Reversing FreeRTOS on embedded devices

Vitor Ventura & Vladan Nikolic
IBM X-Force Red EMEA Team

27th January 2017

Vitor Ventura

Senior Managing Security Consultant
IBM X-Force Red EMEA

Malware reverse Engineer
Penetration Tester
Blah
Blah blah
Blah blah blah

Twiter: @__VVentura

About us - Vladan

- Senior Managing Security Consultant in IBM EMEA XFR team
- 20+ years of experience with electronics and IT
- Embedded development, reverse engineering and ethical hacking

IBM X-Force Red

CISSP® Certified Information Systems Security Professional

OFFENSIVE security OSCE

CISCO CERTIFIED CCNA ROUTING & SWITCHING

CERTIFIED SCADA SECURITY ARCHITECT IACRB

JUNIPER NETWORKS CERTIFIED SPECIALIST JNCIS-SEC

Disclamer

- This presentation represents our own views on the topics discussed and doesn’t represent IBM position.
- All trademarks and copyrights are acknowledged.

Why?

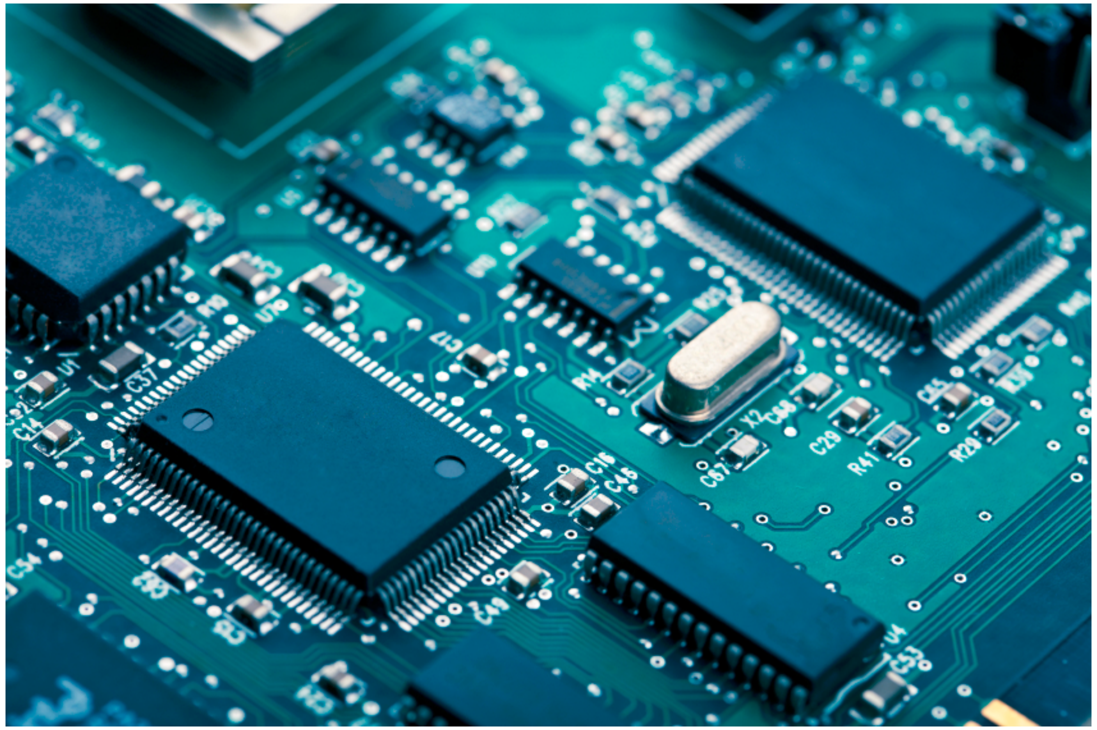
- Recent project challenges
- Interesting findings
- We believe it will be useful for community to share

From the desktop…

- Desktop Intel based platform is around for a very long time
- A lot of skills, tools and techniques developed for it
- 2 major flavors – x86 and x64 with some extensions(MMX, SSE, AVS…)
- Hardware is abstracted by OS and drivers

…To Embedded

- Usually around some micro CPU
- There are a lot of choices(PIC, AVR, Intel, MIPS, ESP…)
- Very common cores are ARM Cortex M0, M3 and M4 based
- Those devices comes with a lot of peripherals to support virtually any need
- Need to develop both hardware and software side

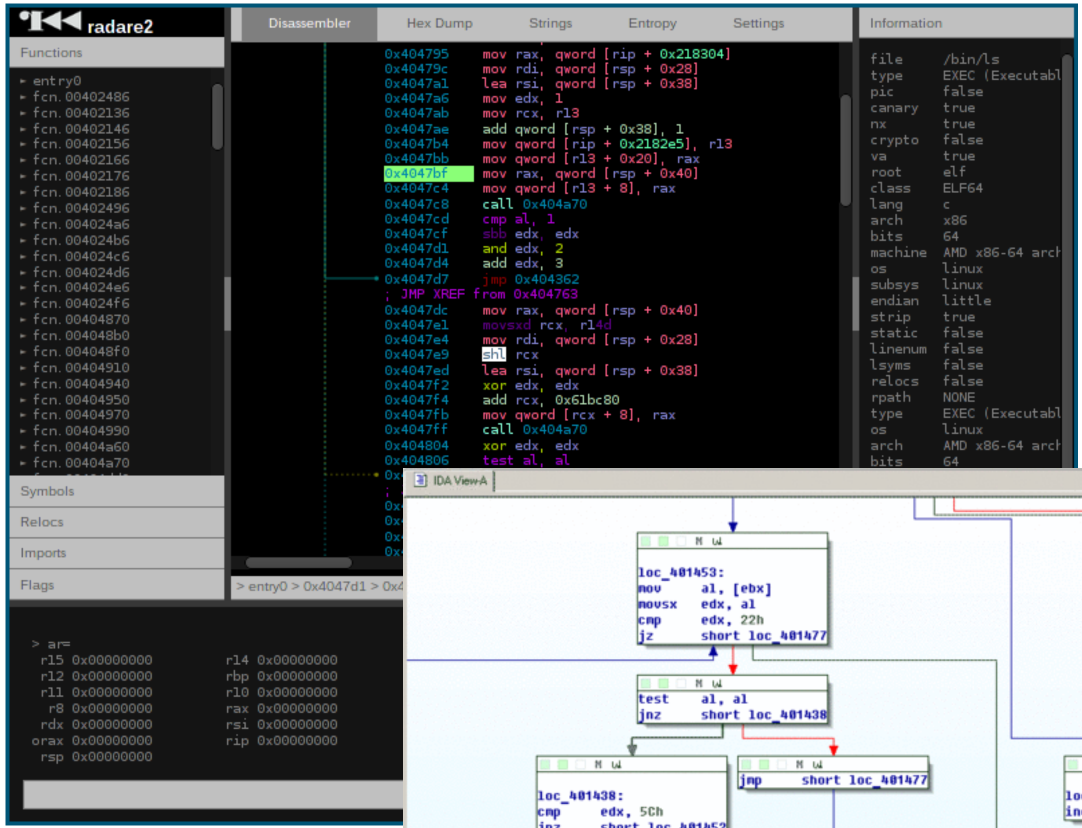


Everywhere

- IoT devices are massively deployed
- Previously isolated devices becomes connected
- With expanded capabilities
- Like SCADA systems
- Even cars

Tools of choice

- IDA Pro
- Capstone
- Hex editors

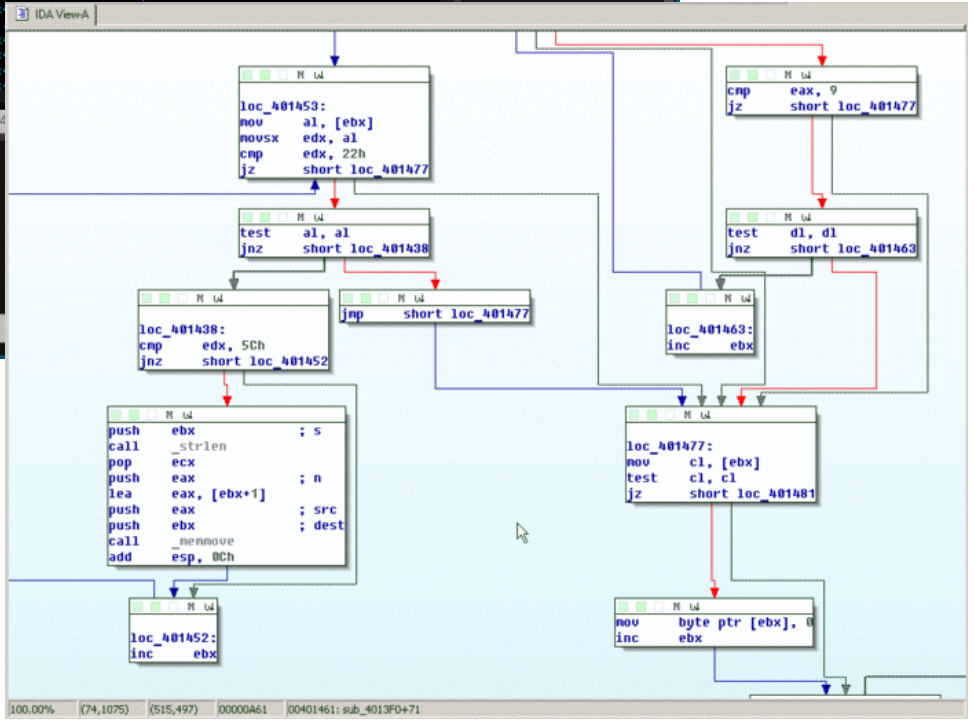


```
entry0
fcn.00402486
fcn.00402136
fcn.00402146
fcn.00402156
fcn.00402166
fcn.00402176
fcn.00402186
fcn.00402496
fcn.004024a6
fcn.004024b6
fcn.004024c6
fcn.004024d6
fcn.004024e6
fcn.004024f6
fcn.00404870
fcn.004048b0
fcn.004048f0
fcn.00404910
fcn.00404940
fcn.00404950
fcn.00404970
fcn.00404990
fcn.00404a60
fcn.00404a70
```

```
0x404795   mov rax, qword [rip + 0x218304]
0x40479c   mov rdi, qword [rsp + 0x28]
0x4047a1   lea rsi, qword [rsp + 0x38]
0x4047a6   mov edx, 1
0x4047ab   mov rcx, r13
0x4047ae   add qword [rsp + 0x38], 1
0x4047b4   mov qword [rip + 0x2182e5], r13
0x4047bb   mov qword [r13 + 0x20], rax
0x4047bf   mov rax, qword [rsp + 0x40]
0x4047c4   mov qword [r13 + 8], rax
0x4047c8   call 0x404a70
0x4047cd   cmp al, 1
0x4047cf   sbb edx, edx
0x4047d1   and edx, 2
0x4047d4   add edx, 3
0x4047d7   jmp 0x404362
; JMP XREF from 0x404763
0x4047dc   mov rax, qword [rsp + 0x40]
0x4047e1   movsxd rcx, r14d
0x4047e4   mov rdi, qword [rsp + 0x28]
0x4047e9   shl rcx
0x4047ed   lea rsi, qword [rsp + 0x38]
0x4047f2   xor edx, edx
0x4047f4   add rcx, 0x61bc80
0x4047fb   mov qword [rcx + 8], rax
0x4047ff   call 0x404a70
0x404804   xor edx, edx
0x404806   test al, al
```

```
file      /bin/ls
type      EXEC (Executabl
pic       false
canary    true
nx        true
crypto    false
va        true
root      elf
class     ELF64
lang      c
arch      x86
bits      64
machine   AMD x86-64 arch
os        linux
subsys    linux
endian    little
strip     true
static    false
linenum   false
lsyms     false
relocs    false
rpath     NONE
type      EXEC (Executabl
os        linux
arch      AMD x86-64 arch
bits      64
```

```
> ar=
 r15 0x00000000          r14 0x00000000
 r12 0x00000000          rbp 0x00000000
 r11 0x00000000          r10 0x00000000
  r8 0x00000000          rax 0x00000000
 rdx 0x00000000          rsi 0x00000000
orax 0x00000000          rip 0x00000000
 rsp 0x00000000
```

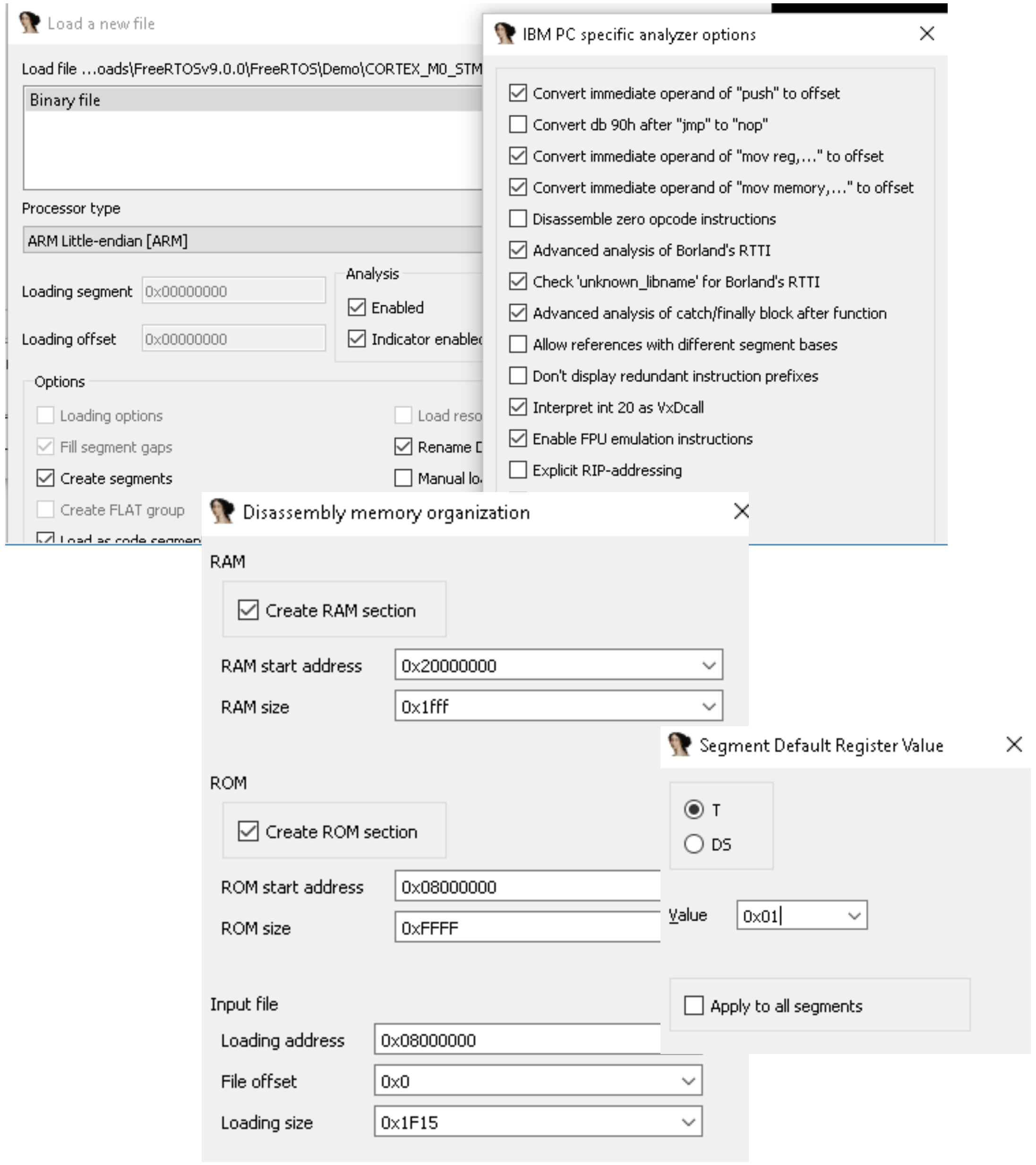


```
loc_401453:
mov     al, [ebx]
movsx   edx, al
cmp     edx, 22h
jz      short loc_401477

test    al, al
jnz     short loc_401438

loc_401438:
cmp     edx, 5Ch
jnz     short loc_401452

jmp     short loc_401477

push    ebx             ; s
call    _strlen
pop     ecx
push    eax             ; n
lea     eax, [ebx+1]
push    eax             ; src
push    ebx             ; dest
call    _memmove
add     esp, 0Ch

loc_401452:
inc     ebx

cmp     eax, 9
jz      short loc_401477

test    dl, dl
jnz     short loc_401463

loc_401463:
inc     ebx

loc_401477:
mov     cl, [ebx]
test    cl, cl
jz      short loc_401481

mov     byte ptr [ebx], 0
inc     ebx
```

```
100.00%  (74,1075)  (515,497)  00000A61  00401461: sub_4013F0+71
```

IDA Pro for embedded

- Excellent tool… but with some quirks
- Firmware loading – custom loader or manual
- Incomplete disassembly
- Problem with modes and instructions



Load a new file

Load file …oads\FreeRTOSv9.0.0\FreeRTOS\Demo\CORTEX_M0_STM

Binary file

Processor type

ARM Little-endian [ARM]

Loading segment 0x00000000

Loading offset 0x00000000

Analysis
[✓] Enabled
[✓] Indicator enabled

Options
[ ] Loading options
[✓] Fill segment gaps
[✓] Create segments
[ ] Create FLAT group
[✓] Load as code segmen
[ ] Load reso
[✓] Rename D
[ ] Manual lo

IBM PC specific analyzer options

[✓] Convert immediate operand of "push" to offset
[ ] Convert db 90h after "jmp" to "nop"
[✓] Convert immediate operand of "mov reg,…" to offset
[✓] Convert immediate operand of "mov memory,…" to offset
[ ] Disassemble zero opcode instructions
[✓] Advanced analysis of Borland's RTTI
[✓] Check 'unknown_libname' for Borland's RTTI
[✓] Advanced analysis of catch/finally block after function
[ ] Allow references with different segment bases
[ ] Don't display redundant instruction prefixes
[✓] Interpret int 20 as VxDcall
[✓] Enable FPU emulation instructions
[ ] Explicit RIP-addressing

Disassembly memory organization

RAM
[✓] Create RAM section
RAM start address 0x20000000
RAM size 0x1fff

ROM
[✓] Create ROM section
ROM start address 0x08000000
ROM size 0xFFFF

Input file
Loading address 0x08000000
File offset 0x0
Loading size 0x1F15

Segment Default Register Value

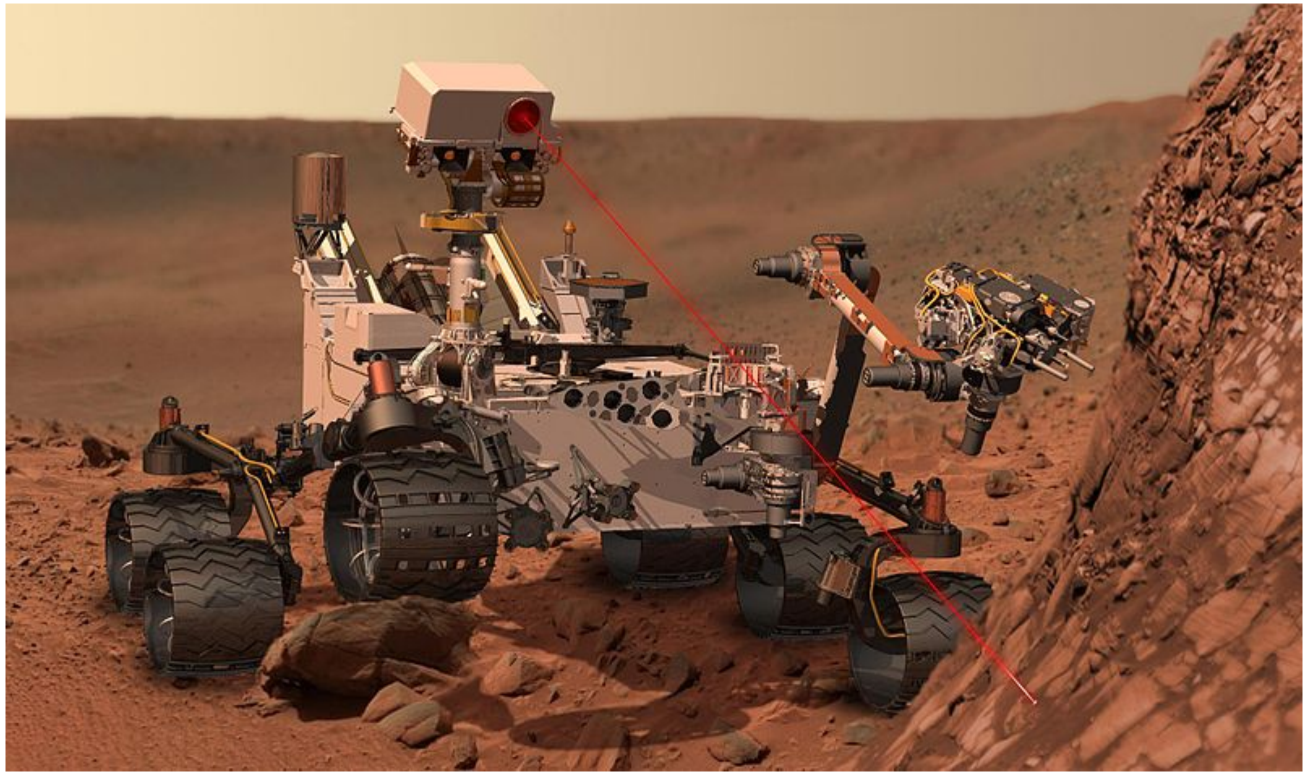
(●) T
( ) DS

Value 0x01

[ ] Apply to all segments

Main constraints

- Limited amount of memory and resources
- Power consumption awareness
- Real time responses
- Self sustainable and resilient



Common hardware issues

- JTAG & UART ports available
- Exposed busses (I^2C, IIS, Serial)
- Unprotected external storages (FLASH, sdcard…)
- Unprotected radio interfaces (WiFi, Bluetooth, microwaves…)
- Debugging consoles left active
- Unprotected bootloader and fw updates
- Fuses not set to make internal flash unreadable

Software Requirements

- Needs to be tailored to fit the hardware
- CPU speed, memory constrains, available storage
- Power consumption
- Error handling and bugs resilience
- Software developer needs to be versatile with the platform hardware developers designed.

- Common solution – choose some of numerous existing frameworks and RTOS which enables some level of hardware abstraction.

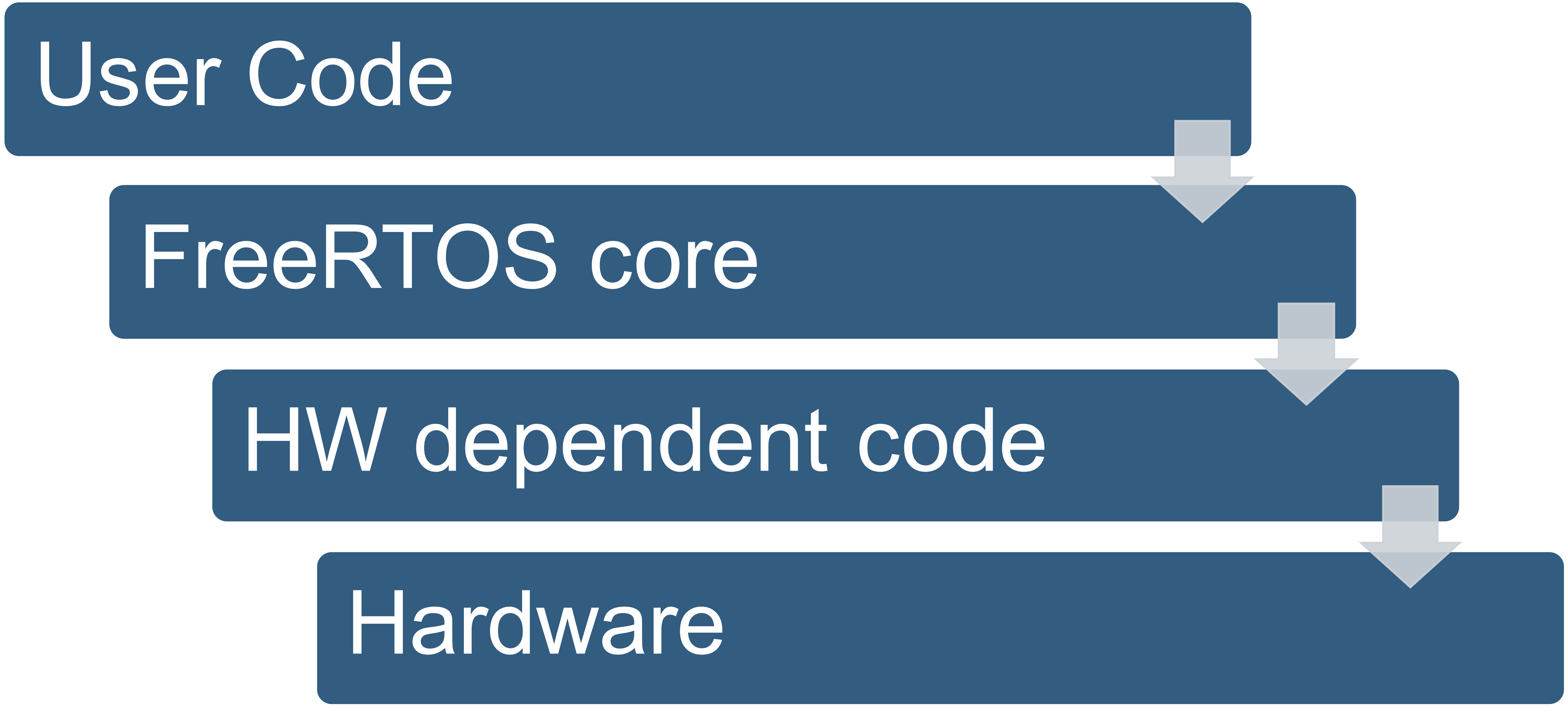
Introducing Free RTOS

- Small and very lean RTOS developed by Real Time Engineers Ltd
- Free and open source environment (there is a commercial version)
- Runs on almost everything (30+ platform supported oob)
- Yes, even Arduino
- Easy to customize for new platforms
  - port.c, portasm.s and portmacro.h needs to be ported for a new platform
- Widely supported by open source community
- Preemptive and cooperative multitasking
- Tickles mode of operation supported
- Tiny footprint
- More details: www.freertos.org

Supported high level functionalities

- Custom developed TCP and UDP IP stack
- FAT FS
- CLI
- I/O support including GPIO

FreeRTOS Structure

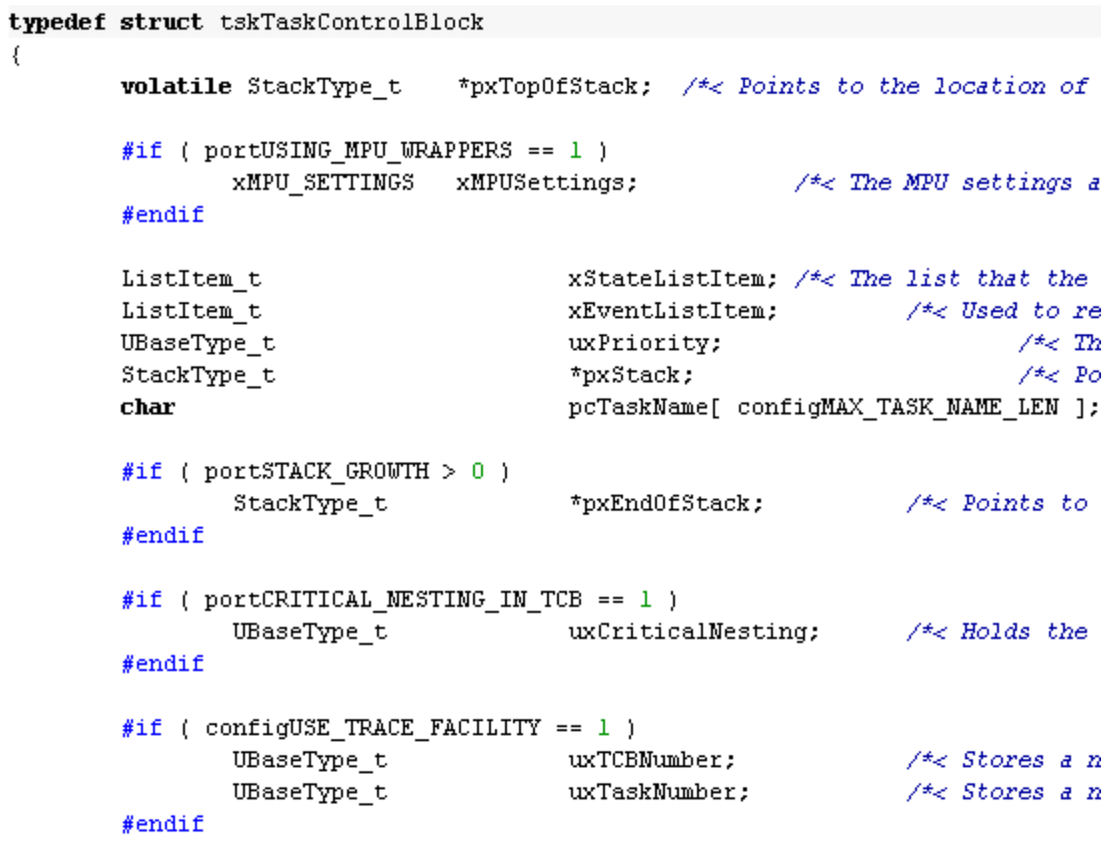


FreeRTOS main components

- Task Scheduler
- Tasks
  - independent piece of code which runs in its own context and with a separate stack under the Task Scheduler
- Co-routines
  - Not commonly used. All co-routines share the same stack with prioritized cooperative multitasking
- Data queues
- Semaphores & Mutexes
- Timers

Internals

- Heavily relies on double linked circular lists
- pxTaskReadyList contains a list of tasks that needs to be executed
- Every task has its own Task Control Block. TCB has a pointer to the stack allocated for the task.
- Queue is a list with additional pointers indicating where is the next read or write address.
- Semaphore is a specific instance of queue which doesn't track the data but the number of used elements in uxMessageWaiting field.
- Mutex is similar to semaphore, but head pointer is always 0 indicating it is a mutex and pointer to the task owning it is in the tail pointer.



```
typedef struct tskTaskControlBlock
{
        volatile StackType_t    *pxTopOfStack;  /*< Points to the location of

        #if ( portUSING_MPU_WRAPPERS == 1 )
                xMPU_SETTINGS   xMPUSettings;           /*< The MPU settings a
        #endif

        ListItem_t                      xStateListItem; /*< The list that the
        ListItem_t                      xEventListItem;         /*< Used to re
        UBaseType_t                     uxPriority;                     /*< Th
        StackType_t                     *pxStack;                       /*< Po
        char                            pcTaskName[ configMAX_TASK_NAME_LEN ];

        #if ( portSTACK_GROWTH > 0 )
                StackType_t             *pxEndOfStack;          /*< Points to
        #endif

        #if ( portCRITICAL_NESTING_IN_TCB == 1 )
                UBaseType_t             uxCriticalNesting;      /*< Holds the
        #endif

        #if ( configUSE_TRACE_FACILITY == 1 )
                UBaseType_t             uxTCBNumber;            /*< Stores a n
                UBaseType_t             uxTaskNumber;           /*< Stores a n
        #endif
```

Security Features overview

- By design, not much of them
- Since it is not designed as multitenant environment it lacks security controls we're used to on the desktop
- It supports:
  - Tasks with different privilege levels (only on ARM Cortex M3 with MPU enabled)
  - Stack overflow protection
  - SSL library as an add-on

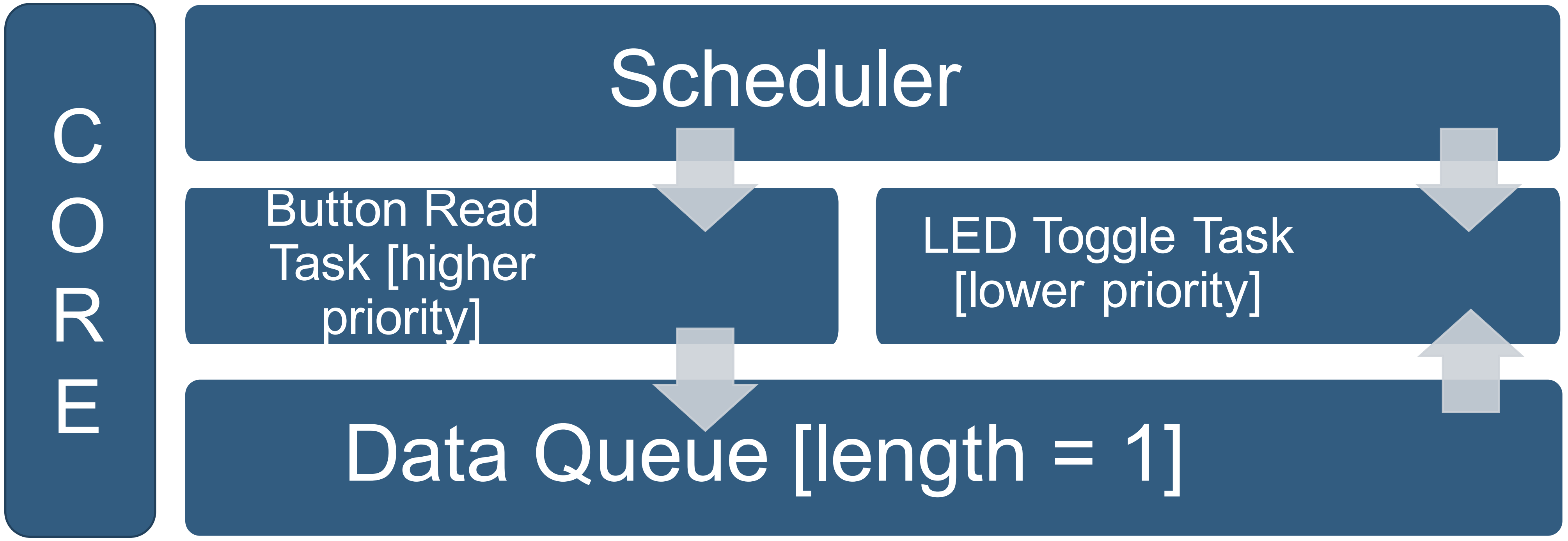
Security issues

- These are not real bugs in FreeRTOS, this is just observation from the point of adversary who wants to do some exploitation!
  - TCP/IP stack is not very resilient
  - Stack overflow protection is rudimentary
  - MPU usage is not very common (supported only on ARM M3 platforms anyway)
    - Unprivileged task can spawn privileged task, if MPU is used; or
    - Everything runs in the same context otherwise
  - It is developed in C inheriting all possible security problems as any other C programs (buffer overflows, heap corruptions…)

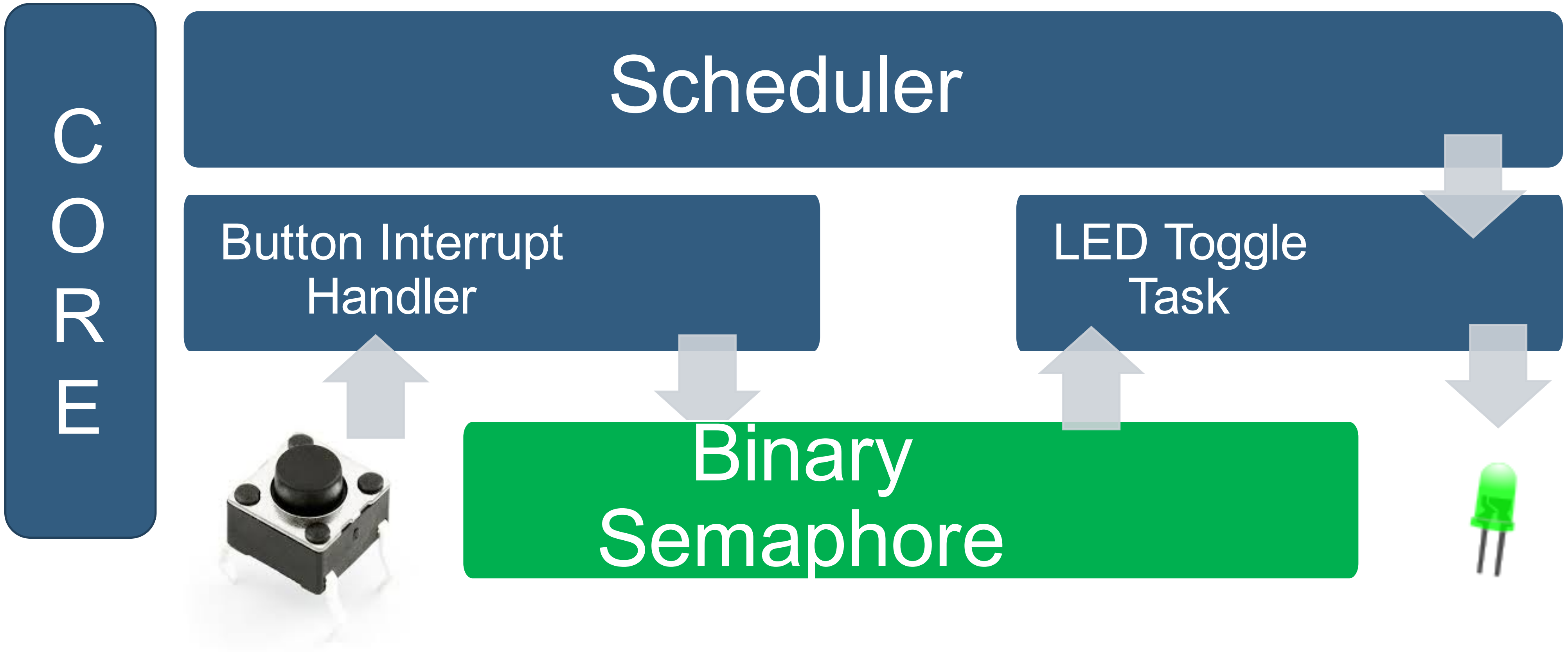
Sample application

- Sample => simple
- Goals
  - Get the button state
  - Toggle the LED if button is pushed
- Idea is to create a simple firmware for a device which will have MCU, a button and a LED.
- When button is pressed, LED will change its state.

Basic architecture


C O R E
Scheduler
Button Read Task [higher priority]
LED Toggle Task [lower priority]
Data Queue [length = 1]

Application architecture



Initialize

```
int main( void ){

    /* Setup hardware */
    STM_EVAL_LEDInit( LED1 );
    STM_EVAL_PBInit( BUTTON_KEY, BUTTON_MODE_EXTI );

    /* Create Semaphore  */
    vSemaphoreCreateBinary( xLedSemaphore  );

    /* Create Tasks */
    xTaskCreate( xLedSemaphoreHandler,  "LedSemaphoreHandlerTask",
configMINIMAL_STACK_SIZE, NULL, 3, NULL );

     /* Start Scheduler  */
    vTaskStartScheduler();

    return 1;
}
```

Run

```
// LED connected to GPIO port PC10
static void xLedSemaphoreHandler(void  *pvParameters){
    for ( ;; ) {
        xSemaphoreTake( xLedSemaphore, portMAX_DELAY );
        STM_EVAL_LEDToggle( LED1 );
        //GPIOWriteBit(GPIOC,GPIOPin10,Bit_SET); // turn on LED
    }
}

// Button connected to GPIO port PB8
static void EXTI4_15_IRQHandler(void ){
  if( EXTI_GetITStatus( KEY_BUTTON_EXTI_LINE ) != RESET )  {
    long lHigherPriorityTaskWoken;
    lHigherPriorityTaskWoken  = pdFALSE;
    xSemaphoreGiveFromISR ( xLedSemaphore,&lHigherPriorityTaskWoken  );
    portEND_SWITCHING_ISR( lHigherPriorityTaskWoken  );
    EXTI_ClearITPendingBit( KEY_BUTTON_EXTI_LINE );
  }
}
```

From source to hardware

- Wire up the platform – button and led must be where expected
- Compile for the specific platform
- Upload resulting image to the target MCU
- On reset, led should remain off
- Push the button, led should lit
- Push the button again and led should shut
- With minor changes in imports and ensuring button and led are on known positions as defined we can compile for any other platform

What is next?

- Now we have created our first embedded device
- …
- Profit ☺

Bad stuff will happen

- Somewhere in the world, dark forces are at works…
- Some people are trying to do some bad stuff to our valued product
- Since we made a hardware mistake, it was possible to dump the firmware from our device…

Reverse engineering on embedded systems

The good thing about embedded systems firmware:

It is that it’s deeply tied to the MCU

The bad thing about embedded systems firmware:

It is that it’s deeply tied to the MCU

Reverse engineering on Embedded

- String analysis does not help
- There are no syscalls on FreeRTOS
- There is no memory protection
- IDA by default will not detect the Entry point.

……… can we find the Entry point ?

YES

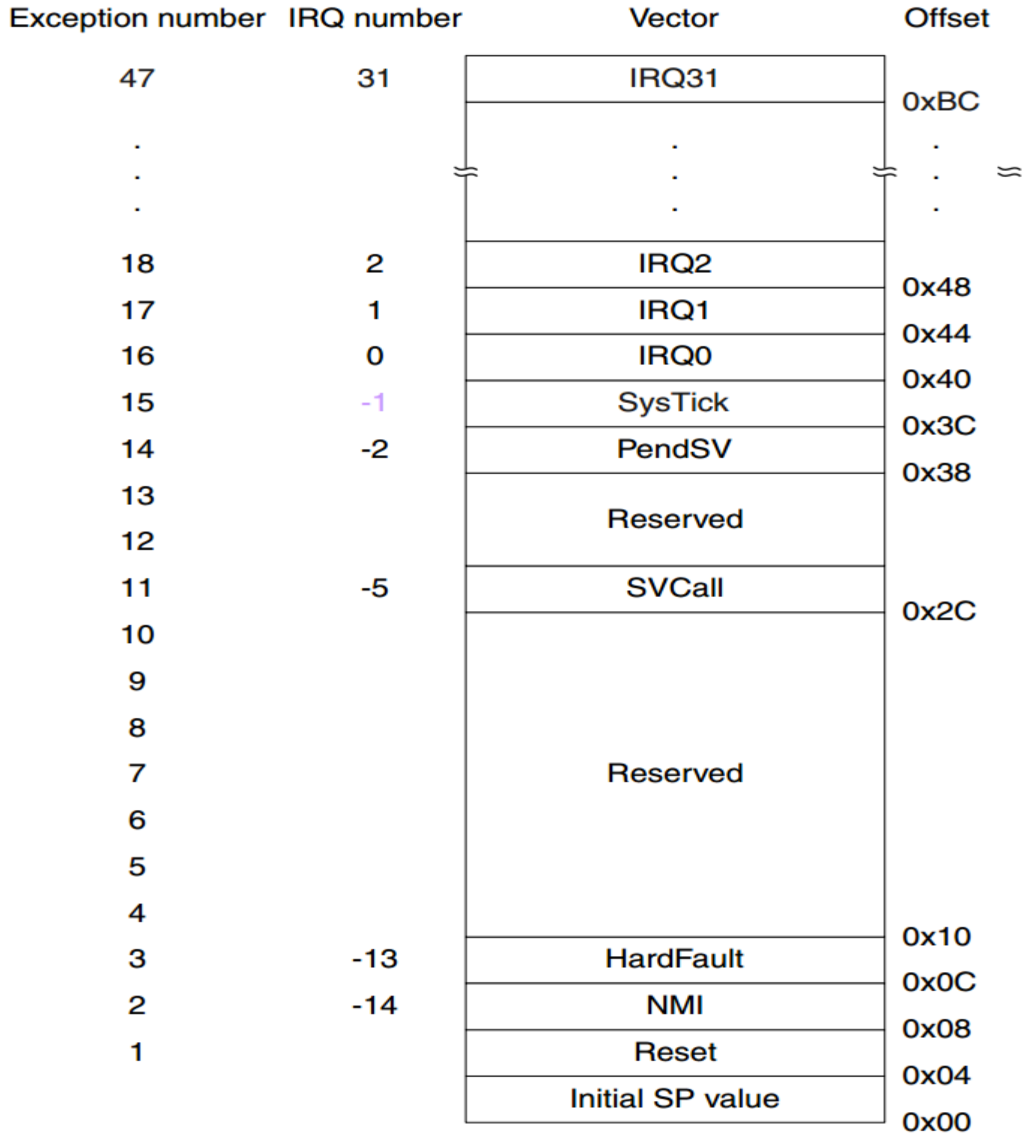
The entry point

- STM32 has some default interrupts which are controlled by handlers.
- In order to know where is each handler there is table called Interrupt Vector Table, which holds the address for each interrupt.
- One of these interrupts is the reset.
- What is boot rather then a reset interrupt?!

The entry point - Interrupt Vector Table (IVT)

This table contains the addresses of the routines that will handle some of the interrupts.

This table is located at offset 0x00.

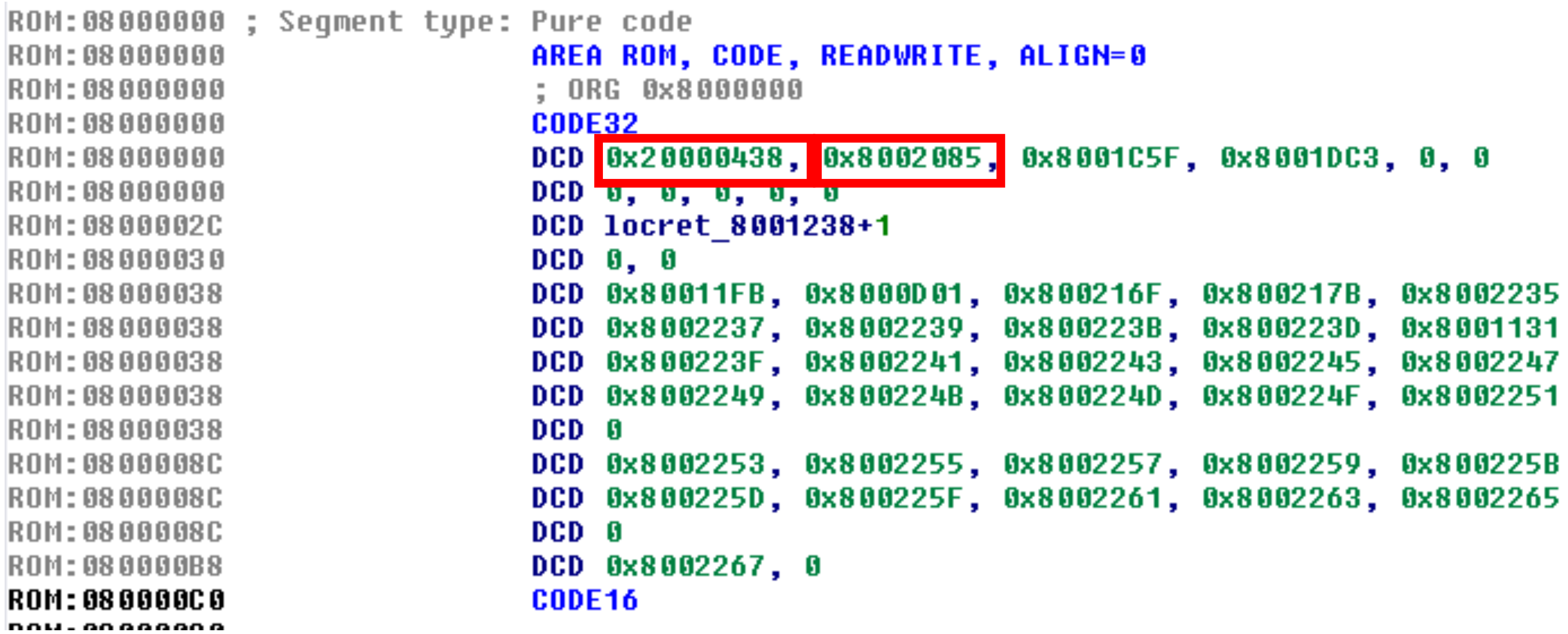


Exception number	IRQ number	Vector	Offset
47	31	IRQ31	0xBC
.		.	.
.		.	.
.		.	.
18	2	IRQ2	0x48
17	1	IRQ1	0x44
16	0	IRQ0	0x40
15	-1	SysTick	0x3C
14	-2	PendSV	0x38
13		Reserved	
12			
11	-5	SVCall	0x2C
10		Reserved	
9			
8			
7			
6			
5			
4			0x10
3	-13	HardFault	0x0C
2	-14	NMI	0x08
1		Reset	0x04
		Initial SP value	0x00

http://www.st.com/content/ccc/resource/technical/document/programming_manual/fc/90/c7/17/a1/44/43/89/DM00051352.pdf/files/DM00051352.pdf/jcr:content/translations/en.DM00051352.pdf

The entry point – IVT raw

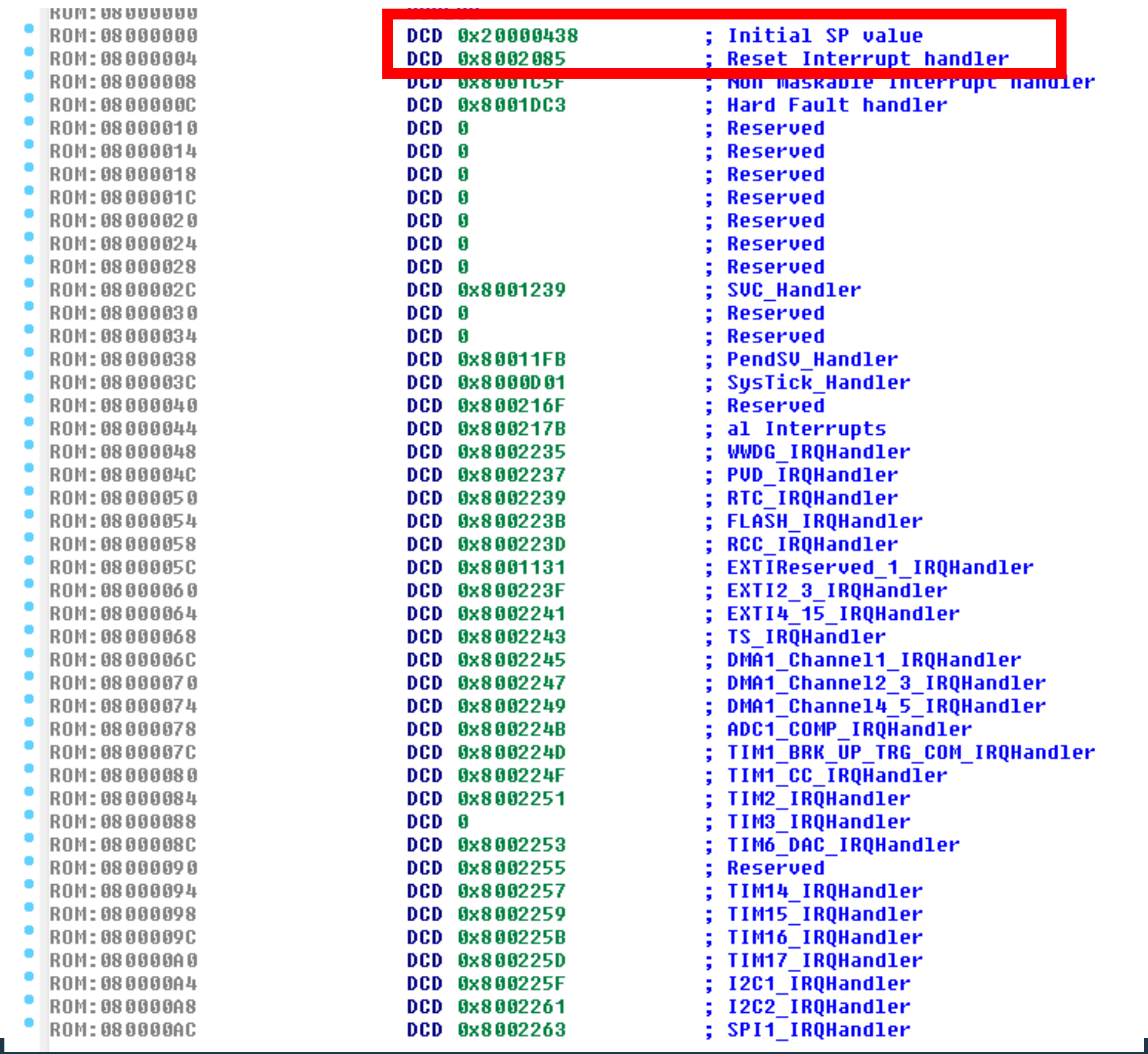
- Contents of the 0x00 offset of a FreeRTOS image



```
ROM:08000000 ; Segment type: Pure code
ROM:08000000                 AREA ROM, CODE, READWRITE, ALIGN=0
ROM:08000000                 ; ORG 0x8000000
ROM:08000000                 CODE32
ROM:08000000                 DCD 0x20000438, 0x8002085, 0x8001C5F, 0x8001DC3, 0, 0
ROM:08000000                 DCD 0, 0, 0, 0, 0
ROM:0800002C                 DCD locret_8001238+1
ROM:08000030                 DCD 0, 0
ROM:08000038                 DCD 0x80011FB, 0x8000D01, 0x800216F, 0x800217B, 0x8002235
ROM:08000038                 DCD 0x8002237, 0x8002239, 0x800223B, 0x800223D, 0x8001131
ROM:08000038                 DCD 0x800223F, 0x8002241, 0x8002243, 0x8002245, 0x8002247
ROM:08000038                 DCD 0x8002249, 0x800224B, 0x800224D, 0x800224F, 0x8002251
ROM:08000038                 DCD 0
ROM:0800008C                 DCD 0x8002253, 0x8002255, 0x8002257, 0x8002259, 0x800225B
ROM:0800008C                 DCD 0x800225D, 0x800225F, 0x8002261, 0x8002263, 0x8002265
ROM:0800008C                 DCD 0
ROM:080000B8                 DCD 0x8002267, 0
ROM:080000C0                 CODE16
```

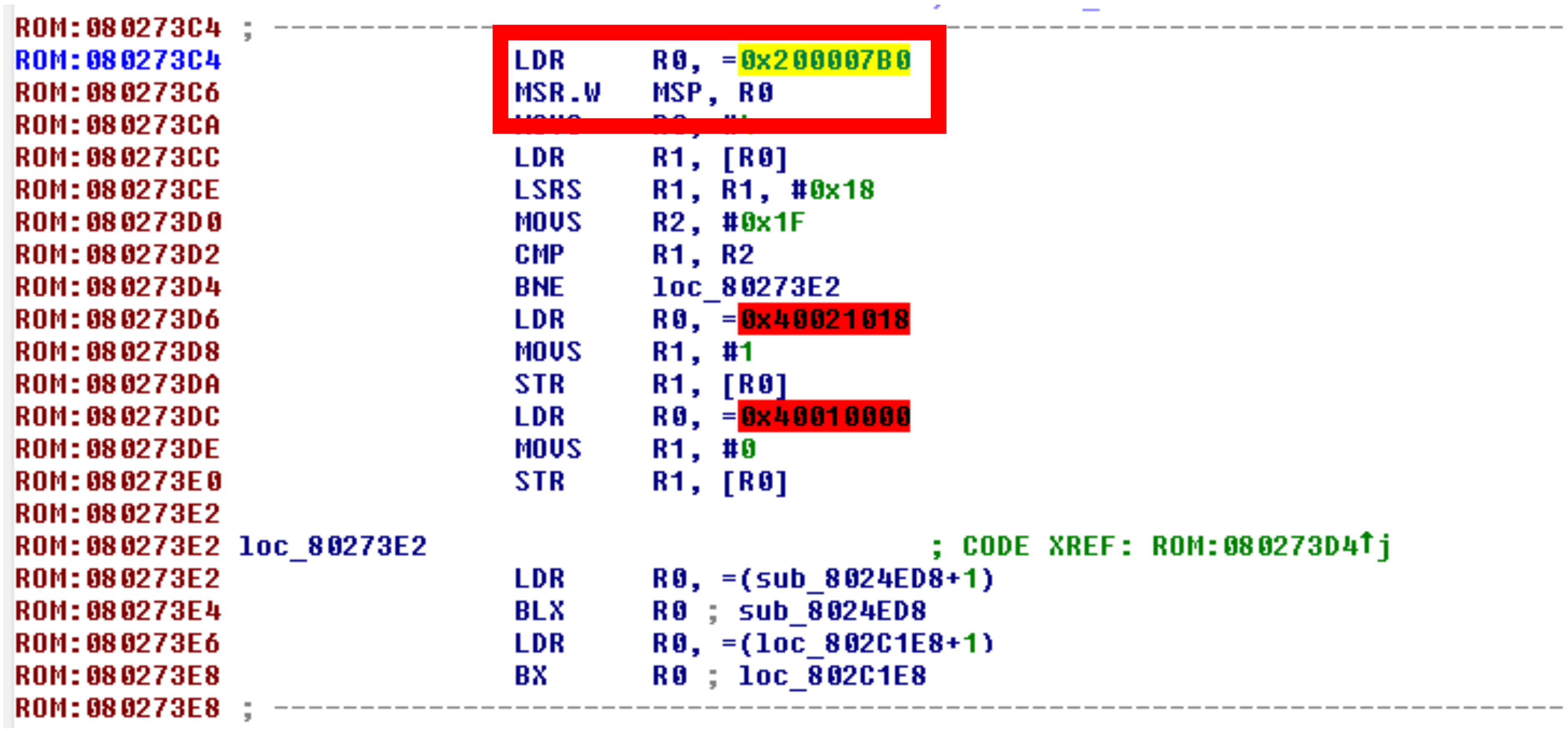
The entry point – IVT decoded

- The plugin packs the data into a table format
- And adds the comment for what is reserved



```
ROM:08000000
ROM:08000000                 DCD 0x20000438          ; Initial SP value
ROM:08000004                 DCD 0x8002085           ; Reset Interrupt handler
ROM:08000008                 DCD 0x8001C5F           ; Non maskable interrupt handler
ROM:0800000C                 DCD 0x8001DC3           ; Hard Fault handler
ROM:08000010                 DCD 0                   ; Reserved
ROM:08000014                 DCD 0                   ; Reserved
ROM:08000018                 DCD 0                   ; Reserved
ROM:0800001C                 DCD 0                   ; Reserved
ROM:08000020                 DCD 0                   ; Reserved
ROM:08000024                 DCD 0                   ; Reserved
ROM:08000028                 DCD 0                   ; Reserved
ROM:0800002C                 DCD 0x8001239           ; SVC_Handler
ROM:08000030                 DCD 0                   ; Reserved
ROM:08000034                 DCD 0                   ; Reserved
ROM:08000038                 DCD 0x80011FB           ; PendSV_Handler
ROM:0800003C                 DCD 0x8000D01           ; SysTick_Handler
ROM:08000040                 DCD 0x800216F           ; Reserved
ROM:08000044                 DCD 0x800217B           ; al Interrupts
ROM:08000048                 DCD 0x8002235           ; WWDG_IRQHandler
ROM:0800004C                 DCD 0x8002237           ; PVD_IRQHandler
ROM:08000050                 DCD 0x8002239           ; RTC_IRQHandler
ROM:08000054                 DCD 0x800223B           ; FLASH_IRQHandler
ROM:08000058                 DCD 0x800223D           ; RCC_IRQHandler
ROM:0800005C                 DCD 0x8001131           ; EXTIReserved_1_IRQHandler
ROM:08000060                 DCD 0x800223F           ; EXTI2_3_IRQHandler
ROM:08000064                 DCD 0x8002241           ; EXTI4_15_IRQHandler
ROM:08000068                 DCD 0x8002243           ; TS_IRQHandler
ROM:0800006C                 DCD 0x8002245           ; DMA1_Channel1_IRQHandler
ROM:08000070                 DCD 0x8002247           ; DMA1_Channel2_3_IRQHandler
ROM:08000074                 DCD 0x8002249           ; DMA1_Channel4_5_IRQHandler
ROM:08000078                 DCD 0x800224B           ; ADC1_COMP_IRQHandler
ROM:0800007C                 DCD 0x800224D           ; TIM1_BRK_UP_TRG_COM_IRQHandler
ROM:08000080                 DCD 0x800224F           ; TIM1_CC_IRQHandler
ROM:08000084                 DCD 0x8002251           ; TIM2_IRQHandler
ROM:08000088                 DCD 0                   ; TIM3_IRQHandler
ROM:0800008C                 DCD 0x8002253           ; TIM6_DAC_IRQHandler
ROM:08000090                 DCD 0x8002255           ; Reserved
ROM:08000094                 DCD 0x8002257           ; TIM14_IRQHandler
ROM:08000098                 DCD 0x8002259           ; TIM15_IRQHandler
ROM:0800009C                 DCD 0x800225B           ; TIM16_IRQHandler
ROM:080000A0                 DCD 0x800225D           ; TIM17_IRQHandler
ROM:080000A4                 DCD 0x800225F           ; I2C1_IRQHandler
ROM:080000A8                 DCD 0x8002261           ; I2C2_IRQHandler
ROM:080000AC                 DCD 0x8002263           ; SPI1_IRQHandler
```

The entry point – Reset Handler



```
ROM:080273C4 ; ---------------------------------------------------------------------------
ROM:080273C4                 LDR     R0, =0x200007B0
ROM:080273C6                 MSR.W   MSP, R0
ROM:080273CA                 [illegible]
ROM:080273CC                 LDR     R1, [R0]
ROM:080273CE                 LSRS    R1, R1, #0x18
ROM:080273D0                 MOVS    R2, #0x1F
ROM:080273D2                 CMP     R1, R2
ROM:080273D4                 BNE     loc_80273E2
ROM:080273D6                 LDR     R0, =0x40021018
ROM:080273D8                 MOVS    R1, #1
ROM:080273DA                 STR     R1, [R0]
ROM:080273DC                 LDR     R0, =0x40010000
ROM:080273DE                 MOVS    R1, #0
ROM:080273E0                 STR     R1, [R0]
ROM:080273E2
ROM:080273E2 loc_80273E2                             ; CODE XREF: ROM:080273D4↑j
ROM:080273E2                 LDR     R0, =(sub_8024ED8+1)
ROM:080273E4                 BLX     R0 ; sub_8024ED8
ROM:080273E6                 LDR     R0, =(loc_802C1E8+1)
ROM:080273E8                 BX      R0 ; loc_802C1E8
ROM:080273E8 ; ---------------------------------------------------------------------------
```

Now we have an entry point

Reverse engineering on Embedded

- Now we have an entry point.
- all peripheral access is done by reading and writing into specific memory addresses.
  - Address ranges and offsets are mapped to the MCU buses.

….. so can these ranges and offsets be useful?

YES

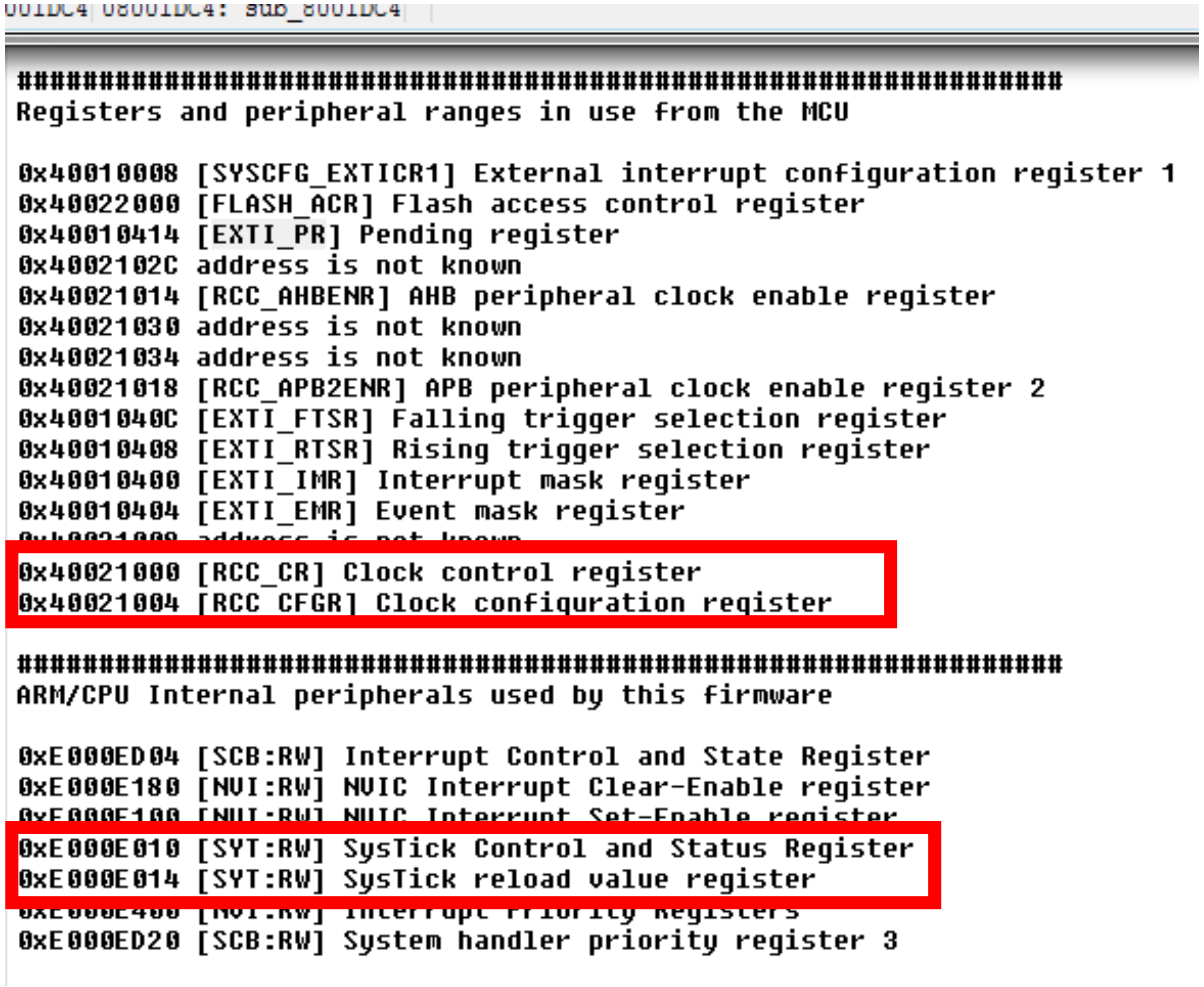
Reverse engineering on STM32F0

- The MCU documentation will contain the registers addresses and their functions.

- How does the plugin help?

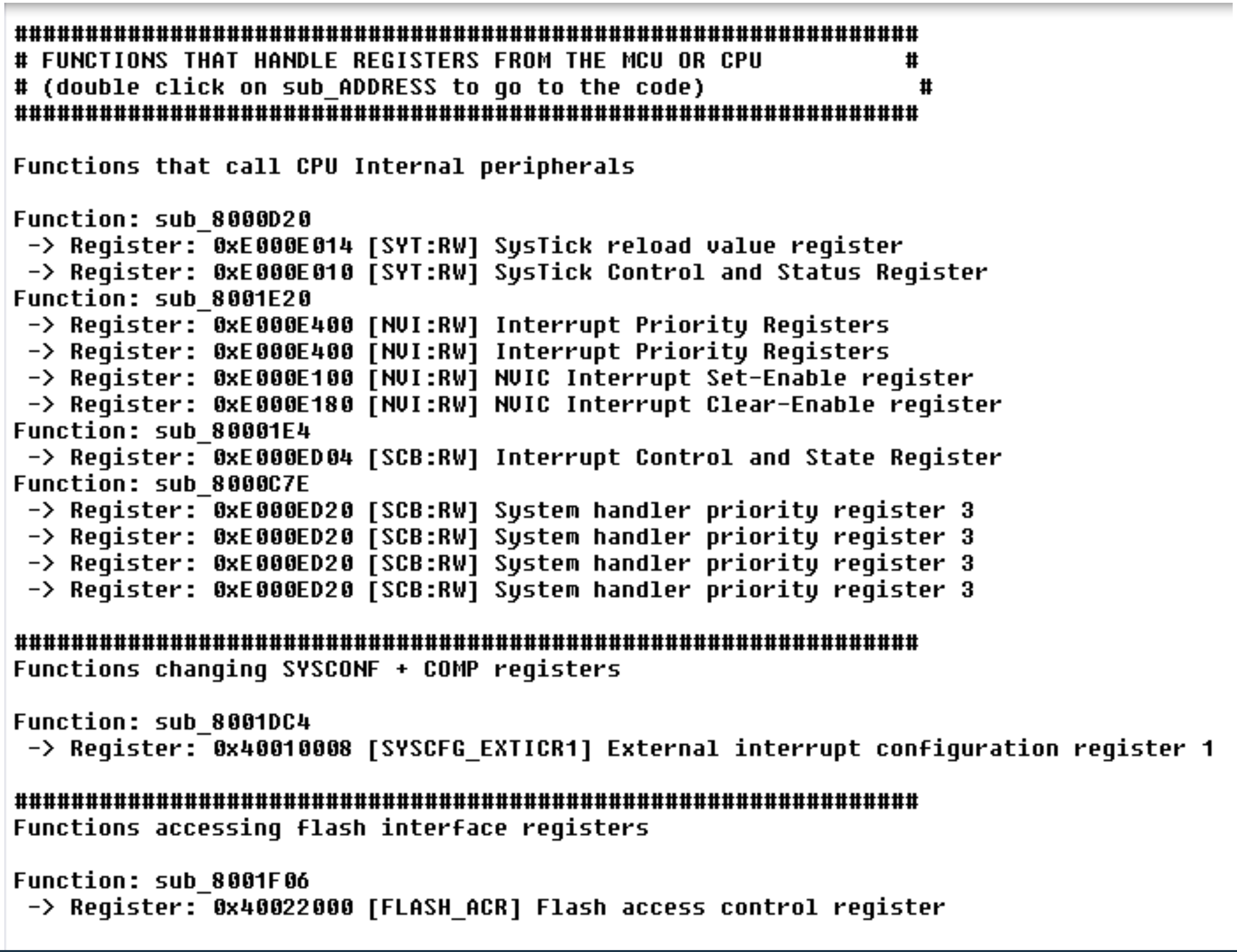
- It:
  - Lists the registers manipulated
  - Lists functions that manipulate each register
  - Adds comments to the code with description of each register

IDA Plugin - Registers descriptions



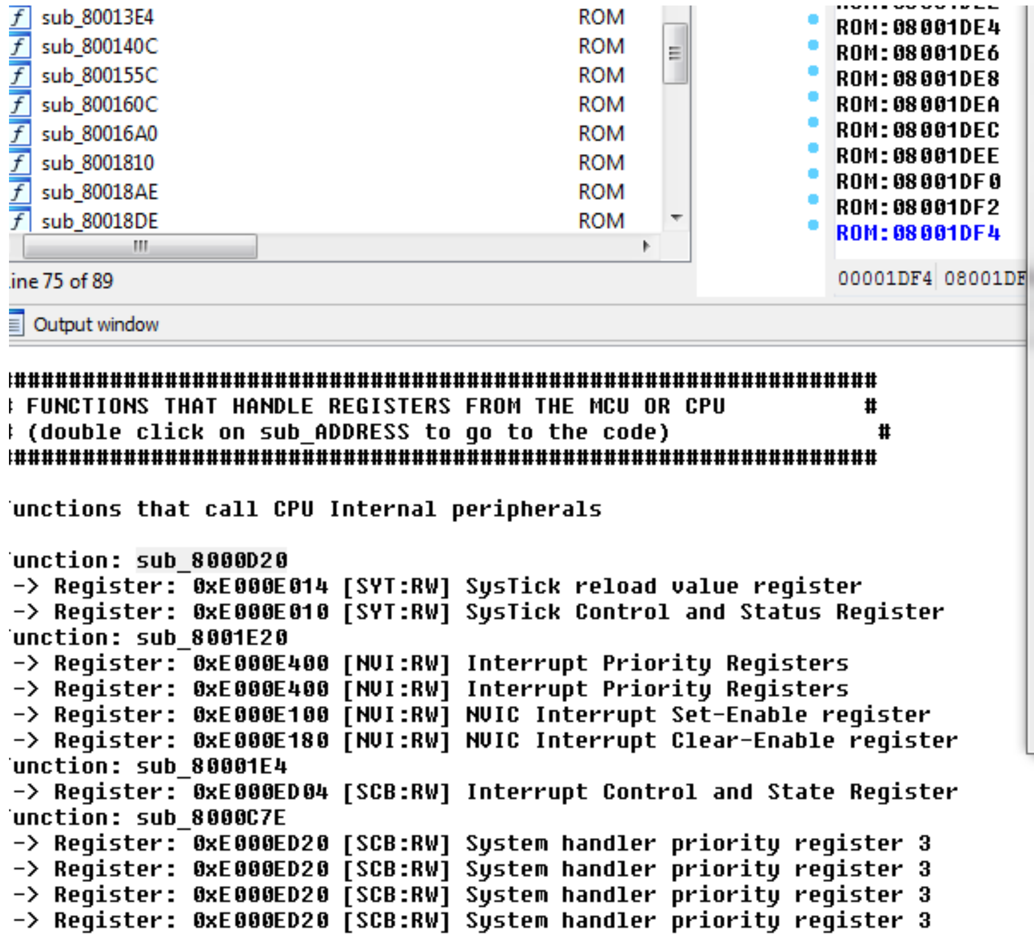
```
001DC4 08001DC4: sub_8001DC4

################################################################
Registers and peripheral ranges in use from the MCU

0x40010008 [SYSCFG_EXTICR1] External interrupt configuration register 1
0x40022000 [FLASH_ACR] Flash access control register
0x40010414 [EXTI_PR] Pending register
0x4002102C address is not known
0x40021014 [RCC_AHBENR] AHB peripheral clock enable register
0x40021030 address is not known
0x40021034 address is not known
0x40021018 [RCC_APB2ENR] APB peripheral clock enable register 2
0x4001040C [EXTI_FTSR] Falling trigger selection register
0x40010408 [EXTI_RTSR] Rising trigger selection register
0x40010400 [EXTI_IMR] Interrupt mask register
0x40010404 [EXTI_EMR] Event mask register
0x40021008 address is not known
0x40021000 [RCC_CR] Clock control register
0x40021004 [RCC_CFGR] Clock configuration register

################################################################
ARM/CPU Internal peripherals used by this firmware

0xE000ED04 [SCB:RW] Interrupt Control and State Register
0xE000E180 [NVI:RW] NVIC Interrupt Clear-Enable register
0xE000E100 [NVI:RW] NVIC Interrupt Set-Enable register
0xE000E010 [SYT:RW] SysTick Control and Status Register
0xE000E014 [SYT:RW] SysTick reload value register
0xE000E400 [NVI:RW] Interrupt Priority Registers
0xE000ED20 [SCB:RW] System handler priority register 3
```

IDA Plugin – Functions manipulating registers



```
##################################################################
# FUNCTIONS THAT HANDLE REGISTERS FROM THE MCU OR CPU           #
# (double click on sub_ADDRESS to go to the code)                #
##################################################################

Functions that call CPU Internal peripherals

Function: sub_8000D20
 -> Register: 0xE000E014 [SYT:RW] SysTick reload value register
 -> Register: 0xE000E010 [SYT:RW] SysTick Control and Status Register
Function: sub_8001E20
 -> Register: 0xE000E400 [NVI:RW] Interrupt Priority Registers
 -> Register: 0xE000E400 [NVI:RW] Interrupt Priority Registers
 -> Register: 0xE000E100 [NVI:RW] NVIC Interrupt Set-Enable register
 -> Register: 0xE000E180 [NVI:RW] NVIC Interrupt Clear-Enable register
Function: sub_80001E4
 -> Register: 0xE000ED04 [SCB:RW] Interrupt Control and State Register
Function: sub_8000C7E
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3

##################################################################
Functions changing SYSCONF + COMP registers

Function: sub_8001DC4
 -> Register: 0x40010008 [SYSCFG_EXTICR1] External interrupt configuration register 1

##################################################################
Functions accessing flash interface registers

Function: sub_8001F06
 -> Register: 0x40022000 [FLASH_ACR] Flash access control register
```

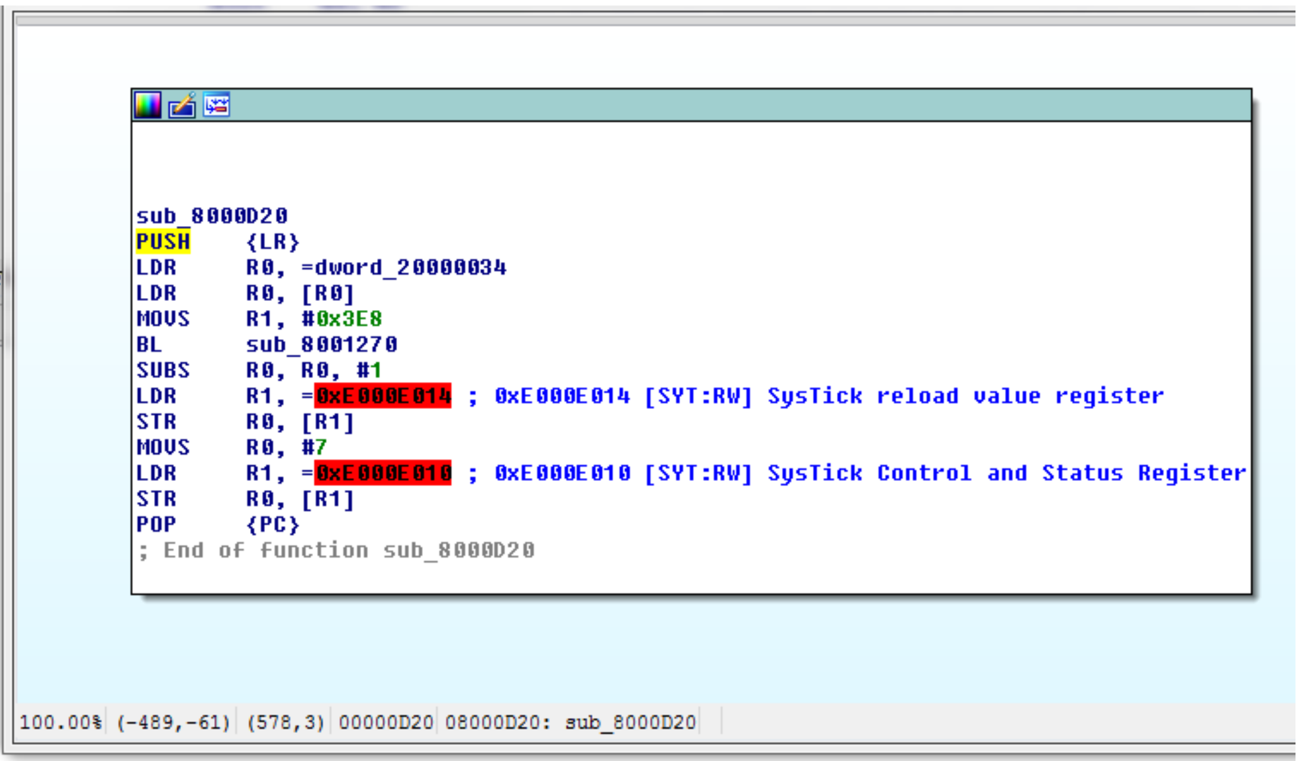
IDA Plugin – Comments on the code



```
sub_80013E4     ROM          ROM:08001DE4
sub_800140C     ROM          ROM:08001DE6
sub_800155C     ROM          ROM:08001DE8
sub_800160C     ROM          ROM:08001DEA
sub_80016A0     ROM          ROM:08001DEC
sub_8001810     ROM          ROM:08001DEE
sub_80018AE     ROM          ROM:08001DF0
sub_80018DE     ROM          ROM:08001DF2
                             ROM:08001DF4
ine 75 of 89                 00001DF4 08001DF
```

Output window

```
################################################################
 FUNCTIONS THAT HANDLE REGISTERS FROM THE MCU OR CPU          #
 (double click on sub_ADDRESS to go to the code)              #
################################################################

unctions that call CPU Internal peripherals

unction: sub_8000D20
 -> Register: 0xE000E014 [SYT:RW] SysTick reload value register
 -> Register: 0xE000E010 [SYT:RW] SysTick Control and Status Register
unction: sub_8001E20
 -> Register: 0xE000E400 [NVI:RW] Interrupt Priority Registers
 -> Register: 0xE000E400 [NVI:RW] Interrupt Priority Registers
 -> Register: 0xE000E100 [NVI:RW] NVIC Interrupt Set-Enable register
 -> Register: 0xE000E180 [NVI:RW] NVIC Interrupt Clear-Enable register
unction: sub_80001E4
 -> Register: 0xE000ED04 [SCB:RW] Interrupt Control and State Register
unction: sub_8000C7E
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3
```



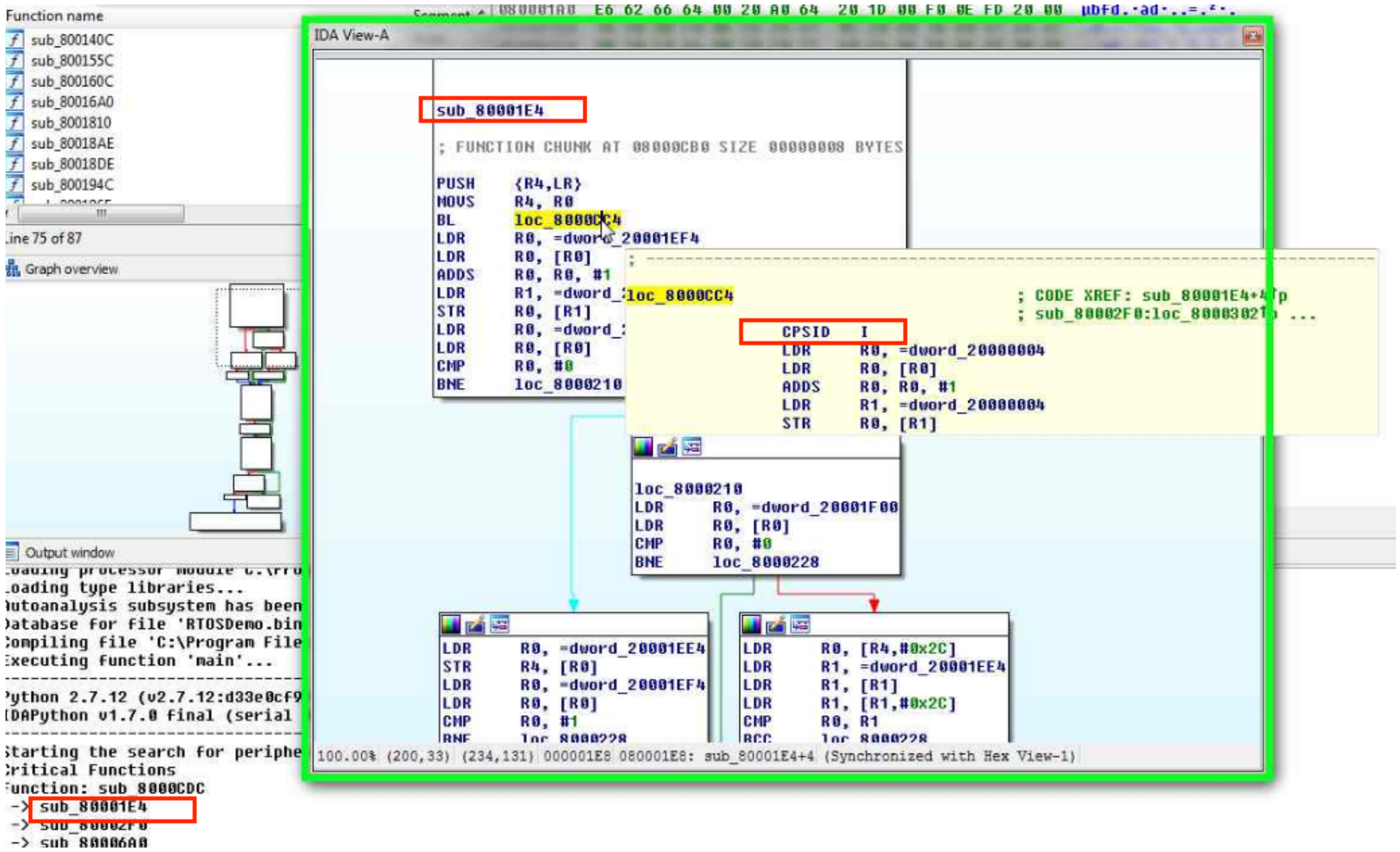
```
sub_8000D20
PUSH    {LR}
LDR     R0, =dword_20000034
LDR     R0, [R0]
MOVS    R1, #0x3E8
BL      sub_8001270
SUBS    R0, R0, #1
LDR     R1, =0xE000E014 ; 0xE000E014 [SYT:RW] SysTick reload value register
STR     R0, [R1]
MOVS    R0, #7
LDR     R1, =0xE000E010 ; 0xE000E010 [SYT:RW] SysTick Control and Status Register
STR     R0, [R1]
POP     {PC}
; End of function sub_8000D20
```

```
100.00% (-489,-61) (578,3) 00000D20 08000D20: sub_8000D20
```

Reverse engineering on STM32F0

- Generically RTOS need to define critical code areas where the interrupts cannot break the execution flow.
- This is done by using the ARM CPSID and CPSIE instructions.
- So a good place to start looking in your code is before CPSIE instruction.

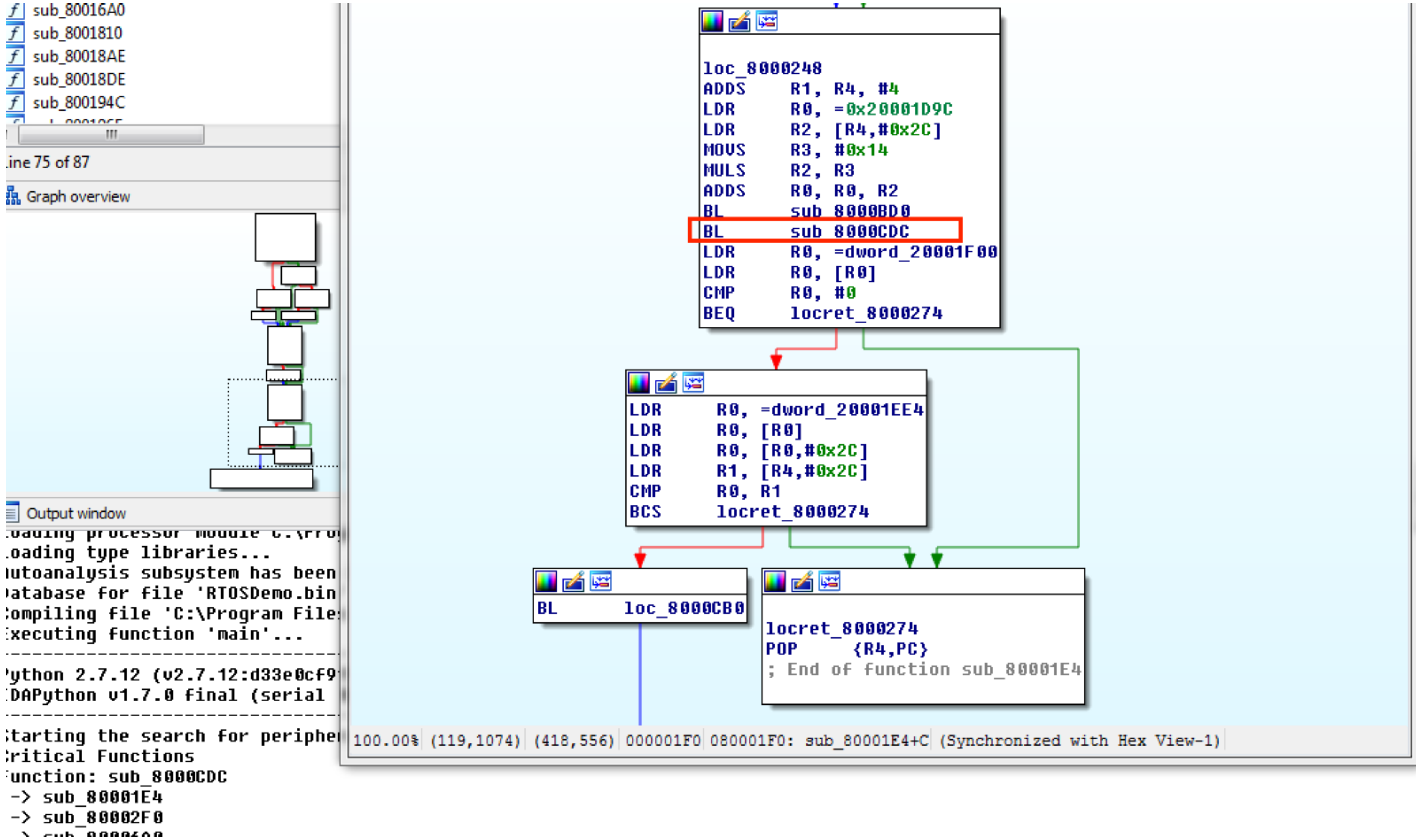
Critical code decoding and listing



```
080001A8  E6 62 66 64 00 20 A0 64  20 1D 00 F0 0E FD 20 00  µbfd.·ad·..=.²·.

sub_80001E4

; FUNCTION CHUNK AT 08000CB0 SIZE 00000008 BYTES

PUSH    {R4,LR}
MOVS    R4, R0
BL      loc_8000CC4
LDR     R0, =dword_20001EF4
LDR     R0, [R0]
ADDS    R0, R0, #1
LDR     R1, =dword_
STR     R0, [R1]
LDR     R0, =dword_
LDR     R0, [R0]
CMP     R0, #0
BNE     loc_8000210

; ---------------------------------------------------------------
loc_8000CC4                          ; CODE XREF: sub_80001E4+4↑p
                                     ; sub_80002F0:loc_8000302↑p ...
        CPSID   I
        LDR     R0, =dword_20000004
        LDR     R0, [R0]
        ADDS    R0, R0, #1
        LDR     R1, =dword_20000004
        STR     R0, [R1]

loc_8000210
LDR     R0, =dword_20001F00
LDR     R0, [R0]
CMP     R0, #0
BNE     loc_8000228

LDR     R0, =dword_20001EE4
STR     R4, [R0]
LDR     R0, =dword_20001EF4
LDR     R0, [R0]
CMP     R0, #1
BNE     loc_8000228

LDR     R0, [R4,#0x2C]
LDR     R1, =dword_20001EE4
LDR     R1, [R1]
LDR     R1, [R1,#0x2C]
CMP     R0, R1
BCC     loc_8000228

100.00% (200,33) (234,131) 000001E8 080001E8: sub_80001E4+4 (Synchronized with Hex View-1)
```

```
Loading processor module C:\Pro
Loading type libraries...
Autoanalysis subsystem has been
Database for file 'RTOSDemo.bin
Compiling file 'C:\Program File
Executing function 'main'...
--------------------------------
Python 2.7.12 (v2.7.12:d33e0cf9
IDAPython v1.7.0 final (serial
--------------------------------
Starting the search for periphe
Critical Functions
Function: sub_8000CDC
 -> sub_80001E4
 -> sub_80002F0
 -> sub_80006A0
```

Critical code decoding and listing



```
sub_80016A0
sub_8001810
sub_80018AE
sub_80018DE
sub_800194C
```

ine 75 of 87

Graph overview

Output window

```
oading processor module C:\Pro
oading type libraries...
utoanalysis subsystem has been
atabase for file 'RTOSDemo.bin
ompiling file 'C:\Program File
xecuting function 'main'...
-----------------------------------
ython 2.7.12 (v2.7.12:d33e0cf9
DAPython v1.7.0 final (serial
-----------------------------------
tarting the search for periphe
ritical Functions
unction: sub_8000CDC
 -> sub_80001E4
 -> sub_80002F0
```

```
loc_8000248
ADDS    R1, R4, #4
LDR     R0, =0x20001D9C
LDR     R2, [R4,#0x2C]
MOVS    R3, #0x14
MULS    R2, R3
ADDS    R0, R0, R2
BL      sub_8000BD0
BL      sub_8000CDC
LDR     R0, =dword_20001F00
LDR     R0, [R0]
CMP     R0, #0
BEQ     locret_8000274
```

```
LDR     R0, =dword_20001EE4
LDR     R0, [R0]
LDR     R0, [R0,#0x2C]
LDR     R1, [R4,#0x2C]
CMP     R0, R1
BCS     locret_8000274
```

```
BL      loc_8000CB0
```

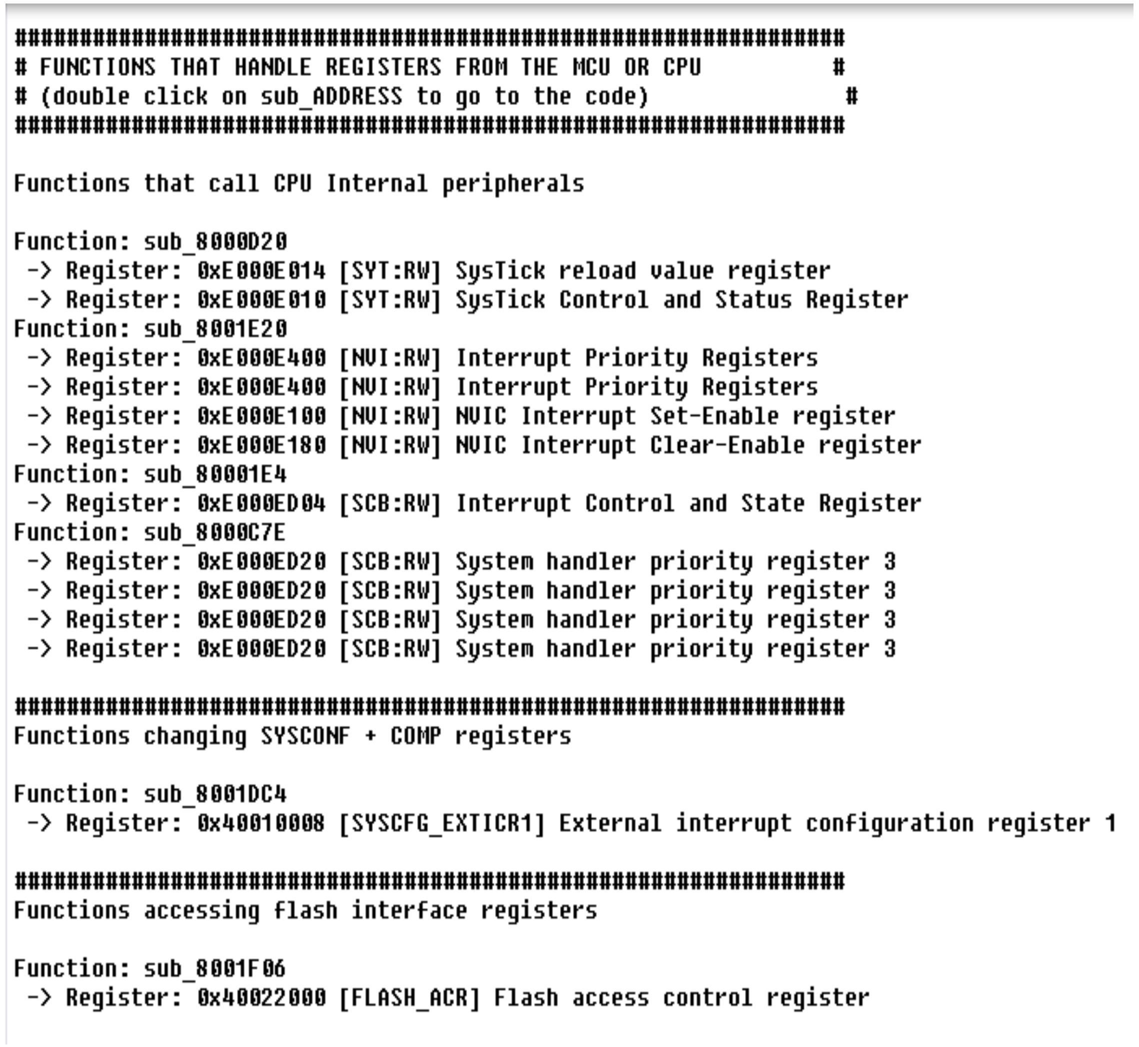
```
locret_8000274
POP     {R4,PC}
; End of function sub_80001E4
```

```
100.00% (119,1074) (418,556) 000001F0 080001F0: sub_80001E4+C (Synchronized with Hex View-1)
```

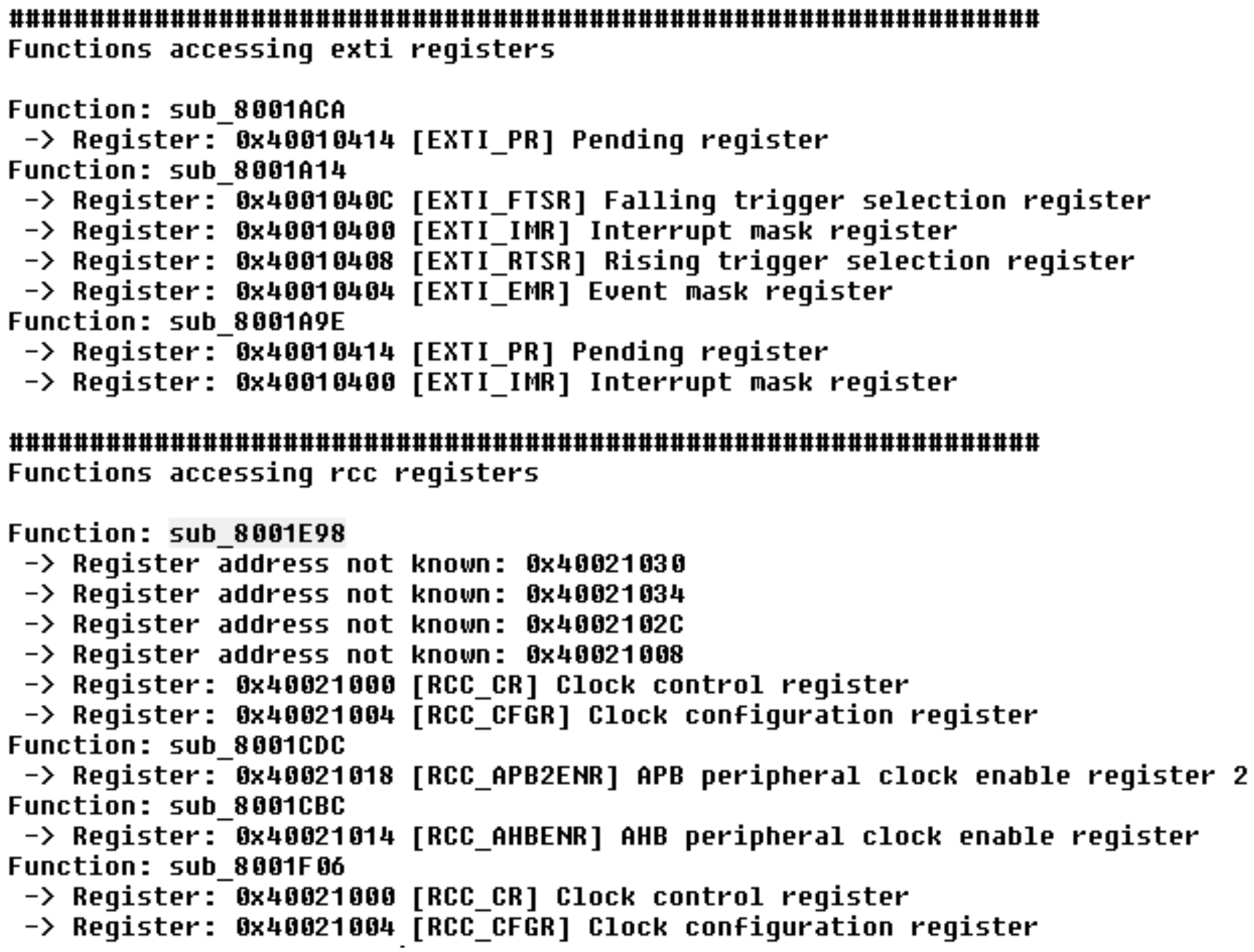
Interesting registers

- External interrupts where activated using SYSCFGEXT register.
- External interrupts are manipulated using EXTI registers
- Clock source and reload values configured using the SysTick registers
  - Used on all kind of timers if the clock is given by the CPU
- Nested Vector Interrupt control can be clear or set using the NVI registers
- Real Clock Controller can be manipulated with the RCC registers
  - crucial for input/output operations on peripherals

Interesting registers



```
##############################################################
# FUNCTIONS THAT HANDLE REGISTERS FROM THE MCU OR CPU        #
# (double click on sub_ADDRESS to go to the code)             #
##############################################################

Functions that call CPU Internal peripherals

Function: sub_8000D20
 -> Register: 0xE000E014 [SYT:RW] SysTick reload value register
 -> Register: 0xE000E010 [SYT:RW] SysTick Control and Status Register
Function: sub_8001E20
 -> Register: 0xE000E400 [NVI:RW] Interrupt Priority Registers
 -> Register: 0xE000E400 [NVI:RW] Interrupt Priority Registers
 -> Register: 0xE000E100 [NVI:RW] NVIC Interrupt Set-Enable register
 -> Register: 0xE000E180 [NVI:RW] NVIC Interrupt Clear-Enable register
Function: sub_80001E4
 -> Register: 0xE000ED04 [SCB:RW] Interrupt Control and State Register
Function: sub_8000C7E
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3
 -> Register: 0xE000ED20 [SCB:RW] System handler priority register 3

##############################################################
Functions changing SYSCONF + COMP registers

Function: sub_8001DC4
 -> Register: 0x40010008 [SYSCFG_EXTICR1] External interrupt configuration register 1

##############################################################
Functions accessing flash interface registers

Function: sub_8001F06
 -> Register: 0x40022000 [FLASH_ACR] Flash access control register
```



```
################################################################
Functions accessing exti registers

Function: sub_8001ACA
 -> Register: 0x40010414 [EXTI_PR] Pending register
Function: sub_8001A14
 -> Register: 0x4001040C [EXTI_FTSR] Falling trigger selection register
 -> Register: 0x40010400 [EXTI_IMR] Interrupt mask register
 -> Register: 0x40010408 [EXTI_RTSR] Rising trigger selection register
 -> Register: 0x40010404 [EXTI_EMR] Event mask register
Function: sub_8001A9E
 -> Register: 0x40010414 [EXTI_PR] Pending register
 -> Register: 0x40010400 [EXTI_IMR] Interrupt mask register

################################################################
Functions accessing rcc registers

Function: sub_8001E98
 -> Register address not known: 0x40021030
 -> Register address not known: 0x40021034
 -> Register address not known: 0x4002102C
 -> Register address not known: 0x40021008
 -> Register: 0x40021000 [RCC_CR] Clock control register
 -> Register: 0x40021004 [RCC_CFGR] Clock configuration register
Function: sub_8001CDC
 -> Register: 0x40021018 [RCC_APB2ENR] APB peripheral clock enable register 2
Function: sub_8001CBC
 -> Register: 0x40021014 [RCC_AHBENR] AHB peripheral clock enable register
Function: sub_8001F06
 -> Register: 0x40021000 [RCC_CR] Clock control register
 -> Register: 0x40021004 [RCC_CFGR] Clock configuration register
```

Future work

- Implement heuristics to find registers dynamically addressed
- Automatically re-analyse the interrupt handlers based on the decoded IVT
- Comment calls that will ReEnable Interrupts
- Improve analysis on registers manipulation and identification
- Identification of the Realtime operating system

THANK YOU