The Quest of the Geckos

LimitedResults

NoHat 21, Bergamo

LR

- Security Researcher
  - www.limitedresults.com
  - No Affiliation
- Main Focus
  - Hardware Vulns
  - Reverse of embedded devices

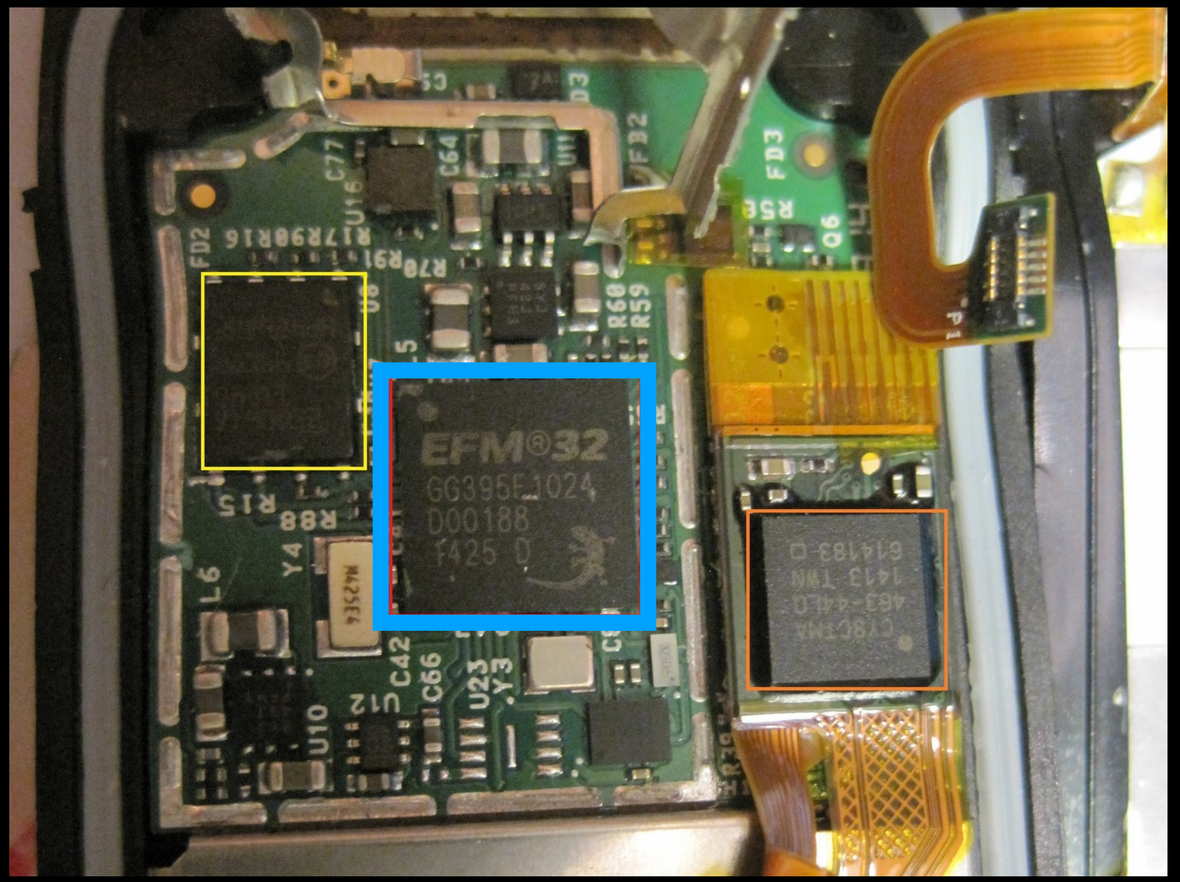
Agenda

- Intro
- The Target
- Preparation
- Hacking journey
- Conclusion

INTRO

What is it?

- Gecko is short name for the SiliconLabs EFM32
  - Quite popular platform
  - Released in 2010
- Lot of EFM32-based products out-there
  - Ex: Fitbit Surge **EFM32GG**



SiliconLabs

- US Semiconductor Company
  - Ultra Low Power MCUs/Wireless SoCs
  - www.silabs.com
  - NASDAQ Market Cap = 7 $Billions
- Main Applications
  - Smart Metering, Health & Fitness, Home automation, Industrial…

EFM32 aka Geckos

- Originally designed by Energy Micro until 2013

Family	Core	Speed (MHz)	Flash memory (kB)	RAM (kB)	USB	LCD	Communications	Packages
Zero Gecko	ARM Cortex M0+	24	4,8,16,32	2,4	No	No	I2C, I2S, SPI, UART, USART	QFN24, QFN32, QFP48
Happy Gecko	ARM Cortex M0+	25	32,64	4,8	No, Yes	No	I2C, I2S, SPI, UART, USART	CSP36, QFN24, QFN32, QFP48
Tiny Gecko	ARM Cortex M3	32	4,8,16,32	2,4	No	Yes	I2C, I2S, SPI, UART, USART	BGA48, QFN24, QFN32, QFN64, QFP48, QFP64
Gecko	ARM Cortex M3	32	16,32,64,128	8,16	No	Yes	I2C, SPI, UART, USART	BGA112, QFN32, QFN64, QFP100, QFP48, QFP64
Jade Gecko	ARM Cortex M3	40	128,256,1024	32,256	No	No	I2C, I2S, SPI, UART, USART	QFN32, QFN48, BGA125
Leopard Gecko	ARM Cortex M3	48	64,128,256	32	Yes	Yes	I2C, I2S, SPI, UART, USART	BGA112, BGA120, CSP81, QFN64, QFP100, QFP64
Giant Gecko	ARM Cortex M3	48	512,1024	128	Yes	Yes	I2C, I2S, SPI, UART, USART	BGA112, BGA120, QFN64, QFP100, QFP64
Pearl Gecko	ARM Cortex M4	40	128,256,1024	32,256	No	No	I2C, I2S, SPI, UART, USART	QFN32, QFN48, BGA125
Wonder Gecko	ARM Cortex M4	48	64,128,256	32	Yes	Yes	I2C, I2S, SPI, UART, USART	BGA112, BGA120, CSP81, QFN64, QFP100, QFP64

- Min. Longevity Commitment until March 2026 (for the oldest)

Classic Attack Path

- Identify “bankable” target
- Get the Firmware
- Find vulnerability (at least one)
  - Emulation/Fuzzing/Reverse
- Develop exploit
  - Require debug access (it makes it easier…)
- Enjoy

What’s wrong when you connect a debugger?

- Debug is locked
  - Could not Find MEM-AP to control the Core

```
xPack OpenOCD, x86_64 Open On-Chip Debugger 0.11.0-00155-ge392e485e (2021-03-15-1
6:43)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 6000 kHz

swd
cortex_m reset_config sysresetreq

Info : Silicon Labs J-Link OB compiled Mar 27 2020 10:55:55
Info : Hardware version: 1.00
Info : VTarget = 2.014 V
Info : clock speed 1000 kHz
Info : SWD DPIDR 0x2ba01477
Error: Could not find MEM-AP to control the core
Warn : target etm32.cpu examination failed
```

Code Readout Protection

- Most of the time, the vendor does not provide Firmware
  - Firmware Encryption is (also) common
- CRP
  - HW Security Mechanism disabling the debug interface
  - Present in most of the MCUs/SoCs with integrated Flash
  - Prevent an attacker to access the Code/Data stored in Flash
  - Protect against Reverse-Engineering
- No Firmware, no results :(
  - Need to find a way to get FW/Debug access

THE TARGET

ARM Debug Access Port (DAP)

- External Interface is Debug Port (DP)
  - SW-DP Serial Wire Debug (SWD) Port
    - ARM proprietary
    - Only 2 pins SWDIO and SWCLK
- Ressource interface is Access Port (AP)
  - Simplest implementation has only one (MEM-AP)
  - Complex implementation can have several (AHB-AP, CTRL-AP…)
- ARM Debug Interface v5 Arch. Specification

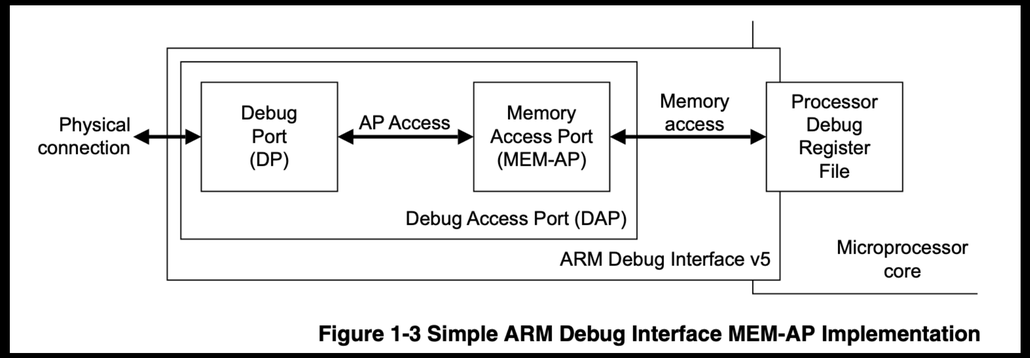
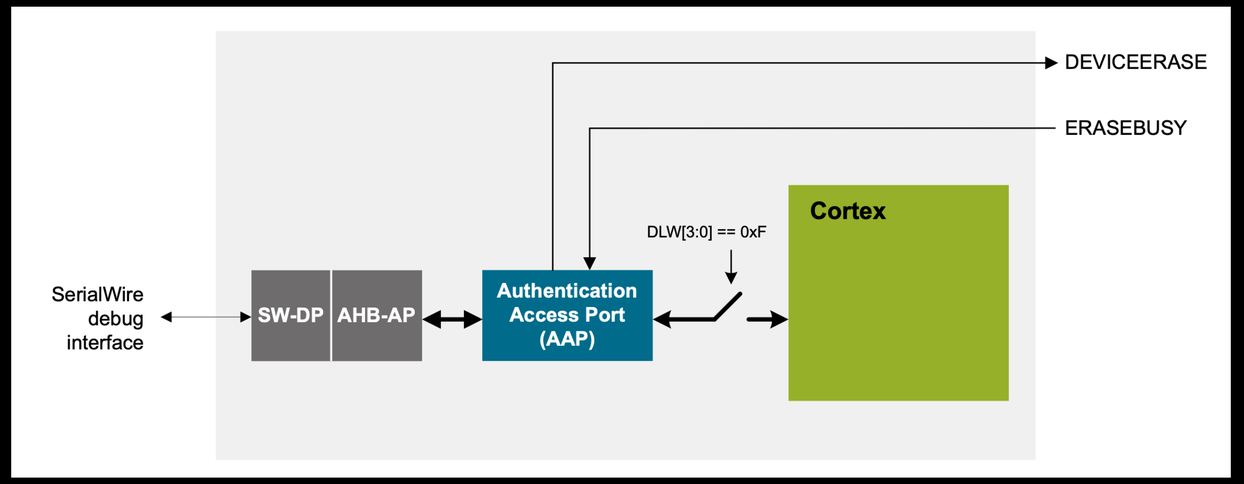


Figure 1-3 Simple ARM Debug Interface MEM-AP Implementation

Authentication Access Port (AAP)

- Prevent direct access to the Flash and the RAM, but also to all the registers (AHB-AP)
- Reading datasheet does not bring so much details…
  - *By the way, this is more an authorisation mechanism than authentication but…*



- AAP never hacked (As Far As I Know)

Lock Bits

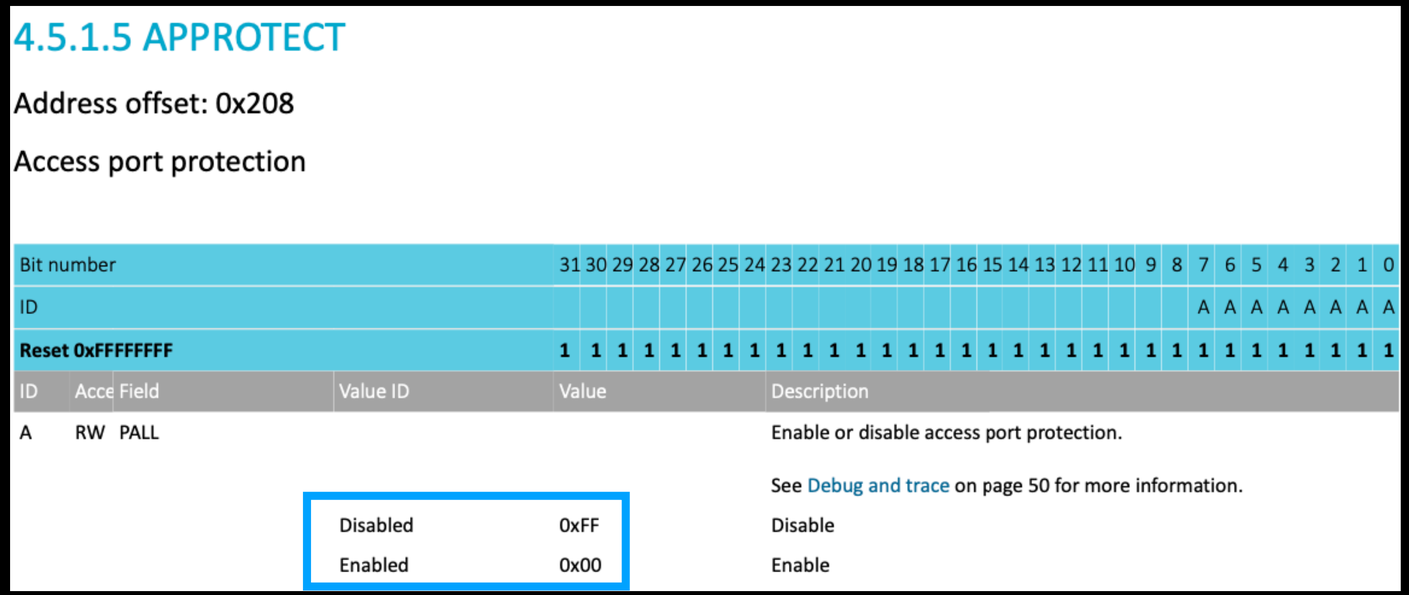
- Located in Flash at 0x0EF0_4000

```
382 #define LOCKBITS_BASE      (0x0FE04000UL) /**< Lock-bits page base address */
383 #define USERDATA_BASE      (0x0FE00000UL) /**< User data page base address */
```

- If Debug Lock Word DLW[3:0] == 0xF
  - Debug access is enabled
- Else
  - Debug access is locked
- AAP is activated by writing 0x0 into DLW and then reset

Deja-Vu on nRF52 APPROTECT

- nRF52 design
  - APPROTECT is also mapped in Flash Memory (UICR at 0x1000_1208)
  - Write 0xFFFFFF00 enable the Access Port Protection
  - Cannot be disabled without erasing all the RAM and Memory Flash



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

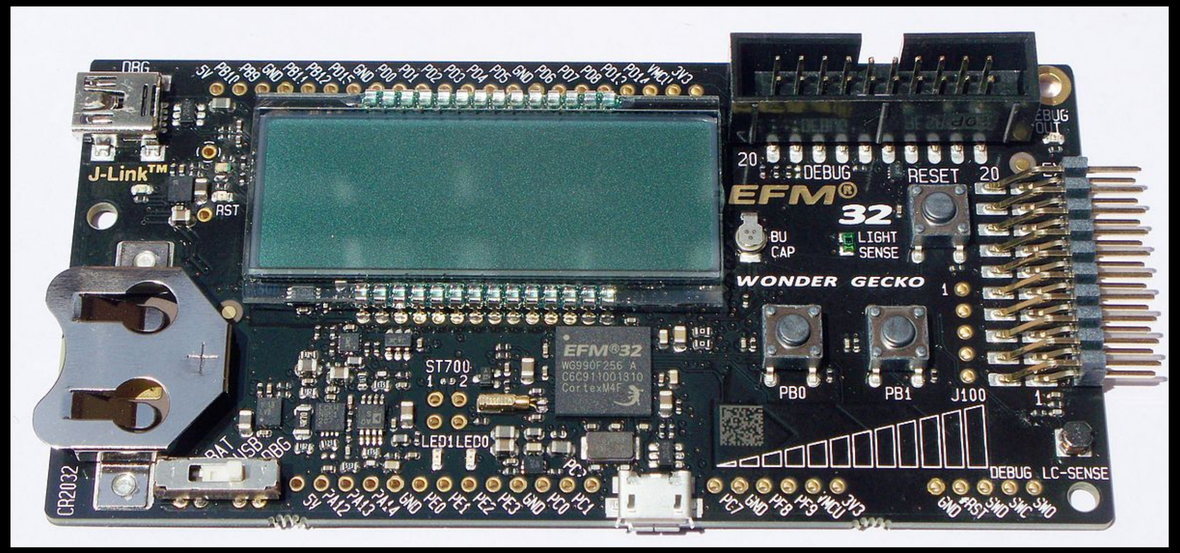
The Plan

- Context
  - Stuck at home :/ (Winter 2021)
- My Goal
  - Bypass the AAP to reactivate debug capabilities on EFM32
- Why am I confident to achieve this?
  - Similar to the nRF52 design
    - Debug resurrection on Nordic nRF52
  - Should be doable using voltage glitching, just after a Power-On-Reset, when the AAP is initialized

PREPARATION

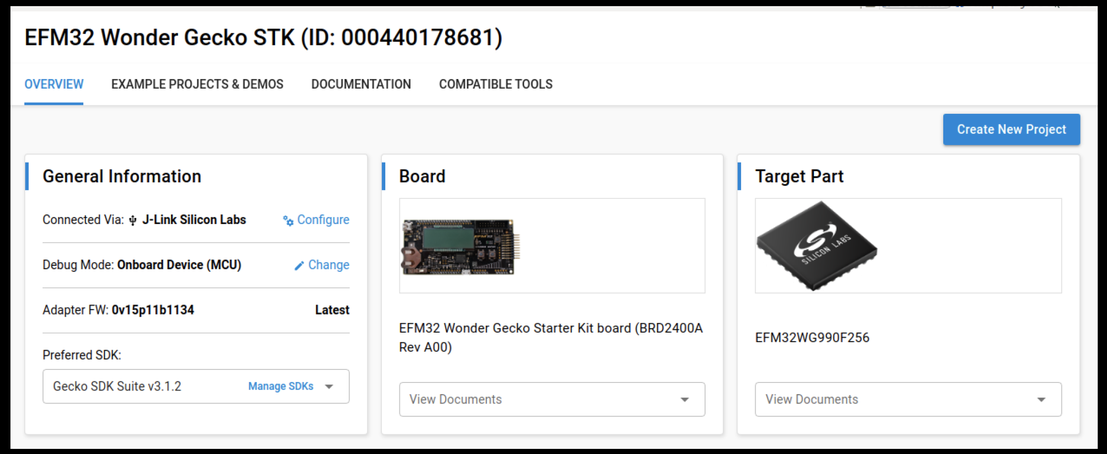
EFM32WG Dev-Kit

- EFM32WG-STK3800
  - 100$:/
- Why using dev-kit?
  - Exposed interfaces
  - J-LINk Debugger is embedded
    - less wires on the desk
  - Reference dev-kit design is generally close to real products
- A dev-kit can be reused for other dev-projects



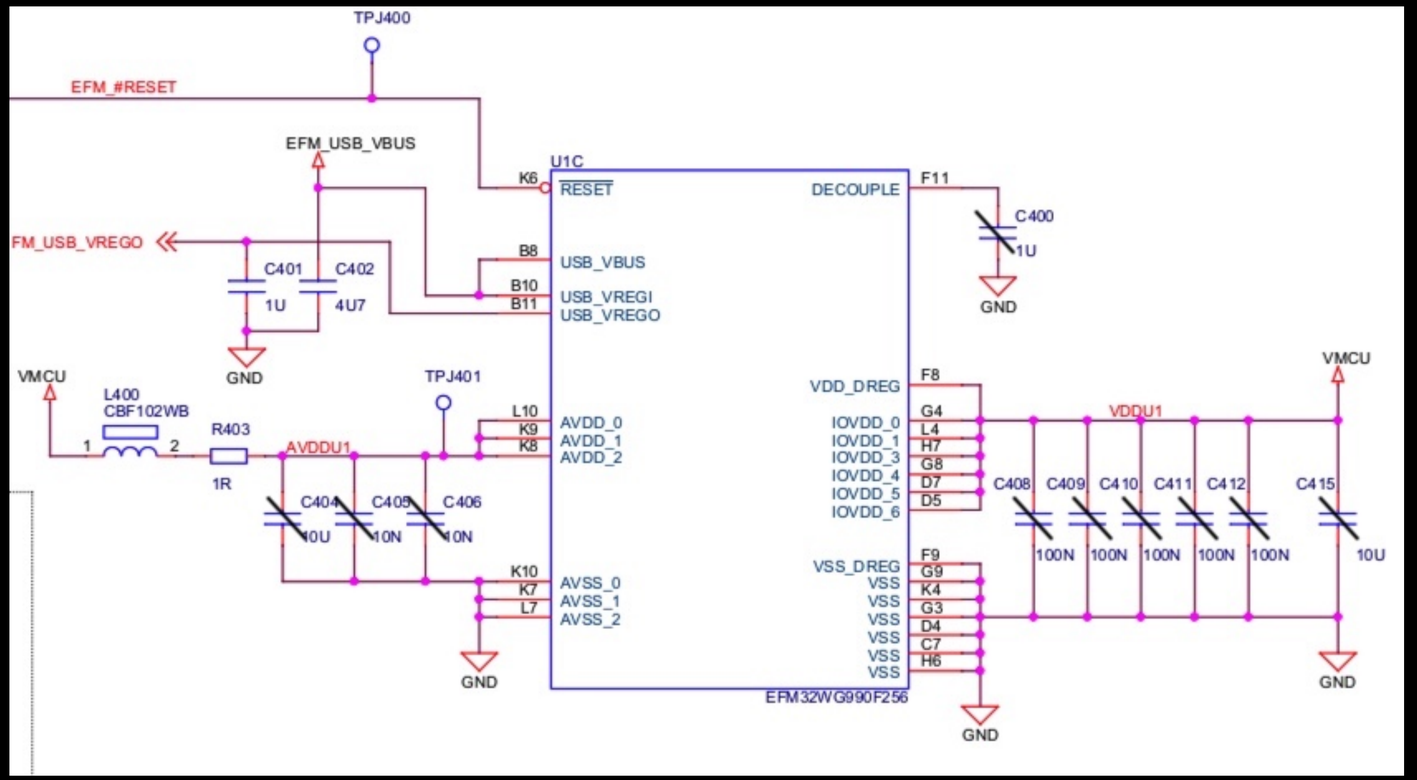
Silabs tools

- Simplicity Studio 5
  - Create project && Compile
  - Flash the binary
  - It works
- Enable the AAP / Disable the AAP
  - `$ ./commander device lock`
  - `$ ./commander device unlock`
  - This unlock operation will erase the Flash Memory content



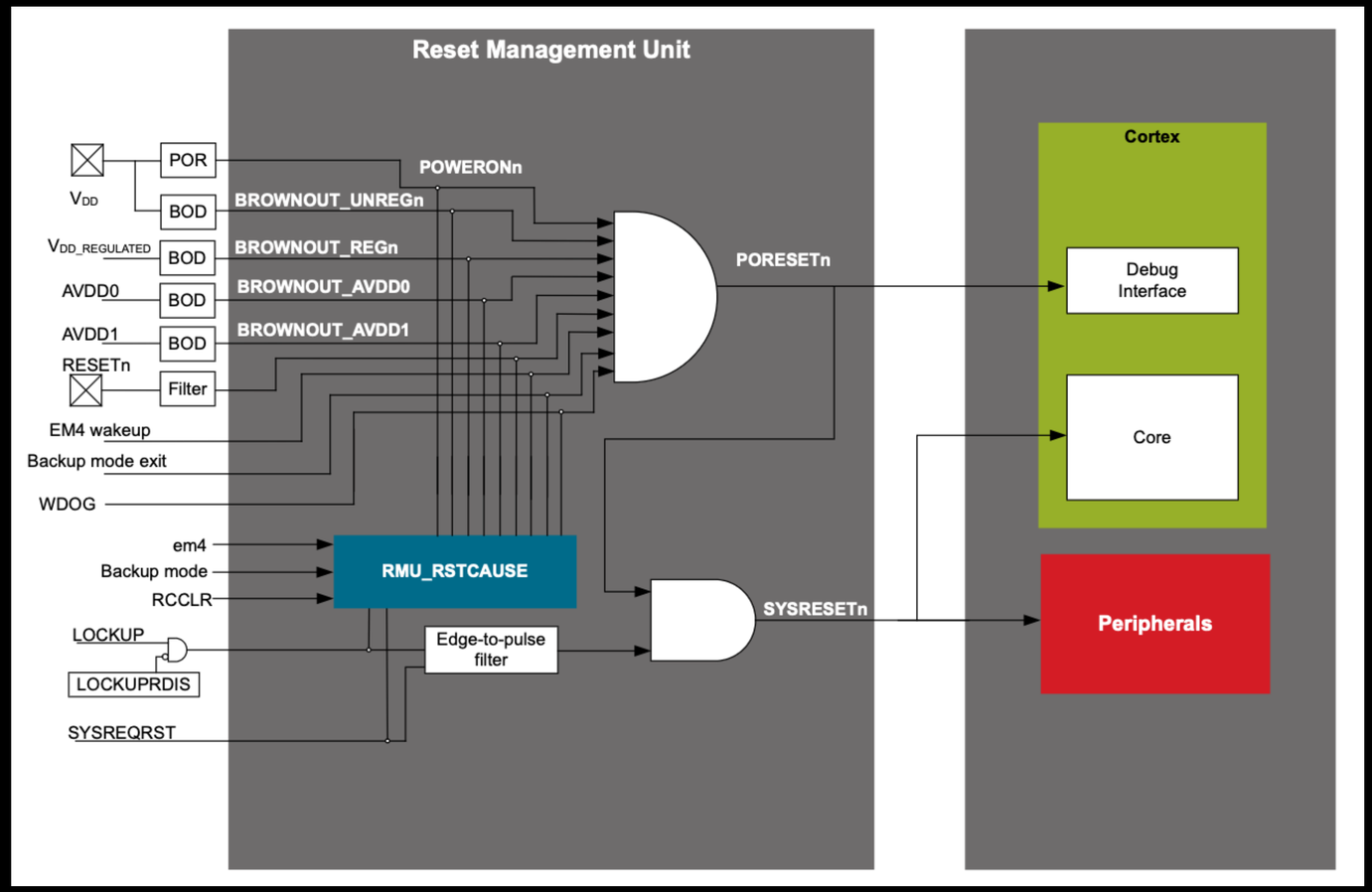
PCB Modification

- Schematics present in the IDE, thank you
- Why removing capacitors?
  - To study power consumption. No big low-pass filter effect (RC)
  - Sharper drop-out for voltage glitching
  - Better repeatability and fine-tuning parameters



Power domain

- Not clearly defined in the data sheet
- Several domains
  - Digital Vdd_reg
  - Analog AVDD
- Presence of BODs and Filters
  - Not really a problem…*optimistic mode ;)*



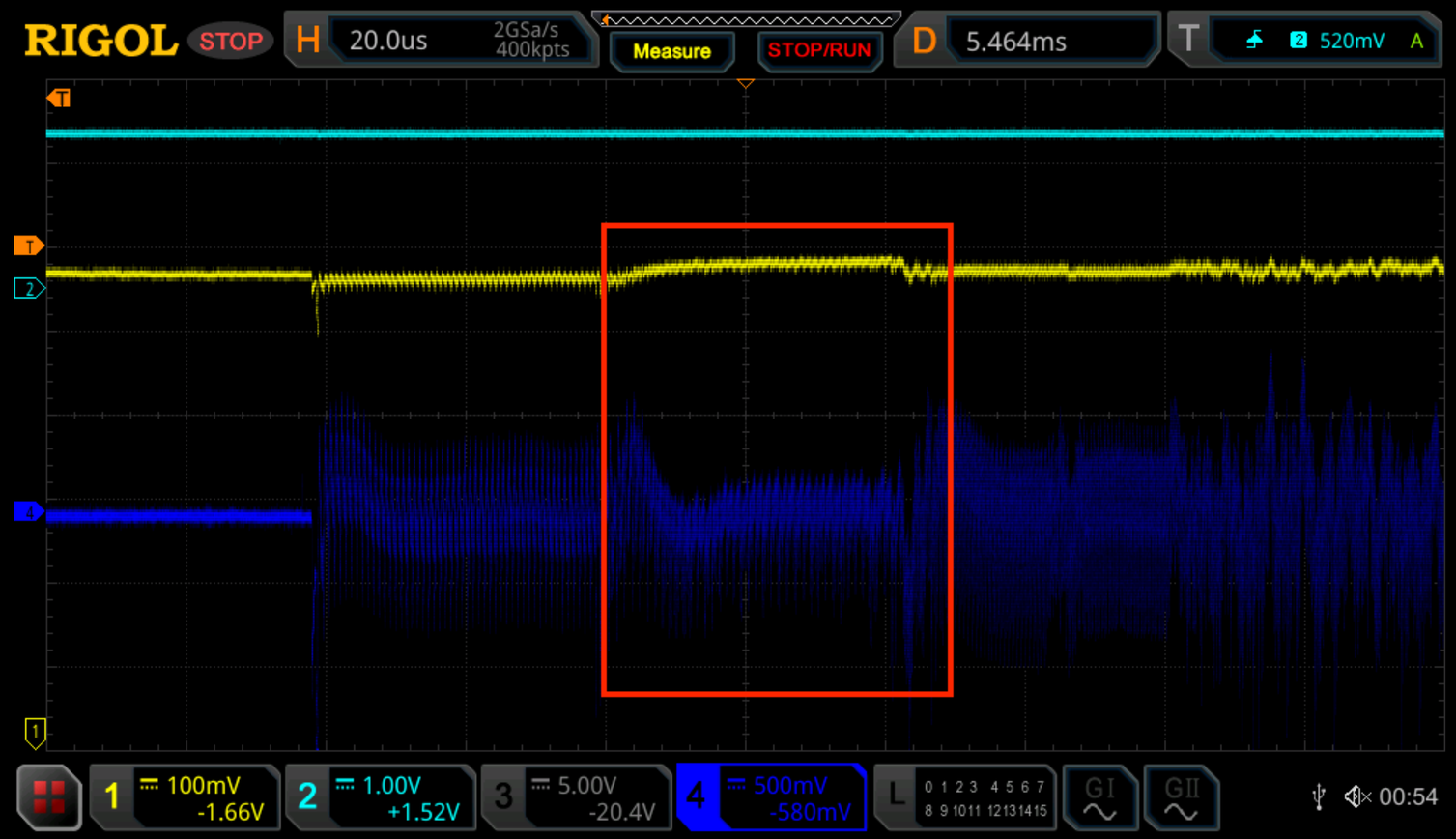
TIME TO HACK

Glitching

- Voltage Fault Injection, the cheapest technique
- Well known technique, lot of public resources
  - Glitching for Noobs – Exide
  - Glitching and Side Channel Analysis for all – Colin O’Flynn and more…
- Disrupt the power supply to induce a fault during critical SW/HW operations
- Known effects
  - Skipping Instruction, Data/Code Mutation
  - Difficult to predict the fault model (CPU arch, Memory…)
- Commercial Tools are available…
  - but you can also do your own glitcher

Power Consumption

- Scope is ON
- Glitching require to study the power Consumption
  - Essential step of the process
  - Identification of the targeted process



Brown-Out Detectors

- First glitching attempts were not very successful…
- The EFM resets after a glitch
- This is due to these voltage sensors called BOD

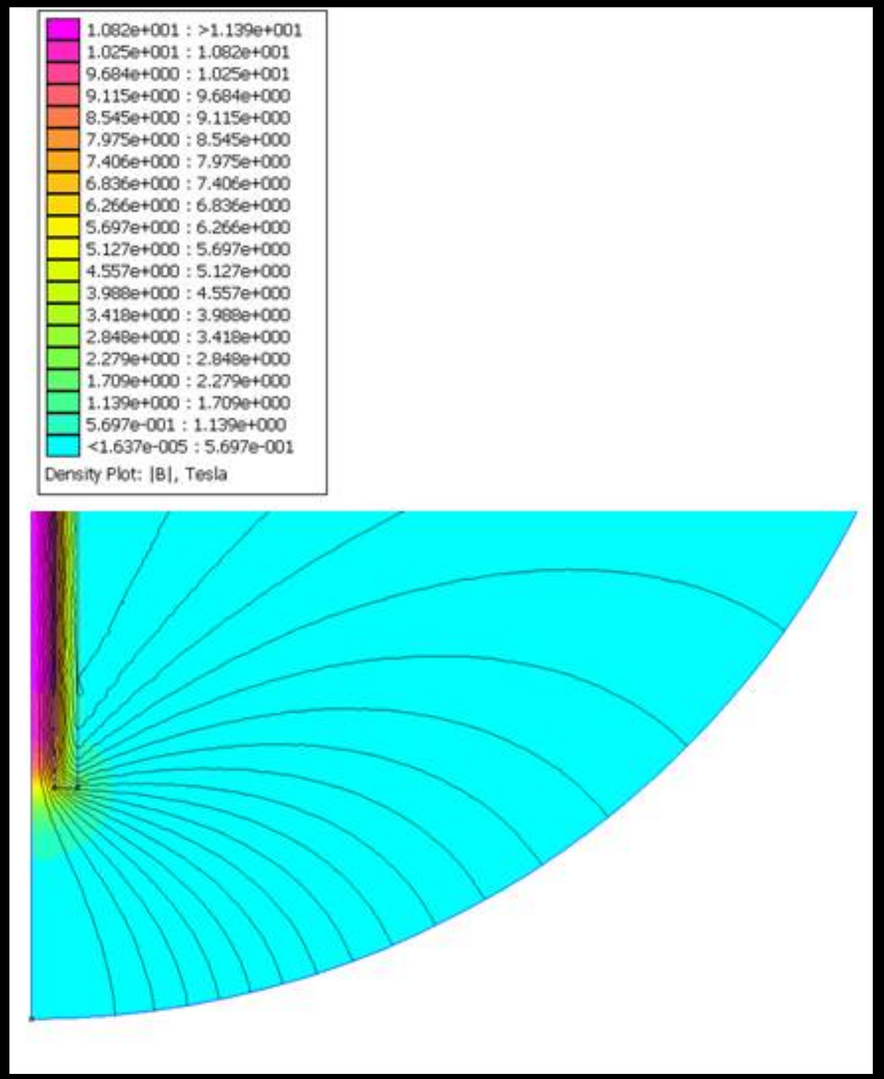
Further Attempts

- I characterised BOD to fully understand the behaviour and their range of operations
  - *Special Shoot-out to the EFM32 mixed analog team, remarkable work*
- I tried different glitching techniques
  - Under/Over Voltage
- Long story short, no way to bypass these detectors…
  - I need to find something else..

DO NOT GIVE UP…

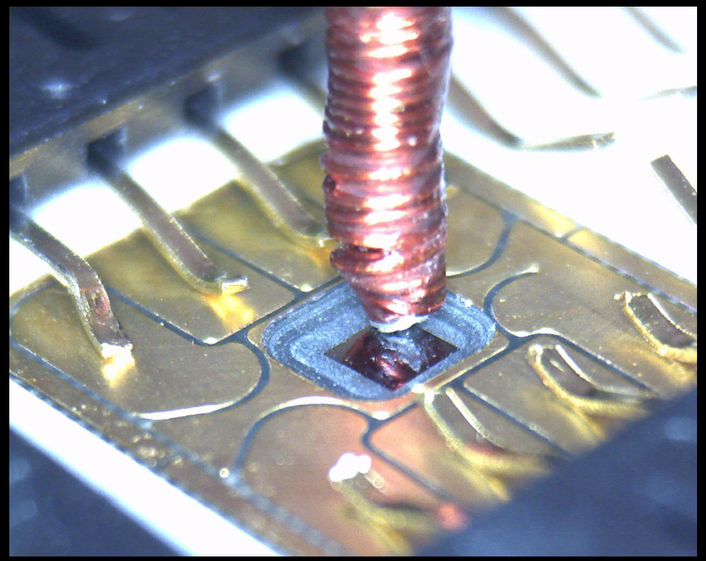
EMFI

- Integrated Circuits are sensitive to ElectroMagnetic fields (EMC)
- Principle
  - Inject strong EM fields induced from Current Pulse into an Integrated Circuit
  - EM Bomb but at little scale :)
- Local Glitch Effect (spatial resolution) depends on
  - Probe design
  - EM field coupling (quite complex, require dedicated simulations)



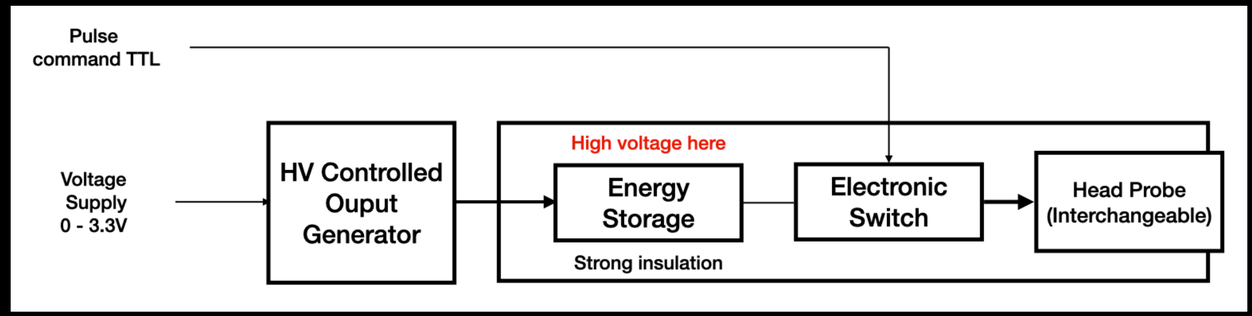
When you cannot afford a Tool…

- EMFI is not new technique
  - Used during smartcard security evaluation
  - Lot of academic research papers 10 years ago
- Commercial solutions
  - Riscure EM-FI 20K$
  - Newae ChipShouter 3K$
- Obvisouly, I CANNOT pay this price to break a 5$ chip



You have to make your own

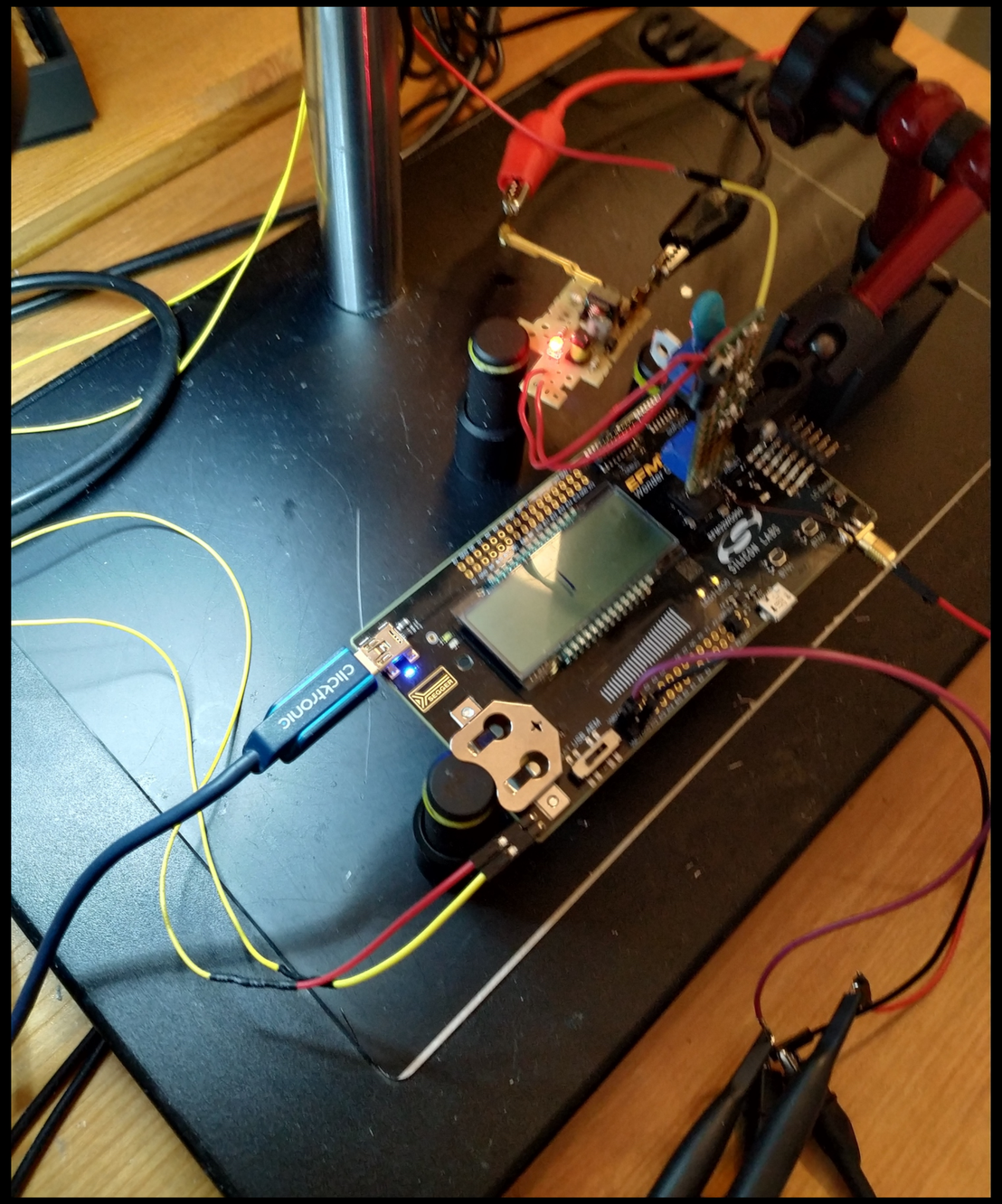
- Block Diagram



- Requirements
  - Input Voltage from 0.5V to 3.3V (AAA battery)
  - Pulse Width 20ns–200ns, acceptable Jitter (< 100ns)
  - Sufficiently powerful and localised to perform Fault Injection

Limited Setup

- DER INJEKTOR
  - One of my first prototype
    - Based on disposable camera
    - Triggered by TTL signal
  - Ultra Low cost

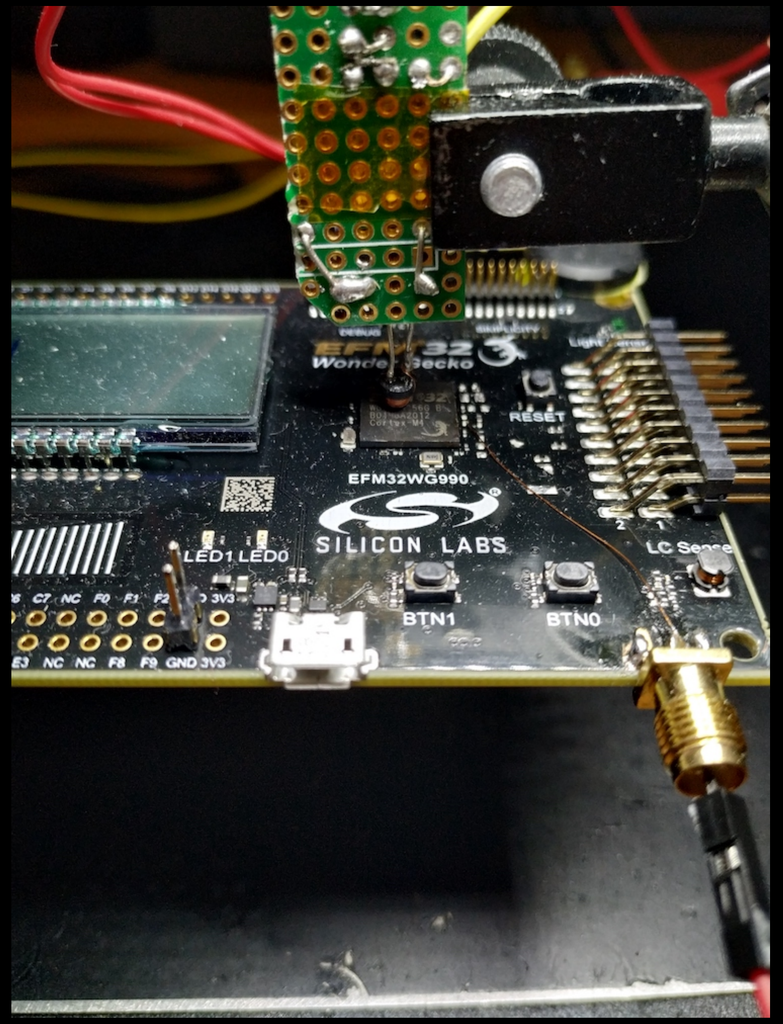


Safety Disclaimer

- Do not try this at Home
- The author shall not be hold liable for any of your actions

Limited Probes

- Hand-made probes
  - Different designs from empirical tests
- Pretty Localized effects
  - Location at the top of the circuit is important
  - XYZ stage? Not really needed…



PoC

- Debug Unlocked
  - Full R/W
  - Full debug
- Once FW extracted, device can then be erased and flashed again
  - Persistent debug
- Attack has to be done only once

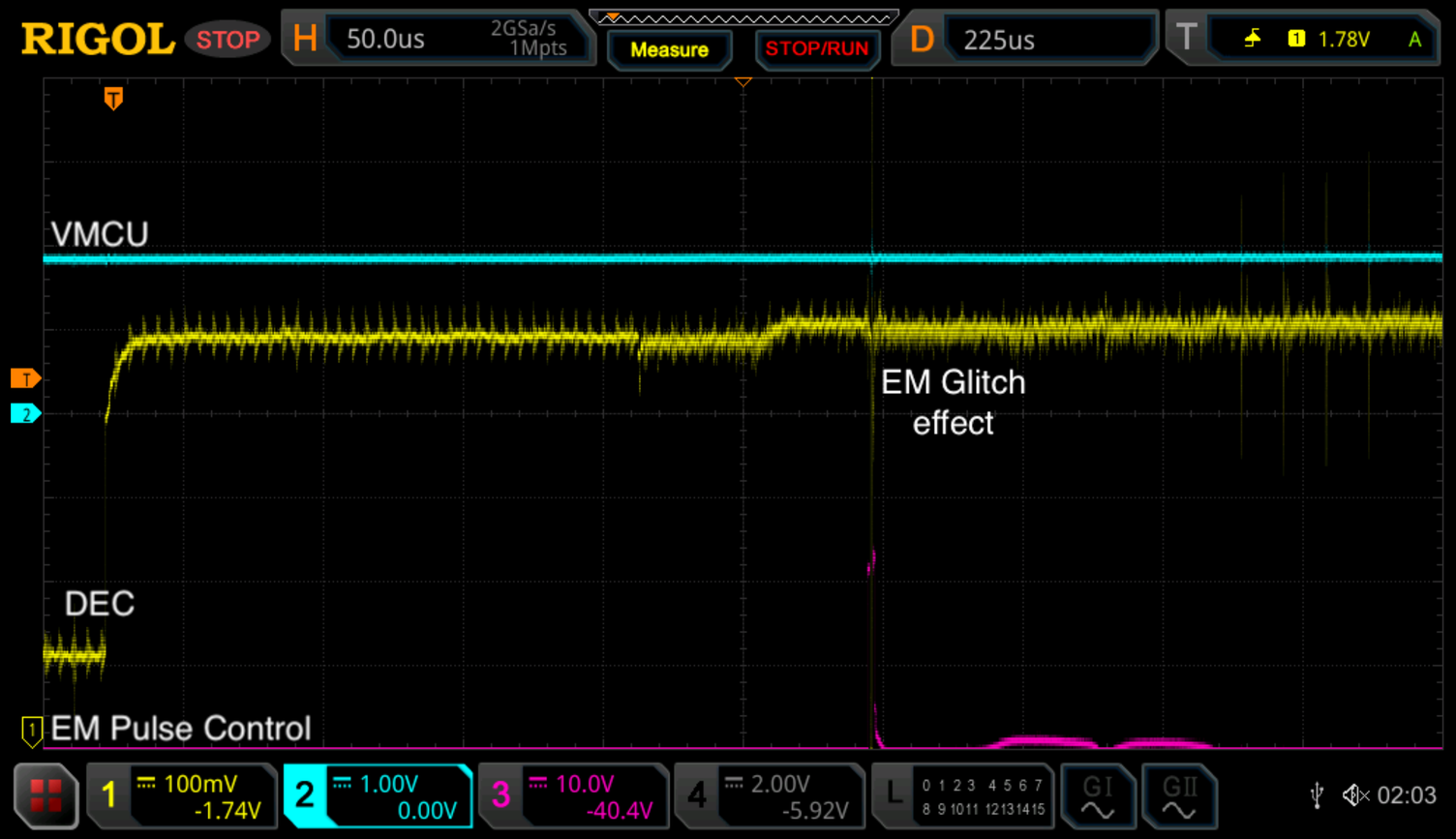
```
interface/jlink.cfg -c 'transport select swd' -f /home/xisco/opt/xP
penocd/0.11.0-1.1/.content/scripts/target/efm32.cfg
xPack OpenOCD, x86_64 Open On-Chip Debugger 0.11.0-00155-ge392e485e
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 6000 kHz

swd
cortex_m reset_config sysresetreq

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : Silicon Labs J-Link OB compiled Mar 27 2020 10:55:55
Info : Hardware version: 1.00
Info : VTarget = 2.018 V
Info : clock speed 1000 kHz
Info : SWD DPIDR 0x2ba01477
Info : efm32.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : starting gdb server for efm32.cpu on 3333
Info : Listening on port 3333 for gdb connections
Info : accepting 'gdb' connection on tcp/3333
Info : detected part: EFM32WG Wonder Gecko, rev 164
Info : flash size = 256kbytes
Info : flash page size = 2048bytes
```

Scope view

- The EM effect is very sharp and deep on the VDD
  - 50ns width
  - No bounce
- Fault timing is indicated
  - You can reproduce easily



Disclosure

- Vendor disclosure policy clearly mentions NO REWARD so…
  - I Posted (¬◡¬)
- Four days after… Security Advisory A-00000310 confirmed my results and the impact
  - CVSS Score 6.8 (Medium)
  - No fix for EFM32 and EFR32 Series

Impacted Products:

- EFR32xG22 or EFM32PG22 SoC's and associated modules running VSE firmware v1.2.6 or earlier.
- All EFM32 and EFR32 Series 0 and Series 1 MCUs, SoC's and associated modules.
- All EM35x SoCs and associated modules.

Conclusion

- Research on Embedded Devices require access to the Firmware or/and (protected) debug interfaces
  - More and more difficult Nowadays
- This work identified a new entire MCUs family vulnerable to a debug resurrection
  - Design Vuln. are spread accross different silicon vendors (Nordic, Silabs…)
  - List of vulnerable devices available in Security Advisory (EFM32, EFR32…)
- EMFI
  - Very good technique, and low-cost :)
  - Der Injektor coming soon

Thank you

@limitedResults
www.limitedresults.com