Debug Resurrection on nRF52 Series

LimitedResults

BlackHat Europe 2020

About

Me

- www.limitedresults.com
  - Hardware Security
  - Low-Level vulns
- No affiliation

Agenda

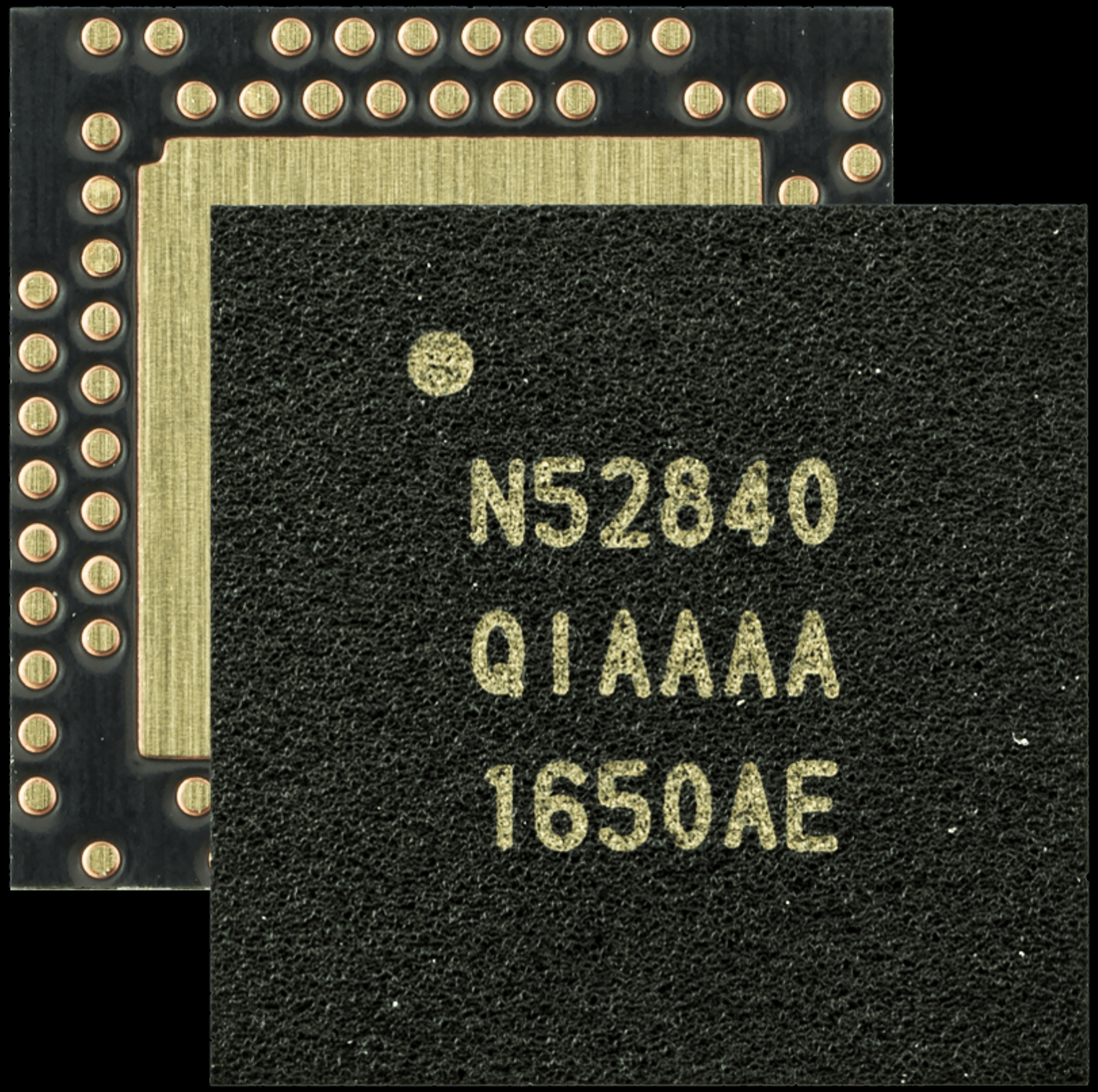
Today

- Introduction
  - nRF52
  - APPROTECT
- Approach & Analysis
- Time to Hack
- Results
- Conclusion

INTRO

NordicSemiconductor
nRF52

- Released in 2015
- Popular IoT Platforms
- Short Range communications
  - Bluetooth, Zigbee, Thread products
- But generally
  - Debug interface (SWD) is disabled :(



Code Readout Protection
What is it?

- Hardware Security Mechanism
  - Present in most of the MCUs/SoCs with integrated Flash nowadays
- Protect against Reverse–Engineering
  - Prevent an attacker to access the Code/Data stored in Flash
  - Protect hardcoded secrets present in the Firmware

Previously…

Long tradition of CRP attacks

- Microchip PIC
  - Heart of Darkness – Milosch Meriac
  - Hacking the PIC 18F1320 – Andrew Huang aka Bunnie
- NXP LPC
  - Breaking Code Read Protection on the NXP LPC-family Microcontrollers – Chris Gerlinsky
  - Breaking boot loader on the Cheap – Qais Temeiza, David Oswald
- STMICRO STM32
  - Shedding too much Light on a Microcontroller's Firmware Protection – J. Obermaier, S. Tatschner
  - Wallet.fail – D. Nedospasov, J. Datko, T. Roth
- Not exhaustive list of course…
- Some professional companies propose FW extraction as a Service

And what about Nordic MCUs?
nRF51

- RBPCONF Protection
  - Prevent direct access to the Flash and the RAM
  - But ALL the registers are still R/W…
  - And the debugger can still control the Code Flow Execution
- In 2015, IncludeSecurity disclosed how to bypass RBPCONF
  - Just find a gadget (load word instruction with a register operand)
  - Set the operand register to a target address and execute that one instruction
  - Read the value from the register.
  - Script this using OpenOCD. Dumped.

APPROTECT

nRF52

- On nRF52
  - Nordic decided to design a more restricted security mechanism
  - This new feature is called APPROTECT
    - Prevent direct access to the Flash and the RAM, but also to all the registers
    - Not too much details in nRF52 documentation (security by obscurity)
  - Never hacked (As Far As I Know)

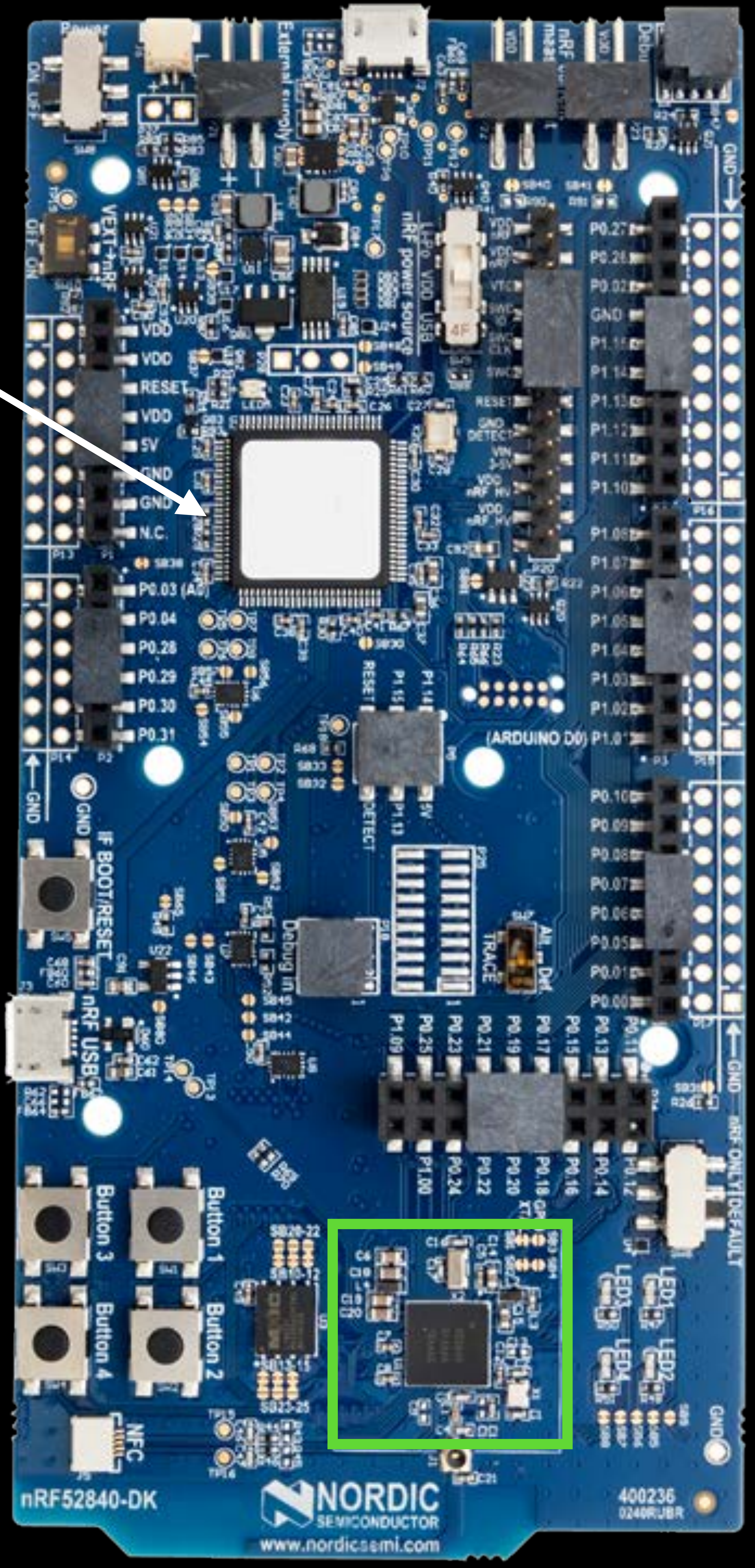
My Objective
Defeat the APPROTECT

- Context
  - 2 months (Lockdown)
- Main Objective
  - Find a way to break the APPROTECT on nRF52840
- How to do that?
  - I don’t know yet…

THE LIMITED APPROACH

The Dev-Kit

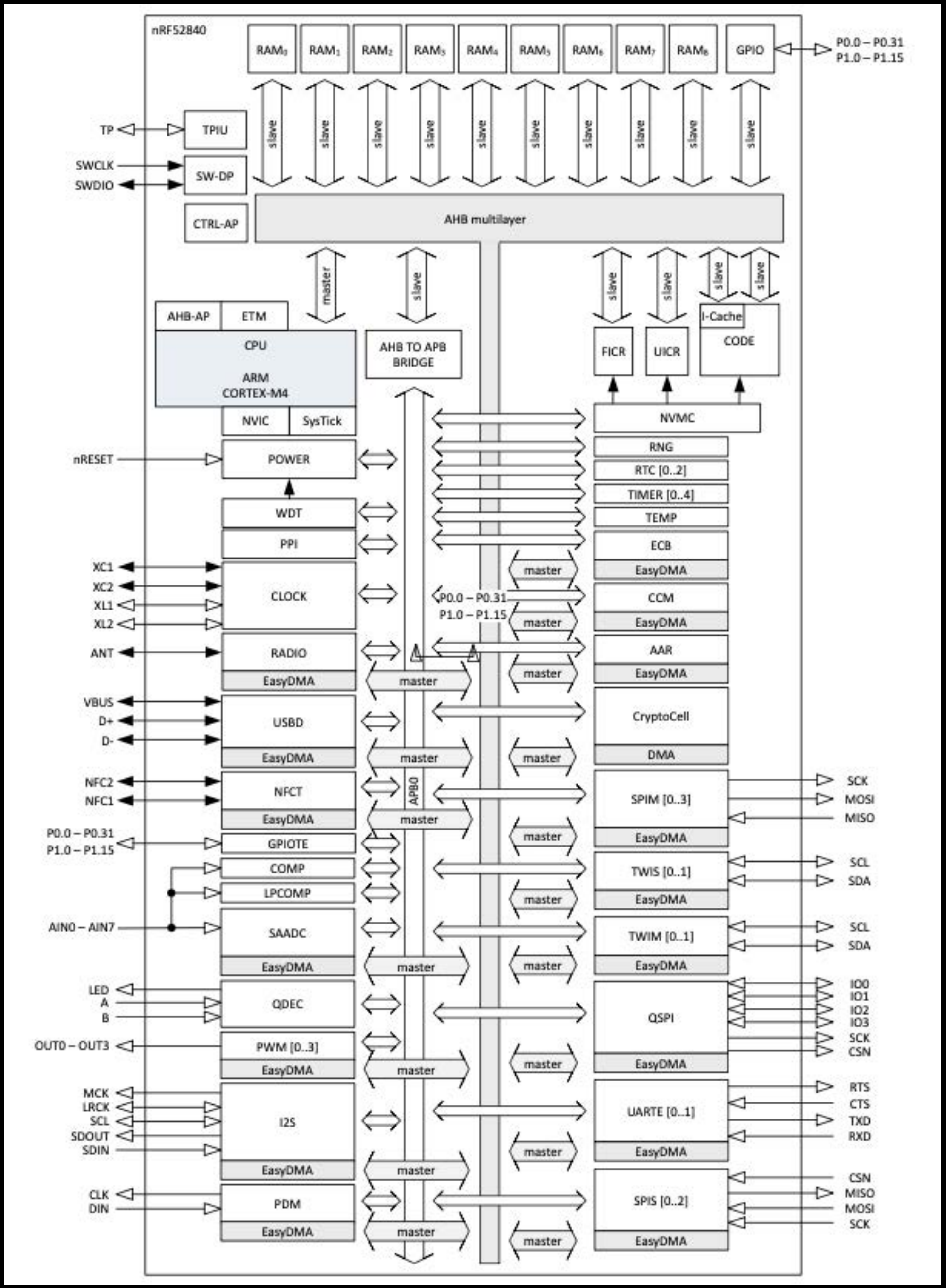
nRF52840-DK



- 50 Euros :\
- Download the SDK (nRF5_SDK_16.0.0)
- Install Nrfjprog (v10.6.0)
- Embedded Segger J-Link debugger
  - Based on a MicroChip ATSAM3U2C (white sticker)
  - Install debugger driver JLink (v6.64)
  - Can be used independently :)

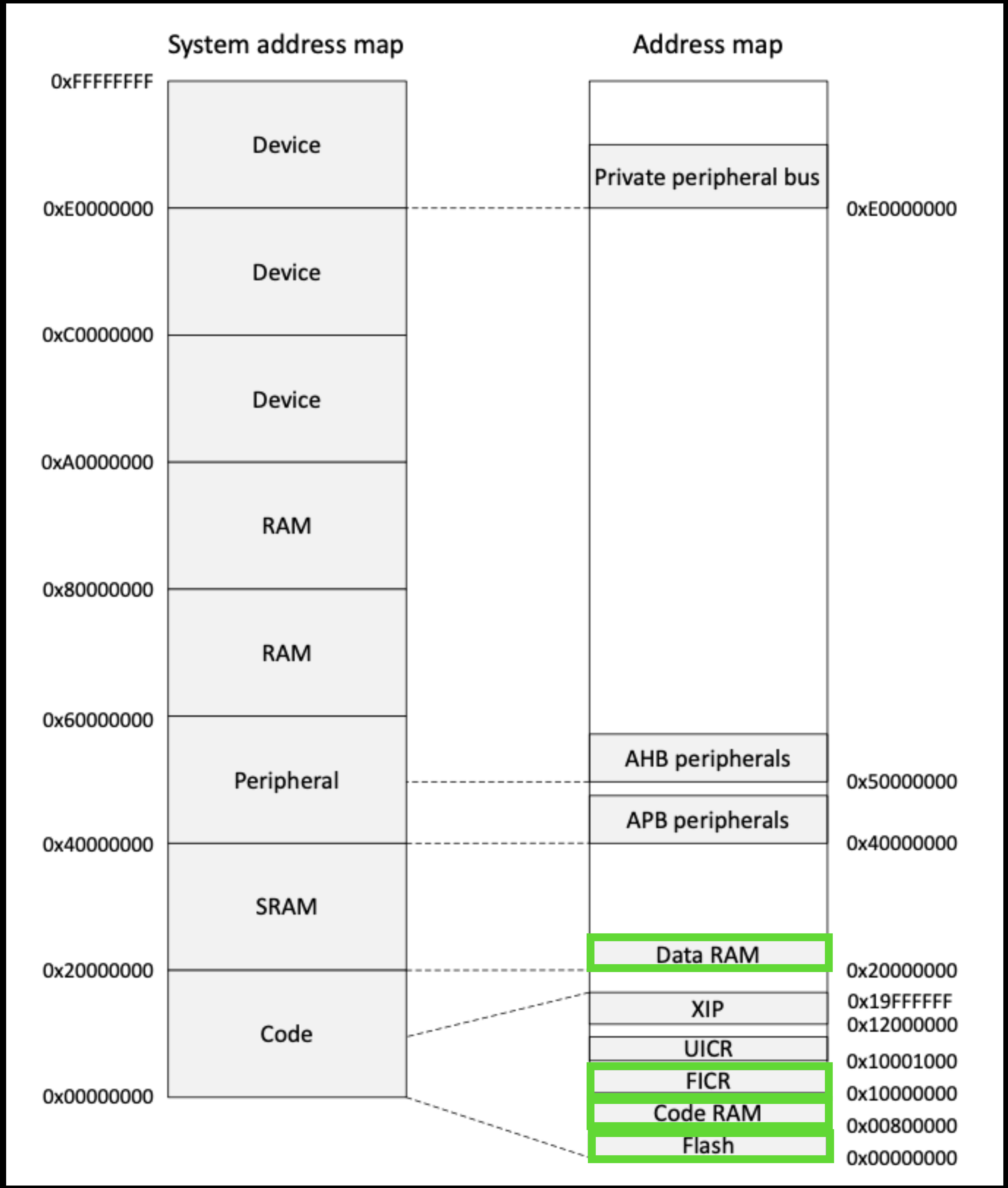
The Target
nRF52840

- Nordic nRF52840
  - Bluetooth, Thread and Zigbee SoC
  - 64 MHz Cortex-M4F CPU
  - Integrated 1MB Flash, 256kB RAM
  - Security
    - ARM Crypto-cell
    - Code Readout Protection (APPROTECT)



Memory Map
nRF52840

- Flash Memory located at 0x00000000
- Physical RAM is mapped to both the Data RAM and the Code RAM regions
  - Code RAM @0x00800000–0x00840000
  - Data RAM @0x20000000–0x20040000
- Factory information configuration registers (FICR) at 0x1000 0000
  - Chip Specific information Pre-programmed in Factory like Device ID, Encryption Root, Identity Root, device address (bluetooth)
- User Information Configuration Registers (UICR) at 0x10001000



UICR

User Information Configuration Registers

- Non-Volatile Memory (NVM) Registers to configure Specific Settings

- APPROTECT is mapped at 0x10001208
  - Write 0xFFFFFF00 enable the Access Port Protection
  - Cannot be disabled without erase all the RAM and Memory Flash (Nordic)

4.5.1.5 APPROTECT

Address offset: 0x208

Access port protection

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID																									A	A	A	A	A	A	A	A
Reset 0xFFFFFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

ID	Acce	Field	Value ID	Value	Description
A	RW	PALL			Enable or disable access port protection. See Debug and trace on page 50 for more information.
			Disabled	0xFF	Disable
			Enabled	0x00	Enable

Let’s start
nRF52

- Compile/Reuse a sample Code (SDK)
- Connect the board via USB
- Flash the hex into the nRF52
    - $ nrfjprog -f NRF52 --program nrf_pca10056.hex --verify --chiperase && nrfjprog -f NRF52 —reset
    - Or $ make && make install (inside the SDK)
- Verify the debug access by reading the APPROTECTSTATUS register via OpenOCD
    - > nrf52.dap apreg 1 0x0c
        - 0x00000000 APPROTECT enabled
        - 0x00000001 APPROTECT disabled

APPROTECT
Command Line

- Enabling (via OpenOCD)
  - $ openocd -s /usr/local/share/openocd/scripts -f ./interface/jlink.cfg -c "transport select swd" -f ./target/nrf52.cfg
  - *# Telnet*
    - $ telnet localhost 4444
      - > flash fillw 0x10001208 0xFFFFFF00 0x01
      - > reset
- Enabling (via nrfjprog)
  - $ nrfjprog --memwr 0x10001208 --val 0xFFFFFF00

- Disabling (once connected to OpenOcd & telnet)
  - *#Write ERASEALL register*
  - > nrf52.dap apreg 1 0x04 0x01
  - > reset
- Disabling (via nrfjrog)
  - $ nrfjprog -f NRF52 —recover

APPROTECT Enabled

nRF52

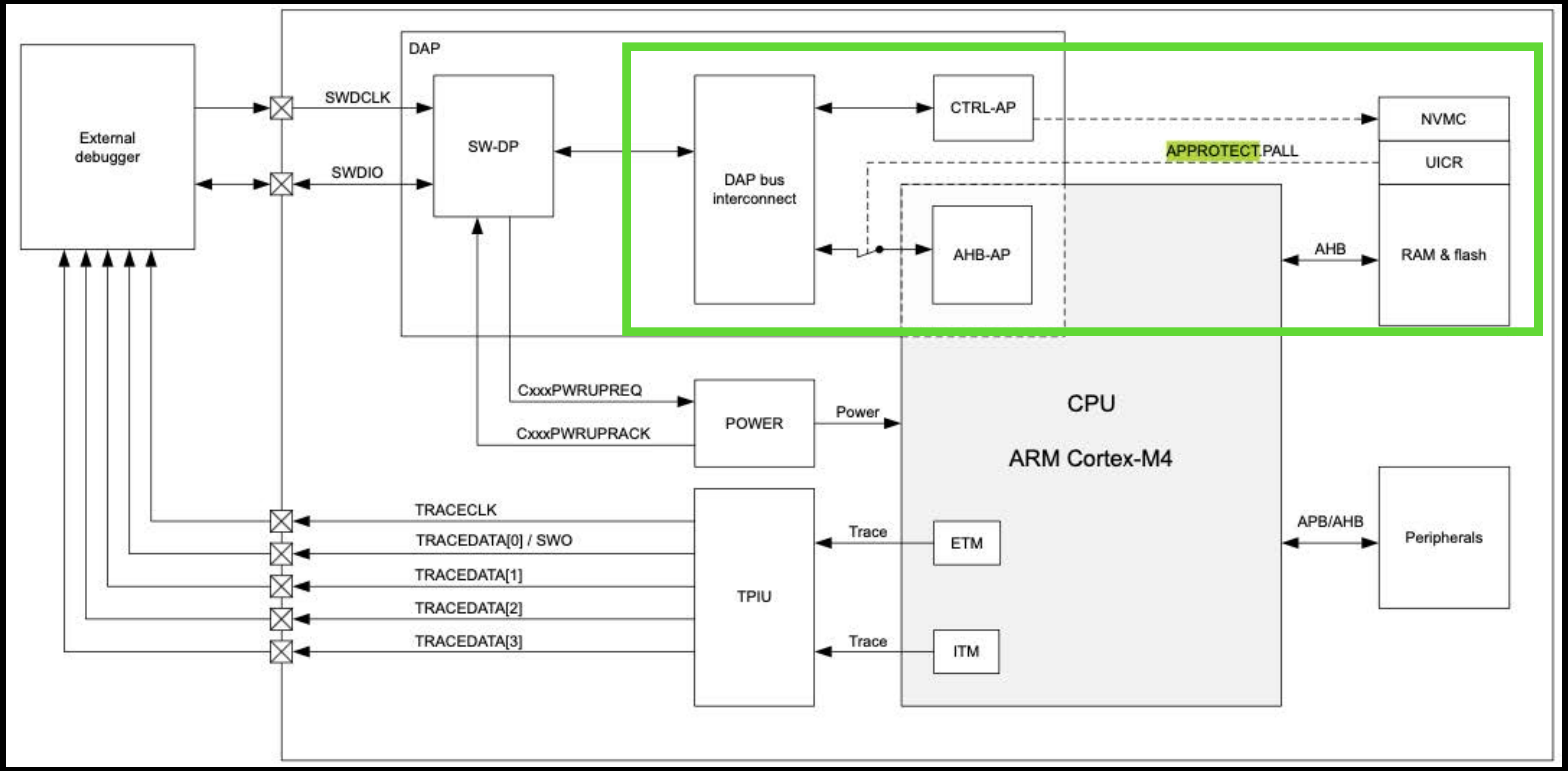
- Once APPROTECT enabled
  - Could not Find MEM-AP to control the Core

```
xPack OpenOCD, 64-bit Open On-Chip Debugger 0.10.0+dev (2019-07-17-11:25)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
swd
Info : J-Link OB-SAM3U128-V2-NordicSemi compiled Jan 21 2020 17:30:48
Info : Hardware version: 1.00
Info : VTarget = 3.300 V
Info : clock speed 1000 kHz
Info : SWD DPIDR 0x2ba01477
Error: Could not find MEM-AP to control the core
Info : Listening on port 3333 for gdb connections
Error: Target not examined yet
```

APPROTECT Analysis

ARM-DAP

- Accessing ARM CPU throught the Debug Access Port (DAP)
  - CRTL-AP
    - Master Debug Port
    - Not dependent of the APPROTECT
  - AHB-AP
    - Access Memories and Control the CPU via SWD
    - This is the “Real” Debug Port
- ARM Cortex-M reference pretty useful here



Boot Process

Relatively simple

- No BootROM
  - no embedded boot-loader routines, nor IAP/ISP routines to reverse
- Power Sequence
  - No info in the documentation
- Reset Vector
  - Entry Point located at 0x0000 02B4

```
                          Reset
00000004 b5 02 00 00          addr        DAT_000002b5
                          NMI
00000008 dd 02 00 00          addr        DAT_000002dd
                          HardFault
0000000c df 02 00 00          addr        DAT_000002df
                          MemManage
00000010 e1 02 00 00          addr        DAT_000002e1
```

Reset Policy
Understand the power sequence

Reset source	Reset target								
	CPU	Peripherals	GPIO	Debug[a]	SWJ-DP	RAM	WDT	Retained registers	RESETREAS
CPU lockup [6]	x	x	x						
Soft reset	x	x	x						
Wakeup from System OFF mode reset	x	x		x [7]		x [8]			
Watchdog reset [9]	x	x	x	x		x	x	x	
Pin reset	x	x	x	x		x	x	x	
Brownout reset	x	x	x	x	x	x	x	x	x
Power-on reset	x	x	x	x	x	x	x	x	x

- Consequently
  - at Power-on Reset, the AHB-AP has to initialise itself depending on the value of the APPROTECT, which is stored in UICR

The Limited Plan

- Goal
  - Access to the AHB-AP Debug, despite of the APPROTECT
- I know
  - No BootROM in nRF52
  - At boot-up, the CPU/AHB-AP Block has to receive the APPROTECT value stored in Flash from NVMC to set the Protection State accordingly
  - This is implemented in Pure Hardware and has to be done before the CPU start to load and execute Code stored in Flash
- What am I going to do?
  - Fault Injection after a Power-On-Reset, when the NVMC/AHB-AP are initialised with the UICR values

Time To Hack

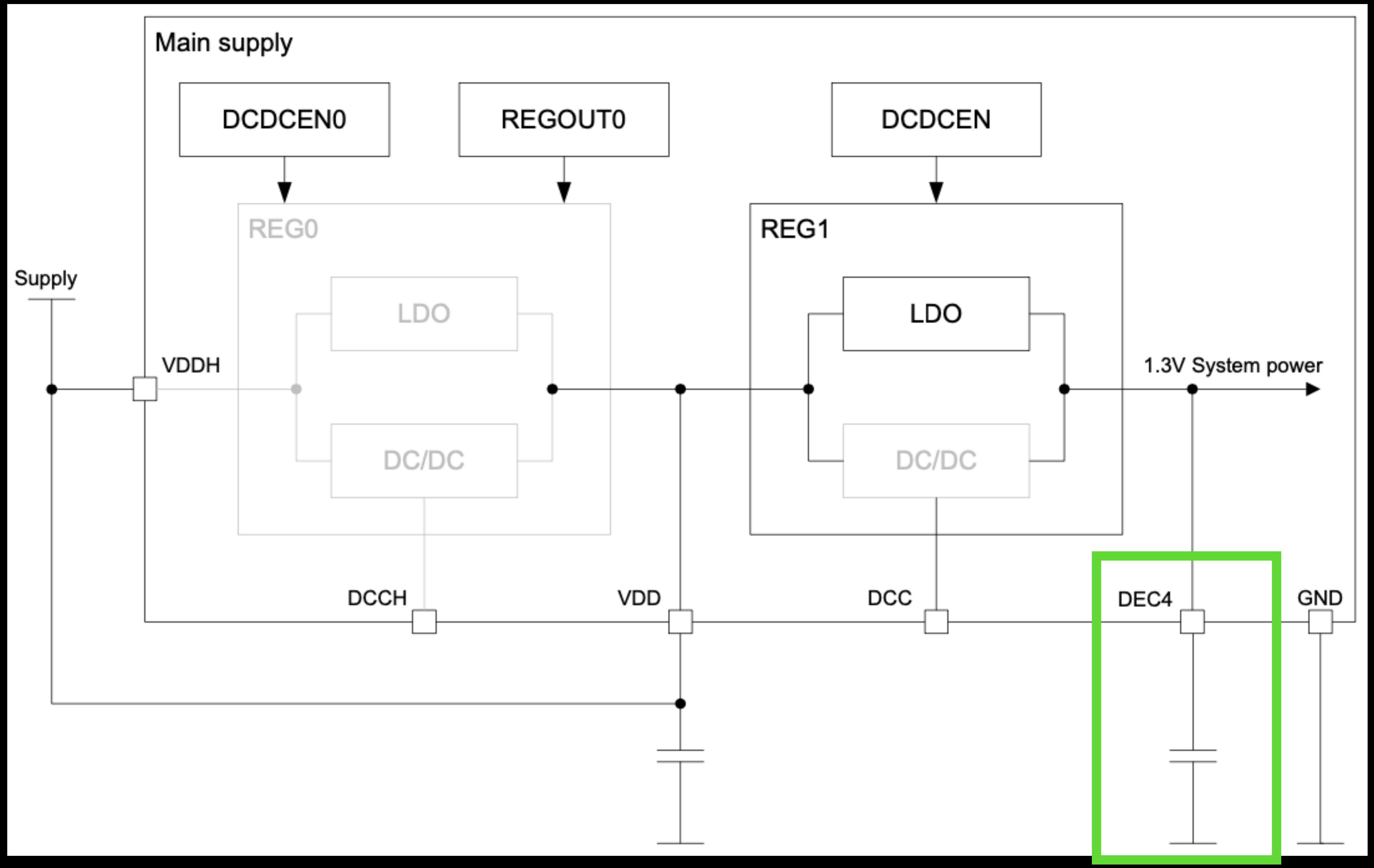
Fault injection
Voltage glitching

- The cheapest Fault Injection technique
- Perturb the Power Supply to induce a fault during critical SW/HW operations
  - Skip instruction, Data/Code modification
  - Difficult to predict the glitch effect and to defend against
- Commercial Tools are available…but you can also DIY
- Lot of Public resources
  - Glitching for Noobs – Exide
  - Glitching and Side Channel Analysis for all – Colin O'Flynn
  - And more…

nRF52 Power Domain

Which line do I need to glitch?

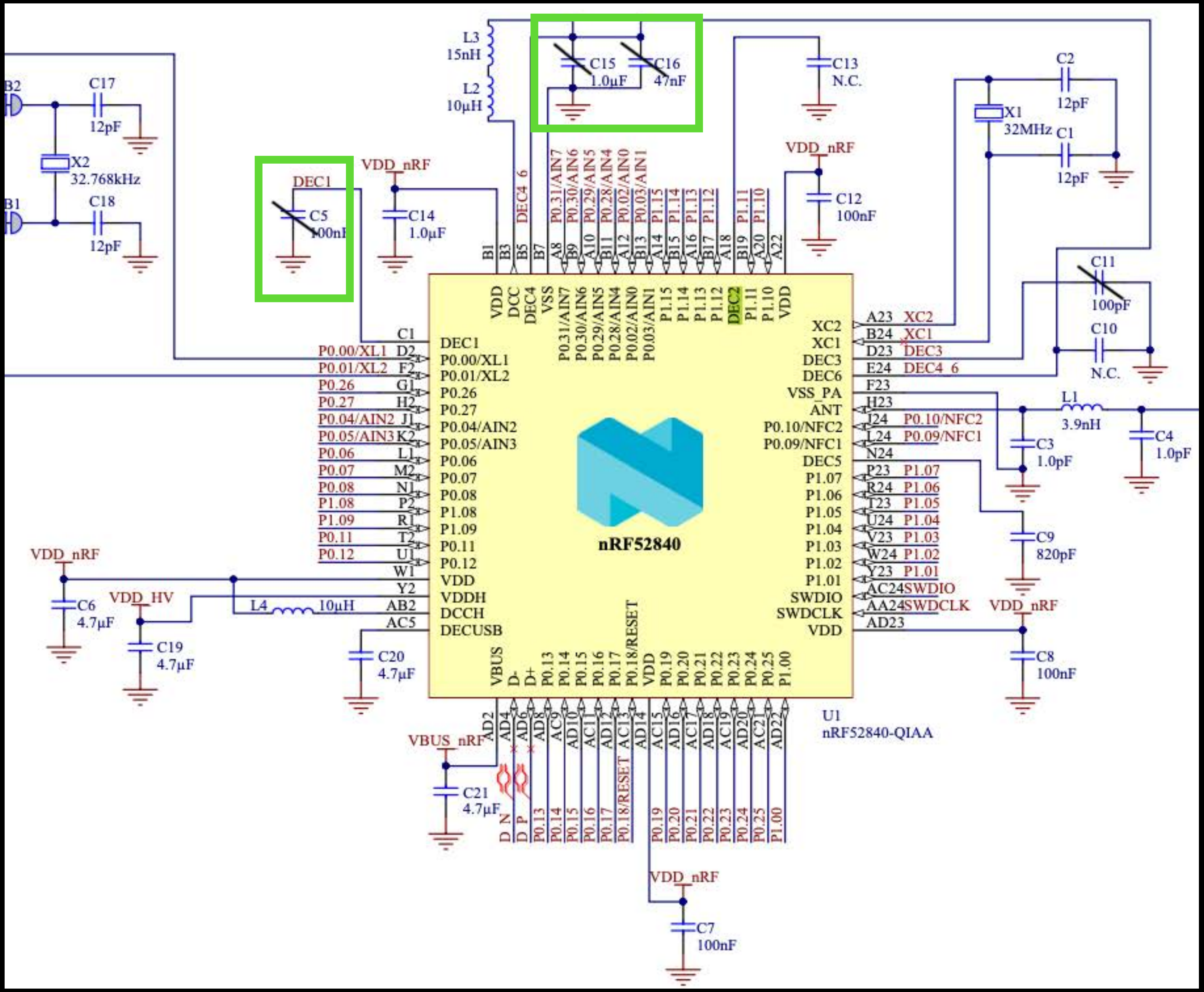
- Six different Power Pins, zero info from Nordic
  - DEC1 = 1.1V Regulator supply decoupling
  - DEC2 = 1.3V Regulator supply decoupling
  - DEC3 = Power Supply, decoupling
  - DEC4 = 1.3V Regulator supply decoupling
  - DEC5 = 1.3V Regulator supply decoupling
  - DEC6 = 1.3V Regulator supply decoupling



- Probing these lines lead to
  - DEC4 (1.2V–1.3V) is the power after the REG1 Stage, which supplies the digital blocks (CPU and Memories)
  - DEC1 (0.8V–0.9V) is the CPU dedicated power line

PCB modification

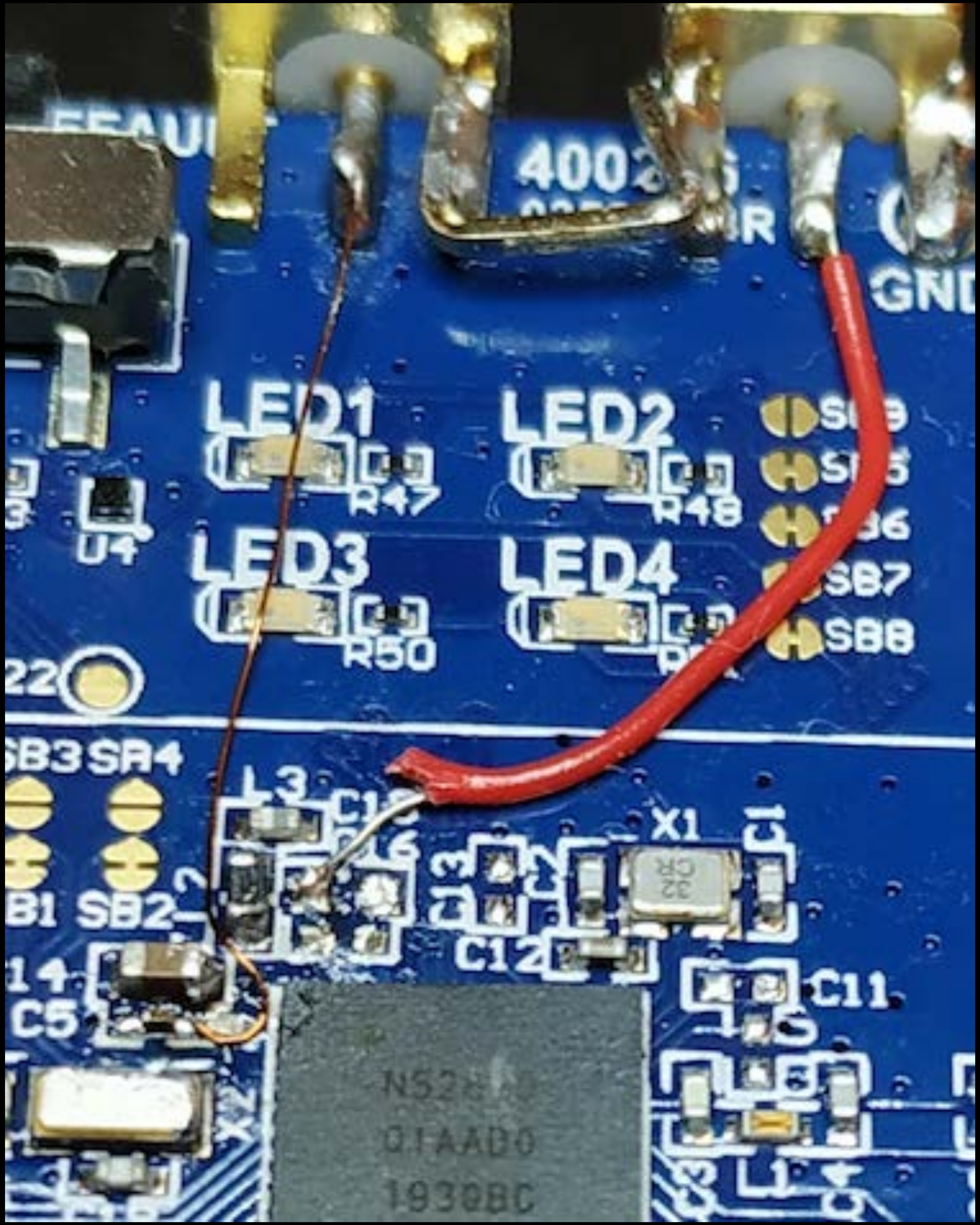
- Schematics available, so easy job
  - Focus on DEC1 and DEC4
- Why do I remove capacitors?
  - Improve the monitoring of power consumption. No big low-pass filter effect (RC)
  - The voltage glitch has a sharper drop-out. Better repeatability and fine-tuning parameters



PCB Modification

Soldering

- Magnet wire soldered to DEC1 then connected to a SMA connector
- Red wire soldered to DEC4 then connected to a second SMA connector
- GND point as a main Ground



The attack flow

- nRF52 Boot-up
- Inject the fault
- Attempt a debugger connection
  - $ openocd -s /usr/local/share/openocd/scripts -f ./interface/jlink.cfg -c "transport select swd" -f ./target/nrf52.cfg -c "init;dump_image nrf52_dumped.bin 0x0 0x100000"
- Check Status (if OK, that will dump the firmware)
- If not, Reset the chip and try again

Final Setup
Automation is key

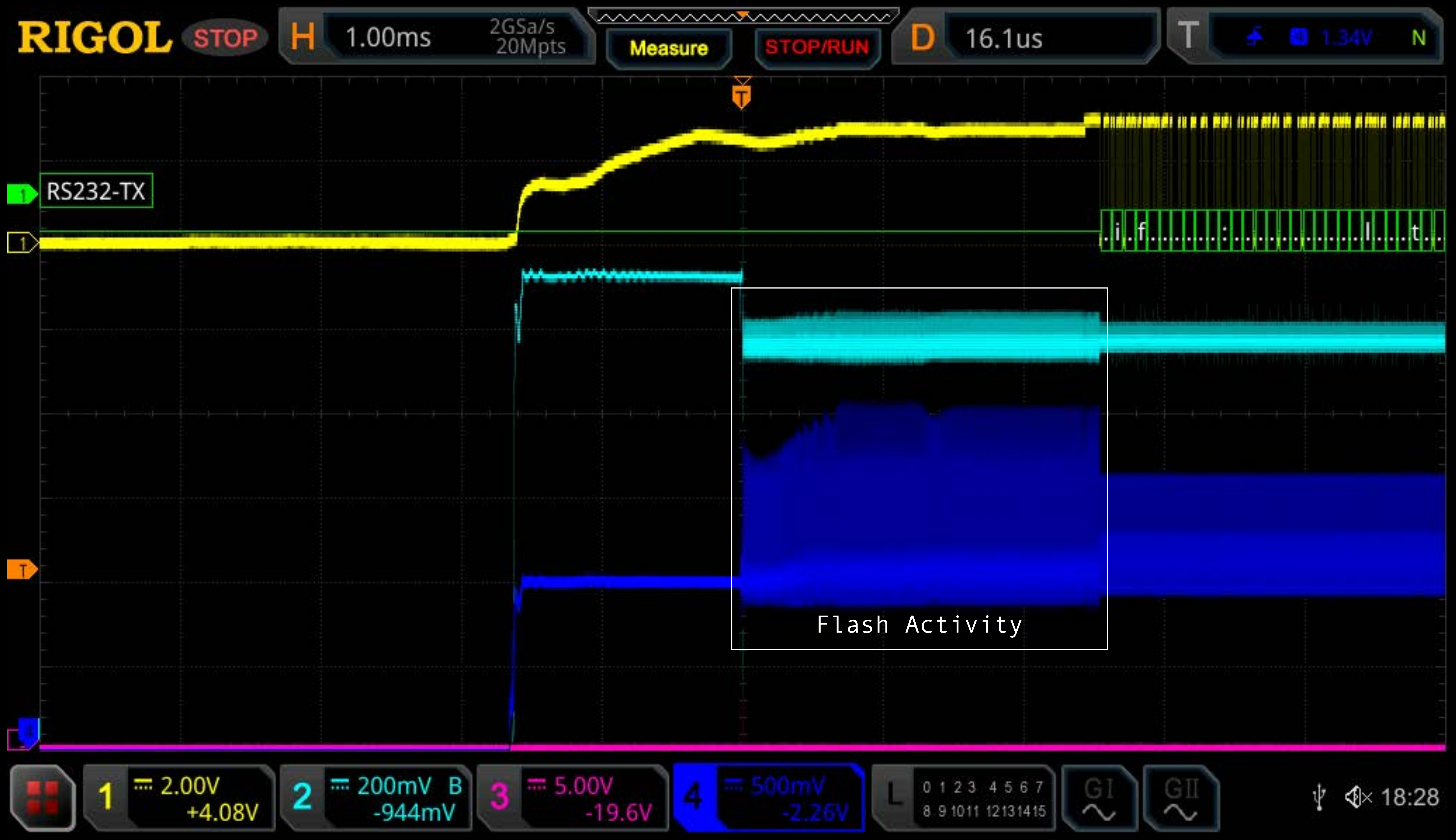
- Home-made Glitcher System
  - Based on Mosfet, passive components and SMA connectors for inputs/outputs (5$)
  - Synchronised by Scope
- USB commands to set the different parameters like Delay, Width and Amplitude
- Fully controlled in Python

RESULTS

Black Box Reverse

Power analysis is the only way

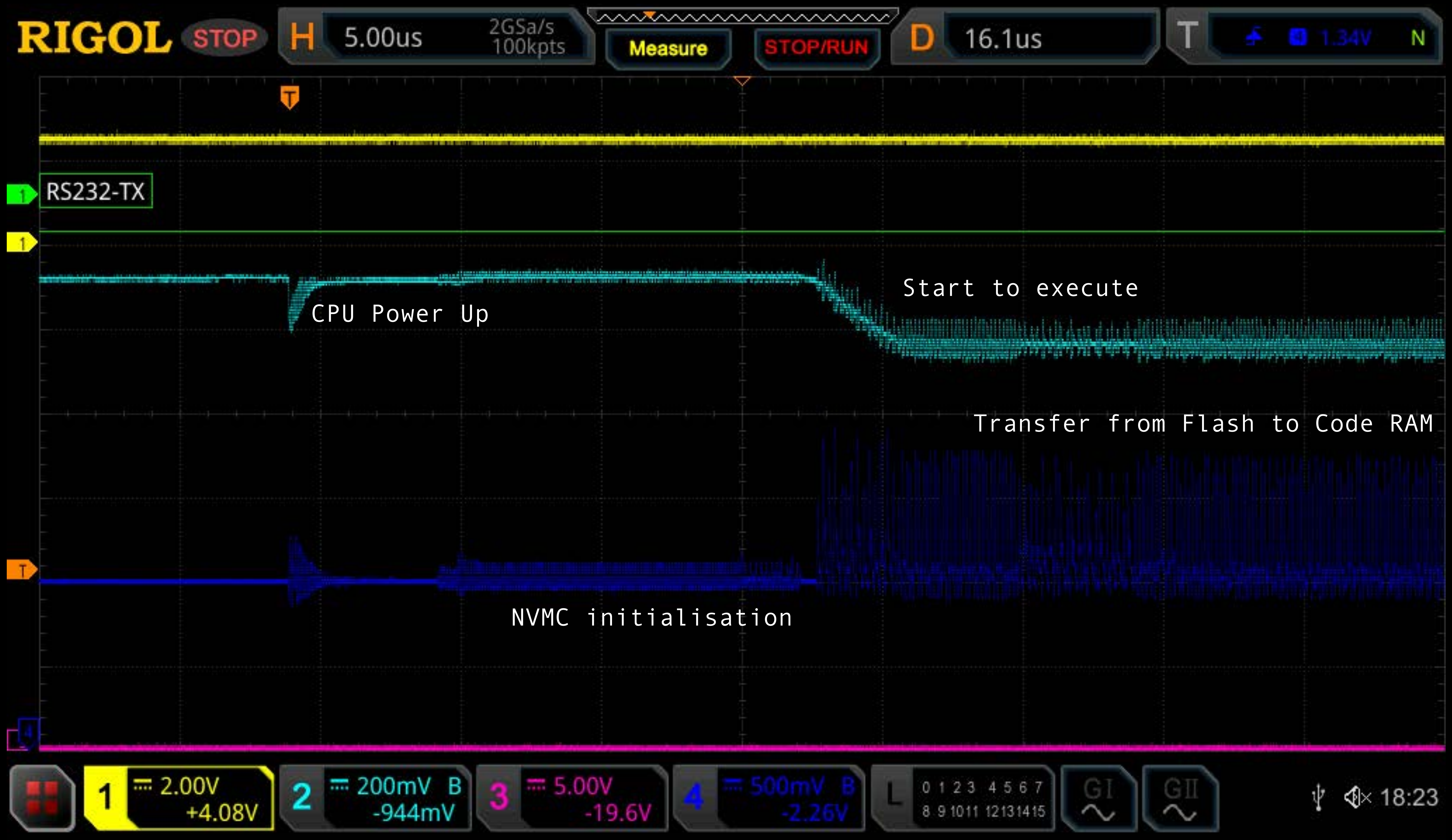
- CH1 = TxD (pin P0.06)
- CH2 = Power consumption through DEC1
- CH3 = Pulse command to trig the Glitch
- CH4 = Power consumption through DEC4 (Scope Trigger)



Black Box Reverse

Identification of the targeted Hardware Process

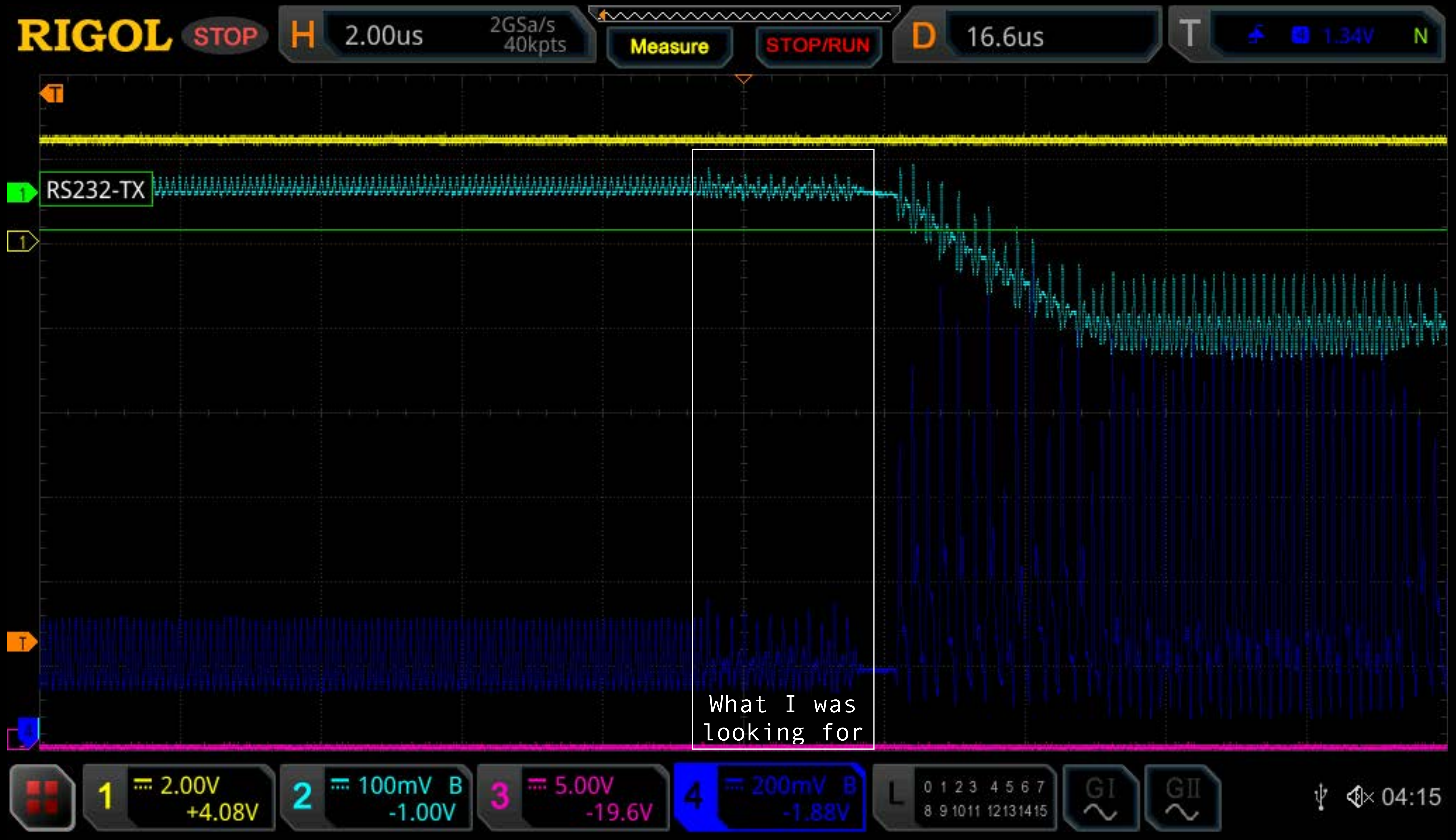
- Identification
  - CPU Power-Up
  - NVMC Init
  - Flash Activity



Black Box Reverse

Identification of the targeted Hardware Process

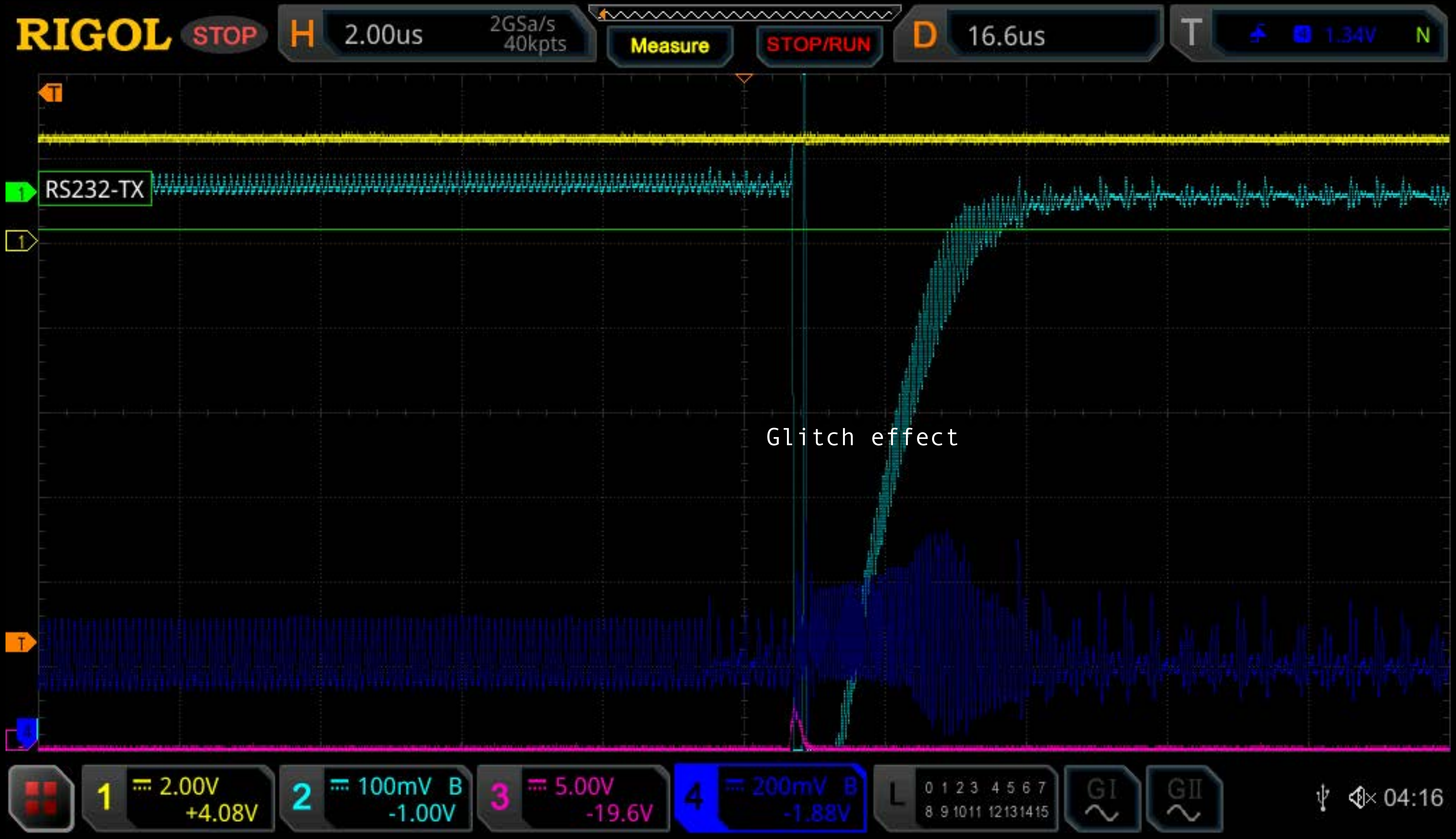
- Hardware Process identified
- Visible on DEC1 and DEC4
- Timing window of 2.5 us



Results

Bypass of APPROTECT on nRF52840

- Successful glitch
- Differences on both Power Lines
- No Flash Activity after



Results

Debug access despite APPROTECT enabled

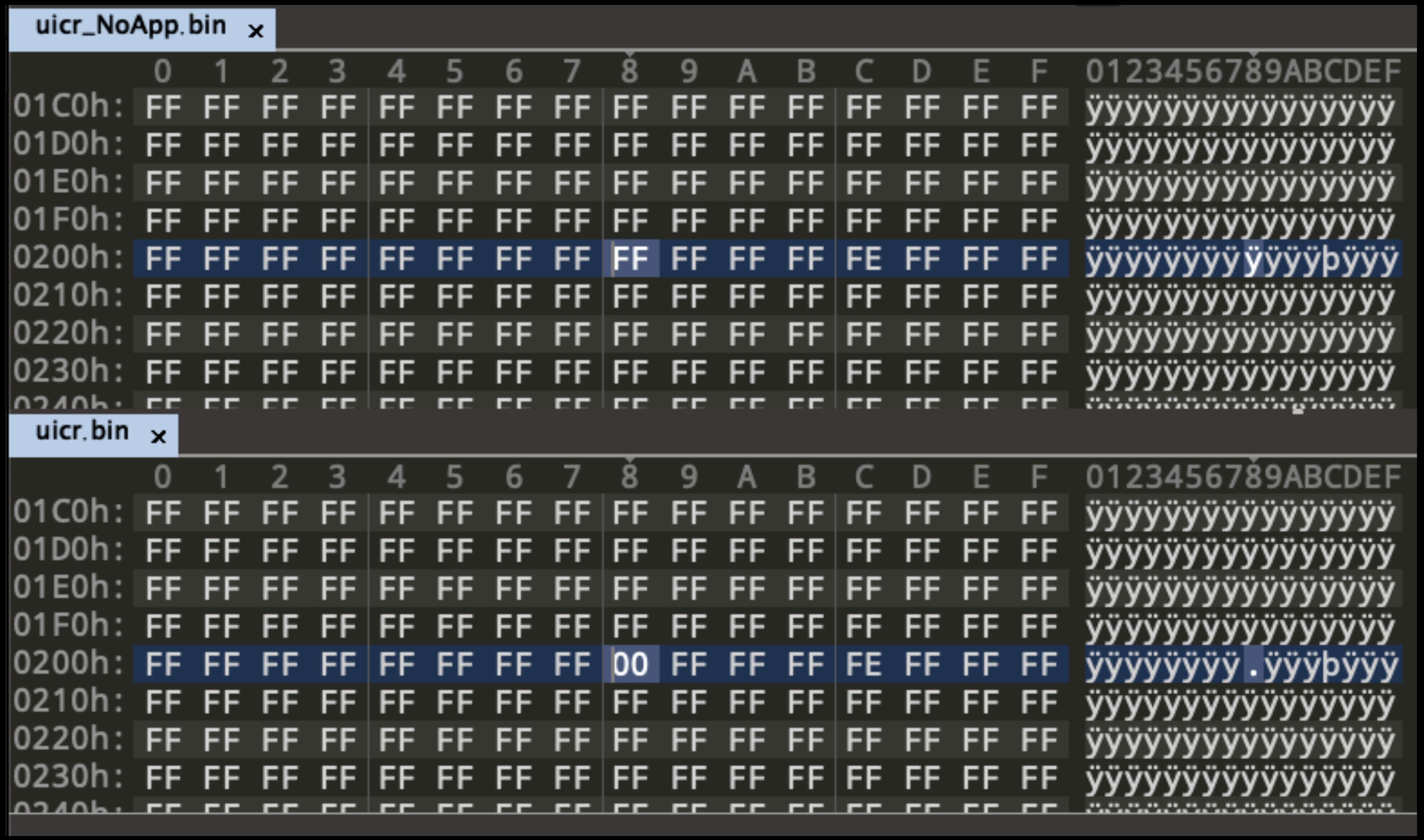
- The Debugger can now connect to the AHB–AP bus and enumerate the breakpoints and watchpoints
  - Classic debug session openOCD + arm–none–eabi–gdb

```
 ----- APPROTECT BYPASS 0 -----
#### OpenOcd test ####
xPack OpenOCD, 64-bit Open On-Chip Debugger 0.10.0+dev (2019-07-17-11:25)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
swd
Info : J-Link OB-SAM3U128-V2-NordicSemi compiled Jan 21 2020 17:30:48
Info : Hardware version: 1.00
Info : VTarget = 3.300 V
Info : clock speed 1000 kHz
Info : SWD DPIDR 0x2ba01477
Info : nrf52.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : nrf52.cpu: external reset detected
Info : Listening on port 3333 for gdb connections
```

Persistence

Reactivating the debug interface permanently

- Dump Flash and UICR
  - >(gdb) mon dump_impage flash.bin 0x10000000 0x1000
  - >(gdb) mon dump_impage uicr.bin 0x10001000 0x1000
- Patch 0x00 -> 0xFF in the UICR.bin at 0x10001208 in your hex editor
- Erase all
  - nrfjprog -f NRF52 --recover
- Reflash the nRF52 with uicr.bin (and flash.bin)



Consumer Product
Logitech G Pro

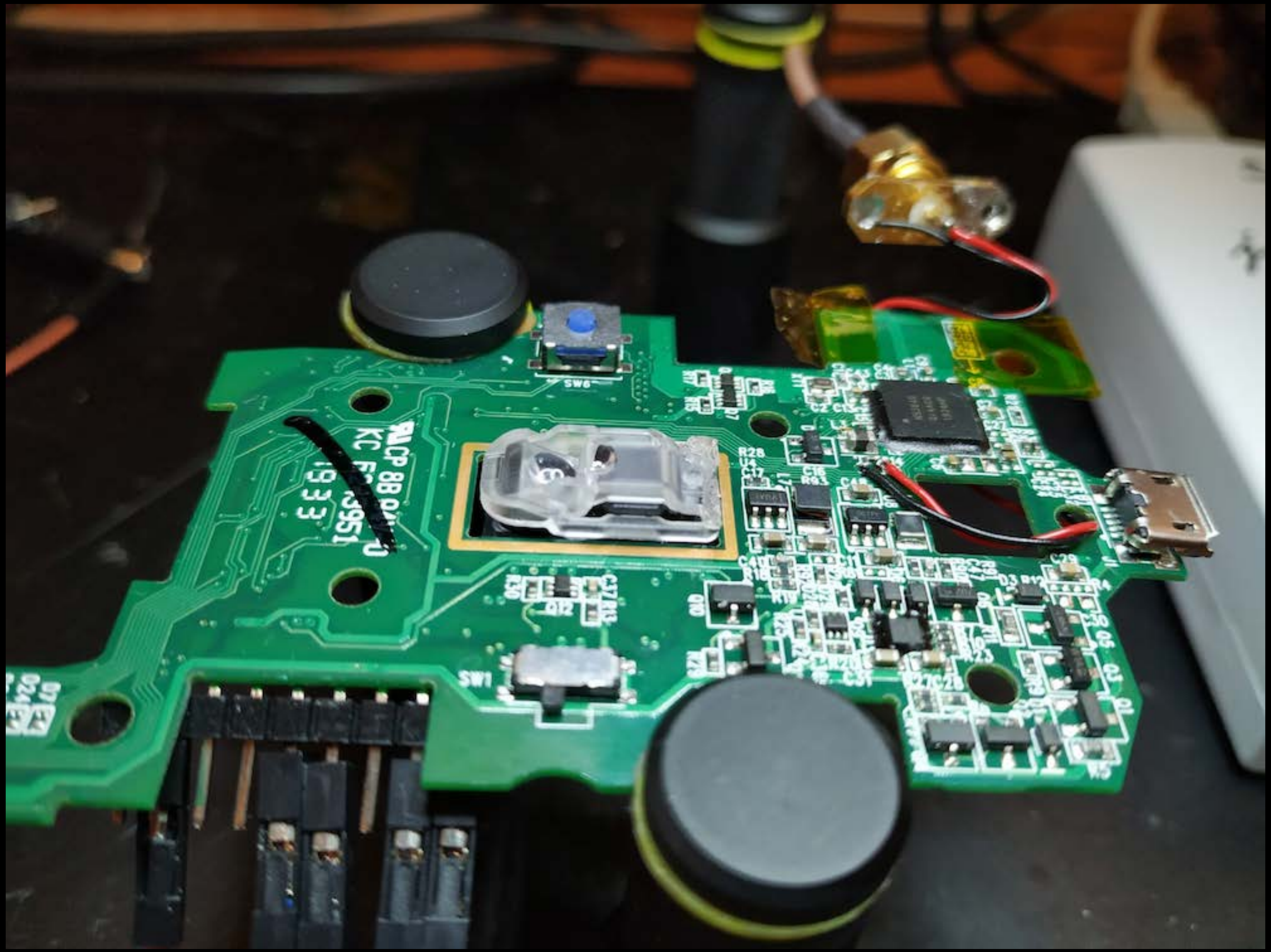
- Need to validate this on a real product
- Sacrifice of my G Pro
  - 120 Euros :\…rip
  - Based on nRF52840
  - APPROTECT is activated
- *The goal is not to attack the Logitech Product here*

Soldering
Logitech G Pro

- PCB Reference design
  - provided by Nordic
  - Reused by Logitech
  - Silkscreen
- No need to reverse PCB
  - Easy job

Final Setup

- DEC1 is connected to the glitcher
  - Successful glitch attack using the known parameters
- Firmware is Dumped
- UICR is rewritten to add Debug Persistence to the Device
- Reset



Results
Attack is Validated

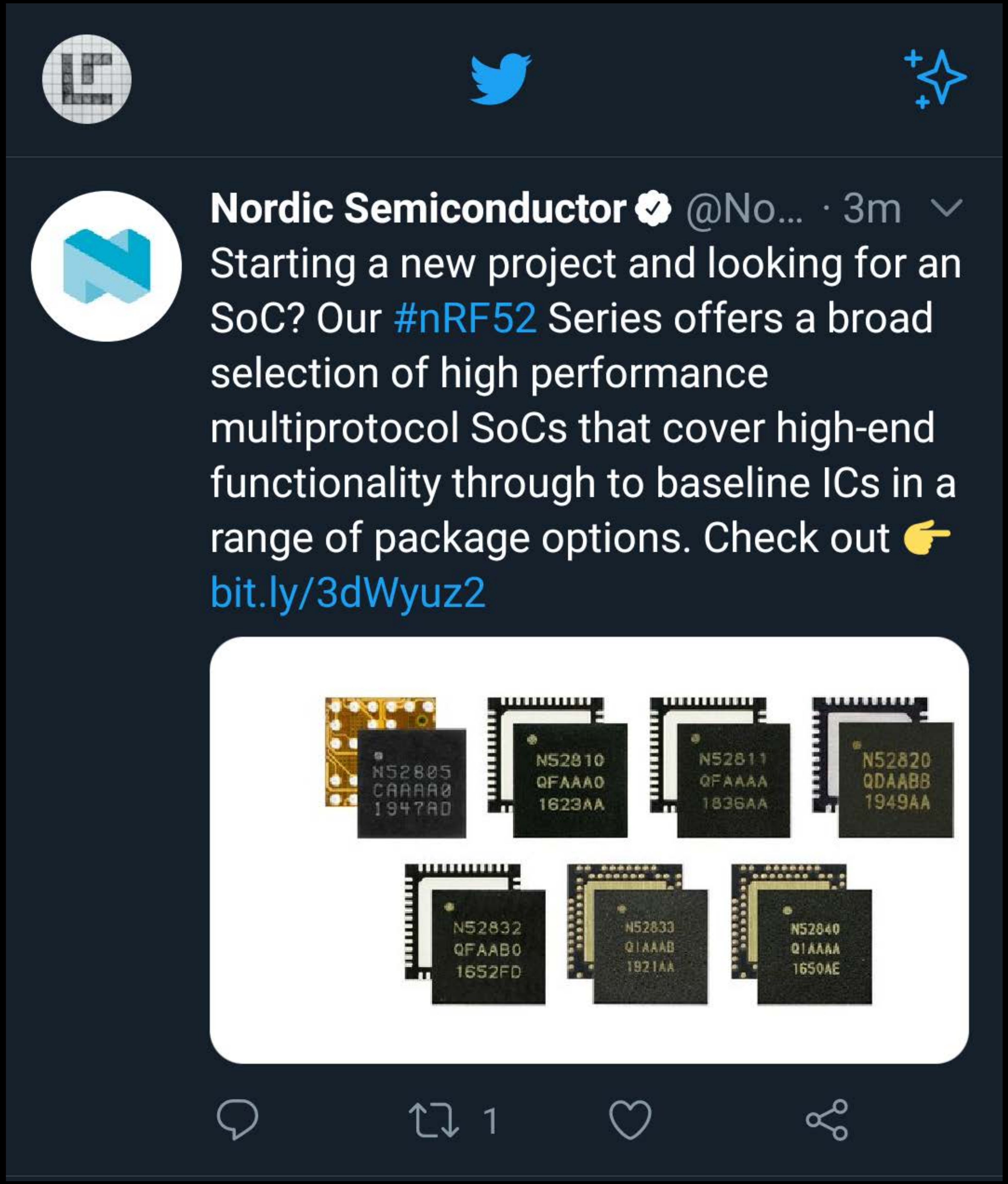
- End up with a full-functional device
- Ideal conditions for
  - Static analysis (firmware is dumped)
  - Dynamic analysis (full debug on device)
    - Attach debug to IDA
    - Or just use gdb

End of Story?

Not yet…

Not yet

- I was cleaning my desk…
  - when I received this tweet
- Is all the nRF52 Family vulnerable to the APPROTECT Bypass?
  - Immediately order two more boards
    - nRF52833-DK
    - nRF52-DK



Quick Analysis

The entire family is likely vulnerable

- IP blocks such as Flash Controller, Cortex-M Core are the same across the nRF52 Family
- Comparison between the nRF52 from Nordic Website

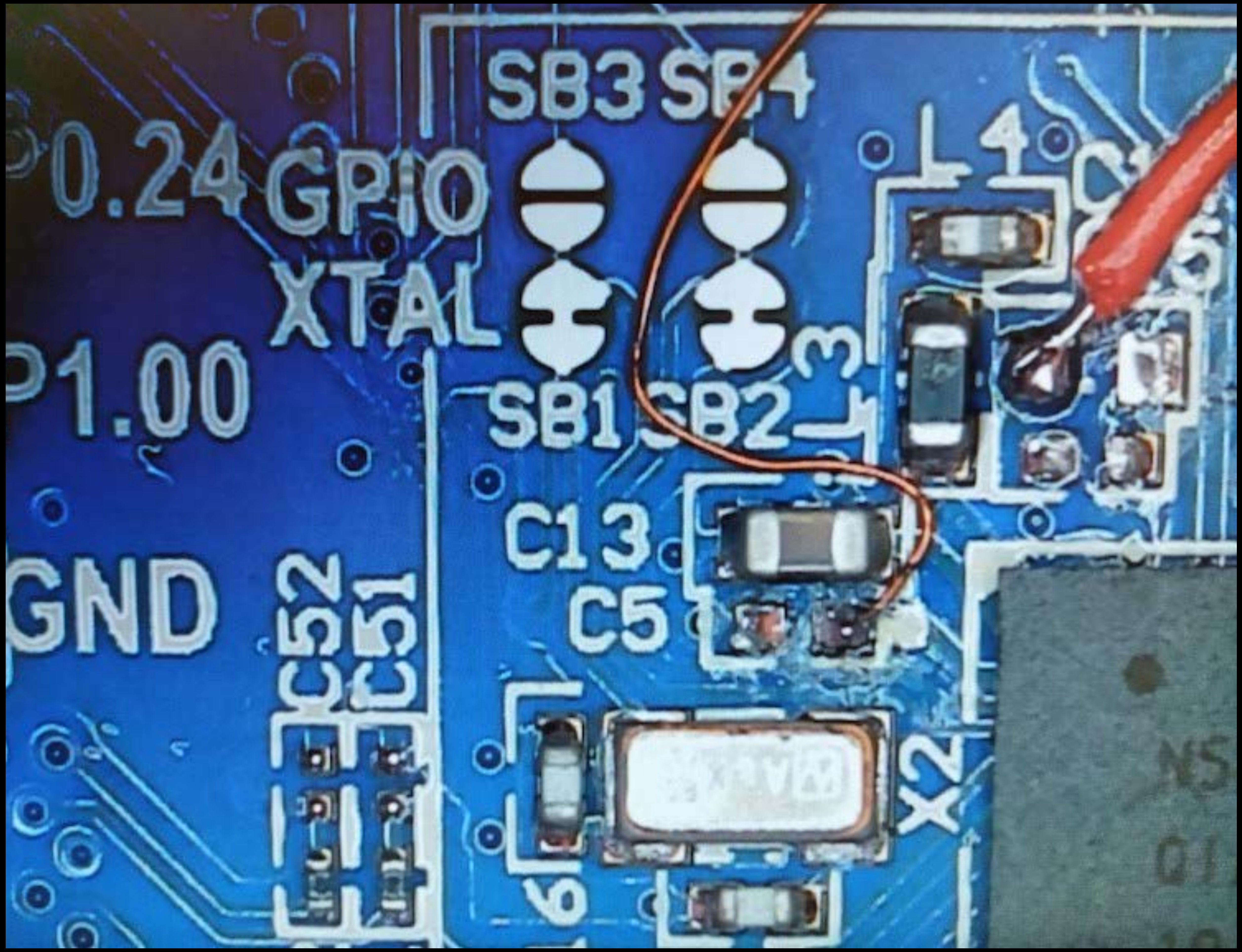
Features	nRF52805	nRF52810	nRF52811	nRF52820	nRF52832	nRF52833	nRF52840
CPU	Cortex-M4	Cortex-M4	Cortex-M4	Cortex-M4	Cortex-M4 with FPU	Cortex-M4 with FPU	Cortex-M4 with FPU
	64 MHz	64 MHz	64 MHz	64 MHz	64 MHz	64 MHz	64 MHz
Memory	192 kB flash	192 kB flash	192 kB flash	256 kB flash	512/256 kB flash	512 kB flash	1 MB flash
	-	-	-	-	Cache	Cache	Cache
	24 kB RAM	24 kB RAM	24 kB RAM	32 kB RAM	64/32 kB RAM	128 kB RAM	256 kB RAM

- Let’s confirm that

Some modifications

nRF52833

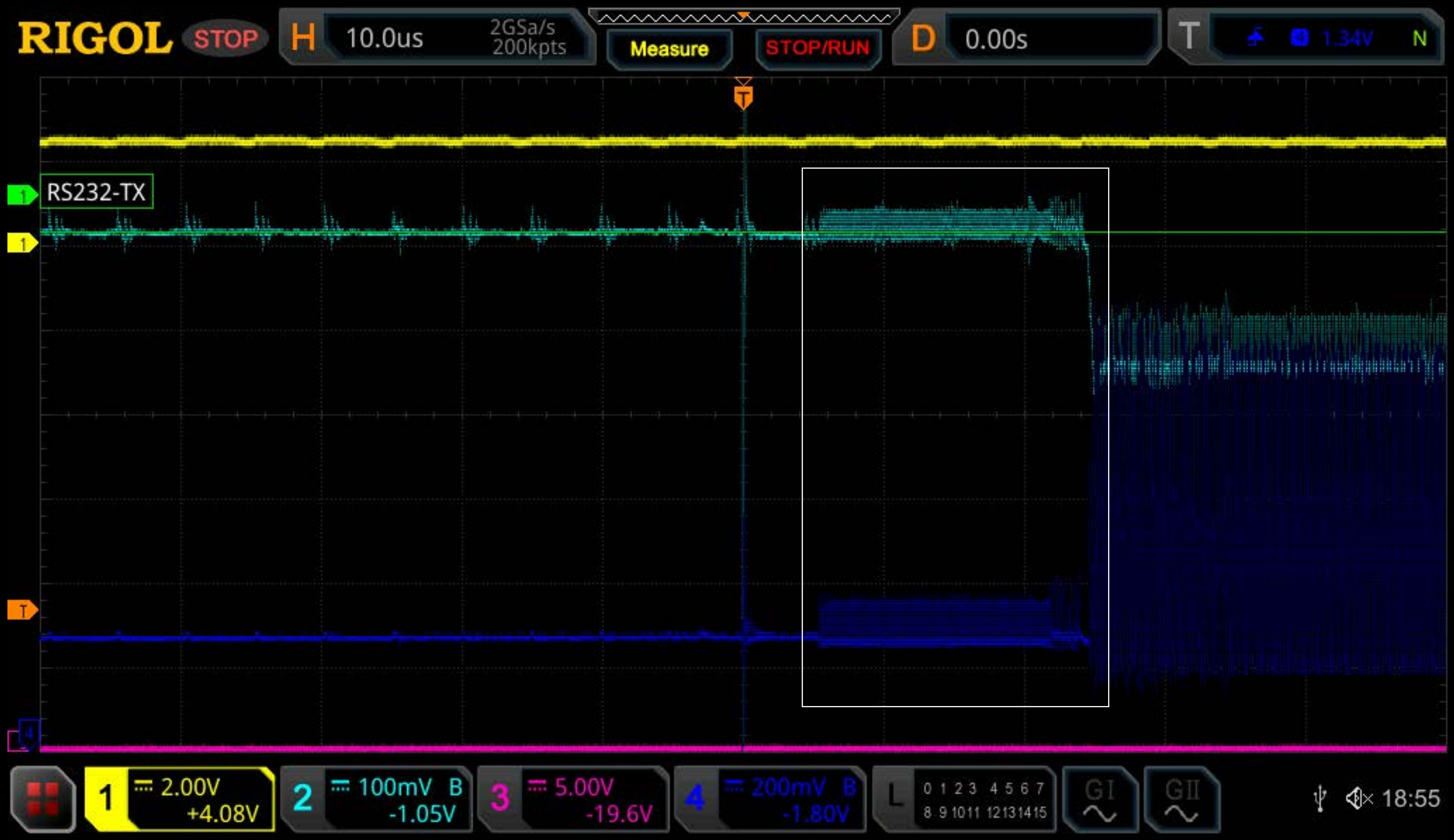
- Enameled wire = DEC1 (CPU_VCC)
- Red wire = DEC4 useful to monitor the chip activity
  - Not necessary
- Don’t forget to put GND somewhere…



Same Pattern

nRF52833

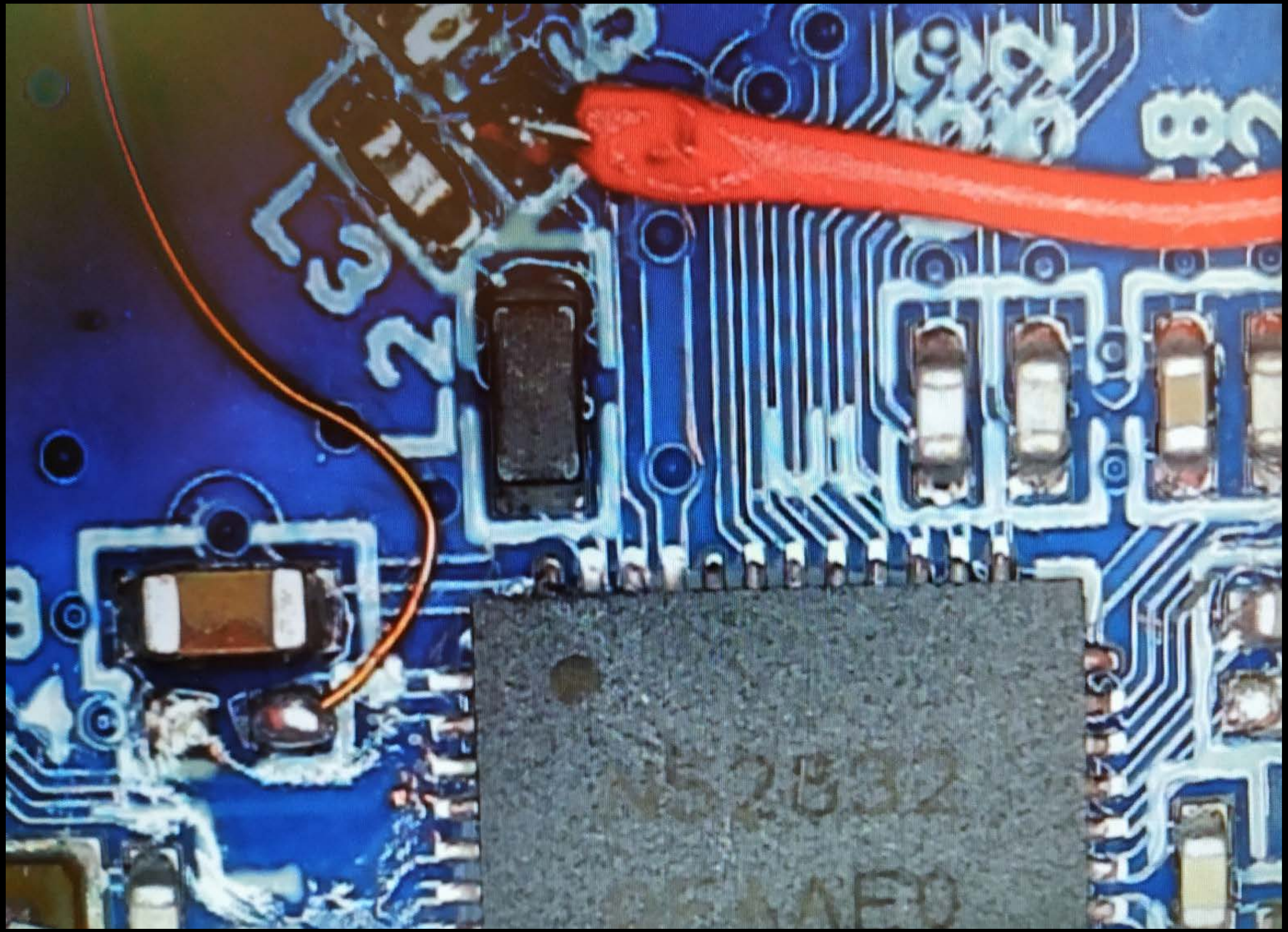
- Clean Power Consumption on DEC1 (CPU)
- Clean Power Consumption on DEC4 (global)



Some modifications

nRF52832

- Enameled wire = DEC1 (CPU_VCC)
- Red wire = DEC4 useful to monitor the chip activity
  - Not necessary
- Don’t forget to put GND somewhere…



Same Pattern

nRF52832

- Clean Power Consumption on DEC1 (CPU)
- The same pattern can be easily identified

Same Consequence

nRF52832/nRF52833

- nRF52833 (after a glitch)

```
Type "apropos word" to search for commands related to "word".
(gdb) target remote :3333
Remote debugging using :3333
0x000060ac in ?? ()
(gdb) info mem
Using memory regions provided by the target.
Num Enb Low Addr   High Addr  Attrs
0   y   0x00000000 0x00080000 flash blocksize 0x1000 nocache
1   y   0x00080000 0x10001000 rw nocache
2   y   0x10001000 0x10001100 flash blocksize 0x100 nocache
3   y   0x10001100 0x100000000 rw nocache
(gdb) x/1x 0x10001208
0x10001208:     0xffffff00
(gdb) x/10x 0x0
0x0:    0x20010000      0x000002b5      0x000002dd      0x000002df
0x10:   0x000002e1      0x000002e3      0x000002e5      0x00000000
0x20:   0x00000000      0x00000000
```

```
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
0
Info : J-Link OB-SAM3U128-V2-NordicSemi compiled Jan 21 2020 17:30:48
Info : Hardware version: 1.00
Info : VTarget = 3.300 V
Info : clock speed 1000 kHz
Info : SWD DPIDR 0x2ba01477
Info : nrf52.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : accepting 'gdb' connection on tcp/3333
target halted due to debug-request, current mode: Thread
xPSR: 0x61000000 pc: 0x000060ac msp: 0x2000fff8
Warn : Unknown device (HWID 0x00000197)
```

- Same for nRF52832

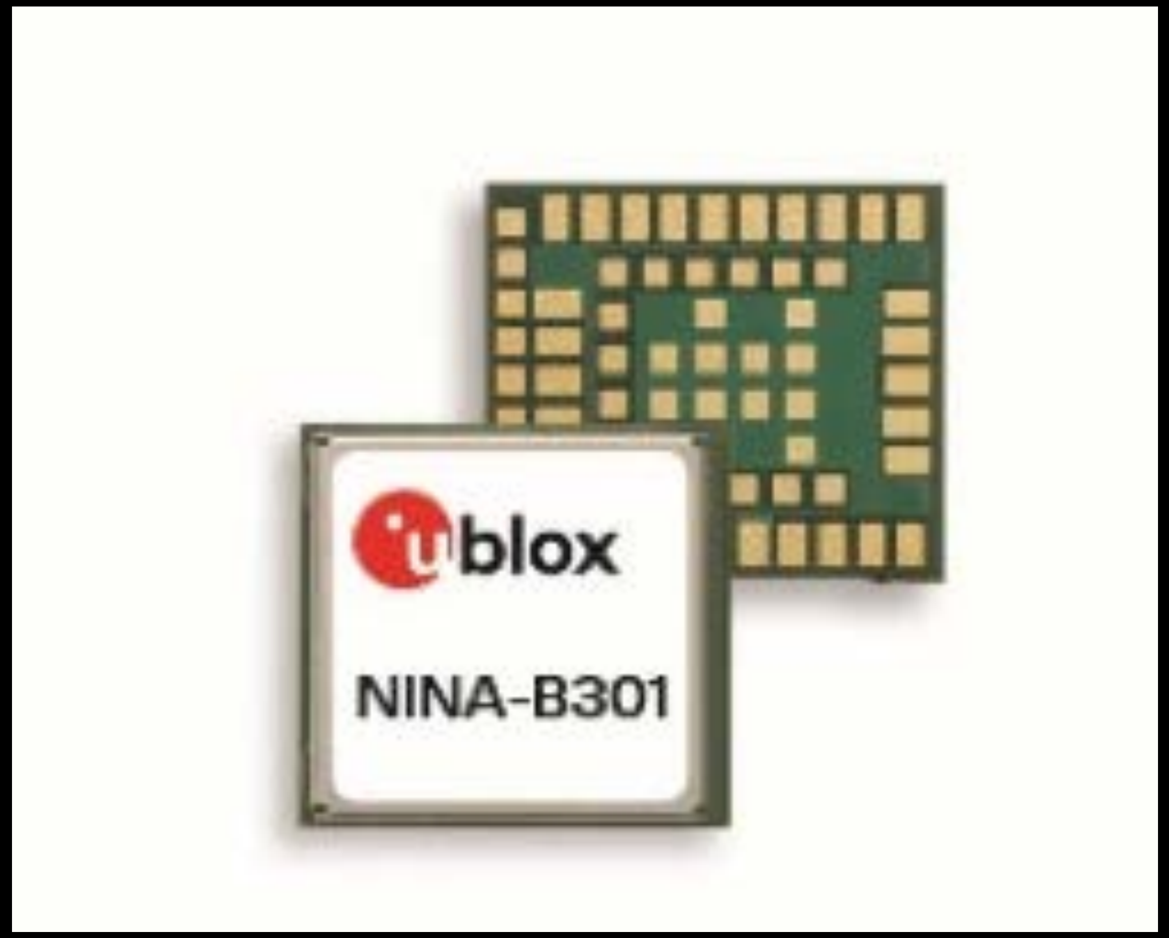
Impact

All the nRF52 are vulnerable…Forever

- Security Researchers/Hackers
  - It is Good news
- Developers
  - In case you rely on nRF52 Flash Content's Confidentiality…
  - Update your threat model
- Nordic
  - No cooperation, that's life
  - Following the post, they sent Information Notice 2 days later
  - No patch, no redesign… nRF52 will stay vulnerable FOREVER

More Impact?
nRF52 based Modules

- Third Party Modules based on nRF52 are impacted
  - Fanstel Corp.
  - Laird
  - Minew
  - Raytac
  - Taio Yuden
  - U-blox
  - Wurth Elektronik
  - Murata
  - Dynastream
  - Fujitsu…and others



Conclusion

- APPROTECT Bypass on nRF52

- Fault Injection attack to allow Debug Resurrection
  - Physical Access required
  - Low-Cost equipment and Low-level Hacking skills to reproduce
  - Has to be achieved only once
- Results
  - Firmware Extraction (including FICR)
  - Full Debug access (R/W Memory, Breakpoints…)
- No way to patch

Thank you

@LimitedResults