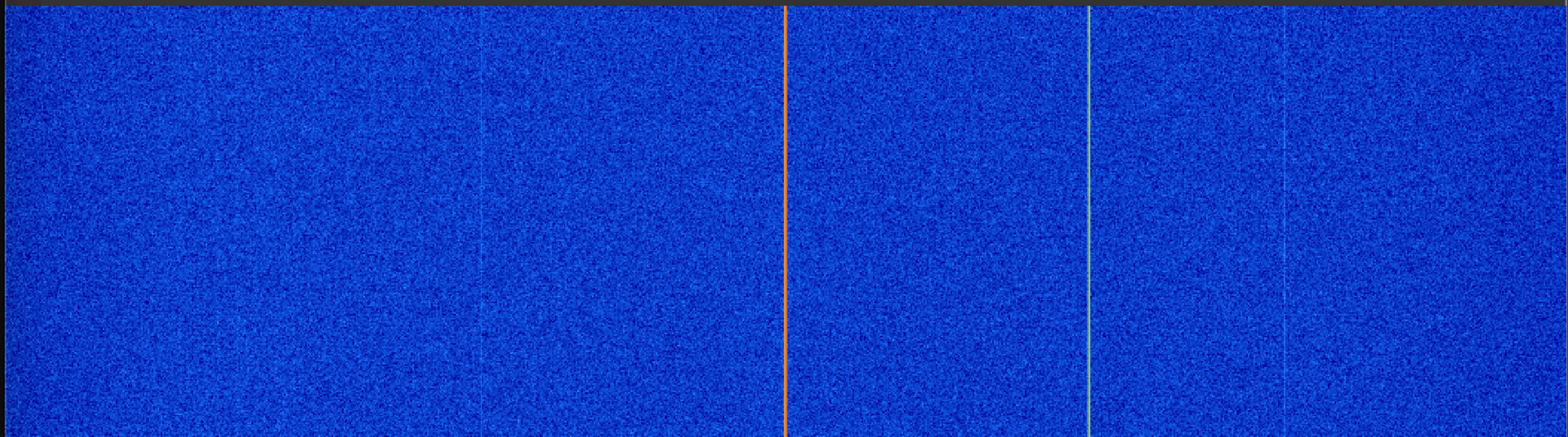
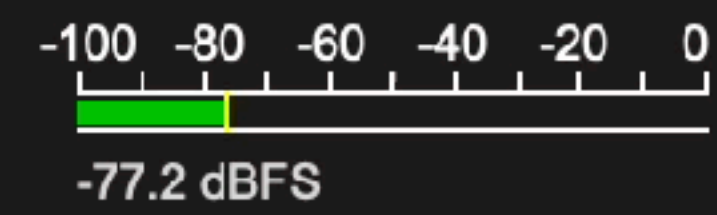






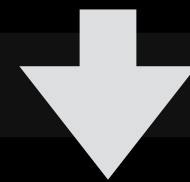
Capture magic
chip waves

0 050.445.000



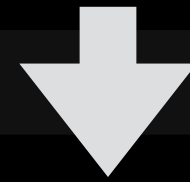
Start recording
spectrum on HackRF

Start recording
spectrum on HackRF

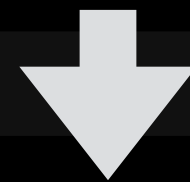


Reboot ACE3 via
acetooll3

Start recording
spectrum on HackRF



Reboot ACE3 via
acetoool3



Get EM recording

← →

m3 pro notes

radiotest copy.ipynb

radiotest.ipynb

radiotest copy 4.ipynb

radiotest copy 5.ipynb

radiotest copy 3.ipynb

radiotest copy 5.ipynb > ..

code

+ Markdown

▶ Run All

↺ Restart

≡ Clear All Outputs

| 📄 Variables

≡ Outline

⋮

glitching2 (Python 3.11)

import matplotlib.pyplot as plt

filename = "test_reboot2"

def waterfall_file(filename):

data = np.fromfile(filename, dtype=np.int8)

data_complex = data[:,2] + 1j * data[1:,2]

Reshape data for the waterfall plot

num_samples_per_row = 1024 # This value can be adjusted

num_rows = len(data_complex) // num_samples_per_row

data_resaped = data_complex[:num_rows * num_samples_per_row].reshape((num_rows, num_samples_per_row))

Compute FFT

fft_data = np.fft.fftshift(np.fft.fft(data_resaped, axis=1), axes=1)

magnitude = 20 * np.log10(np.abs(fft_data))

Frequency range for the x-axis

sampling_rate = 10e6 # 10 MHz

frequency_start = -sampling_rate / 2 # -5 MHz

frequency_end = sampling_rate / 2 # 5 MHz

center_frequency = 44366500 # 44.3665 MHz

frequencies = np.linspace(frequency_start, frequency_end, num_samples_per_row) + center_frequency

Plotting

plt.imshow(magnitude, aspect='auto', extent=[frequencies[0], frequencies[-1], 0, num_rows], cmap='viridis')

plt.colorbar(label='Magnitude (dB)')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Time Index')

plt.title(filename)

plt.show()

def waterfall_file16(filename):

data = np.fromfile(filename, dtype=np.int16)

data_complex = data[:,2] + 1j * data[1:,2]

Reshape data for the waterfall plot

num_samples_per_row = 1024 # This value can be adjusted

num_rows = len(data_complex) // num_samples_per_row

data_resaped = data_complex[:num_rows * num_samples_per_row].reshape((num_rows, num_samples_per_row))

Compute FFT

fft_data = np.fft.fftshift(np.fft.fft(data_resaped, axis=1), axes=1)

magnitude = 20 * np.log10(np.abs(fft_data))

Frequency range for the x-axis

sampling_rate = 10e6 # 10 MHz

frequency_start = -sampling_rate / 2 # -5 MHz

frequency_end = sampling_rate / 2 # 5 MHz

center_frequency = 44366500 # 44.3665 MHz

frequencies = np.linspace(frequency_start, frequency_end, num_samples_per_row) + center_frequency

Plotting

plt.imshow(magnitude, aspect='auto', extent=[frequencies[0], frequencies[-1], 0, num_rows], cmap='viridis')

plt.colorbar(label='Magnitude (dB)')

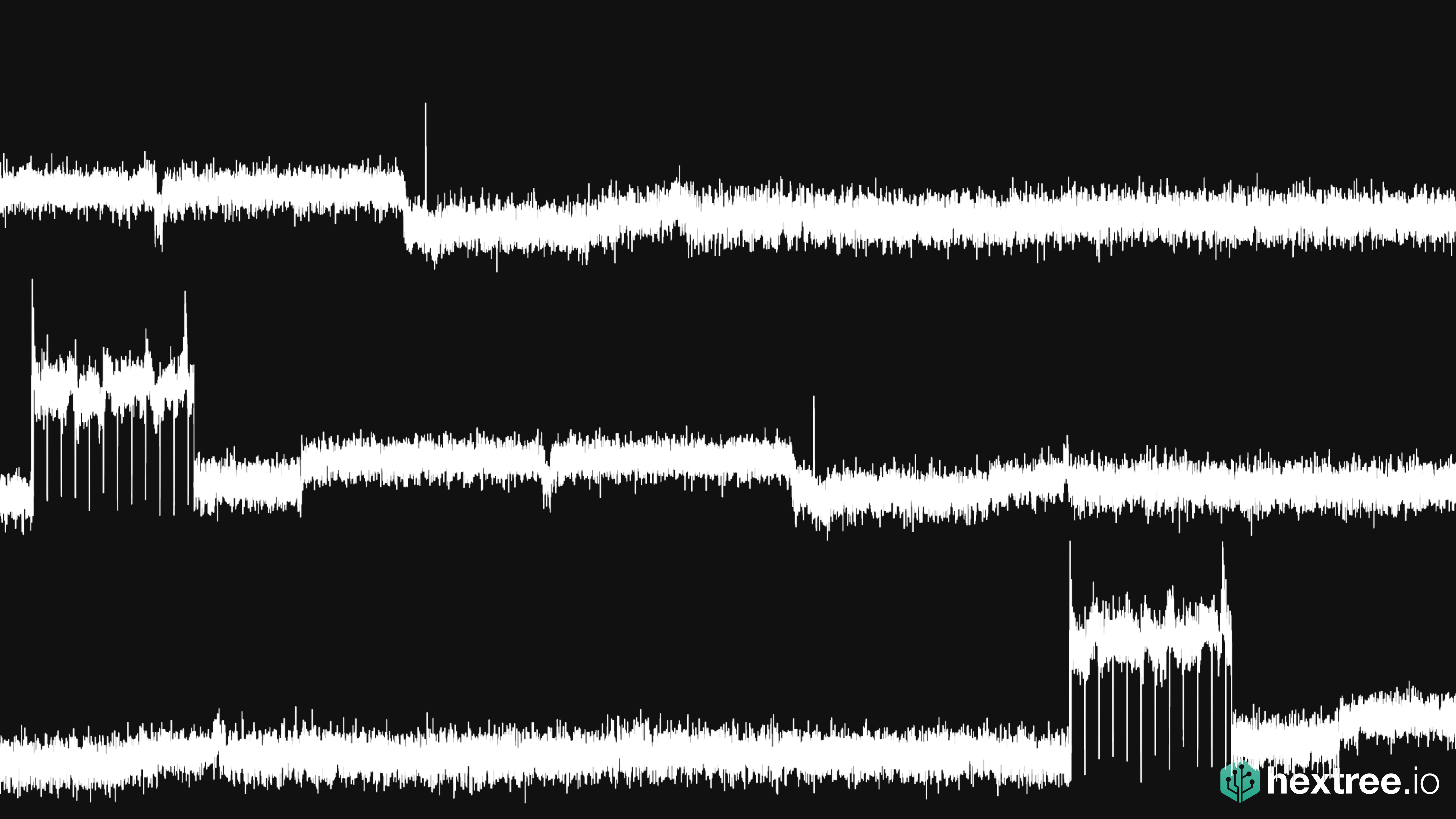
plt.xlabel('Frequency (Hz)')

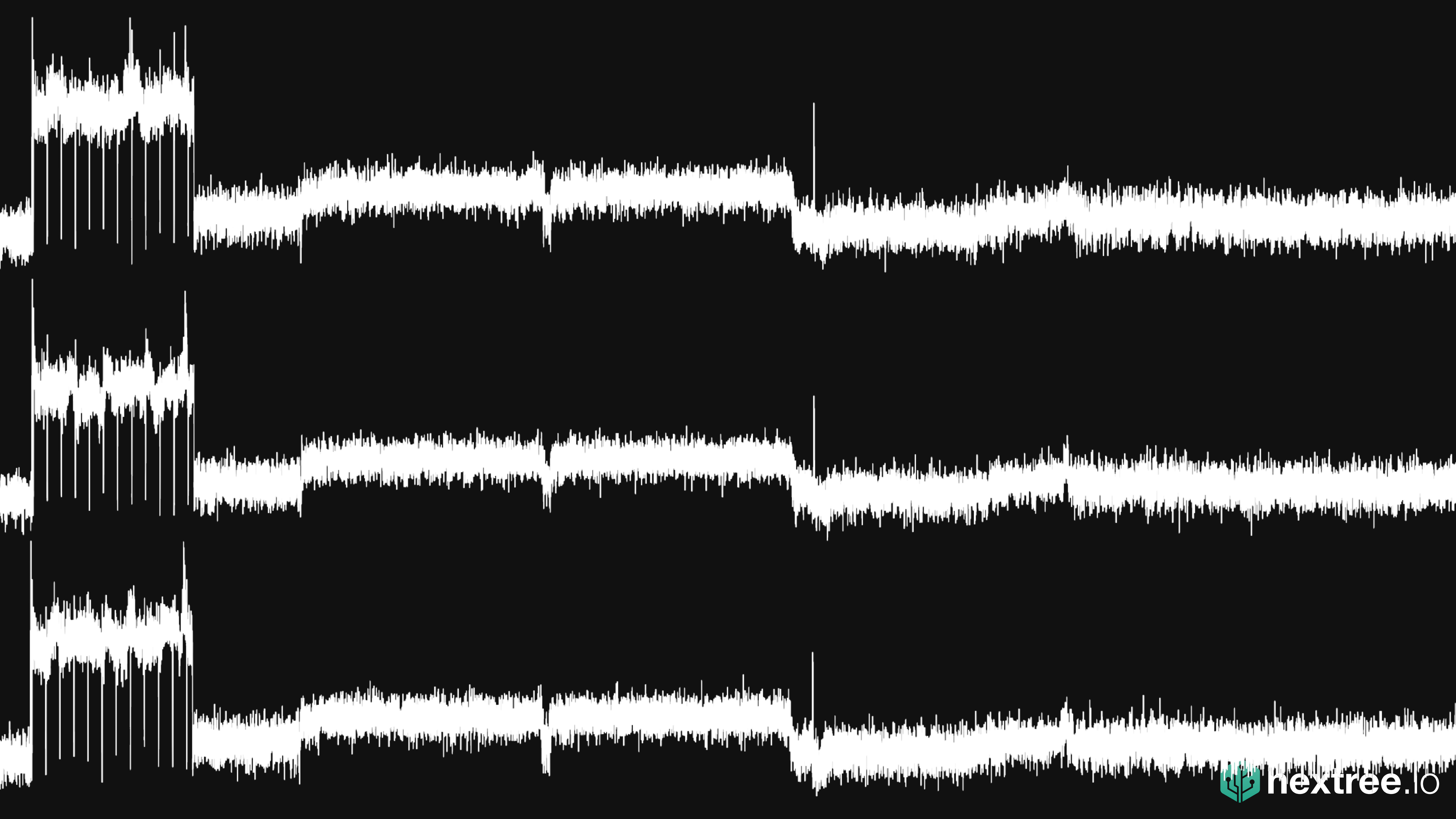
plt.ylabel('Time Index')

plt.title(filename)

plt.show()







Private

<

>

1

hackrf.readthedocs.io

+

HACKRF ONE HARDWARE

HackRF One

Minimum Host System Requirements for HackRF

List of Hardware Revisions

Hardware Components

LEDs

Buttons

Connectors

External Clock Interface (CLKIN and CLKOUT)

Expansion Interface

Hardware Triggering

Clock Synchronization

Requirements

Open Your HackRF One

Identify the Trigger Pins

Connect the Trigger Output to the Trigger Input

Usage

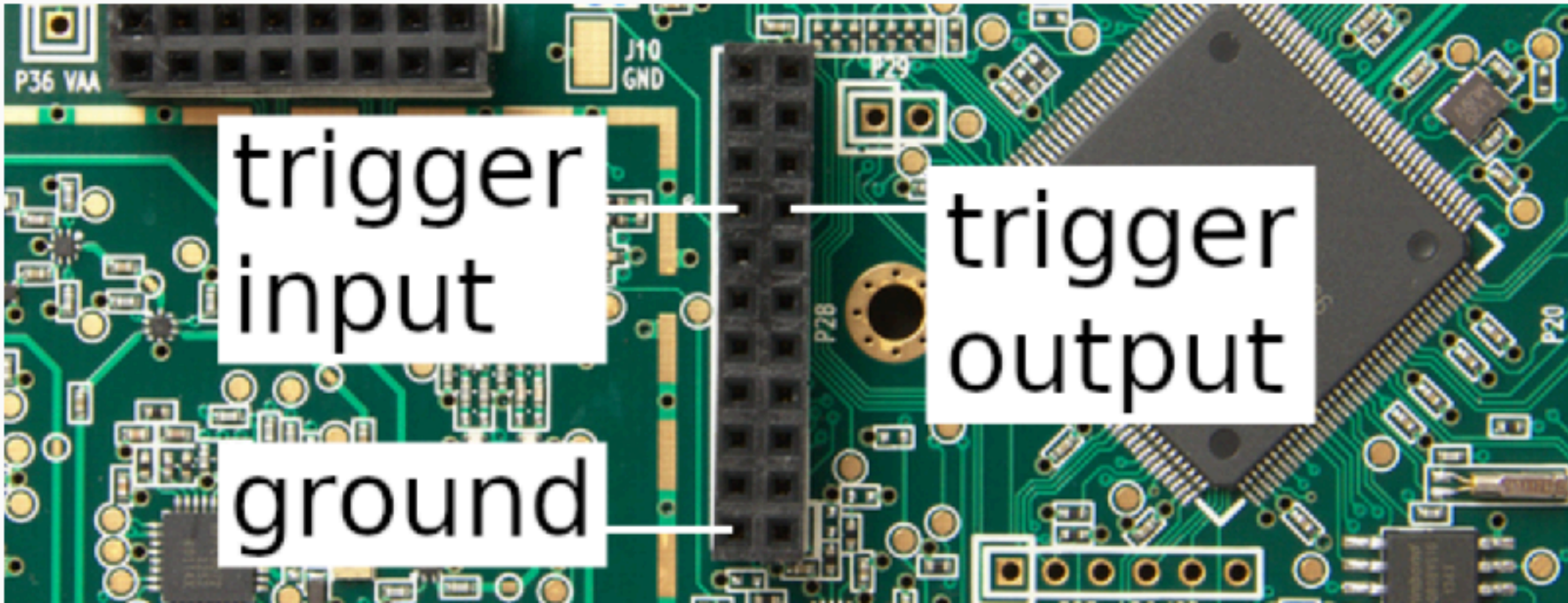
Additional Devices

Read the Docs

v: latest

Identify the Trigger Pins

HackRF One has four normally-populated pin headers, three of which are arranged in a 'C' shape. On the circuit board these are marked P28, P22, and P20. P28 is the header nearest to the center of the board. Locate pins 15 (trigger output) and 16 (trigger input) on header P28.




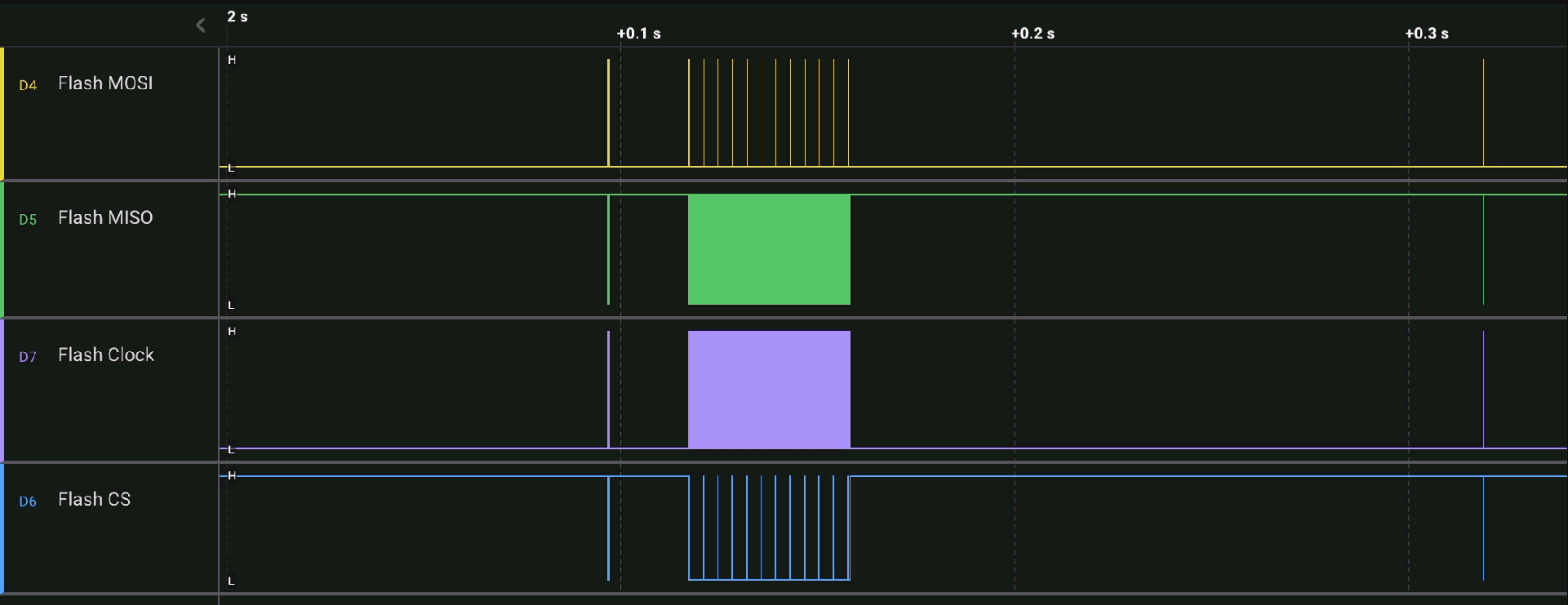
Connect the Trigger Output to the Trigger Input

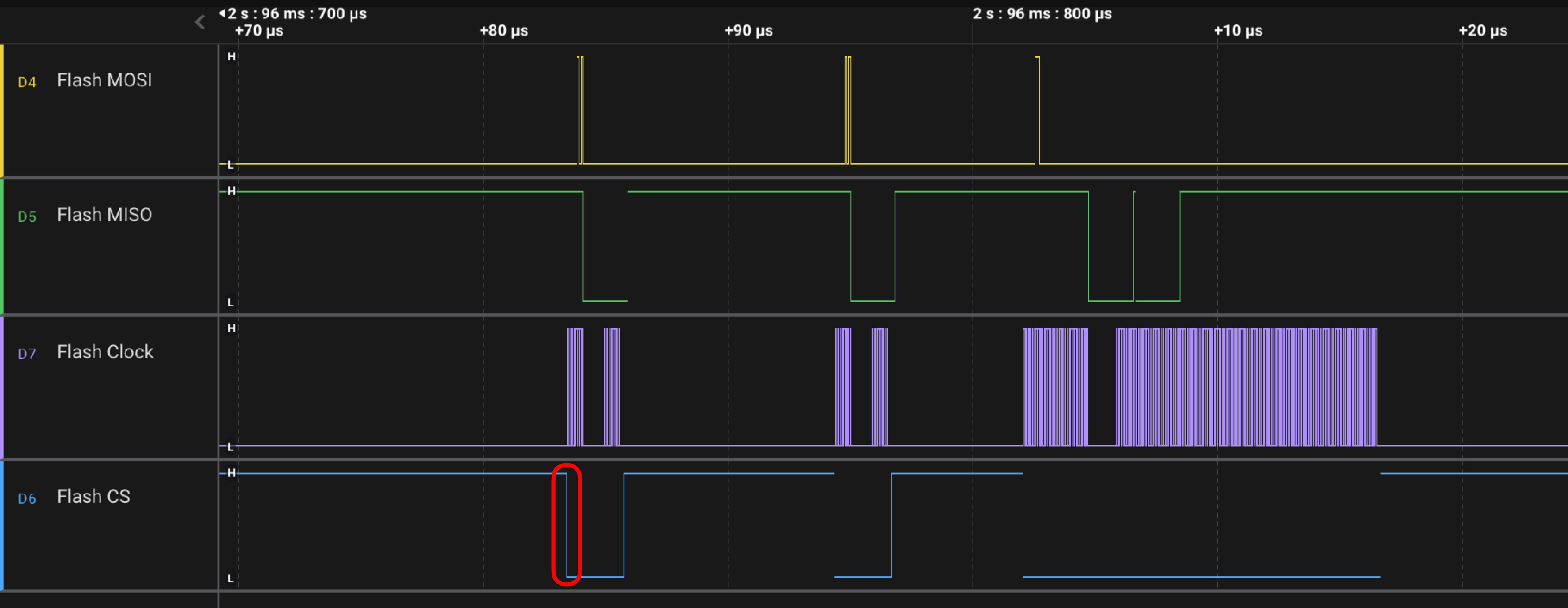
First ensure that the two devices share a common ground. This may be accomplished by connecting one's CLKIN to the other's CLKOUT as recommended above. Alternatively, connect a jumper wire from P28 pin 2 on one HackRF One to P28 pin 2 on the other HackRF One.

Next use a jumper wire to connect P28 pin 15 (trigger output) on one HackRF One to P28 pin 16 (trigger input) on the other HackRF One.

Usage

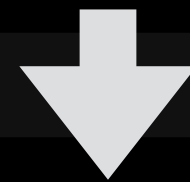






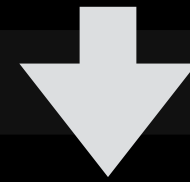
Start recording on CS
trigger on HackRF

Start recording on CS
trigger on HackRF

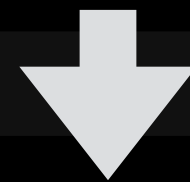


Reboot ACE3 via
acetooll3

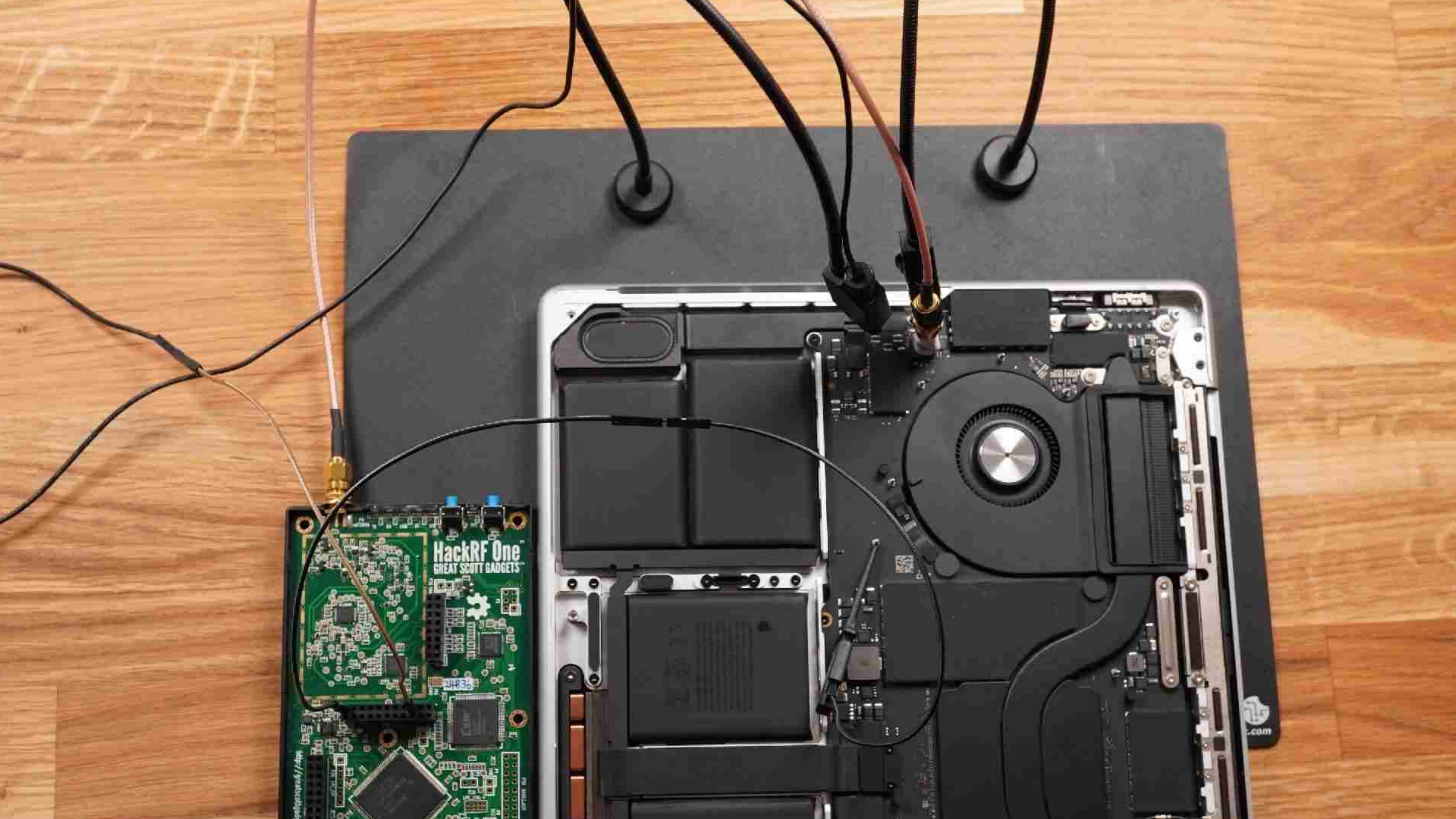
Start recording on CS
trigger on HackRF



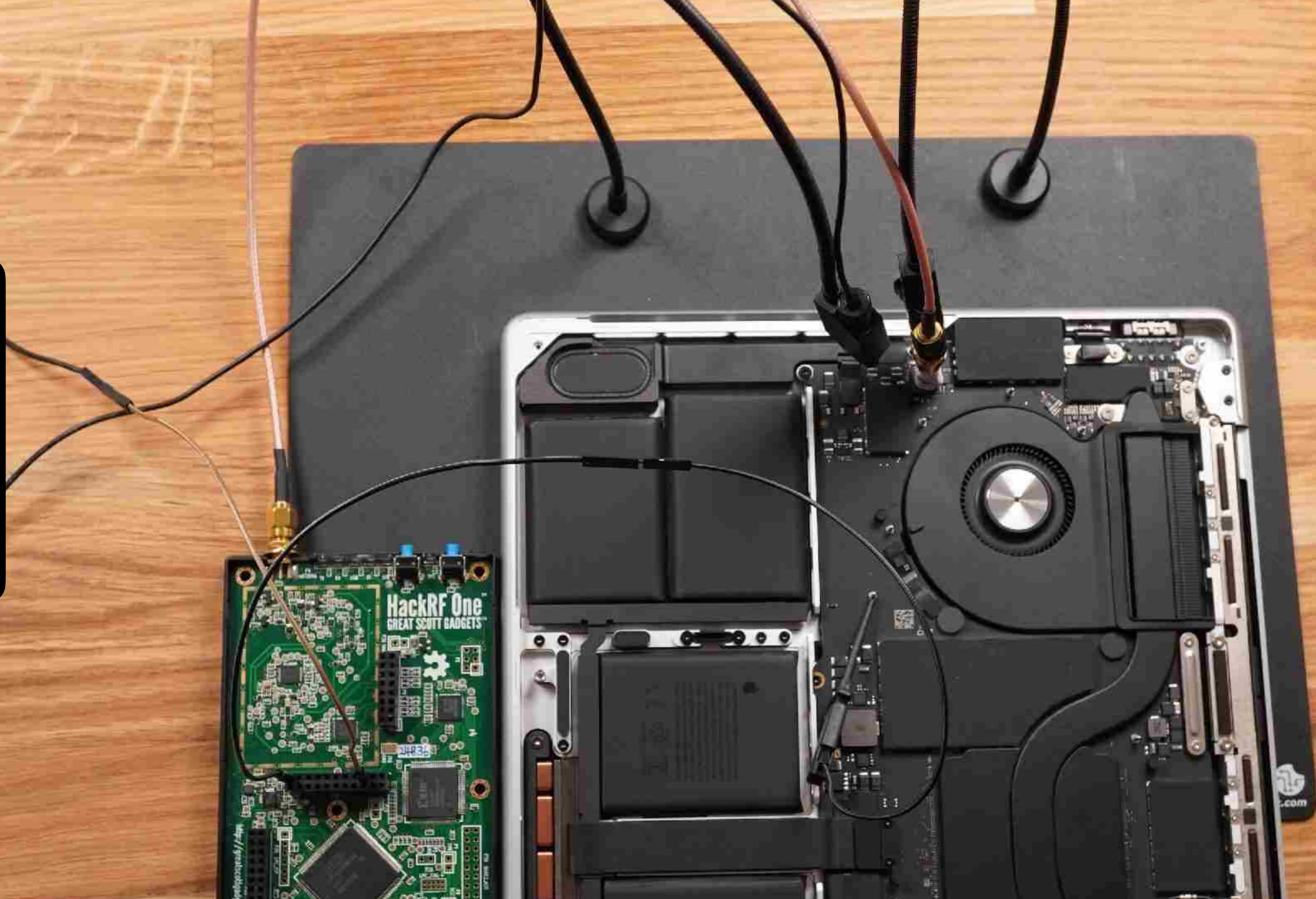
Reboot ACE3 via
acetoool3



Get perfectly aligned
recording 🎉



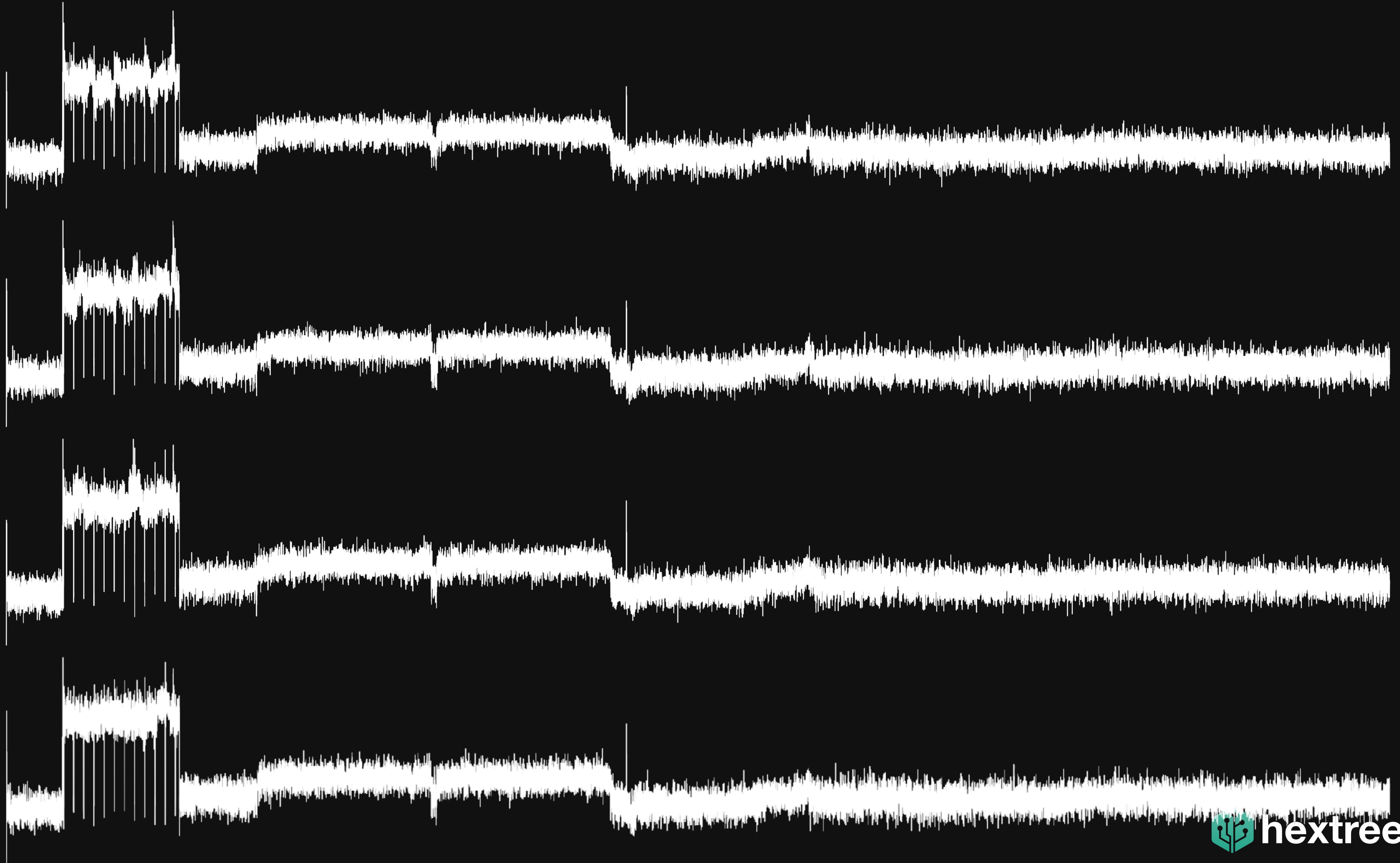
Trigger Inverter





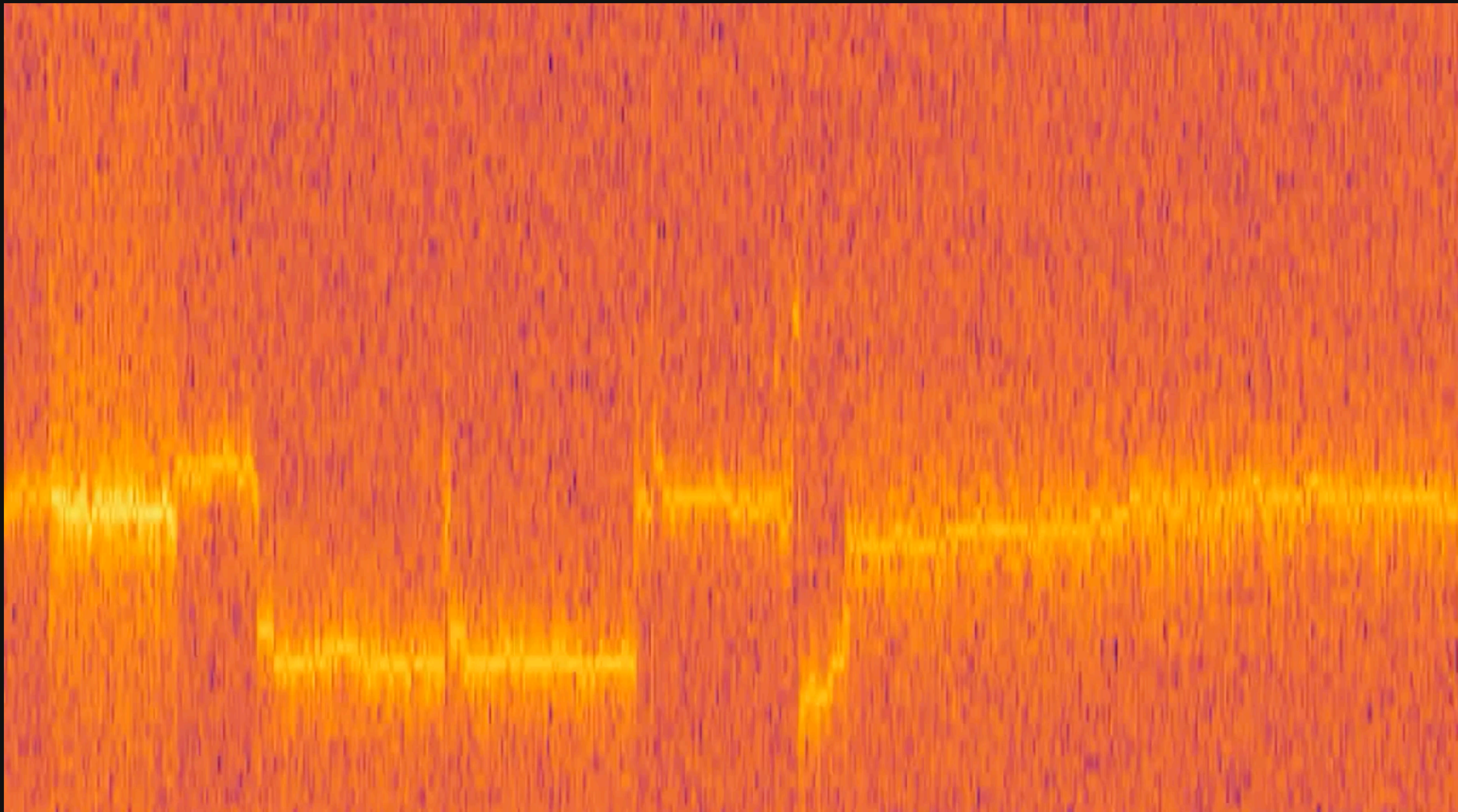
Flash chip-select
line for trigger







Perfect recordings!



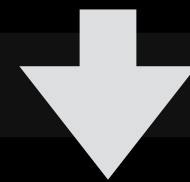




How do we use this side-channel
to time our glitch?

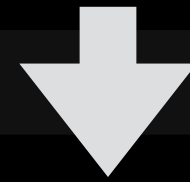
Change firmware

Change firmware

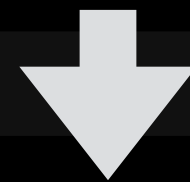


Reboot ACE3 via
acetool3 & record

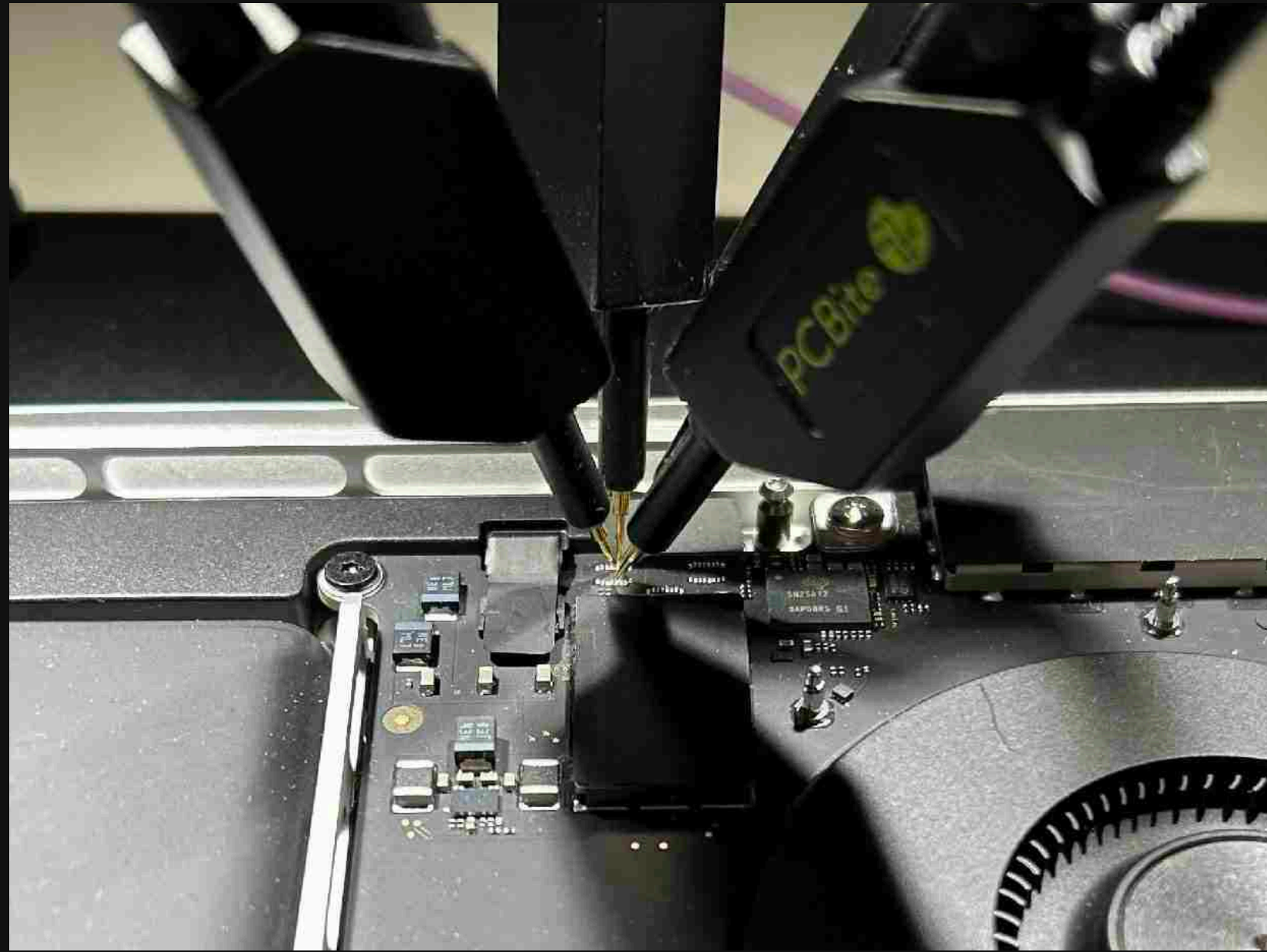
Change firmware

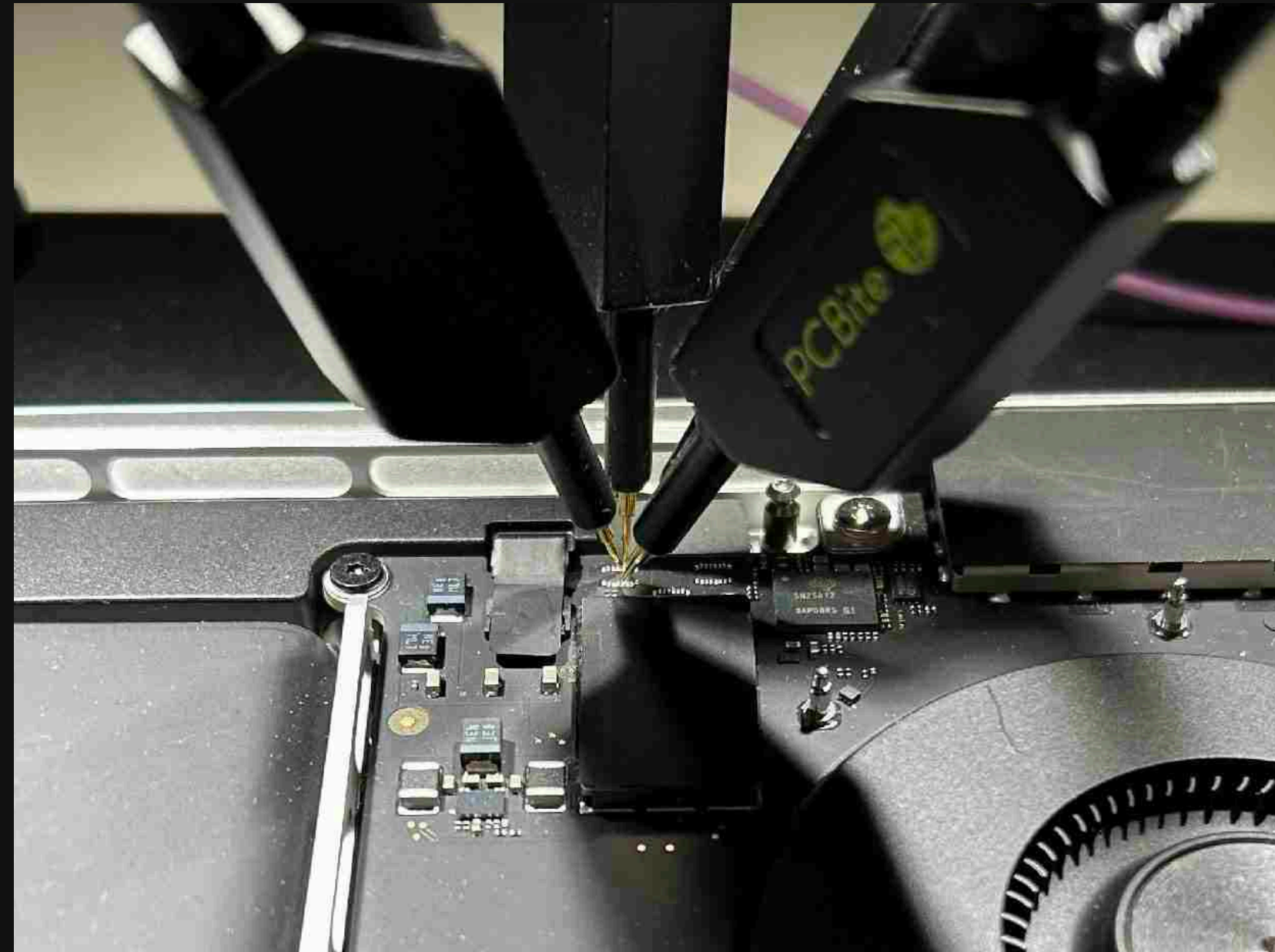


Reboot ACE3 via
acetool3 & record

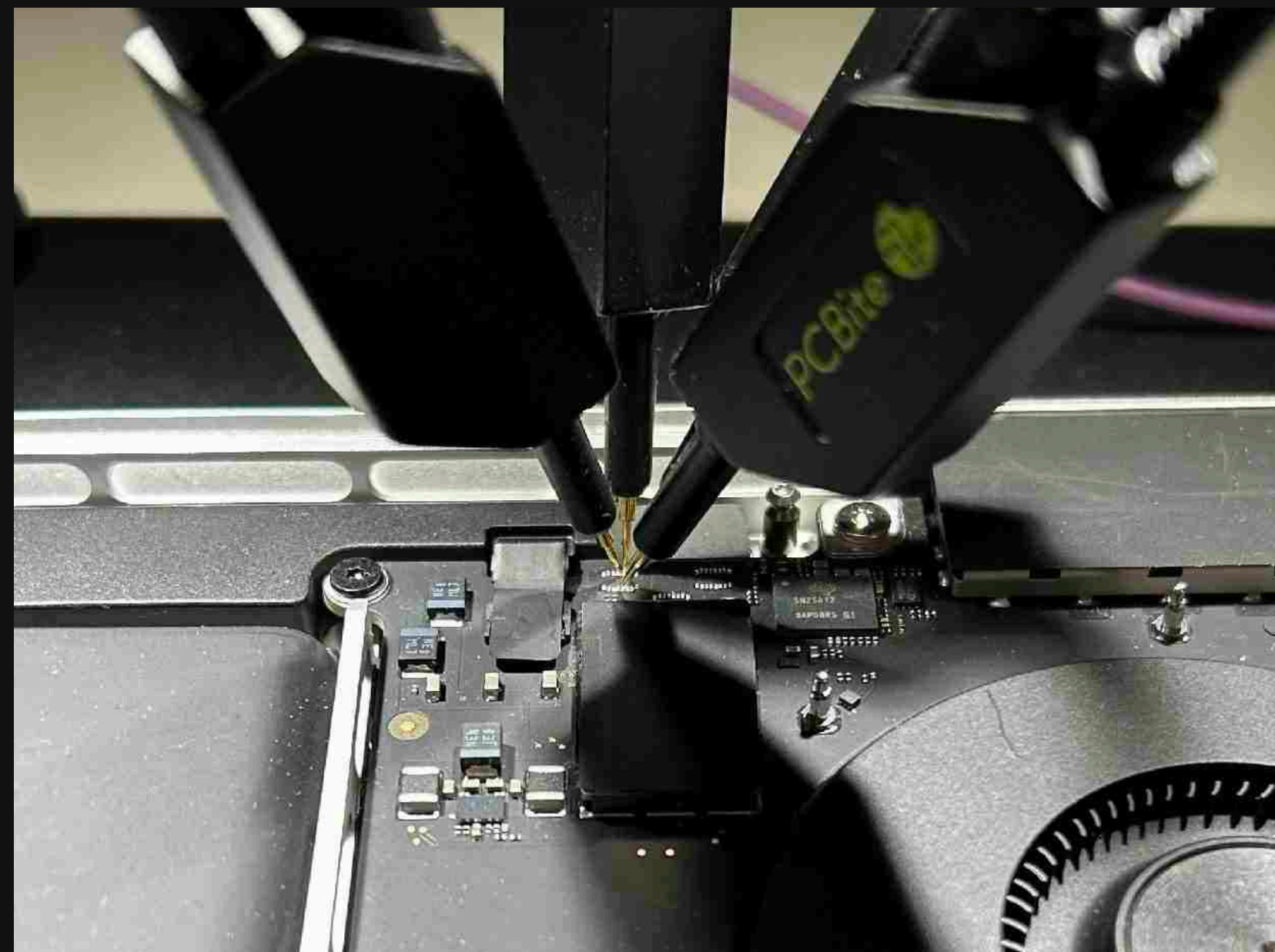


Compare differences





Try flashing til it
works



Try flashing til it
works

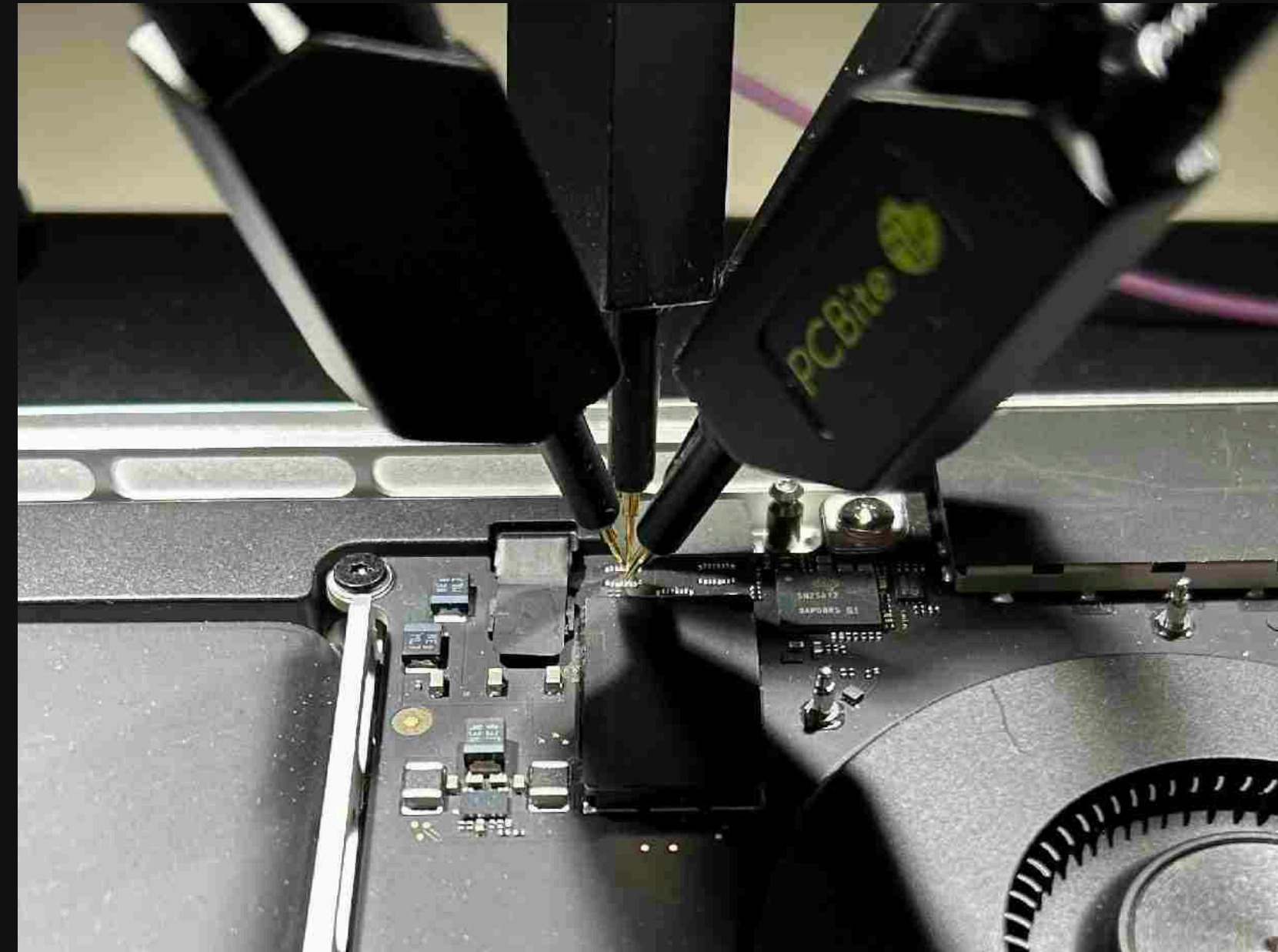




Try flashing til it
works

Measure





Try flashing til it
works

Measure



Original Firmware



Original Firmware



Modified firmware



Original Firmware



Modified firmware




```

00004000 struct ace3_firmware_header data_4000 =
00004000 {
00004000     uint32_t ace_id = 0xace00003
00004004     uint32_t ace_version = 0x204000
00004008     uint32_t unknown1 = 0x2800
0000400c     uint32_t ace_binary_start_relative = 0x40
00004010     uint32_t boot_config = 0xa8c0
00004014     uint32_t boot_config_size = 0x27f
00004018     uint32_t im4m_offset = 0xab7f
0000401c     uint32_t im4m_size = 0x77f
00004020     uint32_t ace_binary_size = 0xb2be
00004024     uint32_t ace_binary_crc = 0x19089c3d
00004028 }

```

```

00004028     ff ff ff ff ff ff ff ff
00004030 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

```

```

00004040 struct ace3_binary_header data_4040 =
00004040 {
00004040     uint32_t binary_size = 0xa7dc
00004044     uint32_t u[0x7] =
00004044     {
00004044         [0x0] = 0x00720000
00004048         [0x1] = 0x20051ef4
0000404c         [0x2] = 0x20051ef4
00004050         [0x3] = 0x20051f68
00004054         [0x4] = 0x20047725
00004058         [0x5] = 0x010cb105
0000405c         [0x6] = 0x00200000
00004060     }
00004060     uint32_t binary_crc = 0x3400337d
00004064     uint32_t version = 0x204000
00004068     uint32_t boot_config_offset = 0xa8c0
0000406c     uint32_t u2[0x4] =
0000406c     {
0000406c         [0x0] = 0x20047718
00004070         [0x1] = 0x00000000
00004074         [0x2] = 0x00000000
00004078         [0x3] = 0x00000000
0000407c     }
0000407c     uint32_t header_crc = 0xe8fc067e
00004080 }

```

```

00004000 struct ace3_firmware_header data_4000 =
00004000 {
00004000     uint32_t ace_id = 0xace00003
00004004     uint32_t ace_version = 0x204000
00004008     uint32_t unknown1 = 0x2800
0000400c     uint32_t ace_binary_start_relative = 0x40
00004010     uint32_t boot_config = 0xa8c0
00004014     uint32_t boot_config_size = 0x27f
00004018     uint32_t im4m_offset = 0xab7f
0000401c     uint32_t im4m_size = 0x77f
00004020     uint32_t ace_binary_size = 0xb2be
00004024     uint32_t ace_binary_crc = 0x19089c3d
00004028 }

```

```

00004028 ff ff ff ff ff ff ff ff
00004030 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

```

```

00004040 struct ace3_binary_header data_4040 =
00004040 {
00004040     uint32_t binary_size = 0xa7dc
00004044     uint32_t u[0x7] =
00004044     {
00004044         [0x0] = 0x00720000
00004048         [0x1] = 0x20051ef4
0000404c         [0x2] = 0x20051ef4
00004050         [0x3] = 0x20051f68
00004054         [0x4] = 0x20047725
00004058         [0x5] = 0x010cb105
0000405c         [0x6] = 0x00200000
00004060     }
00004060     uint32_t binary_crc = 0x3400337d
00004064     uint32_t version = 0x204000
00004068     uint32_t boot_config_offset = 0xa8c0
0000406c     uint32_t u2[0x4] =
0000406c     {
0000406c         [0x0] = 0x20047718
00004070         [0x1] = 0x00000000
00004074         [0x2] = 0x00000000
00004078         [0x3] = 0x00000000
0000407c     }
0000407c     uint32_t header_crc = 0xe8fc067e
00004080 }

```



Original Firmware

Original Firmware



Wrong first CRC



Original Firmware



Wrong first CRC



Wrong second CRC



Original Firmware



Wrong first CRC



Wrong second CRC



Wrong third CRC



Original Firmware



Wrong third CRC



Original Firmware



Wrong third CRC



Original Firmware



Glitch here!

Wrong third CRC







ChipSHOUTER® CW520
NewAE Technology Inc.

⚠ DANGER HIGH VOLTAGE ⚠
Equipment to be used by trained personnel only.

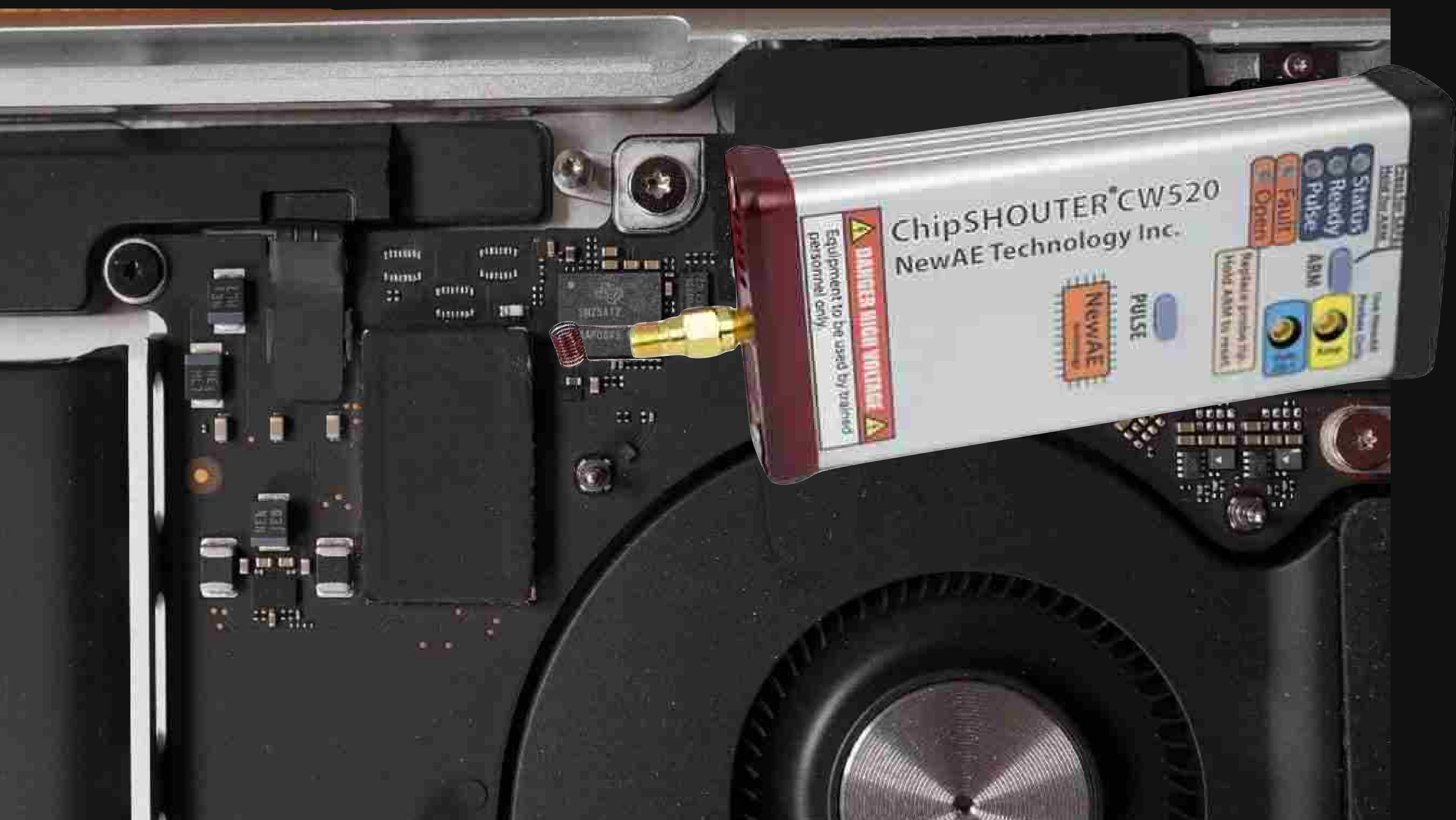
NEWAE
PULSE

Replace probe tip
Hold push to reset

ARM

Status
Ready
Pulse

Open
Fault



ChipSHOUTER[®] CW520
NewAE Technology Inc.

⚠ DANGER HIGH VOLTAGE ⚠
Equipment to be used by trained personnel only.

NewAE

PULSE

STATUS
READY
PULSE
FAULT
OK

ARM

Apply a positive 5V pulse to ARM to reset.

Test NewAE
Status OK



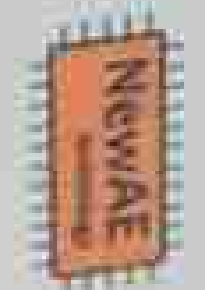






ChipSHOUTER® CW520
NewAE Technology Inc.

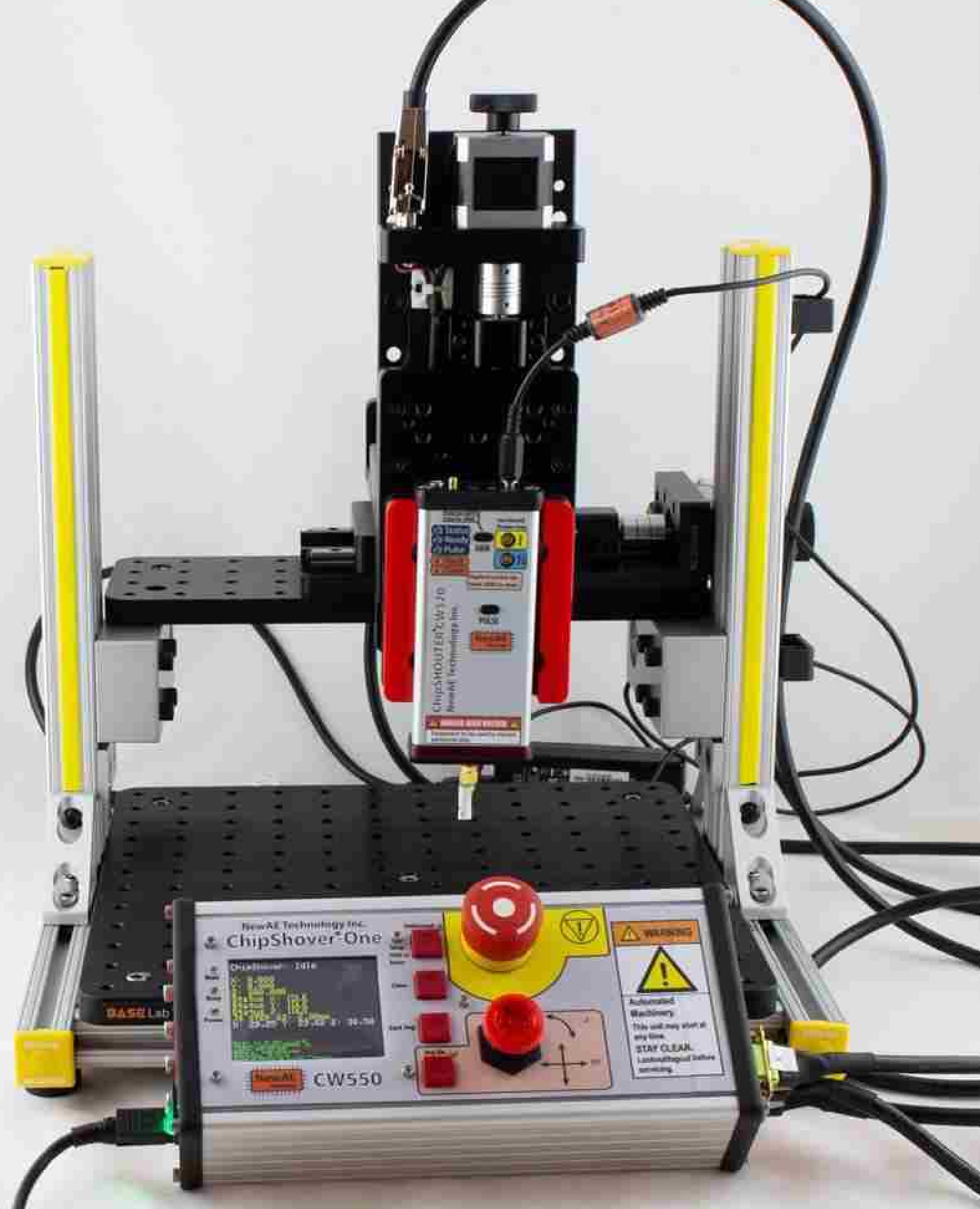
⚠ DANGER HIGH VOLTAGE ⚠
Equipment to be used by trained personnel only.

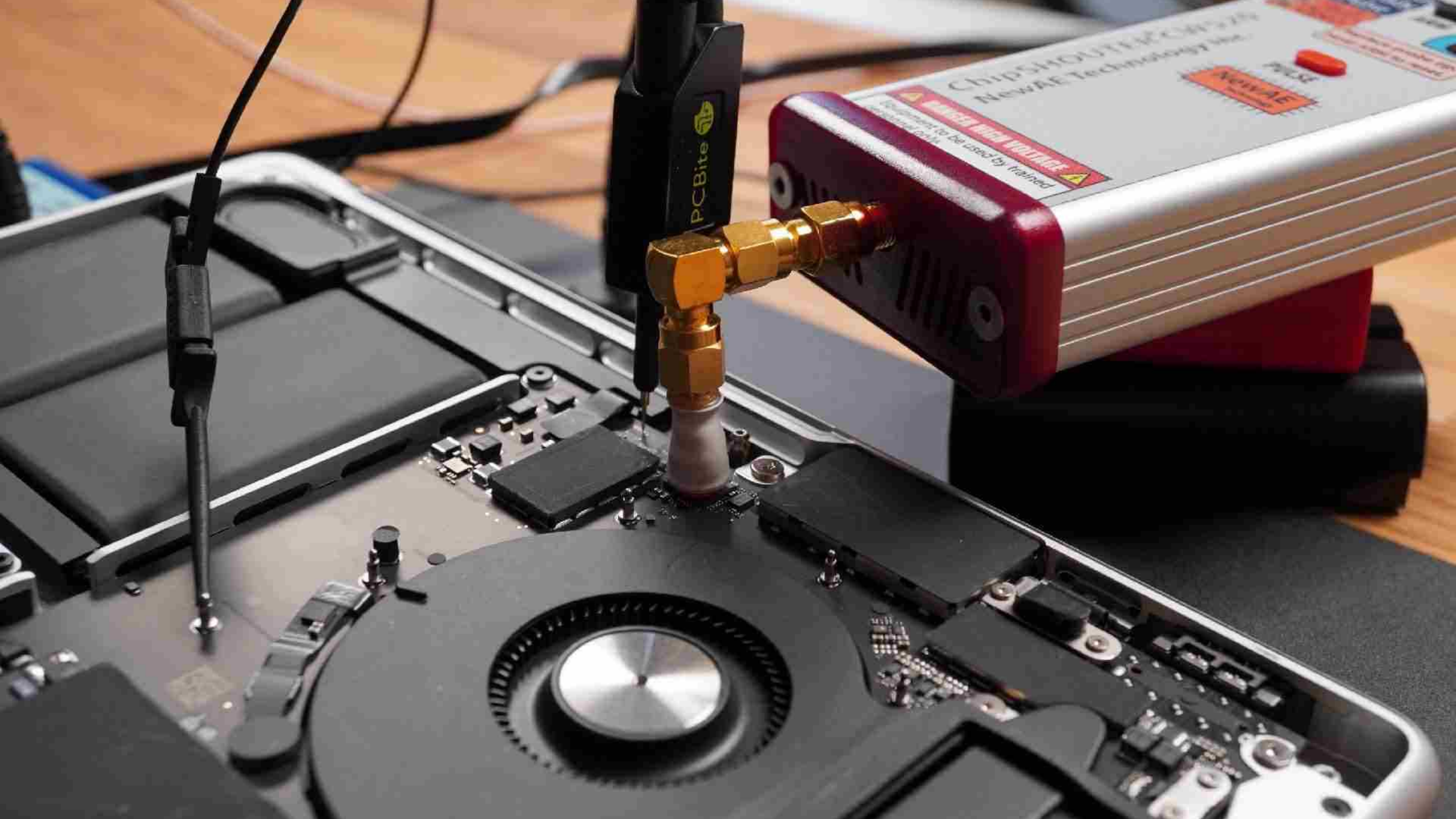


PULSE

Control Panel:

- Status
- Ready
- Pulse
- Fault
- Open
- ASIM
- Buttons: [Blue], [Yellow], [Blue]
- Display: [Small screen]
- Text: "Display a pulse signal. Hold ASIM to reset."





ChipSHOUTER



USB ChipWhisperer® Husky 20-Pin Connector

Trigger/Glitch Out (3.3V)

DIO 0-15

V+ V-

GPIO 0-15

GPIO 16-31

GPIO 32-47

GPIO 48-63

GPIO 64-79

GPIO 80-95

GPIO 96-111

GPIO 112-127

GPIO 128-143

GPIO 144-159

GPIO 160-175

GPIO 176-191

GPIO 192-207

GPIO 208-223

GPIO 224-239

GPIO 240-255

Aux In/Out (3.3V)

ADC+Glitch Blinking together indicates a stored event.

Measurement

Triggerbar Neg Pos

ChipSHOUTER® CW520 NewAE Technology Inc.

Replace probe tip. Hold ARM to reset.

PULSE

NewAE Technology

⚠ DANGER HIGH VOLTAGE ⚠

Equipment to be used by trained personnel only.

ChipSHOUTER

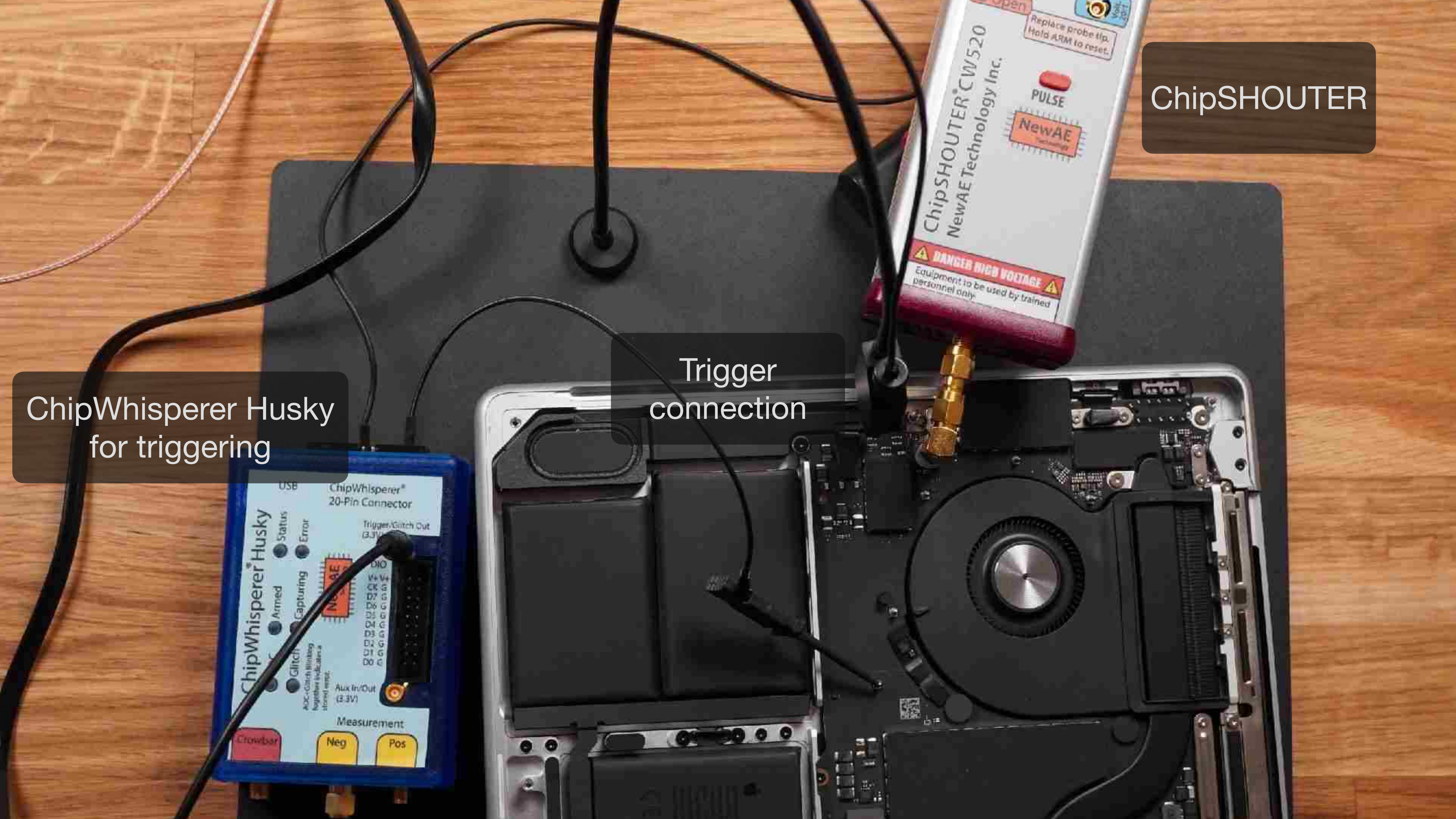
ChipWhisperer Husky
for triggering



ChipSHOUTER

Trigger connection

ChipWhisperer Husky for triggering

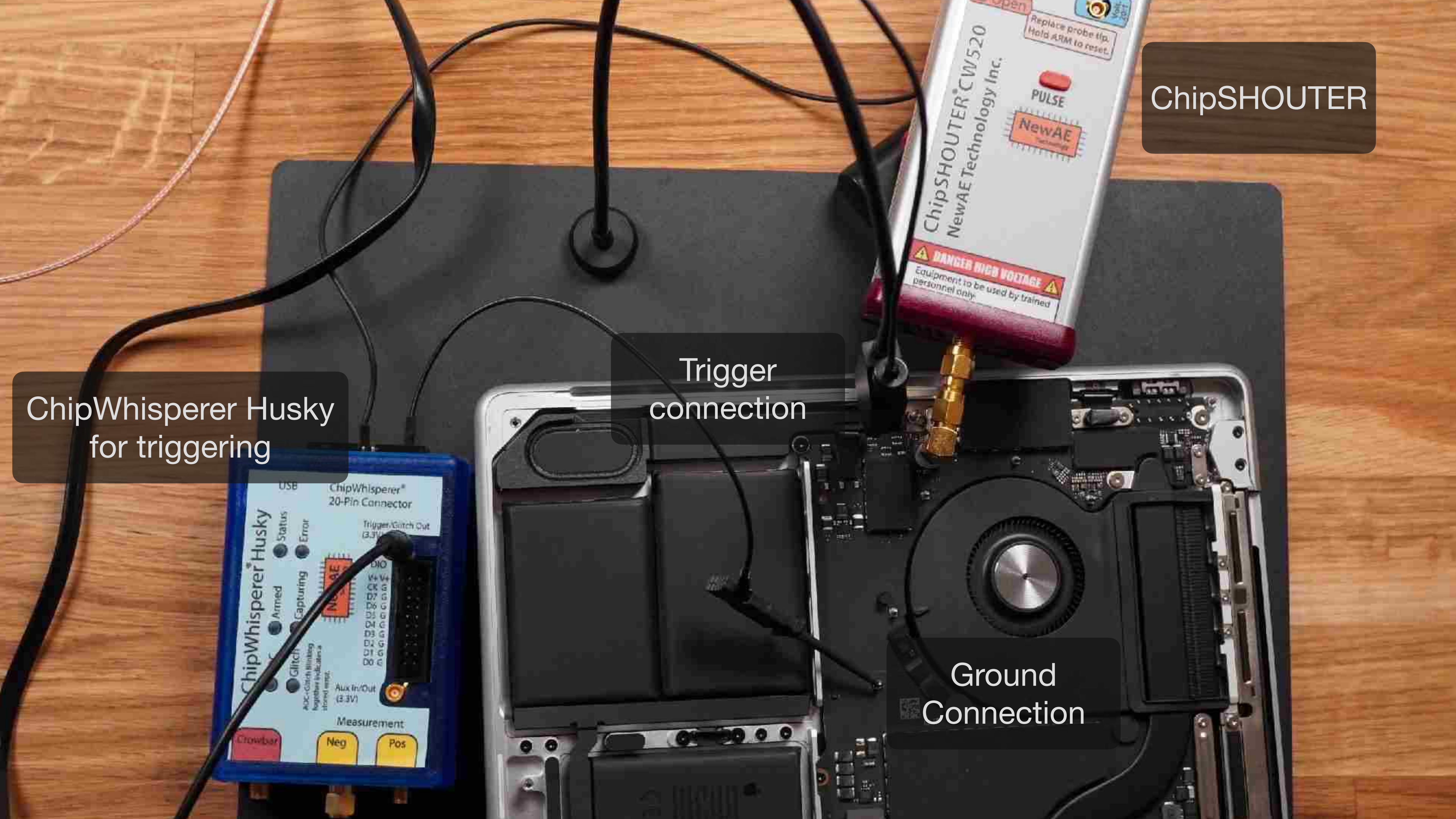


ChipSHOUTER

Trigger connection

ChipWhisperer Husky for triggering

Ground Connection



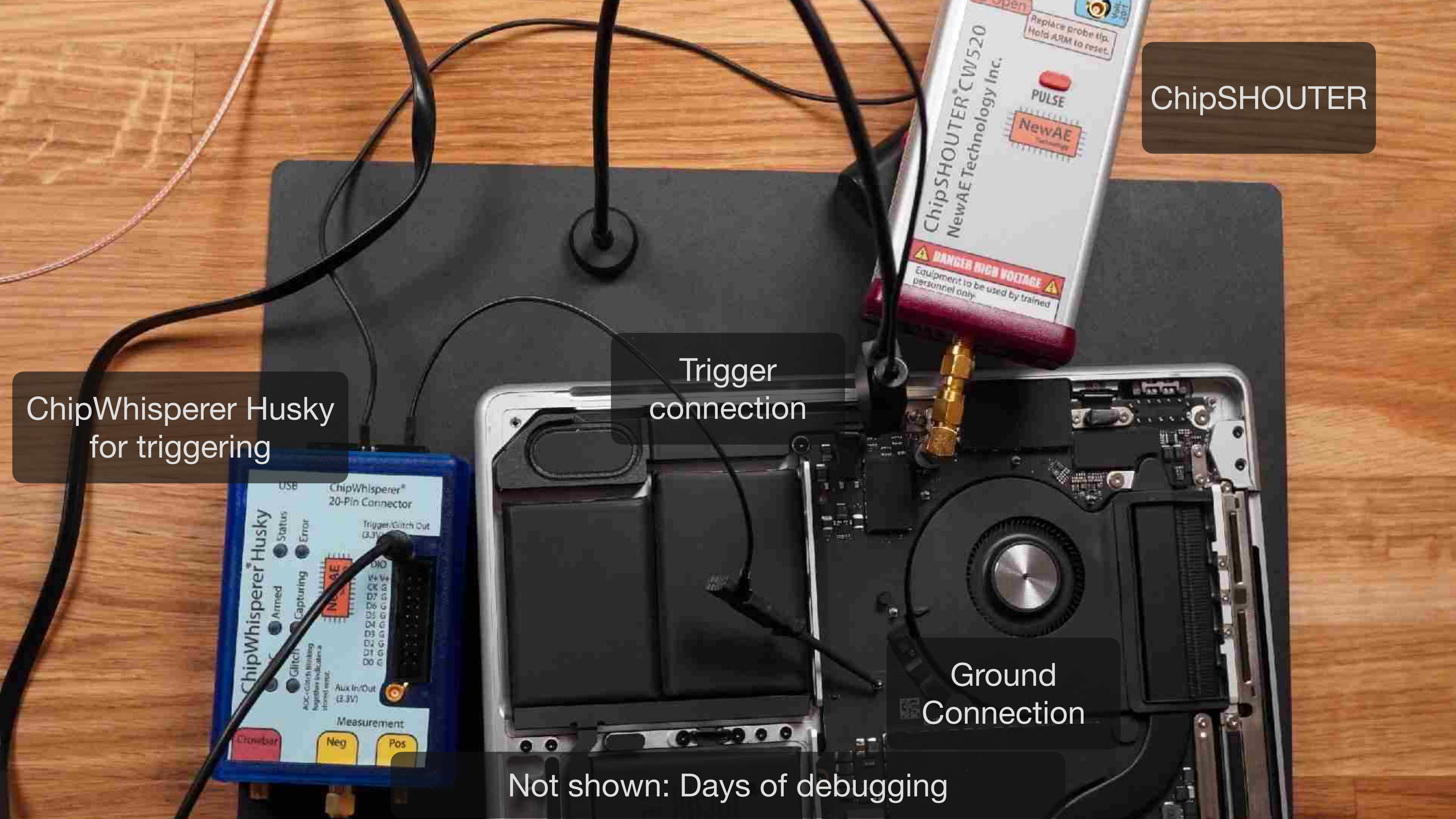
ChipSHOUTER

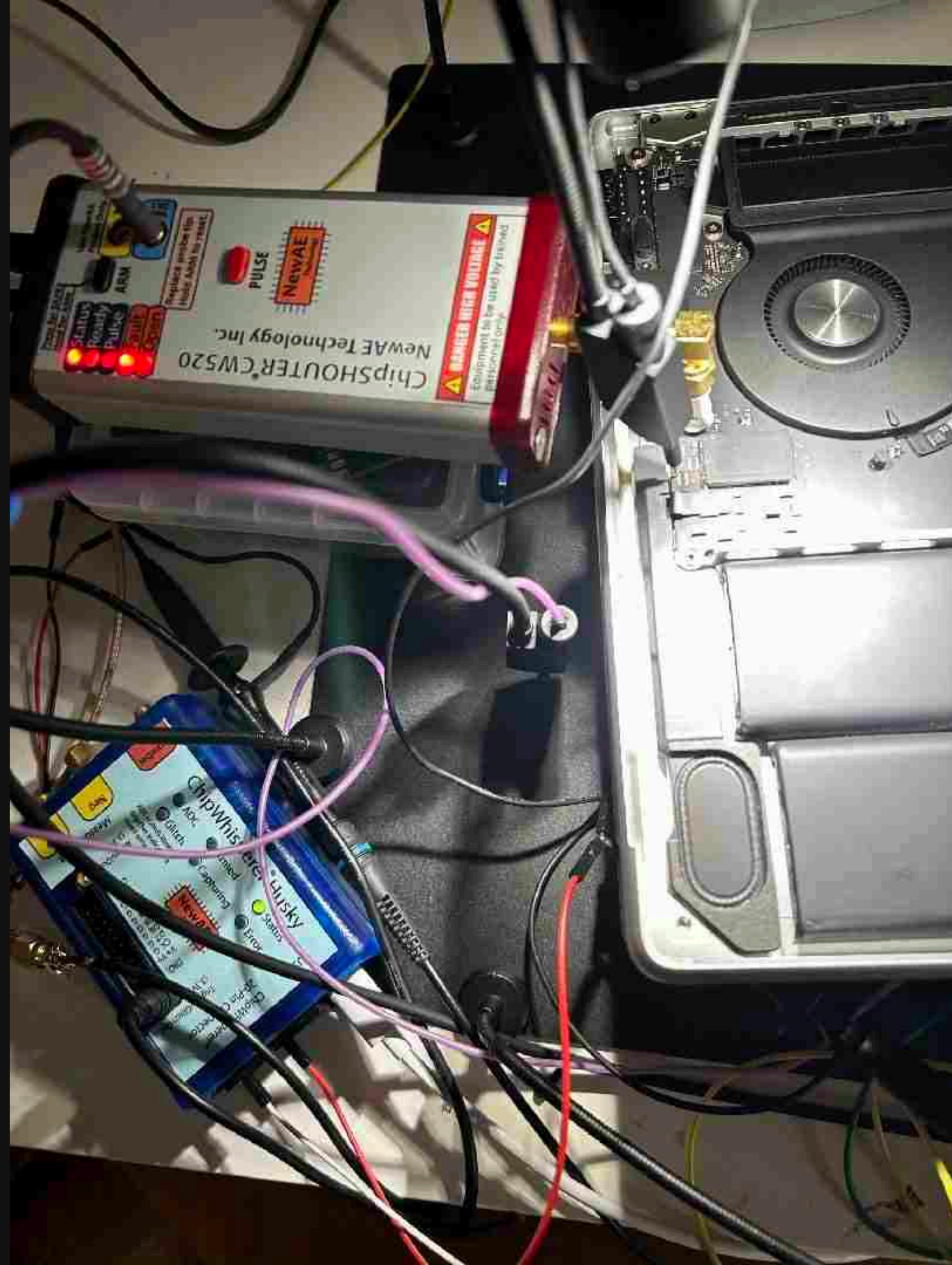
ChipWhisperer Husky
for triggering

Trigger
connection

Ground
Connection

Not shown: Days of debugging





Attempt 1: Change version string


```
[research@researchs-MBP aceglitch % sudo hpmdiagnose | grep 0x2f
```

```
0x2f      0x40      0x534E32303132303235204857303041312046573030322E3034302E3030205A414345332D4A3531345030310000000000000000000000000000000000000000000000000000
```



```
[research@researchs-MBP aceglitch % sudo hpmdiagnose | grep 0x2f
```

```
0x2f      0x40      0x534E32303132303235204857303041312046573030322E3034302E3030205A414345332D4A353134503031000000000000000000000000000000000000
```

SN2012025 HW00A1 FW002.045.00 ZACE3

```
[research@researchs-MBP aceglitch % sudo hpmidiagnose | grep 0x2f  
0x2f      0x40      0x534E32303132303235204857303041312046573030322E3034302E3030205A414345332D4A3531345030310000000000000000000000000000000000000000
```

SN2012025 HW00A1 FW002.045.00 ZACE3

```
[research@researchs-MBP aceglitch % sudo hpmidiagnose | grep 0x2f  
0x2f      0x40      0x534E32303132303235204857303041312046573030322E3034302E3030205A414345332D4A35313450303100000000000000000000000000000000
```

SN2012025 HW00A1 FW002.045.00 ZACE3

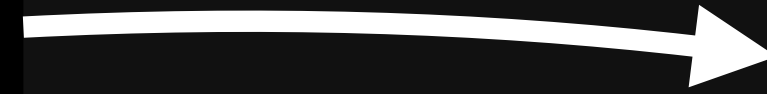
```

*
0x00082000  03 00 E0 AC 00 45 20 00 00 28 00 00 40 00 00 00 | .....E...(...@... | \ 03 00 E0 AC 00 42 20 00 00 28 00 00 40 00 00 00 | .....B...(...@... |
*
0x00082020  3F B6 00 00 7F C0 DC 1A FF FF FF FF FF FF FF FF | ?..... | / 3F B6 00 00 8C 1E 9C AC FF FF FF FF FF FF FF FF | ?..... |
*
0x00082060  82 E3 C5 A7 00 45 20 00 C0 AC 00 00 18 77 04 20 | .....E.....w.. | \ 82 E3 C5 A7 00 42 20 00 C0 AC 00 00 18 77 04 20 | .....B.....w.. |
0x00082070  00 00 00 00 00 00 00 00 00 00 00 00 CC 04 04 67 | .....g | / 00 00 00 00 00 00 00 00 00 00 00 00 A2 99 1D 12 | ..... |
*

```


Arm Husky &
ChipSHOUTER

Arm Husky &
ChipSHOUTER



Reboot ACE3 via
acetool3

Arm Husky &
ChipSHOUTER

Reboot ACE3 via
acetool3

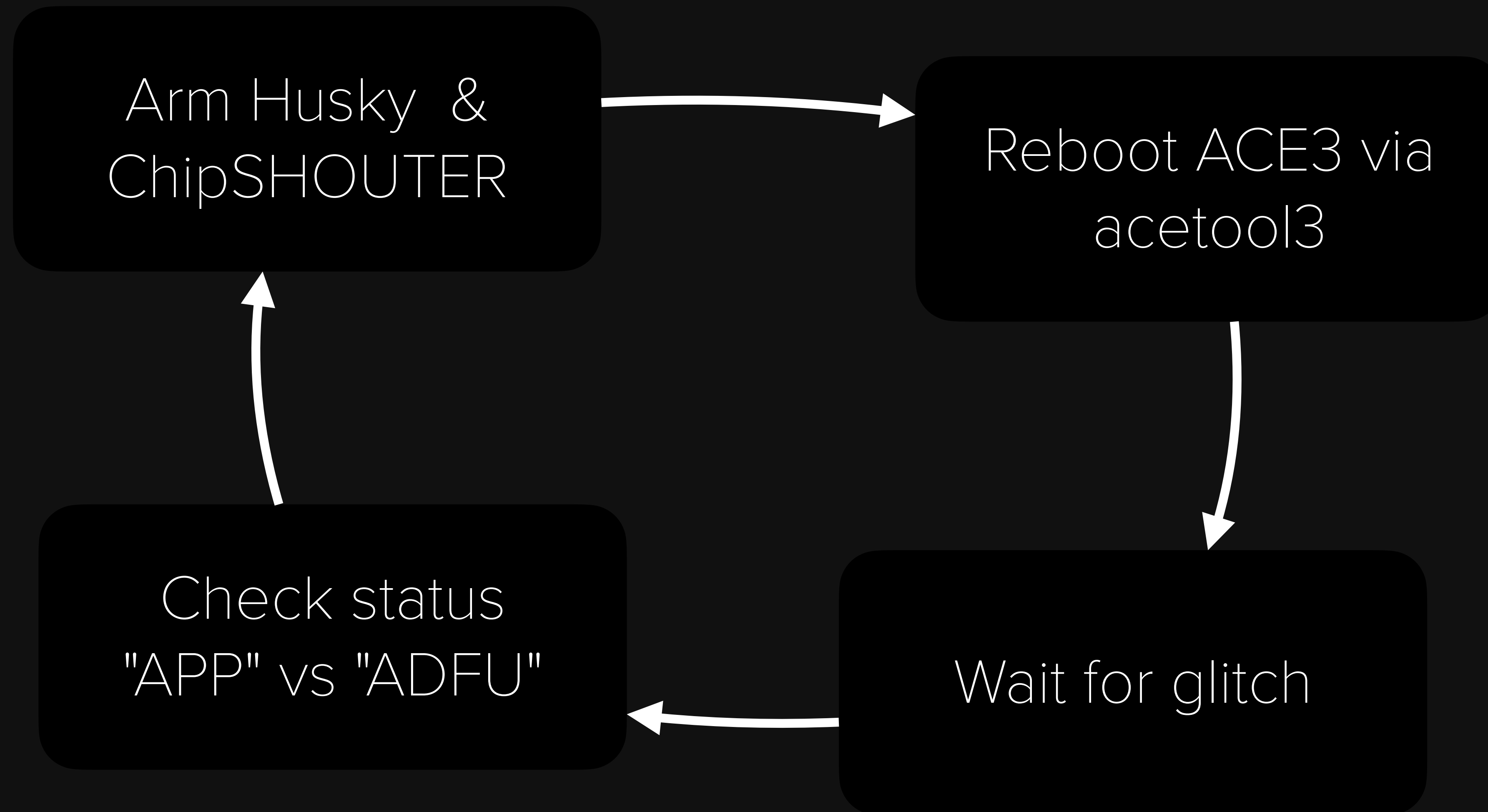
Wait for glitch

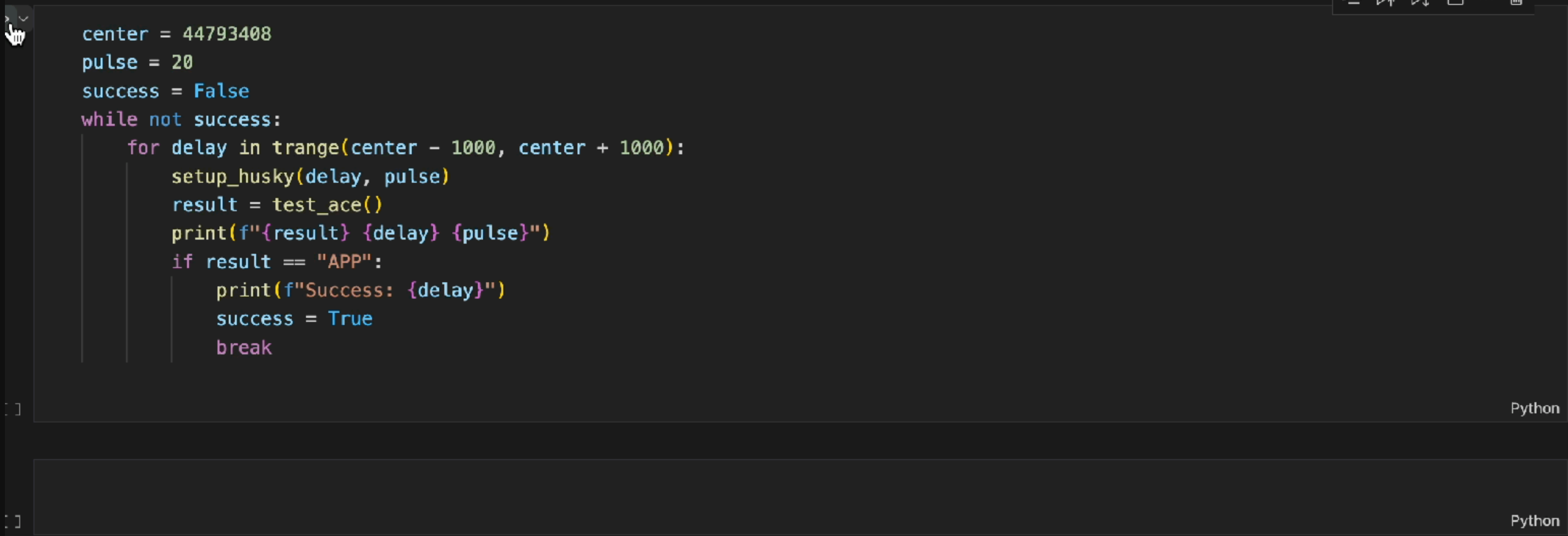
Arm Husky &
ChipSHOUTER

Reboot ACE3 via
acetool3

Check status
"APP" vs "ADFU"

Wait for glitch





```
center = 44793408
pulse = 20
success = False
while not success:
    for delay in trange(center - 1000, center + 1000):
        setup_husky(delay, pulse)
        result = test_ace()
        print(f"{result} {delay} {pulse}")
        if result == "APP":
            print(f"Success: {delay}")
            success = True
            break
```

Python

Python

The troubles...

- ACE sometimes completely stops responding → Auto-reboot
- After reboot, MacBook stops charging while one ACE is "bricked"
- 8 hours, then have to restore ACE3 and start over...

```
ADFU 44792408 20
ADFU 44792409 20
ADFU 44792410 20
ADFU 44792411 20
ADFU 44792412 20
ADFU 44792413 20
ADFU 44792414 20
ADFU 44792415 20
ADFU 44792416 20
ADFU 44792417 20
ADFU 44792418 20
ADFU 44792419 20
ADFU 44792420 20
ADFU 44792421 20
ADFU 44792422 20
ADFU 44792423 20
ADFU 44792424 20
ADFU 44792425 20
ADFU 44792426 20
ADFU 44792427 20
ADFU 44792428 20
ADFU 44792429 20
ADFU 44792430 20
ADFU 44792431 20
ADFU 44792432 20
...
ADFU 44793142 20
ADFU 44793143 20
APP 44793144 20
Success: 44793144
```



```
[research@researchs-MBP aceglitch % sudo hpmdiagnose | grep 0x2f
```

[illegible]

SN2012025 HW00A1 FW002.042.00 ZACE3

We glitched the ACE3! 🎉

But...



- We can only modify patches
- We don't know what the patches patch
- We have no input/output

But...



- We can only modify patches
- We don't know what the patches patch
- We have no input/output

But we have the ACE2 firmware...

00040eee	int32_t r5 = 0
00040ef2	if (data_2004005c_CURRENT_MODE_1 == 'DISC')
00040f84	label_40f84:
00040f84	r5 = 3
00040ef8	else
00040ef8	if (data_2004005c_CURRENT_MODE_1 == 'UFPf')
00040ef8	goto label_40f84
00040efe	if (data_2004005c_CURRENT_MODE_1 == 'DFUf')
00040efe	goto label_40f84
00040f02	var_18 = &data_20042988
00040f08	if (zx.d(data_20042991) == 0)
00040f08	goto label_40f84
00040f10	if (sub_44cc8() == 0)
00040f10	goto label_40f84
00040f28	if (zx.d(**(arg2 + 4)) u>> 7 == 0 (zx.d(**(arg2 + 4)) u>> 7 != 0 &
00040f40	int32_t data_2004005c_CURRENT_MODE_2 = data_2004005c_CURRENT_MODE
00040f42	if (zx.d(*r6) << 0x1f == 0)
00040f82	if (data_2004005c_CURRENT_MODE_2 != 'USBw')
00040f82	goto label_40f84
00040f88	data_2005671e = 0
00040f8a	uint32_t r0_18 = zx.d(*r6)
00040f8c	data_200424f1 = r0_18.b
00040f8e	sub_40e14(r0_18, 'USBw', 0, 0x2005671e)
00040f48	else
00040f48	if (data_2004005c_CURRENT_MODE_2 == 'USBw')
00040f48	goto label_40f84
00040f4e	if (data_2004005c_CURRENT_MODE_2 == 'CFUp')
00040f52	*(var_18 + 0x19) = 0
00040f58	if (var_20 == 7)
00040f5e	var_20 = 0x183
00040f62	sub_35d22(&var_20)
00040f68	__builtin_strncpy(dest: &data_2004005c_CURRENT_MODE, src: "US
00040f6c	data_200424f1 = *r6
00040f6e	sub_40c6c()

ACE3 Flash


```

20406818 int32_t r7 = 0
2040681c if (probably_current_mode_1 == 'DISC')
204068f0     label_204068f0:
204068f0         r7 = 3
20406822 else
20406822     if (probably_current_mode_1 == 'UFPf')
20406822         goto label_204068f0
20406828     if (probably_current_mode_1 == 'DFUF')
20406828         goto label_204068f0
20406832     if (zx.d(data_200430b9) == 0)
20406832         goto label_204068f0
2040683a     if (sub_2040910c() == 0)
2040683a         goto label_204068f0
20406858     if (zx.d(**(arg2 + 8)) u>> 7 == 0 || (zx.d(**(arg2 + 8))
2040685a         uint32_t r3_5 = zx.d(*r5)
20406862         if (r3_5 << 0x1f != 0)
20406866             int32_t probably_current_mode_2 = probably_curr
2040686a             if (probably_current_mode_2 == 'USBw')
2040686a                 goto label_204068f0
20406872             if (probably_current_mode_2 == 'CFUp')
20406876                 data_200430c9 = 0
2040687a                 data_20046bdc = 0
2040687e                 data_20046bdd = 0
20406886                 if (zx.d(data_20042cb2) u>> 4 == 0xf)
204068a0                     int32_t var_18_1 = 0x183
204068ca                     __builtin_strncpy(dest: &probably_current_mode,
204068ce                         data_200429d9 = *r5
204068d8                         int32_t var_1c_1 = 0x8084
204068e8                     else if (zx.d(data_20042cb2) u>> 4 == 0xe)
204068f6                         data_200429d9 = r3_5.b
204068fa                         data_20046bdd = 1
204068ee                     else
204068ee                         if (probably_current_mode != 'USBw')
204068ee                             goto label_204068f0

```

ACE2 Firmware

```

00040eee int32_t r5 = 0
00040ef2 if (data_2004005c_CURRENT_MODE_1 == 'DISC')
00040f84     label_40f84:
00040f84         r5 = 3
00040ef8 else
00040ef8     if (data_2004005c_CURRENT_MODE_1 == 'UFPf')
00040ef8         goto label_40f84
00040efe     if (data_2004005c_CURRENT_MODE_1 == 'DFUF')
00040efe         goto label_40f84
00040f02     var_18 = &data_20042988
00040f08     if (zx.d(data_20042991) == 0)
00040f08         goto label_40f84
00040f10     if (sub_44cc8() == 0)
00040f10         goto label_40f84
00040f28     if (zx.d(**(arg2 + 4)) u>> 7 == 0 || (zx.d(**(arg2 + 4)) u>> 7 != 0 &
00040f40         int32_t data_2004005c_CURRENT_MODE_2 = data_2004005c_CURRENT_MODE
00040f42         if (zx.d(*r6) << 0x1f == 0)
00040f82             if (data_2004005c_CURRENT_MODE_2 != 'USBw')
00040f82                 goto label_40f84
00040f8a                 data_2005671e = 0
00040f8c                 uint32_t r0_18 = zx.d(*r6)
00040f8e                 data_200424f1 = r0_18.b
00040f48                 sub_40e14(r0_18, 'USBw', 0, 0x2005671e)
00040f48     else
00040f48         if (data_2004005c_CURRENT_MODE_2 == 'USBw')
00040f48             goto label_40f84
00040f4e         if (data_2004005c_CURRENT_MODE_2 == 'CFUp')
00040f52             *(var_18 + 0x19) = 0
00040f58             if (var_20 == 7)
00040f5e                 var_20 = 0x183
00040f62                 sub_35d22(&var_20)
00040f68         __builtin_strncpy(dest: &data_2004005c_CURRENT_MODE, src: "US
00040f6c         data_200424f1 = *r6
00040f6e         sub_40c6c()

```

ACE3 Flash

USBw Command Handler

```
20406818 int32_t r7 = 0
2040681c if (probably_current_mode_1 == 'DISC')
204068f0     label_204068f0:
204068f0     r7 = 3
20406822 else
20406822     if (probably_current_mode_1 == 'UFPf')
20406822     |     goto label_204068f0
20406828     if (probably_current_mode_1 == 'DFUf')
20406828     |     goto label_204068f0
20406832     if (zx.d(data_200430b9) == 0)
20406832     |     goto label_204068f0
2040683a     if (sub_2040910c() == 0)
2040683a     |     goto label_204068f0
20406858     if (zx.d(**(arg2 + 8)) u>> 7 == 0 || (zx.d(**(arg2 + 8))
2040685a     |     uint32_t r3_5 = zx.d(*r5)
20406862     |     if (r3_5 << 0x1f != 0)
20406866     |     |     int32_t probably_current_mode_2 = probably_current_mode_1
2040686a     |     |     if (probably_current_mode_2 == 'USBw')
2040686a     |     |     |     goto label_204068f0
20406872     |     |     if (probably_current_mode_2 == 'CFUp')
20406876     |     |     |     data_200430c9 = 0
2040687a     |     |     |     data_20046bdc = 0
2040687e     |     |     |     data_20046bdd = 0
20406886     |     |     |     if (zx.d(data_20042cb2) u>> 4 == 0xf)
204068a0     |     |     |     |     int32_t var_18_1 = 0x183
204068ca     |     |     |     |     __builtin_strncpy(dest: &probably_current_mode,
204068ce     |     |     |     |     data_200429d9 = *r5
204068d8     |     |     |     |     int32_t var_1c_1 = 0x8084
204068e8     |     |     |     else if (zx.d(data_20042cb2) u>> 4 == 0xe)
204068f6     |     |     |     |     data_200429d9 = r3_5.b
204068fa     |     |     |     |     data_20046bdd = 1
204068ee     |     |     else
204068ee     |     |     |     if (probably_current_mode != 'USBw')
204068ee     |     |     |     |     goto label_204068f0
```

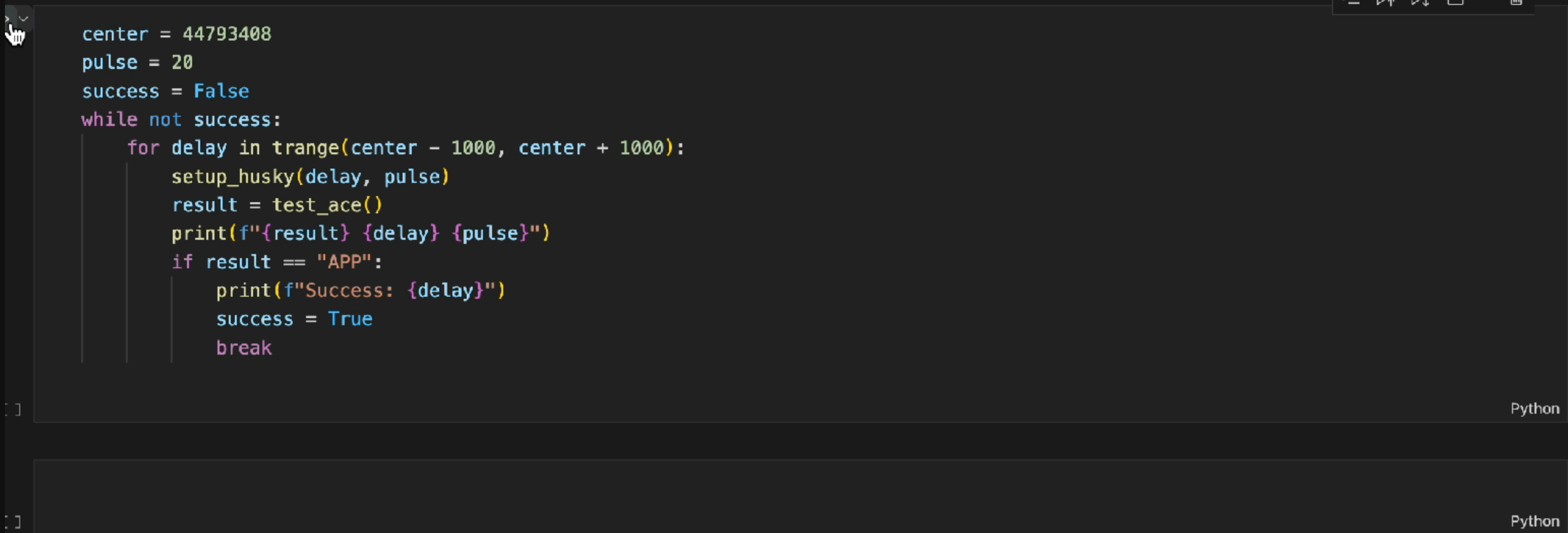
```
00040eee int32_t r5 = 0
00040ef2 if (data_2004005c_CURRENT_MODE_1 == 'DISC')
00040f84     label_40f84:
00040f84     r5 = 3
00040ef8 else
00040ef8     if (data_2004005c_CURRENT_MODE_1 == 'UFPf')
00040ef8     |     goto label_40f84
00040efe     if (data_2004005c_CURRENT_MODE_1 == 'DFUf')
00040efe     |     goto label_40f84
00040f02     var_18 = &data_20042988
00040f08     if (zx.d(data_20042991) == 0)
00040f08     |     goto label_40f84
00040f10     if (sub_44cc8() == 0)
00040f10     |     goto label_40f84
00040f28     if (zx.d(**(arg2 + 4)) u>> 7 == 0 || (zx.d(**(arg2 + 4)) u>> 7 != 0 &
00040f40     |     int32_t data_2004005c_CURRENT_MODE_2 = data_2004005c_CURRENT_MODE
00040f52     |     if (zx.d(*r6) << 0x1f == 0)
00040f62     |     |     if (data_2004005c_CURRENT_MODE_2 != 'USBw')
00040f6c     |     |     |     goto label_40f84
00040f6e     |     |     |     data_2005671e = 0
00040f6e     |     |     |     uint32_t r0_18 = zx.d(*r6)
00040f6e     |     |     |     data_200424f1 = r0_18.b
00040f6e     |     |     |     sub_40e14(r0_18, 'USBw', 0, 0x2005671e)
00040f6e     |     else
00040f6e     |     |     if (data_2004005c_CURRENT_MODE_2 == 'USBw')
00040f6e     |     |     |     goto label_40f84
00040f6e     |     |     if (data_2004005c_CURRENT_MODE_2 == 'CFUp')
00040f6e     |     |     |     |     *(var_18 + 0x19) = 0
00040f6e     |     |     |     if (var_20 == 7)
00040f6e     |     |     |     |     var_20 = 0x183
00040f6e     |     |     |     |     sub_35d22(&var_20)
00040f6e     |     |     |     __builtin_strncpy(dest: &data_2004005c_CURRENT_MODE, src: "US
00040f6e     |     |     data_200424f1 = *r6
00040f6e     |     |     sub_40c6c()
```


Payload

```
push    {r4, r5, r6, lr}
ldr     r4, [r1, #4]
ldr     r0, [r4]
ldr     r0, [r0]
str     r0, [r4]
movs    r0, #0xFF
pop     {r4, r5, r6, pc}
```

- Trivial memory reader
- Takes in address
- Returns bytes at address

Attempt 2: Replaced USBw Command



The image shows a dark-themed IDE window with a Python script. The script defines variables for a center value and a pulse, then enters a while loop that iterates over a range of delay values. Inside the loop, it calls setup_husky and test_ace functions, prints the results, and checks for a successful outcome. The window has a toolbar at the top right with icons for file operations and a sidebar on the left with a file explorer icon.

```
center = 44793408
pulse = 20
success = False
while not success:
    for delay in trange(center - 1000, center + 1000):
        setup_husky(delay, pulse)
        result = test_ace()
        print(f"{result} {delay} {pulse}")
        if result == "APP":
            print(f"Success: {delay}")
            success = True
            break
```

Python

Python

```
$ sudo ./acetool USBw 00000000
```

```
$ sudo ./acetoool USBw 00000000  
Status: APP  
Running command: USBw – Data: 4  
Executing command  
Result is: 00 00 06 20  
Status: APP
```



```
$ sudo ./acetooll USBw 00000000  
Status: APP  
Running command: USBw – Data: 4  
Executing command  
Result is: 00 00 06 20  
Status: APP
```

```
$ sudo ./acetooll USBw 00000000  
Status: APP  
Running command: USBw – Data: 4  
Executing command  
Result is: 00 00 06 20    20 06 00 00  
Status: APP
```

```
$ sudo ./acetooll USBw 00000000
```

```
Status: APP
```

```
Running command: USBw – Data: 4
```

```
Executing command
```

```
Result is: 00 00 06 20    20 06 00 00 Stack pointer reset value
```

```
Status: APP
```



```
$ sudo ./acetool USBw 00000000
Status: APP
Running command: USBw – Data: 4
Executing command
Result is: 00 00 06 20    20 06 00 00 Stack pointer reset value
Status: APP
```

We can read (and write) arbitrary memory!

Time to dump

[illegible]

image_combined.bin.bndb — Binary Ninja 4.0.4958-Stable

image_combined.bin.bndb

Types Search types

Name Size

User Types: image_combined.bin.bndb

command_handler 0x8

System Types: image_combined.bin....

Platform: thumb2

```
struct command_handler __packed
{
    char command[0x4];
    int32_t (* handler)(int32_t cmd,
        int32_t* args);
};
```

Cross References

Filter (191)

Data References {10}

- 00000008 void* data_8
- 00000010 void* data_10
- 00000014 void* data_14
- 00000018 void* data_18
- 0000001c void* data_1c
- 00000020 void* data_20
- 00000024 void* data_24
- 00000028 void* data_28
- 00000030 void* data_30
- 00000034 void* data_34

0x0

Architecture: thumb2

Segments:

- r-x 0x00000000-0x00058000
- 0x00058000-0x00058014
- rwX 0x20040000-0x20058000

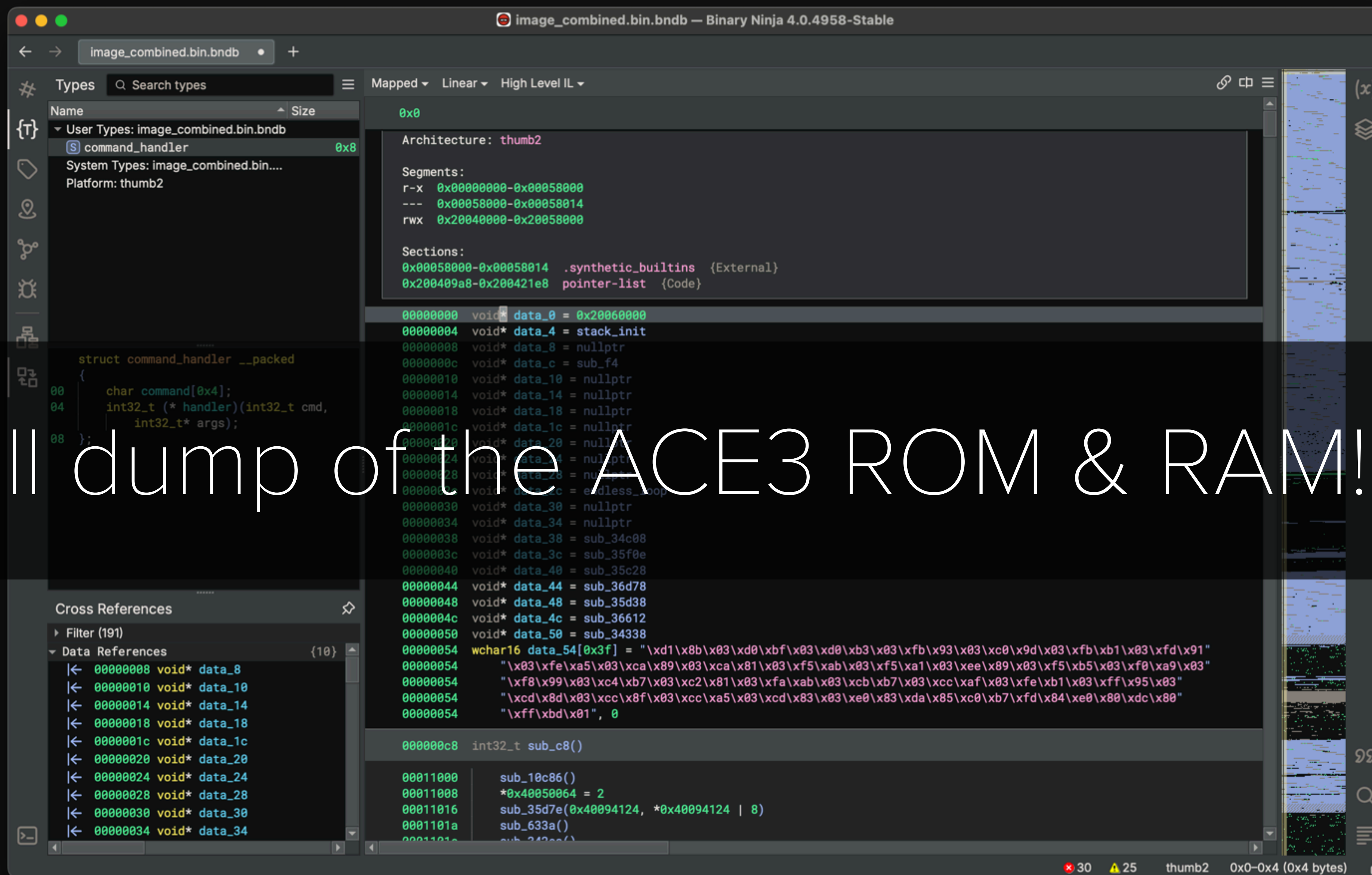
Sections:

- 0x00058000-0x00058014 .synthetic_builtins {External}
- 0x200409a8-0x200421e8 pointer-list {Code}

```
00000000 void* data_0 = 0x20060000
00000004 void* data_4 = stack_init
00000008 void* data_8 = nullptr
0000000c void* data_c = sub_f4
00000010 void* data_10 = nullptr
00000014 void* data_14 = nullptr
00000018 void* data_18 = nullptr
0000001c void* data_1c = nullptr
00000020 void* data_20 = nullptr
00000024 void* data_24 = nullptr
00000028 void* data_28 = nullptr
0000002c void* data_2c = endless_loop
00000030 void* data_30 = nullptr
00000034 void* data_34 = nullptr
00000038 void* data_38 = sub_34c08
0000003c void* data_3c = sub_35f0e
00000040 void* data_40 = sub_35c28
00000044 void* data_44 = sub_36d78
00000048 void* data_48 = sub_35d38
0000004c void* data_4c = sub_36612
00000050 void* data_50 = sub_34338
00000054 wchar16 data_54[0x3f] = "\xd1\x8b\x03\xd0\xbf\x03\xd0\xb3\x03\xfb\x93\x03\xc0\x9d\x03\xfb\xb1\x03\xfd\x91"
00000054 "\x03\xfe\xa5\x03\xca\x89\x03\xca\x81\x03\xf5\xab\x03\xf5\xa1\x03\xee\x89\x03\xf5\xb5\x03\xf0\xa9\x03"
00000054 "\xf8\x99\x03\xc4\xb7\x03\xc2\x81\x03\xfa\xab\x03\xcb\xb7\x03\xcc\xaf\x03\xfe\xb1\x03\xff\x95\x03"
00000054 "\xcd\x8d\x03\xcc\x8f\x03\xcc\xa5\x03\xcd\x83\x03\xe0\x83\xda\x85\xc0\xb7\xfd\x84\xe0\x80xdc\x80"
00000054 "\xff\xbd\x01", 0
000000c8 int32_t sub_c8()
00011000 sub_10c86()
00011008 *0x40050064 = 2
00011016 sub_35d7e(0x40094124, *0x40094124 | 8)
0001101a sub_633a()
0001101c sub_242cc()
```

30 25 thumb2 0x0-0x4 (0x4 bytes)

Full dump of the ACE3 ROM & RAM!



Dumping unknown silicon is possible



- And it's not super difficult
- We can get code-execution without having the firmware
- We can now start researching the ACE3!

Thank you!



hextree.io