Reversing a M32C Firmware

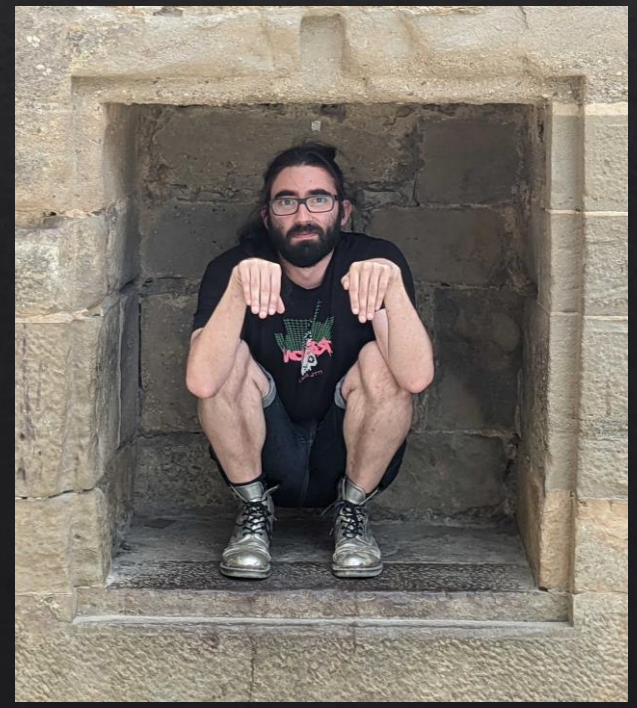
Lessons Learned while playing with a weird architecture

Philippe Laulheret (@phLaul)

Senior Security Researcher @ Trellix

Who am I?

- Philippe Laulheret — Senior Security Researcher @ Trellix
  - 3.5 years @ Advanced Threat Research
  - Work on Vulnerability Research in Consumer/IoT/Hardware/Enterprise/…
  - Blogs, talks, …
- Find me on Twitter
  - @phLaul

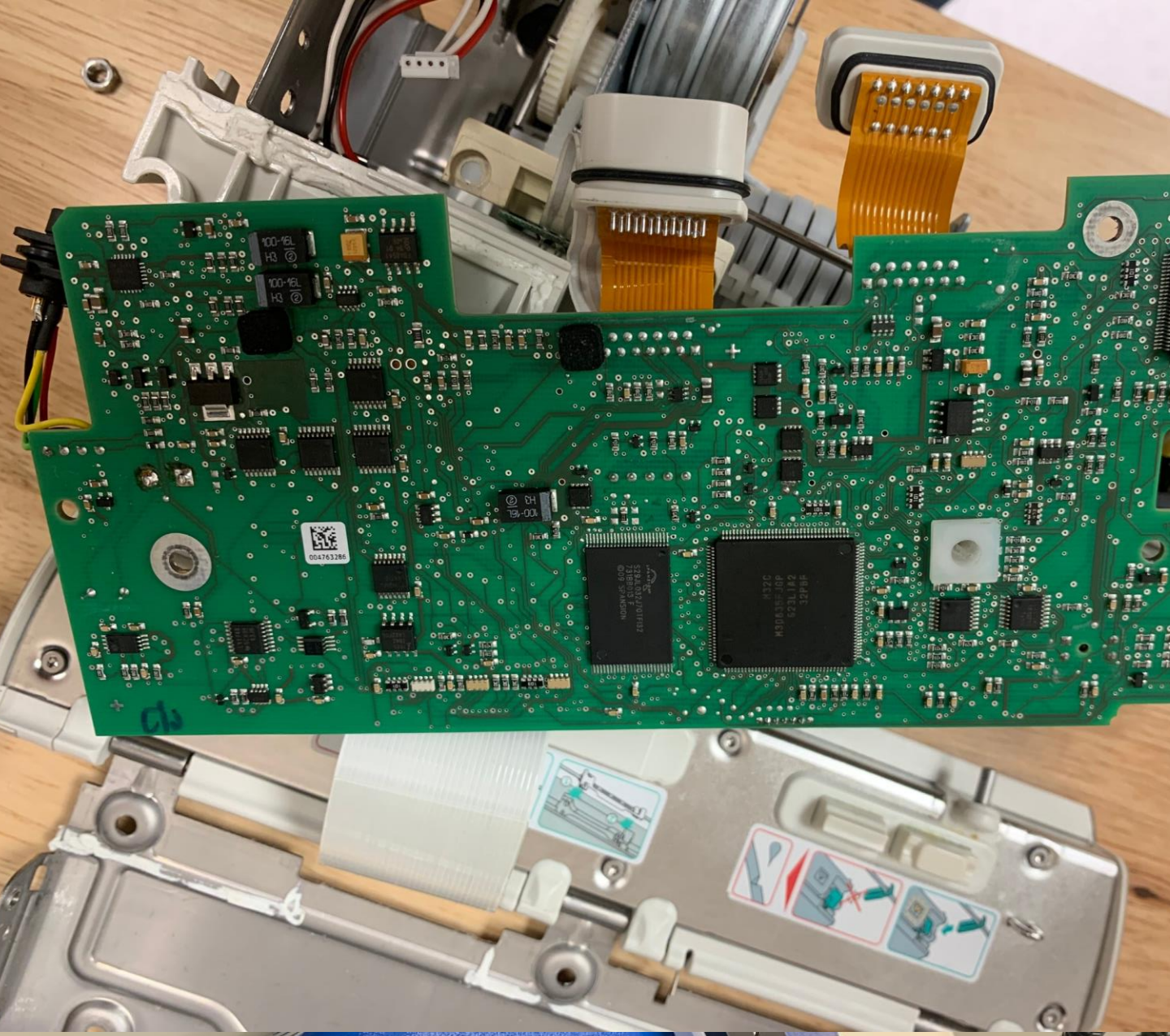


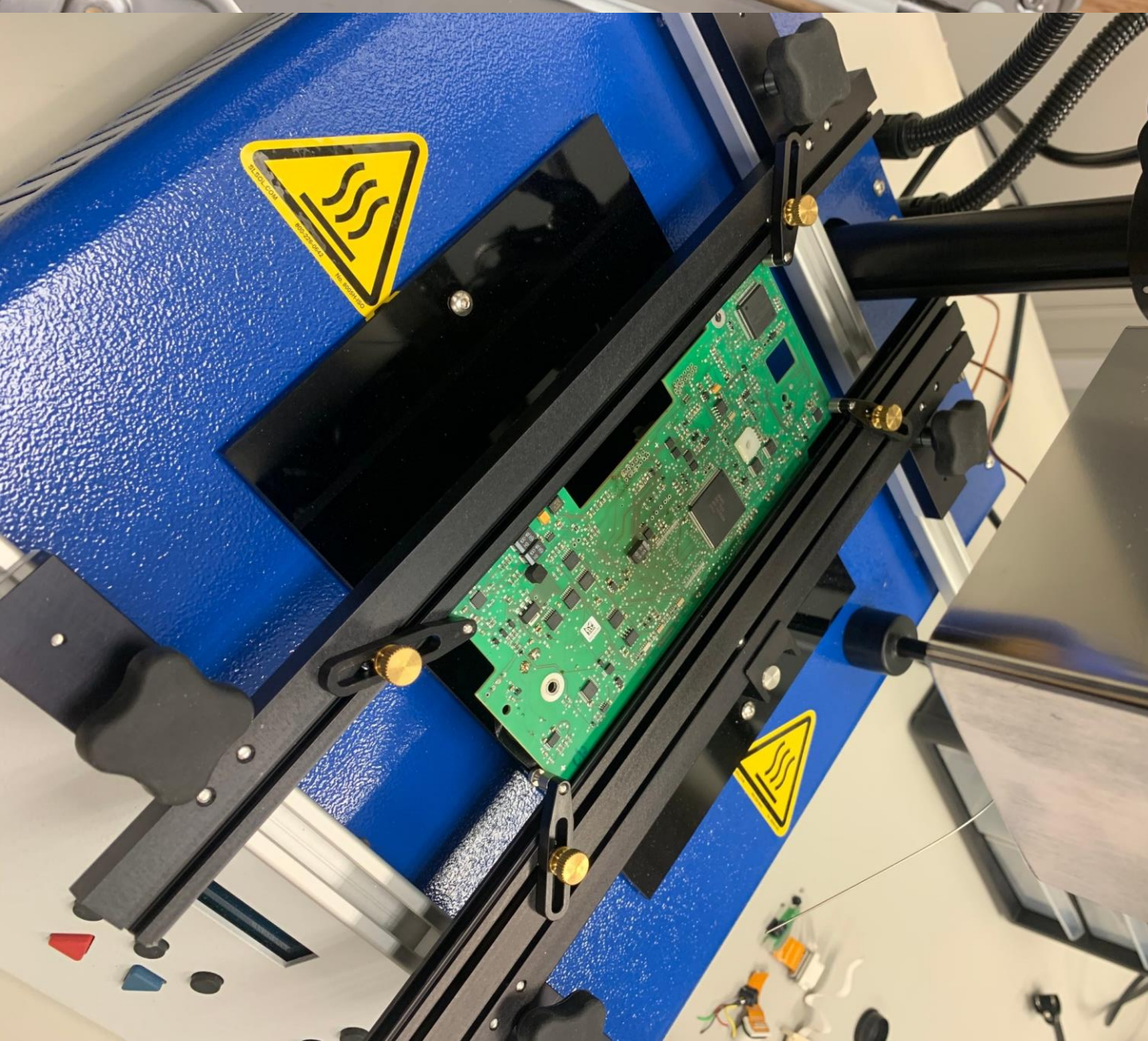
What is this talk about?

- ~1-year-long research involving infusion pump using M32C microcontroller
  - See: https://www.trellix.com/en-us/about/newsroom/stories/threat-labs/mcafee-enterprise-atr-uncovers-vulnerabilities-in-globally-used-b-braun-infusion-pump.html
  - Collaboration with Douglas McKee (Dir. Vulnerability Research & Principal Engineer @ Trellix)
- Lesson learned with this embedded controller:
  - How to start/what to do when reversing an unfamiliar architecture
  - This controller had lots of documentation; but what can we learn and re-use when no documentation is available?

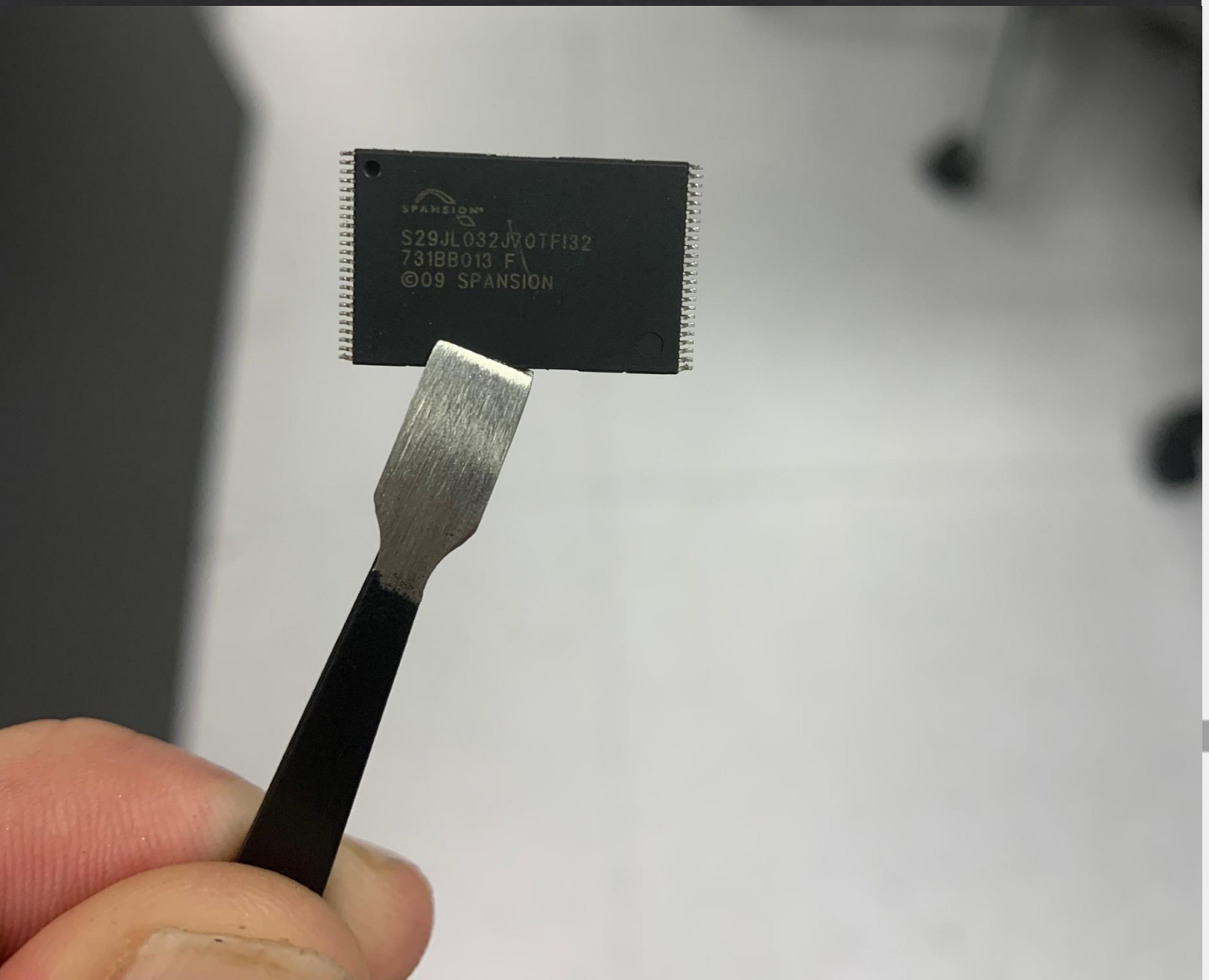
HOW DID IT START?


Heparin
Rate: 10.5ml/h
15
U/kg/h
BOL
OK





Is there code on this?



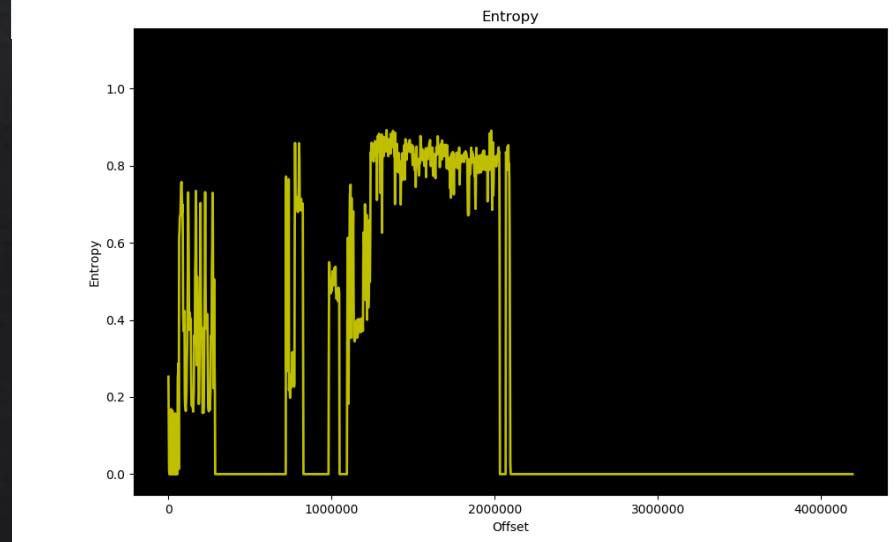
1.. dump_cropped.bin

```
      0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
00000 0000 0001 DC00 FFFF 0000 0000 0000 8D02  ....Ü.ÿÿ..... .
00010 300B E40B 0900 CA00 6D02 3A0B 260B 0900  0.ä...Ê.m.:.&...
00020 8001 1002 440B 300B 0900 1002 8001 4E0B   ...D.0..... .N.
00030 3A0B 0901 6D02 CA00 580B 440B 0900 8D02  :...m.Ê.X.D... .
00040 0000 620B 4E0B 0500 6D02 CA00 6C0B 580B  ..b.N...m.Ê.l.X.
00050 0500 1002 8001 760B 620B 0500 8001 1002  .... .v.b... ...
00060 800B 6C0B 0501 CA00 6D02 8A0B 760B 0500   .l...Ê.m. .v...
00070 0000 8D02 940B 800B 0600 CA00 6D02 9E0B  .. . . ...Ê.m. .
00080 8A0B 0600 8001 1002 A80B 940B 0600 1002   ... ...¨. .....
00090 8001 B20B 9E0B 0601 6D02 CA00 BC0B A80B   .². ...m.Ê.¼.¨.
000A0 0600 8D02 0000 C60B B20B 0A00 6D02 CA00  .. ...Æ.²...m.Ê.
000B0 D00B BC0B 0A00 1002 8001 DA0B C60B 0A00  Ð.¼..... .Ú.Æ...
000C0 8001 1002 E40B D00B 0A01 CA00 6D02 260B   ...ä.Ð...Ê.m.&.
000D0 DA0B 0A00 ED85 0000 9B80 0000 B177 0000  Ú...í .. ..±w..
000E0 CD69 0000 375B 0000 BA4D 0000 1A42 0000  Íi..7[..ºM...B..
000F0 3238 0000 6D30 0000 322A 0000 0A25 0000  28..m0..2*...%..
00100 DA20 0000 721D 0000 A91A 0000 5F18 0000  Ú ..r...©..._...
00110 7C16 0000 EF14 0000 A713 0000 9A12 0000  |...ï...§... ...
00120 BF11 0000 1011 0000 8710 0000 2010 0000  ¿....... ... ...
00130 D80F 0000 AD0F 0000 A00F 0000 260B 0000  Ø...-... ...&...
00140 0000 000A 0100 0001 2802 E001 E001 3002  ........(.à.à.0.
00150 00FF 9EFF 0020 2020 2020 2020 2020 2828  .ÿ ÿ.         ((
00160 2828 2820 2020 2020 2020 2020 2020 2020  (((
00170 2020 2020 2088 1010 1010 1010 1010 1010        ..........
00180 1010 1010 1004 0404 0404 0404 0404 0410  ................
00190 1010 1010 1010 4141 4141 4141 0101 0101  ......AAAAAA....
001A0 0101 0101 0101 0101 0101 0101 0101 0101  ................
001B0 1010 1010 1010 4242 4242 4242 0202 0202  ......BBBBBB....
001C0 0202 0202 0202 0202 0202 0202 0202 0202  ................
001D0 1010 1010 2030 3132 3334 3536 3738 3941  .... 0123456789A
001E0 4243 4445 4600 286E 756C 6C20 706F 696E  BCDEF.(null poin
001F0 7465 7229 0030 3132 3334 3536 3738 3961  ter).0123456789a
00200 6263 6465 6600 3F3F 3F00 00FF 0220 E000  bcdef.???..ÿ. à.
00210 0200 0100 0820 E000 0400 0000 0C20 E000  ..... à..... à.
00220 0100 0000 2120 E000 0100 0000 2A20 E000  ....! à.....* à.
00230 0100 0000 3320 E000 0100 0000 3D20 E000  ....3 à.....= à.
```

Is there code on this?

- Eyeballing (previous slide):
  → Looks like it!
- Entropy analysis (bottom right)
  → Yup, it looks like it!
- Binwalk
  → MIPS? Or ARM? Or Arcadyan (?!) ?
- IDA Pro (top right)
  → Nope! neither MIPS nor ARM and wtf is Arcadyan? ☹

```
ROM:00032605                .byte 0xF8
ROM:00032606                .byte 0x93
ROM:00032607                .byte 0x73  # s
ROM:00032608 # ---------------------------------------------------------------------------
ROM:00032608                sd      $a0, -0x5D3F($zero)
ROM:0003260C                jal     0x2880420
ROM:00032610                sb      $at, -0x35E($s6)
ROM:00032614                ll      $t8, -0x3243($t7)
ROM:00032618                sd      $s2, -0x4ED1($ra)
ROM:0003261C                sdl     $t7, 0x2B3($t9)
ROM:00032620                sltiu   $a0, $t8, -0x4CD1
ROM:00032624                tgeiu   $s4, 0xA38E
ROM:00032628                addi    $gp, $a3, -0x13C6
ROM:0003262C                lw      $s4, -0x7E1D($k0)
ROM:00032630                andi    $t6, $fp, 0xA2C1
ROM:00032634                bne     $a1, $v0, unk_22A78
ROM:00032638                sdr     $s3, 0($s2)
ROM:0003263C                mtlo    $zero, $ac2
ROM:00032640                nop
ROM:00032644                pref    0xB, 0x5F3($t5)
ROM:00032648                lwr     $t6, -0x4FAF($s0)
ROM:0003264C                sllv    $t6, $s2, $a1
ROM:00032650                dsra32  $fp, $zero, 0
ROM:00032654                pref    0x13, 0x3A2($s0)
ROM:00032658                ll      $t4, -0x5D3F($t0)
ROM:0003265C                j       0x2D94C00
ROM:00032660                tne     $zero, $zero  #2
ROM:00032664 # ---------------------------------------------------------------------------
ROM:00032664                beqzl   $t8, loc_32668
ROM:00032668
ROM:00032668 loc_32668:                             # CODE XREF: ROM:00032664↑j
ROM:00032668                lsa     $s5, $a2, $t5, 1
ROM:0003266C                scd     $k0, 0xBB0($gp)
ROM:00032670                beqzl   $t0, unk_1EFB8
ROM:00032670 # ---------------------------------------------------------------------------
ROM:00032674                .byte    4
```



OSINT

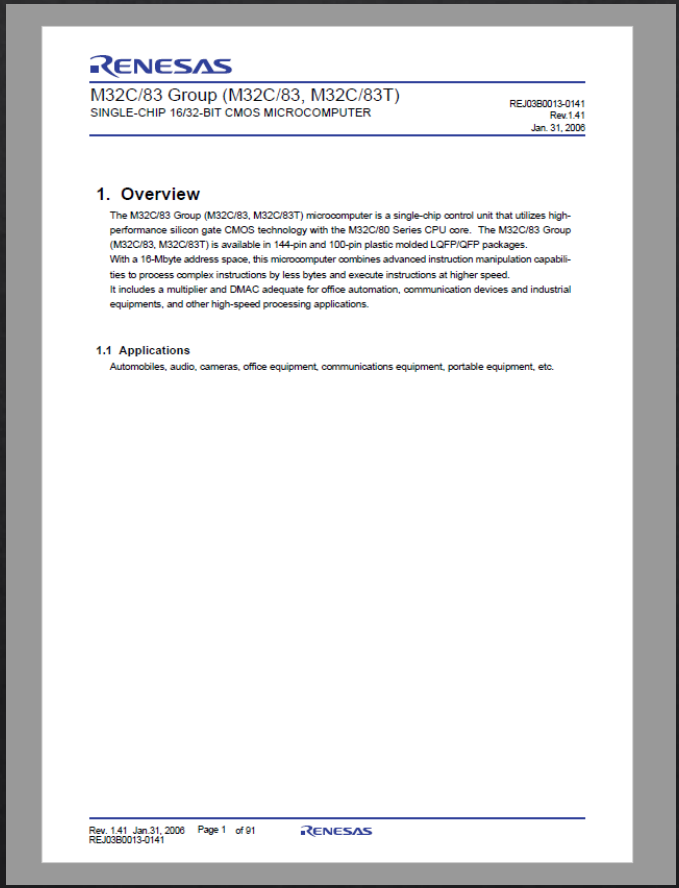
- FDA pre-market filings for the device:

Micro-processor	Renesas Electronics Corporation Dual processors • Functional (FμP)
	• Control (KμP)
Program language	FμP - I-Logix Rhapsody KμP - Hitachi IDE C/C++
Operating system	FμP - EMBOS KμP - CMX

https://www.accessdata.fda.gov/cdrh_docs/pdf17/K172831.pdf

⇒ Renesas M32C

Datasheet + Manuals



RENESAS

M32C/83 Group (M32C/83, M32C/83T)
SINGLE-CHIP 16/32-BIT CMOS MICROCOMPUTER

REJ03B0013-0141
Rev.1.41
Jan. 31, 2006

1. Overview

The M32C/83 Group (M32C/83, M32C/83T) microcomputer is a single-chip control unit that utilizes high-performance silicon gate CMOS technology with the M32C/80 Series CPU core. The M32C/83 Group (M32C/83, M32C/83T) is available in 144-pin and 100-pin plastic molded LQFP/QFP packages.
With a 16-Mbyte address space, this microcomputer combines advanced instruction manipulation capabilities to process complex instructions by less bytes and execute instructions at higher speed.
It includes a multiplier and DMAC adequate for office automation, communication devices and industrial equipments, and other high-speed processing applications.

1.1 Applications

Automobiles, audio, cameras, office equipment, communications equipment, portable equipment, etc.

REJ09B0319-0100

Everywhere you imagine. RENESAS

16/32

M32C/80 Series
Software Manual

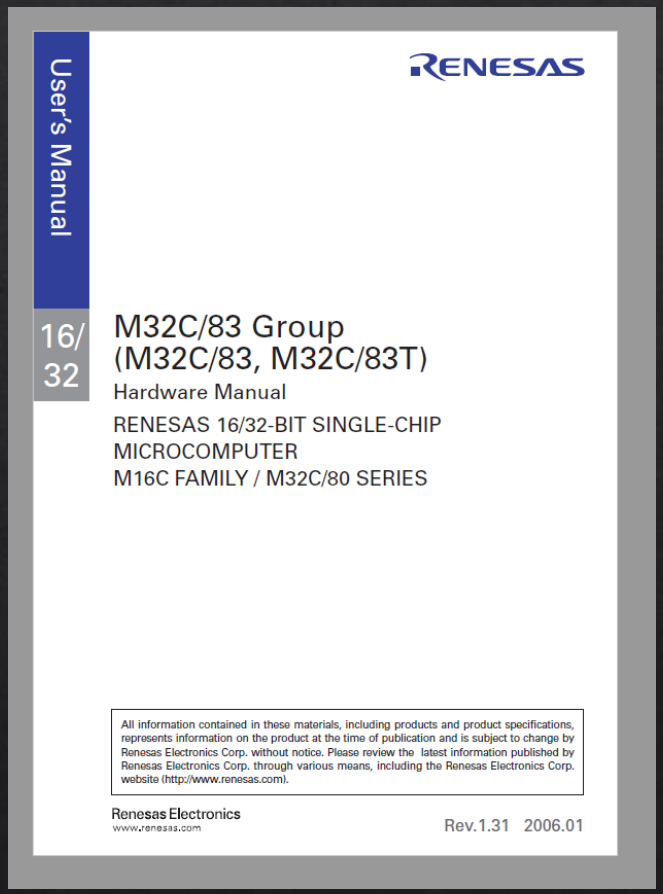
RENESAS 16/32-BIT SINGLE-CHIP MICROCOMPUTER
M16C FAMILY / M32C/80 SERIES

Software Manual

Before using this material, please visit our website to verify that this is the most updated document available.

Rev. 1.00
Revision date: May 31, 2006

Renesas Technology
www.renesas.com



User's Manual

16/32

RENESAS

M32C/83 Group
(M32C/83, M32C/83T)
Hardware Manual

RENESAS 16/32-BIT SINGLE-CHIP MICROCOMPUTER
M16C FAMILY / M32C/80 SERIES

Renesas Electronics
www.renesas.com

Rev.1.31 2006.01

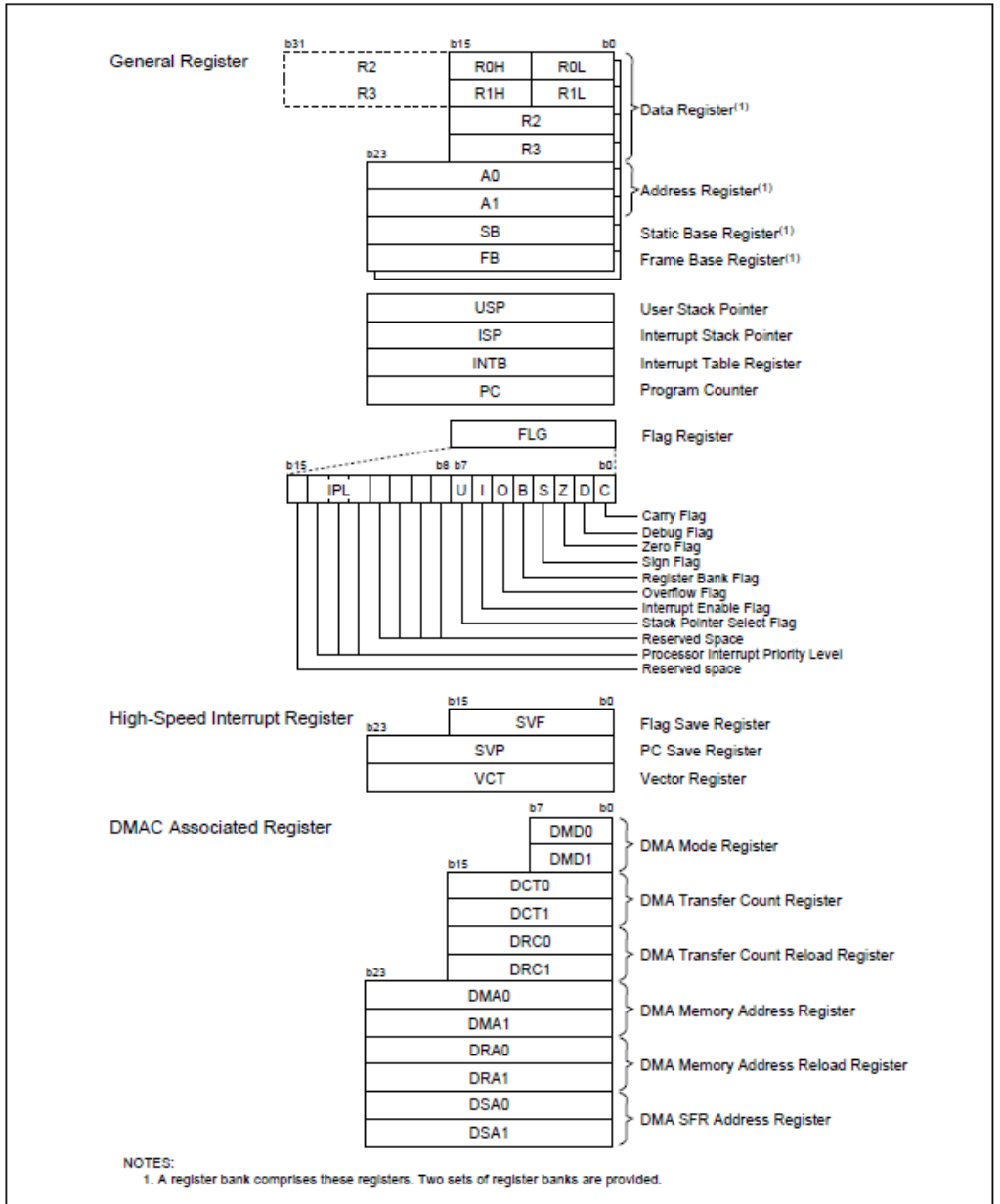
M32C ASSEMBLY (TL;DR!)

- R0-R3 → 16 bits registers
- R2R0, R3R1 → 32 bits registers made by combining two 16 bits regs
- A0, A1: “address” 24 bits registers
- INTB → location of interrupt table

- CISC instructions set; ADD, MOV, MUL, RTS (return) etc.
  - Goes left to right (src first, dst second)
  - MOV #42, R0
  - Read the manual for the funkier operations

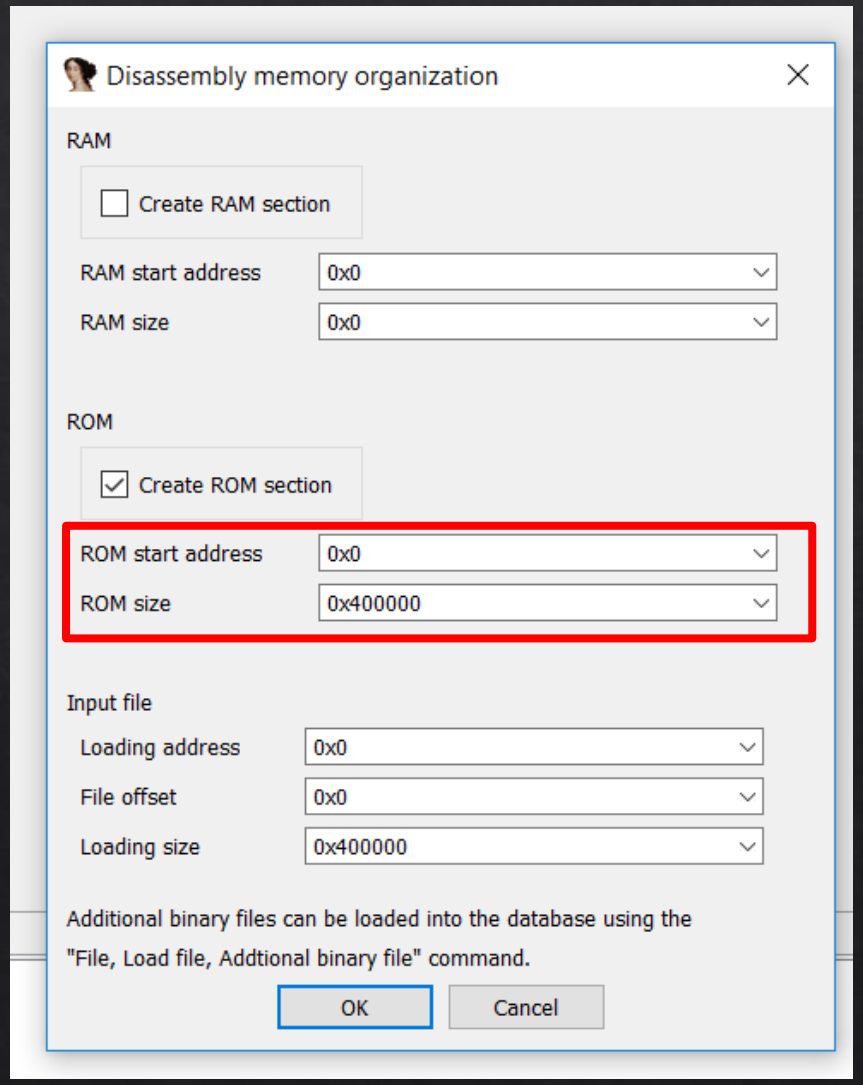
2. Central Processing Unit (CPU)

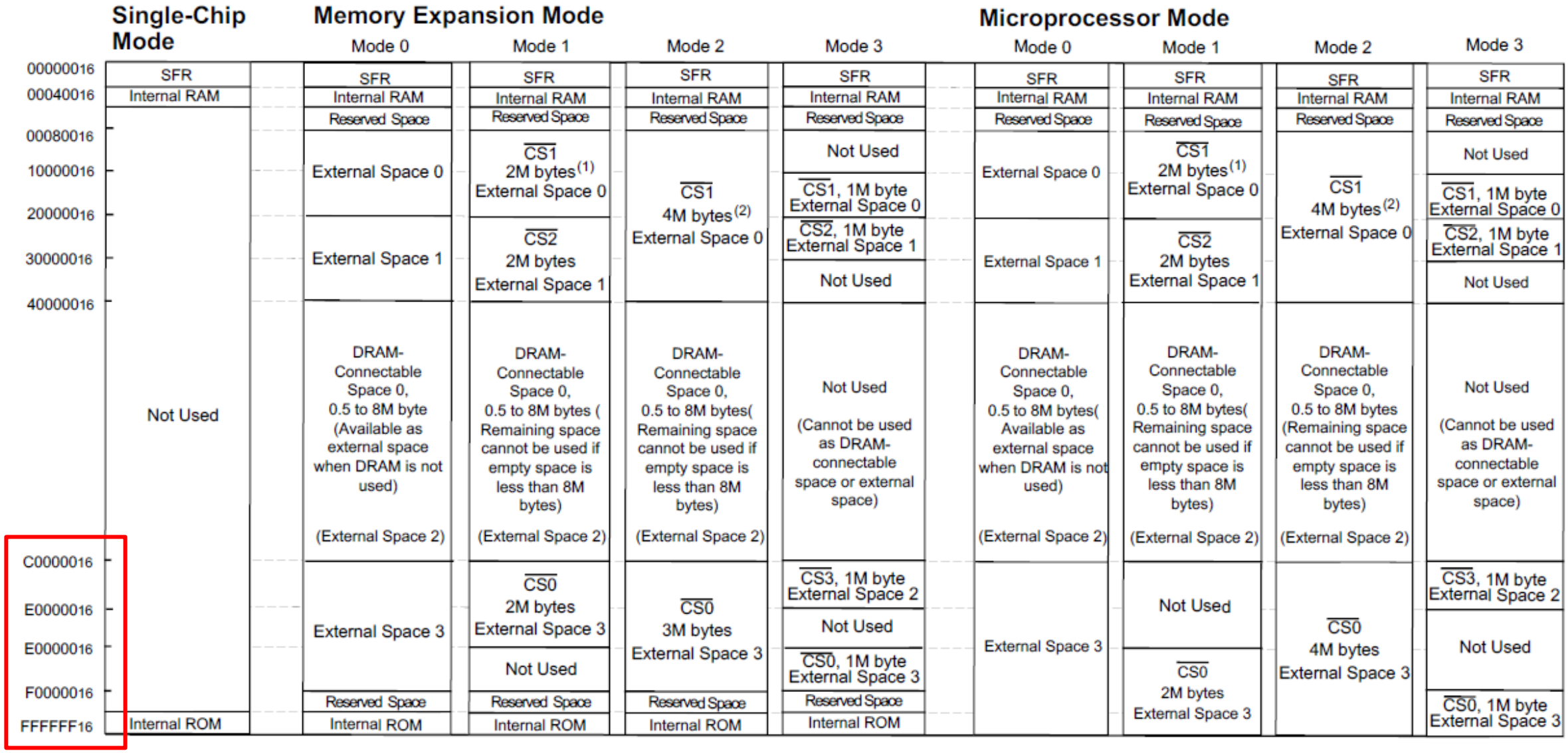
Figure 2.1 shows the CPU registers.
A register bank comprises 8 registers (R0, R1, R2, R3, A0, A1, SB and FB) out of 28 CPU registers. Two sets of register banks are provided.



Importing in IDA?

- Support for M32C/80
  - We have a M32C/83, close enough….
- Finding the load address?
  - Common way
    - Strings, pointers, interrupt table ⇒ guess best candidate
  - Datasheet!





The WCR register determines how many wait states are inserted for each space $\overline{CS0}$ to $\overline{CS3}$.

NOTES:
1. $20000016 - 00800016 = 2016K$ bytes. 32K bytes less than 2M bytes.
2. $40000016 - 00800016 = 4064K$ bytes. 32K bytes less than 4M bytes.

Memory Map

- Potential interesting values:
  - 0xC00000 (External space?)
  - 0xE00000 (External space?)
  - 0xFFFE00 (Vectors)

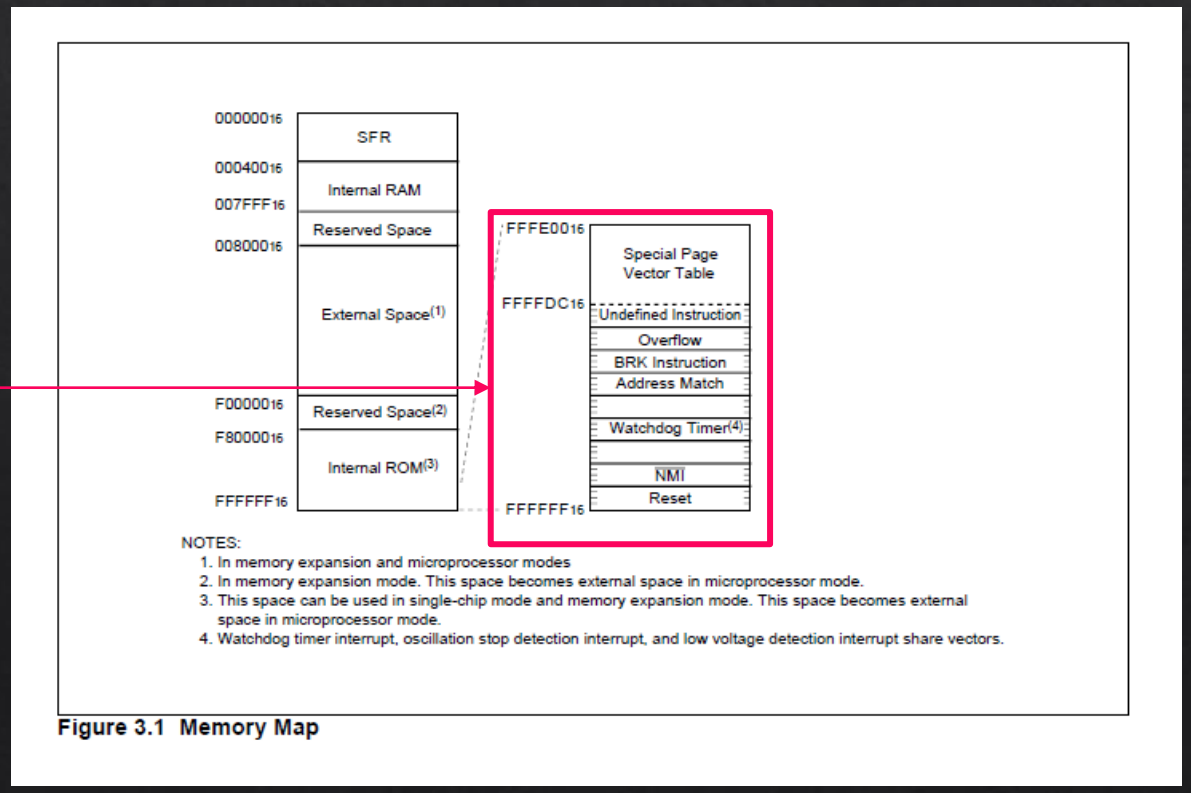


Figure 3.1 Memory Map

Memory Map

- Potential interesting values:
  - 0xC00000 (External space?)
  - 0xE00000 (External space?)
  - 0xFFFE00 (Vectors)

⇒ Try to load the flash dump at 0xE00000, see pointers at 0xFFFFFC

```
ROM:00FFFFDC                    .LWORD int_fixed_default
ROM:00FFFFE0                    .LWORD int_fixed_default
ROM:00FFFFE4                    .LWORD int_fixed_default
ROM:00FFFFE8                    .LWORD int_fixed_default
ROM:00FFFFEC                    .LWORD int_fixed_default
ROM:00FFFFF0                    .LWORD int_fixed_default
ROM:00FFFFF4                    .LWORD int_fixed_default
ROM:00FFFFF8                    .LWORD int_NMI
ROM:00FFFFFC    off_FFFFFC      .LWORD reset_interrupt
ROM:00FFFFFC    ; end of 'ROM'
ROM:00FFFFFC
```

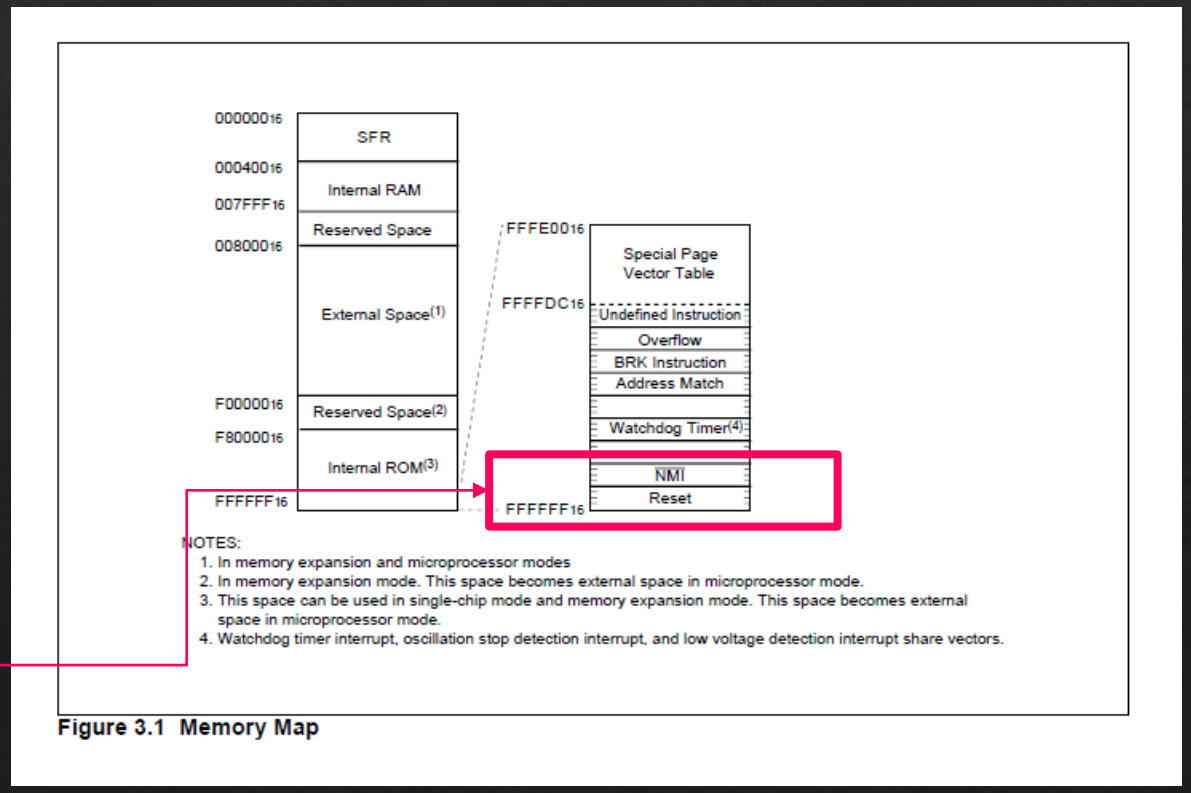


Figure 3.1 Memory Map

Memory Map

- Potential interesting values:
  - 0xC00000 (External space?)
  - 0xE00000 (External space?)
  - 0xFFFE00 (Vectors)

⇒ Try to load the flash dump at 0xE00000, see pointers at 0xFFFFFC

```
ROM:00FFFFDC                    .LWORD int_fixed_default
ROM:00FFFFE0                    .LWORD int_fixed_default
ROM:00FFFFE4                    .LWORD int_fixed_default
ROM:00FFFFE8                    .LWORD int_fixed_default
ROM:00FFFFEC                    .LWORD int_fixed_default
ROM:00FFFFF0                    .LWORD int_fixed_default
ROM:00FFFFF4                    .LWORD int_fixed_default
ROM:00FFFFF8                    .LWORD int_NMI
ROM:00FFFFFC    off_FFFFFC      .LWORD reset_interrupt
ROM:00FFFFFC    ; end of 'ROM'
ROM:00FFFFFC
```

FLASH DATA (Permanent storage)

```
off_F121BC      .LWORD unk_E00000       ; DATA XREF:
                                        ; flash_secto
                .LWORD flash_calib_data_checksum
                .LWORD flash_ADJDATACHKSUM
                .LWORD flash_disposable
                .LWORD unk_E08000
                .LWORD off_E0A000
                .LWORD unk_E0C000
                .LWORD unk_E0E000
                .LWORD unk_E10000
                .LWORD byte_E20000
                .LWORD unk_E30000
                .LWORD unk_E40000
                .LWORD unk_E50000
                .LWORD unk_E60000
                .LWORD unk_E70000
                .LWORD unk_E80000
                .LWORD unk_E90000
                .LWORD unk_EA0000
                .LWORD dword_EB0000
                .LWORD unk_EC0000
                .LWORD unk_ED0000
                .LWORD unk_EE0000
                .LWORD unk_EF0000
                .LWORD unk_F00000
                .BYTE    0
```

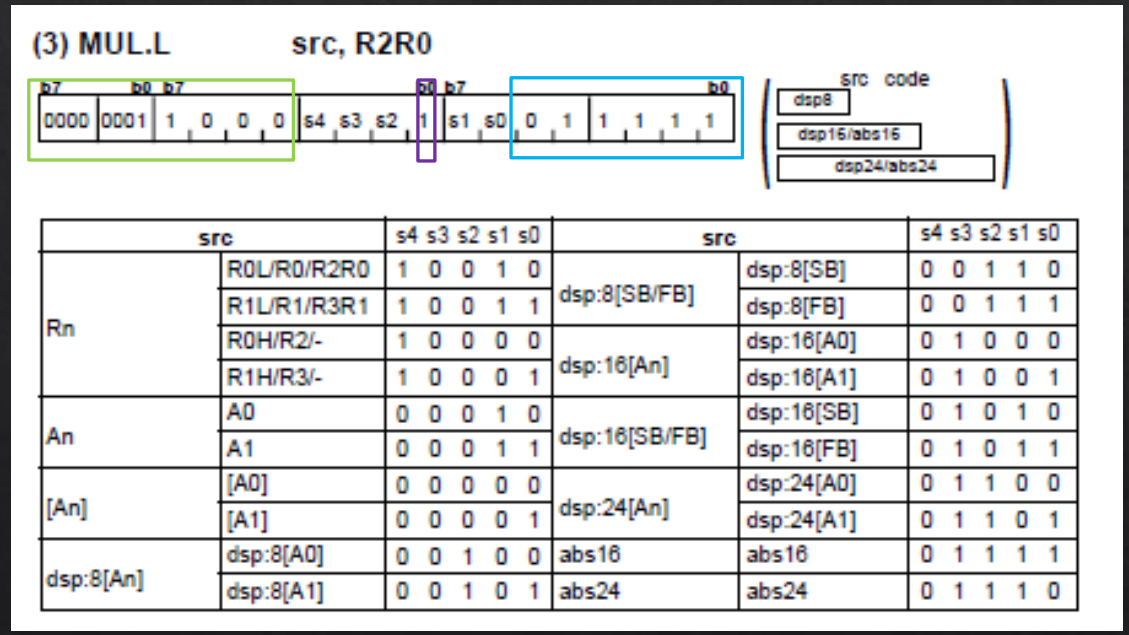
Importing in IDA

- Problem! Some instructions not decoded properly in 7.4 (fixed in later updates)
  - \x01\x89\xDF is not recognized
  - Need to look into M32C software manual!
- Convert opcode into binary representation
  - 0000.0001 | 1000.100 1 | 11 01.1111

```
ROM:00FEDA01 sub_FEDA01:
ROM:00FEDA01
ROM:00FEDA01 var_6           = -6
ROM:00FEDA01
ROM:00FEDA01                 MOV.L   #24837Ch, A0
ROM:00FEDA05                 EXTZ    var_6[FB], R1
ROM:00FEDA09                 MOV.W   #0, R3
ROM:00FEDA0B                 MOVX    #0Eh, R2R0
ROM:00FEDA0B ; ---------------------------------------
ROM:00FEDA0E                 .BYTE    1
ROM:00FEDA0F                 .BYTE  89h
ROM:00FEDA10                 .BYTE 0DFh
ROM:00FEDA11 ; ---------------------------------------
ROM:00FEDA11 mul r3r1, r2r0
ROM:00FEDA11                 ADD.L   R2R0, A0
ROM:00FEDA13                 RTS
ROM:00FEDA13 ; End of function sub_FEDA01
ROM:00FEDA13
```

Importing in IDA

- Problem! Some instructions not decoded properly in 7.4 (fixed in later updates)
  - \x01\x89\xDF is not recognized
  - Need to look into M32C software manual!
- Convert opcode into binary representation
  - 0000.0001 | 1000.100 1 | 11 01.1111
  - MUL.L src, R2R0
    - Src: 10011 → R3R1
- Instruction is MUL.L R3R1, R2R0



(3) MUL.L src, R2R0

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

IDENTIFYING USEFUL CODE PATTERNS

API Call Conventions

- Not quite commanded by arch; arbitrary for OS/Compiler/…
- R0 or R2R0 usually hold the return value
- First arg is usually R0 (value) or A0 (address)
  - next args pushed onto the stack; cleanup by caller

```
loc_FEF144:                                     ; CO
                MOV.L   R3R1, R2R0
                POPM    R1,R3
                RTS
; End of function system_probably_get_time
```

R2R0 gets the return value of the function

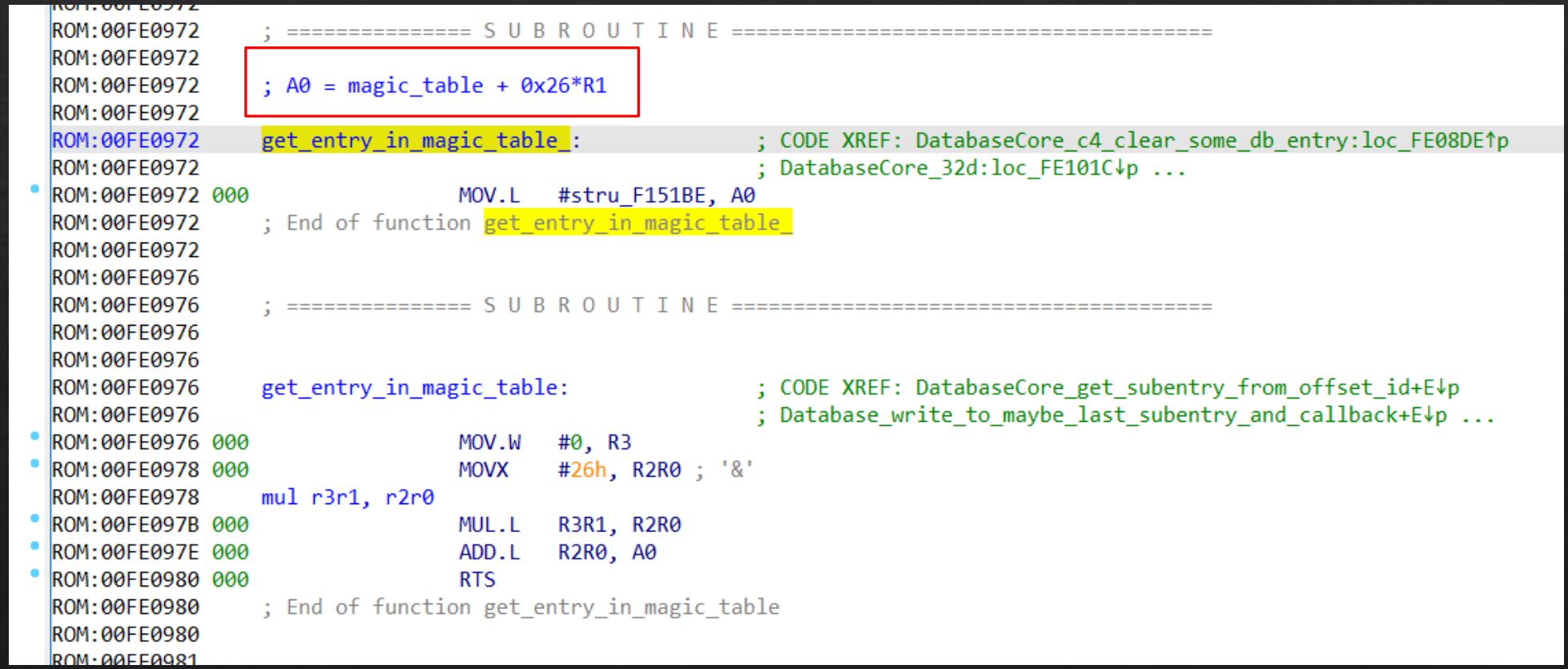
```
PUSH.L  #210h
PUSH.W  #0
MOV.L   #unk_213B0C, A0
JSR.A   memset              ; memset(A0: dst, arg_2:val, arg_6: n)
ADD.L   #12h, SP
```

Args pushed on the stack, cleanup by caller.

API Call Conventions

- YOLO functions: ignore calling convention and modify random registers; fall through the next func

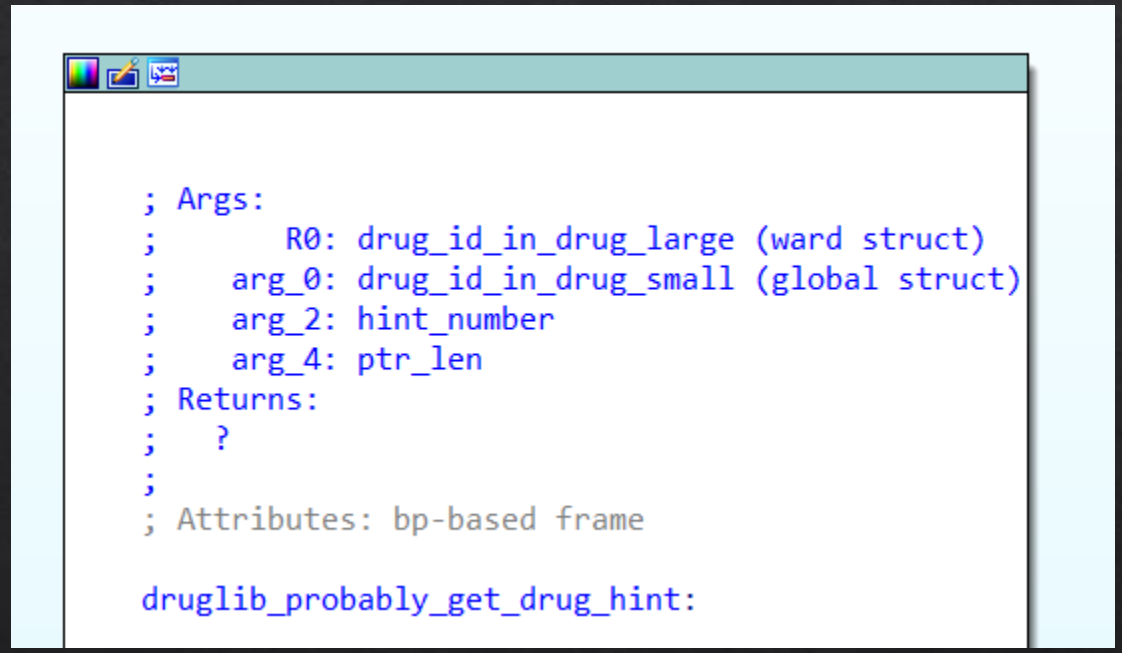
  → probably compiler optimization ("helper" functions optimized for space)



```
ROM:00FE0972     ; =============== S U B R O U T I N E =======================================
ROM:00FE0972
ROM:00FE0972     ; A0 = magic_table + 0x26*R1
ROM:00FE0972
ROM:00FE0972     get_entry_in_magic_table_:              ; CODE XREF: DatabaseCore_c4_clear_some_db_entry:loc_FE08DE↑p
ROM:00FE0972                                             ; DatabaseCore_32d:loc_FE101C↓p ...
ROM:00FE0972 000                 MOV.L   #stru_F151BE, A0
ROM:00FE0972     ; End of function get_entry_in_magic_table_
ROM:00FE0972
ROM:00FE0976
ROM:00FE0976     ; =============== S U B R O U T I N E =======================================
ROM:00FE0976
ROM:00FE0976
ROM:00FE0976     get_entry_in_magic_table:               ; CODE XREF: DatabaseCore_get_subentry_from_offset_id+E↓p
ROM:00FE0976                                             ; Database_write_to_maybe_last_subentry_and_callback+E↓p ...
ROM:00FE0976 000                 MOV.W   #0, R3
ROM:00FE0978 000                 MOVX    #26h, R2R0 ; '&'
ROM:00FE0978     mul r3r1, r2r0
ROM:00FE097B 000                 MUL.L   R3R1, R2R0
ROM:00FE097E 000                 ADD.L   R2R0, A0
ROM:00FE0980 000                 RTS
ROM:00FE0980     ; End of function get_entry_in_magic_table
ROM:00FE0980
ROM:00FE0981
```

API Call Conventions

- Advice: comment the function arguments
  - Verbose function name and args
  - Specify all the arguments w/ position on stack
  - Markers of certainty (maybe, probably, …)
- Mix of regular comment (‘:’ key) and repeat comments (‘;’ key)
  - Regular: only show up where you put it
  - Repeat: will show up everywhere code is referenced

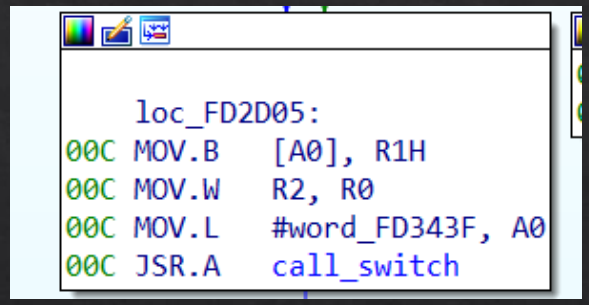



```
PUSH.L  #210h
PUSH.W  #0
MOV.L   #unk_213B0C, A0
JSR.A   memset
ADD.L   #12h, SP
```

memset(A0: dst, arg_2:val, arg_6: n)

Switch Statement

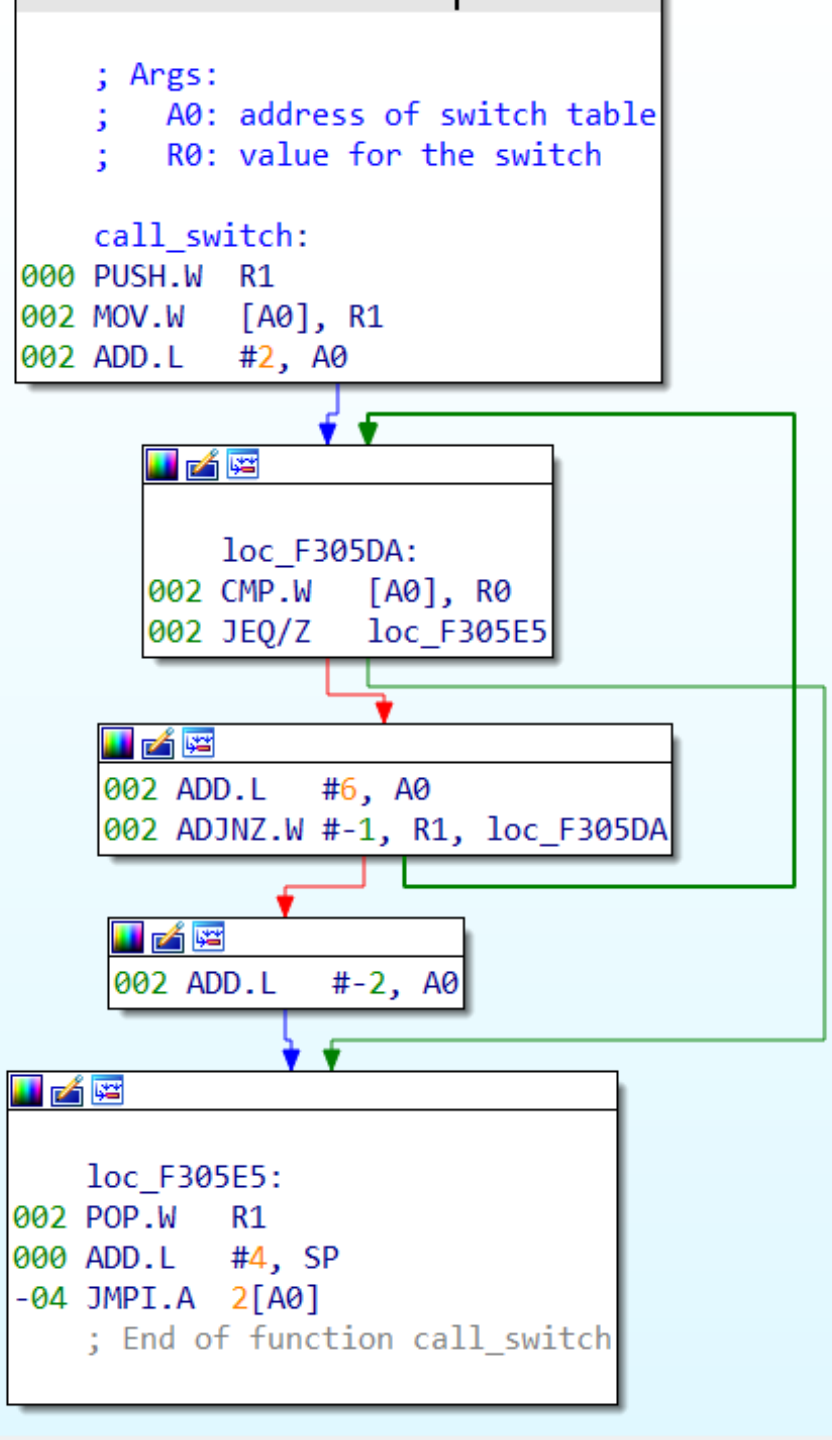
- Firmware uses a dedicated function to implement switch pattern
  - A0: table containing list of cases and addresses
  - R0: value to switch on



- IDA doesn't recognize it automatically
  - There's a "manual switch declaration" but seems unapplicable here
- IDAPython to make disassembly easier to read
  - Find all switch statements,
  - Automatically add switch label to every basic block (ex: 'case 1:', case '0x42', etc.)

```
.WORD 0Ch                ; DATA XREF: CustTrigger_3c+1A↑o
                         ; <- entry count
.WORD 3Fh
.LWORD loc_FD2D1A
.WORD 40h
.LWORD loc_FD2D2C
.WORD 41h
.LWORD loc_FD2D3E
.WORD 42h
.LWORD loc_FD2D50
.WORD 46h
.LWORD loc_FD2D62
.WORD 15Ch
.LWORD loc_FD2DD9
.WORD 1E3h
.LWORD loc_FD2D73
.WORD 1E4h
.LWORD loc_FD2D84
.WORD 1E5h
.LWORD loc_FD2D95
.WORD 1E6h
.LWORD loc_FD2DA6
.WORD 1E7h
.LWORD loc_FD2DC8
.WORD 5FAh
.LWORD loc_FD2DB7
.LWORD loc_FD2D11        ; <- default case
```

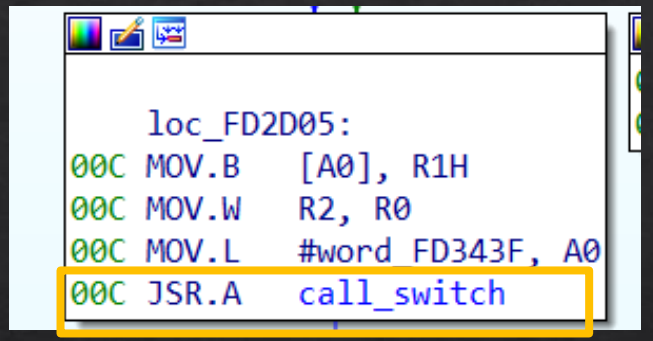
- Start with number of entries (ex: 0xC)
- Then 0xC entries: [value, address]
- Final address is the default case



- Switch function (left)
- Example of switch table (bottom left)
  - Start with number of entries (ex: 0xC)
  - Then 0xC entries: [value, address]
  - Final address is the default case

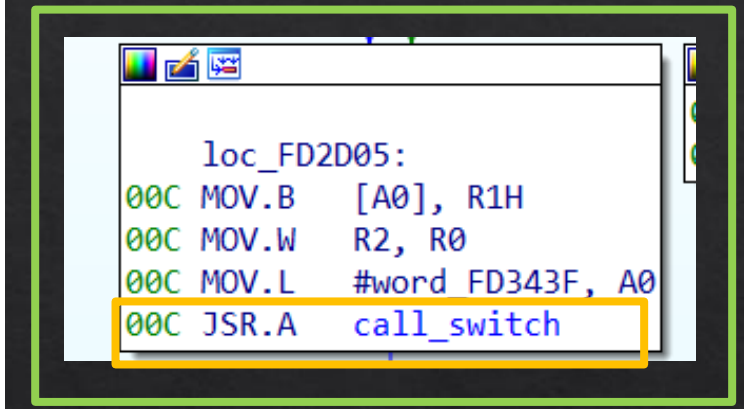
```
.WORD 0Ch               ; DATA XREF: CustTrigger_3c+1A↑o
                        ; <- entry count
.WORD 3Fh
.LWORD loc_FD2D1A
.WORD 40h
.LWORD loc_FD2D2C
.WORD 41h
.LWORD loc_FD2D3E
.WORD 42h
.LWORD loc_FD2D50
.WORD 46h
.LWORD loc_FD2D62
.WORD 15Ch
.LWORD loc_FD2DD9
.WORD 1E3h
.LWORD loc_FD2D73
.WORD 1E4h
.LWORD loc_FD2D84
.WORD 1E5h
.LWORD loc_FD2D95
.WORD 1E6h
.LWORD loc_FD2DA6
.WORD 1E7h
.LWORD loc_FD2DC8
.WORD 5FAh
.LWORD loc_FD2DB7
.LWORD loc_FD2D11       ; <- default case
```

```
switch_ea = 0x0F305D4
rom_start_ea = 0x00F00000
rom_end_ea =  0x00FFFFFF

for xref in XrefsTo(switch_ea):
    call_ea = xref.frm
    table_ea = -1
    print hex(call_ea)
    prev_heads = map(long, Heads(call_ea - 20, call_ea)) # Look a few instruction backwards to
    prev_heads.reverse()
    for h in prev_heads:
        line = generate_disasm_line(h, 0)
        if "MOV" in line:
            if ", A0" in line:
                ins = DecodeInstruction(h)
                table_ea = ins.Op1.value # address of the table should be in the mov op1
                ida_offset.op_offset(h , 0, idc.REF_OFF32) #mark it as an offset
                break
    print table_ea
    if  (table_ea < rom_start_ea) or (table_ea > rom_end_ea):
        print "Warning, table not in range at " + hex(call_ea)
    else:
        print hex(table_ea)
        process_switch_table(table_ea)
```



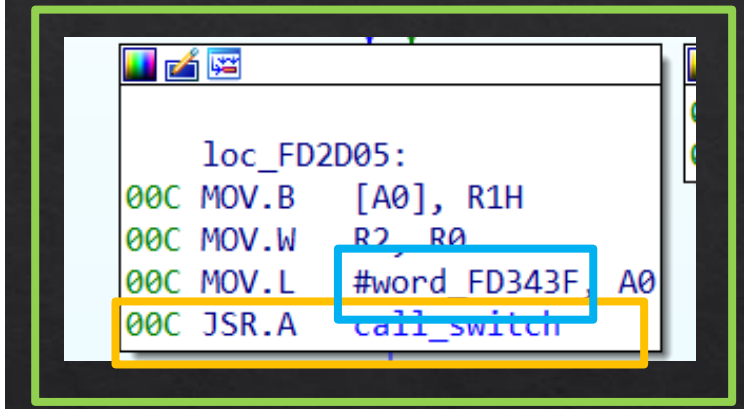
1. Iterates over the XREF of the call_switch function.
2. Get a few instruction backwards from each call to the function.
3. Recover the content of A0 which will be the address of the switch table.
4. Run our process_switch_table function (next slide)

```
switch_ea = 0x0F305D4
rom_start_ea = 0x00F00000
rom_end_ea =  0x00FFFFFF

for xref in XrefsTo(switch_ea):
    call_ea = xref.frm
    table_ea = -1
    print hex(call_ea)
    prev_heads = map(long, Heads(call_ea - 20, call_ea)) # Look a few instruction backwards to
    prev_heads.reverse()
    for h in prev_heads:
        line = generate_disasm_line(h, 0)
        if "MOV" in line:
            if ", A0" in line:
                ins = DecodeInstruction(h)
                table_ea = ins.Op1.value # address of the table should be in the mov op1
                ida_offset.op_offset(h , 0, idc.REF_OFF32) #mark it as an offset
                break
    print table_ea
    if  (table_ea < rom_start_ea) or (table_ea > rom_end_ea):
        print "Warning, table not in range at " + hex(call_ea)
    else:
        print hex(table_ea)
        process_switch_table(table_ea)
```



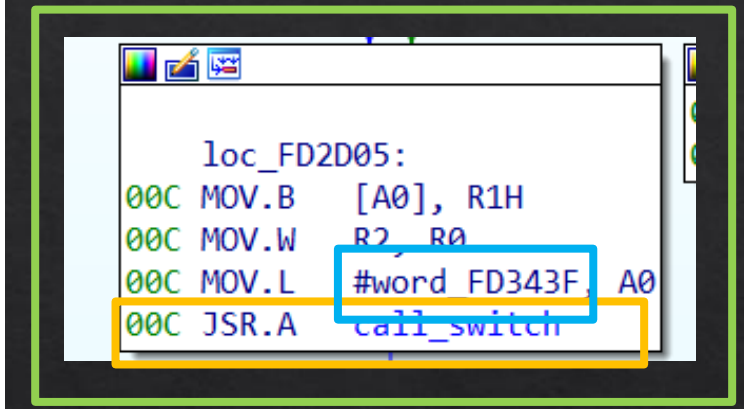
1. Iterates over the XREF of the call_switch function.
2. Get a few instruction backwards from each call to the function.
3. Recover the content of A0 which will be the address of the switch table.
4. Run our process_switch_table function (next slide)

```
switch_ea = 0x0F305D4
rom_start_ea = 0x00F00000
rom_end_ea =  0x00FFFFFF

for xref in XrefsTo(switch_ea):
    call_ea = xref.frm
    table_ea = -1
    print hex(call_ea)
    prev_heads = map(long, Heads(call_ea - 20, call_ea)) # Look a few instruction backwards to
    prev_heads.reverse()
    for h in prev_heads:
        line = generate_disasm_line(h, 0)
        if "MOV" in line:
            if ", A0" in line:
                ins = DecodeInstruction(h)
                table_ea = ins.Op1.value # address of the table should be in the mov op1
                ida_offset.op_offset(h , 0, idc.REF_OFF32) #mark it as an offset
                break
    print table_ea
    if  (table_ea < rom_start_ea) or (table_ea > rom_end_ea):
        print "Warning, table not in range at " + hex(call_ea)
    else:
        print hex(table_ea)
        process_switch_table(table_ea)
```



1. Iterates over the XREF of the call_switch function.
2. Get a few instruction backwards from each call to the function.
3. Recover the content of A0 which will be the address of the switch table.
4. Run our process_switch_table function (next slide)

```
switch_ea = 0x0F305D4
rom_start_ea = 0x00F00000
rom_end_ea =  0x00FFFFFF

for xref in XrefsTo(switch_ea):
    call_ea = xref.frm
    table_ea = -1
    print hex(call_ea)
    prev_heads = map(long, Heads(call_ea - 20, call_ea)) # Look a few instruction backwards to
    prev_heads.reverse()
    for h in prev_heads:
        line = generate_disasm_line(h, 0)
        if "MOV" in line:
            if ", A0" in line:
                ins = DecodeInstruction(h)
                table_ea = ins.Op1.value # address of the table should be in the mov op1
                ida_offset.op_offset(h , 0, idc.REF_OFF32) #mark it as an offset
                break
    print table_ea
    if  (table_ea < rom_start_ea) or (table_ea > rom_end_ea):
        print "Warning, table not in range at " + hex(call_ea)
    else:
        print hex(table_ea)
        process_switch_table(table_ea)
```

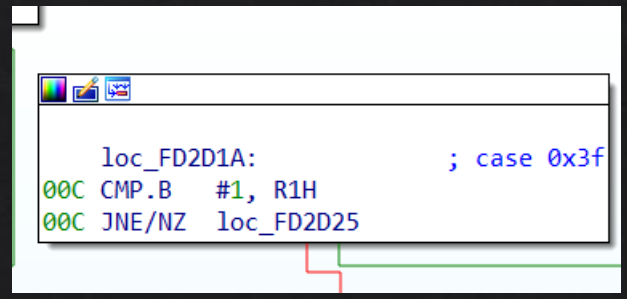


1. Iterates over the XREF of the call_switch function.
2. Get a few instruction backwards from each call to the function.
3. Recover the content of A0 which will be the address of the switch table.
4. Run our process_switch_table function (next slide)

```python
def process_switch_table(table_ea):
    ea = table_ea
    n_entries = get_wide_word(ea)
    print "n_entries: " + hex(n_entries)
    del_items(ea, DELIT_SIMPLE, 2 + n_entries*6 + 4) #unmark in case of crap
    create_word(ea)
    ea +=2
    for i in xrange(0, n_entries):
        create_word(ea)
        case_val = get_wide_word(ea)
        offset = get_wide_dword(ea+2)
        set_cmt(offset, "case " + hex(case_val), False)
        ida_offset.op_offset(ea+2, 0, idc.REF_OFF32)
        create_insn(offset) # MakeCode
        ea += 6
    offset = get_wide_dword(ea)
    set_cmt(offset, "default case", False)
    ida_offset.op_offset(ea, 0, idc.REF_OFF32)
    create_insn(offset)
    pass
```

```
.WORD 0Ch               ; DATA XREF: CustTrigger_3c+1A↑o
                        ; <- entry count
.WORD 3Fh
.LWORD loc_FD2D1A
.WORD 40h
.LWORD loc_FD2D2C
.WORD 41h
.LWORD loc_FD2D3E
.WORD 42h
.LWORD loc_FD2D50
.WORD 46h
.LWORD loc_FD2D62
.WORD 15Ch
.LWORD loc_FD2DD9
.WORD 1E3h
.LWORD loc_FD2D73
.WORD 1E4h
.LWORD loc_FD2D84
.WORD 1E5h
.LWORD loc_FD2D95
.WORD 1E6h
.LWORD loc_FD2DA6
.WORD 1E7h
.LWORD loc_FD2DC8
.WORD 5FAh
.LWORD loc_FD2DB7
.LWORD loc_FD2D11       ; <- default case
```

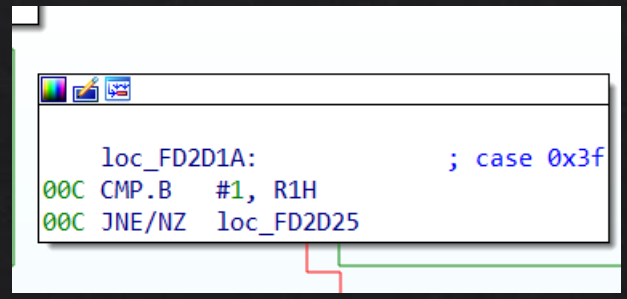
1. Read number of entries in the table
2. Undefine everything (prevent conflicts)
3. Create words (each entry number) and offsets (each entry address)
4. Create comment “case xxx” in the basic block of the case
5. Mark the relevant basic blocks as code



```python
def process_switch_table(table_ea):
    ea = table_ea
    n_entries = get_wide_word(ea)
    print "n_entries: " + hex(n_entries)
    del_items(ea, DELIT_SIMPLE, 2 + n_entries*6 + 4) #unmark in case of crap
    create_word(ea)
    ea +=2
    for i in xrange(0, n_entries):
        create_word(ea)
        case_val = get_wide_word(ea)
        offset = get_wide_dword(ea+2)
        set_cmt(offset, "case " + hex(case_val), False)
        ida_offset.op_offset(ea+2, 0, idc.REF_OFF32)
        create_insn(offset) # MakeCode
        ea += 6
    offset = get_wide_dword(ea)
    set_cmt(offset, "default case", False)
    ida_offset.op_offset(ea, 0, idc.REF_OFF32)
    create_insn(offset)
    pass
```

```
.WORD 0Ch               ; DATA XREF: CustTrigger_3c+1A↑o
                        ; <- entry count
.WORD 3Fh
.LWORD loc_FD2D1A
.WORD 40h
.LWORD loc_FD2D2C
.WORD 41h
.LWORD loc_FD2D3E
.WORD 42h
.LWORD loc_FD2D50
.WORD 46h
.LWORD loc_FD2D62
.WORD 15Ch
.LWORD loc_FD2DD9
.WORD 1E3h
.LWORD loc_FD2D73
.WORD 1E4h
.LWORD loc_FD2D84
.WORD 1E5h
.LWORD loc_FD2D95
.WORD 1E6h
.LWORD loc_FD2DA6
.WORD 1E7h
.LWORD loc_FD2DC8
.WORD 5FAh
.LWORD loc_FD2DB7
.LWORD loc_FD2D11       ; <- default case
```

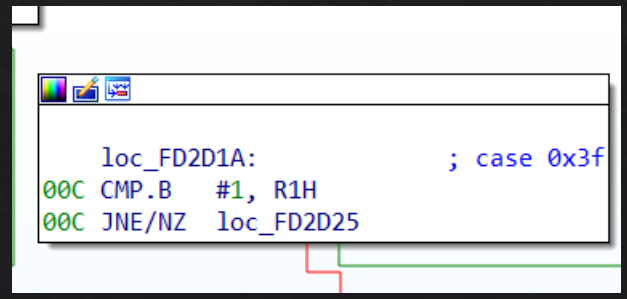
1. Read number of entries in the table
2. Undefine everything (prevent conflicts)
3. Create words (each entry number) and offsets (each entry address)
4. Create comment “case xxx” in the basic block of the case
5. Mark the relevant basic blocks as code



```
def process_switch_table(table_ea):
    ea = table_ea
    n_entries = get_wide_word(ea)
    print "n_entries: " + hex(n_entries)
    del_items(ea, DELIT_SIMPLE, 2 + n_entries*6 + 4) #unmark in case of crap
    create_word(ea)
    ea +=2
    for i in xrange(0, n_entries):
        create_word(ea)
        case_val = get_wide_word(ea)
        offset = get_wide_dword(ea+2)
        set_cmt(offset, "case " + hex(case_val), False)
        ida_offset.op_offset(ea+2, 0, idc.REF_OFF32)
        create_insn(offset) # MakeCode
        ea += 6
    offset = get_wide_dword(ea)
    set_cmt(offset, "default case", False)
    ida_offset.op_offset(ea, 0, idc.REF_OFF32)
    create_insn(offset)
    pass
```

```
.WORD 0Ch               ; DATA XREF: CustTrigger_3c+1A↑o
                        ; <- entry count
.WORD 3Fh
.LWORD loc_FD2D1A
.WORD 40h
.LWORD loc_FD2D2C
.WORD 41h
.LWORD loc_FD2D3E
.WORD 42h
.LWORD loc_FD2D50
.WORD 46h
.LWORD loc_FD2D62
.WORD 15Ch
.LWORD loc_FD2DD9
.WORD 1E3h
.LWORD loc_FD2D73
.WORD 1E4h
.LWORD loc_FD2D84
.WORD 1E5h
.LWORD loc_FD2D95
.WORD 1E6h
.LWORD loc_FD2DA6
.WORD 1E7h
.LWORD loc_FD2DC8
.WORD 5FAh
.LWORD loc_FD2DB7
.LWORD loc_FD2D11       ; <- default case
```

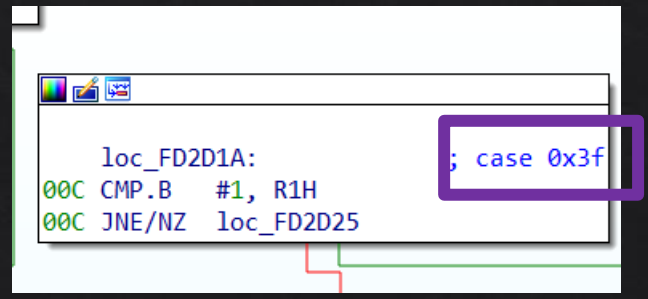
1. Read number of entries in the table
2. Undefine everything (prevent conflicts)
3. Create words (each entry number) and offsets (each entry address)
4. Create comment “case xxx” in the basic block of the case
5. Mark the relevant basic blocks as code



```
def process_switch_table(table_ea):
    ea = table_ea
    n_entries = get_wide_word(ea)
    print "n_entries: " + hex(n_entries)
    del_items(ea, DELIT_SIMPLE, 2 + n_entries*6 + 4) #unmark in case of crap
    create_word(ea)
    ea +=2
    for i in xrange(0, n_entries):
        create_word(ea)
        case_val = get_wide_word(ea)
        offset = get_wide_dword(ea+2)
        set_cmt(offset, "case " + hex(case_val), False)
        ida_offset.op_offset(ea+2, 0, idc.REF_OFF32)
        create_insn(offset) # MakeCode
        ea += 6
    offset = get_wide_dword(ea)
    set_cmt(offset, "default case", False)
    ida_offset.op_offset(ea, 0, idc.REF_OFF32)
    create_insn(offset)
    pass
```

```
.WORD 0Ch               ; DATA XREF: CustTrigger_3c+1A↑o
                        ; <- entry count
.WORD 3Fh
.LWORD loc_FD2D1A
.WORD 40h
.LWORD loc_FD2D2C
.WORD 41h
.LWORD loc_FD2D3E
.WORD 42h
.LWORD loc_FD2D50
.WORD 46h
.LWORD loc_FD2D62
.WORD 15Ch
.LWORD loc_FD2DD9
.WORD 1E3h
.LWORD loc_FD2D73
.WORD 1E4h
.LWORD loc_FD2D84
.WORD 1E5h
.LWORD loc_FD2D95
.WORD 1E6h
.LWORD loc_FD2DA6
.WORD 1E7h
.LWORD loc_FD2DC8
.WORD 5FAh
.LWORD loc_FD2DB7
.LWORD loc_FD2D11       ; <- default case
```

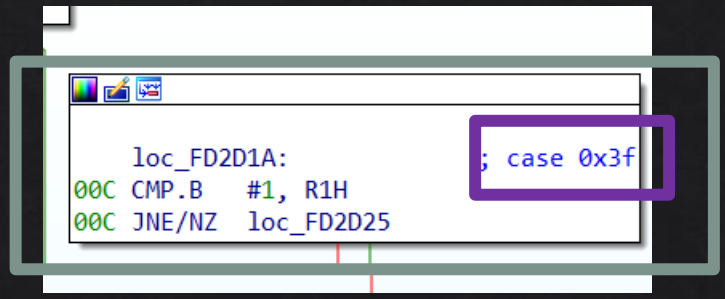
1. Read number of entries in the table
2. Undefine everything (prevent conflicts)
3. Create words (each entry number) and offsets (each entry address)
4. Create comment “case xxx” in the basic block of the case
5. Mark the relevant basic blocks as code

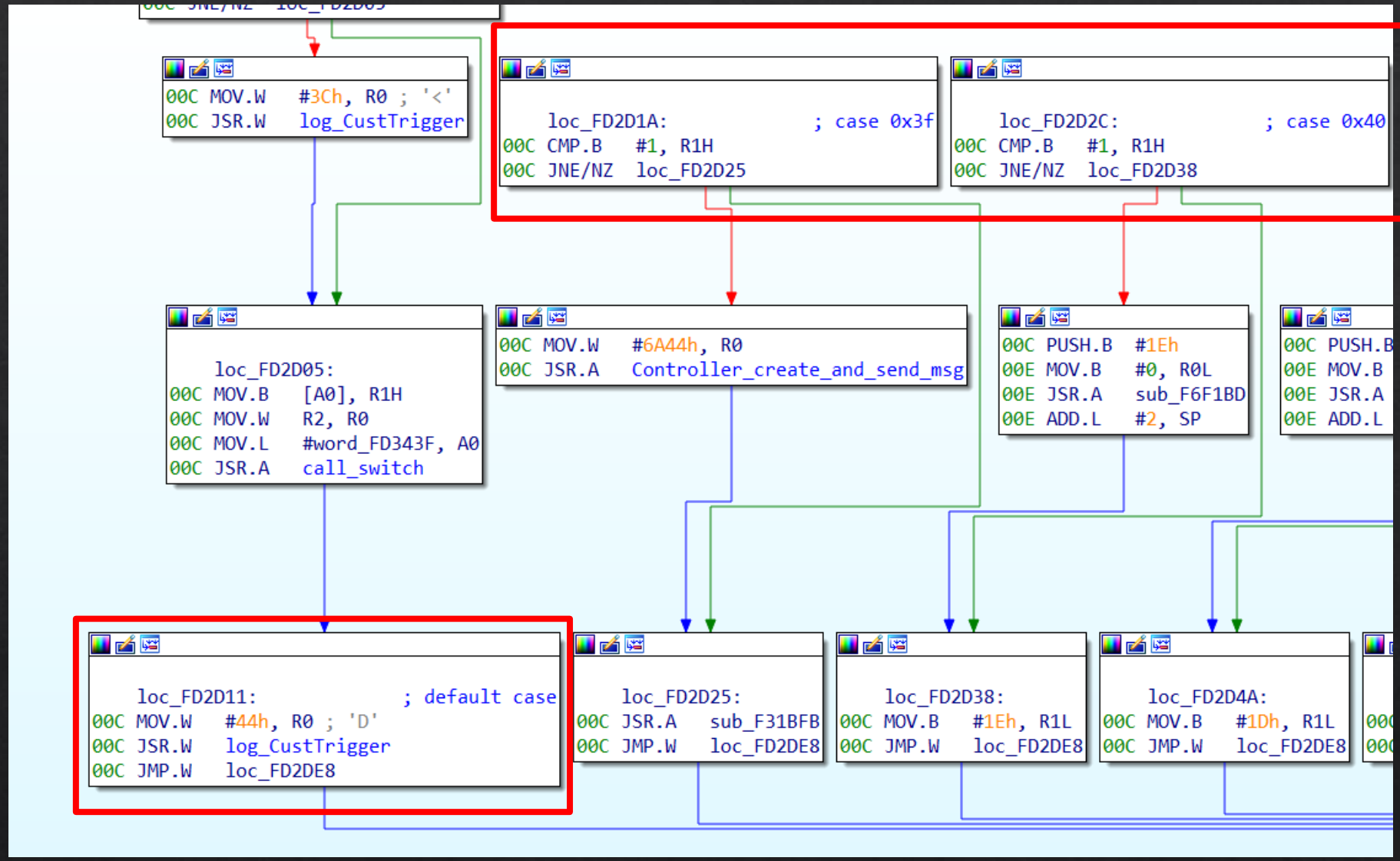


```
def process_switch_table(table_ea):
    ea = table_ea
    n_entries = get_wide_word(ea)
    print "n_entries: " + hex(n_entries)
    del_items(ea, DELIT_SIMPLE, 2 + n_entries*6 + 4) #unmark in case of crap
    create_word(ea)
    ea +=2
    for i in xrange(0, n_entries):
        create_word(ea)
        case_val = get_wide_word(ea)
        offset = get_wide_dword(ea+2)
        set_cmt(offset, "case " + hex(case_val), False)
        ida_offset.op_offset(ea+2, 0, idc.REF_OFF32)
        create_insn(offset) # MakeCode
        ea += 6
    offset = get_wide_dword(ea)
    set_cmt(offset, "default case", False)
    ida_offset.op_offset(ea, 0, idc.REF_OFF32)
    create_insn(offset)
    pass
```

```
.WORD 0Ch               ; DATA XREF: CustTrigger_3c+1A↑o
                        ; <- entry count
.WORD 3Fh
.LWORD loc_FD2D1A
.WORD 40h
.LWORD loc_FD2D2C
.WORD 41h
.LWORD loc_FD2D3E
.WORD 42h
.LWORD loc_FD2D50
.WORD 46h
.LWORD loc_FD2D62
.WORD 15Ch
.LWORD loc_FD2DD9
.WORD 1E3h
.LWORD loc_FD2D73
.WORD 1E4h
.LWORD loc_FD2D84
.WORD 1E5h
.LWORD loc_FD2D95
.WORD 1E6h
.LWORD loc_FD2DA6
.WORD 1E7h
.LWORD loc_FD2DC8
.WORD 5FAh
.LWORD loc_FD2DB7
.LWORD loc_FD2D11       ; <- default case
```

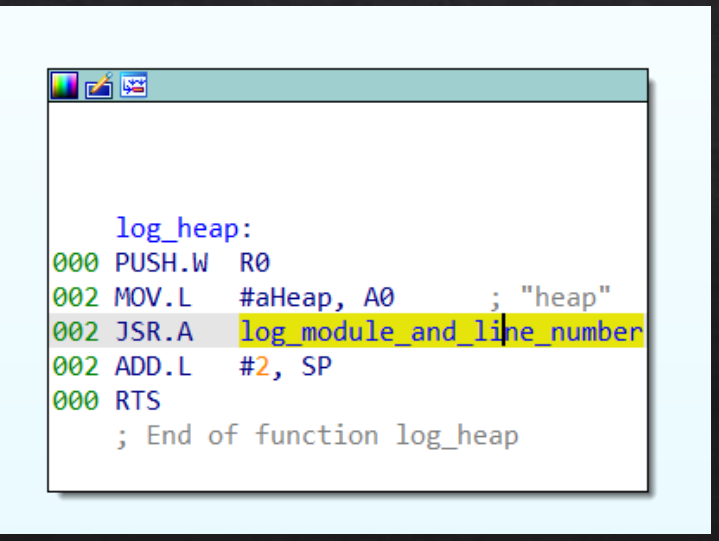
1. Read number of entries in the table
2. Undefine everything (prevent conflicts)
3. Create words (each entry number) and offsets (each entry address)
4. Create comment “case xxx” in the basic block of the case
5. Mark the relevant basic blocks as code

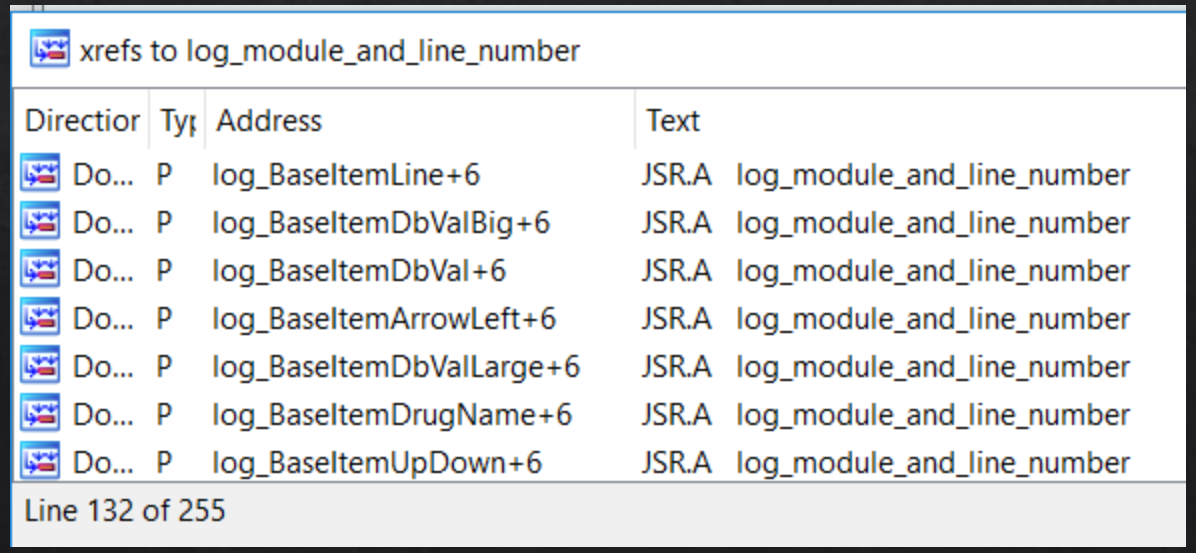


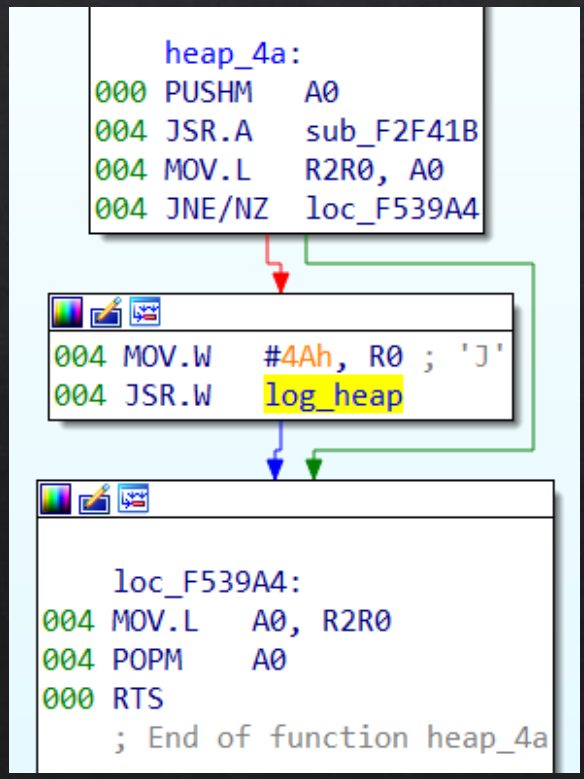

00C MOV.W #3Ch, R0 ; '<'
00C JSR.W log_CustTrigger
loc_FD2D1A: ; case 0x3f
00C CMP.B #1, R1H
00C JNE/NZ loc_FD2D25
loc_FD2D2C: ; case 0x40
00C CMP.B #1, R1H
00C JNE/NZ loc_FD2D38
loc_FD2D05:
00C MOV.B [A0], R1H
00C MOV.W R2, R0
00C MOV.L #word_FD343F, A0
00C JSR.A call_switch
00C MOV.W #6A44h, R0
00C JSR.A Controller_create_and_send_msg
00C PUSH.B #1Eh
00E MOV.B #0, R0L
00E JSR.A sub_F6F1BD
00E ADD.L #2, SP
00C PUSH.B
00E MOV.B
00E JSR.A
00E ADD.L
loc_FD2D11: ; default case
00C MOV.W #44h, R0 ; 'D'
00C JSR.W log_CustTrigger
00C JMP.W loc_FD2DE8
loc_FD2D25:
00C JSR.A sub_F31BFB
00C JMP.W loc_FD2DE8
loc_FD2D38:
00C MOV.B #1Eh, R1L
00C JMP.W loc_FD2DE8
loc_FD2D4A:
00C MOV.B #1Dh, R1L
00C JMP.W loc_FD2DE8

Logging function

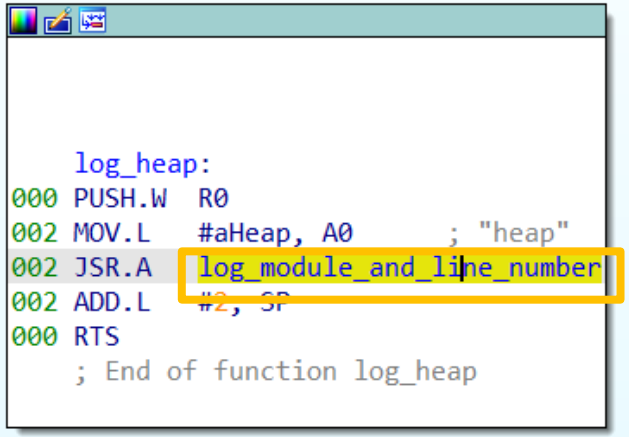
- Generic log function (logs module name + line number)
- Extended to provide 255 functions, one for each internal module
- Automate renaming log functions to add context when reversing



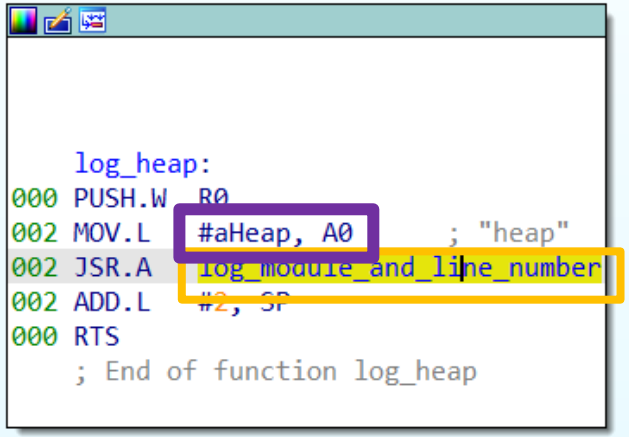




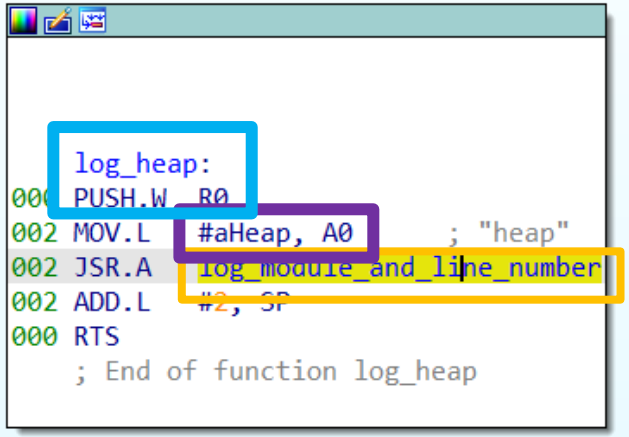
```
def list_modules(bRename= bRename,  bGenerateJson= _bGenerateJson):
    xrefs = XrefsTo(log_function_address)
    modules = []
    for x in xrefs:
        addr = x.frm -3
        addr = get_prev_ins(addr)

        #print addr
        ins = DecodeInstruction(addr)
        if ins is None:
            print "ERR"
            continue
        module_ea = ins.Op1.value
        module_name = get_string(module_ea)
        module_start = get_func_attr(addr, 0)


        if bRename:
            cur_name = get_func_name(module_start)
            if "sub_" in cur_name[:4]:
                set_name(module_start, "log_" + module_name)

        entry = {
            "name": module_name,
            "start": module_start,
            "end": last_func_in_modules
        }
        modules.append(entry)


    for i in xrange(0, len(modules) - 1):
       modules[i]["end"] =  modules[i+1]["start"] - 1 # assume they're back to back
    return modules
```



```
def list_modules(bRename= bRename,  bGenerateJson= _bGenerateJson):
    xrefs = XrefsTo(log_function_address)
    modules = []
    for x in xrefs:
        addr = x.frm -3
        addr = get_prev_ins(addr)

        #print addr
        ins = DecodeInstruction(addr)
        if ins is None:
            print "ERR"
            continue
        module_ea = ins.Op1.value
        module_name = get_string(module_ea)
        module_start = get_func_attr(addr, 0)


        if bRename:
            cur_name = get_func_name(module_start)
            if "sub_" in cur_name[:4]:
                set_name(module_start, "log_" + module_name)

        entry = {
            "name": module_name,
            "start": module_start,
            "end": last_func_in_modules
        }
        modules.append(entry)


    for i in xrange(0, len(modules) - 1):
       modules[i]["end"] =  modules[i+1]["start"] - 1 # assume they're back to back
    return modules
```

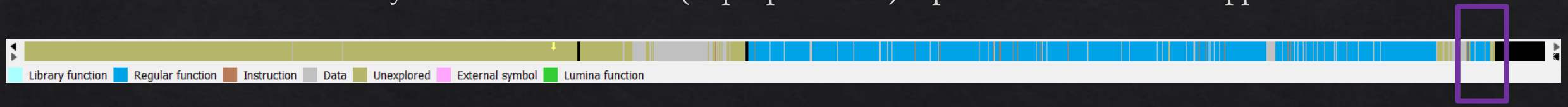


```
def list_modules(bRename= bRename,  bGenerateJson= _bGenerateJson):
    xrefs = XrefsTo(log_function_address)
    modules = []
    for x in xrefs:
        addr = x.frm -3
        addr = get_prev_ins(addr)

        #print addr
        ins = DecodeInstruction(addr)
        if ins is None:
            print "ERR"
            continue
        module_ea = ins.Op1.value
        module_name = get_string(module_ea)
        module_start = get_func_attr(addr, 0)

        if bRename:
            cur_name = get_func_name(module_start)
            if "sub_" in cur_name[:4]:
                set_name(module_start, "log_" + module_name)

        entry = {
            "name": module_name,
            "start": module_start,
            "end": last_func_in_modules
        }
        modules.append(entry)


    for i in xrange(0, len(modules) - 1):
       modules[i]["end"] =  modules[i+1]["start"] - 1 # assume they're back to back
    return modules
```



Code Layout

- There is "locality" in the code
  - Functions belonging to the same module are located next to each other
  - Modules seems to start with their own "log" functions
    - It is possible to identify all the functions belonging to the same module by looking at their address
    - Helpful context when reversing an unknown function
  - Could be fun to make a map, which modules calls into which module, etc.
- There's a gap in the code region
  - We can identify a "bootloader" code (in purple below) separate from the main application



Lessons learned – Part I

- Don’t be scared! Assembly languages tend to resemble each other
- Google for all the documentation and keywords (FDA finding! and more…)
- Know the limits of your tools & bring in your own knowledge
- IDAPython to automate tedious work (comments, structure, …)
- Compiler can be funky
  - Bad: weird optimizations and irregular patterns
  - Good: code locality
- Firmware load address
  - Usually a round number; maybe in datasheet
  - Find interrupt vectors (address) at the beginning/end of the code blob

EMBEDDED CONSIDERATIONS

Peripherals of Interest

- Mostly interesting in how the firmware communicates with the rest of the world:
  - UART
  - CAN bus
  - Tons of I/O pins
  - A/D and D/A converters

    → useful for physical processes
- Two ways to find them
  - Interrupt handlers
  - SFR

Peripheral Function	I/O Port	123 I/O pins and 1 input pin
	Multifunction Timer	Timer A: 16 bits x 5 channels, Timer B: 16 bits x 6 channels Three-phase motor control circuit
	Intelligent I/O	Time measurement function: 16 bits x 12 channels Waveform generating function: 16 bits x 28 channels Communication function (Clock synchronous serial I/O, Clock asynchronous se rial I/O, HDLC data processing, Clock synchronous variable length serial I/C IEBus(1), 8-bit or 16-bit Clock synchronous serial I/O)
	Serial I/O	5 Channels Clock synchronous serial I/O, Clock asynchronous serial I/O, IEBus(1), I2C bus(
	CAN Module	1 channel Supporting CAN 2.0B specification
	A/D Converter	10-bit A/D converter: 2 circuit, 34 channels
	D/A Converter	8 bits x 2 channels
	DMAC	4 channels
	DMAC II	Can be activated by all peripheral function interrupt sources Immediate transfer, Calculation transfer and Chain transfer functions
	DRAM	CAS before RAS refresh, Self-reflesh, EDO, EP
	CRC Calculation Circuit	CRC-CCITT
	X/Y Converter	16 bits x 16 bits
	Watchdog Timer	15 bits x 1 channel (with prescaler)
	Interrupt	42 internal and 8 external sources, 5 software sources, Interrupt priority level: 7
	Clock Generation Circuit	4 circuits Main clock oscillation circuit(*), Sub clock oscillation circuit(*), On-chip oscillato PLL frequency synthesizer (*)Equipped with a built-in feedback resistor. Ceramic resonator or crystal oscilla tor must be connected externally
	Oscillation Stop Detect Function	Main clock oscillation stop detect function

Interrupt Vector Table

- Fixed Vector Table
  - For reset, watchdog, etc.
- Relocatable Vector Table
  - Address set in INTB register
  - Triggered for UART, Timer, CAN, …
- Entrypoints for important features!

10.5.1 Fixed Vector Tables

The fixed vector tables are allocated addresses FFFFDC16 to FFFFFF16. Table 10.1 lists the fixed vector tables. Refer to **25.2 Functions to Prevent Flash Memory from Rewriting** for fixed vectors of the flash memory.

Table 10.1 Fixed Vector Table

Interrupt Generated by	Vector Addresses Address (L) to Address (H)	Remarks	Reference
Undefined Instruction	FFFFDC16 to FFFFDF16		M32C/80 series software manual
Overflow	FFFFE016 to FFFFE316		
BRK Instruction	FFFFE416 to FFFFE716	If the content of address FFFFE716 is FF16, the program is executed from the address stored into software interrupt number 0 in the relocatable vector table	
Address Match	FFFFE816 to FFFFEB16		
-	FFFFEC16 to FFFFEF16	Reserved space	
Watchdog Timer	FFFFF016 to FFFFF316	These addresses are used for the watchdog timer interrupt and the oscillation stop detect interrupt	Clock oscillation circuit, Watchdog timer
-	FFFFF416 to FFFFF716	Reserved space	
$\overline{\text{NMI}}$	FFFFF816 to FFFFFB16		
Reset	FFFFFC16 to FFFFFF16		Reset

10.5.2 Relocatable Vector Tables

The relocatable vector tables occupy 256 bytes from the starting address set in the INTB register. Table 10.2 lists the relocatable vector tables.

Set an even address as the starting address of the vector table set in the INTB register to increase interrupt sequence execution rate.

Table 10.2 Relocatable Vector Tables

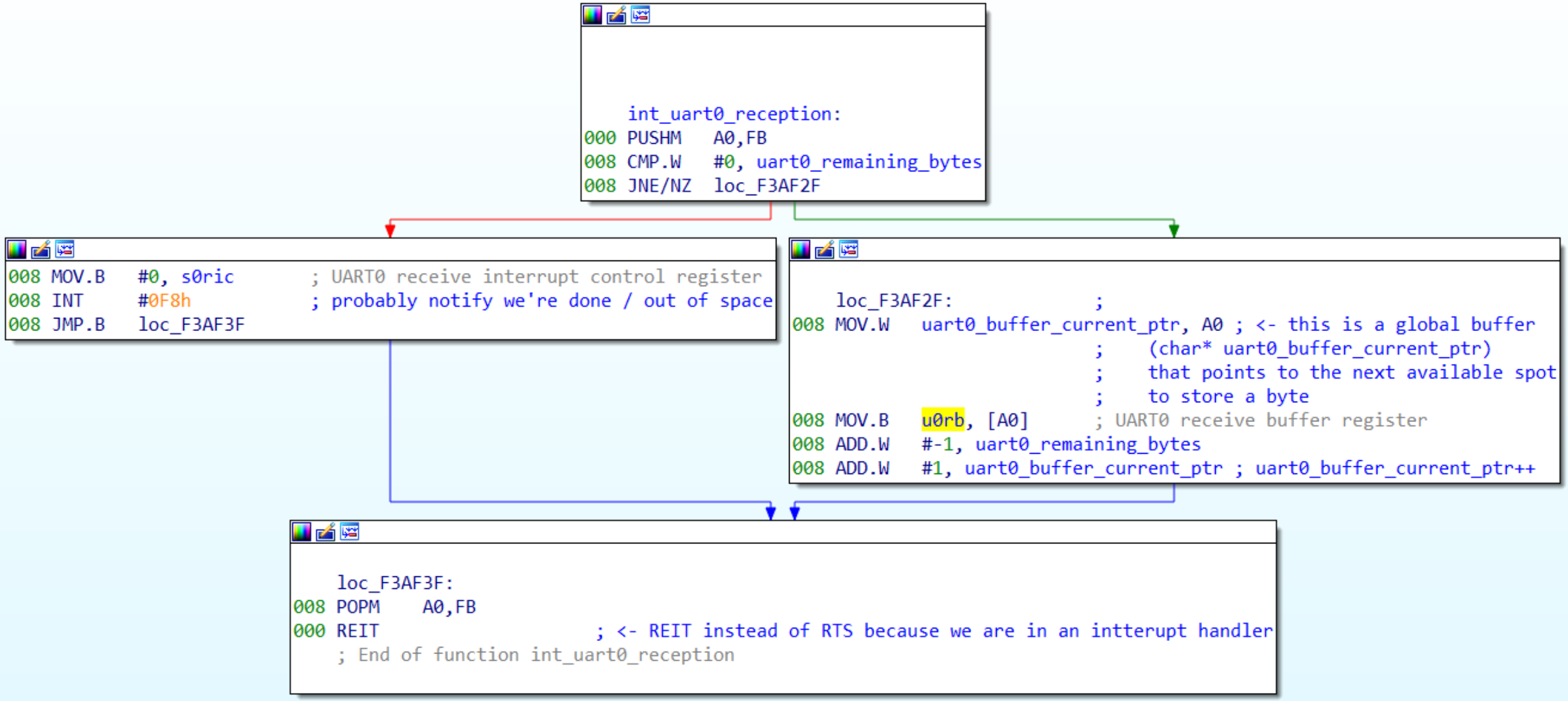
Interrupt Generated by	Vector Table Address Address(L) to Address(H)[1]	Software Interrupt Number	Reference
BRK Instruction[2]	+0 to +3 (000016 to 000316)	0	M32C/80 Series
Reserved Space	+4 to +27 (000416 to 001B16)	1 to 6	Software Manual
A/D1	+28 to +31 (001C16 to 001F16)	7	A/D Converter
DMA0	+32 to +35 (002016 to 002316)	8	DMAC
DMA1	+36 to +39 (002416 to 002716)	9	
DMA2	+40 to +43 (002816 to 002B16)	10	
DMA3	+44 to +47 (002C16 to 002F16)	11	
Timer A0	+48 to +51 (003016 to 003316)	12	Timer A
Timer A1	+52 to +55 (003416 to 003716)	13	
Timer A2	+56 to +59 (003816 to 003B16)	14	
Timer A3	+60 to +63 (003C16 to 003F16)	15	
Timer A4	+64 to +67 (004016 to 004316)	16	
UART0 Transmission, NACK[3]	+68 to +71 (004416 to 004716)	17	Serial I/O
UART0 Reception, ACK[3]	+72 to +75 (004816 to 004B16)	18	
UART1 Transmission, NACK[3]	+76 to +79 (004C16 to 004F16)	19	
UART1 Reception, ACK[3]	+80 to +83 (005016 to 005316)	20	
Timer B0	+84 to +87 (005416 to 005716)	21	Timer B
Timer B1	+88 to +91 (005816 to 005B16)	22	
Timer B2	+92 to +95 (005C16 to 005F16)	23	
Timer B3	+96 to +99 (006016 to 006316)	24	
Timer B4	+100 to +103 (006416 to 006716)	25	
$\overline{INT5}$	+104 to +107 (006816 to 006B16)	26	Interrupt
$\overline{INT4}$	+108 to +111 (006C16 to 006F16)	27	
$\overline{INT3}$	+112 to +115 (007016 to 007316)	28	
$\overline{INT2}$	+116 to +119 (007416 to 007716)	29	
$\overline{INT1}$	+120 to +123 (007816 to 007B16)	30	
$\overline{INT0}$	+124 to +127 (007C16 to 007F16)	31	
Timer B5	+128 to +131 (008016 to 008316)	32	Timer B
UART2 Transmission, NACK[3]	+132 to +135 (008416 to 008716)	33	Serial I/O
UART2 Reception, ACK[3]	+136 to +139 (008816 to 008B16)	34	
UART3 Transmission, NACK[3]	+140 to +143 (008C16 to 008F16)	35	
UART3 Reception, ACK[3]	+144 to +147 (009016 to 009316)	36	
UART4 Transmission, NACK[3]	+148 to +151 (009416 to 009716)	37	
UART4 Reception, ACK[3]	+152 to +155 (009816 to 009B16)	38	

Table 10.2 Relocatable Vector Tables (Continued)

Interrupt Generated by	Vector Table Address Address(L)to Address(H)[1]	Software Interrupt Number	Reference
Bus Conflict Detect, Start Condition Detect, Stop Condition Detect, (UART2)[3], Fault Error[4]	+156 to +159 (009C16 to 009F16)	39	Serial I/O
Bus Conflict Detect, Start Condition Detect, Stop Condition Detect, (UART3/UART0)[5], Fault Error[4]	+160 to +163 (00A016 to 00A316)	40	
Bus Conflict Detect, Start Condition Select, Stop Condition Detect, (UART4/UART1)[5], Fault Error[4]	+164 to +167 (00A416 to 00A716)	41	
A/D0	+168 to +171 (00A816 to 00AB16)	42	A/D Converter
Key Input	+172 to +175 (00AC16 to 00AF16)	43	Interrupts
Intelligent I/O Interrupt 0	+176 to +179 (00B016 to 00B316)	44	Intelligent I/O
Intelligent I/O Interrupt 1	+180 to +183 (00B416 to 00B716)	45	CAN
Intelligent I/O Interrupt 2	+184 to +187 (00B816 to 00BB16)	46	
Intelligent I/O Interrupt 3	+188 to +191 (00BC16 to 00BF16)	47	
Intelligent I/O Interrupt 4	+192 to +195 (00C016 to 00C316)	48	
Intelligent I/O Interrupt 5	+196 to +199 (00C416 to 00C716)	49	
Intelligent I/O Interrupt 6	+200 to +203(00C816 to 00CB16)	50	
Intelligent I/O Interrupt 7	+204 to +207(00CC16 to 00CF16)	51	
Intelligent I/O Interrupt 8	+208 to +211(00D016 to 00D316)	52	
Intelligent I/O Interrupt 9, CAN 0	+212 to +215 (00D416 to 00D716)	53	
Intelligent I/O Interrupt 10, CAN 1	+216 to +219 (00D816 to 00DB16)	54	
Reserved Space	+220 to +227 (00DC16 to 00E316)	55 to 56	—
Intelligent I/O Interrupt 11, CAN 2	+228 to +231 (00E416 to 00E716)	57	Intelligent I/O CAN
Reserved Space	+232 to +255 (00E816 to 00FF16)	58 to 62	—
INT Instruction[2]	+0 to +3 (000016 to 000316) to +252 to +255 (00FC16 to 00FF16)	0 to 63	Interrupts

NOTES:

```
call_table .LWORD 0FFFFFFFFh
                        ; DATA XREF: app_from_flash_init+C↑o
                        ; ROM:00F306DD↑o
                        ; see page 108 of hardware manual
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD int_DMA0         ; sw int 8
    .LWORD int_DMA1         ; sw int 9
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD int_uart0_transmission ; sw int 17
    .LWORD int_uart0_reception ; sw int 18
    .LWORD int_uart1_transmission_and_i2cNack ; sw
    .LWORD int_uart1_reception_andi2c_ack ; sw int
    .LWORD int_timer_B0     ; sw int 21
    .LWORD int_timer_B1     ; sw int 22
    .LWORD int_timer_B2     ; sw int 23
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
    .LWORD 0FFFFFFFFh
ROM:00FEFD94    .LWORD int_uart4_transmission ; sw int 37
ROM:00FEFD98    .LWORD int_uart4_reception ; sw int 38
ROM:00FEFD9C    .LWORD 0FFFFFFFFh
ROM:00FEFDA0    .LWORD 0FFFFFFFFh
ROM:00FEFDA4    .LWORD 0FFFFFFFFh
ROM:00FEFDA8    .LWORD 0FFFFFFFFh
ROM:00FEFDAC    .LWORD 0FFFFFFFFh
ROM:00FEFDB0    .LWORD 0FFFFFFFFh
ROM:00FEFDB4    .LWORD 0FFFFFFFFh
ROM:00FEFDB8    .LWORD 0FFFFFFFFh
ROM:00FEFDBC    .LWORD 0FFFFFFFFh
ROM:00FEFDC0    .LWORD 0FFFFFFFFh
ROM:00FEFDC4    .LWORD 0FFFFFFFFh
ROM:00FEFDC8    .LWORD 0FFFFFFFFh
ROM:00FEFDCC    .LWORD 0FFFFFFFFh
ROM:00FEFDD0    .LWORD 0FFFFFFFFh
ROM:00FEFDD4    .LWORD int_CAN_53       ; INT 53
ROM:00FEFDD8    .LWORD 0FFFFFFFFh
ROM:00FEFDDC    .LWORD 0FFFFFFFFh
ROM:00FEFDE0    .LWORD 0FFFFFFFFh
ROM:00FEFDE4    .LWORD 0FFFFFFFFh
ROM:00FEFDE8    .LWORD 0FFFFFFFFh
ROM:00FEFDEC    .LWORD 0FFFFFFFFh
ROM:00FEFDF0    .LWORD 0FFFFFFFFh
ROM:00FEFDF4    .LWORD 0FFFFFFFFh
ROM:00FEFDF8    .LWORD int_62           ; INT 62, called by int_DMA0, int_uart0_transmission, int_uart0_reception
ROM:00FEFDFC    .LWORD 0FFFFFFFFh
```


int_uart0_reception:
000 PUSHM A0,FB
008 CMP.W #0, uart0_remaining_bytes
008 JNE/NZ loc_F3AF2F
008 MOV.B #0, s0ric ; UART0 receive interrupt control register
008 INT #0F8h ; probably notify we're done / out of space
008 JMP.B loc_F3AF3F
loc_F3AF2F: ;
008 MOV.W uart0_buffer_current_ptr, A0 ; <- this is a global buffer
; (char* uart0_buffer_current_ptr)
; that points to the next available spot
; to store a byte
008 MOV.B u0rb, [A0] ; UART0 receive buffer register
008 ADD.W #-1, uart0_remaining_bytes
008 ADD.W #1, uart0_buffer_current_ptr ; uart0_buffer_current_ptr++
loc_F3AF3F:
008 POPM A0,FB
000 REIT ; <- REIT instead of RTS because we are in an intterupt handler
; End of function int_uart0_reception

SFR

- Special Function Registers
  - Lots of them!
  - Some unused or undocumented
- Special "memory" you read/write to
  - MOV, BSET, BCLR
  - "Memory" range 0x0000 to 0x0400
- Multiple purposes:
  - Trigger an action
  - Change an internal setting
  - Read/Write data to peripherals
- IDA knows the M32C/80 SFR
  - Not 100% the same as for the M32C/83
  - Good start though

4. Special Function Registers (SFR)

Address	Register	Symbol	Value after RESET
000016			
000116			
000216			
000316			
000416	Processor Mode Register 0(1)	PM0	1000 00002 (CNVss pin ="L") 0000 00112 (CNVss pin ="H")
000516	Processor Mode Register 1	PM1	0X00 00002
000616	System Clock Control Register 0	CM0	0000 X0002
000716	System Clock Control Register 1	CM1	0010 00002
000816	Wait Control Register(2)	WCR	1111 11112
000916	Address Match Interrupt Enable Register	AIER	XXXX 00002
000A16	Protect Register	PRCR	XXXX 00002
000B16	External Data Bus Width Control Register(2)	DS	XXXX 10002 (BYTE pin ="L") XXXX 00002 (BYTE pin ="H")
000C16	Main Clock Division Register	MCD	XXX0 10002
000D16	Oscillation Stop Detection Register	CM2	0016
000E16	Watchdog Timer Start Register	WDTS	XX16
000F16	Watchdog Timer Control Register	WDC	000X XXXX2
001016 001116 001216	Address Match Interrupt Register 0	RMAD0	00 00 0016
001316			
001416 001516 001616	Address Match Interrupt Register 1	RMAD1	00 00 0016
001716	VDC Control Register for PLL	PLV	XXXX XX012
001816 001916 001A16	Address Match Interrupt Register 2	RMAD2	00 00 0016
001B16	VDC Control Register 0	VDC0	0016
001C16 001D16	Address Match Interrupt Register 3	RMAD3	00 00 0016

```
SFR:0000     ; ===========================================================================
SFR:0000
SFR:0000     ; Segment type: Internal processor memory & SFR
SFR:0000                     .segment SFR
SFR:0000     word_0          .equ 0                  ; DATA XREF: ROM_DATA:00F0C3AE↓o
SFR:0000                                             ; ROM
SFR:0002     unk_2           .equ 2                  ; DAT
SFR:0002                                             ; ROM
SFR:0003     unk_3           .equ 3                  ; DAT
SFR:0003                                             ; Fla
SFR:0004     pm0             .equ 4                  ; Pro
SFR:0004      pm00           .equ 0 ; Processor mode bit
SFR:0004      pm01           .equ 1 ; Processor mode bit
SFR:0004      pm02           .equ 2 ; R/W mode select bit
SFR:0004      pm03           .equ 3 ; Software reset bit
SFR:0004      pm04           .equ 4 ; Multiplexed bus spac
SFR:0004      pm05           .equ 5 ; Multiplexed bus spac
SFR:0004      pm06           .equ 6
SFR:0004      pm07           .equ 7 ; BCLK output function
SFR:0004
SFR:0005     pm1             .equ 5                  ; DATA XREF: CAN_init_cpu_controller+10↓w
SFR:0005                                             ; reset_interrupt+F↓w ...
SFR:0005      pm10           .equ 0 ; External memory area mode bit ; Processor mode register 1
SFR:0005      pm11           .equ 1 ; External memory area mode bit
SFR:0005      pm12           .equ 2 ; Internal memory wait bit
SFR:0005      pm13           .equ 3 ; SFR wait bit
SFR:0005      pm14           .equ 4 ; ALE pin select bit
SFR:0005      pm15           .equ 5 ; ALE pin select bit
SFR:0005
```

xrefs to pm1

Direction	Type	Address	Text
Do...	w	CAN_init_cpu_controller+10	BSET pm13, pm1; Processor mode register 1
Do...	w	reset_interrupt+F	MOV.B #2, pm1; Processor mode register 1
Do...	w	rom_int_62+F	MOV.B #2, pm1; Processor mode register 1
Do...	w	sub_FFA4EB+10	BSET pm13, pm1; Processor mode register 1

Line 2 of 4

OK | Cancel | Search | Help

```
SFR:0000      ; ===========================================
SFR:0000
SFR:0000      ; Segment type: Internal processor memory & SFR
SFR:0000                      .segment SFR
SFR:0000      word_0          .equ 0                      ; DATA X
SFR:0000                                                  ; ROM
SFR:0002      unk_2           .equ 2                      ; DAT
SFR:0002                                                  ; ROM
SFR:0003      unk_3           .equ 3                      ; DAT
SFR:0003                                                  ; Fla
SFR:0004      pm0             .equ 4                      ; Pro
SFR:0004       pm00           .equ 0 ; Processor mode bit
SFR:0004       pm01           .equ 1 ; Processor mode bit
SFR:0004       pm02           .equ 2 ; R/W mode select bit
SFR:0004       pm03           .equ 3 ; Software reset bit
SFR:0004       pm04           .equ 4 ; Multiplexed bus spac
SFR:0004       pm05           .equ 5 ; Multiplexed bus spac
SFR:0004       pm06           .equ 6
SFR:0004       pm07           .equ 7 ; BCLK output function
SFR:0004
SFR:0005      pm1             .equ 5                      ; DATA X
SFR:0005                                                  ; reset_
SFR:0005       pm10           .equ 0 ; External memory area mo
SFR:0005       pm11           .equ 1 ; External memory area mo
SFR:0005       pm12           .equ 2 ; Internal memory wait bi
SFR:0005       pm13           .equ 3 ; SFR wait bit
SFR:0005       pm14           .equ 4 ; ALE pin select bit
SFR:0005       pm15           .equ 5 ; ALE pin select bit
SFR:0005
```


Processor Mode Register 1(1)

b7	b6	b5	b4	b3	b2	b1	b0
0	X						

Symbol	Address	After Reset
PM1	000516	0X00 00002

Bit Symbol	Bit Name	Function	RW
PM10	External Memory Space Mode Bit(2, 6)	b1 b0 0 0 : Mode 0 (A20 to $\overline{A23}$ for P44 to P47) 0 1 : Mode 1 (A20 for P44, $\overline{CS2}$ to $\overline{CS0}$ for P45 to P47) 1 0 : Mode 2 (A20, A21 for P44, P45, $\overline{CS1}$, $\overline{CS0}$ for P46, P47) 1 1 : Mode 3(3) ($\overline{CS3}$ to $\overline{CS0}$ for P44 to P47)	RW
PM11			RW
PM12	Internal Memory Wait Bit	0 : No wait state 1 : Wait state	RW
PM13	SFR Area Wait Bit 0	0 : 1 wait state 1 : 2 wait states(4)	RW
PM14	ALE Pin Select Bit(2, 6)	b5 b4 0 0 : No ALE 0 1 : P53/BCLK(5) 1 0 : P56/$\overline{RAS}$ 1 1 : P54/$\overline{HLDA}$	RW
PM15			RW
$\overline{(b6)}$	Nothing is assigned. When read, its content is indeterminate.		—
$\overline{(b7)}$	Reserved Bit	Set to "0"	RW

NOTES:
1. Rewrite the PM1 register after the PRC1 bit in the PRCR register is set to "1" (write enable).
2. The PM10 and PM11 bits are available in memory expansion mode or microprocessor mode.
3. The DRAMC is not available when the PM11 and PM10 bits are set to "112" (mode 3).
4. Set the PM13 bit to "1" (2 wait states) to access CAN-associated registers (addresses 01E016 to 024516).
5. Set the CM01 and CM00 bits in the CM0 register to "002" (I/O port P53) when the PM15 and PM14 bits are set to "012" (P53/BCLK select).
6. M32C/83T cannot be used in memory expansion mode and microprocessor mode.

Figure 6.2 PM1 Register

SFR

- Why should you care?
  - XREF to find fun code!
  - Initialization code for UART/Timers/…
    - Find relevant global data structures and label them
    - Can help find higher-level functions
  - Find who writes to UART, toggle a debug pin, etc.
- Other embedded chip will work the same way
  - Explains “weird” memory access
  - Take a guess at register from context and see what other code uses it

HUNTING FOR INTERESTING CODE BLOBS

Interesting Code Blobs

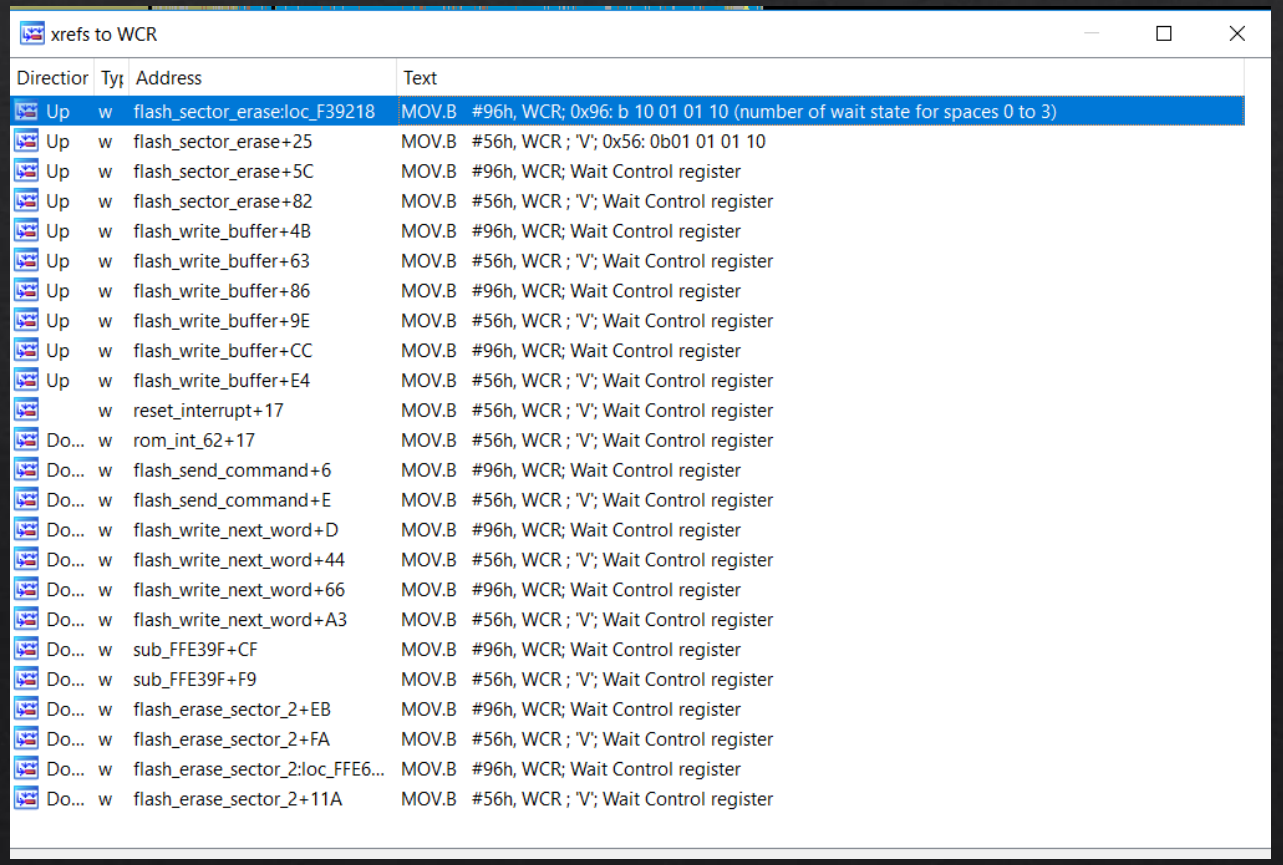
- Original objectives
  - Locate I/O code (UART+CAN)

    → vuln research
  - Locate Flash code

    → can we reflash the firmware code?
  - Bonus: understand physical processes of the devices
- How?
  - UART/CAN → interrupt handler → buffering code → message handler
  - Physical process → PWM (Pulse Width Modulation), ADC/DAC → RE of business logic
  - Flash?

Finding Flash Modifying Code

- WCR (Wait Control Register) register mentioned in datasheet on the memory map page
  - Configure wait time before sending data on the bus (to the flash)
  - Cool Xrefs ☺



Direction	Type	Address	Text
Up	w	flash_sector_erase:loc_F39218	MOV.B #96h, WCR; 0x96: b 10 01 01 10 (number of wait state for spaces 0 to 3)
Up	w	flash_sector_erase+25	MOV.B #56h, WCR ; 'V'; 0x56: 0b01 01 01 10
Up	w	flash_sector_erase+5C	MOV.B #96h, WCR; Wait Control register
Up	w	flash_sector_erase+82	MOV.B #56h, WCR ; 'V'; Wait Control register
Up	w	flash_write_buffer+4B	MOV.B #96h, WCR; Wait Control register
Up	w	flash_write_buffer+63	MOV.B #56h, WCR ; 'V'; Wait Control register
Up	w	flash_write_buffer+86	MOV.B #96h, WCR; Wait Control register
Up	w	flash_write_buffer+9E	MOV.B #56h, WCR ; 'V'; Wait Control register
Up	w	flash_write_buffer+CC	MOV.B #96h, WCR; Wait Control register
Up	w	flash_write_buffer+E4	MOV.B #56h, WCR ; 'V'; Wait Control register
	w	reset_interrupt+17	MOV.B #56h, WCR ; 'V'; Wait Control register
Do...	w	rom_int_62+17	MOV.B #56h, WCR ; 'V'; Wait Control register
Do...	w	flash_send_command+6	MOV.B #96h, WCR; Wait Control register
Do...	w	flash_send_command+E	MOV.B #56h, WCR ; 'V'; Wait Control register
Do...	w	flash_write_next_word+D	MOV.B #96h, WCR; Wait Control register
Do...	w	flash_write_next_word+44	MOV.B #56h, WCR ; 'V'; Wait Control register
Do...	w	flash_write_next_word+66	MOV.B #96h, WCR; Wait Control register
Do...	w	flash_write_next_word+A3	MOV.B #56h, WCR ; 'V'; Wait Control register
Do...	w	sub_FFE39F+CF	MOV.B #96h, WCR; Wait Control register
Do...	w	sub_FFE39F+F9	MOV.B #56h, WCR ; 'V'; Wait Control register
Do...	w	flash_erase_sector_2+EB	MOV.B #96h, WCR; Wait Control register
Do...	w	flash_erase_sector_2+FA	MOV.B #56h, WCR ; 'V'; Wait Control register
Do...	w	flash_erase_sector_2:loc_FFE6...	MOV.B #96h, WCR; Wait Control register
Do...	w	flash_erase_sector_2+11A	MOV.B #56h, WCR ; 'V'; Wait Control register

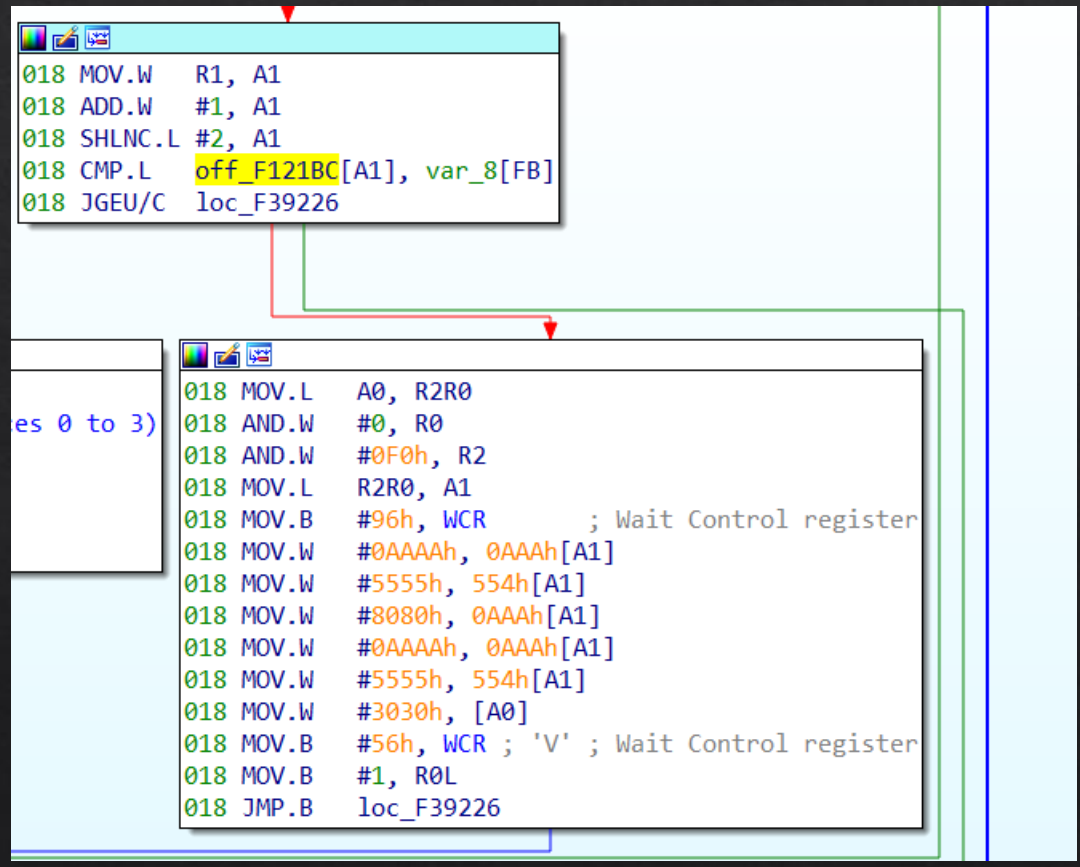
Finding Flash Modifying Code

- But… How do you send commands (erase/write) to the flash?
  - Read Flash datasheet!

Table 13. S29JL032J Command Definitions (Continued)

Command Sequence[10]		Cycles	Bus Cycles (Notes 11–14)											
			First		Second		Third		Fourth		Fifth		Sixth	
			Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Chip Erase	Word	6	555	AA	2AA	55	555	80	555	AA	2AA	55	555	10
	Byte		AAA		555		AAA		AAA		555		AAA	
Sector Erase[26]	Word	6	555	AA	2AA	55	555	80	555	AA	2AA	55	SA	30
	Byte		AAA		555		AAA		AAA		555			
Erase Suspend[23]		1	BA	B0										
Erase Resume[24]		1	BA	30										
CFI Query[25]	Word	1	55	98										
	Byte		AA											

Legend:



Finding Flash Modifying Code

- Results
  - Write/Erase functions found!
  - Can't modify the firmware from the main application ☹
  - The main application will only save "data" in a different part of the flash
- There is a second set of write/erase functions!
  - These are used by a bootloader/recovery code
  - This one can modify the whole flash ☺

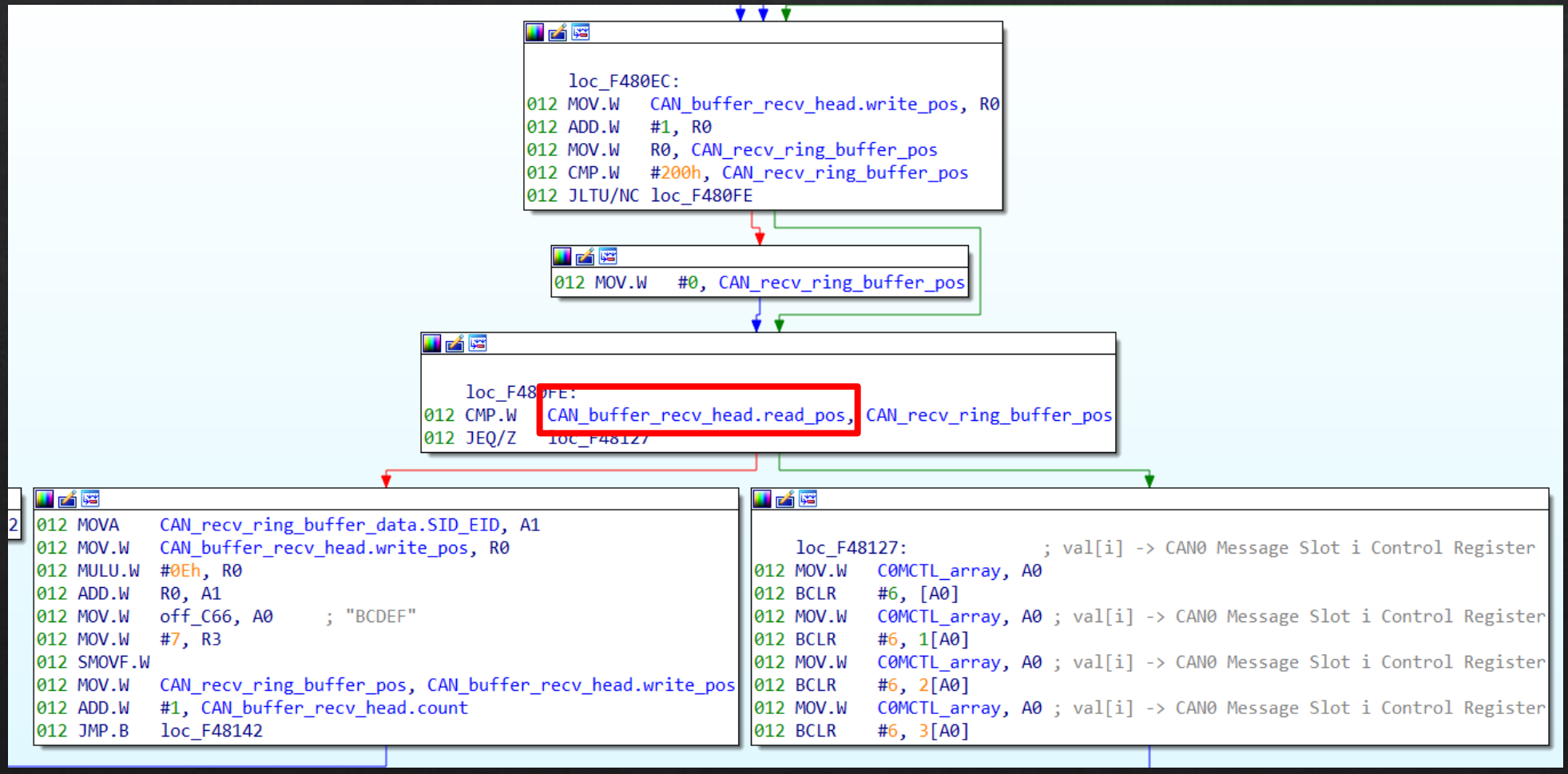
Example 2 – CAN Processing Code

1. Find the CAN interrupt

```
ROM:00FEFD00     probably_syscall_table .LWORD 0FFFFFFFFh
ROM:00FEFD00                                           ; DATA XREF: app_from_flash_init+C↑o
ROM:00FEFD00                                           ; ROM:00F306DD↑o
ROM:00FEFD00                                           ; see page 108 of hardware manual
ROM:00FEFD04                     .LWORD 0FFFFFFFFh
ROM:00FEFD08                     .LWORD 0FFFFFFFFh
ROM:00FEFD0C                     .LWORD 0FFFFFFFFh
ROM:00FEFD10                     .LWORD 0FFFFFFFFh
ROM:00FEFD14                     .LWORD 0FFFFFFFFh
ROM:00FEFD18                     .LWORD 0FFFFFFFFh
ROM:00FEFD1C                     .LWORD 0FFFFFFFFh
ROM:00FEFD20                     .LWORD int_DMA0         ; sw int 8
ROM:00FEFD24                     .LWORD int_DMA1         ; sw int 9
ROM:00FEFD28                     .LWORD 0FFFFFFFFh
ROM:00FEFD2C                     .LWORD 0FFFFFFFFh
ROM:00FEFD30                     .LWORD 0FFFFFFFFh
ROM:00FEFD34                     .LWORD 0FFFFFFFFh
ROM:00FEFD38                     .LWORD 0FFFFFFFFh
ROM:00FEFD3C                     .LWORD 0FFFFFFFFh
ROM:00FEFD40                     .LWORD 0FFFFFFFFh
ROM:00FEFD44                     .LWORD int_uart0_transmission ; sw int 17
ROM:00FEFD48                     .LWORD int_uart0_reception ; sw int 18
ROM:00FEFD4C                     .LWORD int_uart1_transmission_and_i2cNack ; sw
ROM:00FEFD50                     .LWORD int_uart1_reception_andi2c_ack ; sw int
ROM:00FEFD54                     .LWORD int_timer_B0     ; sw int 21
ROM:00FEFD58                     .LWORD int_timer_B1     ; sw int 22
ROM:00FEFD5C                     .LWORD int_timer_B2     ; sw int 23
ROM:00FEFD60                     .LWORD 0FFFFFFFFh
ROM:00FEFD64                     .LWORD 0FFFFFFFFh
ROM:00FEFD68                     .LWORD 0FFFFFFFFh
ROM:00FEFD6C                     .LWORD 0FFFFFFFFh
ROM:00FEFD70                     .LWORD 0FFFFFFFFh
ROM:00FEFD74                     .LWORD 0FFFFFFFFh
ROM:00FEFD78                     .LWORD 0FFFFFFFFh
ROM:00FEFD94                     .LWORD int_uart4_transmission ; sw int 37
ROM:00FEFD98                     .LWORD int_uart4_reception ; sw int 38
ROM:00FEFD9C                     .LWORD 0FFFFFFFFh
ROM:00FEFDA0                     .LWORD 0FFFFFFFFh
ROM:00FEFDA4                     .LWORD 0FFFFFFFFh
ROM:00FEFDA8                     .LWORD 0FFFFFFFFh
ROM:00FEFDAC                     .LWORD 0FFFFFFFFh
ROM:00FEFDB0                     .LWORD 0FFFFFFFFh
ROM:00FEFDB4                     .LWORD 0FFFFFFFFh
ROM:00FEFDB8                     .LWORD 0FFFFFFFFh
ROM:00FEFDBC                     .LWORD 0FFFFFFFFh
ROM:00FEFDC0                     .LWORD 0FFFFFFFFh
ROM:00FEFDC4                     .LWORD 0FFFFFFFFh
ROM:00FEFDC8                     .LWORD 0FFFFFFFFh
ROM:00FEFDCC                     .LWORD 0FFFFFFFFh
ROM:00FEFDD0                     .LWORD 0FFFFFFFFh
ROM:00FEFDD4                     .LWORD int_CAN_53       ; INT 53
ROM:00FEFDD8                     .LWORD 0FFFFFFFFh
ROM:00FEFDDC                     .LWORD 0FFFFFFFFh
ROM:00FEFDE0                     .LWORD 0FFFFFFFFh
ROM:00FEFDE4                     .LWORD 0FFFFFFFFh
ROM:00FEFDE8                     .LWORD 0FFFFFFFFh
ROM:00FEFDEC                     .LWORD 0FFFFFFFFh
ROM:00FEFDF0                     .LWORD 0FFFFFFFFh
ROM:00FEFDF4                     .LWORD 0FFFFFFFFh
ROM:00FEFDF8                     .LWORD int_62           ; INT 62, called by int_DMA0, int_uart0_transmission, int_uart0_reception
ROM:00FEFDFC                     .LWORD 0FFFFFFFFh
```

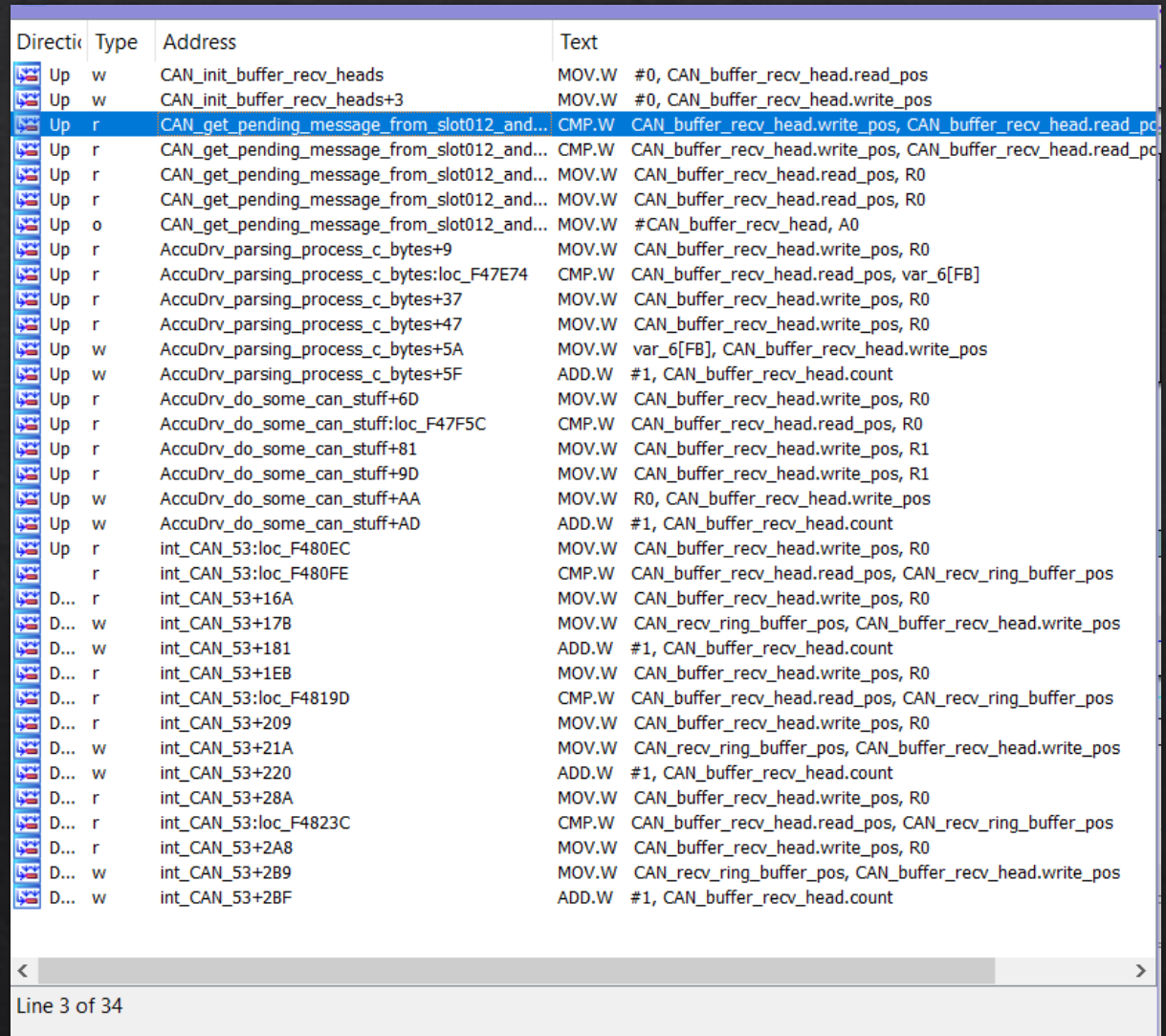
Example 2 – CAN Processing Code

2. Identify ring buffer in interrupt handler



Example 2 – CAN Processing Code

3. Follow XREFs

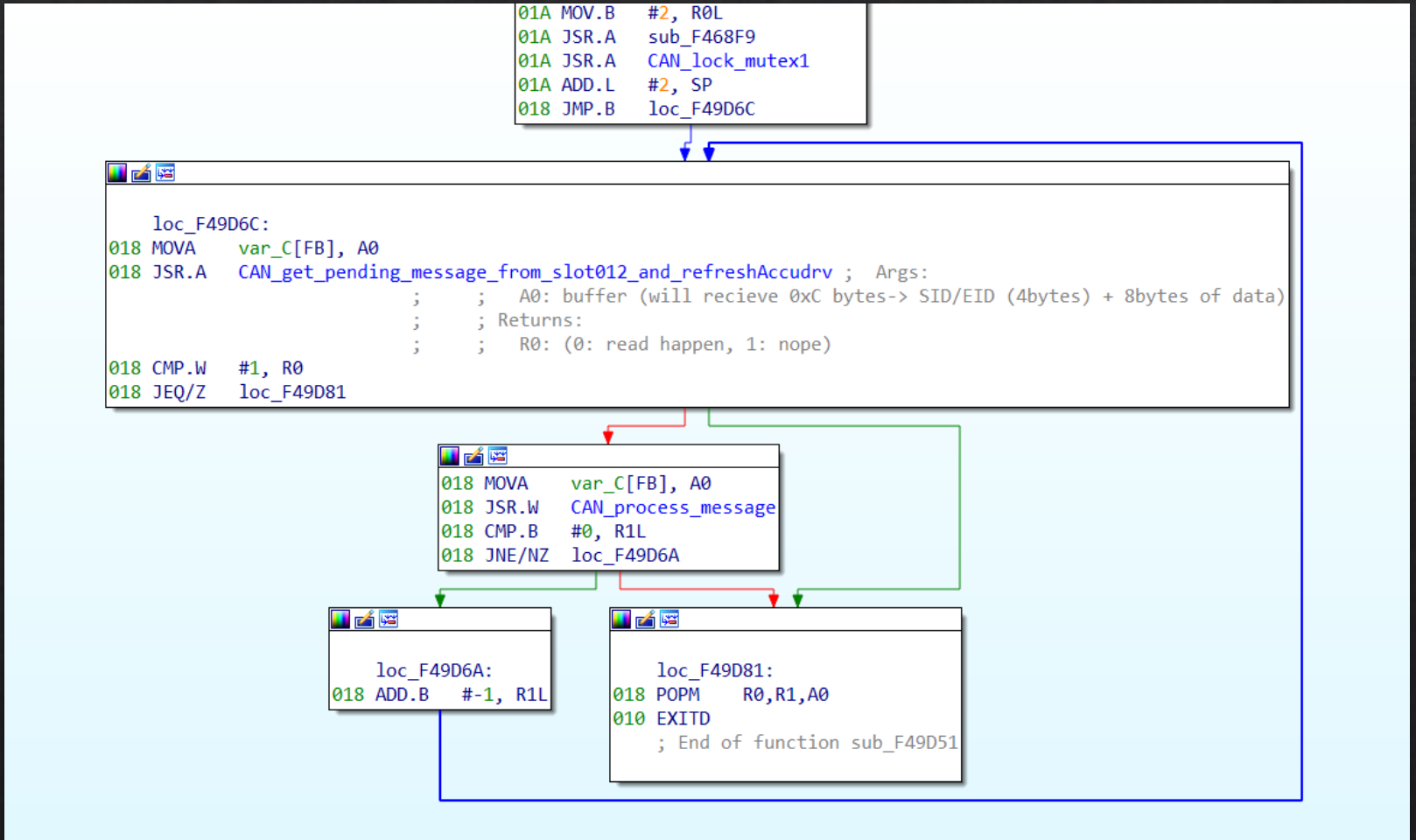


Directi	Type	Address	Text
Up	w	CAN_init_buffer_recv_heads	MOV.W #0, CAN_buffer_recv_head.read_pos
Up	w	CAN_init_buffer_recv_heads+3	MOV.W #0, CAN_buffer_recv_head.write_pos
Up	r	CAN_get_pending_message_from_slot012_and...	CMP.W CAN_buffer_recv_head.write_pos, CAN_buffer_recv_head.read_pc
Up	r	CAN_get_pending_message_from_slot012_and...	CMP.W CAN_buffer_recv_head.write_pos, CAN_buffer_recv_head.read_pc
Up	r	CAN_get_pending_message_from_slot012_and...	MOV.W CAN_buffer_recv_head.read_pos, R0
Up	r	CAN_get_pending_message_from_slot012_and...	MOV.W CAN_buffer_recv_head.read_pos, R0
Up	o	CAN_get_pending_message_from_slot012_and...	MOV.W #CAN_buffer_recv_head, A0
Up	r	AccuDrv_parsing_process_c_bytes+9	MOV.W CAN_buffer_recv_head.write_pos, R0
Up	r	AccuDrv_parsing_process_c_bytes:loc_F47E74	CMP.W CAN_buffer_recv_head.read_pos, var_6[FB]
Up	r	AccuDrv_parsing_process_c_bytes+37	MOV.W CAN_buffer_recv_head.write_pos, R0
Up	r	AccuDrv_parsing_process_c_bytes+47	MOV.W CAN_buffer_recv_head.write_pos, R0
Up	w	AccuDrv_parsing_process_c_bytes+5A	MOV.W var_6[FB], CAN_buffer_recv_head.write_pos
Up	w	AccuDrv_parsing_process_c_bytes+5F	ADD.W #1, CAN_buffer_recv_head.count
Up	r	AccuDrv_do_some_can_stuff+6D	MOV.W CAN_buffer_recv_head.write_pos, R0
Up	r	AccuDrv_do_some_can_stuff:loc_F47F5C	CMP.W CAN_buffer_recv_head.read_pos, R0
Up	r	AccuDrv_do_some_can_stuff+81	MOV.W CAN_buffer_recv_head.write_pos, R1
Up	r	AccuDrv_do_some_can_stuff+9D	MOV.W CAN_buffer_recv_head.write_pos, R1
Up	w	AccuDrv_do_some_can_stuff+AA	MOV.W R0, CAN_buffer_recv_head.write_pos
Up	w	AccuDrv_do_some_can_stuff+AD	ADD.W #1, CAN_buffer_recv_head.count
Up	r	int_CAN_53:loc_F480EC	MOV.W CAN_buffer_recv_head.write_pos, R0
	r	int_CAN_53:loc_F480FE	CMP.W CAN_buffer_recv_head.read_pos, CAN_recv_ring_buffer_pos
D...	r	int_CAN_53+16A	MOV.W CAN_buffer_recv_head.write_pos, R0
D...	w	int_CAN_53+17B	MOV.W CAN_recv_ring_buffer_pos, CAN_buffer_recv_head.write_pos
D...	w	int_CAN_53+181	ADD.W #1, CAN_buffer_recv_head.count
D...	r	int_CAN_53+1EB	MOV.W CAN_buffer_recv_head.write_pos, R0
D...	r	int_CAN_53:loc_F4819D	CMP.W CAN_buffer_recv_head.read_pos, CAN_recv_ring_buffer_pos
D...	r	int_CAN_53+209	MOV.W CAN_buffer_recv_head.write_pos, R0
D...	w	int_CAN_53+21A	MOV.W CAN_recv_ring_buffer_pos, CAN_buffer_recv_head.write_pos
D...	w	int_CAN_53+220	ADD.W #1, CAN_buffer_recv_head.count
D...	r	int_CAN_53+28A	MOV.W CAN_buffer_recv_head.write_pos, R0
D...	r	int_CAN_53:loc_F4823C	CMP.W CAN_buffer_recv_head.read_pos, CAN_recv_ring_buffer_pos
D...	r	int_CAN_53+2A8	MOV.W CAN_buffer_recv_head.write_pos, R0
D...	w	int_CAN_53+2B9	MOV.W CAN_recv_ring_buffer_pos, CAN_buffer_recv_head.write_pos
D...	w	int_CAN_53+2BF	ADD.W #1, CAN_buffer_recv_head.count

Line 3 of 34

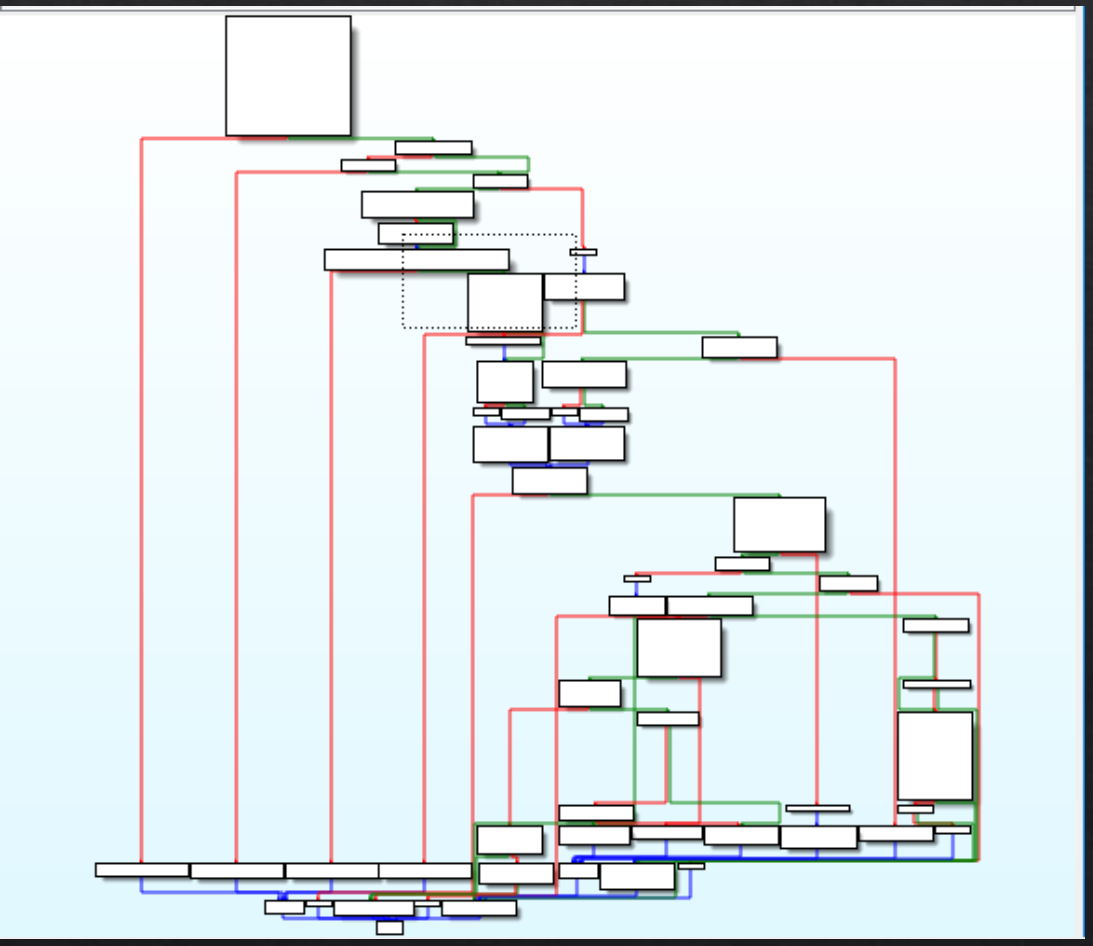
Example 2 – CAN Processing Code

4. Find Command handler



Example 2 – CAN Processing Code

5. Headache

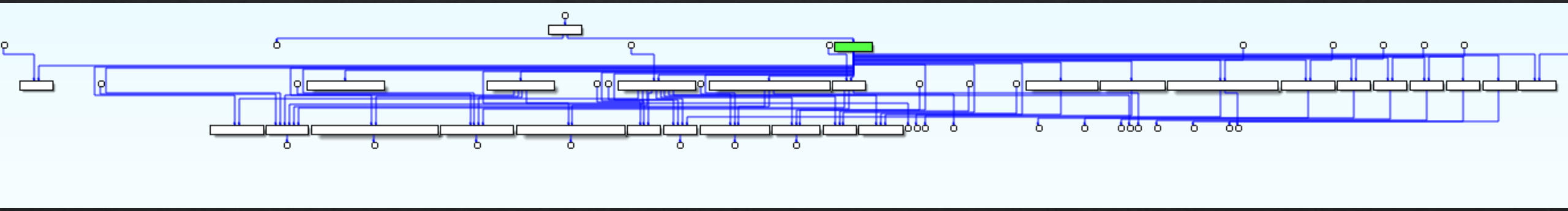


```
    loc_F49E3D:
022 MOV.W   bit_0_to_32_or_CAN_msg_with_room_for_payload[FB], A1
022 ADDX    #8, A1
022 MOV.B   R1L, CAN_body.entry_type[A1]
022 MOV.B   bits_8_to_15_msg_local_ID[FB], CAN_body.ID[A1]
022 MOV.B   [message_ptr[FB]], CAN_body.field_4[A1]
022 MOV.B   #0, [A1]        ; CAN_body.msg_index
022 MOV.B   message_ptr[FB], A0
022 MOV.B   4[A0], CAN_body.data_length[A1]
022 MOV.B   message_ptr[FB], A0
022 MOV.B   5[A0], CAN_body.field_5[A1]
022 MOV.B   message_ptr[FB], A0
022 MOV.B   6[A0], CAN_body.cmd_ID_High[A1]
022 MOV.B   message_ptr[FB], A0
022 MOV.B   7[A0], CAN_body.cmd_ID_low[A1]
022 MOV.B   bit_0_to_32_or_CAN_msg_with_room_for_payload[FB], A0
022 MOV.B   #0, CAN_msg.header.msg_bit_16_was_set[A0]
022 MOV.B   message_ptr[FB], A0
022 BTST    #0, 2[A0]       ; msg_bit_16
022 JEQ/Z   loc_F49E79
```

```
022 MOV.B   bit_0_to_32_or_CAN_msg_with_room_for_payload[FB], A0
022 MOV.B   #1, CAN_msg.header.msg_bit_16_was_set[A0]
```

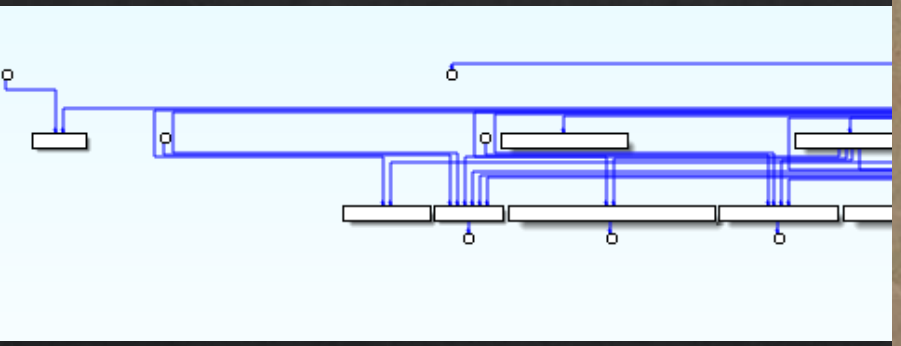
Example 2 – CAN Processing Code

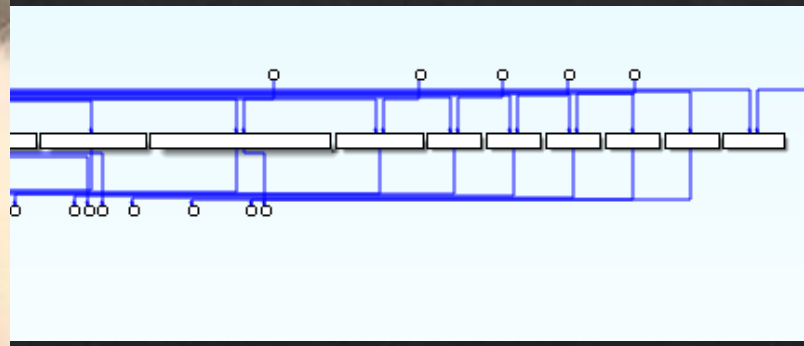
5. Headache



Example 2 – CAN Processing Code

5. Headache





Example 2 – CAN Processing Code

- Results
  - No obvious memory corruption vuln in processing code ☹
  - Various commands identified & exposure to outside world
  - One command can reboot into bootloader mode 😀
- Reasonable next steps
  - Emulation + Fuzzing
  - Explore reflash from bootloader approach
  - Investigate data/logic bugs reachable via CAN code

Lesson Learned – Part II

- Interrupt handlers and SFRs lead to important code
  - CAN/UART/Timers/etc.
- Being familiar with these notions will help when no datasheet is available
- Bringing up datasheet for other components can become handy

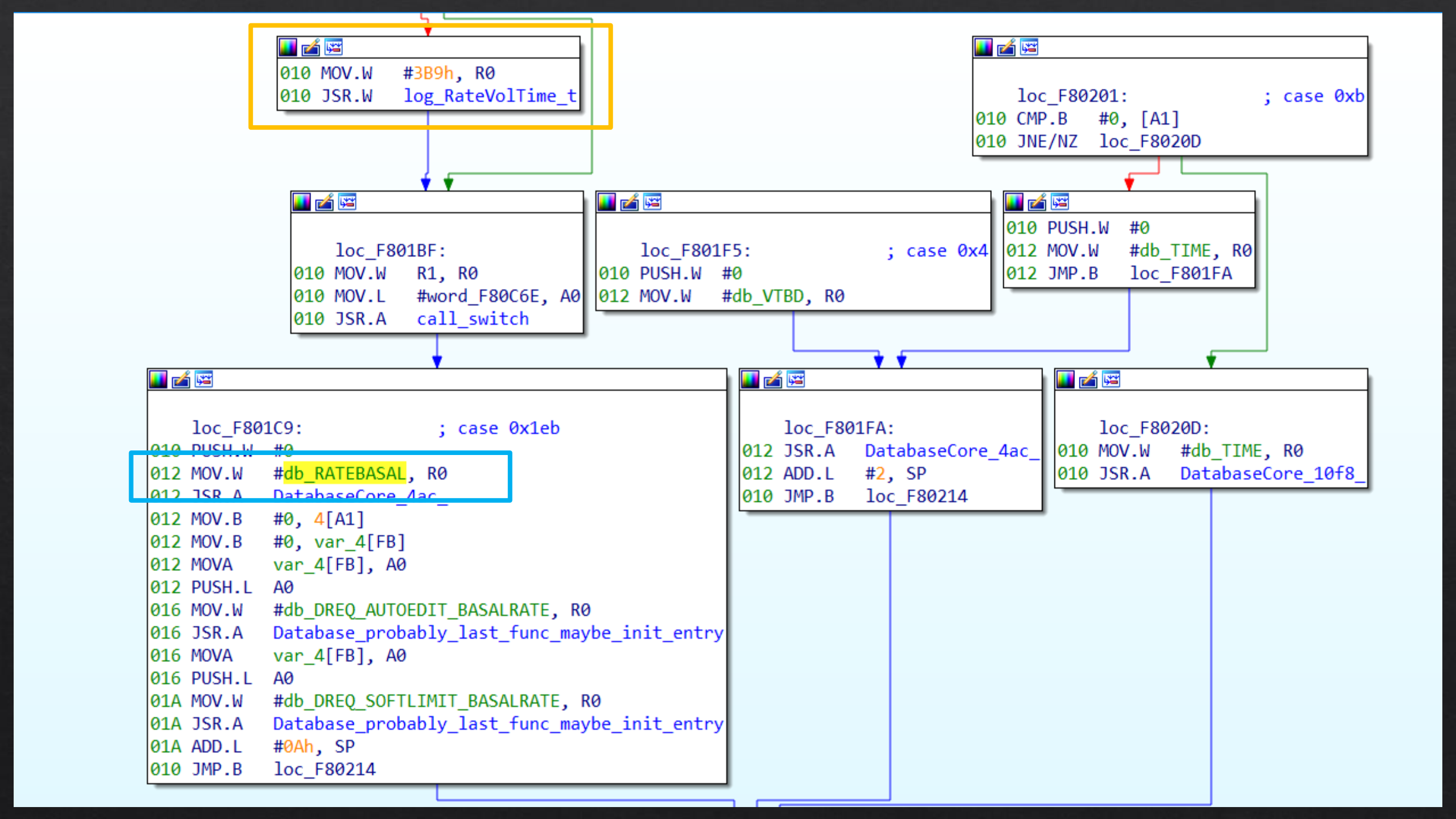
LOOKING FOR HIGHER LEVEL DATA

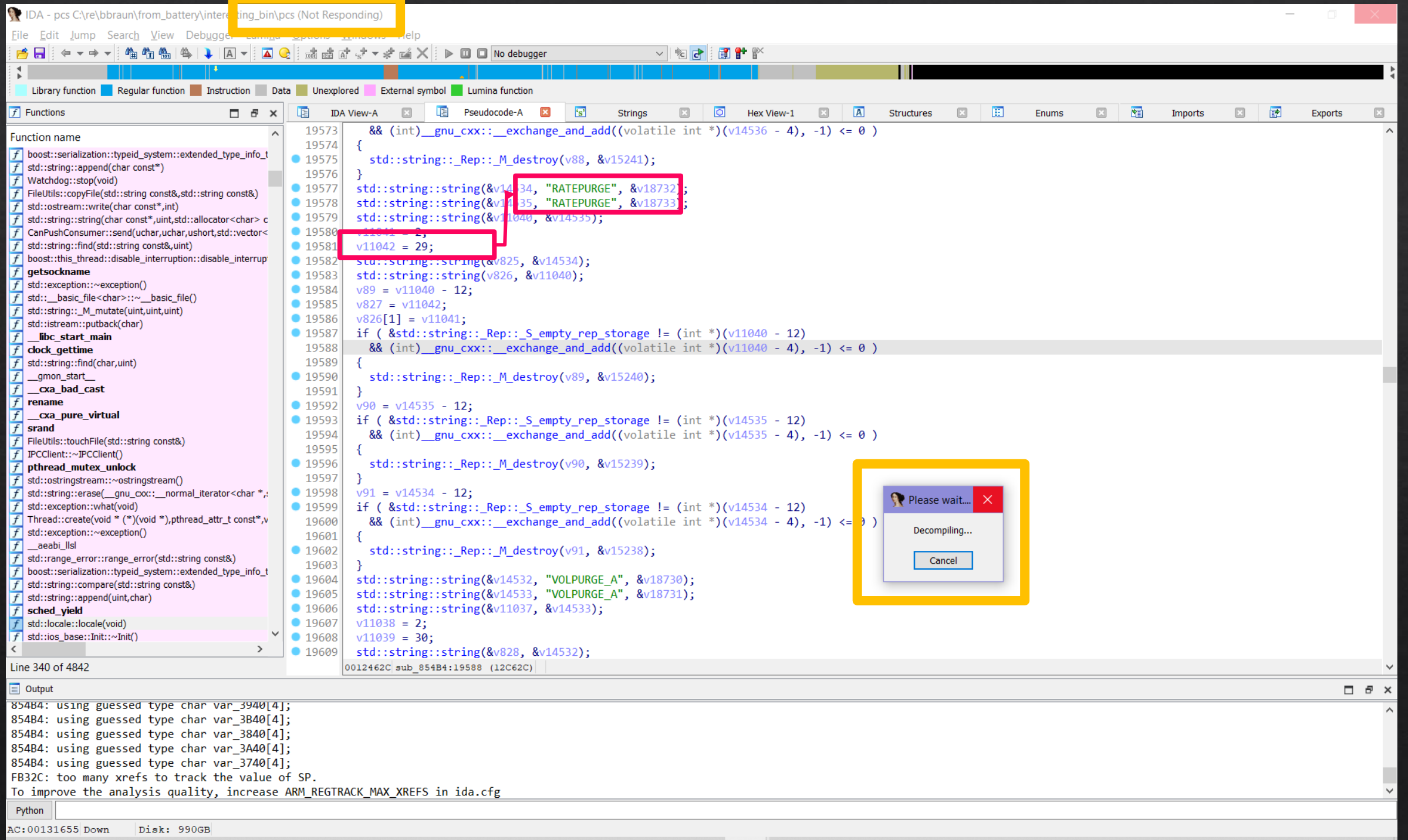
CHEATCODE – INTERNAL DATABASE

- Internal settings represented in a large database
  - Key/Value (+ various params)
  - Exposed over UART/CAN for communication
- Pump talks with communication module
  - Com Module: Linux based, “easy” to reverse its binaries
  - Produces human-readable database backups
  - Map key_id ↔ key_name for easy win!
- Database Read/Write function used throughout firmware
  - Label everything for added context while reversing

```
; enum db_str_key
RATEBASAL       .equ 0          ; XREF: ROM_DATA:00F174A0/s
                                ; ROM_DATA:00F192C2/s ...
RATEBASALMIN    .equ 1          ; XREF: ROM_DATA:00F153F8/s
                                ; ROM_DATA:00F17584/s ...
RATEBASALMAX    .equ 2          ; XREF: ROM_DATA:00F1541E/s
                                ; ROM_DATA:00F175AA/s ...
RATEAKT         .equ 3          ; XREF: ROM_DATA:00F151E4/s
                                ; ROM_DATA:00F16368/s ...
VTBD            .equ 4          ; XREF: ROM_DATA:00F15256/s
                                ; ROM_DATA:00F15F1A/s ...
VTBDMIN         .equ 5          ; XREF: ROM_DATA:00F15490/s
                                ; ROM_DATA:00F15F40/s ...
VTBDMAX         .equ 6          ; XREF: ROM_DATA:00F154B6/s
                                ; ROM_DATA:00F15F66/s ...
VOLAKT          .equ 7          ; XREF: ROM_DATA:00F1527C/s
                                ; ROM_DATA:00F15BB0/s ...
VOLREMAIN       .equ 8          ; XREF: ROM_DATA:00F152A2/s
                                ; ROM_DATA:00F161EC/s ...
TIME            .equ 9          ; XREF: ROM_DATA:00F15360/s
                                ; ROM_DATA:00F16A62/s ...
TIMEMIN         .equ 10         ; XREF: ROM_DATA:00F15528/s
                                ; ROM_DATA:00F16A88/s ...
TIMEMAX         .equ 11         ; XREF: ROM_DATA:00F1554E/s
                                ; ROM_DATA:00F16212/s ...
TIMEAKT         .equ 12         ; XREF: ROM_DATA:00F15386/s
                                ; ROM_DATA:00F16AAE/s
```

```
                                word_0>
call_table_magic_struct <1EBh, RATEBASAL, 5, 0Eh, 2Bh, 1, 0FFFFh, \
                        1436h, 1FFh, 0FFh, 11h, 1, 2, 0, 0, 0, \
                        0FFFFFFFFh, ComTrigger_func51>
call_table_magic_struct <1ECh, SELFTEST_FUPSTEP, 3, 0, 49h, 1, 0FFFFh
                        0, 1FFh, 7Fh, 1, 0, 0, 0, 0, 0, 0FFFFh, \
                        word_0>
call_table_magic_struct <1EDh, SELFTEST_KUPSTEP_A, 3, 0, 4Ah, 2, \
                        0FFFFh, 0, 1FFh, 7Fh, 1, 0, 0, 0, 0, 0, \
                        0FFFFh, GUITrigger_70b>
call_table_magic_struct <1EEh, VOLREMAIN_NL, 5, 1, 2Dh, 1, 0FFFFh, 0,
                        1FFh, 7Fh, 1, 7, 0, 0, 0, 0, 0FFFFFFFFh, \
                        word_0>
call_table_magic_struct <1EFh, VOLAKT_NL, 5, 1, 2Eh, 1, 0FFFFh, 0, \
                        1FFh, 7Fh, 1, 7, 0, 0, 0, 0, 0FFFFFFFFh, \
                        word_0>
call_table_magic_struct <1F0h, VOLINT_NL, 5, 1, 2Fh, 1, 0FFFFh, 0, \
                        1FFh, 7Fh, 1, 7, 0, 0, 0, 0, 0FFFFFFFFh, \
                        word_0>
call_table_magic_struct <1F1h, TIMEREMAIN_MS, 3, 1, 4Ch, 1, 0FFFFh, 0
                        1FFh, 7Fh, 1, 0Dh, 0, 0, 0, 0, 0FFFFh, \
                        word_0>
call_table_magic_struct <1F2h, TIMEAKT_MS, 3, 1, 4Dh, 1, 0FFFFh, 0, \
                        1FFh, 7Fh, 1, 0Dh, 0, 0, 0, 0, 0FFFFh, \
                        word_0>
call_table_magic_struct <1F3h, TIMEINT_MS, 3, 1, 4Eh, 1, 0FFFFh, 0, \
                        1FFh, 7Fh, 1, 0Dh, 0, 0, 0, 0, 0FFFFh, \
                        word_0>
call_table_magic_struct <1F4h, PGM_INTVL_AKT_VOL, 5, 1, 30h, 1, \
                        0FFFFh, 0, 1FFh, 7Fh, 11h, 2, 2, 0, 0, 0, \
                        98961Ch, word_0>
```


010 MOV.W #3B9h, R0
010 JSR.W log_RateVolTime_t
loc_F80201: ; case 0xb
010 CMP.B #0, [A1]
010 JNE/NZ loc_F8020D
loc_F801BF:
010 MOV.W R1, R0
010 MOV.L #word_F80C6E, A0
010 JSR.A call_switch
loc_F801F5: ; case 0x4
010 PUSH.W #0
012 MOV.W #db_VTBD, R0
010 PUSH.W #0
012 MOV.W #db_TIME, R0
012 JMP.B loc_F801FA
loc_F801C9: ; case 0x1eb
010 PUSH.W #0
012 MOV.W #db_RATEBASAL, R0
012 JSR.A DatabaseCore_4ac_
012 MOV.B #0, 4[A1]
012 MOV.B #0, var_4[FB]
012 MOVA var_4[FB], A0
012 PUSH.L A0
016 MOV.W #db_DREQ_AUTOEDIT_BASALRATE, R0
016 JSR.A Database_probably_last_func_maybe_init_entry
016 MOVA var_4[FB], A0
016 PUSH.L A0
01A MOV.W #db_DREQ_SOFTLIMIT_BASALRATE, R0
01A JSR.A Database_probably_last_func_maybe_init_entry
01A ADD.L #0Ah, SP
010 JMP.B loc_F80214
loc_F801FA:
012 JSR.A DatabaseCore_4ac_
012 ADD.L #2, SP
010 JMP.B loc_F80214
loc_F8020D:
010 MOV.W #db_TIME, R0
010 JSR.A DatabaseCore_10f8_


IDA - pcs C:\re\bbraun\from_battery\interesting_bin\pcs (Not Responding)
File Edit Jump Search View Debugger Lumina Options Windows Help
No debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name
boost::serialization::typeid_system::extended_type_info_t
std::string::append(char const*)
Watchdog::stop(void)
FileUtils::copyFile(std::string const&,std::string const&)
std::ostream::write(char const*,int)
std::string::string(char const*,uint,std::allocator<char> c
CanPushConsumer::send(uchar,uchar,ushort,std::vector<
std::string::find(std::string const&,uint)
boost::this_thread::disable_interruption::disable_interrup
getsockname
std::exception::~exception()
std::__basic_file<char>::~__basic_file()
std::string::_M_mutate(uint,uint,uint)
std::istream::putback(char)
__libc_start_main
clock_gettime
std::string::find(char,uint)
__gmon_start__
__cxa_bad_cast
rename
__cxa_pure_virtual
srand
FileUtils::touchFile(std::string const&)
IPCClient::~IPCClient()
pthread_mutex_unlock
std::ostringstream::~ostringstream()
std::string::erase(__gnu_cxx::__normal_iterator<char *,
std::exception::what(void)
Thread::create(void * (*)(void *),pthread_attr_t const*,v
std::exception::~exception()
__aeabi_llsl
std::range_error::range_error(std::string const&)
boost::serialization::typeid_system::extended_type_info_t
std::string::compare(std::string const&)
std::string::append(uint,char)
sched_yield
std::locale::locale(void)
std::ios_base::Init::~Init()
Line 340 of 4842
IDA View-A
Pseudocode-A
Strings
Hex View-1
Structures
Enums
Imports
Exports
19573 && (int)__gnu_cxx::__exchange_and_add((volatile int *)(v14536 - 4), -1) <= 0)
19574 {
19575 std::string::_Rep::_M_destroy(v88, &v15241);
19576 }
19577 std::string::string(&v14534, "RATEPURGE", &v18732);
19578 std::string::string(&v14535, "RATEPURGE", &v18733);
19579 std::string::string(&v11040, &v14535);
19580 v11041 = 2;
19581 v11042 = 29;
19582 std::string::string(&v825, &v14534);
19583 std::string::string(v826, &v11040);
19584 v89 = v11040 - 12;
19585 v827 = v11042;
19586 v826[1] = v11041;
19587 if (&std::string::_Rep::_S_empty_rep_storage != (int *)(v11040 - 12)
19588 && (int)__gnu_cxx::__exchange_and_add((volatile int *)(v11040 - 4), -1) <= 0)
19589 {
19590 std::string::_Rep::_M_destroy(v89, &v15240);
19591 }
19592 v90 = v14535 - 12;
19593 if (&std::string::_Rep::_S_empty_rep_storage != (int *)(v14535 - 12)
19594 && (int)__gnu_cxx::__exchange_and_add((volatile int *)(v14535 - 4), -1) <= 0)
19595 {
19596 std::string::_Rep::_M_destroy(v90, &v15239);
19597 }
19598 v91 = v14534 - 12;
19599 if (&std::string::_Rep::_S_empty_rep_storage != (int *)(v14534 - 12)
19600 && (int)__gnu_cxx::__exchange_and_add((volatile int *)(v14534 - 4), -1) <= 0)
19601 {
19602 std::string::_Rep::_M_destroy(v91, &v15238);
19603 }
19604 std::string::string(&v14532, "VOLPURGE_A", &v18730);
19605 std::string::string(&v14533, "VOLPURGE_A", &v18731);
19606 std::string::string(&v11037, &v14533);
19607 v11038 = 2;
19608 v11039 = 30;
19609 std::string::string(&v828, &v14532);
0012462C sub_854B4:19588 (12C62C)
Please wait...
Decompiling...
Cancel
Output
854B4: using guessed type char var_3940[4];
854B4: using guessed type char var_3B40[4];
854B4: using guessed type char var_3840[4];
854B4: using guessed type char var_3A40[4];
854B4: using guessed type char var_3740[4];
FB32C: too many xrefs to track the value of SP.
To improve the analysis quality, increase ARM_REGTRACK_MAX_XREFS in ida.cfg
Python
AC:00131655 Down Disk: 990GB

IDA - pcs C:\re\bbraun\from_battery\interesting_bin\pcs (Not Responding)

File Edit Jump Search View Debugger Lumina Options Windows Help

No debugger

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

Functions

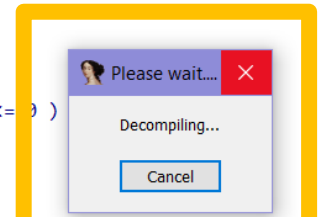
Function name

- boost::serialization::typeid_system::extended_type_info_t
- std::string::append(char const*)
- Watchdog::stop(void)
- FileUtils::copyFile(std::string const&,std::string const&)
- std::ostream::write(char const*,int)
- std::string::string(char const*,uint,std::allocator<char> c
- CanPushConsumer::send(uchar,uchar,ushort,std::vector<
- std::string::find(std::string const&,uint)
- boost::this_thread::disable_interruption::disable_interrup
- **getsockname**
- std::exception::~exception()
- std::__basic_file<char>::~__basic_file()
- std::string::_M_mutate(uint,uint,uint)
- std::istream::putback(char)
- **__libc_start_main**
- **clock_gettime**
- std::string::find(char,uint)
- __gmon_start__
- **__cxa_bad_cast**
- **rename**
- **__cxa_pure_virtual**
- **srand**
- FileUtils::touchFile(std::string const&)
- IPCClient::~IPCClient()
- **pthread_mutex_unlock**
- std::ostringstream::~ostringstream()
- std::string::erase(__gnu_cxx::__normal_iterator<char *,
- std::exception::what(void)
- Thread::create(void * (*)(void *),pthread_attr_t const*,v
- std::exception::~exception()
- __aeabi_llsl
- std::range_error::range_error(std::string const&)
- boost::serialization::typeid_system::extended_type_info_t
- std::string::compare(std::string const&)
- std::string::append(uint,char)
- **sched_yield**
- std::locale::locale(void)
- std::ios_base::Init::~Init()

Line 340 of 4842

IDA View-A | Pseudocode-A | Strings | Hex View-1 | Structures | Enums | Imports | Exports

```
19573     && (int)__gnu_cxx::__exchange_and_add((volatile int *)(v14536 - 4), -1) <= 0 )
19574   {
19575     std::string::_Rep::_M_destroy(v88, &v15241);
19576   }
19577   std::string::string(&v14534, "RATEPURGE", &v18732);
19578   std::string::string(&v14535, "RATEPURGE", &v18733);
19579   std::string::string(&v11040, &v14535);
19580   v11041 = 2;
19581   v11042 = 29;
19582   std::string::string(&v825, &v14534);
19583   std::string::string(v826, &v11040);
19584   v89 = v11040 - 12;
19585
19586
19587                                                    12)
19588                                                    4), -1) <= 0 )
19589
19590
19591
19592
19593                                                    12)
19594                                                    4), -1) <= 0 )
19595
19596
19597
19598
19599                                                    12)
19600                                                    4), -1) <= 0 )
19601
19602
19603
19604
19605
19606
19607
19608
19609
```



Output

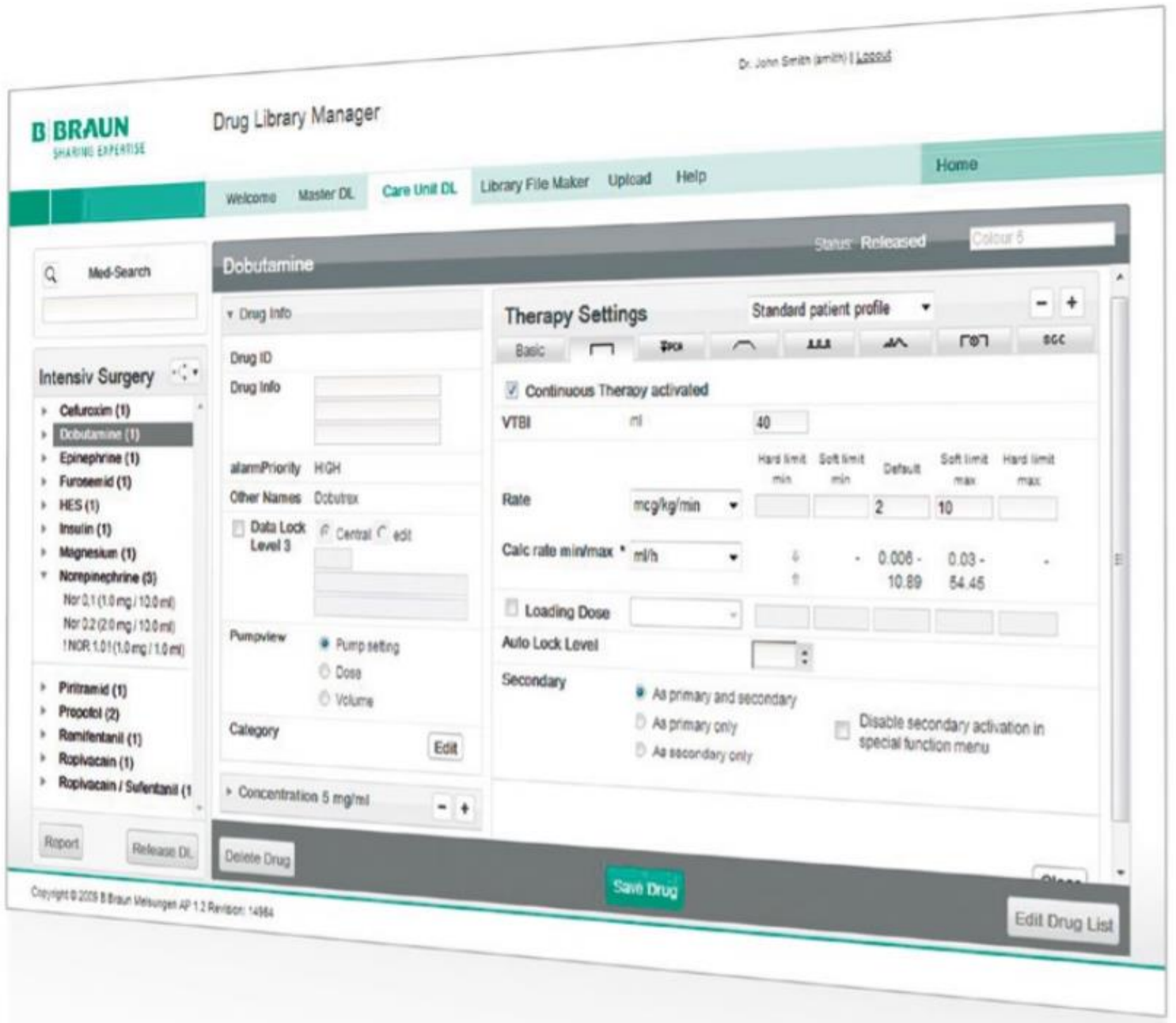
```
854B4: using guessed type char var_3940[4];
854B4: using guessed type char var_3B40[4];
854B4: using guessed type char var_3840[4];
854B4: using guessed type char var_3A40[4];
854B4: using guessed type char var_3740[4];
FB32C: too many xrefs to track the value of SP.
To improve the analysis quality, increase ARM_REGTRACK_MAX_XREFS in ida.cfg
```

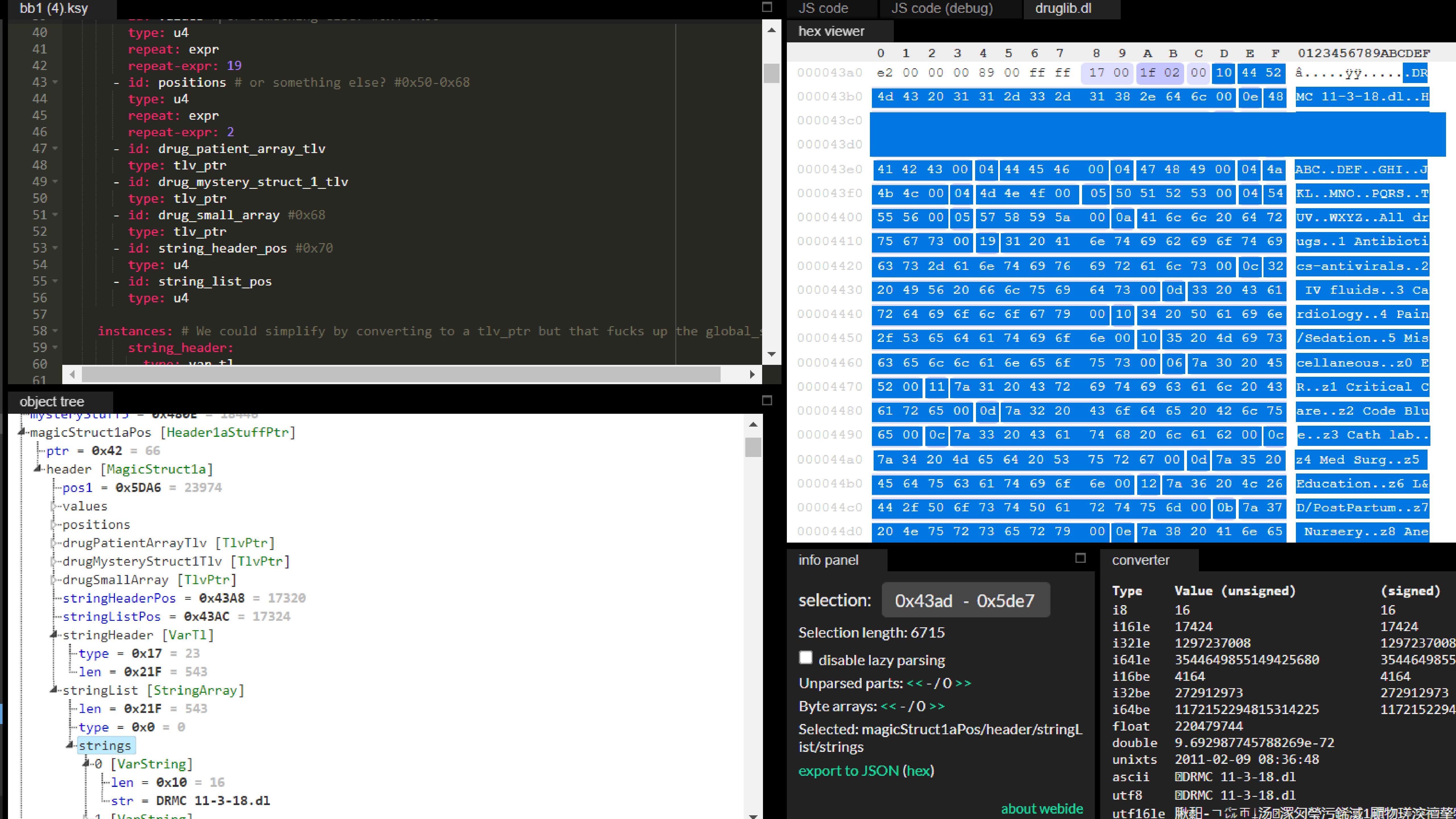
Python

AC:00131655 Down Disk: 990GB

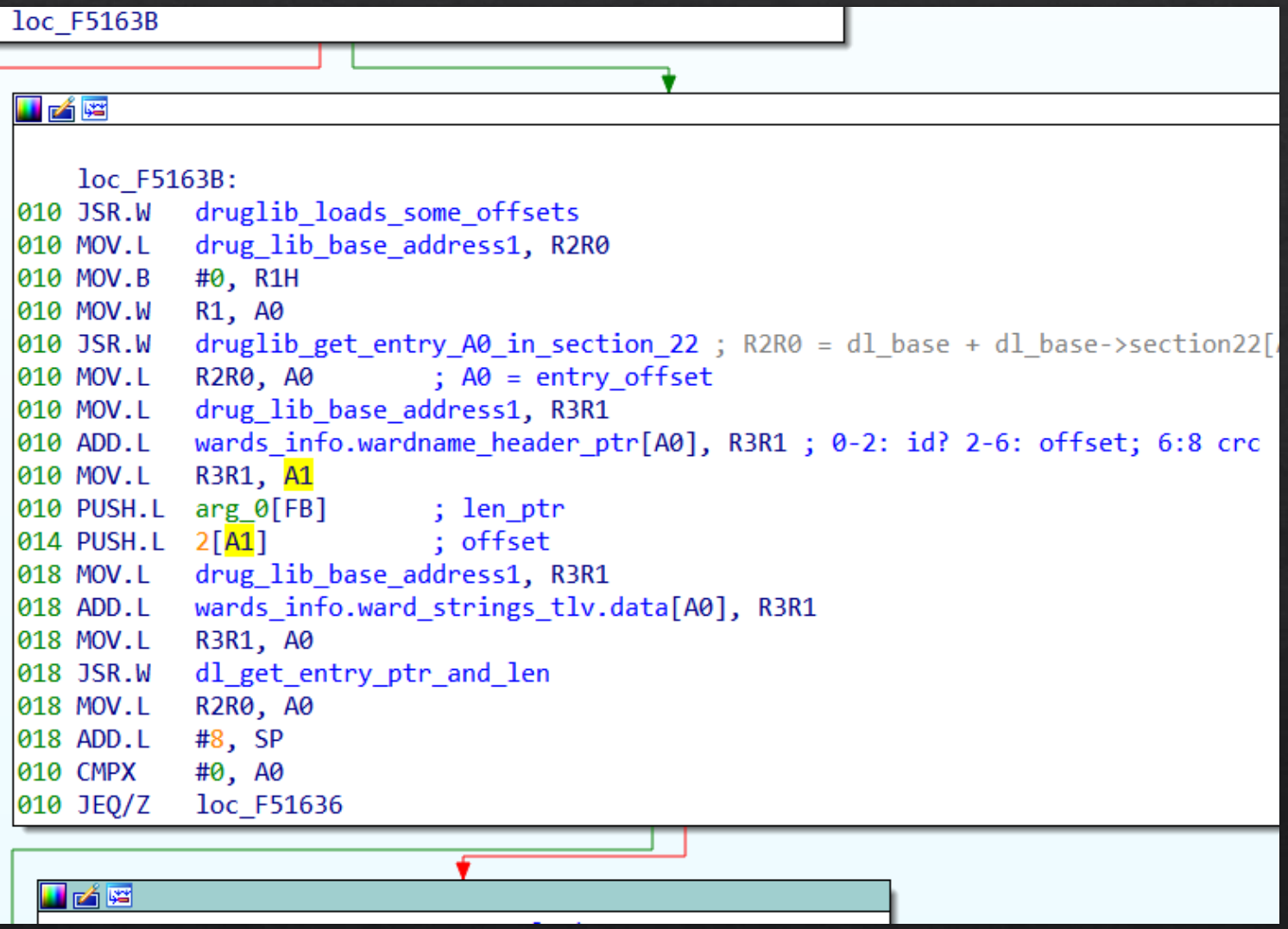
Figuring out the Drug Library

- Undocumented data format
- Business logic!
  - Used to specify various drugs available in the wards
  - Implement Soft and Hard limits to prevent mis-dosage
- Potential target for Vuln Research…


B BRAUN
SHARING EXPERTISE
Drug Library Manager
Dr. John Smith (smith) | Logout
Welcome
Master DL
Care Unit DL
Library File Maker
Upload
Help
Home
Med-Search
Intensiv Surgery
Cefuroxim (1)
Dobutamine (1)
Epinephrine (1)
Furosemid (1)
HES (1)
Insulin (1)
Magnesium (1)
Norepinephrine (3)
Nor 0.1 (1.0 mg / 10.0 ml)
Nor 0.2 (2.0 mg / 10.0 ml)
NOR 1.0 (1.0 mg / 1.0 ml)
Piritramid (1)
Propofol (2)
Remifentanil (1)
Ropivacain (1)
Ropivacain / Sufentanil (1
Report
Release DL
Dobutamine
Status: Released
Drug Info
Drug ID
Drug Info
alarmPriority HIGH
Other Names Dobutrex
Data Lock Level 3
Central
edit
Pumpview
Pump setting
Dose
Volume
Category
Edit
Concentration 5 mg/ml
Therapy Settings
Standard patient profile
Basic
Continuous Therapy activated
VTBI
ml
40
Hard limit min
Soft limit min
Default
Soft limit max
Hard limit max
Rate
mcg/kg/min
2
10
Calc rate min/max *
ml/h
0.006 - 10.89
0.03 - 54.45
Loading Dose
Auto Lock Level
Secondary
As primary and secondary
As primary only
As secondary only
Disable secondary activation in special function menu
Delete Drug
Save Drug
Edit Drug List
Copyright © 2009 B Braun Melsungen AP 1.2 Revision: 14984



bb1 (4).ksy
type: u4
repeat: expr
repeat-expr: 19
- id: positions # or something else? #0x50-0x68
type: u4
repeat: expr
repeat-expr: 2
- id: drug_patient_array_tlv
type: tlv_ptr
- id: drug_mystery_struct_1_tlv
type: tlv_ptr
- id: drug_small_array #0x68
type: tlv_ptr
- id: string_header_pos #0x70
type: u4
- id: string_list_pos
type: u4
instances: # We could simplify by converting to a tlv_ptr but that fucks up the global_
string_header:
JS code
JS code (debug)
druglib.dl
hex viewer
0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
000043a0 e2 00 00 00 89 00 ff ff 17 00 1f 02 00 10 44 52 â.....ÿÿ......DR
000043b0 4d 43 20 31 31 2d 33 2d 31 38 2e 64 6c 00 0e 48 MC 11-3-18.dl..H
000043c0
000043d0
000043e0 41 42 43 00 04 44 45 46 00 04 47 48 49 00 04 4a ABC..DEF..GHI..J
000043f0 4b 4c 00 04 4d 4e 4f 00 05 50 51 52 53 00 04 54 KL..MNO..PQRS..T
00004400 55 56 00 05 57 58 59 5a 00 0a 41 6c 6c 20 64 72 UV..WXYZ..All dr
00004410 75 67 73 00 19 31 20 41 6e 74 69 62 69 6f 74 69 ugs..1 Antibioti
00004420 63 73 2d 61 6e 74 69 76 69 72 61 6c 73 00 0c 32 cs-antivirals..2
00004430 20 49 56 20 66 6c 75 69 64 73 00 0d 33 20 43 61 IV fluids..3 Ca
00004440 72 64 69 6f 6c 6f 67 79 00 10 34 20 50 61 69 6e rdiology..4 Pain
00004450 2f 53 65 64 61 74 69 6f 6e 00 10 35 20 4d 69 73 /Sedation..5 Mis
00004460 63 65 6c 6c 61 6e 65 6f 75 73 00 06 7a 30 20 45 cellaneous..z0 E
00004470 52 00 11 7a 31 20 43 72 69 74 69 63 61 6c 20 43 R..z1 Critical C
00004480 61 72 65 00 0d 7a 32 20 43 6f 64 65 20 42 6c 75 are..z2 Code Blu
00004490 65 00 0c 7a 33 20 43 61 74 68 20 6c 61 62 00 0c e..z3 Cath lab..
000044a0 7a 34 20 4d 65 64 20 53 75 72 67 00 0d 7a 35 20 z4 Med Surg..z5
000044b0 45 64 75 63 61 74 69 6f 6e 00 12 7a 36 20 4c 26 Education..z6 L&
000044c0 44 2f 50 6f 73 74 50 61 72 74 75 6d 00 0b 7a 37 D/PostPartum..z7
000044d0 20 4e 75 72 73 65 72 79 00 0e 7a 38 20 41 6e 65 Nursery..z8 Ane
object tree
magicStruct1aPos [Header1aStuffPtr]
ptr = 0x42 = 66
header [MagicStruct1a]
pos1 = 0x5DA6 = 23974
values
positions
drugPatientArrayTlv [TlvPtr]
drugMysteryStruct1Tlv [TlvPtr]
drugSmallArray [TlvPtr]
stringHeaderPos = 0x43A8 = 17320
stringListPos = 0x43AC = 17324
stringHeader [VarTl]
type = 0x17 = 23
len = 0x21F = 543
stringList [StringArray]
len = 0x21F = 543
type = 0x0 = 0
strings
0 [VarString]
len = 0x10 = 16
str = DRMC 11-3-18.dl
info panel
selection: 0x43ad - 0x5de7
Selection length: 6715
disable lazy parsing
Unparsed parts: << - / 0 >>
Byte arrays: << - / 0 >>
Selected: magicStruct1aPos/header/stringList/strings
export to JSON (hex)
about webide
converter
Type Value (unsigned) (signed)
i8 16 16
i16le 17424 17424
i32le 1297237008 1297237008
i64le 3544649855149425680
i16be 4164 4164
i32be 272912973 272912973
i64be 1172152294815314225
float 220479744
double 9.692987745788269e-72
unixts 2011-02-09 08:36:48
ascii DRMC 11-3-18.dl
utf8 DRMC 11-3-18.dl




```
0 [WardEntry]
  pos = 0x5DEC = 24044
  entry [WardInfo]
    foo = [92, 138, 0, 0, 66, 192, 0, 0, ...]
    wardnameHeader [WardInfoHeaderPtr]
    drugEntryLargeIndexArray [TlvPtr]
    drugEntryLargeArray [TlvPtr]
    drugExtraInfoArray [TlvPtr]
    mysteryTlvTypeC [TlvPtr]
    dummy2 = [240, 208, 0, 0, 255, 255, 255, 255, ...]
    wardStrings [TlvPtr]
      headerPos = 0xDEBC = 57020
      dataPos = 0xDEC0 = 57024
      header
      data [StringArray]
        len = 0x89 = 137
        type = 0x0 = 0
        strings
          0 [VarString]
            len = 0x5 = 5
            str = 0 ER
          1 [VarString]
          2 [VarString]
```


B BRAUN
Select Care Unit
OATR
1 Critical care
2 Code Blue
Infusomat® Space

Disposable Data

- Calibration data used by the pump
- More Business logic!
  - Easier than Druglib
  - Deeper in abstraction stack
- Good target for Vuln Research…

```
Example2: Disposable data
[COMMON0]
TUBETABCHKSUM=35443
TUBETABSIZE=3140
DISPDATA_VERSION=506
TUBECRCKUP=63207
TUBETABSIZE_KUP=688
TUBERELEASE_TABCHKSUM=20106
TUBERELEASE_TABSIZE=22
TUBERELEASE_KUP_TABSIZE=22
TUBERELEASE_KUP_TABCRC=18226
TUBE_COUNT=3
[TUBE0]
TUBENAME_A=
TUBE_HEADVOLUME_A=0
//... snipped out since empty section ..
[TUBE1]
TUBENAME_A=Intrafix PVC
TUBE_HEADVOLUME_A=204
TUBE_MAXBOLVOL_A=500
TUBEPRESSURESURFACE_A=1131
TUBE_AIRALARMLEVEL_A=23
TUBE_DRIPCHAMBER_A=20
TUBE_MAXRATE_A=120000
TUBE_MAXBOLRATE_A=120000
TUBECRCFUP_A=41266
```

Disposable Data

- Allow the pump to handle different infusion tubes (different volume, rates, …)
- TUBE_HEADVOLUME
  - "Amount" of drug per squeeze of the pump
  - Used for calculations
    - Wrong value → incorrect volume estimation → over/under delivery

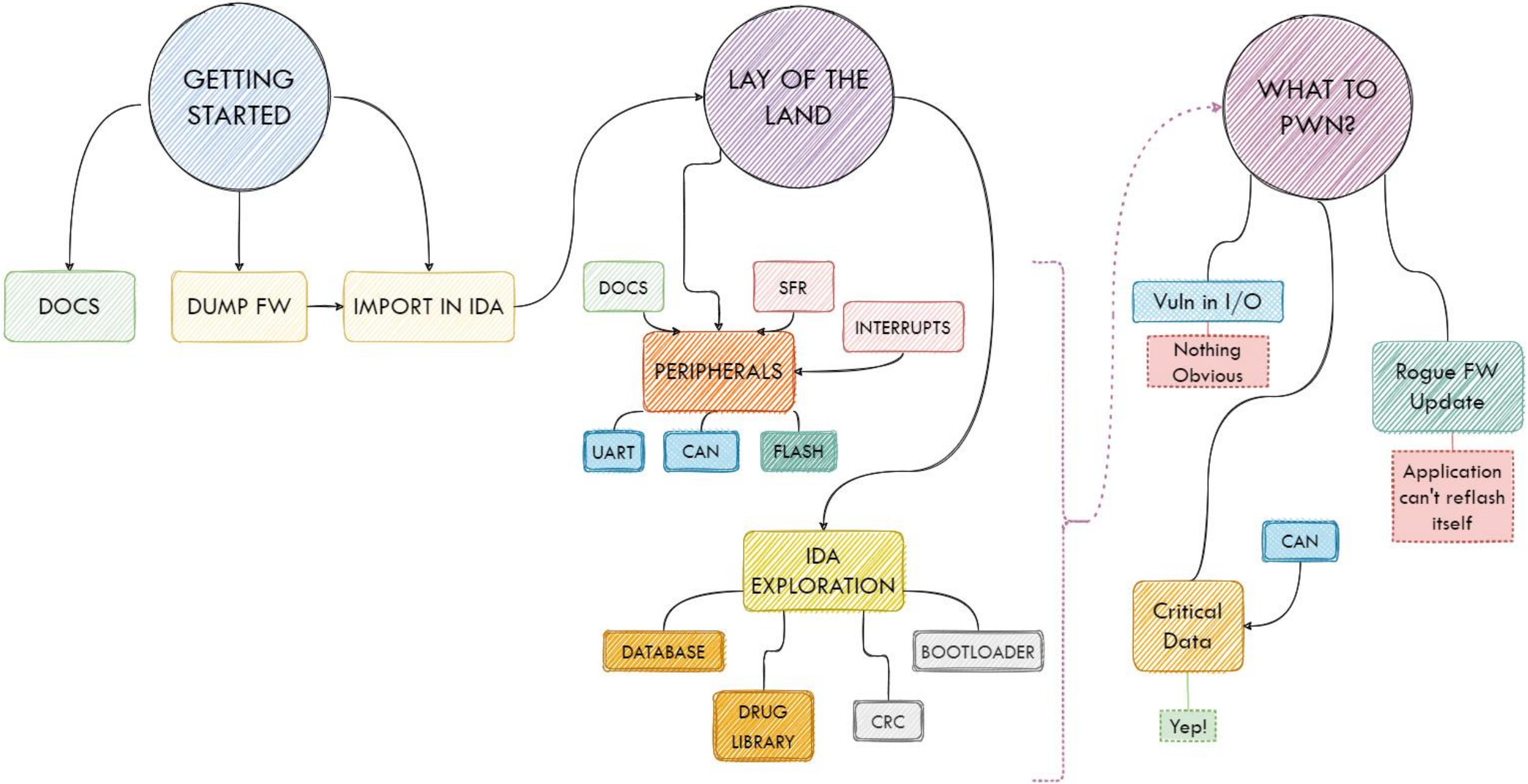
```
ENTER   #8
PUSHM   R1,R3,A0,A1
MOV.L   R2R0, tubetime[FB]
MOV.W   tubeheadvolume[FB], R0
MOV.W   device_delivery_constant[FB], R2
MOV.W   TUBE_TAUHEADVOLUME_10MS[FB], A0
MOV.W   TAUDELIVERYOFFSET[FB], A1
JSR.A   sub_F32A30      ; R0 = ((R0*0xffff)/500) & 0xFFFF
MOV.W   R0, R1
MOV.W   #0, R3
MOV.L   R3R1, var_8[FB] ; var_8 = ((tubeheadvolume*0xffff)//500)&0xFFFF = THV_scaled
MOV.W   R2, R0
EXTS.W  R0              ; EXTS.W -> signs extend R0 into R2 (???)
JSR.W   sub_F3228D      ; R2R0 = R0*R3R1 / 1000
ADD.L   R2R0, var_8[FB] ; var_8 = THV_scaled*(1 + device_delivery_constant/1000) = THV_scaled_adjusted
MOV.B   #0, g_is_computing_delivery_param_done ; maybe copied from unk_F0BD14
MOV.W   #180, const_180 ; = 180
```

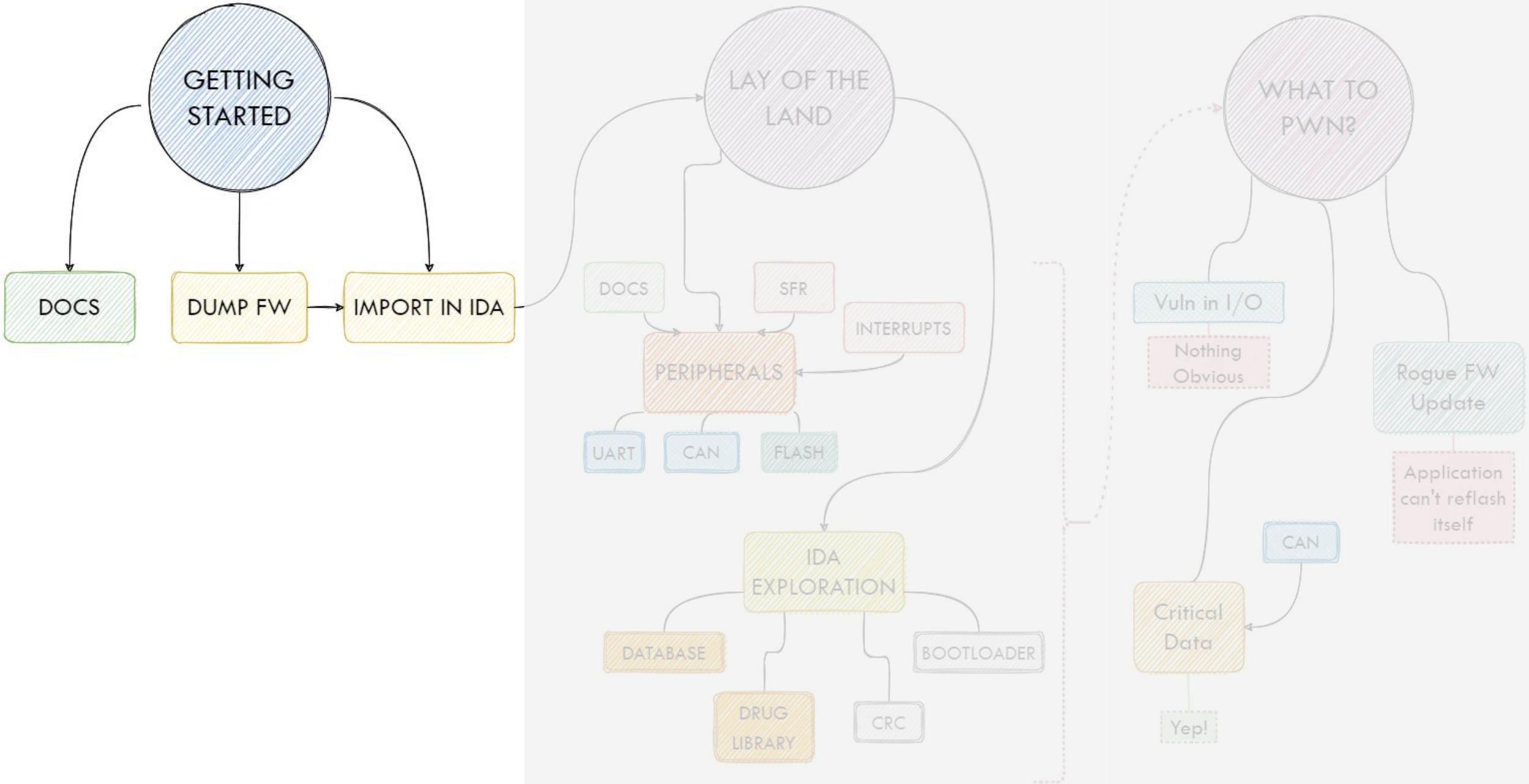
DEMO VIDEO

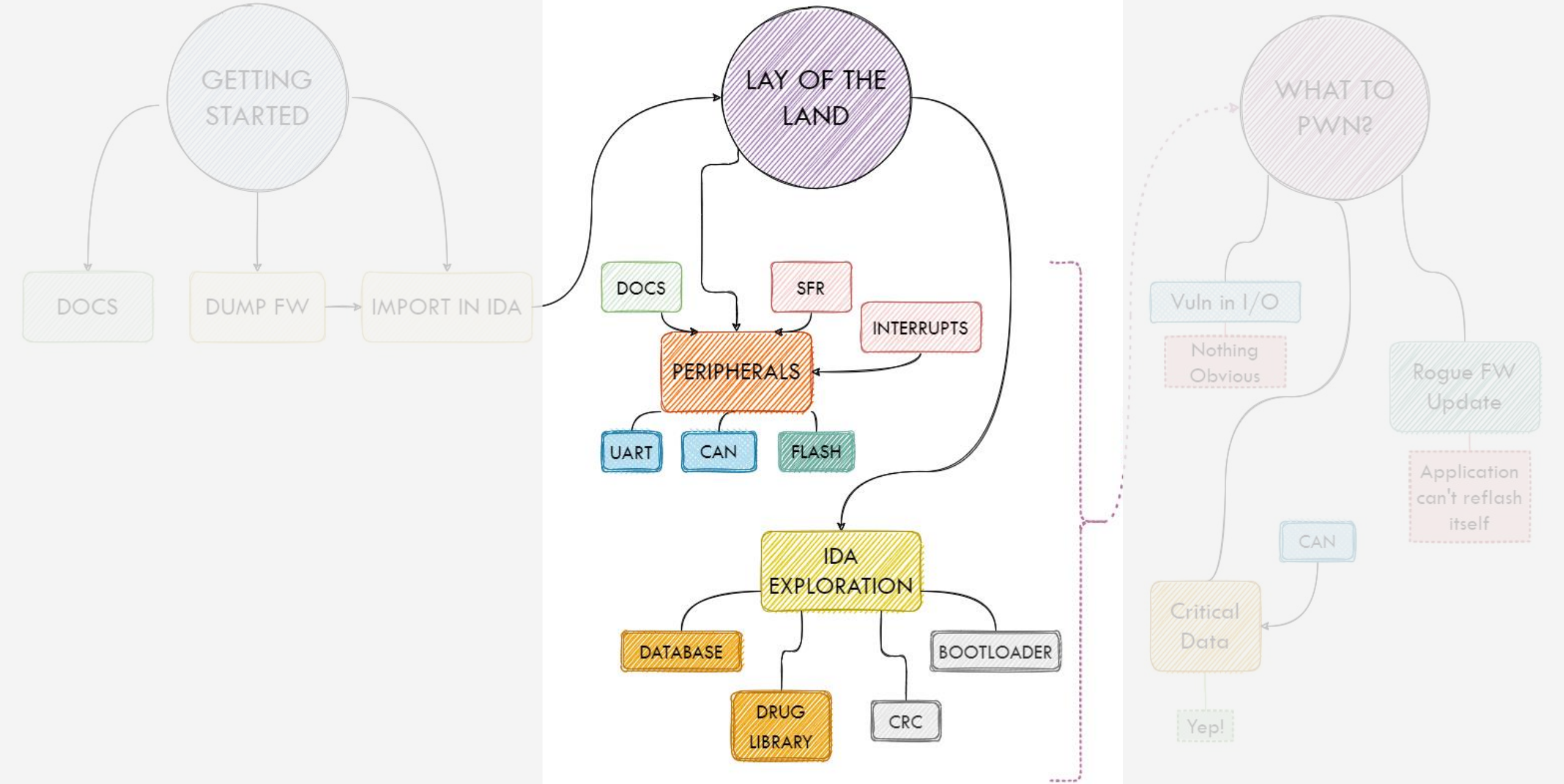
https://www.youtube.com/watch?v=N8SFxH-jNRw

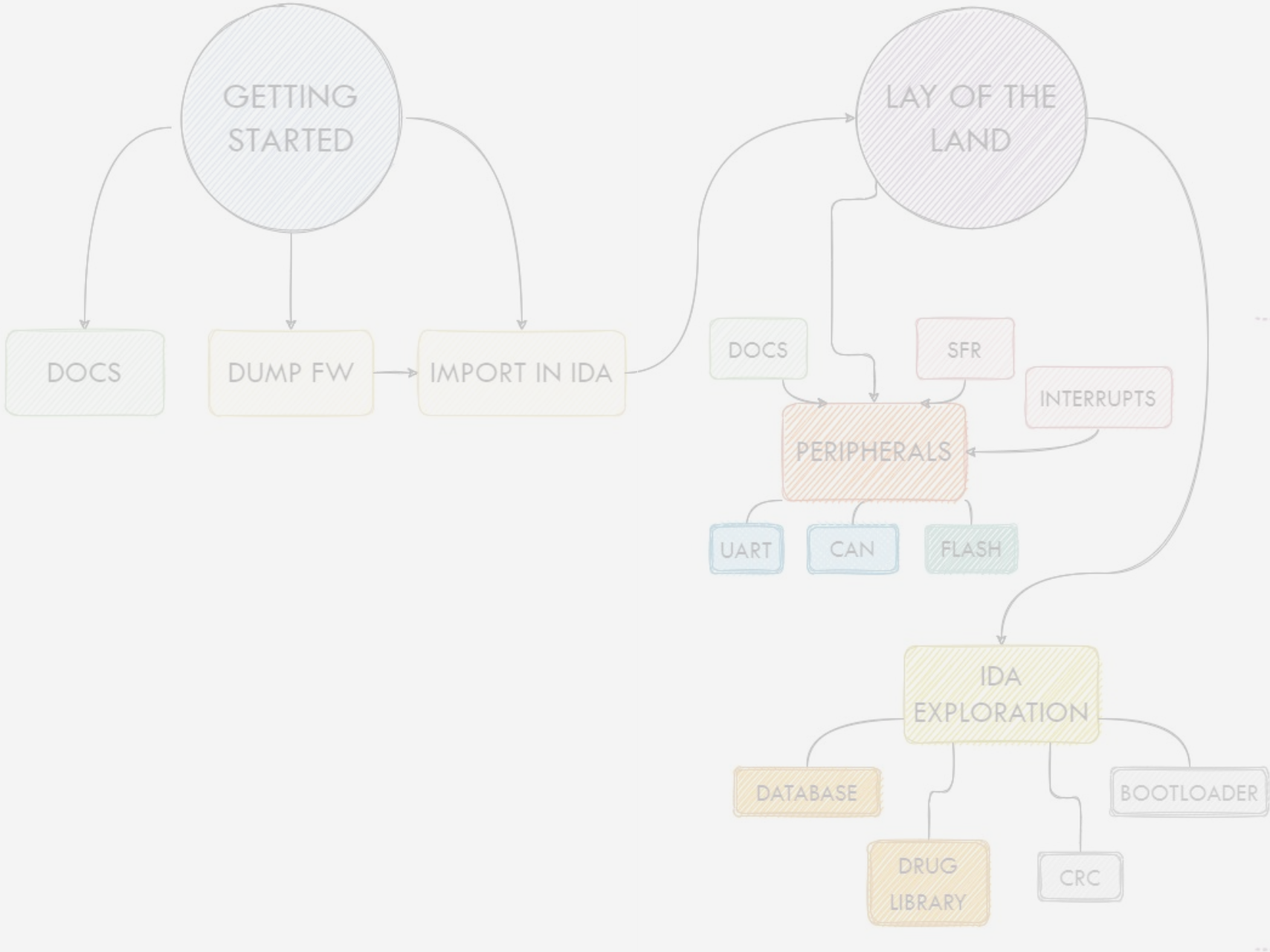
Lesson Learned – Part III

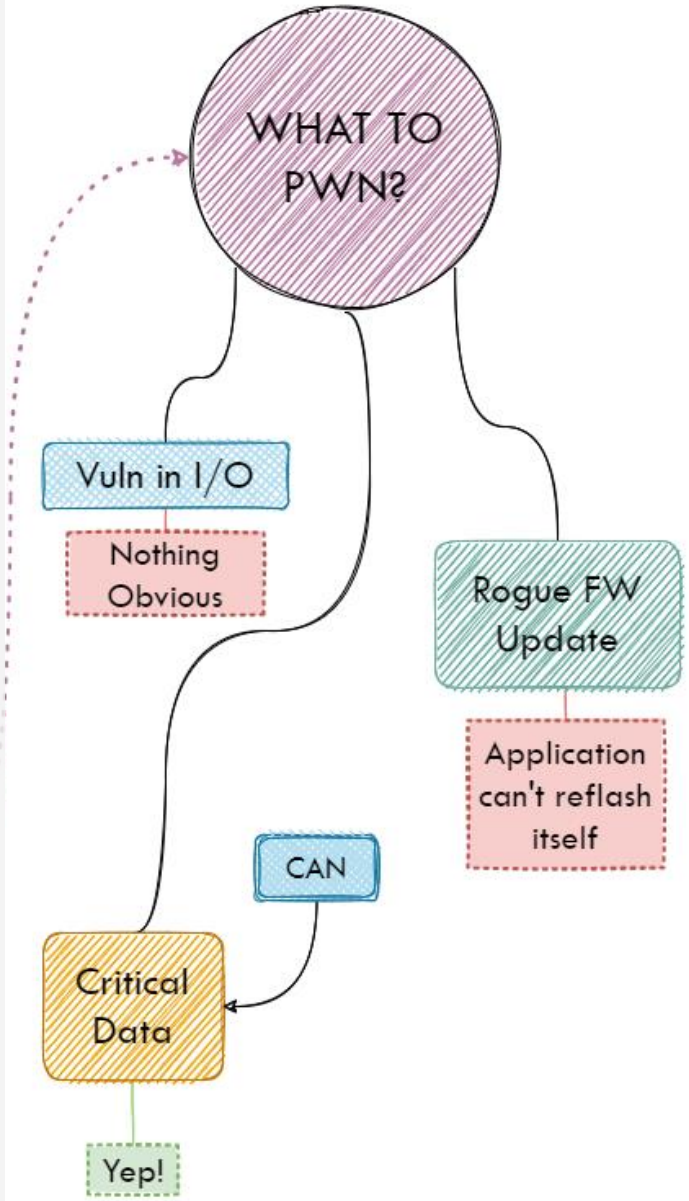
- Remember to look at the system as a whole
  - External code can be used as a Rosetta Stone to clarify embedded behaviors
  - Understanding internal representation of data is crucial
- Kaitai can be super useful for reversing mysterious data structure
- Logic vulns can be as deadly as memory corruptions

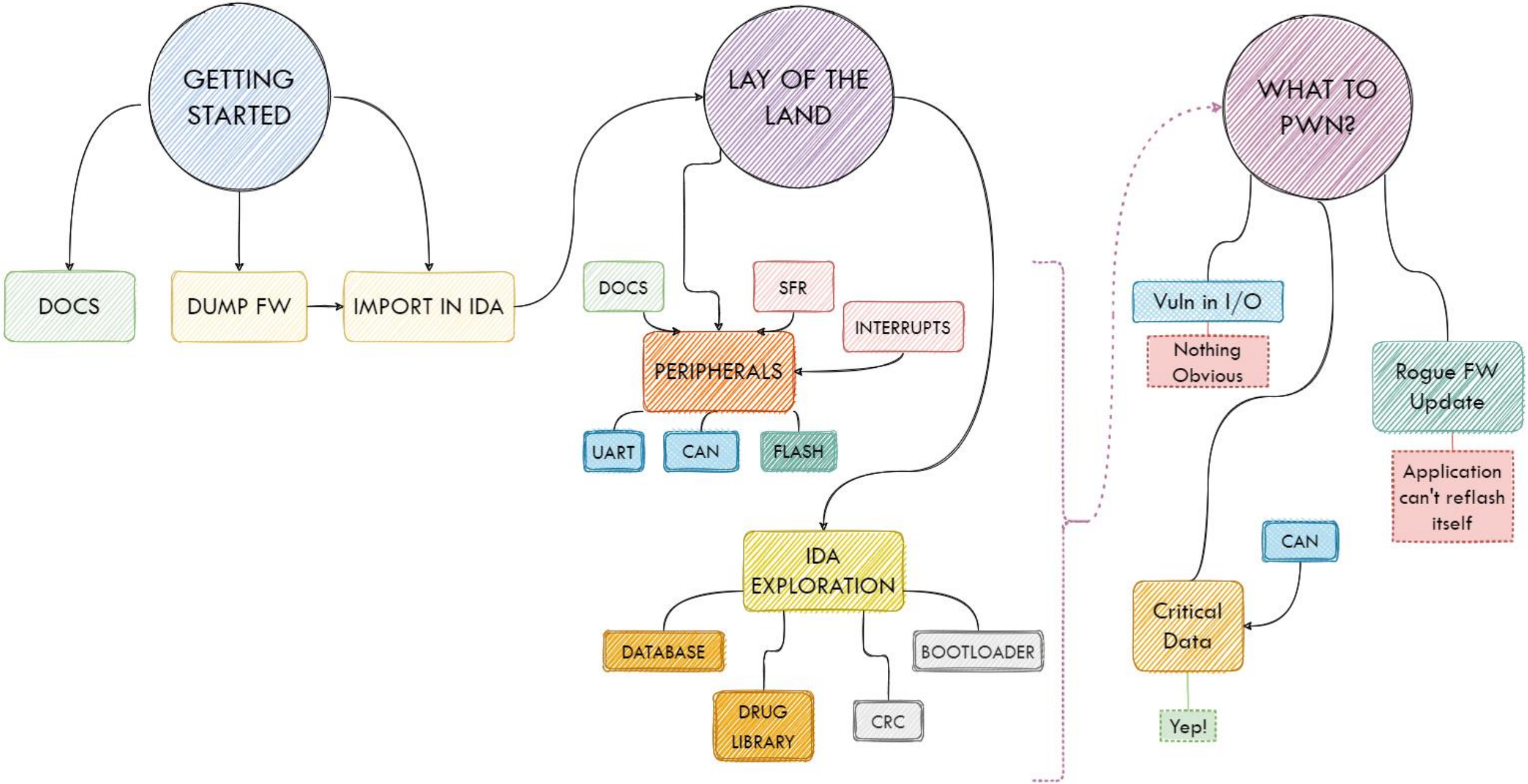

GETTING STARTED
DOCS
DUMP FW
IMPORT IN IDA
LAY OF THE LAND
DOCS
SFR
INTERRUPTS
PERIPHERALS
UART
CAN
FLASH
IDA EXPLORATION
DATABASE
DRUG LIBRARY
CRC
BOOTLOADER
WHAT TO PWN?
Vuln in I/O
Nothing Obvious
Rogue FW Update
Application can't reflash itself
CAN
Critical Data
Yep!


GETTING STARTED
DOCS
DUMP FW
IMPORT IN IDA
LAY OF THE LAND
DOCS
SFR
INTERRUPTS
PERIPHERALS
UART
CAN
FLASH
IDA EXPLORATION
DATABASE
DRUG LIBRARY
CRC
BOOTLOADER
WHAT TO PWN?
Vuln in I/O
Nothing Obvious
Rogue FW Update
Application can't reflash itself
CAN
Critical Data
Yep!


GETTING STARTED
DOCS
DUMP FW
IMPORT IN IDA
LAY OF THE LAND
DOCS
SFR
INTERRUPTS
PERIPHERALS
UART
CAN
FLASH
IDA EXPLORATION
DATABASE
DRUG LIBRARY
CRC
BOOTLOADER
WHAT TO PWN?
Vuln in I/O
Nothing Obvious
Rogue FW Update
Application can't reflash itself
CAN
Critical Data
Yep!


GETTING STARTED
DOCS
DUMP FW
IMPORT IN IDA
LAY OF THE LAND
DOCS
SFR
INTERRUPTS
PERIPHERALS
UART
CAN
FLASH
IDA EXPLORATION
DATABASE
DRUG LIBRARY
CRC
BOOTLOADER


WHAT TO PWN?
Vuln in I/O
Nothing Obvious
Rogue FW Update
Application can't reflash itself
CAN
Critical Data
Yep!


GETTING STARTED
DOCS
DUMP FW
IMPORT IN IDA
LAY OF THE LAND
DOCS
SFR
INTERRUPTS
PERIPHERALS
UART
CAN
FLASH
IDA EXPLORATION
DATABASE
DRUG LIBRARY
CRC
BOOTLOADER
WHAT TO PWN?
Vuln in I/O
Nothing Obvious
Rogue FW Update
Application can't reflash itself
CAN
Critical Data
Yep!

Conclusion

- Don’t be scared to tackle unfamiliar architectures!
- This RE effort was critical for the bigger project
  - Understanding the firmware helped crafting payloads to tamper with the device
  - We also reversed the thing it talks with over CAN
  - Found cool bugs! Full story at: https://www.trellix.com/en-us/about/newsroom/stories/threat-labs/mcafee-enterprise-atr-uncovers-vulnerabilities-in-globally-used-b-braun-infusion-pump.html

Questions?

- Now or around DC 🤜🤛 ☺
- Later on Twitter: @phLaul