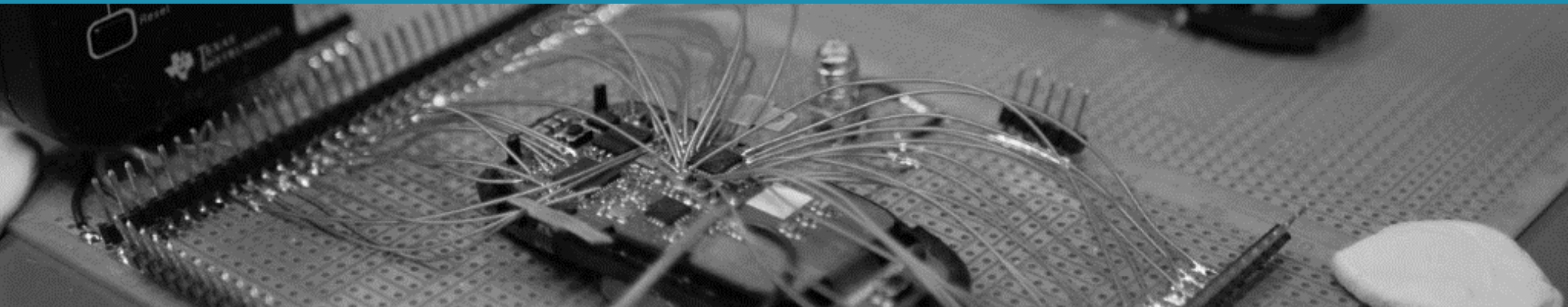
KU LEUVEN

On the susceptibility of Texas Instruments SimpleLink platform microcontrollers to non-invasive physical attacks

COSIC

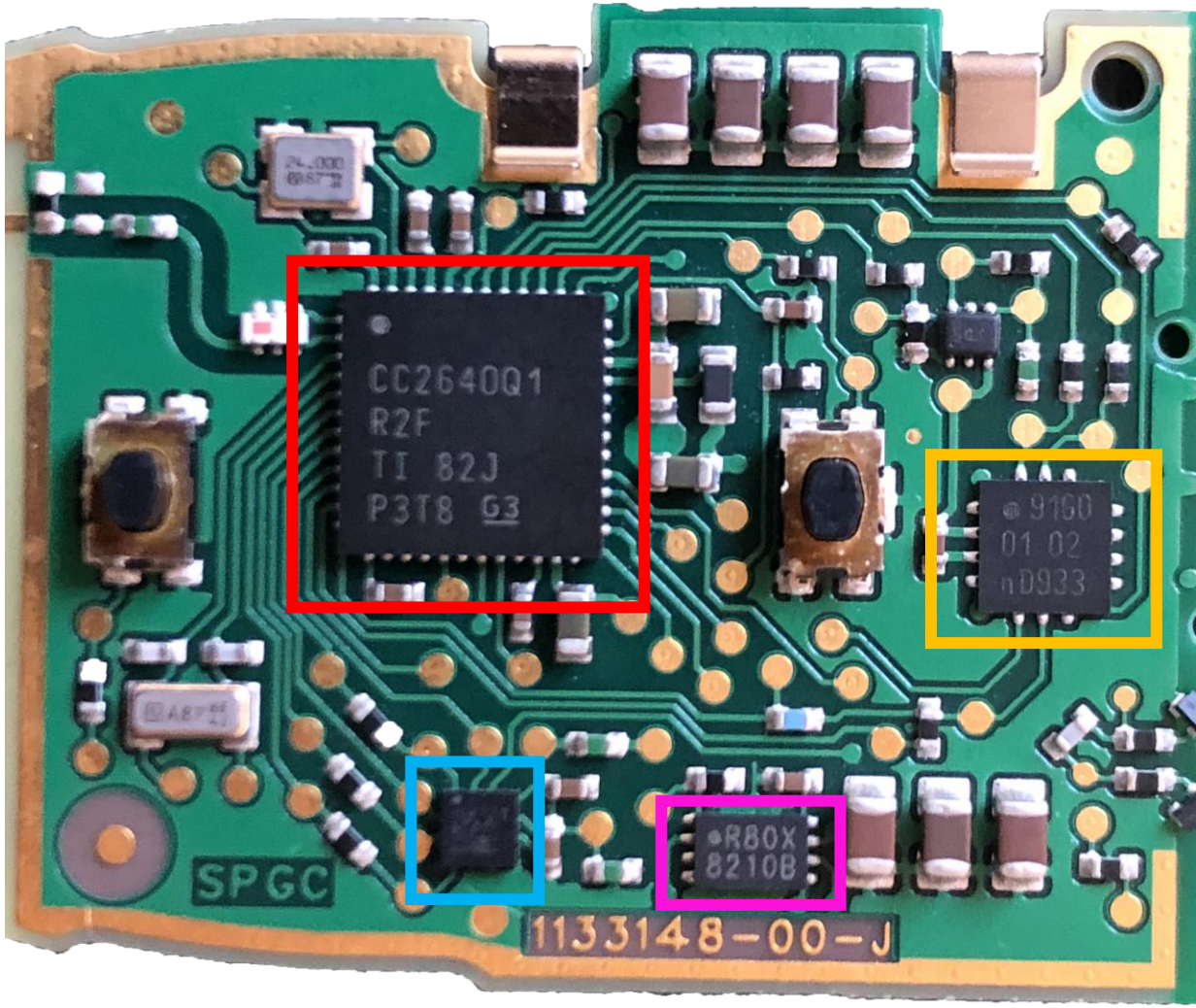
Lennert Wouters, Benedikt Gierlichs, Bart Preneel

COSADE 2022



Tesla Model 3 key fob

- Texas Instruments CC2640R2F-Q1
  - Debug interface is locked
- Flash storage
- NXP Secure Element
- Accelerometer

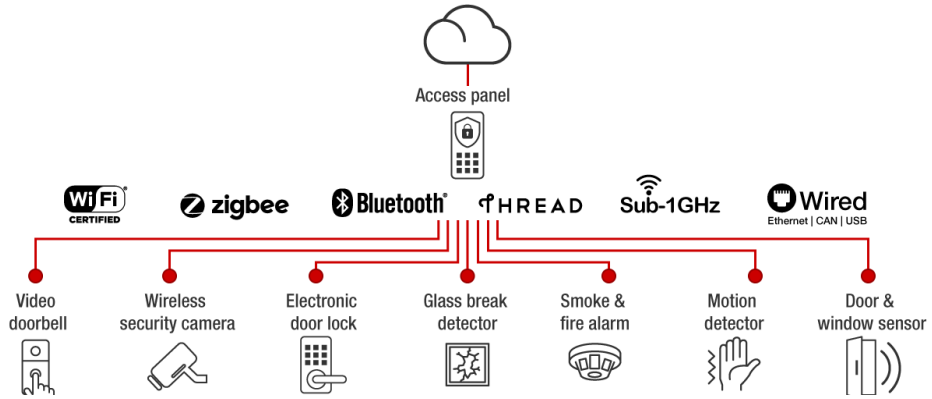


COSIC

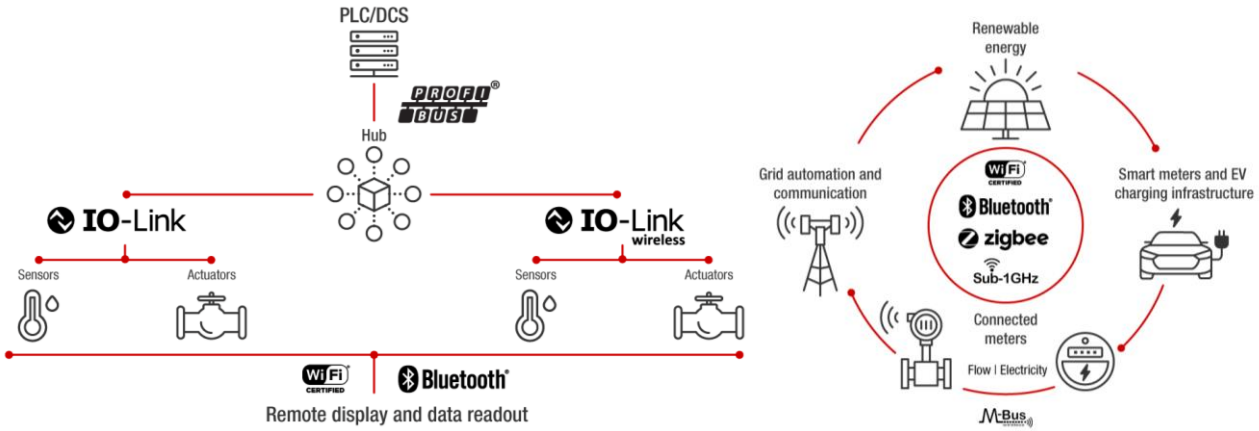
KU LEUVEN

The SimpleLink Platform

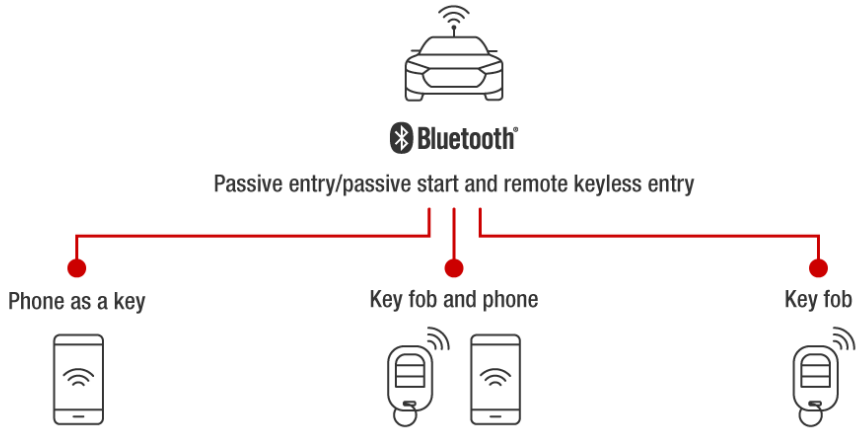
Building security



Critical infrastructure



Automotive



Medical

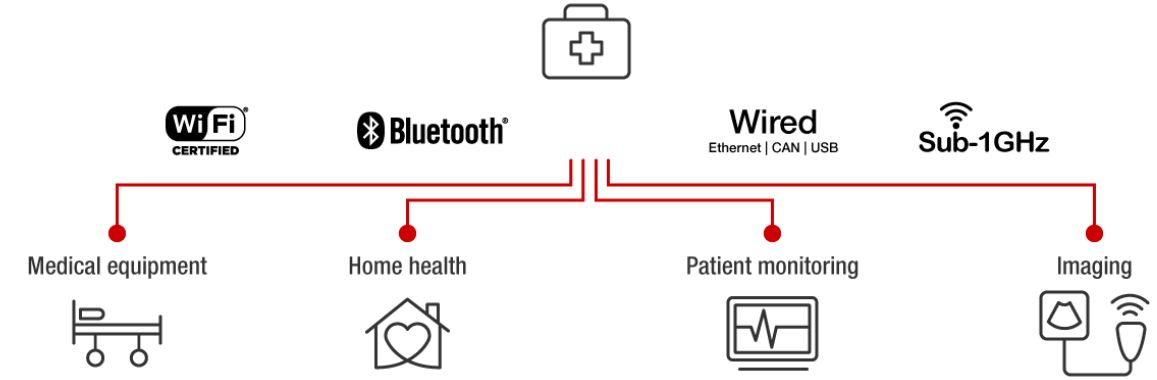


Image source: https://www.ti.com/wireless-connectivity/applications.html

What we hope for

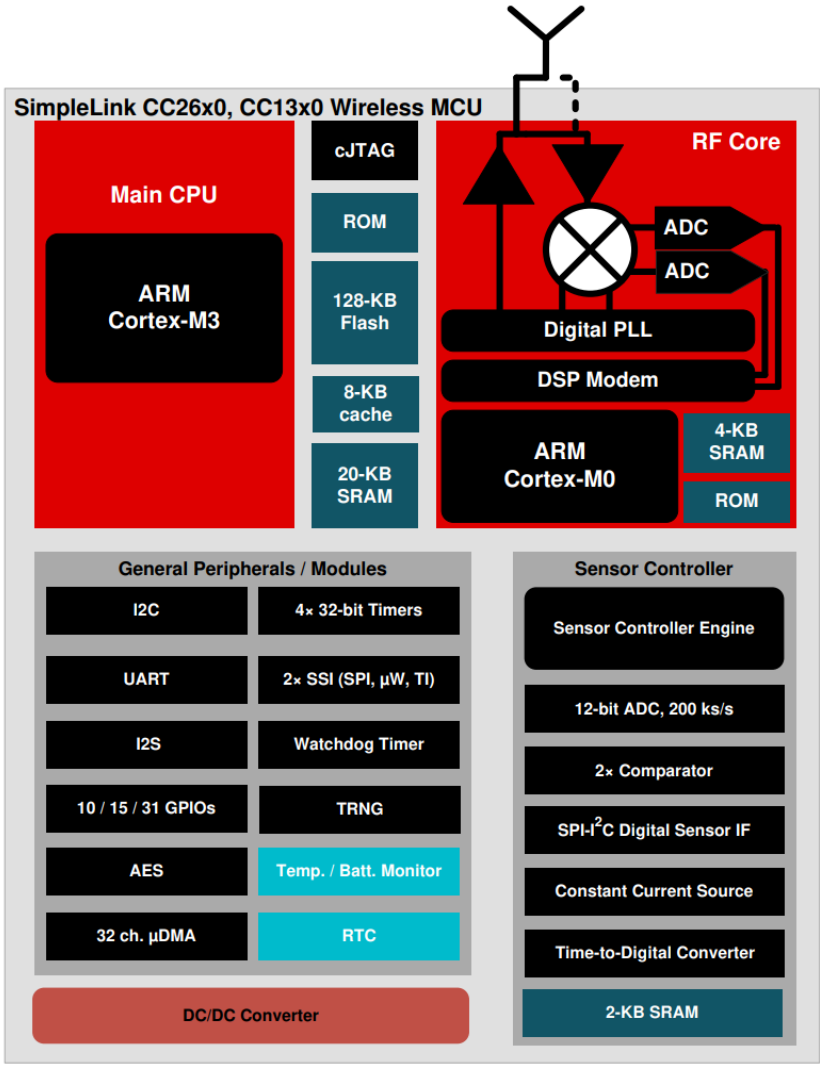
- Secure wireless stack (provided by Texas Instruments)
  - Secure implementations of secure protocols
- Secure application code (provided by developers)
- Hardware security features?
  - Secure boot and remote attestation
  - Secure key storage
  - TRNG
  - (Protected) hardware accelerators
  - Debug security
    - IP protection

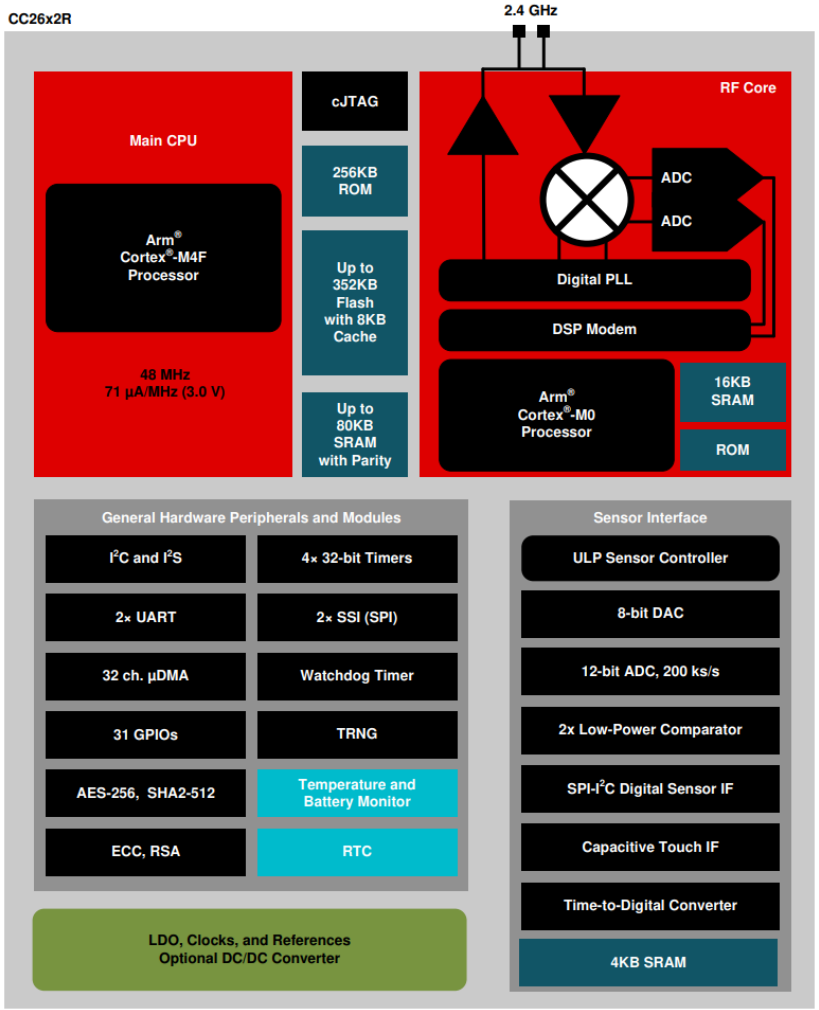
Debug security

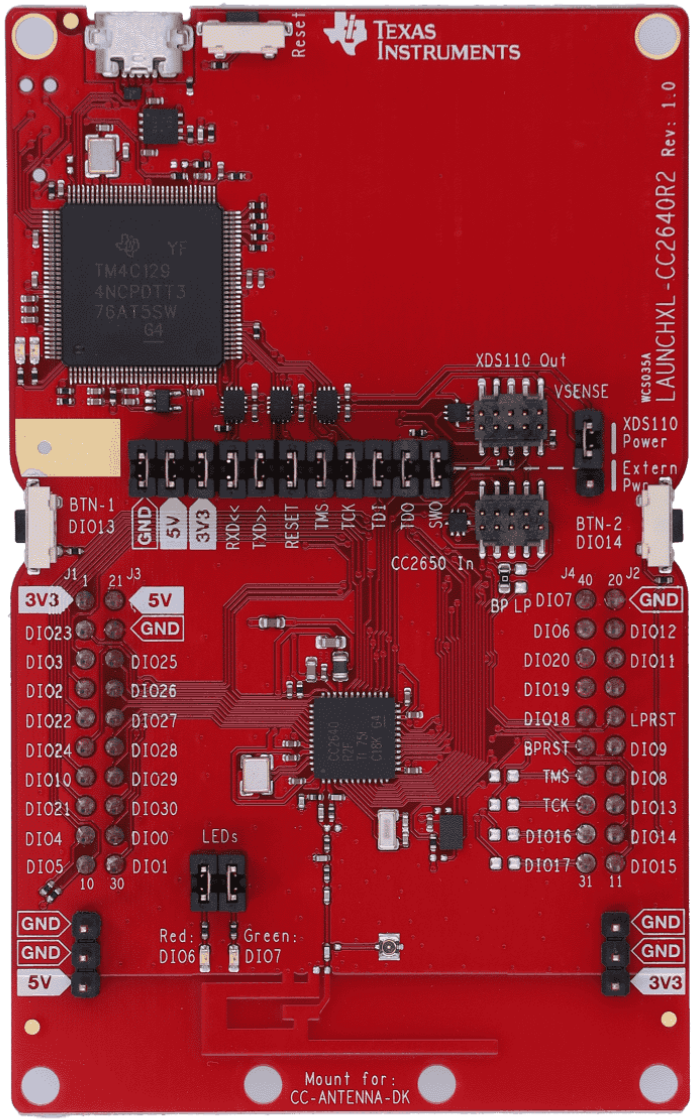
- Defeating debug security allows to:
  - Recover firmware (IP)
    - Counterfeit products
    - Evaluating the security of the application code
  - Defeat secure boot and security features that rely on it

- The presented attacks require physical access
  - Extracting information from one device can lead to attacks that scale
    - Weak key derivation or master keys
    - Proprietary crypto
    - Software vulnerabilities

CC13xx/CC26xx overview







COSIC

KU LEUVEN

The ROM bootloader

- The first piece of code that executes after reset
  - Immutable and mapped at 0x10000000
  - Responsible for initial setup and security configuration
    - Security settings are stored in Flash (CCFG)

- Extracting the ROM bootloader enables us to reverse engineer it
  - How is the debug interface disabled?

ROM bootloader: serial interface

- Allows to read and write memory
- Does not run when a valid firmware image is available
  - Can be activated using a ‘backdoor’
  - Or by erasing the microcontroller
- Implemented an interface in Python
  - Ask the ROM bootloader to read itself

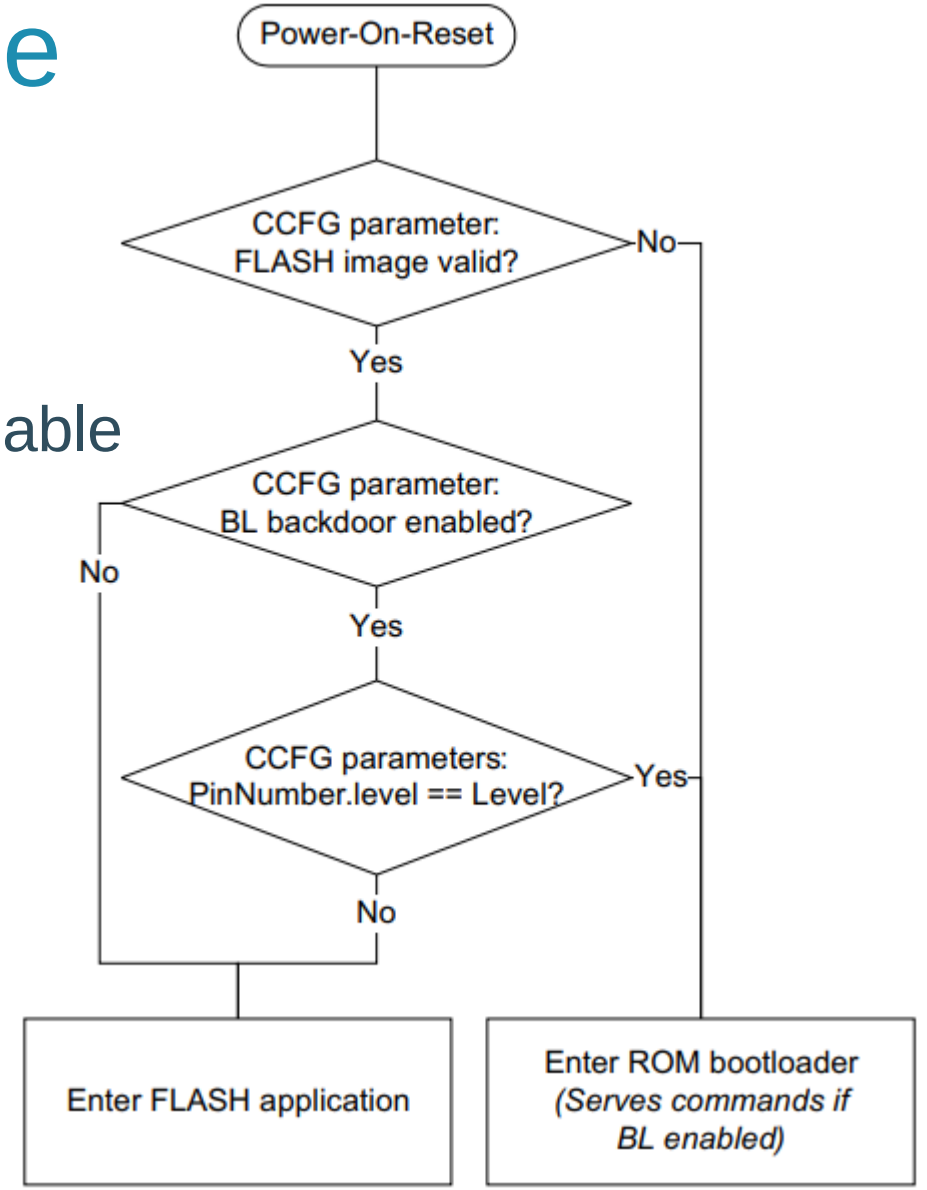


Image source: https://www.ti.com/lit/an/swra466d/swra466d.pdf

ROM bootloader: analysis (1)

- Static analysis in Ghidra

Decompile: FUN_1000103e - (rom_bootloader_cc2640r2.bin)

```
1
2  /* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
3
4  void FUN_1000103e(void)
5
6  {
7    uint uVar1;
8    undefined4 uVar2;
9    uint uVar3;
10   uint uVar4;
11   int iVar5;
12
13   iVar5 = read_volatile_4(Peripherals::FCFG1._712_4_);
14   write_volatile_4(DAT_40082250,(uint)(iVar5 << 0xe) >> 0x1e);
15   write_volatile_4(Peripherals::PRCM.CLKLOADCTL,1);
16   uVar3 = FUN_10000fee();
17   uVar4 = get_ccfg_jtag_config();
18   uVar1 = read_volatile_4(Peripherals::AON_WUC.JTAGCFG);
19   write_volatile_4(Peripherals::AON_WUC.JTAGCFG,uVar3 & uVar4 | uVar1);
20   read_volatile(Peripherals::AON_RTC.SYNC._0_1_);
21   uVar1 = read_volatile_4(DAT_40091090);
22   write_volatile_4(DAT_40091090,uVar1 | 1);
23   uVar1 = read_volatile_4(DAT_40091090);
24   write_volatile_4(DAT_40091090,uVar1 | 4);
25   uVar1 = read_volatile_4(DAT_4008224c);
26   write_volatile_4(DAT_4008224c,uVar1 | 1);
27   iVar5 = read_volatile_4(Peripherals::FCFG1._676_4_);
28   if (iVar5 == 0) {
29     FUN_10000e7c();
30     _DAT_e000ed08 = read_volatile_4(Peripherals::FCFG1._768_4_);
31     uVar2 = read_volatile_4(Peripherals::FCFG1._768_4_);
32     FUN_100003cc(uVar2);
33   }
```

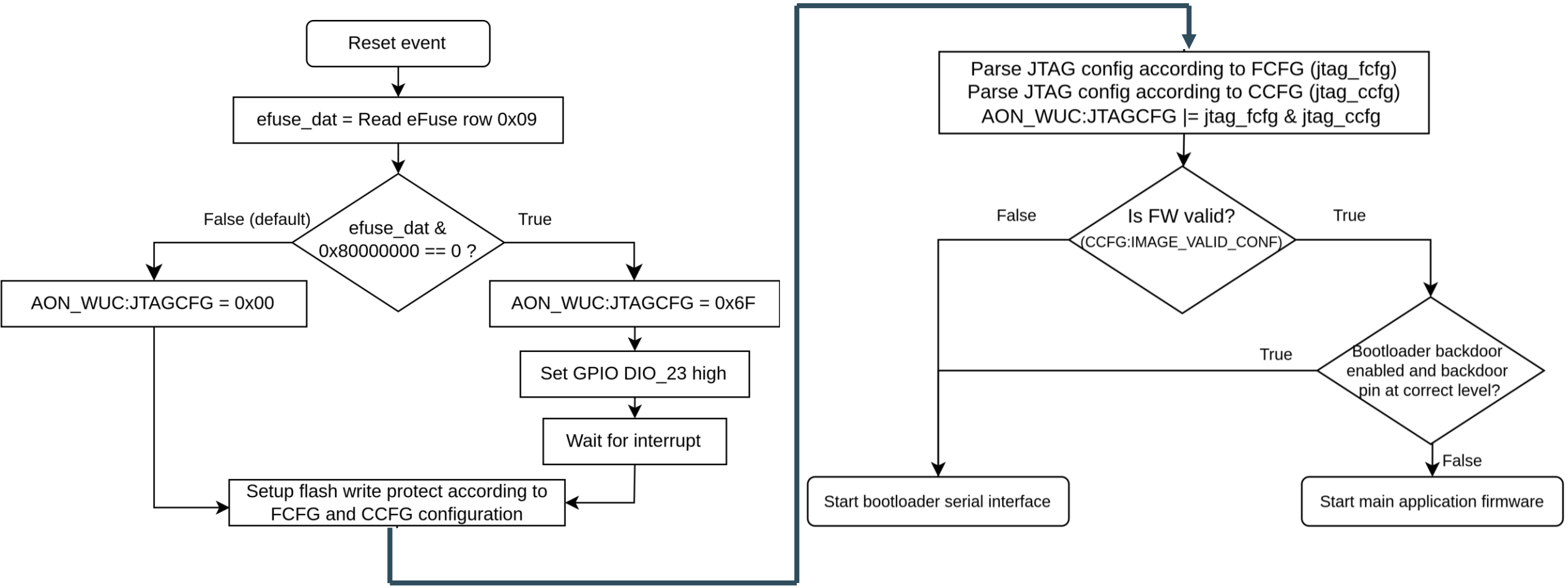
ROM bootloader: analysis (2)

- Emulation of the ROM bootloader using Unicorn
- Code coverage to augment Ghidra

- Other uses:
  - SCA and FI simulation
  - Fuzzing

```
  uVar6 = read_volatile_4(Peripherals::FLASH.FSEQPMP);
  write_volatile_4(Peripherals::FLASH.FSEQPMP,uVar6 & 0xf0ffffff | (uVar3 & 0x1e000000) >> 1);
  write_volatile_4(Peripherals::FLASH.EFUSEERROR,0);
  write_volatile_4(Peripherals::FLASH.EFUSEADDR,0);
  pFVar9 = (FCFGl *)0x4000009;
  write_volatile_4(Peripherals::FLASH.EFUSE,0x4000009);
  do {
    uVar3 = read_volatile_4(Peripherals::FLASH.EFUSEERROR);
  } while ((uVar3 & 0x20) == 0);
  uVar3 = read_volatile_4(Peripherals::FLASH.EFUSEERROR);
  if ((uVar3 & 0x1f) != 0) {
    FUN_10000498();
  }
  uVar3 = read_volatile_4(Peripherals::FLASH.DATALOWER);
  uVar6 = read_volatile_4(Peripherals::AON_SYSCTL.PWRCTL);
  if ((uVar6 & 2) == 0) {
    write_volatile_4(Peripherals::FLASH.CFG,
                     (uVar3 & 0x8000000) >> 0x1a | (uVar3 & 0x30000000) >> 0x16 |
                     (uVar3 & 0x40000000) >> 0x16 | 1);
    uVar6 = read_volatile_4(Peripherals::FLASH.FSEQPMP);
    uVar3 = uVar6 & 0xffff8fff | (uVar3 & 0x7000000) >> 0xc;
  }
  else {
    write_volatile_4(Peripherals::FLASH.CFG,
                     (uVar3 & 0x100000) >> 0x13 | (uVar3 & 0x600000) >> 0xf |
                     (uVar3 & 0x800000) >> 0xf | 1);
    uVar6 = read_volatile_4(Peripherals::FLASH.FSEQPMP);
    uVar3 = uVar6 & 0xffff8fff | (uVar3 & 0xe0000) >> 5;
  }
  write_volatile_4(Peripherals::FLASH.FSEQPMP,uVar3);
  write_volatile_4(Peripherals::FLASH.FLOCK,0x55aa);
  uVar3 = read_volatile_4(Peripherals::AON_WUC.MCUCLK);
  write_volatile_4(Peripherals::AON_WUC.MCUCLK,uVar3 | 4);
  write_volatile_4(Peripherals::FLASH.EFUSEERROR,0);
  write_volatile_4(Peripherals::FLASH.EFUSEADDR,0);
  write_volatile_4(Peripherals::FLASH.EFUSE,0x4000009);
  do {
    uVar3 = read_volatile_4(Peripherals::FLASH.EFUSEERROR);
  } while ((uVar3 & 0x20) == 0);
  uVar3 = read_volatile_4(Peripherals::FLASH.EFUSEERROR);
  if ((uVar3 & 0x1f) != 0) {
    FUN_10000498();
  }
  uVar3 = read_volatile_4(Peripherals::FLASH.DATALOWER);
  if ((uVar3 & 0x80000000) == 0) {
    FUN_10001612();
LAB_10001bd2:
    uVar3 = read_volatile_4(Peripherals::AON_WUC.JTAGCFG);
```

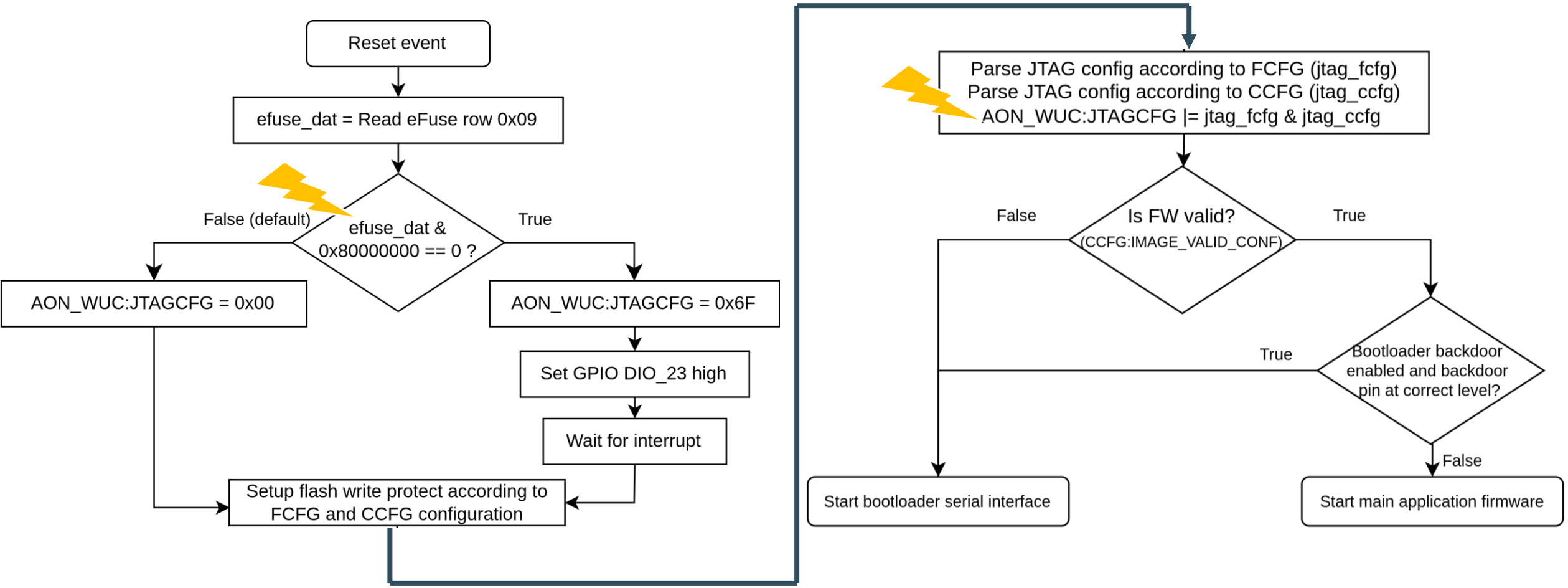
ROM bootloader: flowchart



COSIC

KU LEUVEN

ROM bootloader: fault injection

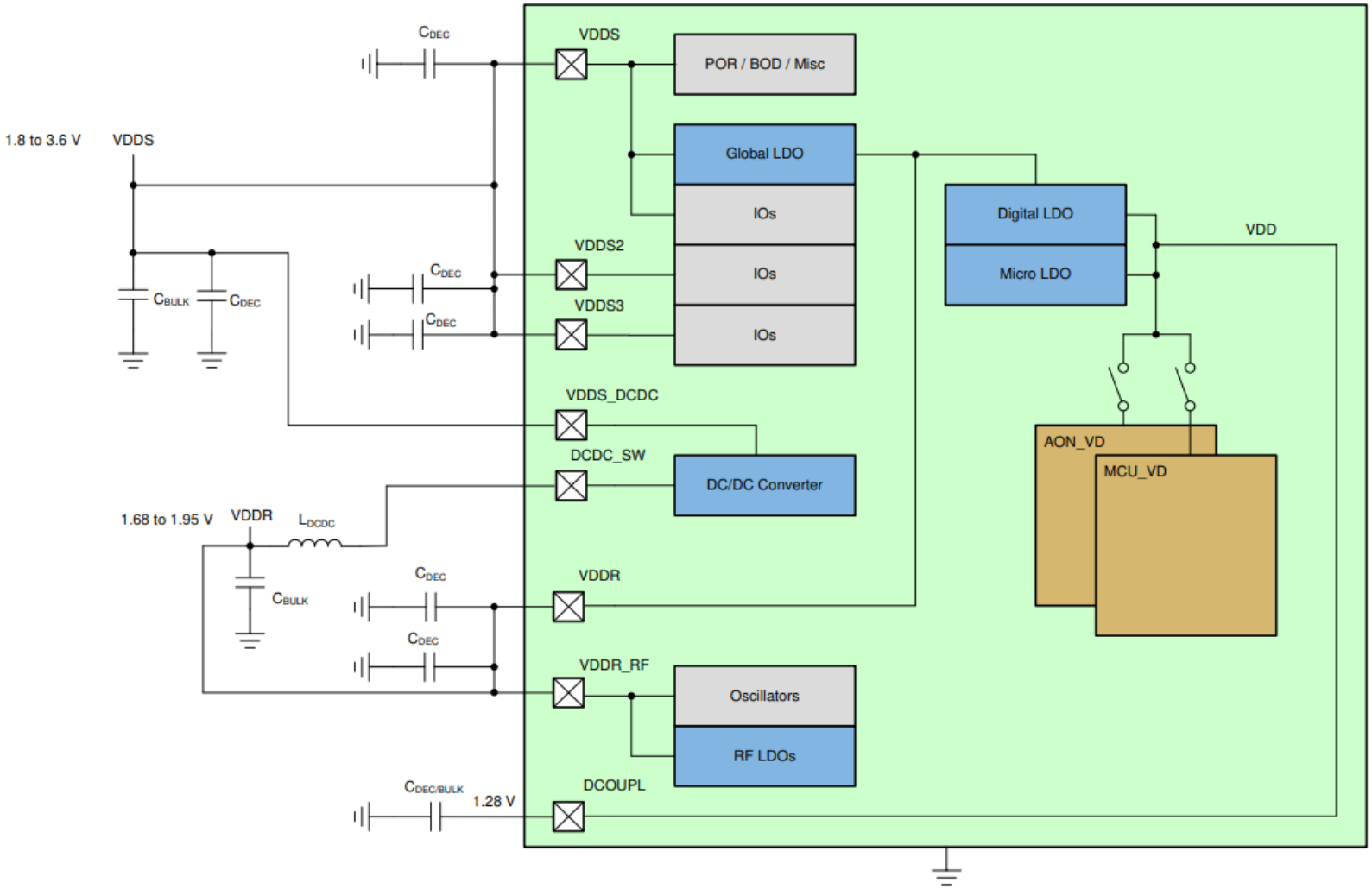


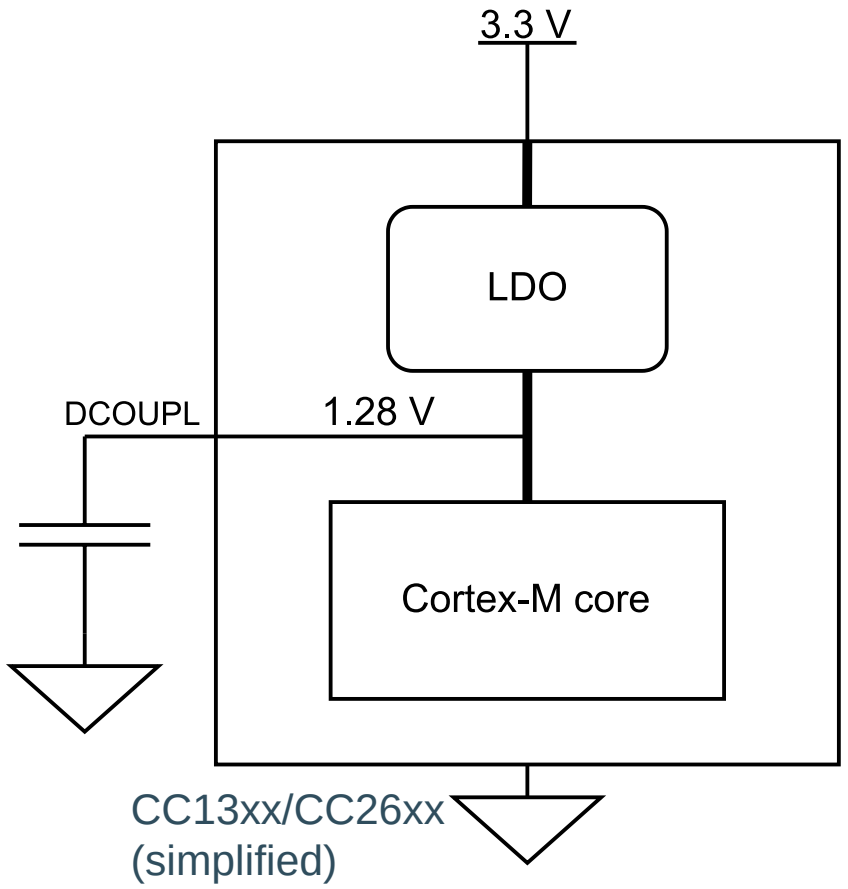
COSIC

KU LEUVEN

Microcontroller power supply

Image source: https://www.ti.com/lit/an/swra640f/swra640f.pdf

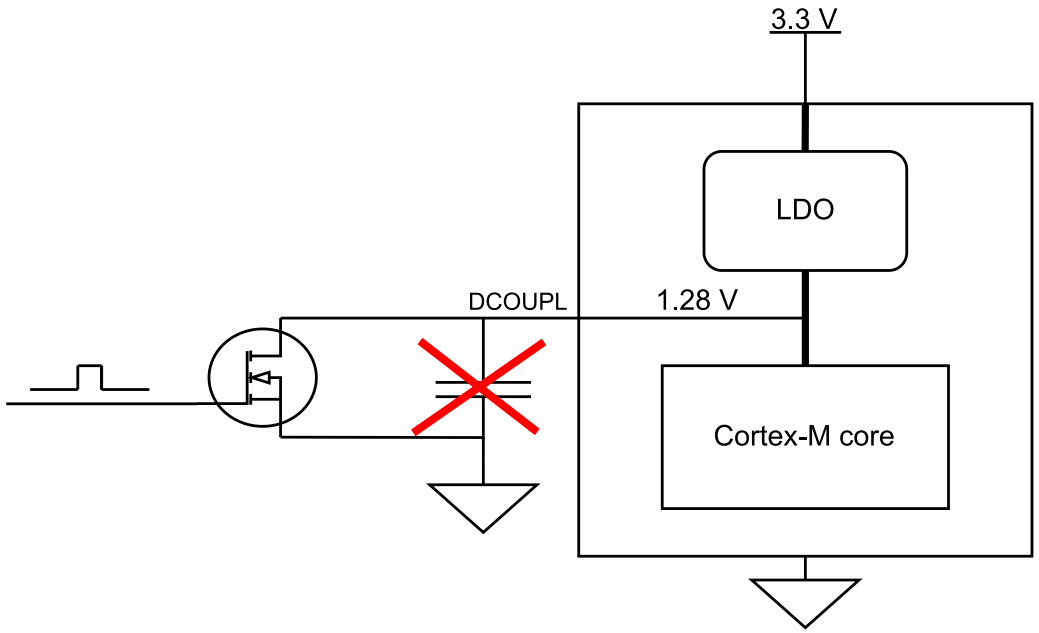




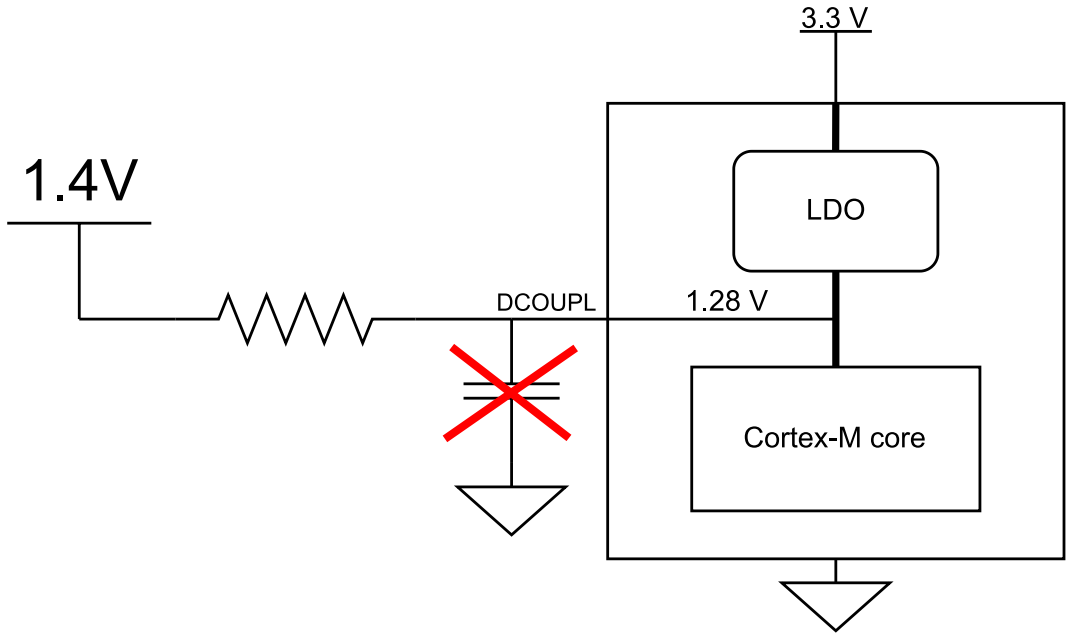
COSIC

KU LEUVEN

Crowbar voltage glitch



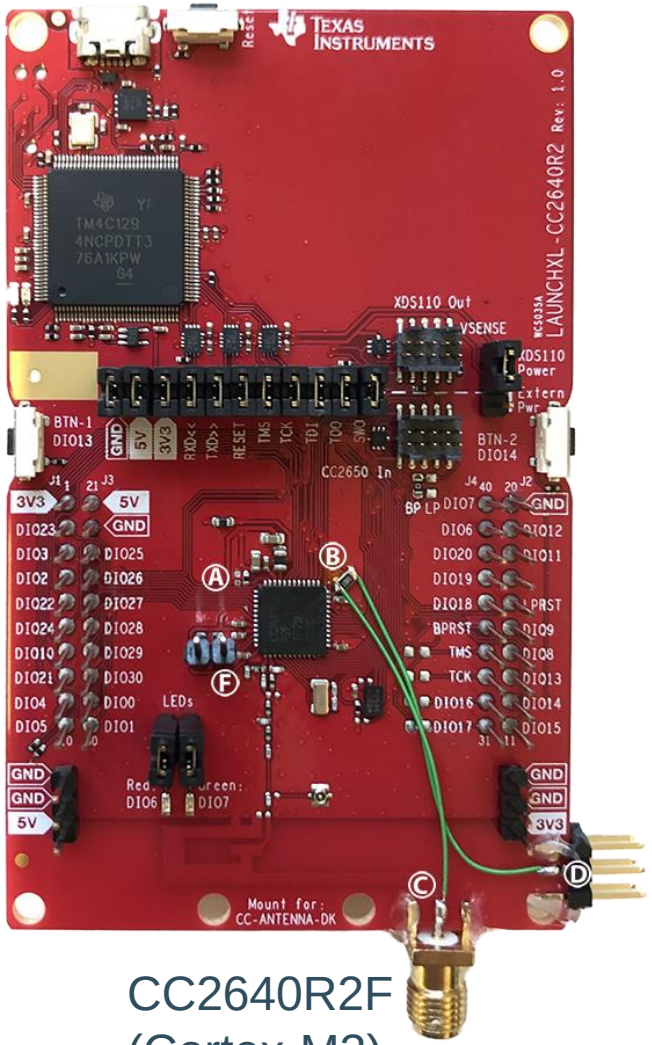
Side-channel analysis



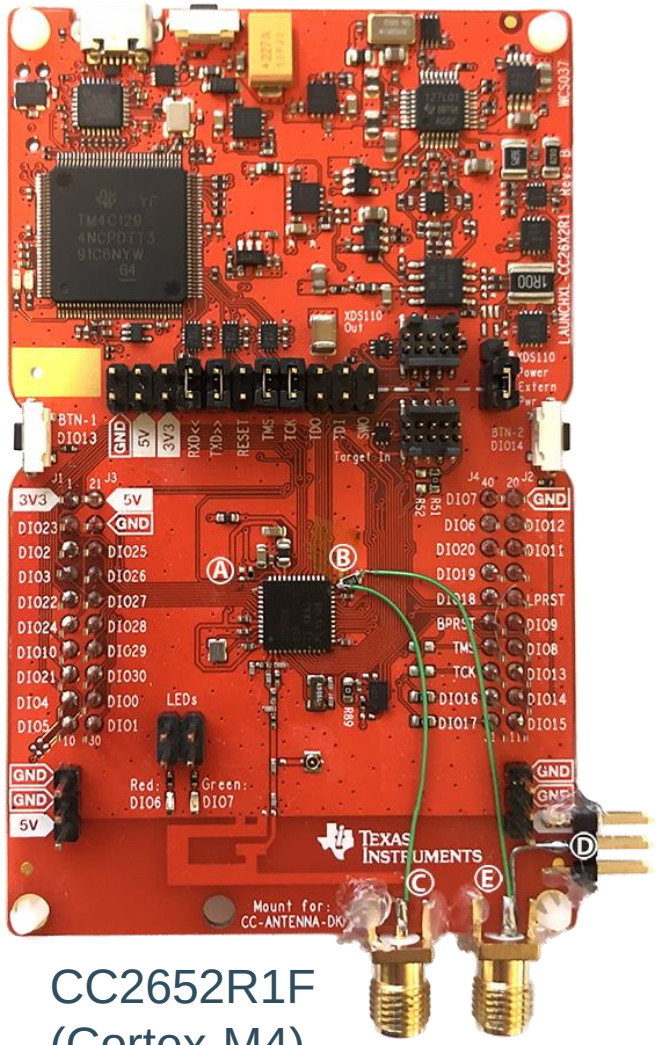
COSIC

KU LEUVEN

Targets



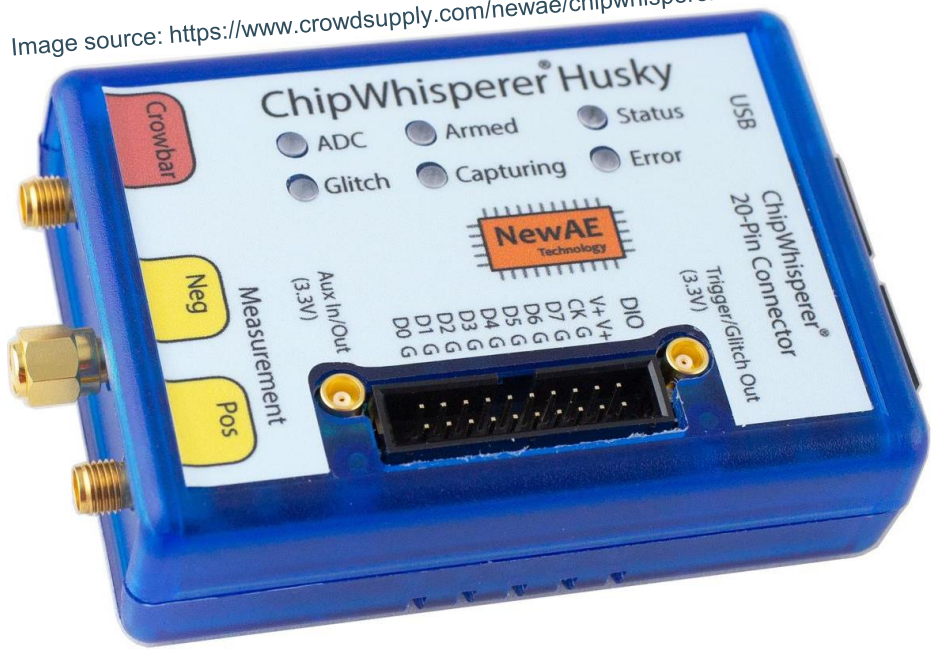
CC2640R2F
(Cortex-M3)



CC2652R1F
(Cortex-M4)

Attack platform

Image source: https://www.crowdsupply.com/newae/chipwhisperer-husky



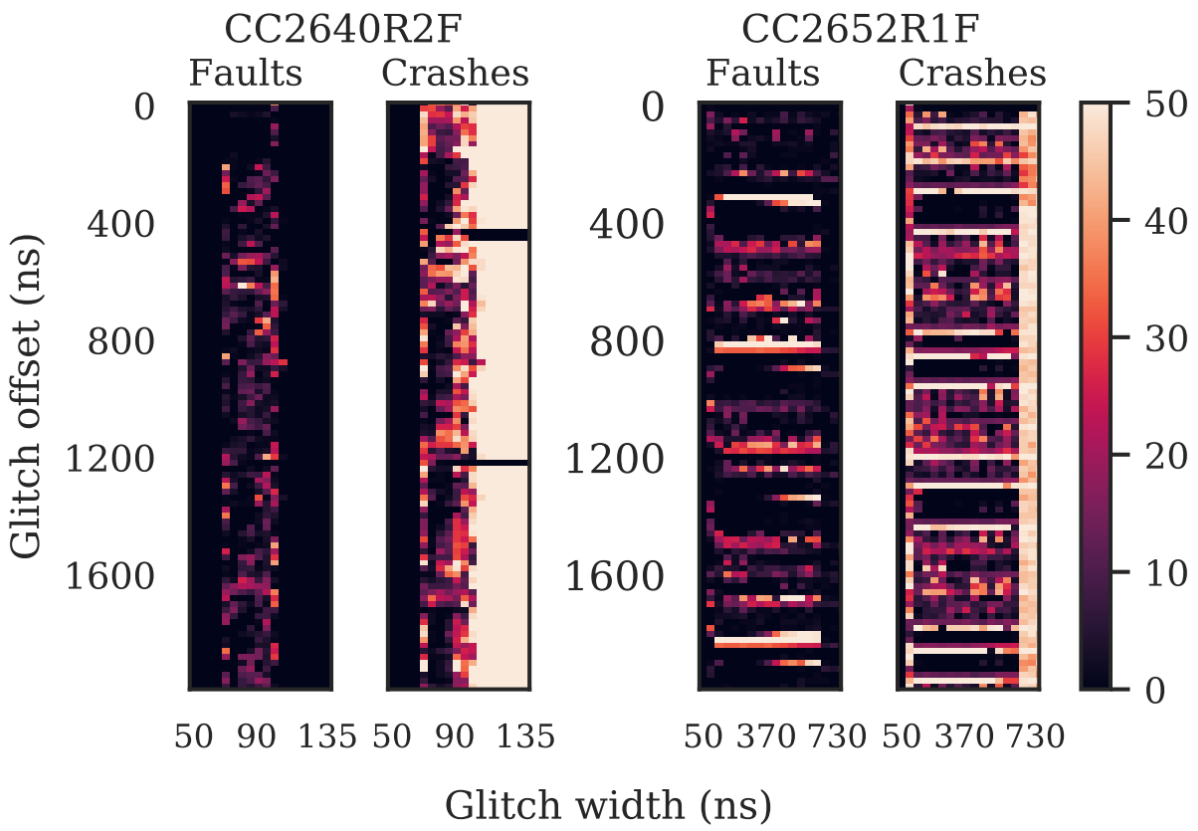
- 200 MSPS, 12-bit ADC
- Sample buffer: ~130k samples
  - Segmented memory feature
- Two crowbar MOSFETs
- High resolution glitch generation (sub-nanosecond resolution)

COSIC

KU LEUVEN

Determining a suitable glitch width

- Target: a dummy program



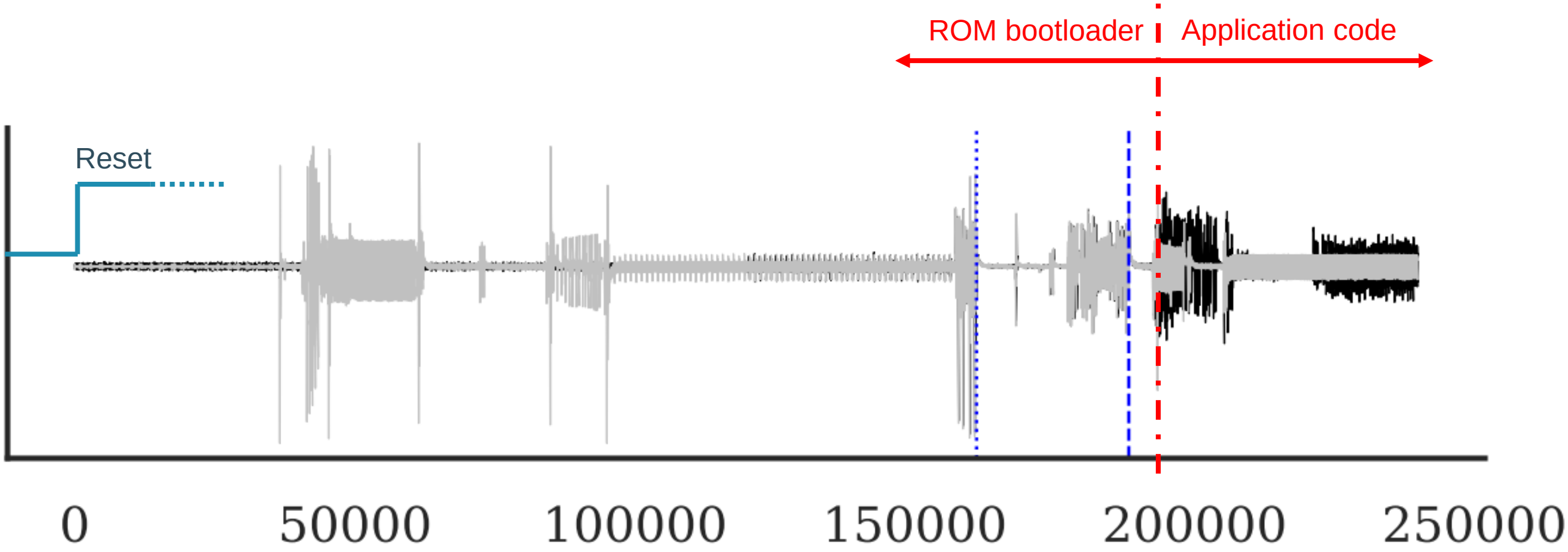

```c
void double_loop()
{
    char input;
    volatile int i, j, cnt;

    while (1)
    {
        UART_read(uart, &input, 1);
        if (input == 0xAA)
        {
            cnt = 0;
            GPIO_write(Board_GPIO_LED0, 1); // Set trigger high
            for (i = 0; i < 100; i++)
            {
                for (j = 0; j < 100; j++)
                {
                    cnt++;
                }
            }
            GPIO_write(Board_GPIO_LED0, 0); // Set trigger low
            UART_write(uart, &cnt, 4);
        }
    }
}
```


COSIC

KU LEUVEN

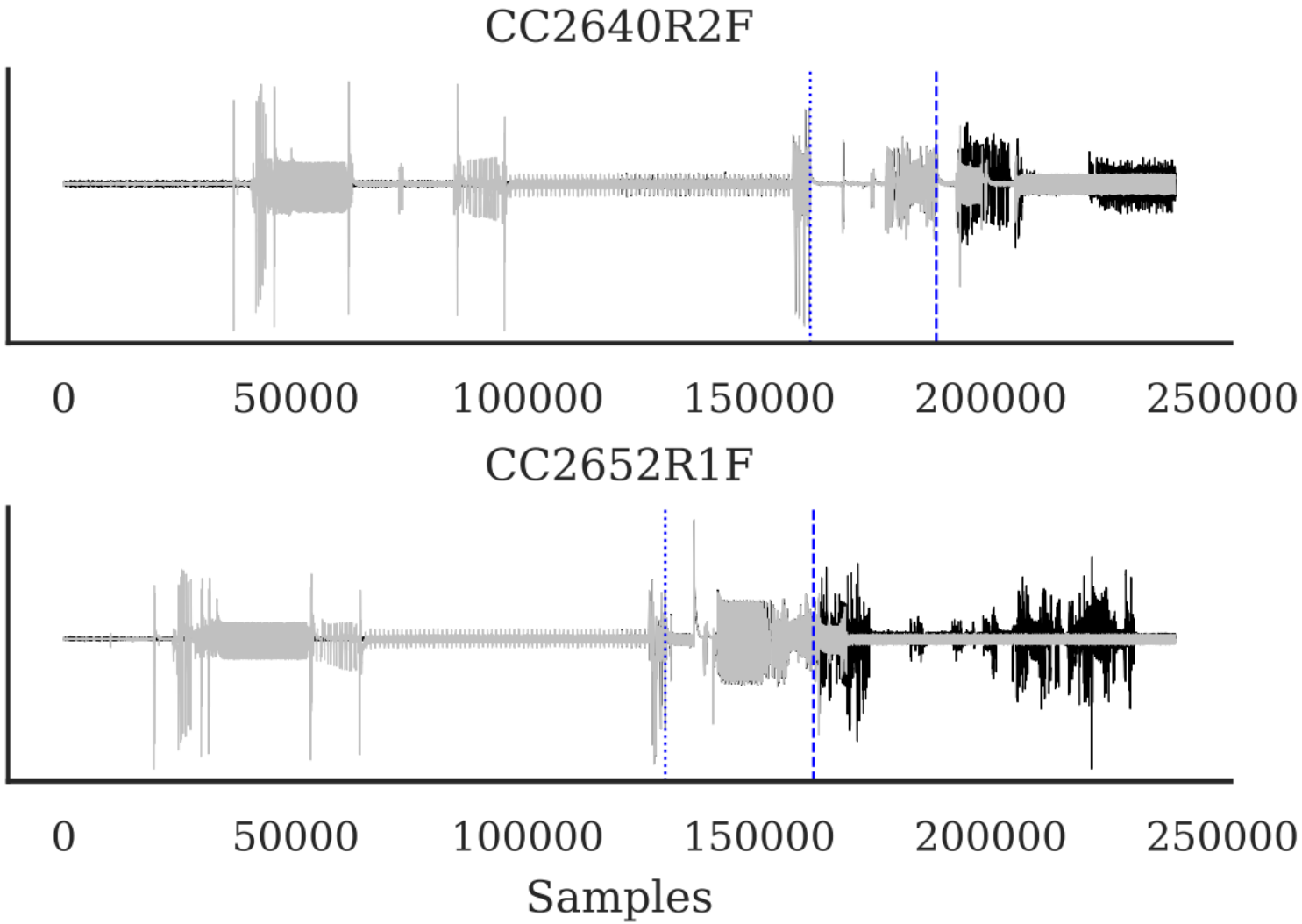
ROM bootloader: glitch offset



COSIC

KU LEUVEN

CC2640R2F VS CC2652R1F



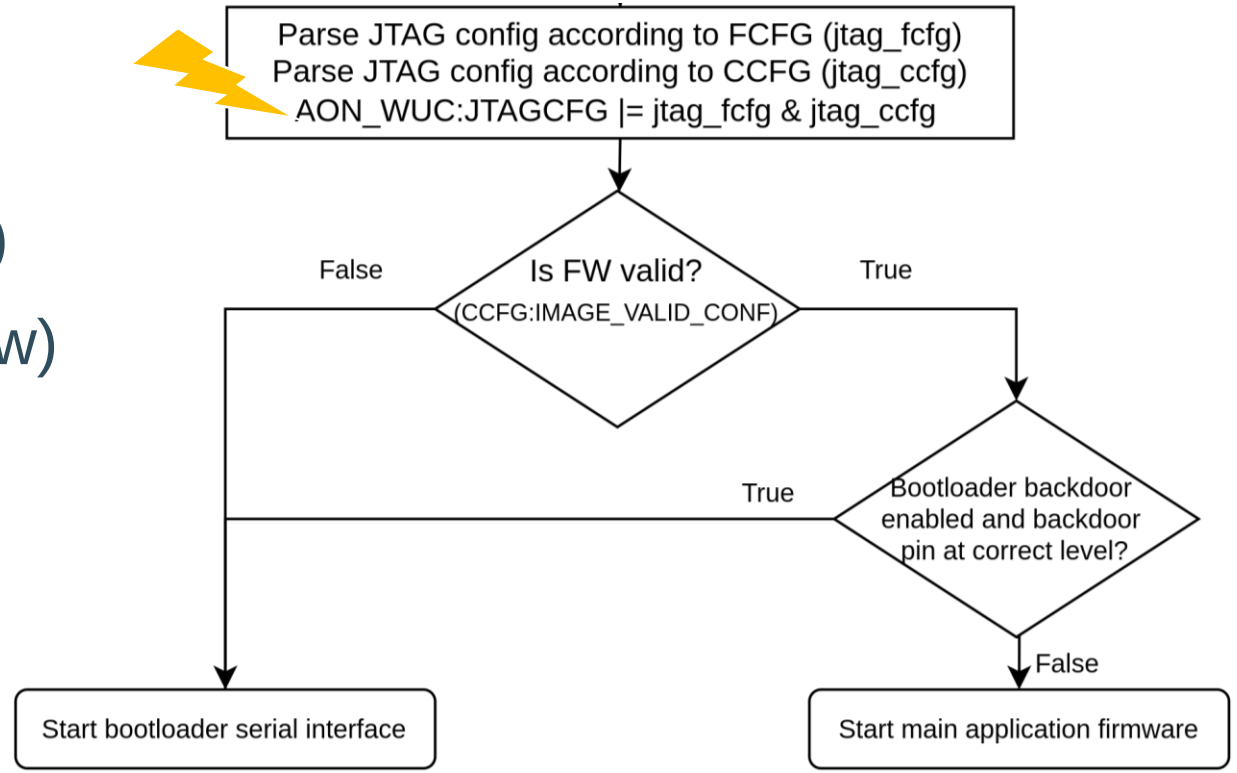
COSIC

KU LEUVEN

Glitching JTAG configuration

1. Reset the microcontroller
2. Wait (glitch offset)
3. Activate glitch MOSFET (glitch width)
4. Try to connect using a debugger (slow)

Step 4: during enumeration we can read AON_WUC:JTAGCFG

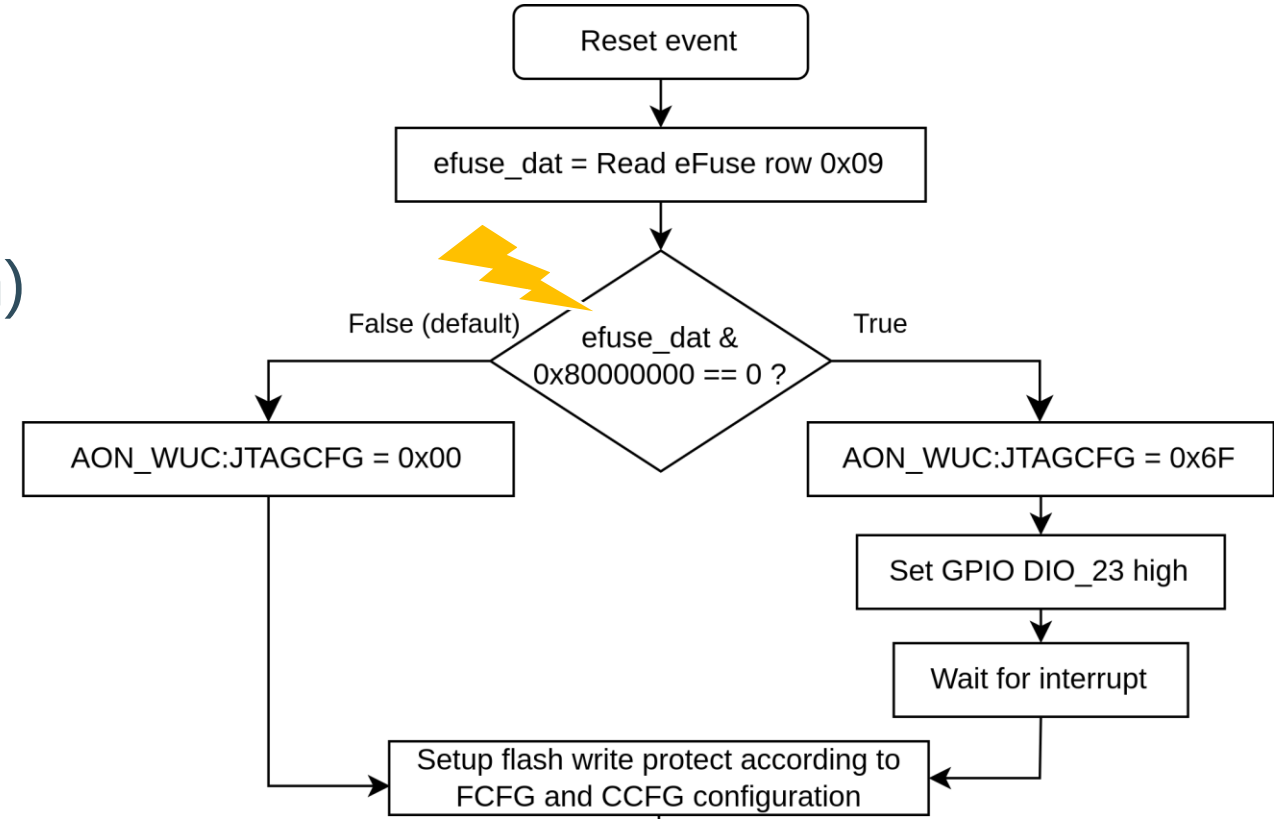


COSIC

KU LEUVEN

Glitching eFuse check

1. Reset the microcontroller
2. Wait (glitch offset)
3. Activate glitch MOSFET (glitch width)
4. Check if DIO_23 is high (fast)



COSIC KU LEUVEN

Debug security bypass results

Target: JTAG configuration

- 2,5 s per glitch attempt
  - (0,1 s during enumeration)

- CC2640R2F: ~5% success rate
- CC2652R1F: ~1% success rate

Target: eFuse readout/check

- 100 glitch attempts per second

- CC2640R2F: ~10% success rate
- CC2652R1F: ~0,1% success rate

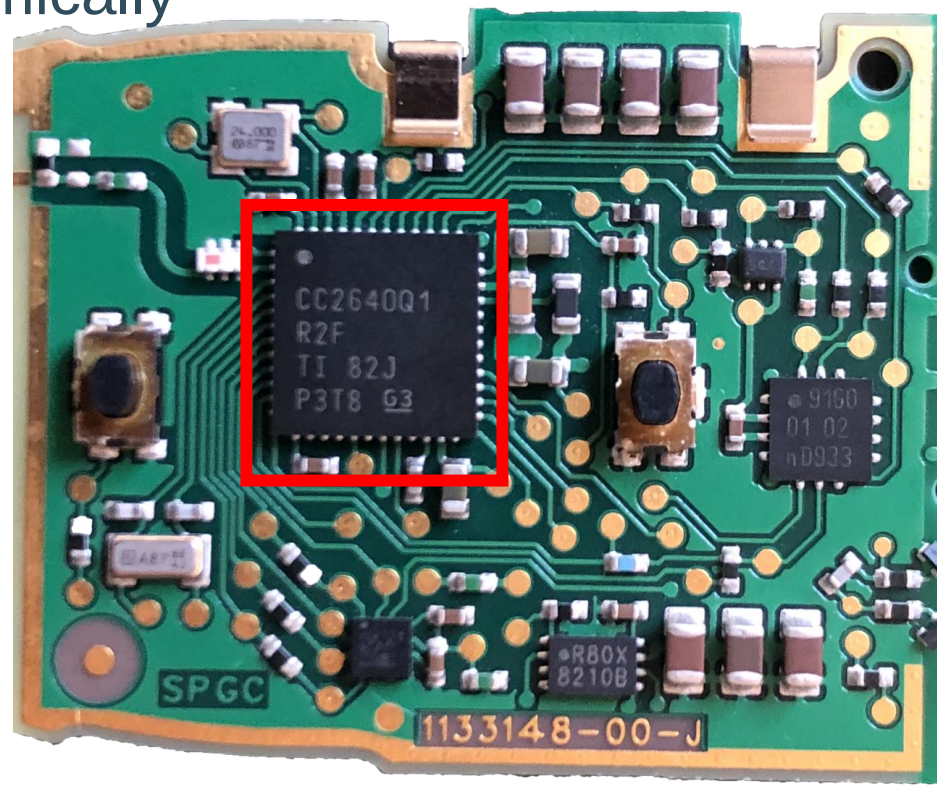
DEMO

https://www.forbes.com/sites/lanceeliot/2019/11/30/top-ten-reasons-teslas-cybertruck-windows-shattered-despite-being-unbreakable/

Verification on a real target

- Extracted the firmware from a Tesla Model 3 key fob
- Recovered an AES key from the firmware (firmware updates?)
- Software can now be analyzed statically and dynamically

```
0001be50: c90c f1cb 0cf4 4b14 f74e 19fa 521d fd58  ......K..N..R..X
0001be60: 2500 6131 0114 4202 184e 031c 5a04 2493  %.a1..B..N..Z.$.
0001be70: 0530 9320 0000 3200 0000 ff00 0000 0000  .0. ..2.........
0001be80: 0100 0200 0400 1000 00ff 0800 0400 0001  ................
0001be90: 0002 0004 0008 0000 2b02 0000 0000 0054  ........+......T
0001bea0: 6573 6c61 204b 6579 666f 6200 0000 0000  esla Keyfob.....
0001beb0: 0000 0000 4cbd 0100 0100 0000 0105 7703  ....L.........w.
0001bec0: 0300 0000 340f 0020 100f 0020 3a01 0000  ....4.. ... :...
0001bed0: 0000 0000 0000 0000 380f 0020 c80e 0020  ........8.. ...
0001bee0: 5465 736c 6120 4b65 7966 6f62 f8b5 0100  Tesla Keyfob....
0001bef0: 0200 0000 8dc3 0100 0100 0000 8388 0100  ................
0001bf00: 5d66 0100 697b 0100 5bc6 0000 b5e4 0000  ]f..i{..[.......
0001bf10: 093c 0100 e90f 0100 4570 0100 3974 0100  .<......Ep..9t..
0001bf20: 0100 1100 0000 0000 0400 0000 0000 0000  ................
```

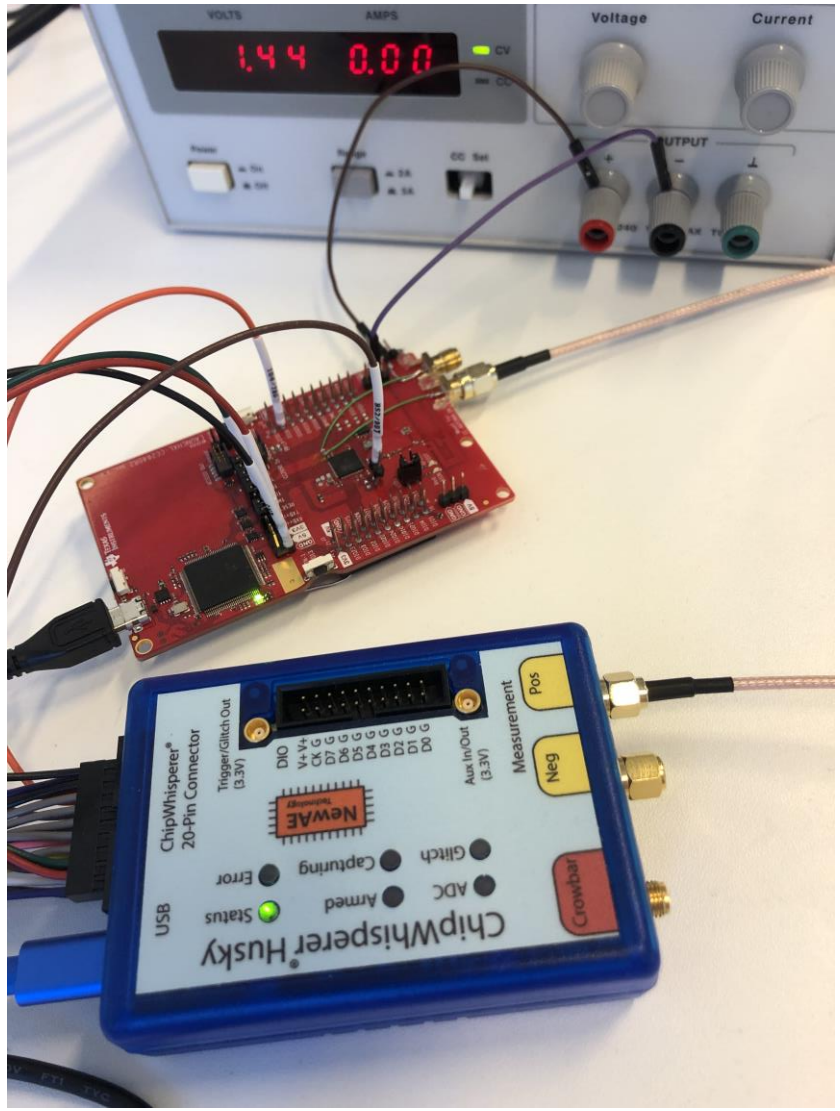


COSIC KU LEUVEN

The hardware AES co-processor

- One AES operation takes $2 + 3 \times r$ clock cycles
  - Or 32 clock cycles for one AES-128 operation
- The implementation operates on the full AES state

- Side-channel analysis
- Differential fault analysis



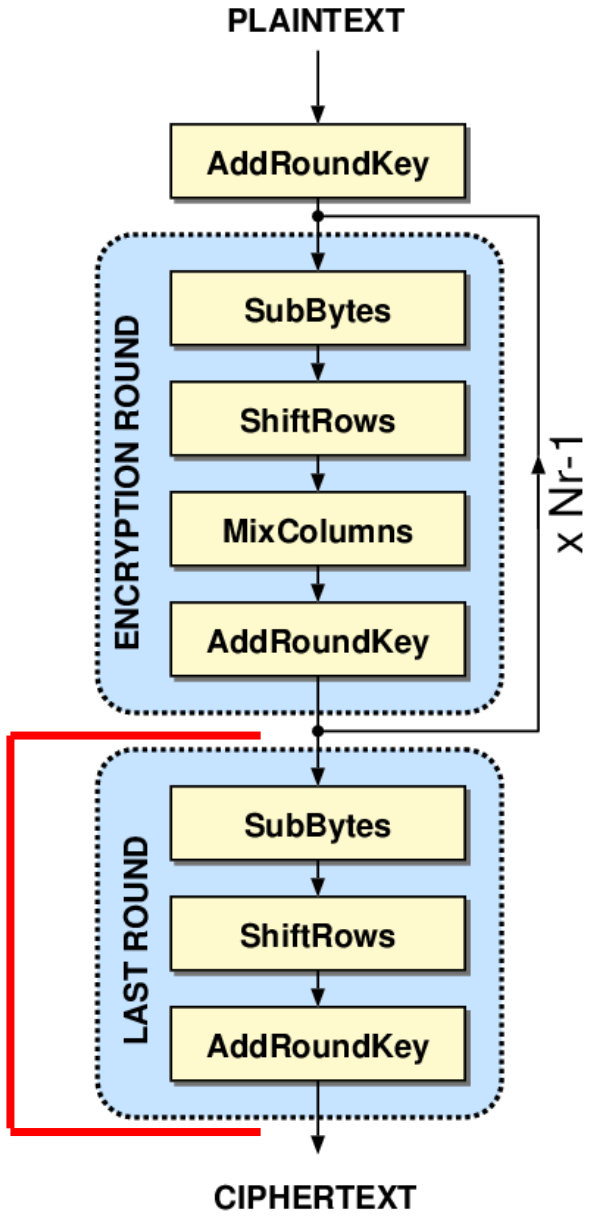
COSIC

KU LEUVEN

HW AES: side-channel analysis

- Determine a suitable leakage model
  - 100k traces with known key and plaintext
  - Compute all intermediate states
  - Perform CPA with all intermediates (HW)
  - And all combinations of intermediates (HD)

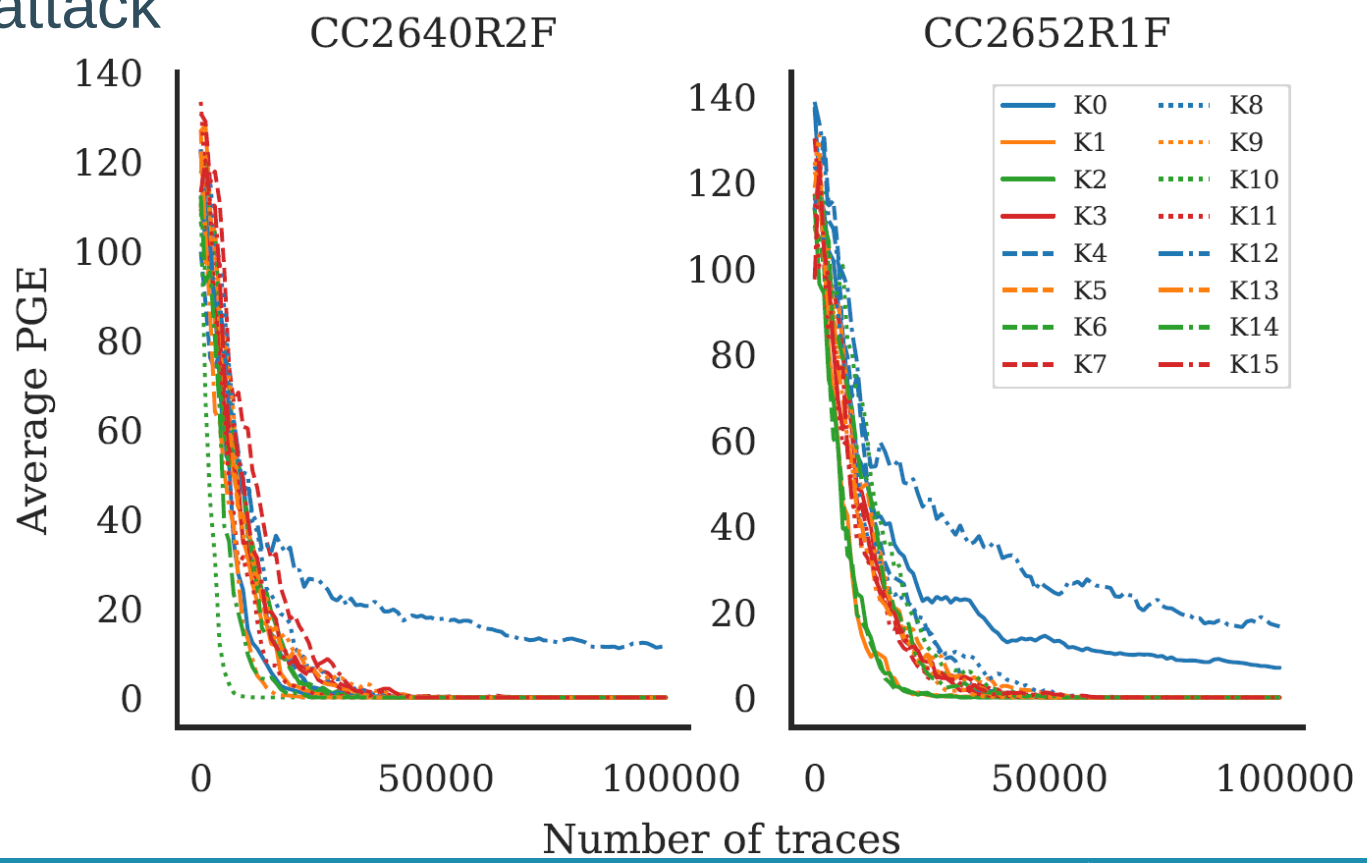
- HW leakage of plaintext and ciphertext
- SubBytes ⊕ ShiftRows
- r AddRoundKey ⊕ r + 1 AddRoundKey
- r9 AddRoundKey out ⊕ Ciphertext



HW AES: Attack results

- In total 100 (x 16) attacks, 100k traces per attack
- 1.5 minutes to acquire 100k traces using segmented memory
- Traces a preprocessed before the attack

- CC2640 @ 24 MHz
  - 12 MHz supplied by CW
  - Synchronous sampling
- CC2652 @ 48 MHz
  - Asynchronous sampling



COSIC

KU LEUVEN

HW AES: Differential Fault Analysis

- Inject a single byte fault before MixColumns in round 9
  - Results in 4 fault ciphertext bytes
  - One valid ciphertext and two such faults for each column allow to recover the key

- Faults injected using the ChipWhisperer
- Key recovery using Jean Grey - PhoenixAES (and Hulk)
  - https://github.com/SideChannelMarvels

Texas Instruments response

- Vulnerability was confirmed by Texas Instruments
  - Cannot be resolved without a new hardware revision
- Physical attacks are considered out of scope for this product
  - (and any other product for which physical security is not advertised)
- TI PSIRT was easy to reach and responsive

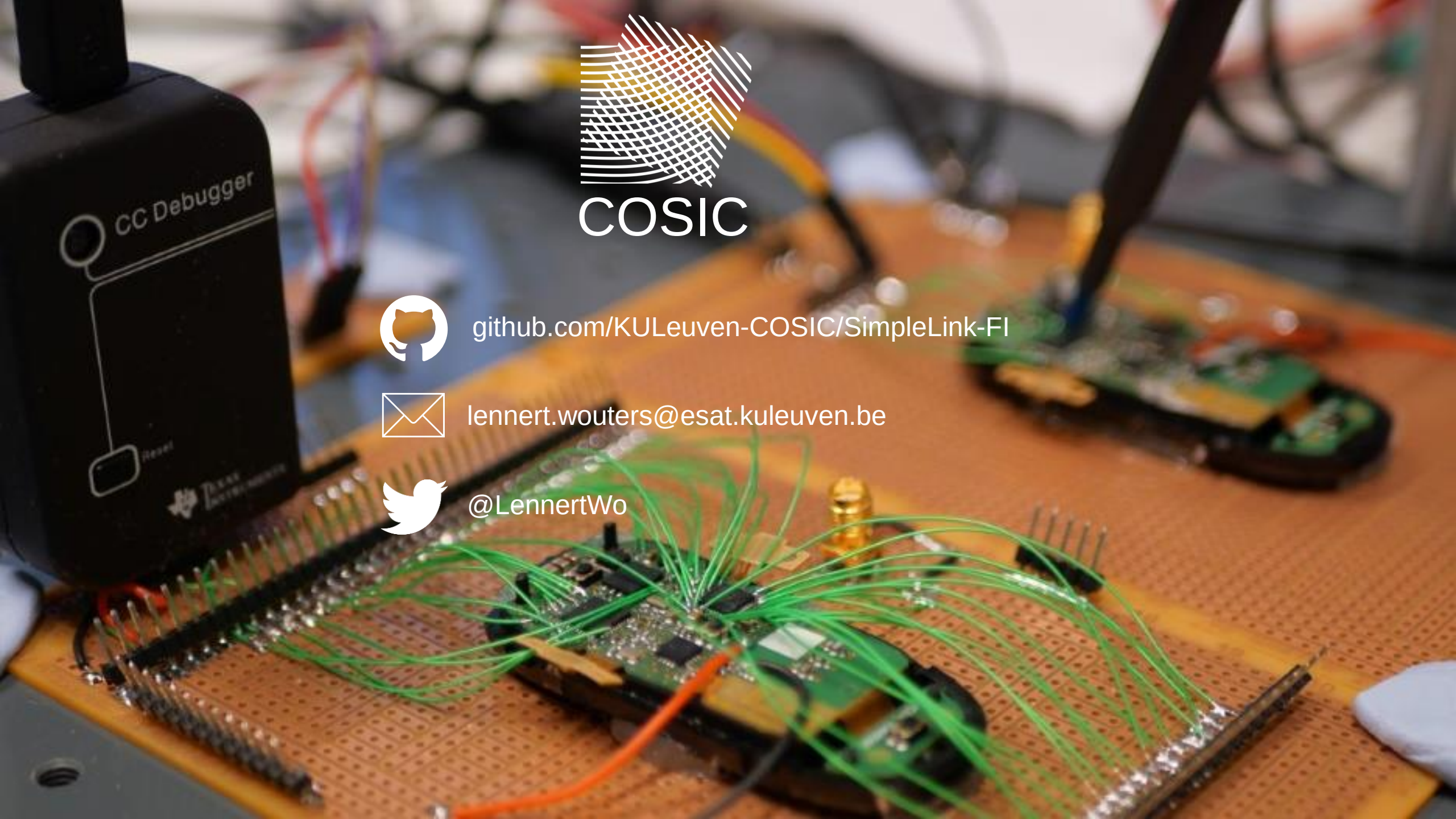
https://www.ti.com/lit/an/swra739/swra739.pdf

Conclusion

- Debug security can be easily compromised on CC13xx/CC26xx microcontrollers
  - Using basic non-invasive physical attacks

- Most general-purpose microcontrollers are vulnerable to similar attacks
  - This type of attack has been known for >20 years!
  - Assume that an attacker will be able to extract the firmware

- Outdated attacker models
  - A physical attacker is allowed to attach a debugger
  - A physical attacker is not allowed to mount physical attacks


CC Debugger
COSIC
github.com/KULeuven-COSIC/SimpleLink-FI
lennert.wouters@esat.kuleuven.be
@LennertWo