

NO  
FF  
ONE  
2023

# GigaVulnerability: GD32 Security Protection bypass

Speaker: Alexey Kovrizhnykh

Security Researcher, Positive Technologies



# whoami



- Reverse engineer, security researcher
- **positive technologies**
- Flare-On 2018-2020, 2022 winner
- Articles, speeches, research: [alexandandy.me](https://alexandandy.me)
  - Mostly checkm8 related stuff
    - for A5(X) SoCs
    - for T2
    - even for VGA Adapters

# GD32



- **GigaDevice**
- Founded in 2005
- Beijing, China
- NOR flash memory designer
- ARM Cortex-M MCUs **GD32** introduced in 2013-2015
- RISC-V MCUs GD32V introduced in 2019



Performance	Arm® Cortex®-M 32-bit MCUs				RISC-V 32-bit MCUs
	Cortex®-M23	Cortex®-M3	Cortex®-M4	Cortex®-M33	RISC-V
High-Performance		<p>GD32F207 120MHz, 3M/256K</p> <p>GD32F205 120MHz, 3M/256K</p>	<p>GD32F470 240MHz, 3M/768K</p> <p>GD32F427 200MHz, 3M/256K</p> <p>GD32F425 200MHz, 3M/256K</p> <p>GD32F450 200MHz, 3M/512K</p> <p>GD32F407 168MHz, 3M/192K</p> <p>GD32F405 168MHz, 3M/192K</p> <p>GD32F403 168MHz, 3M/128K</p>	<p>GD32W515 180MHz, 2048K/448K</p> <p>GD32E508 180MHz, 512K/128K</p> <p>GD32E507 180MHz, 512K/128K</p> <p>GD32E505 180MHz, 512K/128K</p> <p>GD32E503 180MHz, 512K/128K</p>	
Mainstream	<p>GD32L233 64MHz, 256K/32K</p>	<p>GD32F107 108MHz, 1M/96K</p> <p>GD32F105 108MHz, 1M/96K</p> <p>GD32F103 108MHz, 3M/96K</p> <p>GD32F101 56MHz, 3M/80K</p>	<p>GD32F307 120MHz, 1M/96K</p> <p>GD32F305 120MHz, 1M/96K</p> <p>GD32F303 120MHz, 3M/96K</p> <p>GD32C103 120MHz, 128K/32K</p> <p>GD32E103 120MHz, 128K/32K</p>	<p>GD32E501 100MHz, 512k/32K</p>	<p>GD32VF103 108MHz, 128K/32K</p>
Entry-Level	<p>GD32E232 72MHz, 64K/8K</p> <p>GD32E230 72MHz, 64K/8K</p>	<p>GD32F150 72MHz, 64K/8K</p> <p>GD32F130 48MHz, 64K/8K</p>	<p>GD32F350 108MHz, 128K/16K</p> <p>GD32F330 84MHz, 128K/16K</p> <p>GD32F310 72MHz, 64K/8K</p>		
Specific			<p>GD32FFPR 168MHz, 1M/128K</p>	<p>GD32EPRT 168MHz, 384K/96K+4M</p>	

NO  
FF  
ONE  
2023

# Readout Protection

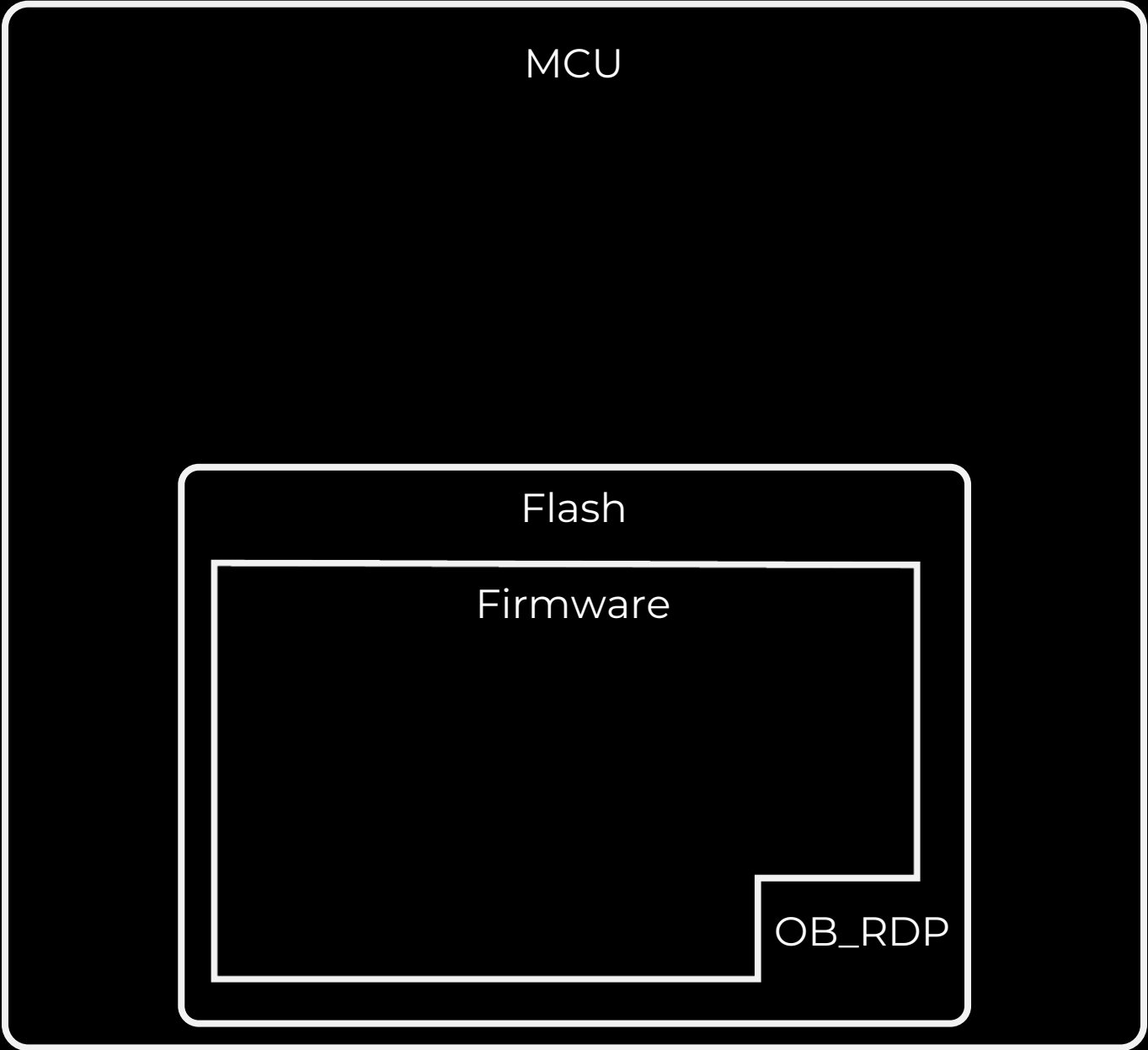
Software providers may need to protect their software intellectual property from malicious users or intrusive attacks

## **Technologies:**

- STM - RDP
- nRF - APPROTECT
- **GD - Security Protection**
- etc.

## **Restrictions:**

- Level 0
  - No restrictions
- Level 1
  - Flash memory is locked (in debug mode)
- Level 2
  - JTAG/SWD interface is disabled
  - Boot from RAM or System memory is disabled
  - Irreversible and cannot be downgraded



NO  
FF  
ONE  
2023

Known Readout  
Protection  
bypasses/vulnerabilities



# Shedding too much Light on a Microcontroller's Firmware Protection

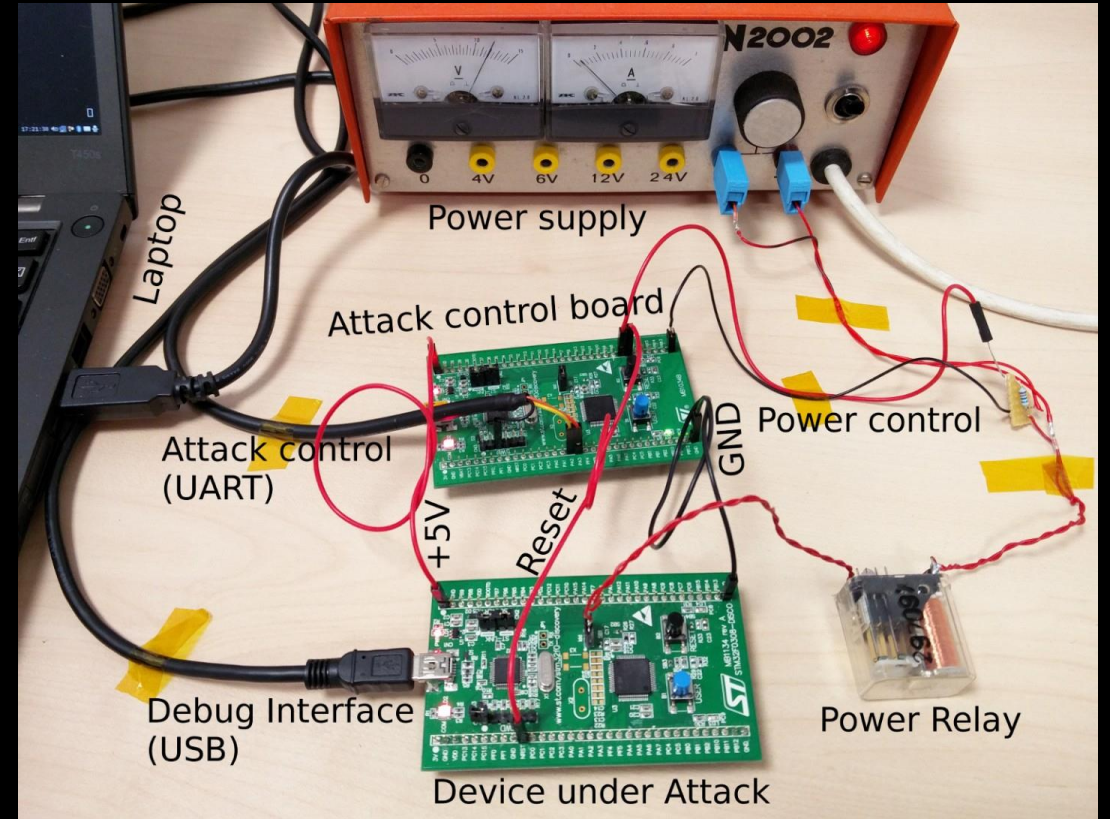


- Authors:
  - Johannes Obermaier
  - Stefan Tatschner
- Main target: STM32F0
- Cold-Boot Stepping
- Security Downgrade
- Debug Interface Exploit
- Links:
  - [Paper](#)
  - [Presentation](#)



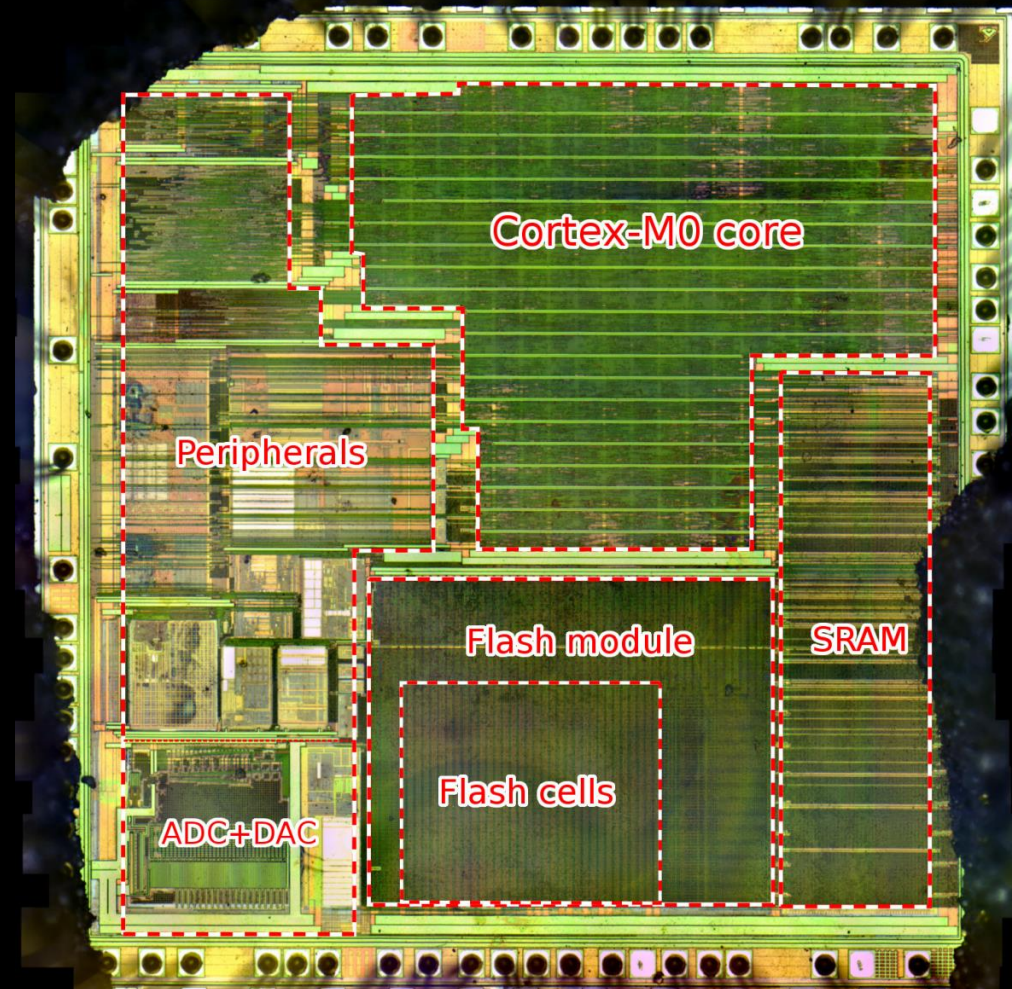
# Cold-Boot Stepping

- RDP1 - SRAM is still available under debugging
- Developer must zero out sensitive data from memory
- CBS technique allows you to get intermediate states of SRAM
- Examples:
  - Firmware CRC32 verification in bootloader
  - Encrypted Firmware Update in bootloader
  - etc.



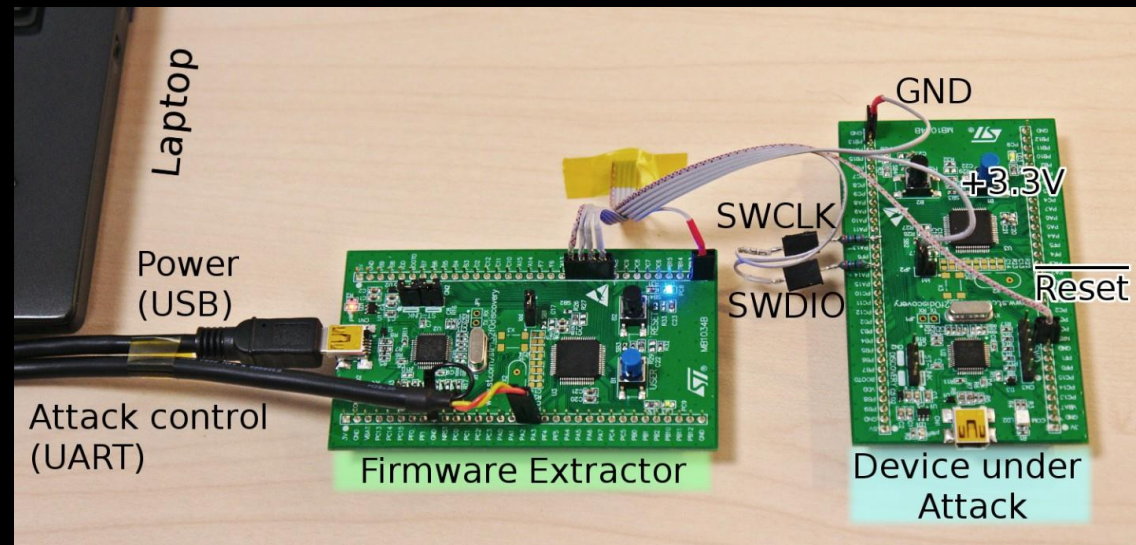
# UV-C Security Downgrade

- UV-C light erases flash memory cells (0 → 1)
- Flipping any bit in Option Bytes causes security downgrade (RDP2 → RDP1)



# Race condition in the debug interface

- STM32F0 in RDP1: only a (bus) access triggers flash lockdown
- If the first bus access targets flash memory, valid data is sometimes returned
- Allows you to extract the entire firmware in parts of 4 bytes
- Can be achieved using J-Link (with openocd) and software controllable relay



# One Exploit to Rule Them All? On the Security of Drop-in Replacement and Counterfeit Microcontrollers



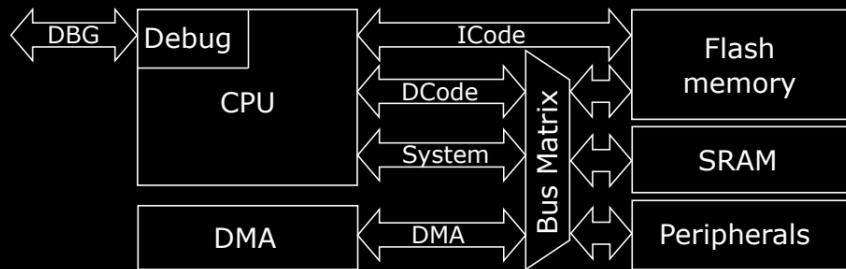
- Authors:
  - Johannes Obermaier
  - Marc Schink
  - Kosma Moczek
- Main targets: STM32F103 & clones (APM, CKS, GD)
- Multiple severe debug interface vulnerabilities
- Invasive hardware attacks on multi-die systems
- A power glitch exploiting software live-patching
- Links:
  - [Paper](#)

# Load Instruction Exploitation

- On some MCUs, the core still has direct access to the flash when RDP1 is enabled
- You can read/write core registers and the program counter, halt/resume the core, do step-by-step execution, etc.
- You can find the proper gadget by analyzing core states step-by-step (CKS32F103)
  - `ldr rX, [rY]`
- Sometimes you can write your own gadget into SRAM and execute it (GD32VF103)
- My keyboard firmware on Sonix SN32F248B was successfully dumped using this technique
- [nRF51822: Firmware dumping technique for an ARM Cortex-M0 SoC](#)

# Extraction via Exceptions

- On an ARM Cortex-M the flash memory is accessed via two buses:
  - Data bus** for data and debug accesses
  - Instruction bus** for instruction and interrupt vector fetches
- On some MCUs the flash memory is blocked for the data bus but not for the instruction bus



0x0800 0000		1	2	3	← VTOR
+0x10	4	5	6	7	
+0x20	8	9	10	11	
+0x30	12	13	14	15	
+0x40	16	17	18	19	
+0x50	20	21	22	23	
+0x60	24	25	26	27	
+0x70	28	29	30	31	← VTOR
0x0800 0080		1	2	3	
+0x10	4	5	6	7	
+0x20	8	9	10	11	
+0x30	12	13	14	15	
+0x40	16	17	18	19	
+0x50	20	21	22	23	
+0x60	24	25	26	27	
+0x70	28	29	30	31	

[Exception\(al\) Failure - Breaking the STM32F1 Read-Out Protection](#)

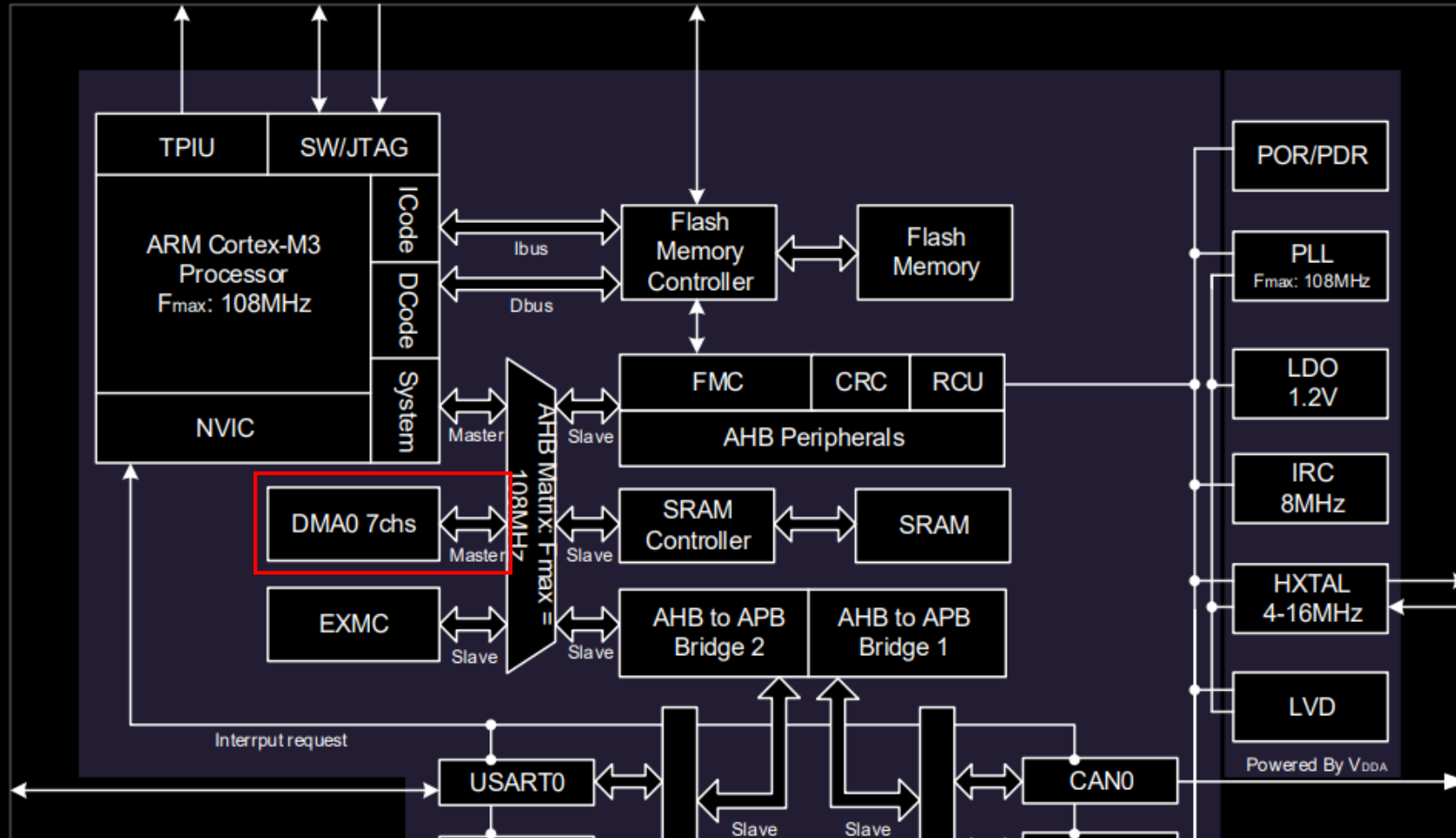
# VTOR Control Flow Redirection



- On GD32F103 flash memory access becomes locked down for all bus masters only if CPU debug module is enabled (C\_DEBUGEN bit in the DHCSR)
- Without enabling the debug module, we have access to SRAM, peripherals, etc.
- We can write a flash memory dumping firmware into SRAM
- But we cannot control the execution flow directly
- Instead, we indirectly redirect the control flow via the VTOR

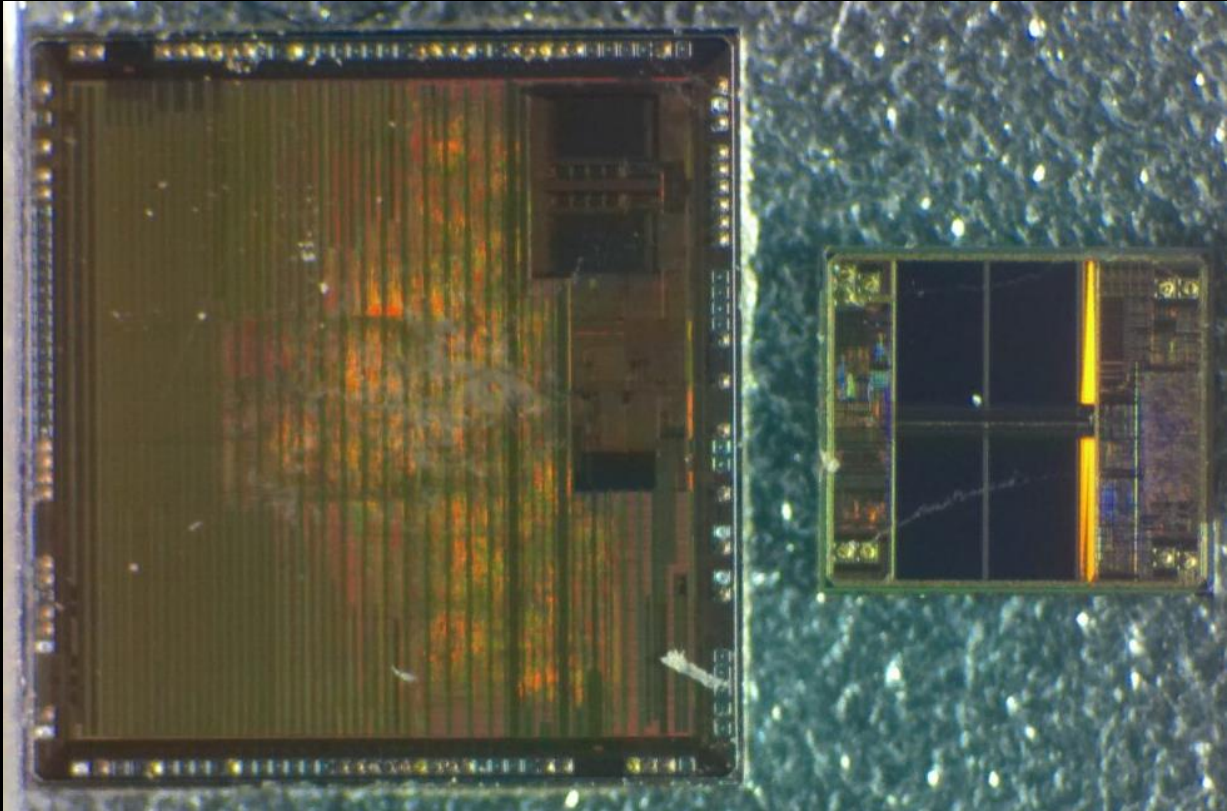
# DMA Access Exploitation

Figure 2-1. GD32F103x4/6/8/B block diagram

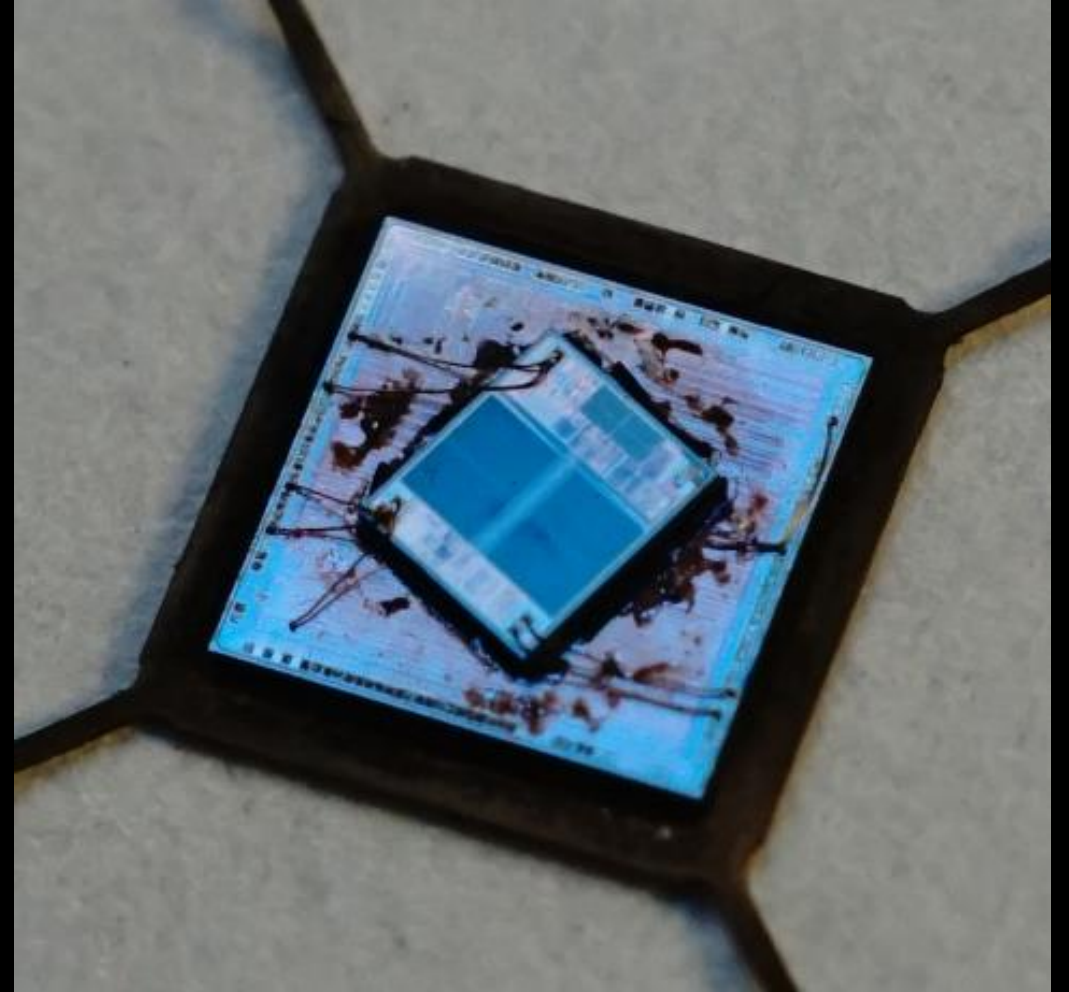




# Invasive Data Eavesdropping/ RDP Manipulation

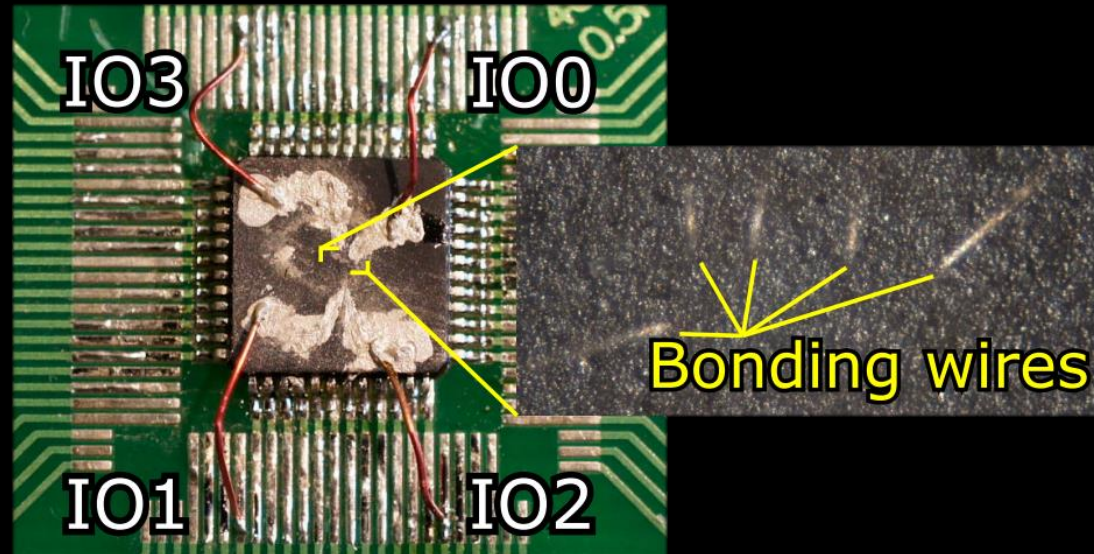
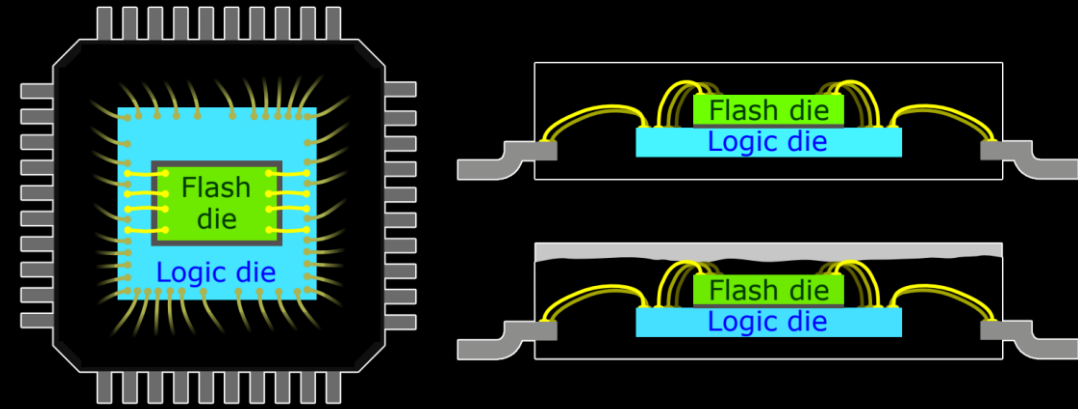


**GD32F103**



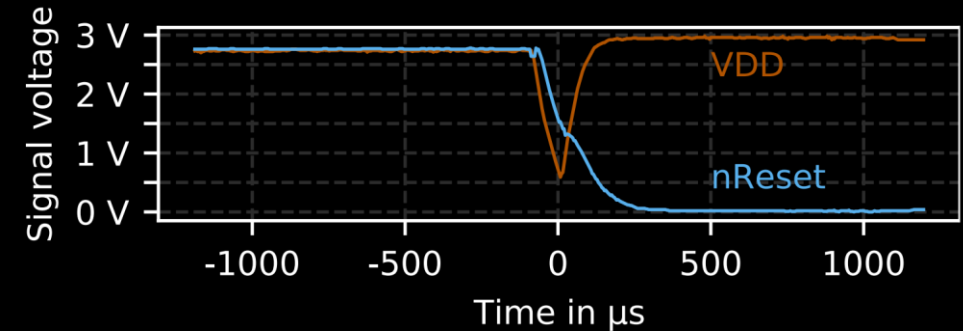
# Invasive Data Eavesdropping/ RDP Manipulation

- We can gain access to the bonding wires for eavesdropping
- To decode the firmware, it is necessary to reverse the obfuscation mechanism: word- and bit-permutation. Hard, but possible.
- Instead, we can actively manipulate QSPI to downgrade the lock level
  - Flipping only two bits is enough



# Shellcode Exec. via Glitch and FPB

- APM32F103, STM32F103
- 1. Upload a two-stage exploit firmware to the SRAM with a debugger and shut down the debugger afterwards
- 2. VDD Glitch to release RDP lock
- 3. Boot from SRAM (first stage) using BOOT0/1 pins
  - Flash is still not available (cause booting from SRAM)
  - But we can configure FPB to patch reset-vector fetch
  - FPB patch survive a device reset
- 4. Boot from FLASH using BOOT0/1 pins
  - Will actually run firmware from SRAM due to FPB configuration
  - Now the flash will be fully accessible



# Other materials



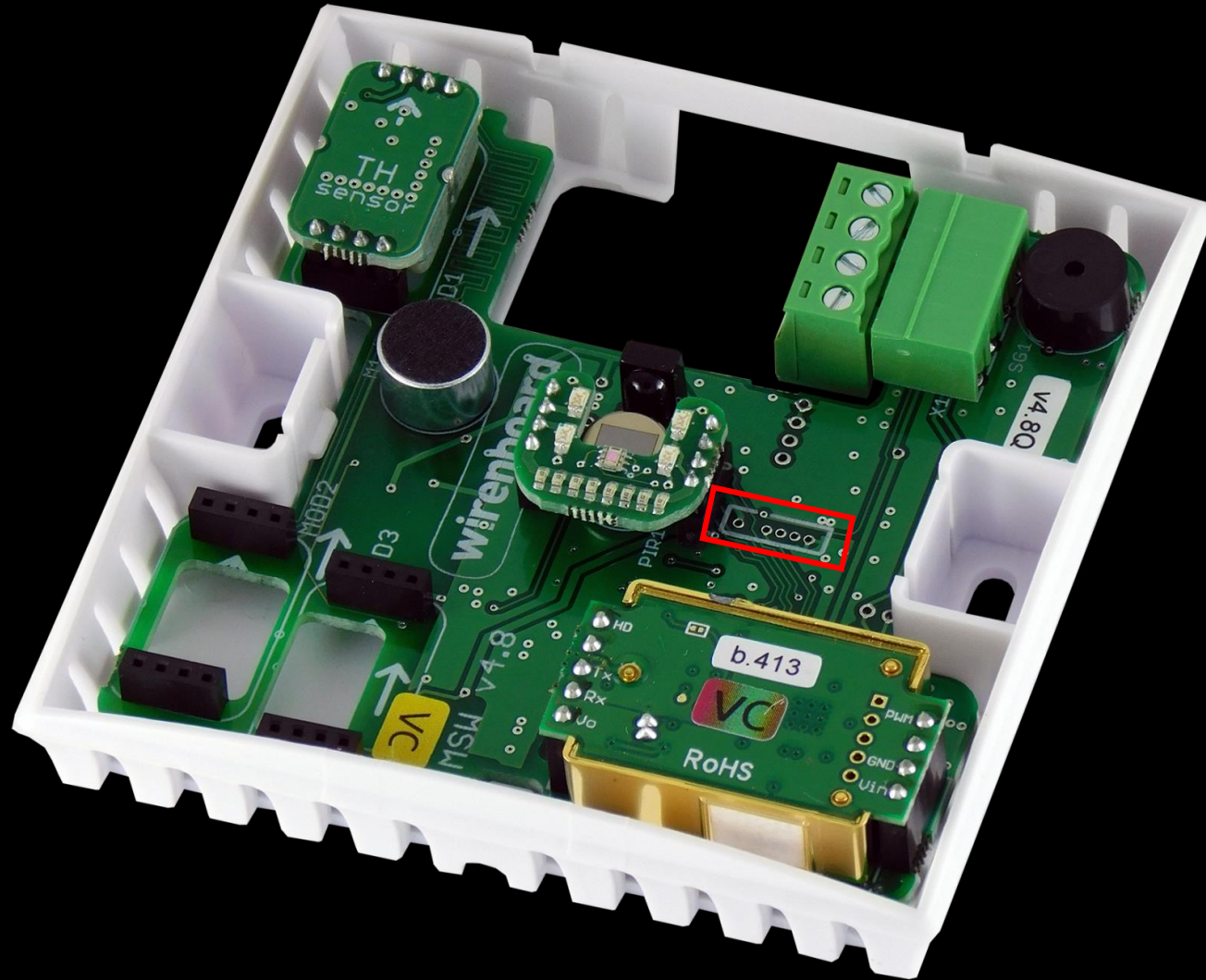
- [Microcontroller Readback Protection: Bypasses and Defenses](#)
- [nRF52 Debug Resurrection \(APPROTECT Bypass\)](#)
- [wallet.fail - Hacking the most popular cryptocurrency hardware wallets](#)
- [chip.fail - Glitching the silicon of the Internet-of-Things](#)

NO  
FF  
ONE  
2023

# GigaVulnerability #1



# How it all started: GD32E230



# J-Link tools vs openocd



- With default settings, J-Link tools are able to read DPIDR but **openocd** cannot
- Using a logic analyzer, we found out that DPIDR is readable with NRST is pulled-down
- Verified by manually pulling NRST to GND
- `reset_config srst_only srst_nogate connect_assert_srst`
- Is it RDP2 or just a pin reconfiguration at startup?



```
Open On-Chip Debugger 0.12.0
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : J-Link V9 compiled Sep  1 2016 18:29:50
Info : Hardware version: 9.60
Info : VTarget = 3.333 V
Info : clock speed 100 kHz
Info : SWD DPIDR 0x0bf11477
```

# GD32 Security Protection

- No protection (OB\_SPC = 0x5A)
- Protection level low (OB\_SPC not in {0x5A, 0xCC})
  - In debug mode, boot from SRAM or boot from boot loader mode, all operations to main flash is forbidden
  - If program back to no protection level a mass erase for main flash will be performed
- Protection level high (OB\_SPC = 0xCC)
  - When this level is programmed in debug mode, boot from SRAM or boot from boot loader mode is disabled
  - The main flash block is accessible by all operations from user code
  - The option byte cannot be erased



# What can we do in RESET?

- SRAM and Flash always read as zero
- Peripherals read but always return the reset value (expected)
- Debug registers work according to documentation

```
Open On-Chip Debugger
> mww 0x20000000 0x1234 SRAM
> mdw 0x20000000
0x20000000: 00000000

>
> mdw 0x08000000 Flash
0x08000000: 00000000

>
> mdw 0x48000000 Peripherals
0x48000000: 28000000

> mww 0x48000000 0x0
> mdw 0x48000000
0x48000000: 28000000

>
> mdw 0xE000EDF0 Debug Units
0xe000edf0: 02000000

> mww 0xE000EDF0 0x1
> mdw 0xE000EDF0
0xe000edf0: 02000000

> mww 0xE000EDF0 0xA05F0001
> mdw 0xE000EDF0
0xe000edf0: 02000001

>
> mdw 0xE0002008
0xe0002008: 00000000

> mww 0xE0002008 0x1234
> mdw 0xE0002008
0xe0002008: 00001234
```

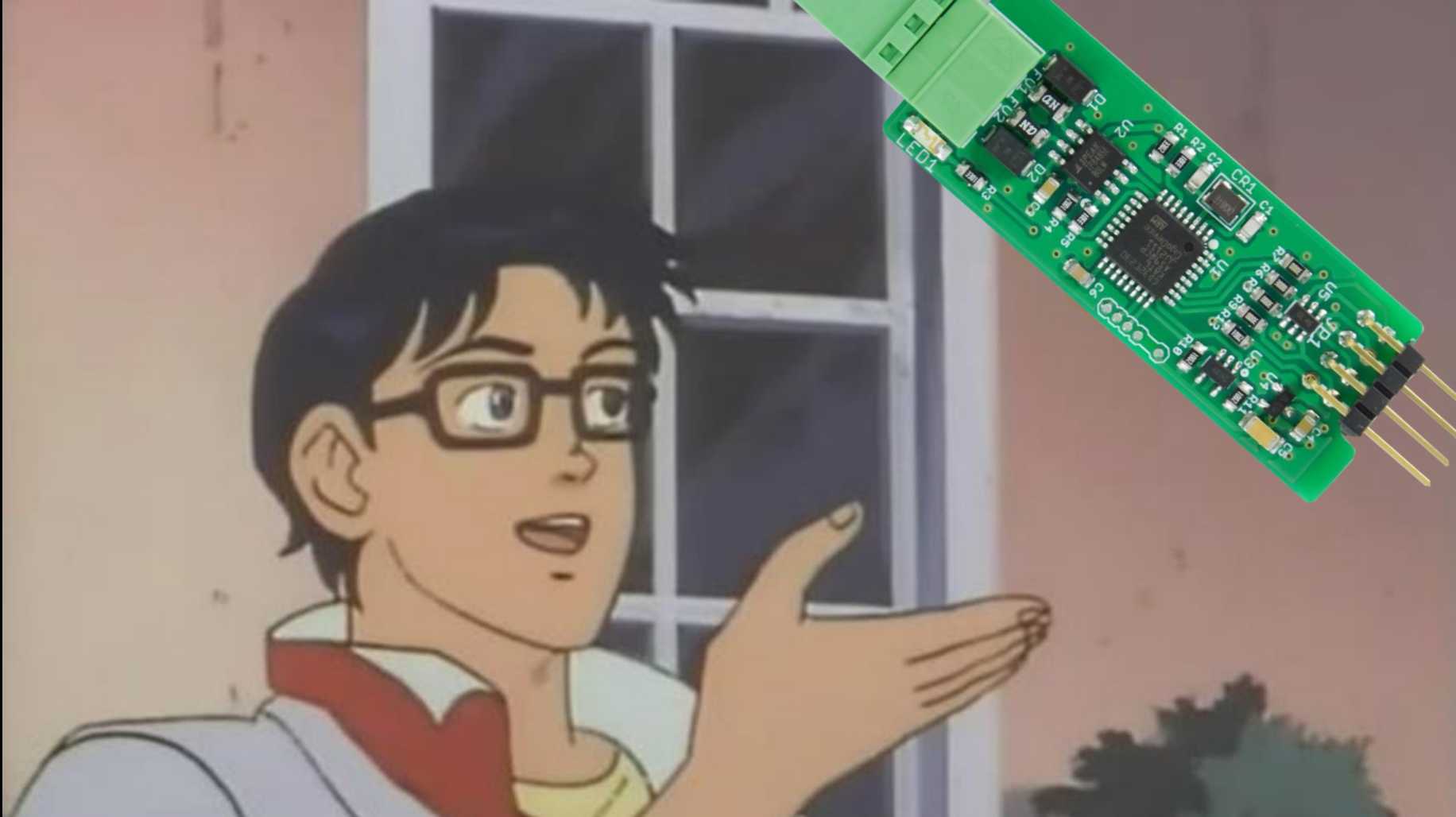


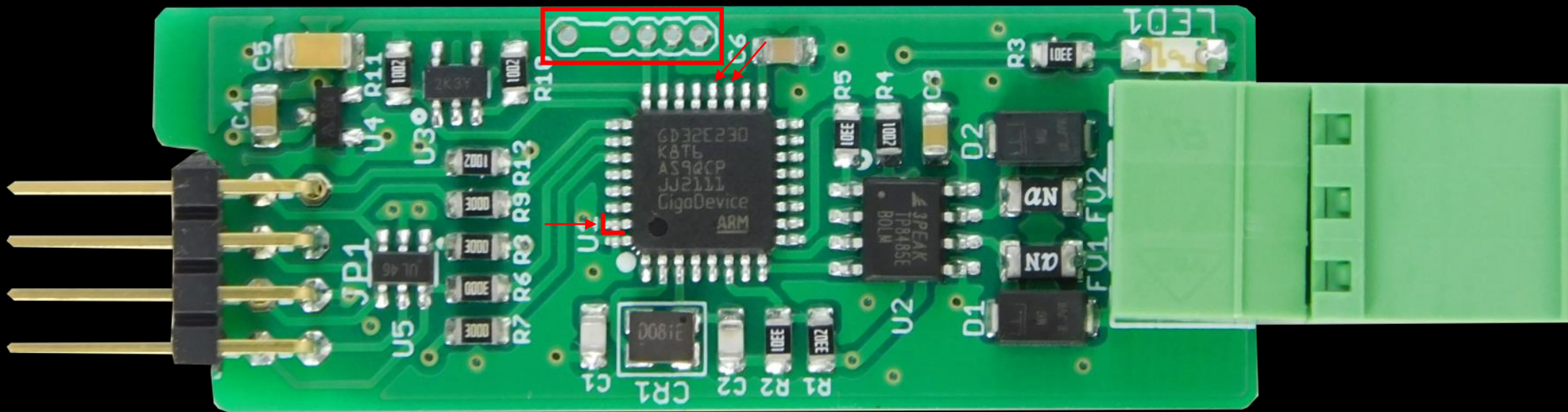
# RDP2 or not RDP2?



- Still not 100% sure
- Decided to buy the same MCU, lock it and check
- MCU is easy to buy ...
- ... but not a dev board

Is this a dev board?



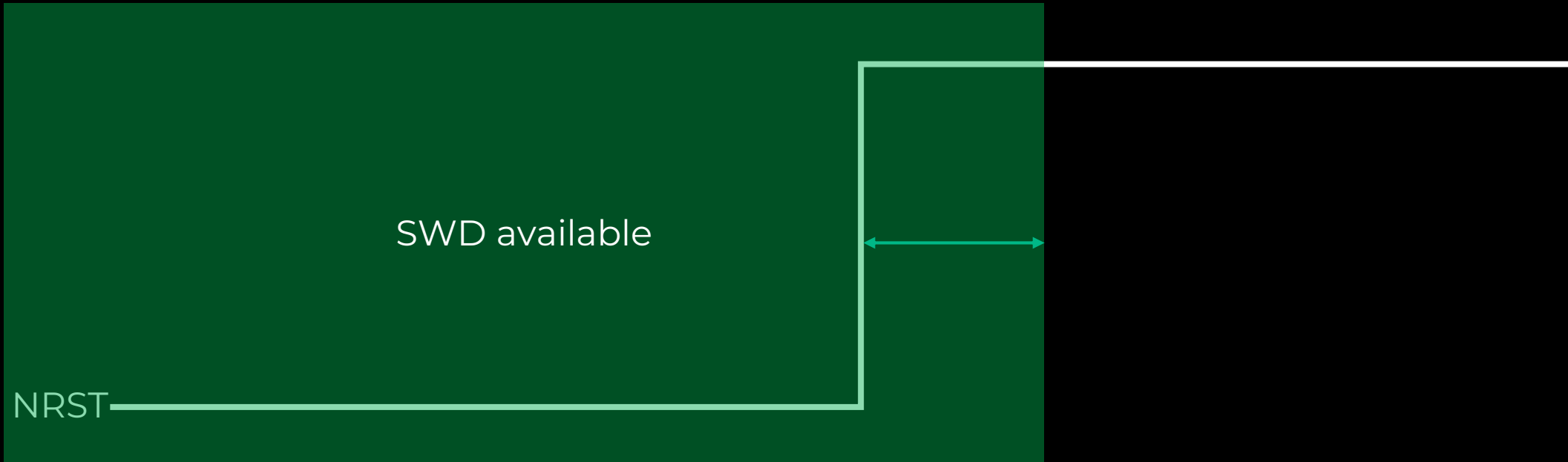


# RDP2 lock check result



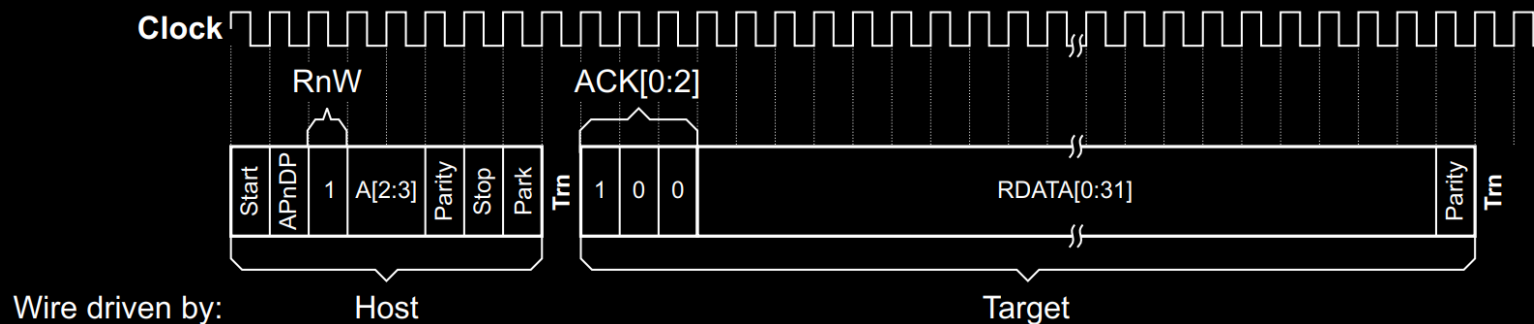
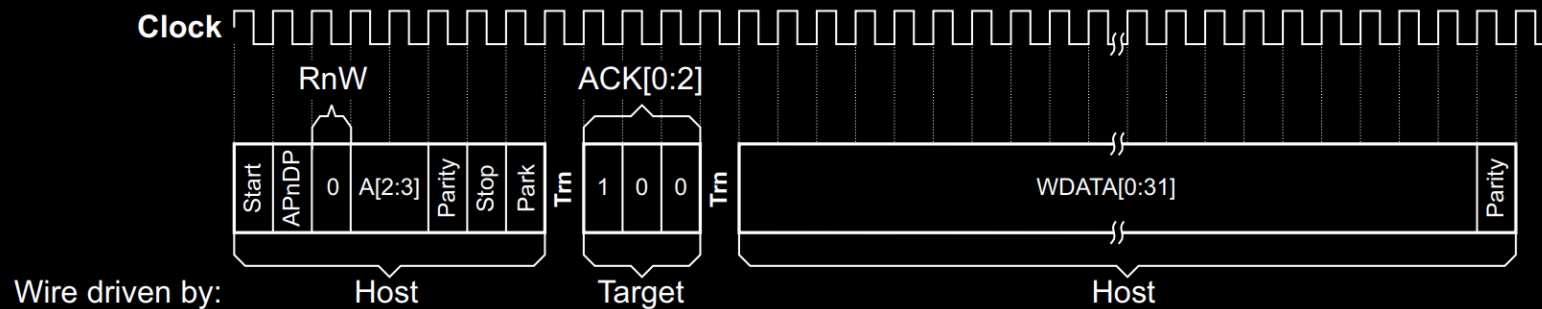
- Same behavior as before
- Almost sure it's RDP2 on the target
- It's probably impossible to dump the firmware 😞
- but...

# Maybe race?



# About SWD

- Two wires: SWCLK, SWDIO
- Packet-based protocol to read or write registers
- Arm Debug Interface Architecture Specification

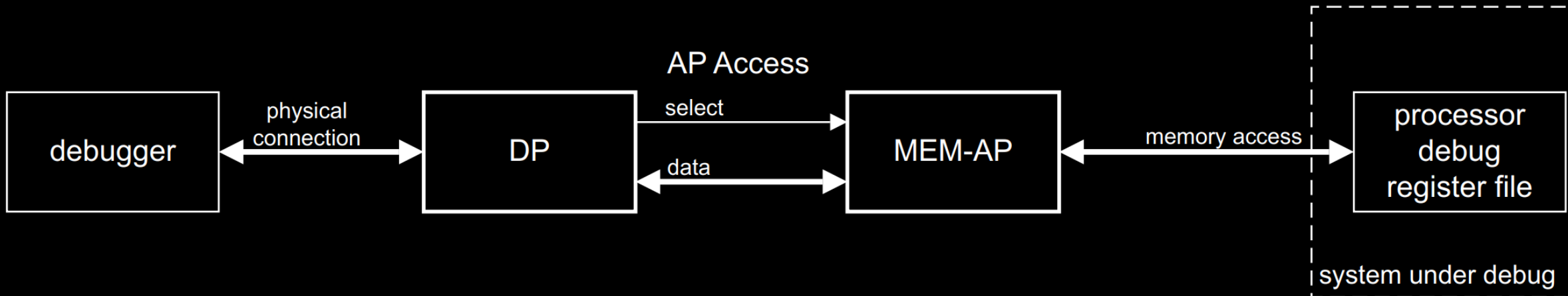


# DP

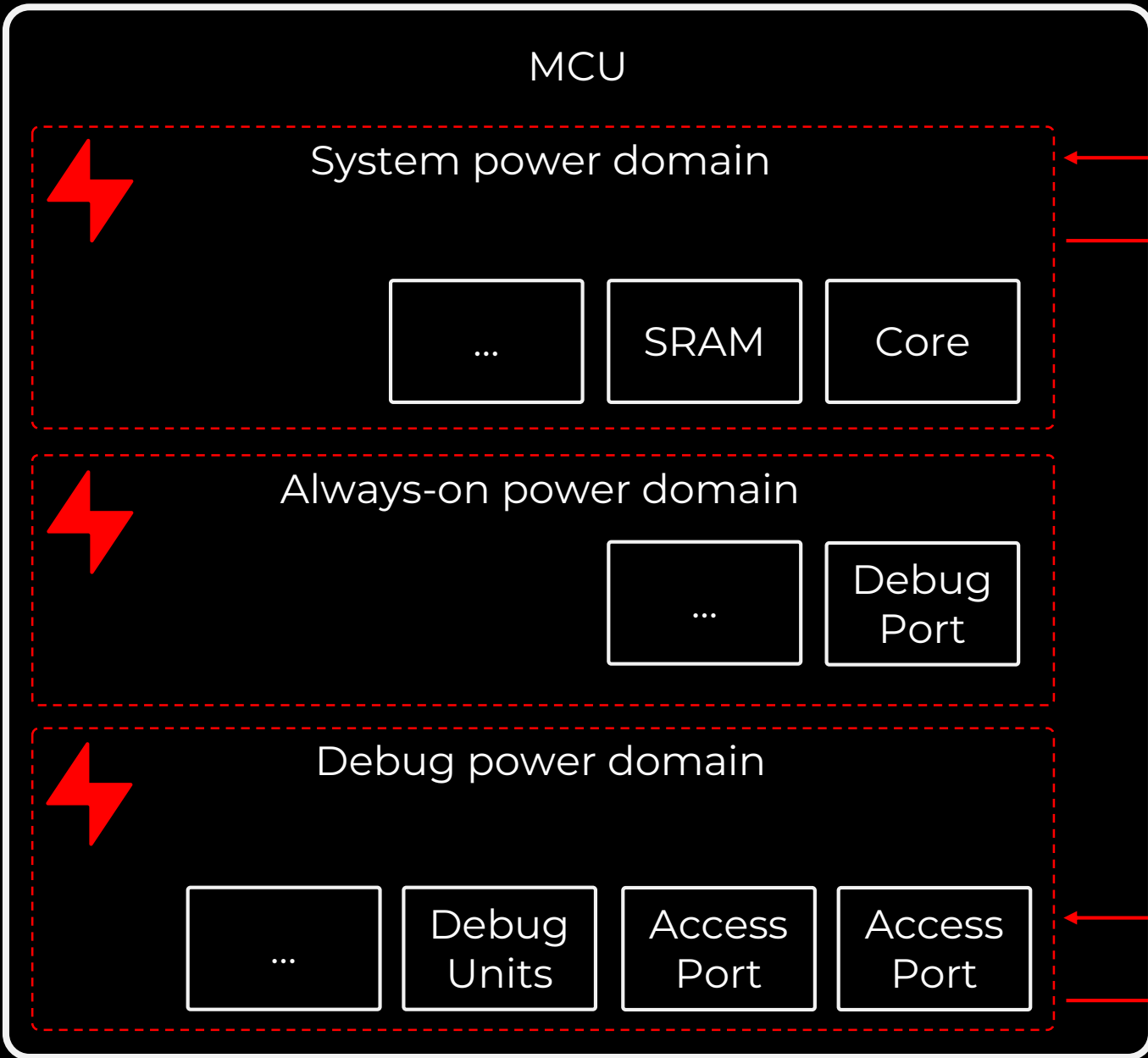
Address	Read	Write
0x00	IDCODE	ABORT
0x04	CTRL/STAT	
0x08		SELECT
0x0C	RDBUFF	

# MEM-AP

Address	Function	Description
0x00	CSW	Control/Status Word Register
0x04	TAR	Transfer Address Register
0x0C	DRW	Data Read/Write Register



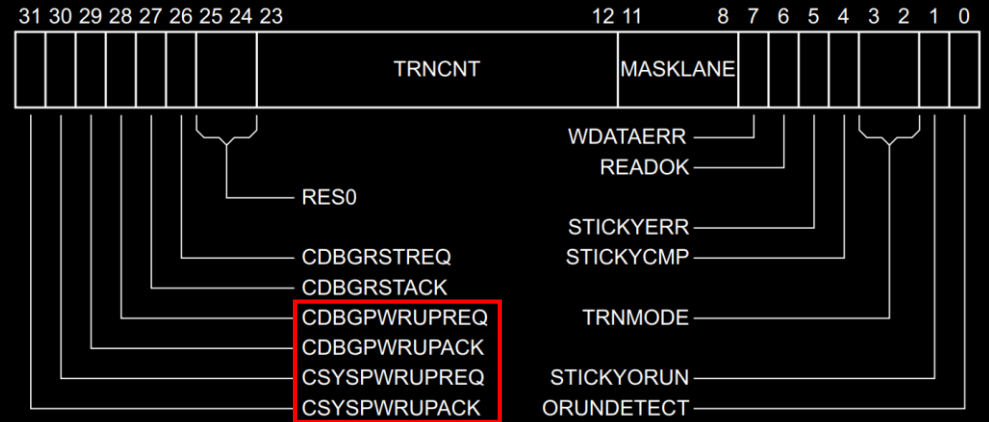




CSYSPWRUPREQ

CSYSPWRUPACK


CTRL/STAT



CDBGPWRUPREQ

CDBGPWRUPACK

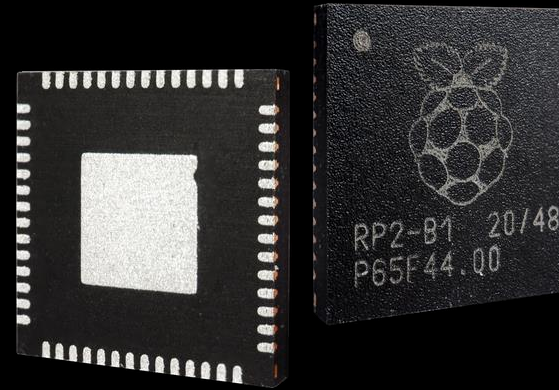
# Memory read

1. Preparation (interface reset, mode selection)
2. Read IDCODE (DPIDR)
3. Enable system & debug power domains
4. Select Bank 0x0 of MEM-AP
5. Configure CSW (e.g. 32-bit access without increments)
6. Configure TAR (target address)
-  7. Read DRW
8. Read RDBUFF

# First attempt: libjaylink + J-Link

- Easy to implement
- Works fast
- Downside: no known way to control NRST in sync with SWD
  - Due to USB, there are large floating delays
- Discovery: SWD works when NRST is HIGH
  - For this we controlled NRST with our hands
  - We have some valid ACKs for the packets

# RP2040 – New Hope



- Raspberry Pi Ltd
- 32-bit dual ARM Cortex-M0+ microcontroller
- 133 MHz (sometimes works fine at 250 MHz)

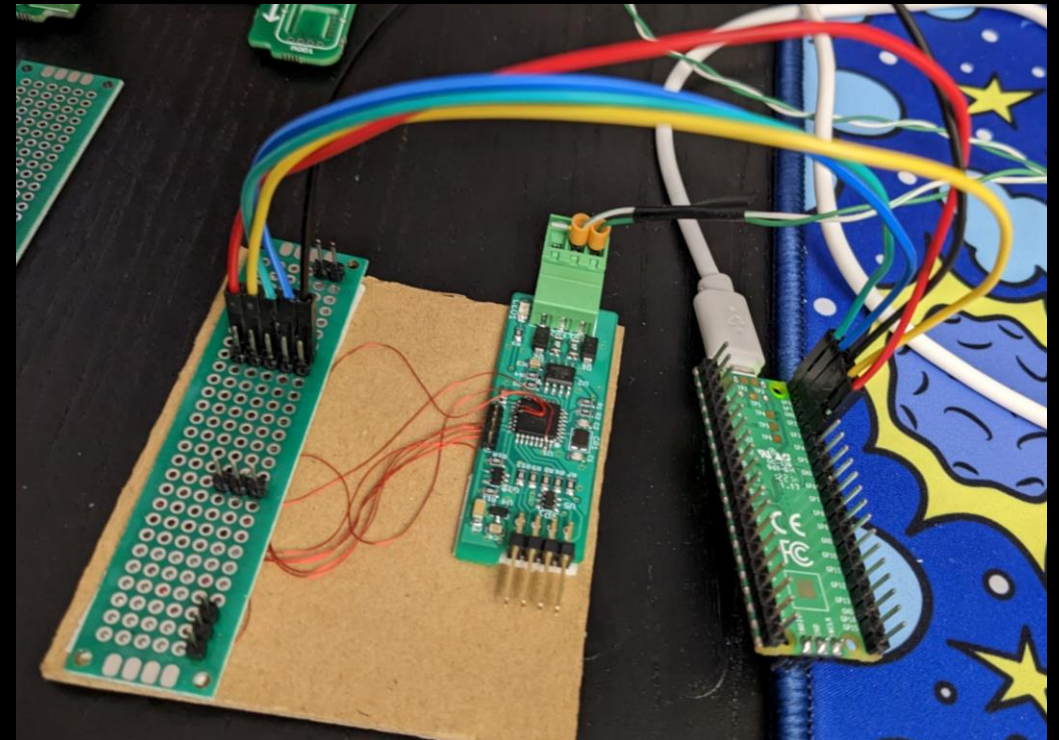


# PIO (Programmed Input–Output)

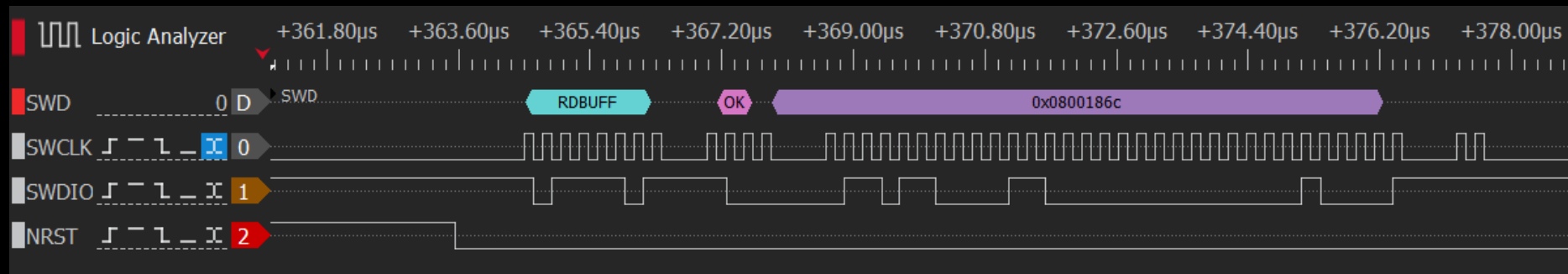
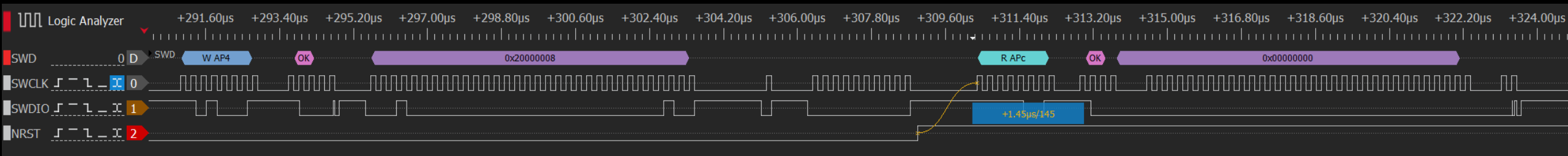
- Two blocks, four state machines each
- The state machines simultaneously execute programs aimed at working with input/output, and independent from the main CPU cores
- Can replace FPGA in some cases. Many protocols can be implemented
  - common protocols (if there are not enough special hardware blocks): UART, I2C, and more.
  - not very common protocols (for MCU): WS2812, DVI, VGA, and so on.
  - custom protocols
- Wide application in hardware security, especially in glitches
  - PicoFly
  - [ChipSHOUTER-PicoEMP](#)
  - [Starlink User Terminal Modchip](#)

## Picoprobe → SwdHack

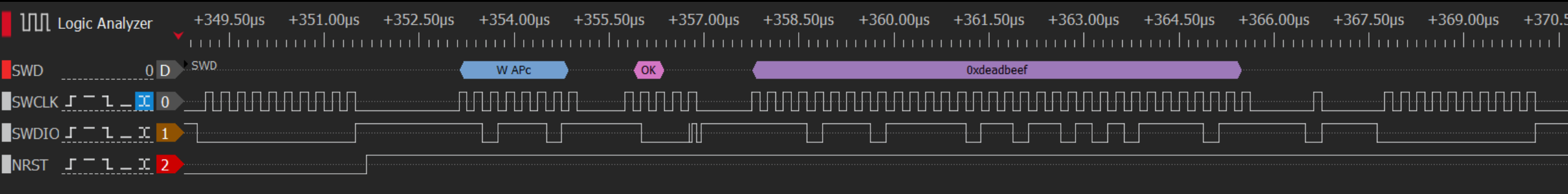
- Picoprobe allows a Pico/RP2040 to be used as a USB → SWD/UART bridge
  - SWD implemented as PIO-program
- The PIO program was taken for SWD and the C function for it from Picoprobe
- Implementation of a simple debugger that sends certain packets and synchronously drives NRST according to the algorithm described earlier
- Control of it done over USB (UART)



# Success! SRAM read



# SRAM write





# Results



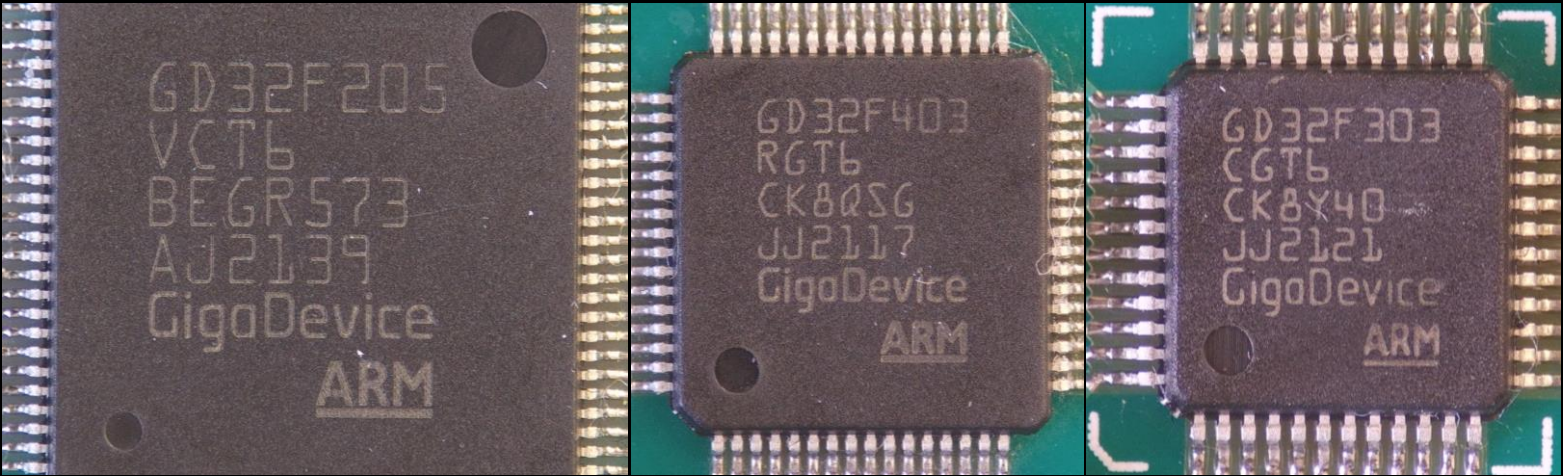
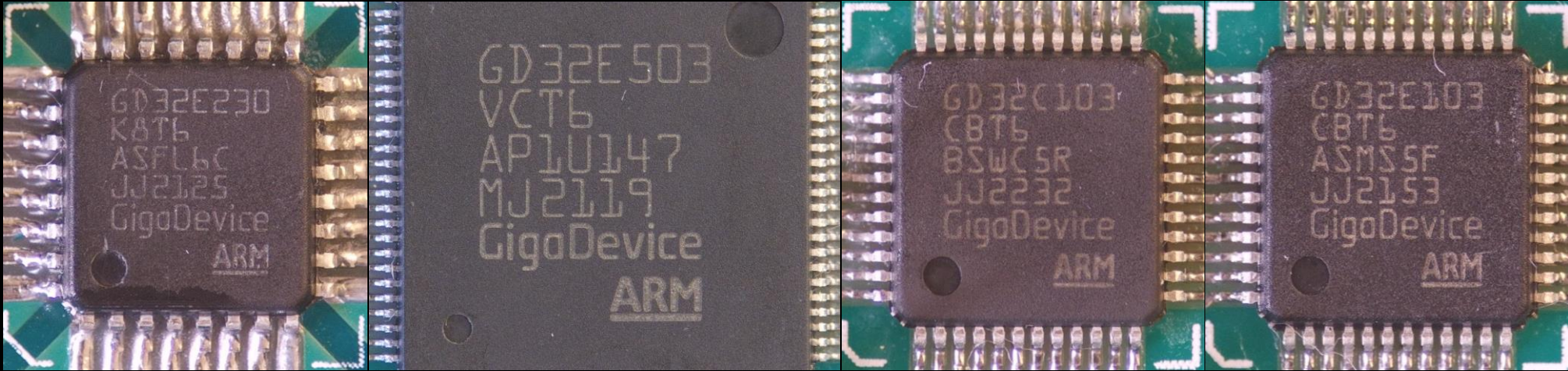
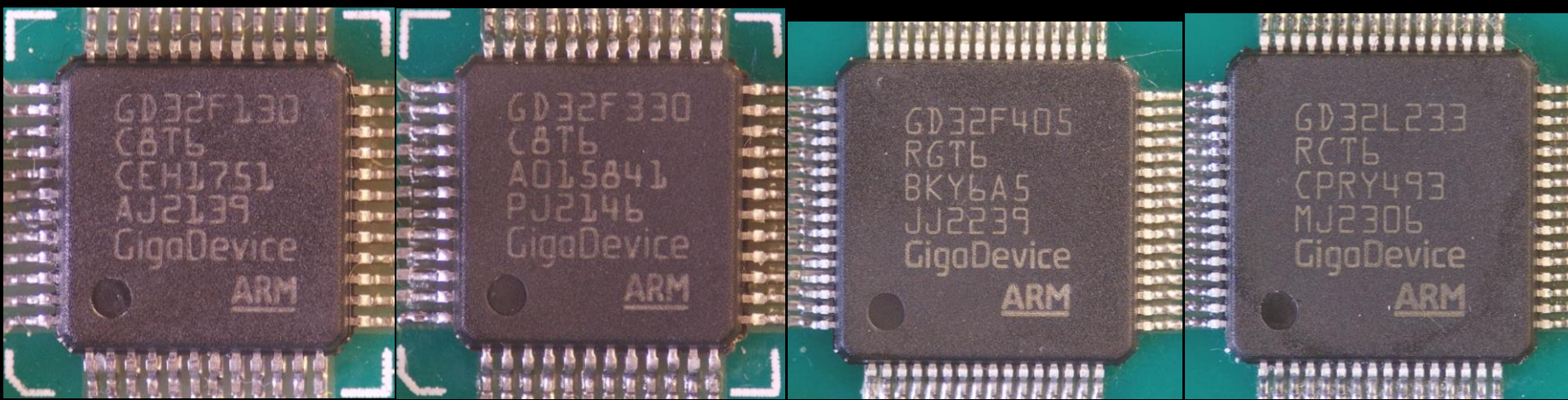
- We can read/write SRAM, Peripherals
  - Option Bytes have been checked for RDP2
  - Write is almost useless because of reset
- Still no direct access to Flash memory ☹️
- Cold-Boot Stepping (CBS) can be adopted
- In simple cases, CBS is not needed, a snapshot of SRAM taken at certain moments when waiting is enough
  - Encrypted WB firmware was decrypted in this way
- Also discovered: the readout lock is activated based on CDBGPWRUP and can be deactivated without a power-on reset

# Next steps

- Test the vulnerability on other families of GD32 microcontrollers
- Difficulty in checking all microcontrollers
  - Need to select one chip per family
- Other difficulties
  - Possibly need to group some families of chips according to defined criteria
- Decision: one chip per common manual
- Sourcing
  - ChipDip - very few options at that time (end of March 2023)
  - AliExpress - almost all found (delivered at the beginning of May 2023)
  - Some were bought relatively recently (August 2023)

Performance	Arm® Cortex®-M 32-bit MCUs				RISC-V 32-bit MCUs
	Cortex®-M23	Cortex®-M3	Cortex®-M4	Cortex®-M33	RISC-V
High-Performance		<p>GD32F207 120MHz, 3M/256K</p> <p>GD32F205 120MHz, 3M/256K</p>	<p>GD32F470 240MHz, 3M/768K</p> <p>GD32F427 200MHz, 3M/256K</p> <p>GD32F425 200MHz, 3M/256K</p> <p>GD32F450 200MHz, 3M/512K</p> <p>GD32F407 168MHz, 3M/192K</p> <p>GD32F405 168MHz, 3M/192K</p> <p>GD32F403 168MHz, 3M/128K</p>	<p>GD32W515 180MHz, 2048K/448K</p> <p>GD32E508 180MHz, 512K/128K</p> <p>GD32E507 180MHz, 512K/128K</p> <p>GD32E505 180MHz, 512K/128K</p> <p>GD32E503 180MHz, 512K/128K</p>	
Mainstream	<p>GD32L233 64MHz, 256K/32K</p>	<p>GD32F107 108MHz, 1M/96K</p> <p>GD32F105 108MHz, 1M/96K</p> <p>GD32F103 108MHz, 3M/96K</p> <p>GD32F101 56MHz, 3M/80K</p>	<p>GD32F307 120MHz, 1M/96K</p> <p>GD32F305 120MHz, 1M/96K</p> <p>GD32F303 120MHz, 3M/96K</p> <p>GD32C103 120MHz, 128K/32K</p> <p>GD32E103 120MHz, 128K/32K</p>	<p>GD32E501 100MHz, 512k/32K</p>	<p>GD32VF103 108MHz, 128K/32K</p>
Entry-Level	<p>GD32E232 72MHz, 64K/8K</p> <p>GD32E230 72MHz, 64K/8K</p>	<p>GD32F150 72MHz, 64K/8K</p> <p>GD32F130 48MHz, 64K/8K</p>	<p>GD32F350 108MHz, 128K/16K</p> <p>GD32F330 84MHz, 128K/16K</p> <p>GD32F310 72MHz, 64K/8K</p>		
Specific			<p>GD32FFPR 168MHz, 1M/128K</p>	<p>GD32EPRT 168MHz, 384K/96K+4M</p>	

NO  
FF  
ONE  
2023





# Success table

Family	MCU	Release	RDP2	GigaVulnerability #1
GD32F1x0	GD32F130C8T6	AJ2139		
GD32F3x0	GD32F330C8T6	PJ2146		No
GD32F4xx	GD32F405RGT6	JJ2239		
GD32L23x	GD32L233RCT6	MJ2306	Yes	Yes
GD32E23x	GD32E230K8T6	JJ2125		
GD32E50x	GD32E503VCT6	MJ2119		
GD32C10x	GD32C103CBT6	JJ2232		
GD32E10x	GD32E103CBT6	JJ2153		
GD32F20x	GD32F205VCT6	AJ2139	No	
GD32F30x	GD32F303CGT6	JJ2121		
GD32F403	GD32F403RGT6	JJ2117		

NO  
FF  
ONE  
2023

# GigaVulnerability #2



# CDBGPWRUPREQ

- As noted earlier, the readout protection lock is triggered when the Debug Domain is enabled (CDBGPWRUPREQ)
- Theory: maybe the readout protection lock can be disabled the same way in runtime?
- Proven: Yes! It can be used for RDP1 bypass
  1. Use the debugger to load into SRAM and run the firmware for dumping
    - Use UART as the dumping channel
  2. Reset DP CDBGPWRUPREQ bit (openocd: **chip.dap dpreg 0x4 0x0**)
  3. Signal the firmware to begin dumping
  4. ???
  5. PROFIT



# Success table



Family	MCU	Release	RDP2	GigaVulnerability #1	GigaVulnerability #2
GD32F1x0	GD32F130C8T6	AJ2139	Yes	No	Yes
GD32F3x0	GD32F330C8T6	PJ2146			No
GD32F4xx	GD32F405RGT6	JJ2239		Yes	Yes
GD32L23x	GD32L233RCT6	MJ2306		No	No
GD32E23x	GD32E230K8T6	JJ2125	No	Yes	Yes
GD32E50x	GD32E503VCT6	MJ2119			
GD32C10x	GD32C103CBT6	JJ2232			
GD32E10x	GD32E103CBT6	JJ2153			
GD32F20x	GD32F205VCT6	AJ2139			
GD32F30x	GD32F303CGT6	JJ2121			
GD32F403	GD32F403RGT6	JJ2117			

NO  
FF  
ONE  
2023

# GigaVulnerability #3



# Comeback to F-series

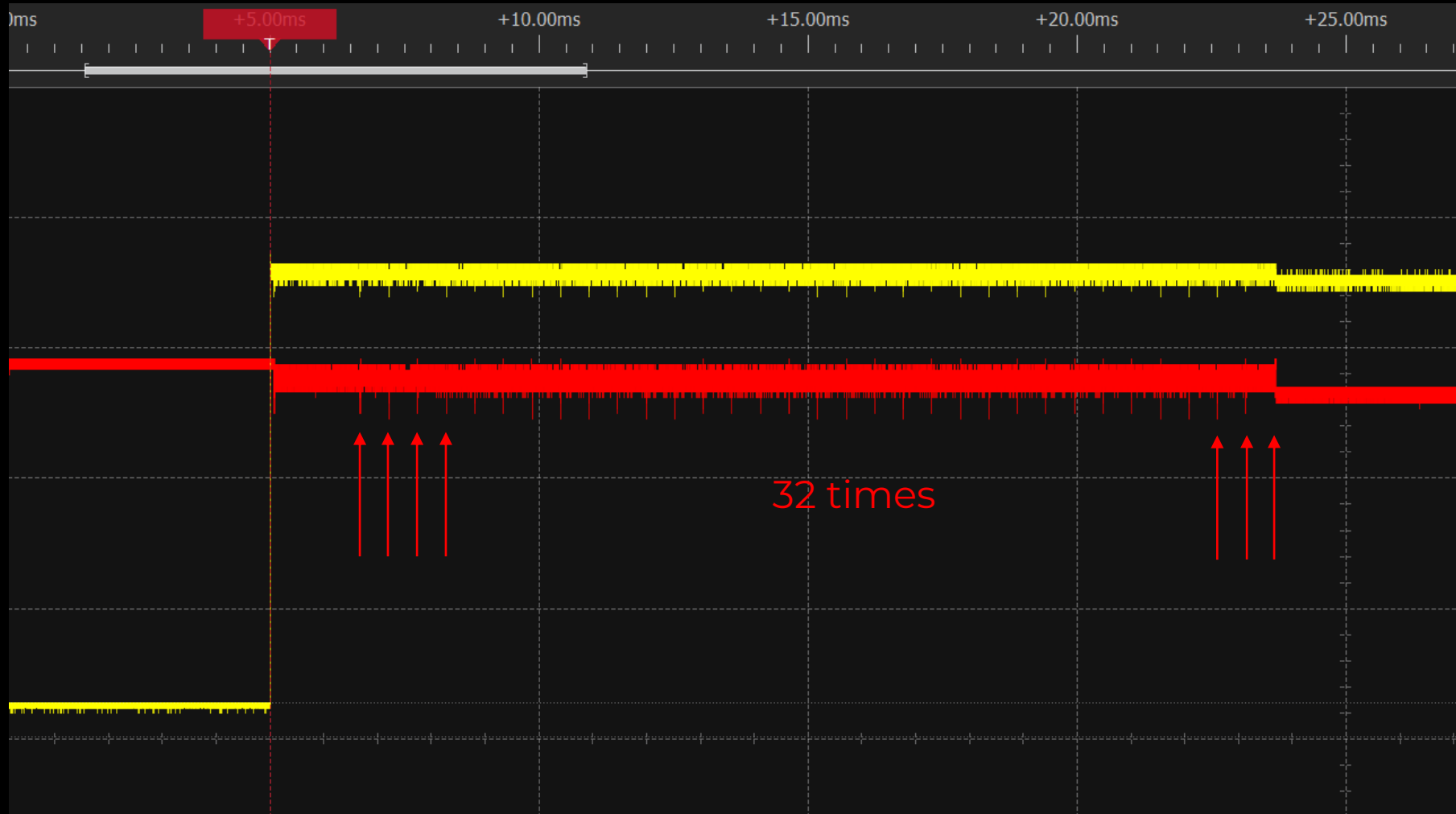
- SWD not working immediately after high NRST
- Accidental discovery: the attack works after power-up reset!
  - Power off
  - Pull NRST to GND
  - Power on
- Race window is much larger than in the E-series
  - More than 1600  $\mu$ S on GD32F130 vs  $\sim$ 20  $\mu$ S on GD32E230
  - Seems useless, because all SRAM is in an uninitialized state
- Can we get something useful out of this?

# First attempts

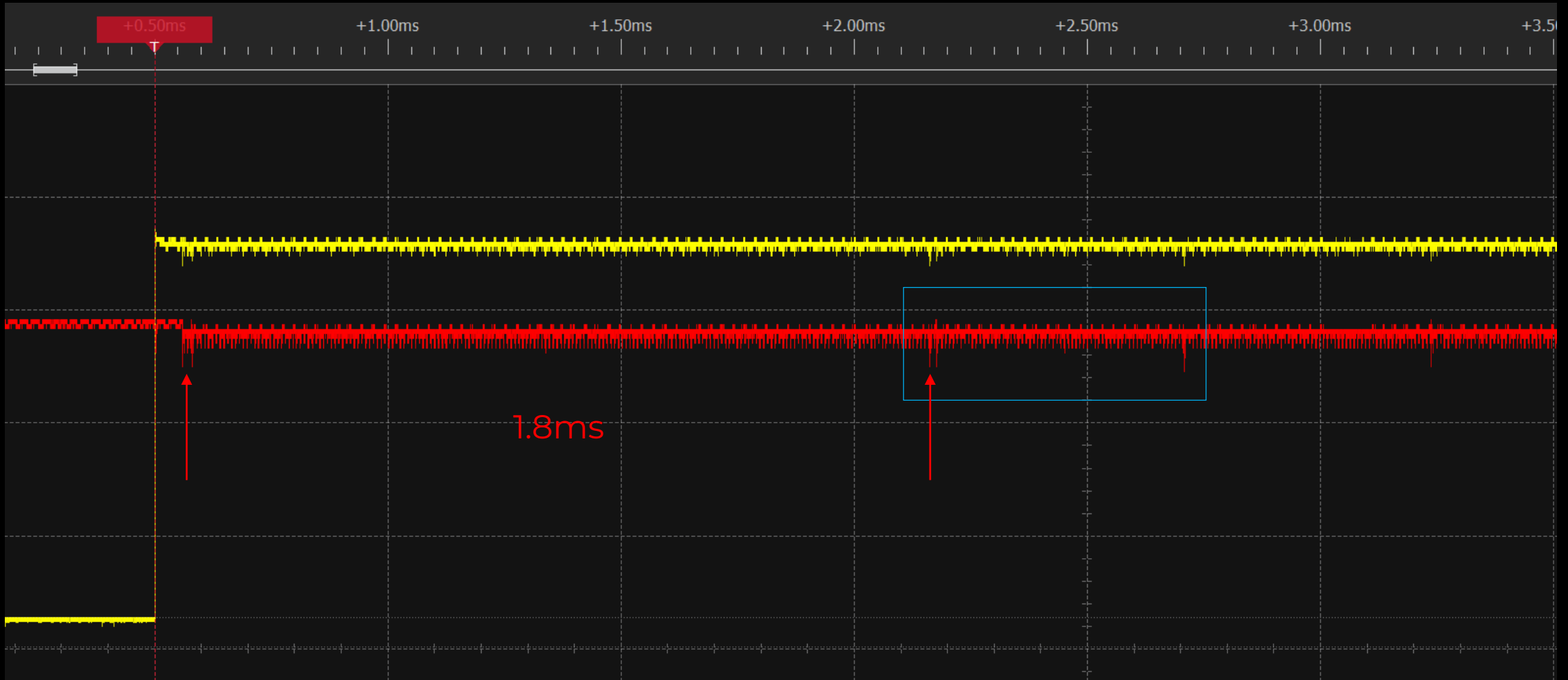


- Manipulation with VTOR?
  - Seems to take a reset value anyway
- Core debugging?
  - It seems that if I enable C\_DEBUG during the race window, the core won't start (even in RDP0)
- Voltage Glitch?

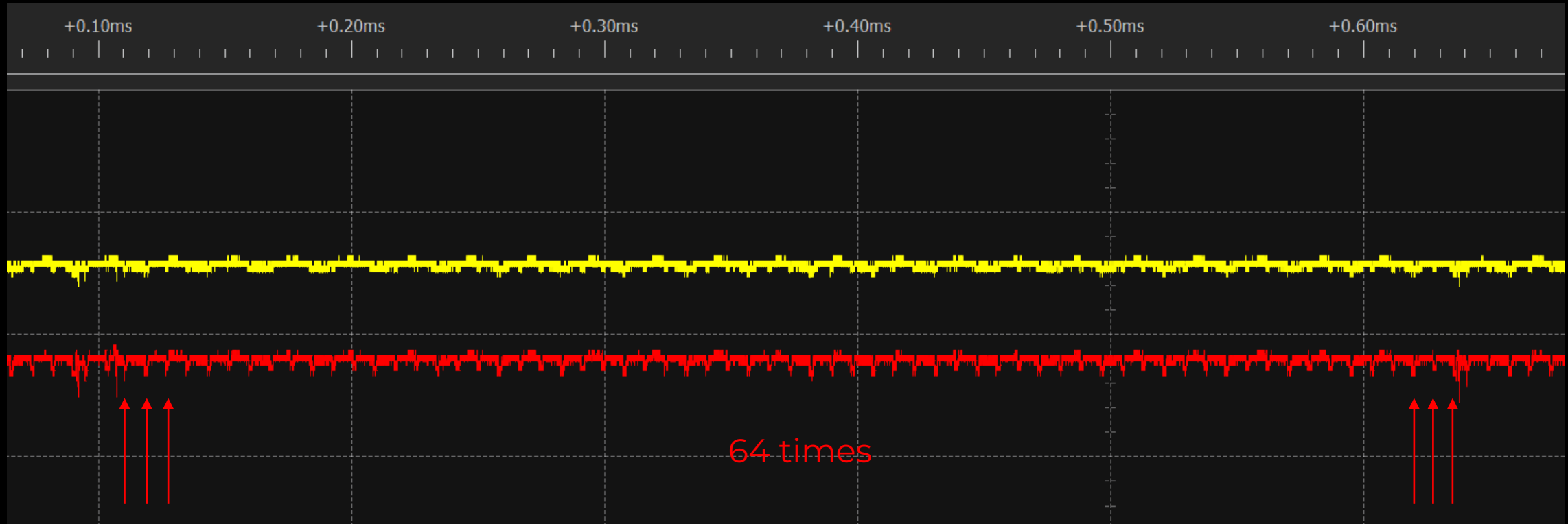
# Power-Analysis



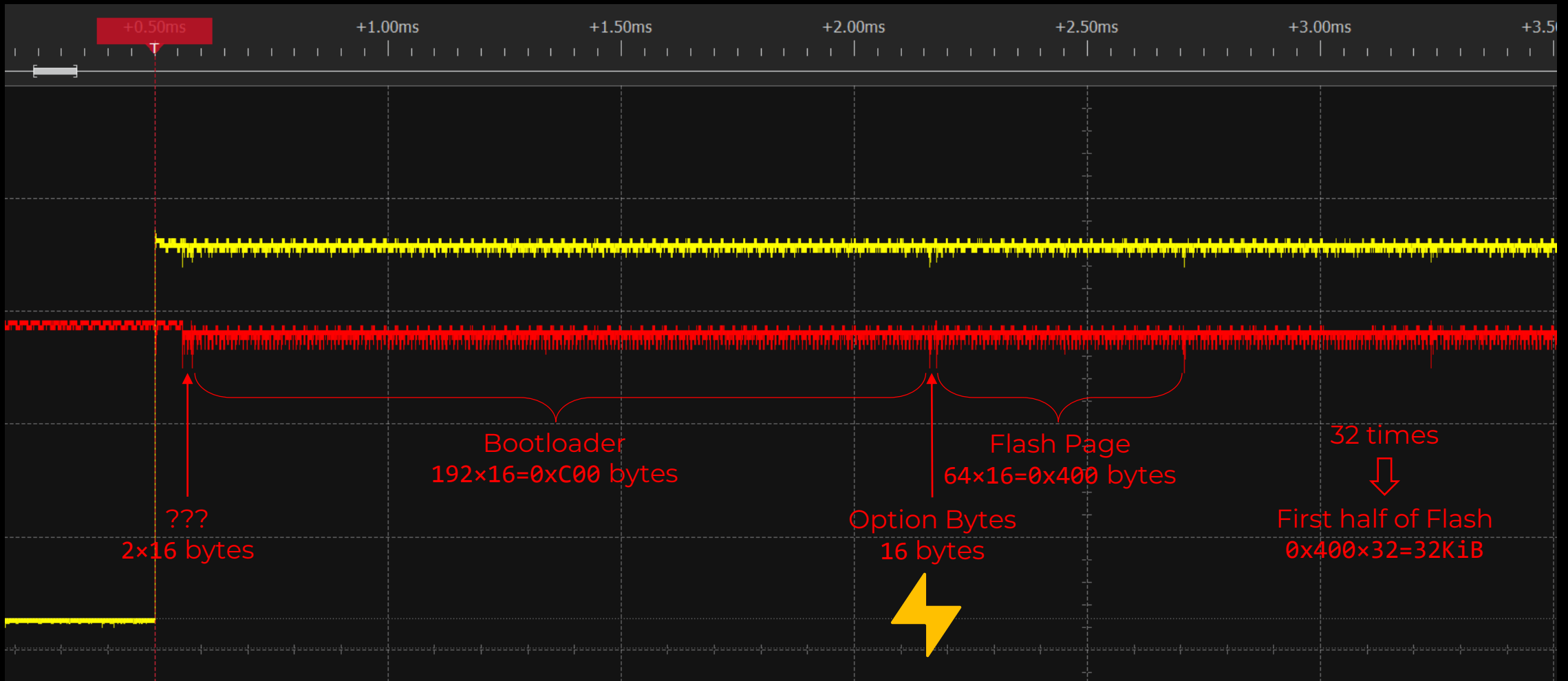
# Power-Analysis



# Power-Analysis

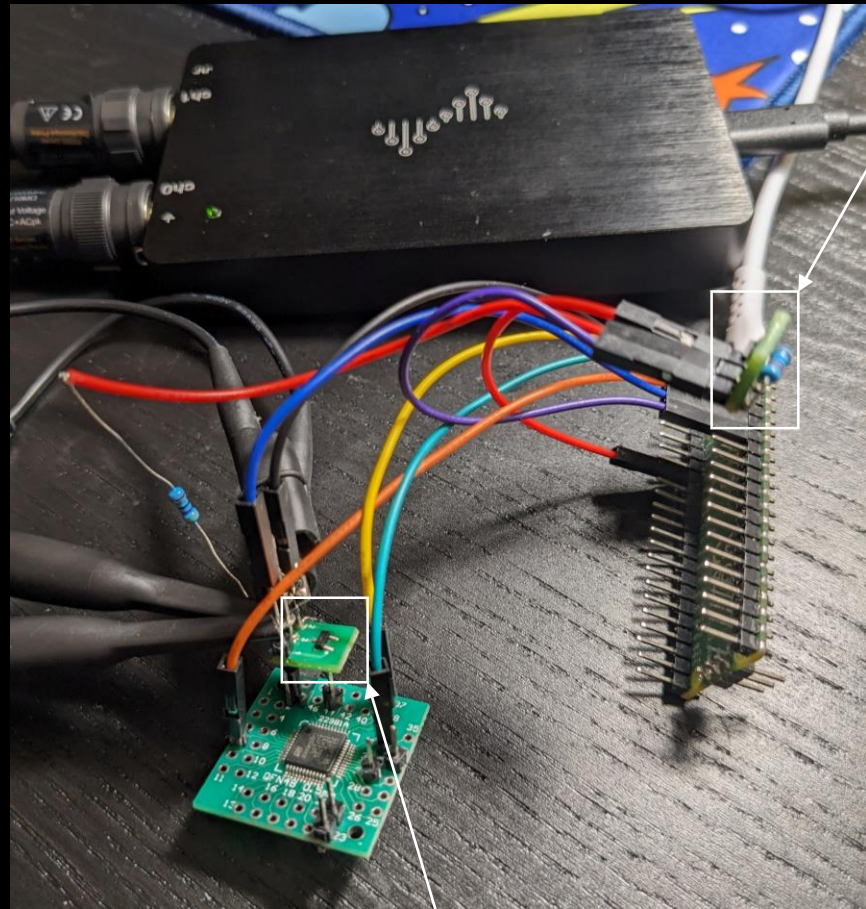


# Power-Analysis: interpretation of results

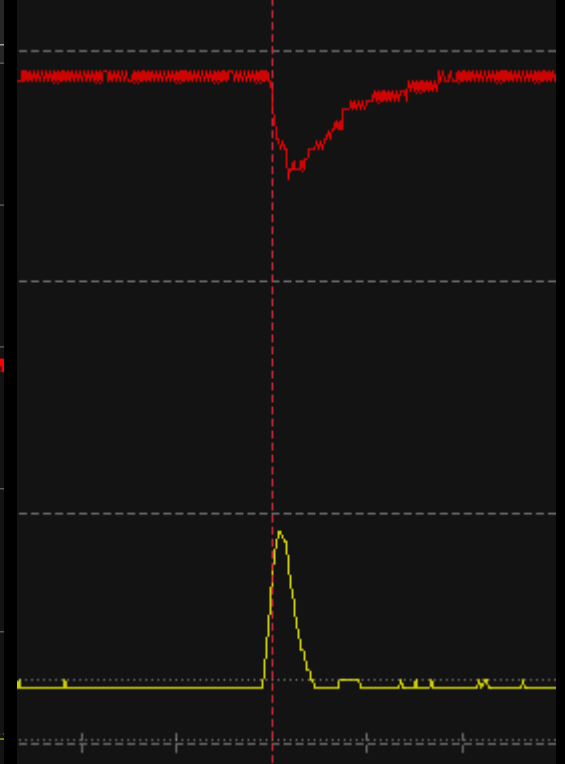




# My First Voltage Glitcher (which doesn't work)



• Power control

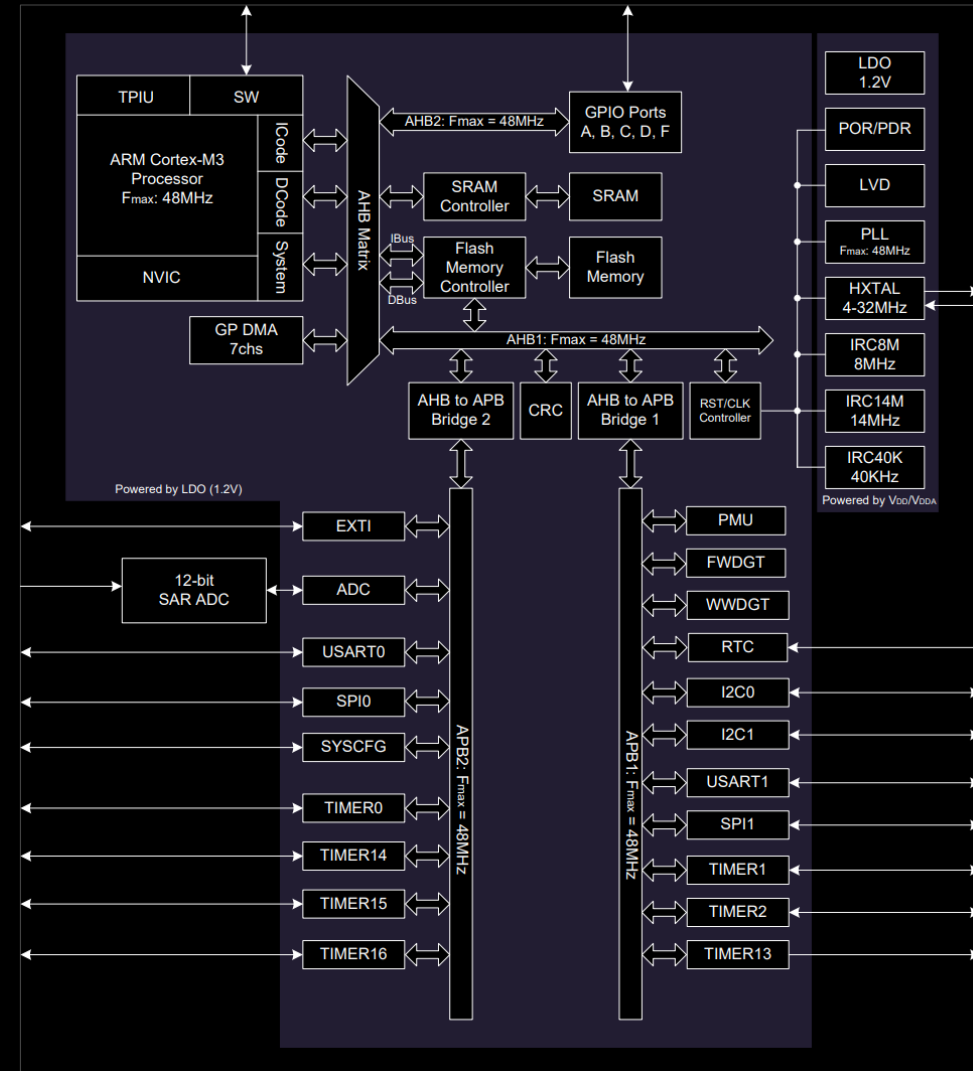


Glitch Shape

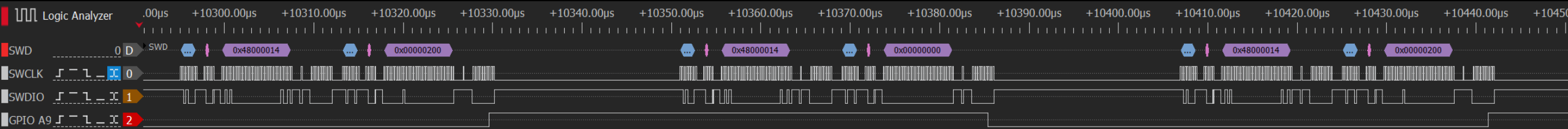
• IRLML2502 N-channel MOSFET like in ChipWhisperer-Lite

# Another attack through SWD

- Core, SRAM, FMC – all tested
- Many untested peripherals TBD
- Maybe something will be useful even after disabling SWD
- Let's start simple



# Peripheral: PIN pull-up



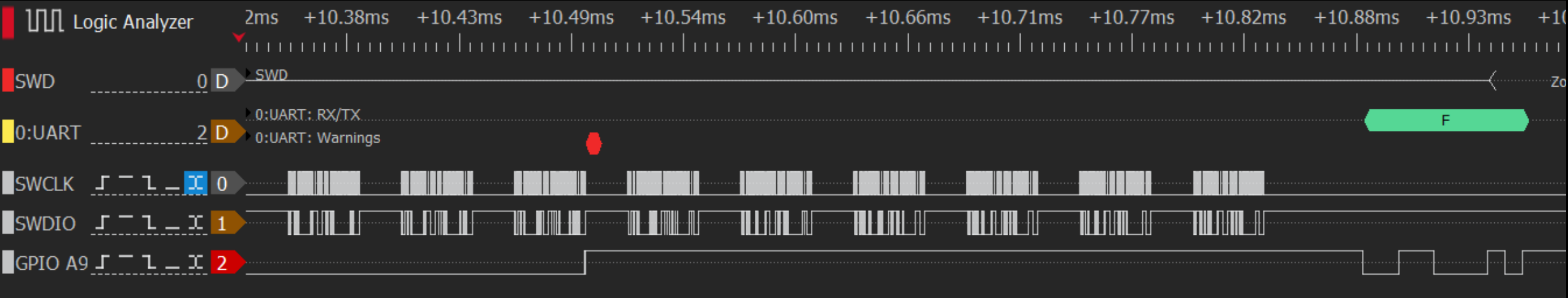
```
// GD32F1x0 GPIO

// rcu
swd_memwrite_noreset(0x40021014, 0x00020014); swdsleep(); // gpioa

// gpio
swd_memwrite_noreset(0x48000000, 0x28040000); swdsleep(); // gpio A9 output mode

swd_memwrite_noreset(0x48000014, 1 << 9); swdsleep(); // gpio A9 HIGH
swd_memwrite_noreset(0x48000014, 0); swdsleep(); // gpio A9 LOW
swd_memwrite_noreset(0x48000014, 1 << 9); swdsleep(); // gpio A10 HIGH
swd_memwrite_noreset(0x48000014, 0); swdsleep(); // gpio A9 LOW
```

# Peripheral: UART



```
// GD32F1x0 GPIO

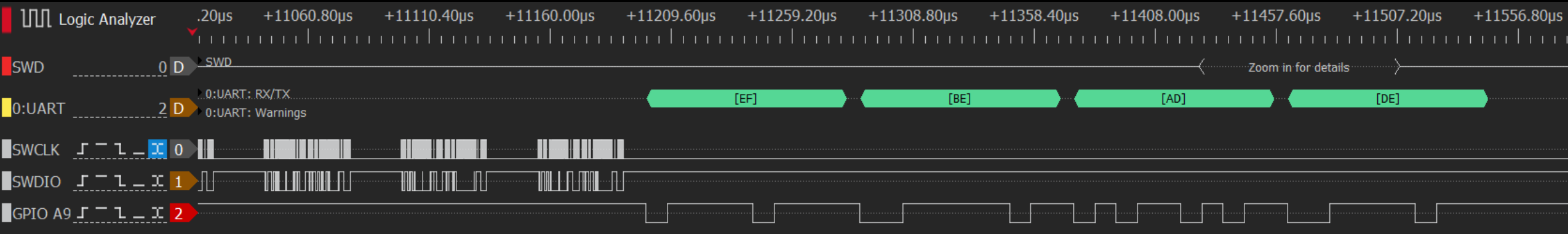
// rcu
swd_memwrite_noreset(0x40021014, 0x00020014); swdsleep(); // gpioa
swd_memwrite_noreset(0x40021018, 0x00004000); swdsleep(); // usart0

// gpio
swd_memwrite_noreset(0x48000000, 0x28280000); swdsleep(); // gpio A9-A10 AF
swd_memwrite_noreset(0x48000004, 0x00000000); swdsleep(); // output mode
swd_memwrite_noreset(0x48000008, 0x0c000000); swdsleep(); // output speed
swd_memwrite_noreset(0x4800000c, 0x24140000); swdsleep(); // pull-up/down cfg
swd_memwrite_noreset(0x48000024, 0x00000110); swdsleep(); // AF select (USART0)

// usart
swd_memwrite_noreset(0x40013800, 0x0000000c); swdsleep();
swd_memwrite_noreset(0x40013808, 0x00000080); swdsleep();
swd_memwrite_noreset(0x4001380c, 0x00000045); swdsleep();
swd_memwrite_noreset(0x40013800, 0x0000000d); swdsleep();

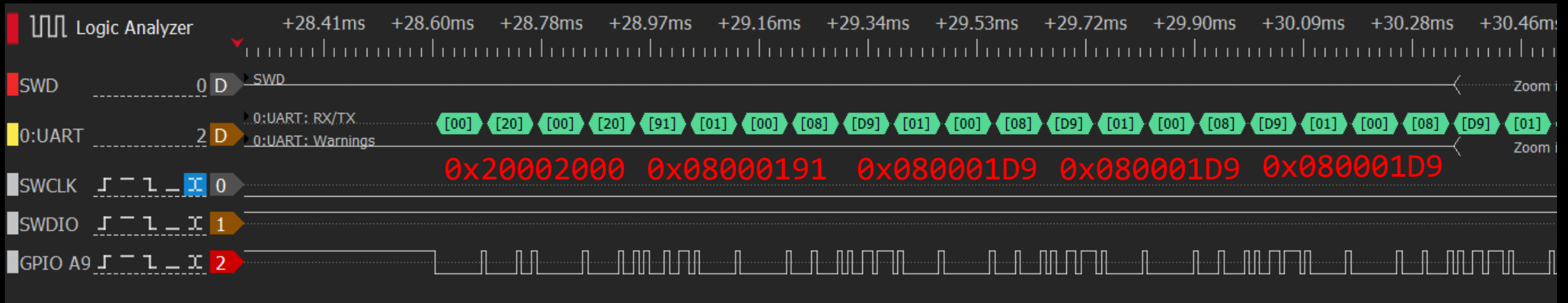
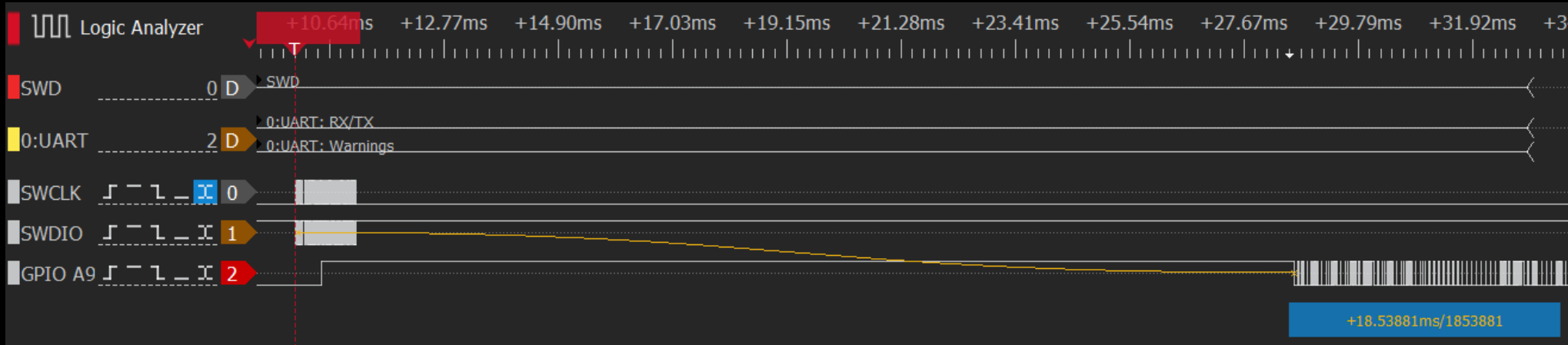
swd_memwrite_noreset(0x40013828, 'F'); swdsleep();
```

# Peripheral: UART+DMA from RAM?



```
swd_memwrite_noreset(0x20000000, 0xdeadbeef); swdsleep();  
swd_memwrite_noreset(0x40020020, 0x00000004); swdsleep(); // DMA_CHxCNT  
swd_memwrite_noreset(0x40020024, 0x40013828); swdsleep(); // DMA_CHxPADDR  
swd_memwrite_noreset(0x40020028, 0x20000000); swdsleep(); // DMA_CHxMADDR  
swd_memwrite_noreset(0x4002001C, 0x00000091); swdsleep(); // DMA_CHxCTL
```

# Peripheral: UART+DMA from FLASH???



# Success table



Family	MCU	Release	RDP2	GigaVulnerability #1	GigaVulnerability #2	GigaVulnerability #3
GD32F1x0	GD32F130C8T6	AJ2139	Yes	No	Yes	Yes
GD32F3x0	GD32F330C8T6	PJ2146			No	
GD32F4xx	GD32F405RGT6	JJ2239			Yes	
GD32L23x	GD32L233RCT6	MJ2306		Yes	No	No
GD32E23x	GD32E230K8T6	JJ2125			Yes	
GD32E50x	GD32E503VCT6	MJ2119				
GD32C10x	GD32C103CBT6	JJ2232	No	Yes		
GD32E10x	GD32E103CBT6	JJ2153				
GD32F20x	GD32F205VCT6	AJ2139				
GD32F30x	GD32F303CGT6	JJ2121				
GD32F403	GD32F403RGT6	JJ2117				

# FMC: E/L vs F family

- GD32E23x
  - 0~2 waiting time within 64K bytes when CPU executes an instruction
- Almost the same for GD32L23x
- E/L doesn't cache flash pages on startup
- Small delay on each reset to read Option Bytes (~20μS, acceptable)
- Small race window on each reset
- GD32F1x0
  - No waiting time within 32K bytes when CPU executes an instruction
  - A long delay when fetching 32K ~ 64K bytes data from flash
- Long delay on power-on reset (~18ms)
  - Needed to fill the cache
- Option Bytes also cached
- Big race window on power-on reset
- No race window on other resets



# Success table



Family	MCU	Release	RDP2	GigaVulnerability #1	GigaVulnerability #2	GigaVulnerability #3	
GD32F1x0	GD32F130C8T6	AJ2139	Yes	No	Yes	Yes	
GD32F3x0	GD32F330C8T6	PJ2146			No		No
GD32F4xx	GD32F405RGT6	JJ2239			Yes		Yes
GD32L23x	GD32L233RCT6	MJ2306		Yes	No		No
GD32E23x	GD32E230K8T6	JJ2125			Yes		
GD32E50x	GD32E503VCT6	MJ2119			Yes		
GD32C10x	GD32C103CBT6	JJ2232	No	Yes	Yes		
GD32E10x	GD32E103CBT6	JJ2153					
GD32F20x	GD32F205VCT6	AJ2139					
GD32F30x	GD32F303CGT6	JJ2121					
GD32F403	GD32F403RGT6	JJ2117					

# Results



- Lots of experience in security of microcontrollers
- New techniques to bypass readout protection
- Three vulnerabilities reported to GigaDevice

# Conclusions



- Some implementations of readout protection technologies are far from perfect
- Consider this when developing your own devices
  - Restriction of physical access to the chip
  - Control the accessibility of the end-product
  - Other points
- In any case, one day your defense will be broken.  
Be prepared for this



**NO**  
**FF**  
**ONE**  
**2023**

