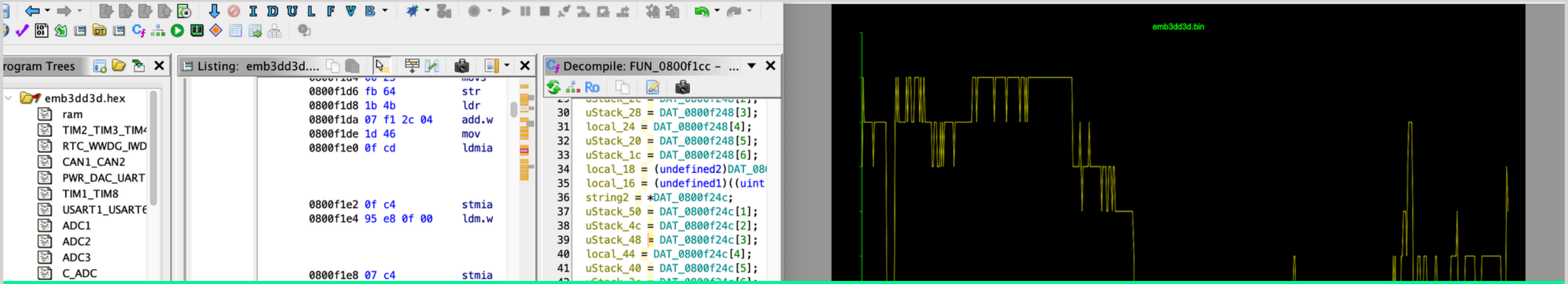
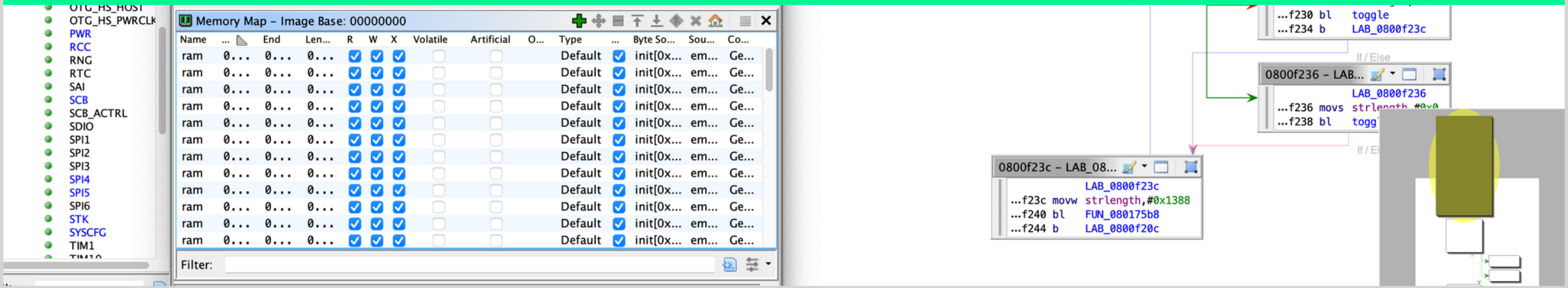


Bare Metal Reverse Engineering



Agenda

1.Bare Metal

2.Firmware Overview

3.ARM Assembly

4.Firmware Development

5.Basic Reverse Engineering

6.Advanced Reverse Engneering

Whoami

Caleb Davis

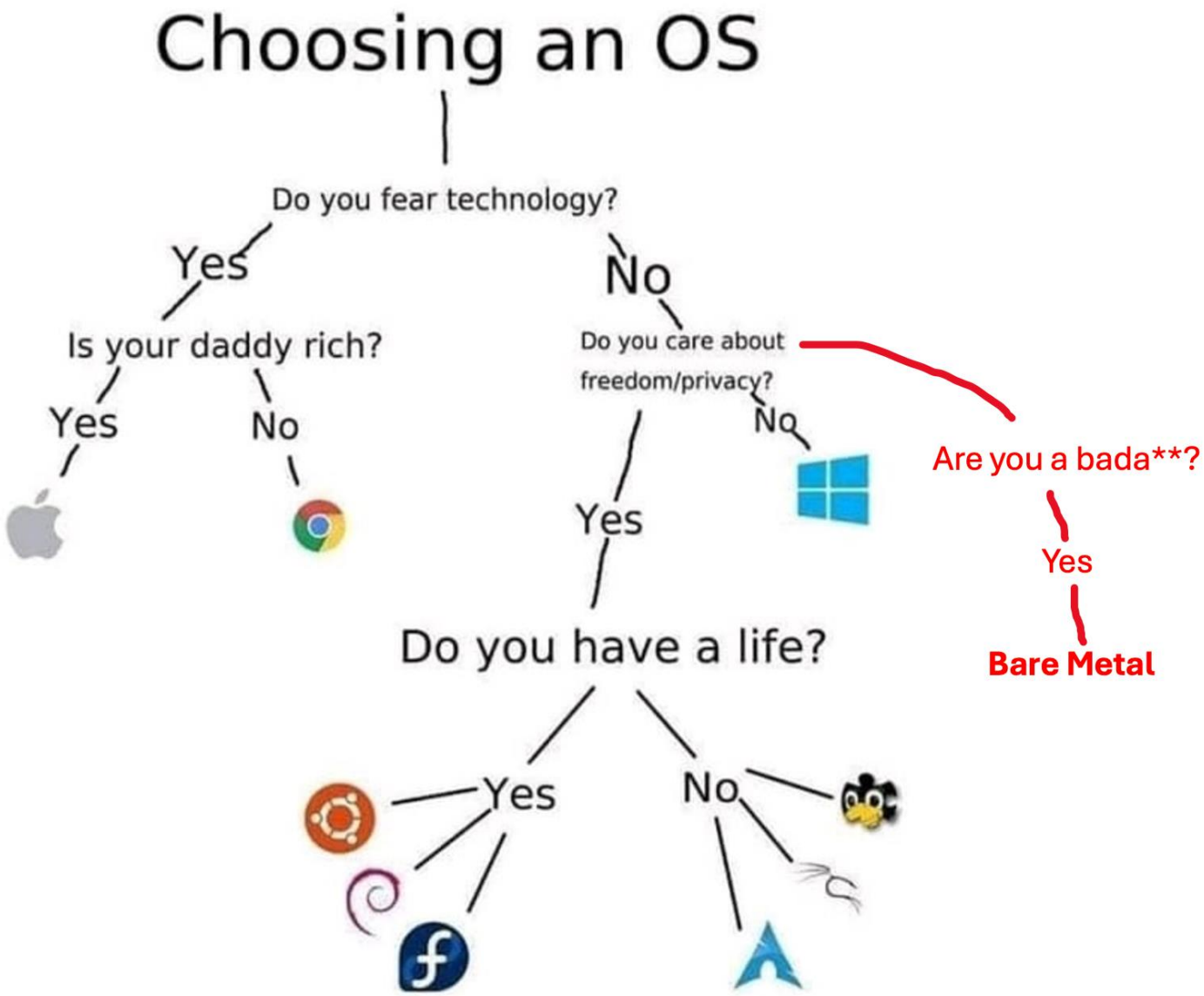
Co-Founder of SolaSec

Penetration Tester | EE | Reverse Engineer

@So11Deo6loria

Bare Metal

Bare-metal systems run directly on the hardware, without the support of a general-purpose operating system like Linux or Windows.



Bare Metal Flow Diagram

Bare Metal

Key Considerations

Minimal Privilege Seperation

Most embedded RTOSes run all tasks in privileged mod
unless a Memory Protection Unit (MPU) is configured.

Lightweight Scheduling, Minimal Abstraction

Real-Time Operating Systems (RTOS) provide multitask
and timing but often rely on direct hardware access and
lack full process isolation unless explicitly configured.

Typical Firmware Extraction

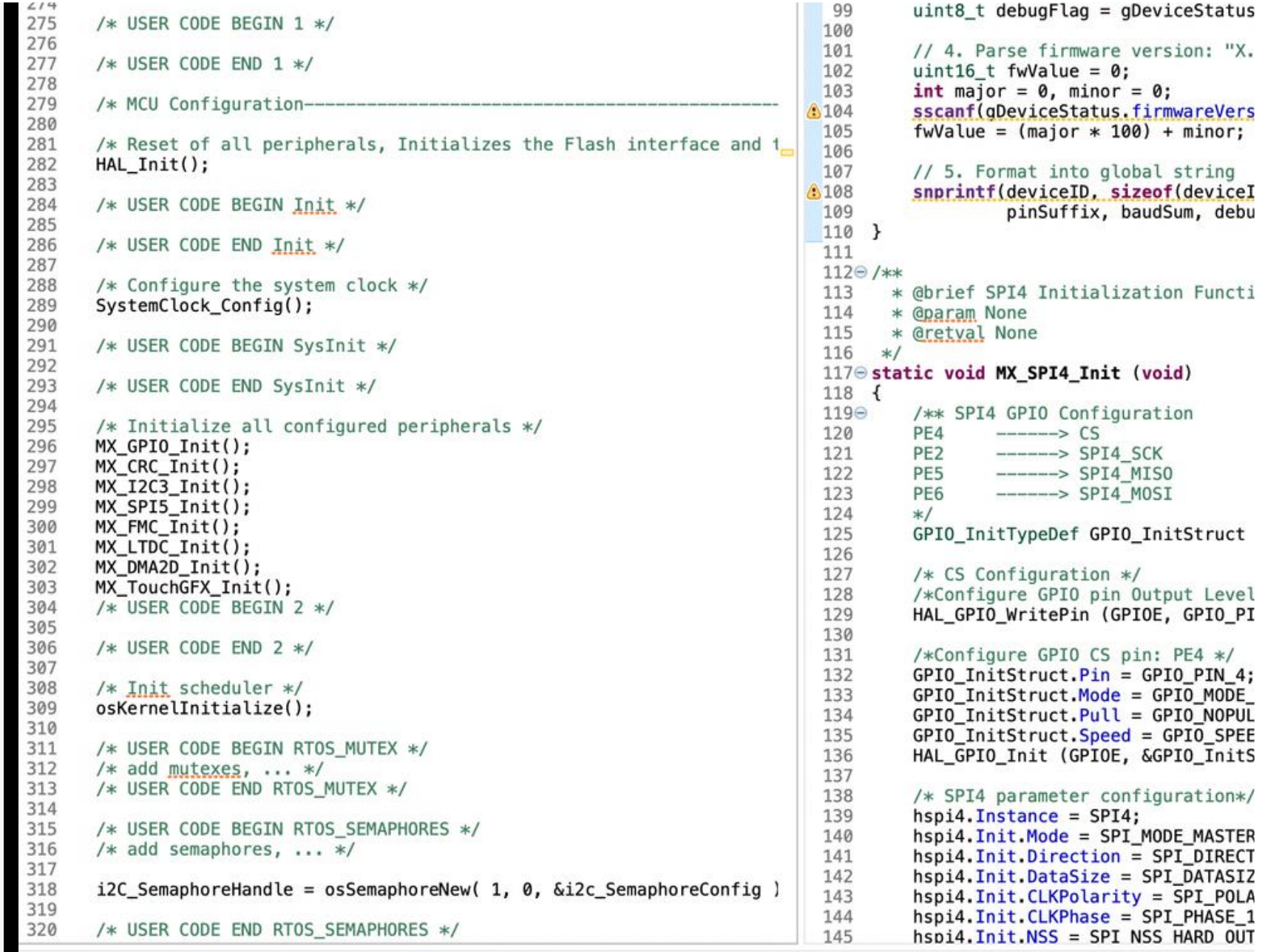
Firmware obtained from flash or memory is usually a ra
binary with no headers, no symbols, and no metadata.

Direct Register Access Is Common

Application code typically accesses memory-mapped
peripherals directly, though some RTOSes offer optiona
high-level APIs to abstract hardware.

Firmware Overview

Firmware is low-level software that runs directly on hardware. In bare-metal systems, it performs everything from initialization to peripheral control without relying on an operating system.



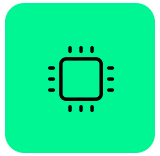
```
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/* MCU Configuration--------------------------------------------------------*/

/* Reset of all peripherals, Initializes the Flash interface and i
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_CRC_Init();
MX_I2C3_Init();
MX_SPI5_Init();
MX_FMC_Init();
MX_LTDC_Init();
MX_DMA2D_Init();
MX_TouchGFX_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Init scheduler */
osKernelInitialize();

/* USER CODE BEGIN RTOS_MUTEX */
/* add mutexes, ... */
/* USER CODE END RTOS_MUTEX */

/* USER CODE BEGIN RTOS_SEMAPHORES */
/* add semaphores, ... */

i2C_SemaphoreHandle = osSemaphoreNew( 1, 0, &i2c_SemaphoreConfig )

/* USER CODE END RTOS_SEMAPHORES */
```

```
    uint8_t debugFlag = gDeviceStatus

    // 4. Parse firmware version: "X.
    uint16_t fwValue = 0;
    int major = 0, minor = 0;
    sscanf(gDeviceStatus.firmwareVers
    fwValue = (major * 100) + minor;

    // 5. Format into global string
    snprintf(deviceID, sizeof(deviceI
             pinSuffix, baudSum, debu
}

/**
  * @brief SPI4 Initialization Functi
  * @param None
  * @retval None
  */
static void MX_SPI4_Init (void)
{
    /** SPI4 GPIO Configuration
    PE4     ------> CS
    PE2     ------> SPI4_SCK
    PE5     ------> SPI4_MISO
    PE6     ------> SPI4_MOSI
    */
    GPIO_InitTypeDef GPIO_InitStruct

    /* CS Configuration */
    /*Configure GPIO pin Output Level
    HAL_GPIO_WritePin (GPIOE, GPIO_PI

    /*Configure GPIO CS pin: PE4 */
    GPIO_InitStruct.Pin = GPIO_PIN_4;
    GPIO_InitStruct.Mode = GPIO_MODE_
    GPIO_InitStruct.Pull = GPIO_NOPUL
    GPIO_InitStruct.Speed = GPIO_SPEE
    HAL_GPIO_Init (GPIOE, &GPIO_InitS

    /* SPI4 parameter configuration*/
    hspi4.Instance = SPI4;
    hspi4.Init.Mode = SPI_MODE_MASTER
    hspi4.Init.Direction = SPI_DIRECT
    hspi4.Init.DataSize = SPI_DATASIZ
    hspi4.Init.CLKPolarity = SPI_POLA
    hspi4.Init.CLKPhase = SPI_PHASE_1
    hspi4.Init.NSS = SPI_NSS_HARD_OUT
```

```
32f4xx_ll_fmc.h    stm32f4xx_ll_fmc.c    stm32f4xx_hal_sdram.c
```

Firmware Overview

Why Learn Development?

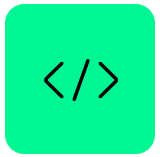


Hardware Insights

Firmware teaches how code talks to hardware through GPIO, timers, and memory-mapped registers. This knowledge is essential for low-level reverse engineering.

Decoding Memory Structures

Firmware development helps you recognize memory layouts like stack frames, peripheral maps, and initialization data when looking at raw binaries.



Identifying Code Patterns

You will spot interrupt handlers, boot sequences, and hardware setup routines even without symbols.

Full-Stack Elitism

When you understand firmware, you become the kind of person who can debug hardware, reverse binaries, and write code for both ends. Your friends may hate you for it.

Firmware Pays

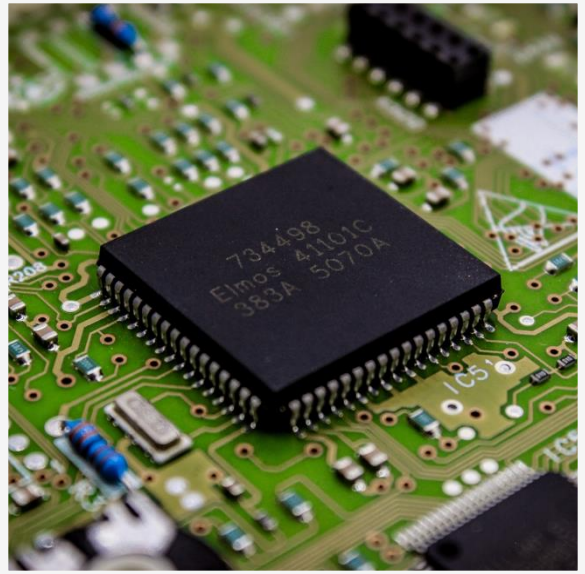
From bug bounties to consulting gigs, knowing firmware opens doors most people don't even know exist.

Just Plain Fun

There is something deeply satisfying about blinking an LED, racing the bootloader, or reversing a binary at 2 a.m. because you can.

Firmware Overview

Embedded Operating Systems

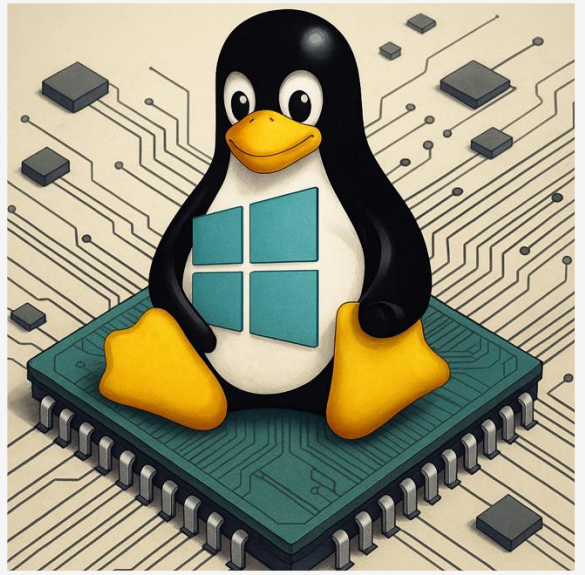


Bare-Metal

Bare-metal firmware runs without an operating system. It gives direct control over hardware and is common in tightly constrained or time-critical systems.

Real-Time Operating Systems (RTOS)

An RTOS provides predictable task scheduling and cooperative or preemptive multitasking. It is widely used in systems requiring timing guarantees, like industrial control or IoT.

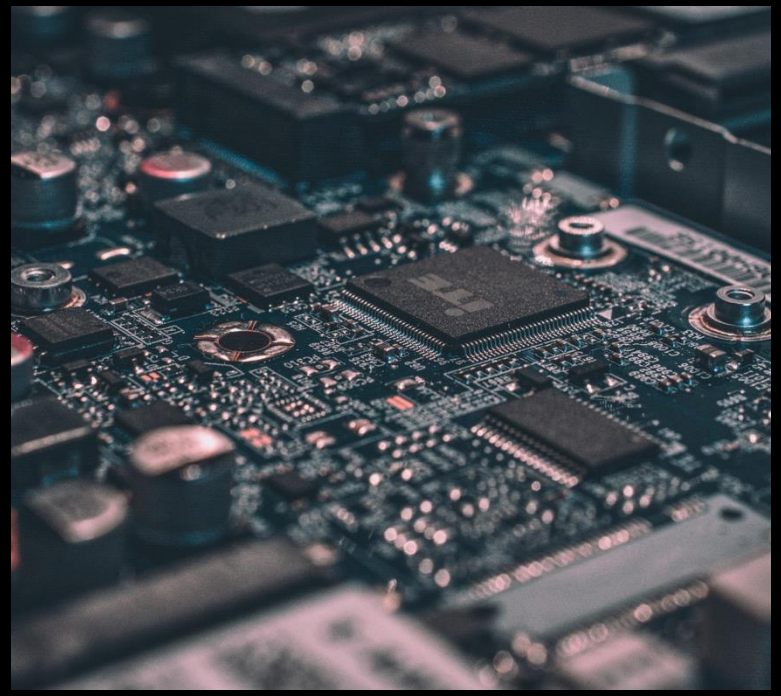


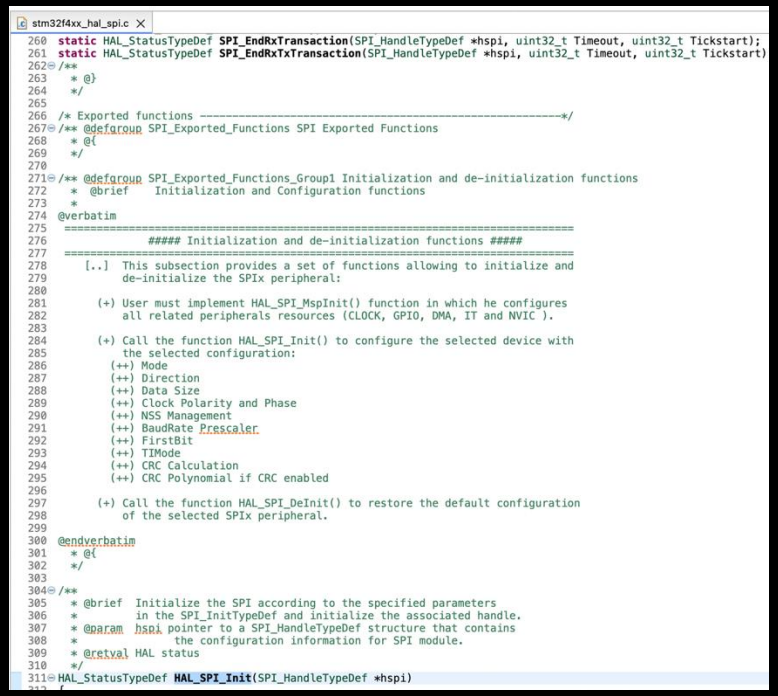
Embedded OS

Embedded OS platforms like Linux or Windows IoT offer process isolation, networking, and UI frameworks for advanced embedded systems.

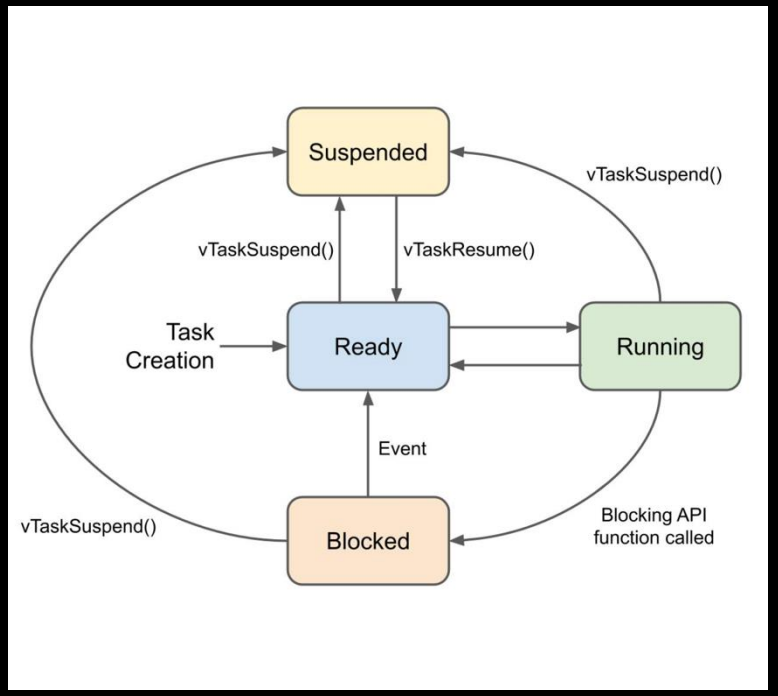
Firmware Overview

Key Software Components





```
stm32f4xx_hal_spi.c
260  static HAL_StatusTypeDef SPI_EndRxTransaction(SPI_HandleTypeDef *hspi, uint32_t Timeout, uint32_t Tickstart);
261  static HAL_StatusTypeDef SPI_EndRxTxTransaction(SPI_HandleTypeDef *hspi, uint32_t Timeout, uint32_t Tickstart)
262  /**
263    * @}
264    */
265
266  /* Exported functions ---------------------------------------------------------*/
267  /** @defgroup SPI_Exported_Functions SPI Exported Functions
268    * @{
269    */
270
271  /** @defgroup SPI_Exported_Functions_Group1 Initialization and de-initialization functions
272    *  @brief    Initialization and Configuration functions
273    *
274  @verbatim
275   ===============================================================================
276                 ##### Initialization and de-initialization functions #####
277   ===============================================================================
278      [..]  This subsection provides a set of functions allowing to initialize and
279            de-initialize the SPIx peripheral:
280
281        (+) User must implement HAL_SPI_MspInit() function in which he configures
282            all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC ).
283
284        (+) Call the function HAL_SPI_Init() to configure the selected device with
285            the selected configuration:
286          (++) Mode
287          (++) Direction
288          (++) Data Size
289          (++) Clock Polarity and Phase
290          (++) NSS Management
291          (++) BaudRate Prescaler
292          (++) FirstBit
293          (++) TIMode
294          (++) CRC Calculation
295          (++) CRC Polynomial if CRC enabled
296
297        (+) Call the function HAL_SPI_DeInit() to restore the default configuration
298            of the selected SPIx peripheral.
299
300  @endverbatim
301    * @{
302    */
303
304  /**
305    * @brief  Initialize the SPI according to the specified parameters
306    *         in the SPI_InitTypeDef and initialize the associated handle.
307    * @param  hspi pointer to a SPI_HandleTypeDef structure that contains
308    *               the configuration information for SPI module.
309    * @retval HAL status
310    */
311  HAL_StatusTypeDef HAL_SPI_Init(SPI_HandleTypeDef *hspi)
```



Board Support Package (BSP)

The BSP initializes hardware components and provides low-level drivers tailored to a specific board or microcontroller.

Hardware Abstraction Layer (HAL)

The HAL offers a standardized API that abstracts direct hardware access, enabling easier portability across hardware platforms.

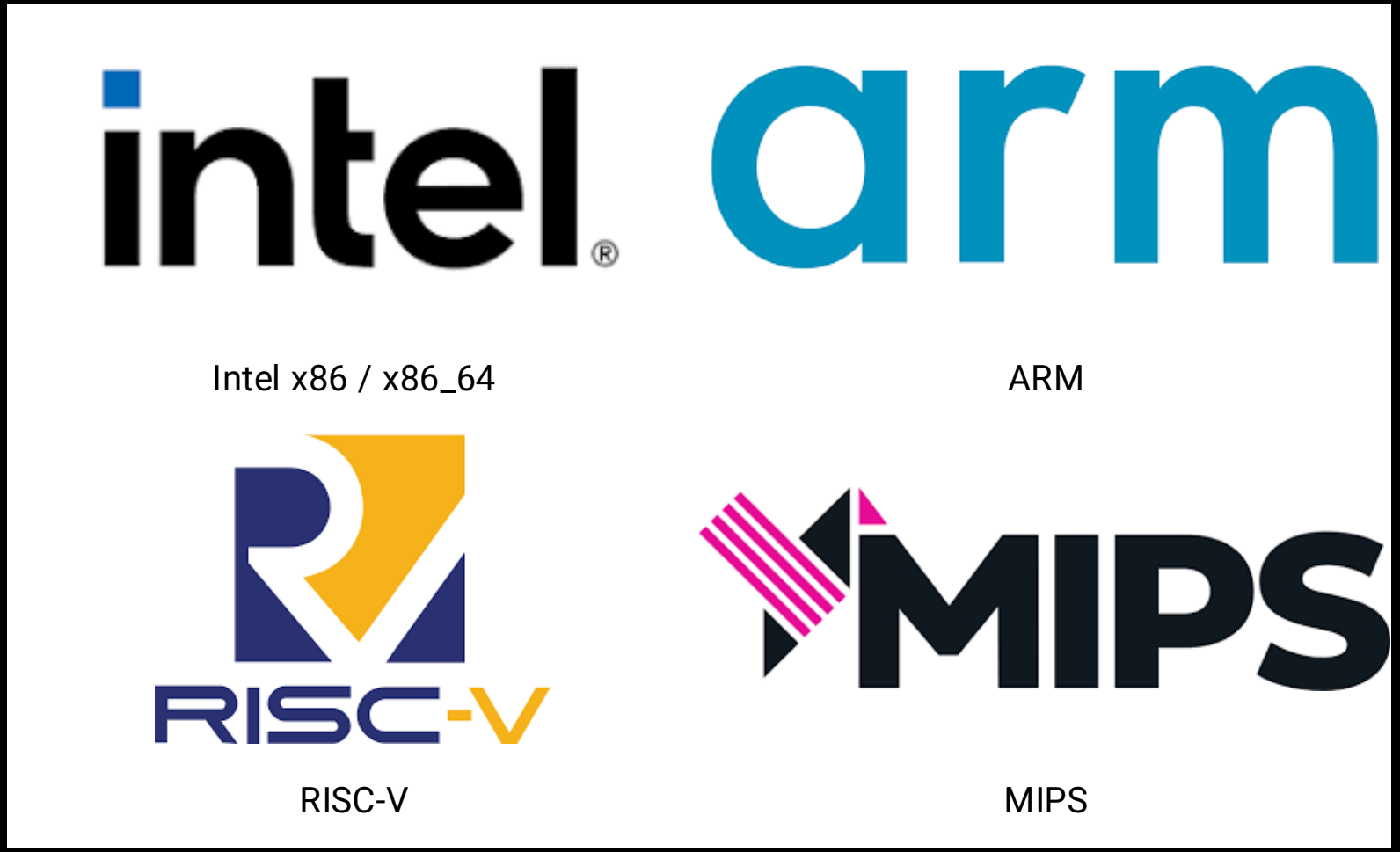
Real-Time Operating System (RTOS)

The RTOS manages task scheduling, timing, and resource allocation to ensure deterministic behavior in embedded systems.

ARM Assembly

Fundamentals

CPU Architectures



ARM Assembly

Fundamentals

Intel vs. ARM

Intel x86 / x86_64

- CISC
- Fewer-general purpose registers
- Heavy Stack Usage
- Complex Memory Addressing

ARM

- RISC
- 16 Registers (R0-R15)
- Arguments in R0-R3 / Returns in R0
- Simple Decoding / PC Relative Addressing

ARM Assembly

Fundamentals

ARM and Thumb

ARM

- 32-Bit
- Larger code, faster speed
- Full register access (R0–R15, including condition flags)

Thumb

- Kinda 32-Bit, Kinda 16-Bit
- Smaller code, slower speed
- Access to fewer registers per instruction

ARM processors can dynamically switch between ARM and Thumb modes using BX or BLX instructions. The least significant bit of the target address determines the mode (0 for ARM, 1 for Thumb). Compilers like GCC often emit mixed-mode code (Thumb-2) to optimize for both speed and code size. The CPSR T-bit reflects the current mode.

ARM Assembly

Fundamentals

Registers and Data Types

Registers

- R0-R12: General Purpose Registers
- R13: Stack Pointer (SP)
- R14: Link Register (LR)
- R15: Program Counter

Data Types

- Word: 32-bit
- Halfword: 16-bit
- Byte: 8-bit

Examples

- ldr: Load Word
- ldrh: Load Halfword
- ldrsb: Load Signed Byte

ARM Assembly

Memory and Flow Control

LDR/STR

```
ldr     r2, [r0]     @ load word from *r0          -> r2
str     r2, [r1]     @ store word r2               -> *r1
ldrb    r3, [r0, #4] @ load unsigned byte *(r0+4)  -> r3
ldrsb   r4, [r0, #5] @ load signed byte *(r0+5)    -> r4
ldrh    r5, [r0, #6] @ load unsigned half *(r0+6)  -> r5
strh    r5, [r1, #2] @ store half r5               -> *(r1+2)
```

Branching

```
b       loop_start      @ jump to 'loop_start' unconditionally
cmp     r0, #0          @ compare r0 to 0 (sets flags)
beq     is_zero         @ branch to 'is_zero' if equal (Z flag set)
bne     not_zero        @ branch to 'not_zero' if not equal (Z flag clear)
bl      do_work         @ branch to 'do_work', save return addr in LR
bx      lr              @ branch to address in LR (return to caller)
```

Relative Addressing

```
adr     r0, const_table @ r0 = address of const_table
ldr     r1, [r0]        @ load first word (0x11111111)
ldr     r2, [r0, #4]    @ load second word (0x22222222)
add     r3, r1, r2      @ r3 = r1 + r2
bx      lr              @ return
```

Stack Frames

```
@ --- Prologue ---
push    {r11, lr}   @ save frame pointer and return address
add     r11, sp, #0 @ set frame pointer to current stack pointer

@ --- Epilogue ---
add     sp, r11, #0 @ restore stack pointer from frame pointer
pop     {r11, pc}   @ restore frame pointer and return (pc = lr)
```

ARM Assembly

Practical Applications

Common ARM Instructions

Instruction	Description	Instruction	Description
MOV	Move data	**EOR**	Bitwise XOR
MVN	Move and negate	**LDR**	Load
ADD	Addition	**STR**	Store
SUB	Subtraction	**LDM**	Load Multiple
MUL	Multiplication	**STM**	Store Multiple
LSL	Logical Shift Left	**PUSH**	Push on Stack
LSR	Logical Shift Right	**POP**	Pop off Stack
ASR	Arithmetic Shift Right	**B**	Branch
ROR	Rotate Right	**BL**	Branch with Link
CMP	Compare	**BX**	Branch and eXchange
AND	Bitwise AND	**BLX**	Branch with Link and eXchange
ORR	Bitwise OR	**SWI/SVC**	System Call

ARM assembly programs are built from a small set of versatile instructions. These cover data movement (MOV, LDR, STR), arithmetic and logic (ADD, SUB, AND, ORR), and control flow (B, BL, BX). Shifts and rotates (LSL, ASR, ROR) efficiently manipulate bits, while stack operations (PUSH, POP) handle function calls and local variables. Understanding these core instructions is key to reading and writing ARM code. Ask ChatGPT

ARM Assembly

Practical Applications

C-Code vs. Disassembly

```c
#include <stdint.h>

int check( uint8_t a, uint32_t b )
{
        if( a == b )
        {
                return 1;
        }
        else
        {
                return 0;
        }

}

int main(void) {
        uint8_t  a = 0;
        uint32_t b =  1;
        int8_t   c = -1;

        c = check( a, b );
        return c;
}
```

```
(remote) gef➤  disassemble check
Dump of assembler code for function check:
   0x00010398 <+0>:     push    {r11}           @ (str r11, [sp, #-4]!)
   0x0001039c <+4>:     add     r11, sp, #0
   0x000103a0 <+8>:     sub     sp, sp, #12
   0x000103a4 <+12>:    mov     r3, r0
   0x000103a8 <+16>:    str     r1, [r11, #-12]
   0x000103ac <+20>:    strb    r3, [r11, #-5]
   0x000103b0 <+24>:    ldrb    r3, [r11, #-5]
   0x000103b4 <+28>:    ldr     r2, [r11, #-12]
   0x000103b8 <+32>:    cmp     r2, r3
   0x000103bc <+36>:    bne     0x103c8 <check+48>
   0x000103c0 <+40>:    mov     r3, #1
   0x000103c4 <+44>:    b       0x103cc <check+52>
   0x000103c8 <+48>:    mov     r3, #0
   0x000103cc <+52>:    mov     r0, r3
   0x000103d0 <+56>:    add     sp, r11, #0
   0x000103d4 <+60>:    pop     {r11}           @ (ldr r11, [sp], #4)
   0x000103d8 <+64>:    bx      lr
End of assembler dump.
(remote) gef➤  disassemble main
Dump of assembler code for function main:
   0x000103dc <+0>:     push    {r11, lr}
   0x000103e0 <+4>:     add     r11, sp, #4
   0x000103e4 <+8>:     sub     sp, sp, #16
   0x000103e8 <+12>:    mov     r3, #0
   0x000103ec <+16>:    strb    r3, [r11, #-5]
   0x000103f0 <+20>:    mov     r3, #1
   0x000103f4 <+24>:    str     r3, [r11, #-12]
   0x000103f8 <+28>:    mvn     r3, #0
   0x000103fc <+32>:    strb    r3, [r11, #-13]
   0x00010400 <+36>:    ldrb    r3, [r11, #-5]
   0x00010404 <+40>:    ldr     r1, [r11, #-12]
   0x00010408 <+44>:    mov     r0, r3
   0x0001040c <+48>:    bl      0x10398 <check>
   0x00010410 <+52>:    mov     r3, r0
   0x00010414 <+56>:    strb    r3, [r11, #-13]
   0x00010418 <+60>:    ldrsb   r3, [r11, #-13]
   0x0001041c <+64>:    mov     r0, r3
   0x00010420 <+68>:    sub     sp, r11, #4
   0x00010424 <+72>:    pop     {r11, pc}
End of assembler dump.
(remote) gef➤
```

Firmware Development

The Hard Way

Simple Goal: Initialize a Single GPIO

How hard could it be?

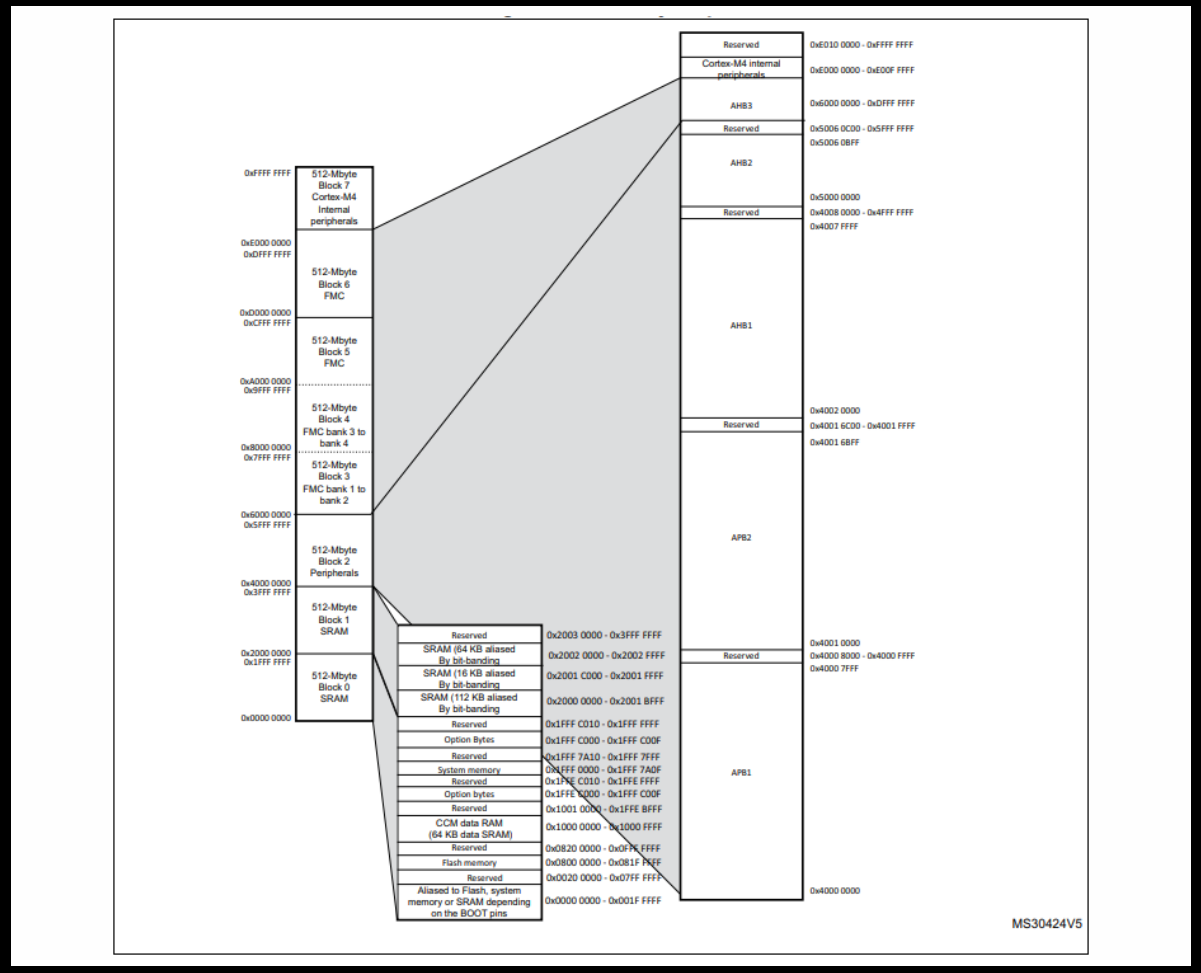
Steps

1. Initialize Peripheral Clock
2. Set GPIO as Input Mode
3. Set GPIO as Pull-Up Resistor

Firmware Development

The Hard Way

STM32 Memory Map



Firmware Development

The Hard Way

RCC Peripheral Boundary Address

Table 1. STM32F4xx register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x5006 0800 - 0x5006 0BFF	RNG	AHB2	*Section 24.4.4: RNG register map on page 774*
0x5006 0400 - 0x5006 07FF	HASH		*Section 25.4.9: HASH register map on page 798*
0x5006 0000 - 0x5006 03FF	CRYP		*Section 23.6.13: CRYP register map on page 766*
0x5005 0000 - 0x5005 03FF	DCMI		*Section 15.8.12: DCMI register map on page 481*
0x5000 0000 - 0x5003 FFFF	USB OTG FS		*Section 34.16.6: OTG_FS register map on page 1329*
0x4004 0000 - 0x4007 FFFF	USB OTG HS		*Section 35.12.6: OTG_HS register map on page 1475*
0x4002 B000 - 0x4002 BBFF	DMA2D		*Section 11.5: DMA2D registers on page 355*
0x4002 8000 - 0x4002 93FF	ETHERNET MAC		*Section 33.8.5: Ethernet register maps on page 1239*
0x4002 6400 - 0x4002 67FF	DMA2		*Section 10.5.11: DMA register map on page 338*
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 4000 - 0x4002 4FFF	BKPSRAM		-
0x4002 3C00 - 0x4002 3FFF	Flash interface register		*Section 3.9: Flash interface registers*
0x4002 3800 - 0x4002 3BFF	RCC		*Section 7.3.24: RCC register map on page 267*
0x4002 3000 - 0x4002 33FF	CRC	AHB1	*Section 4.4.4: CRC register map on page 116*
0x4002 2800 - 0x4002 2BFF	GPIOK		*Section 8.4.11: GPIO register map on page 290*
0x4002 2400 - 0x4002 27FF	GPIOJ		

Firmware Development

The Hard Way

RCC Peripheral Register Map

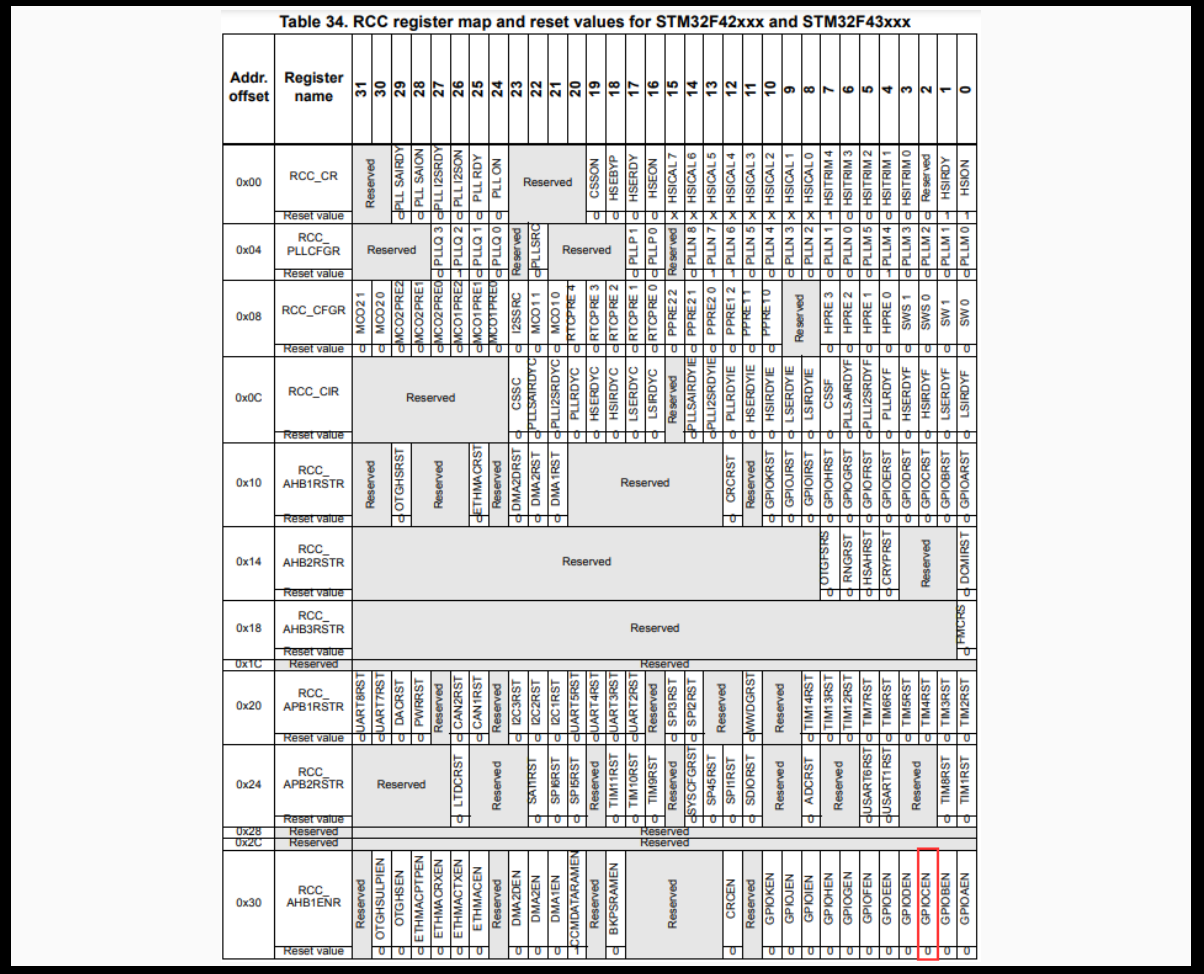


Table 34. RCC register map and reset values for STM32F42xxx and STM32F43xxx

Addr. offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	RCC_CR	Reserved		PLL SAIRDY	PLL SAION	PLL I2SRDY	PLL I2SON	PLL RDY	PLL ON	Reserved				CSSON	HSEBYP	HSERDY	HSEON	HSICAL7	HSICAL6	HSICAL5	HSICAL4	HSICAL3	HSICAL2	HSICAL1	HSICAL0	HSITRIM4	HSITRIM3	HSITRIM2	HSITRIM1	HSITRIM0	Reserved	HSIRDY	HSION
	Reset value			0	0	0	0	0	0					0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	0	0		1	1
0x04	RCC_PLLCFGR	Reserved			PLLQ 3	PLLQ 2	PLLQ 1	PLLQ 0	Reserved	PLLSRC	Reserved			PLLP 1	PLLP 0	Reserved	PLLN 8	PLLN 7	PLLN 6	PLLN 5	PLLN 4	PLLN 3	PLLN 2	PLLN 1	PLLN 0	PLLM 5	PLLM 4	PLLM 3	PLLM 2	PLLM 1	PLLM 0		
	Reset value				0	0	0	0		0				0	0		1	0	1	1	0	1	0	1	0	0	0	0	1	0	0	0	1
0x08	RCC_CFGR	MCO2 1	MCO2 0	MCO2PRE2	MCO2PRE1	MCO2PRE0	MCO1PRE2	MCO1PRE1	MCO1PRE0	I2SSRC	MCO1 1	MCO1 0	RTCPRE 4	RTCPRE 3	RTCPRE 2	RTCPRE 1	RTCPRE 0	PPRE2 2	PPRE2 1	PPRE2 0	PPRE1 2	PPRE1 1	PPRE1 0	Reserved		HPRE 3	HPRE 2	HPRE 1	HPRE 0	SWS 1	SWS 0	SW 1	SW 0	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	
0x0C	RCC_CIR	Reserved								CSSC	PLLSAIRDYC	PLLI2SRDYC	PLLRDYC	HSERDYC	HSIRDYC	LSERDYC	LSIRDYC	Reserved		PLLSAIRDYIE	PLLI2SRDYIE	PLLRDYIE	HSERDYIE	HSIRDYIE	LSERDYIE	LSIRDYIE	CSSF	PLLSAIRDYF	PLLI2SRDYF	PLLRDYF	HSERDYF	HSIRDYF	LSERDYF	LSIRDYF
	Reset value									0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	RCC_AHB1RSTR	Reserved		OTGHSRST	Reserved		ETHMACRST	Reserved		DMA2DRST	DMA2RST	DMA1RST	Reserved						CRCRST	Reserved	GPIOKRST	GPIOJRST	GPIOIRST	GPIOHRST	GPIOGRST	GPIOFRST	GPIOERST	GPIODRST	GPIOCRST	GPIOBRST	GPIOARST		
	Reset value			0			0			0	0	0							0		0	0	0	0	0	0	0	0	0	0	0		
0x14	RCC_AHB2RSTR	Reserved																						OTGFSRST	RNGRST	HSAHRST	CRYPRST	Reserved				DCMIRST	
	Reset value																							0	0	0	0					0	
0x18	RCC_AHB3RSTR	Reserved																													FMCRST		
	Reset value																														0		
0x1C	Reserved	Reserved																															
0x20	RCC_APB1RSTR	UART8RST	UART7RST	DACRST	PWRRST	Reserved	CAN2RST	CAN1RST	Reserved	I2C3RST	I2C2RST	I2C1RST	UART5RST	UART4RST	UART3RST	UART2RST	Reserved	SPI3RST	SPI2RST	Reserved		WWDGRST	Reserved		TIM14RST	TIM13RST	TIM12RST	TIM7RST	TIM6RST	TIM5RST	TIM4RST	TIM3RST	TIM2RST	
	Reset value	0	0	0	0		0	0		0	0	0	0	0	0	0		0	0			0		0	0	0	0	0	0	0	0	0	
0x24	RCC_APB2RSTR	Reserved					LTDCRST	Reserved		SAI1RST	SPI6RST	SPI5RST	Reserved		TIM11RST	TIM10RST	TIM9RST	Reserved	SYSCFGRST	SPI4RST	SPI1RST	SDIORST	Reserved		ADCRST	Reserved			USART6RST	USART1RST	Reserved		TIM8RST	TIM1RST
	Reset value						0			0	0	0			0	0	0		0	0	0	0			0				0	0			0	0
0x28	Reserved	Reserved																															
0x2C	Reserved	Reserved																															
0x30	RCC_AHB1ENR	Reserved		OTGHSULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved	DMA2DEN	DMA2EN	DMA1EN	CCMDATARAMEN	Reserved	BKPSRAMEN	Reserved			CRCEN	Reserved	GPIOKEN	GPIOJEN	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN		
	Reset value		0	0	0	0	0	0	0		0	0	0	1		0				0		0	0	0	0	0	0	0	0	0	0		

Firmware Development

The Hard Way

RCC AHB1 Peripheral Clock Register

6.3.10 RCC AHB1 peripheral clock register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGHSULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Res.	**DMA2DEN**	DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	Reserved	
	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Res.	GPIOKEN	GPIOJEN	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Firmware Development

The Hard Way

GPIOC Peripheral Offset

0x4002 2800 - 0x4002 2BFF	GPIOK	AHB1	Section 8.4.11: GPIO register map on page 290
0x4002 2400 - 0x4002 27FF	GPIOJ		
0x4002 2000 - 0x4002 23FF	GPIOI		
0x4002 1C00 - 0x4002 1FFF	GPIOH		
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		Section 8.4.11: GPIO register map on page 290
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

Firmware Development

The Hard Way

GPIO Port Mode Register

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

Firmware Development

The Hard Way

GPIO Port Pull-Up/Pull-Down Register

8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..I/J/K)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

Firmware Development

The Hard Way

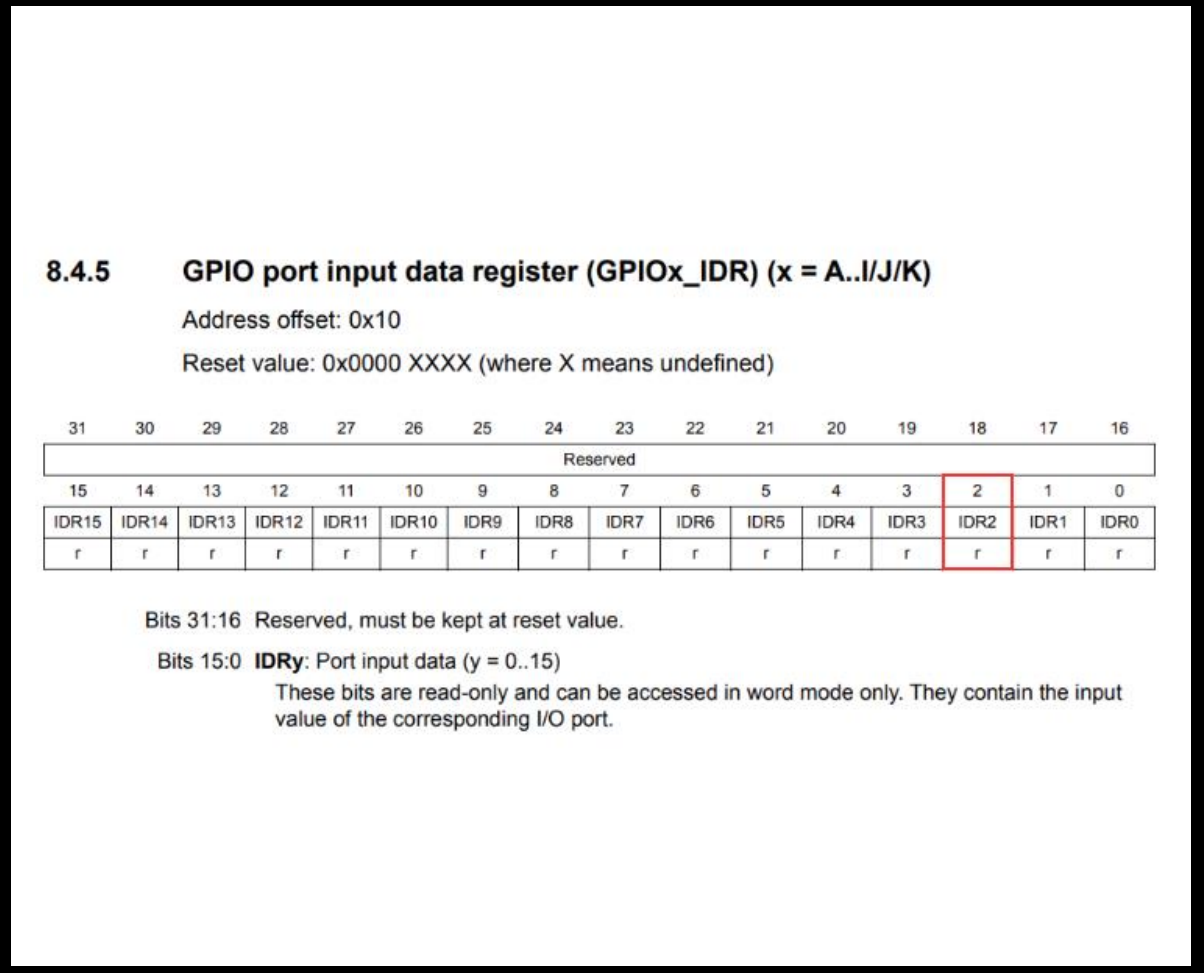
```
static uint8_t pc2InitThHardWay( void )
{
        // Initialize GPIOC Peripheral Clock (0x1 for bit 2 in RCC_AHB1ENR)
        RCC->AHB1ENR |= (1<<2);

        // Set port Input mode for PC2 (0x00 for bit 5 and 4 in GPIOC_MODER)
        GPIOC->MODER &= ~(0x00000030);

        // Reset Pull-Up Value for PC2 (0x00 for bit 5 and 4 in GPIOC_PUPDR)
        GPIOC->PUPDR &= ~(0x00000030);

        // Set port Pull-up for PC2 (0x01 for bit 5 and 4 in GPIOC_PUPDR)
        GPIOC->PUPDR |= 0x00000010;
}
```

The initialization code for the pin can be found above.

Firmware Development

The Hard Way

GPIO Port Input Data Register



8.4.5 GPIO port input data register (GPIOx_IDR) (x = A..I/J/K)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

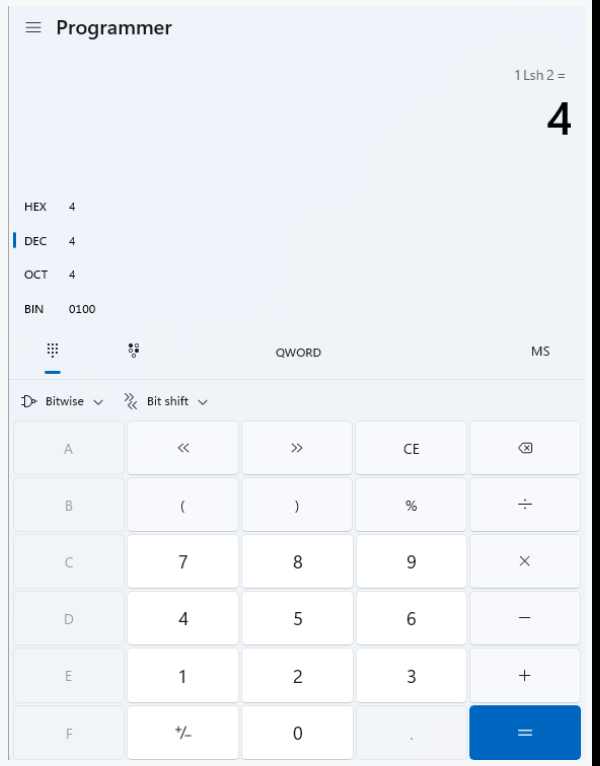
Bits 15:0 **IDRy**: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

Firmware Development

The Hard Way

```
static uint8_t checkPinTheHardWay( void )
{
        // PC2 can be checked by bit shifting a 1 to the correct register and checking the bit status
        if( ( GPIOC->IDR & ( 1 << 2 ) ) != 0 )
        {
                return 0;
        }
        else
        {
                return 1;
        }
}
```



The code to check the state of the pin can be found above.

Firmware Development

The Easy Way

Difficult Goal: Initialize a UART

Solution: HAL Drivers

Steps

1. Set Baud Rate
2. Set Word Length
3. Set Stop Bits
4. Set Parity
5. Set Mode
6. Set HW Flow Control
7. Set Oversampling

Firmware Development

The Easy Way

HAL Usage for USART1 Initialization

HAL drivers accelerate BSP usage by providing a standardized, portable interface for hardware interaction, enabling rapid configuration and integration of peripherals without delving into low-level register programming.

```
static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Firmware Development

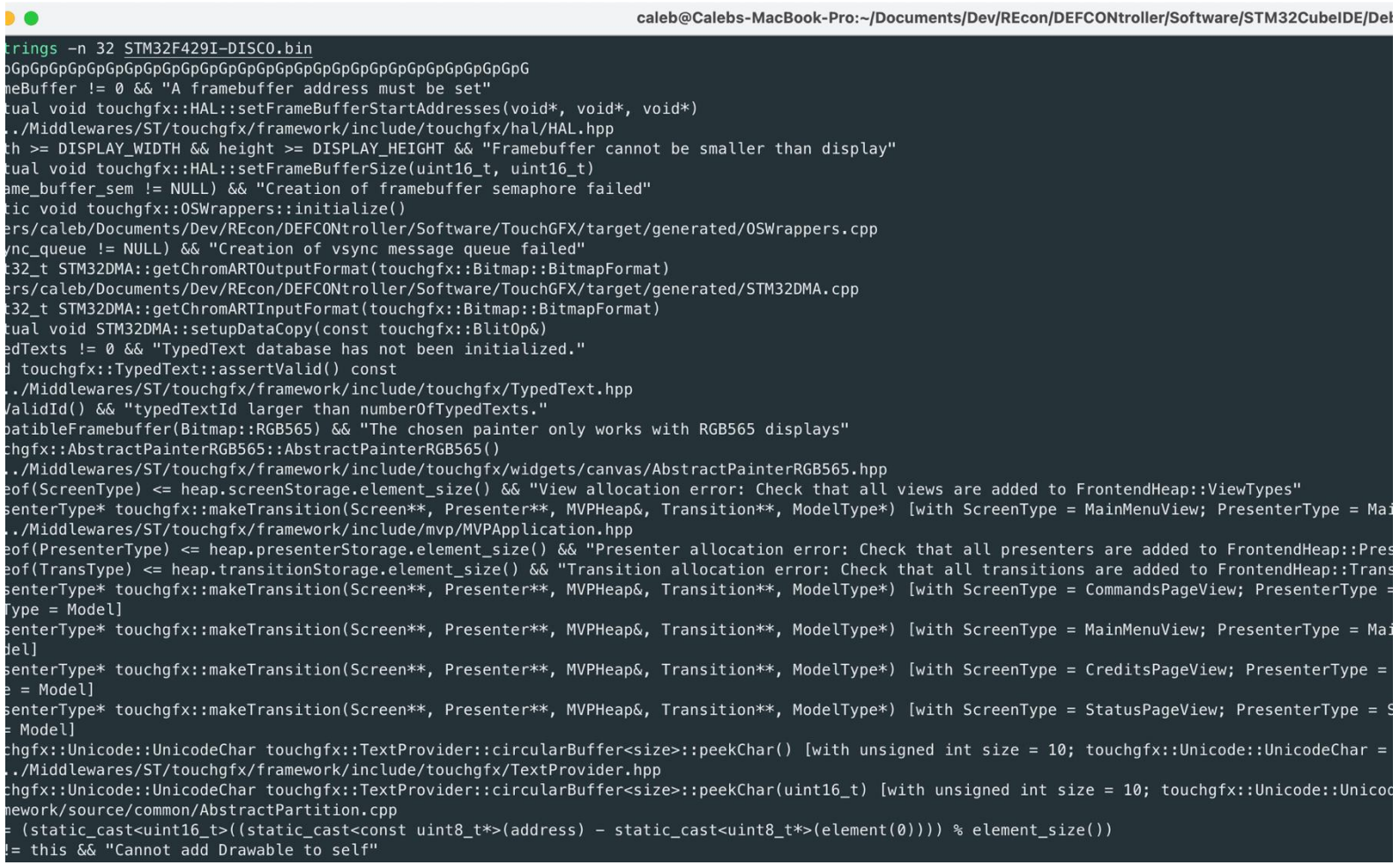
The Easy Way

Using the Peripheral with HAL Drivers

```
/* Transmit the access granted string */
HAL_UART_Transmit(&huart1, (uint8_t*)accessGranted, strlen(accessGranted), 10 );

/* Transmit the access denied string */
HAL_UART_Transmit(&huart1, (uint8_t*)accessDenied, strlen(accessDenied), 10 );

/* Transmit the super user access granted string */
HAL_UART_Transmit(&huart1, (uint8_t*)superUserAccessGranted, strlen(superUserAccessGranted), 10 );

/* Transmit the access denied string */
HAL_UART_Transmit(&huart1, (uint8_t*)accessDenied, strlen(accessDenied), 10 );
```

Basic Reverse Engineering

strings



Basic Reverse Engineering

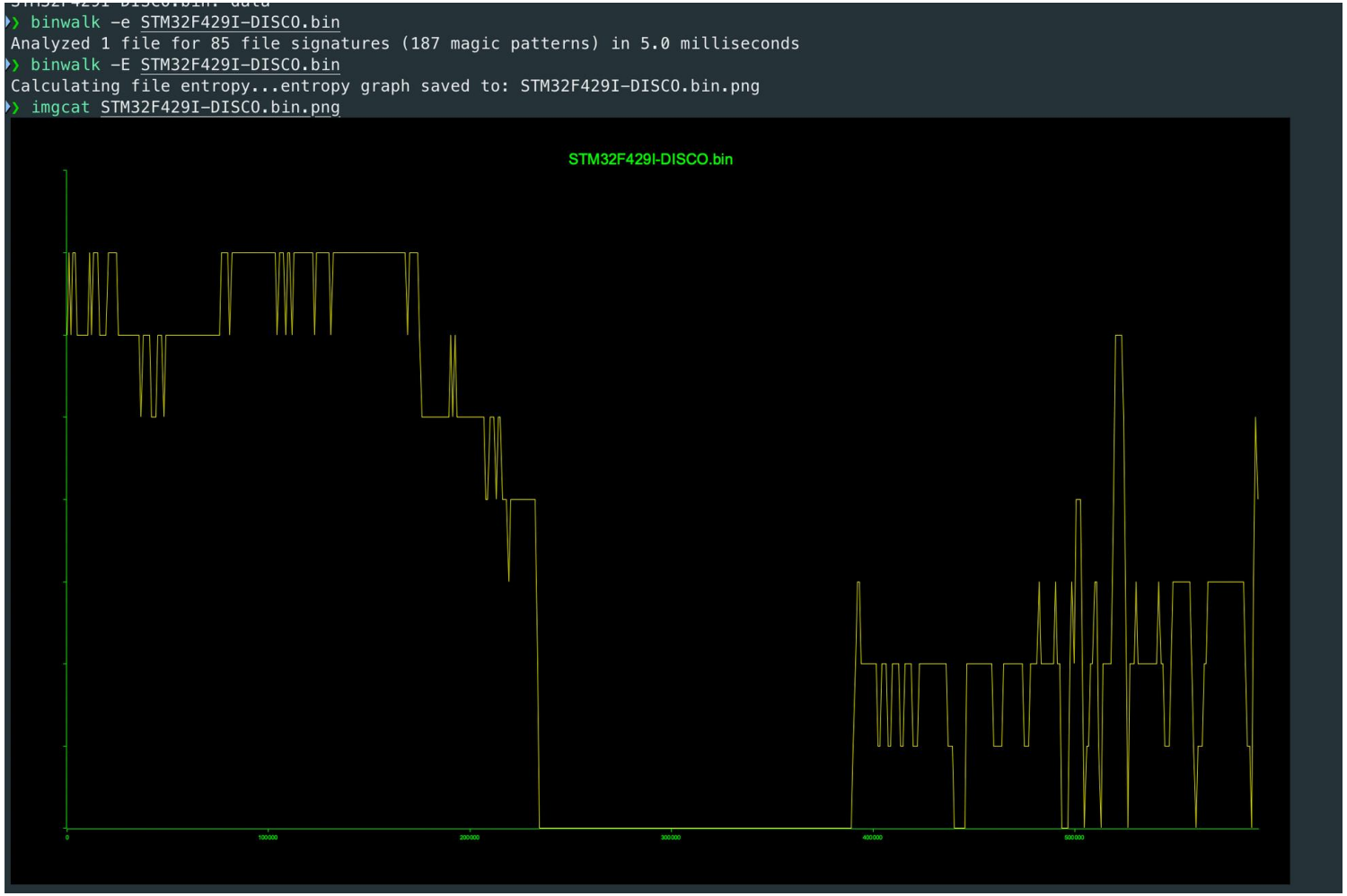
```
❯ file STM32F429I-DISCO.bin
STM32F429I-DISCO.bin: data
```

file

Basic Reverse Engineering

binwalk

```
❯ binwalk -e STM32F429I-DISCO.bin
Analyzed 1 file for 85 file signatures (187 magic patterns) in 5.0 milliseconds
❯ binwalk -E STM32F429I-DISCO.bin
Calculating file entropy...entropy graph saved to: STM32F429I-DISCO.bin.png
❯ imgcat STM32F429I-DISCO.bin.png
```



Basic Reverse Engineering

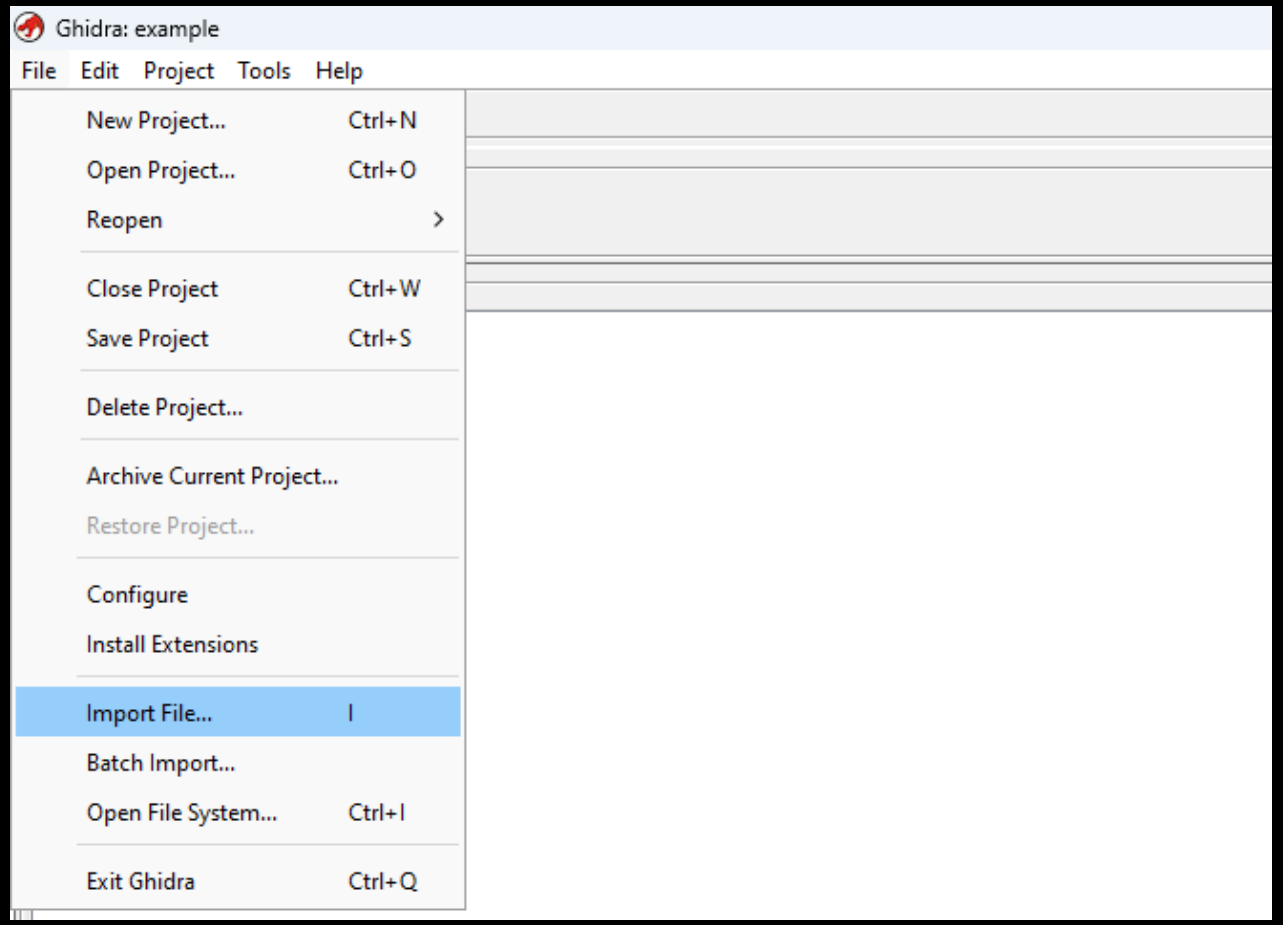
```
❯ ./aeskeyfind /Users/caleb/Documents/Dev/REcon/DEFCONtroller/Software/STM32CubeIDE/Debug/STM32F429I-DISCO.bin
Keyfind progress: 100%
```

aeskeyfind

Advanced Reverse Engineering

Ghidra Basics

File -> Import File

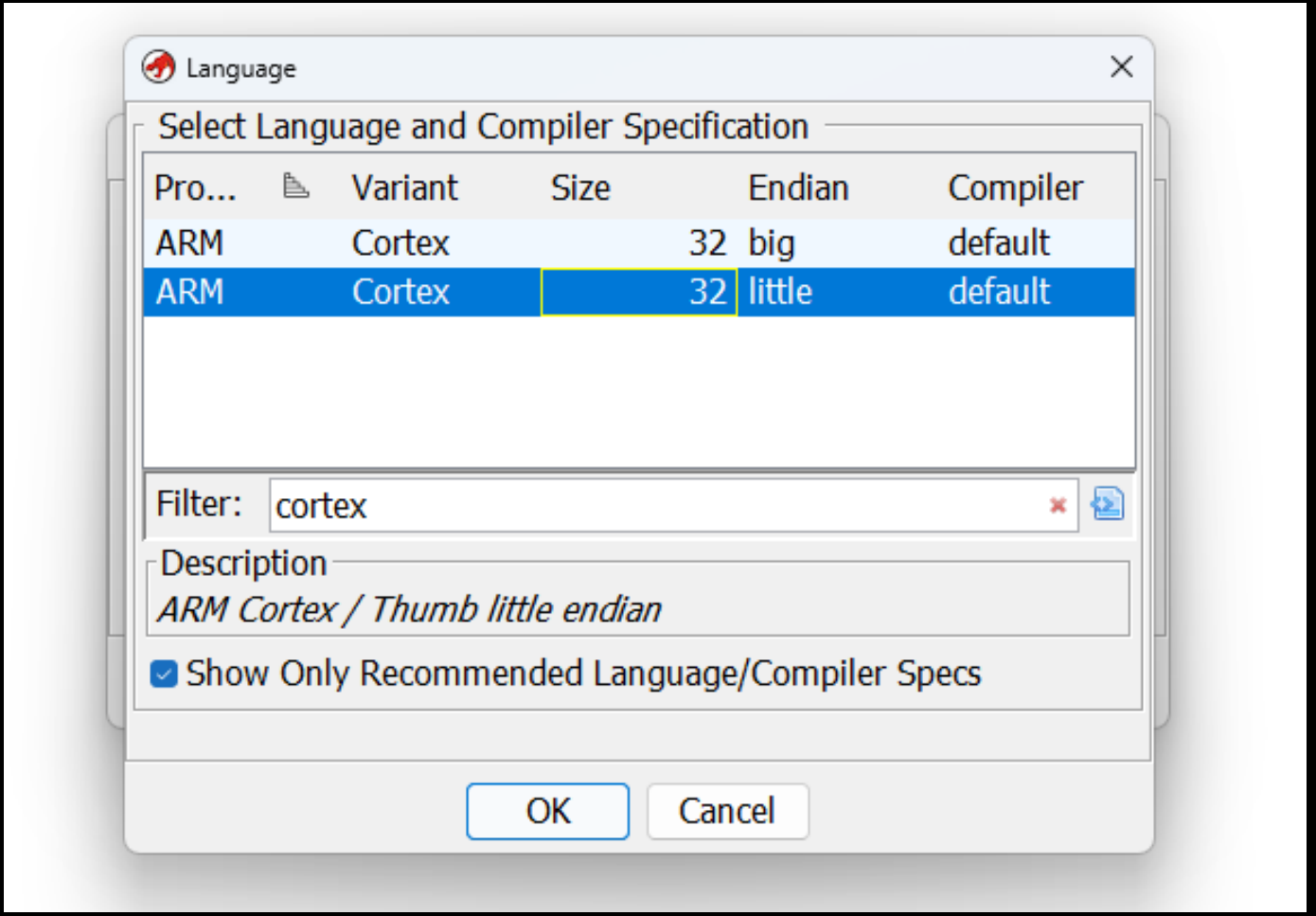


Advanced Reverse Engineering

Ghidra Basics

Select Language and Compiler Specification

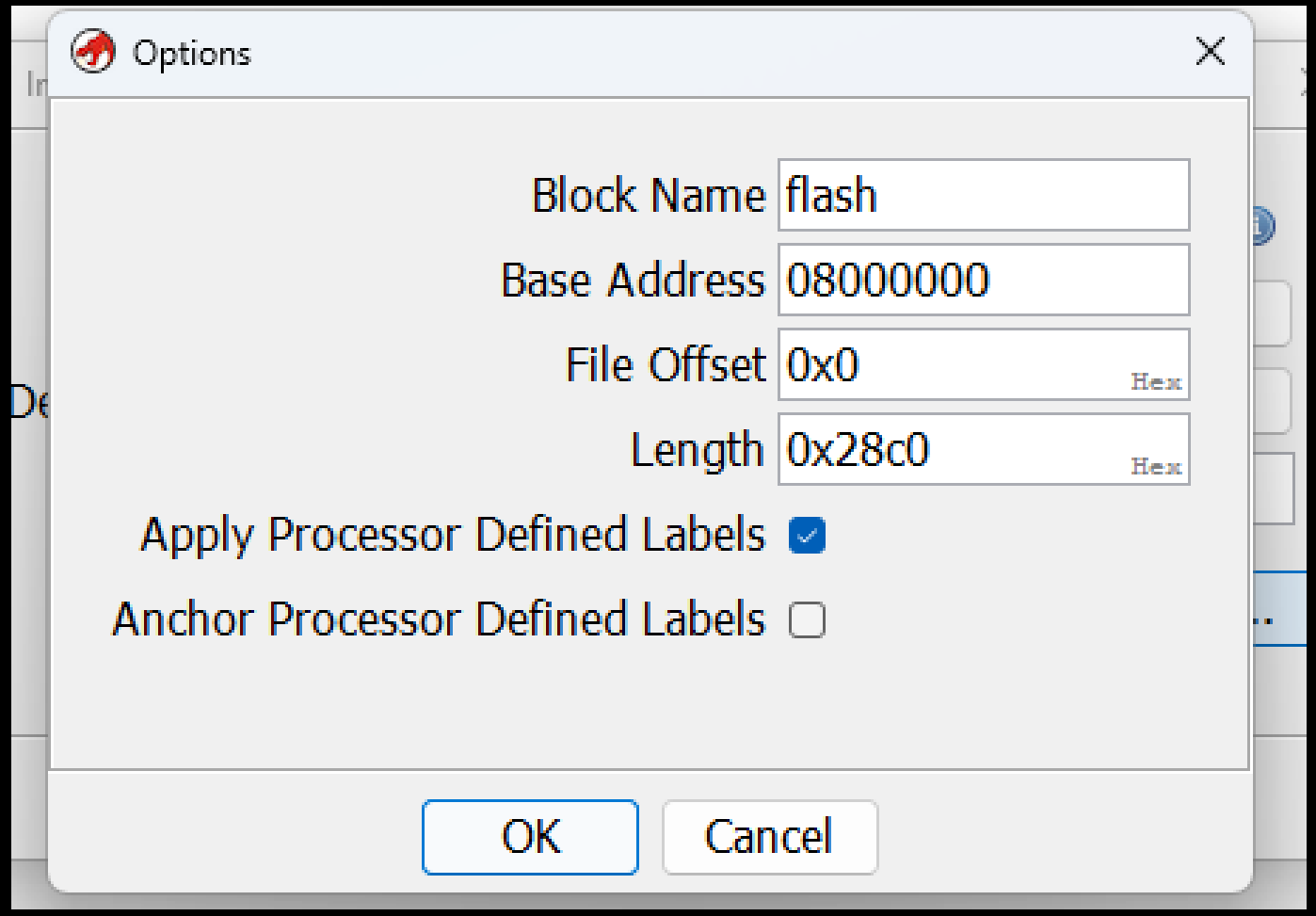
This information can typically be found in the datasheet and reference manual.



Advanced Reverse Engineering

Ghidra Basics

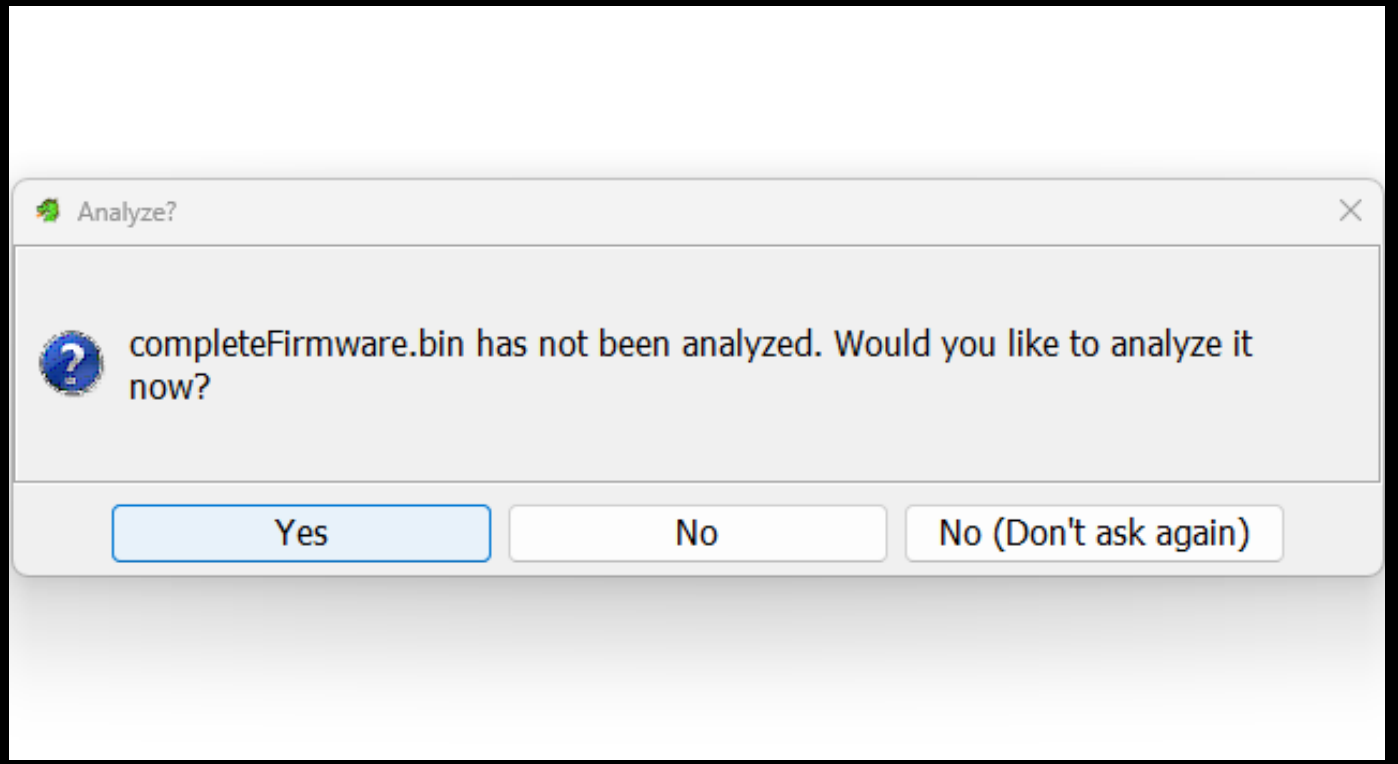
Add Mirrored Flash Memory Block



Advanced Reverse Engineering

Ghidra Basics

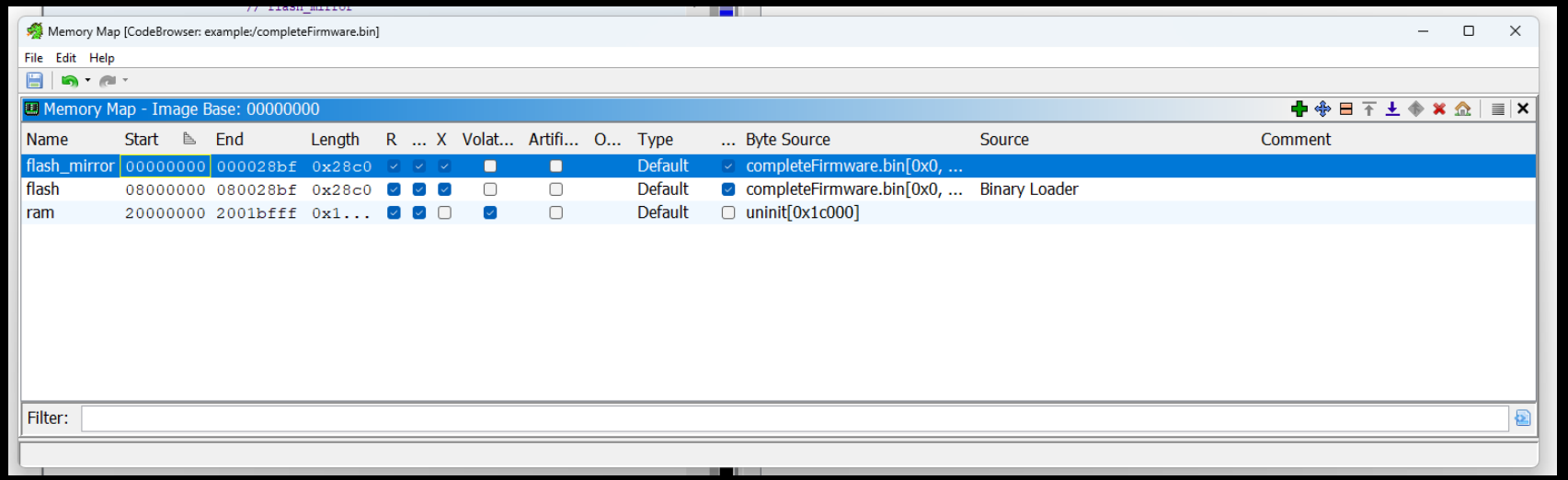
Do Not Analyze the Firmware Immediately



Advanced Reverse Engineering

Ghidra Basics

Basic Memory Map Configuration

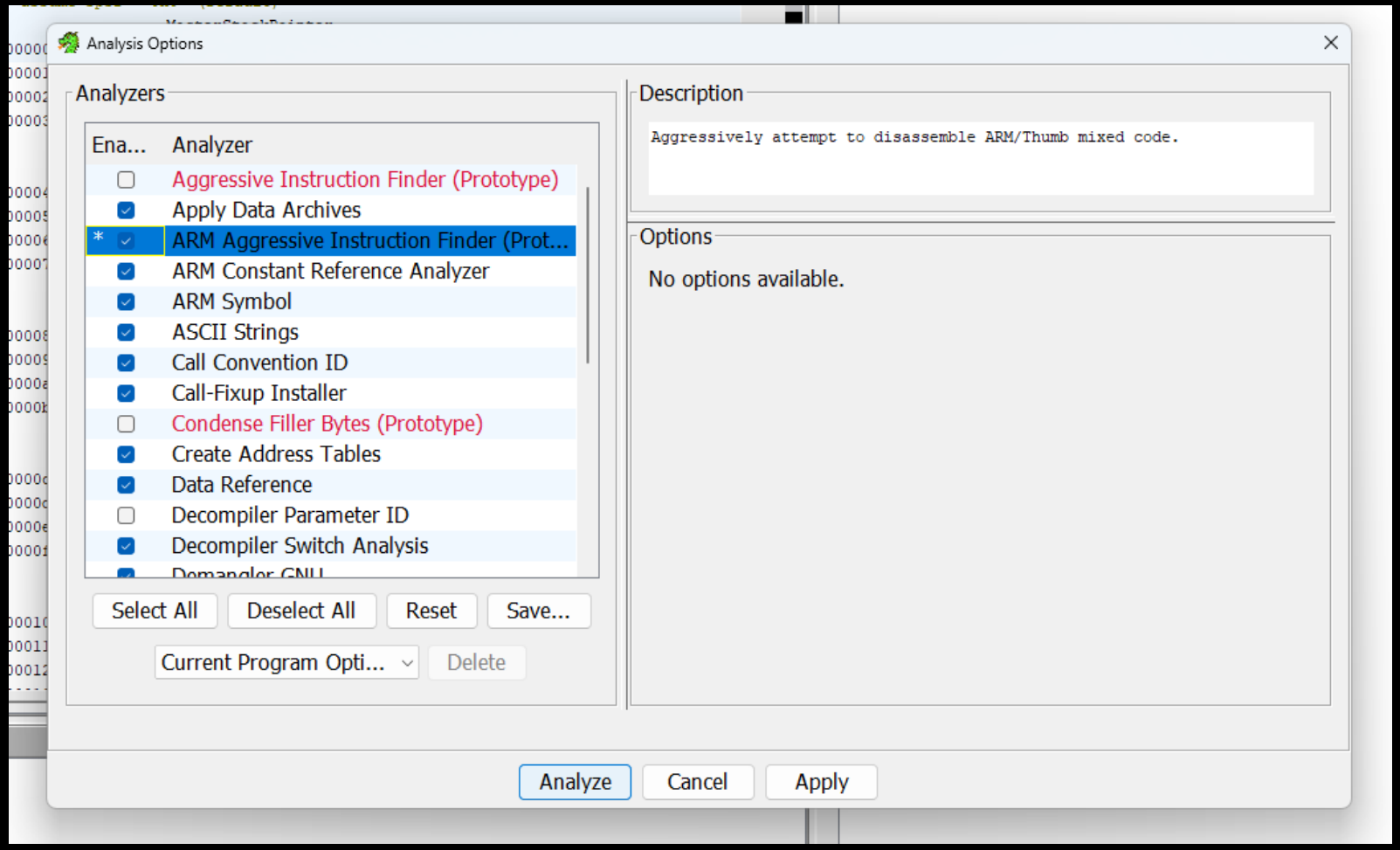


Advanced Reverse Engineering

Ghidra Basics

Analyse the Binary

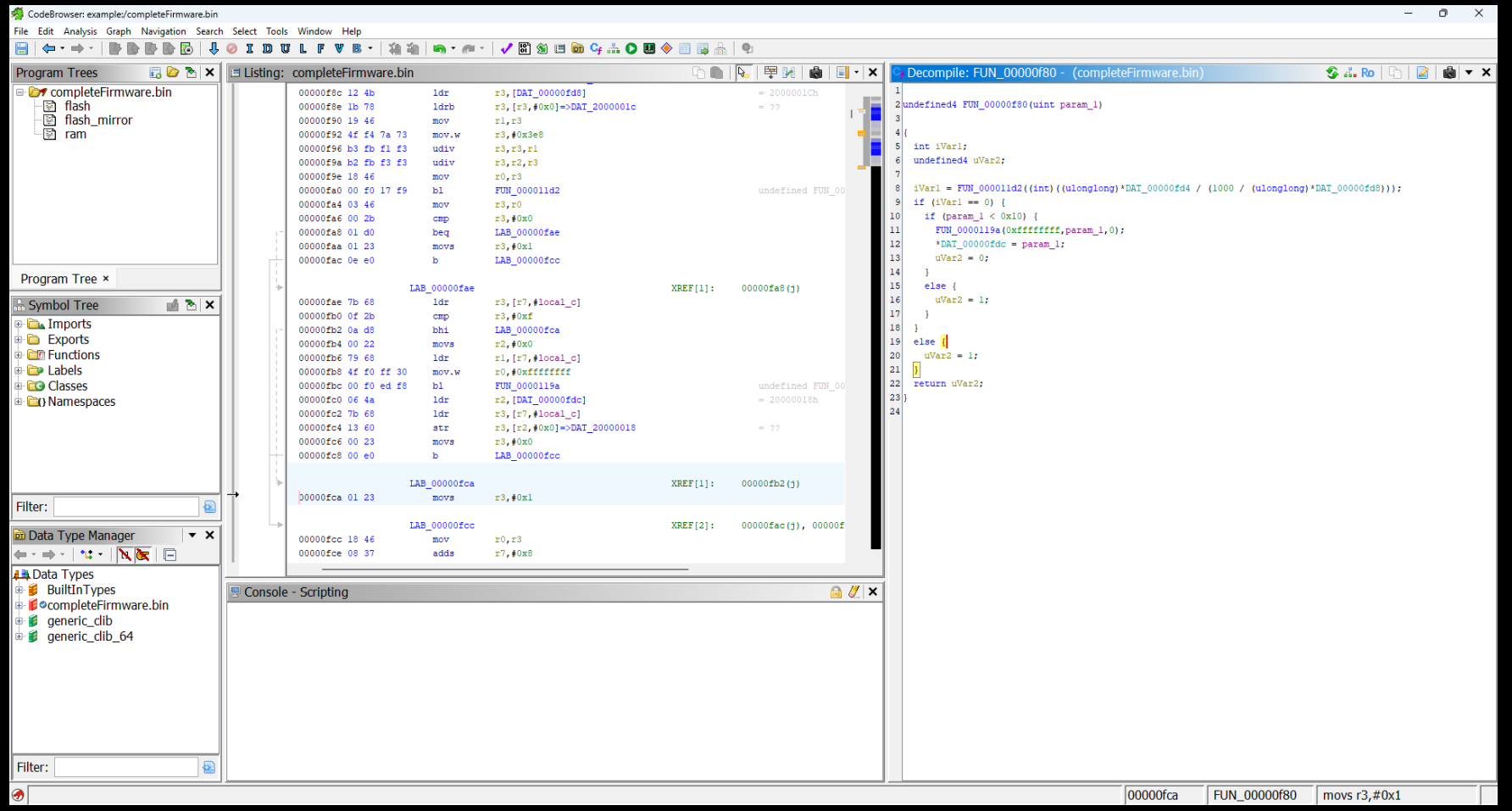
Ensure ARM Aggressive Instruction Finder is enabled



Advanced Reverse Engineering

Ghidra Basics

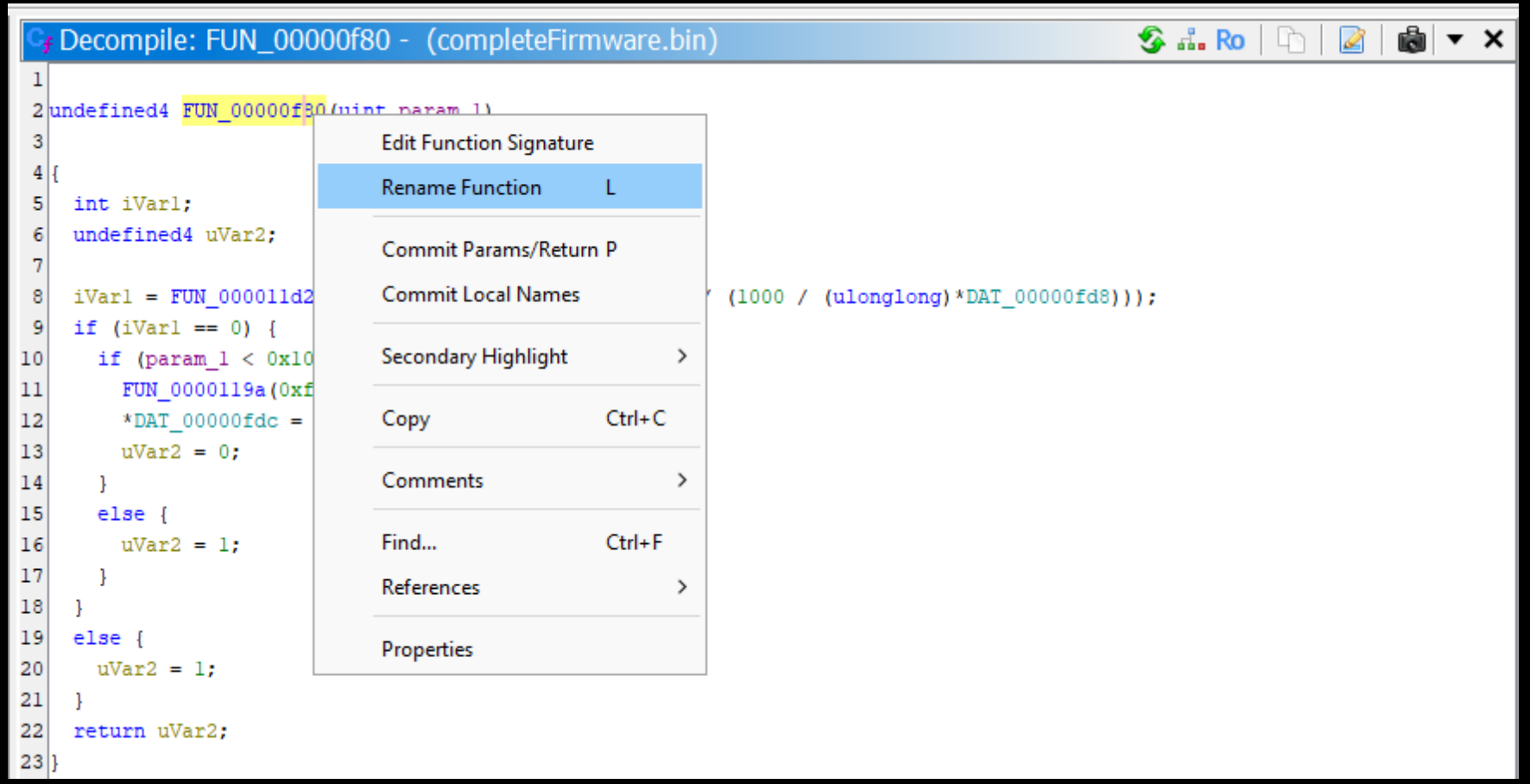
Ghidra View Basics



Advanced Reverse Engineering

Ghidra Basics

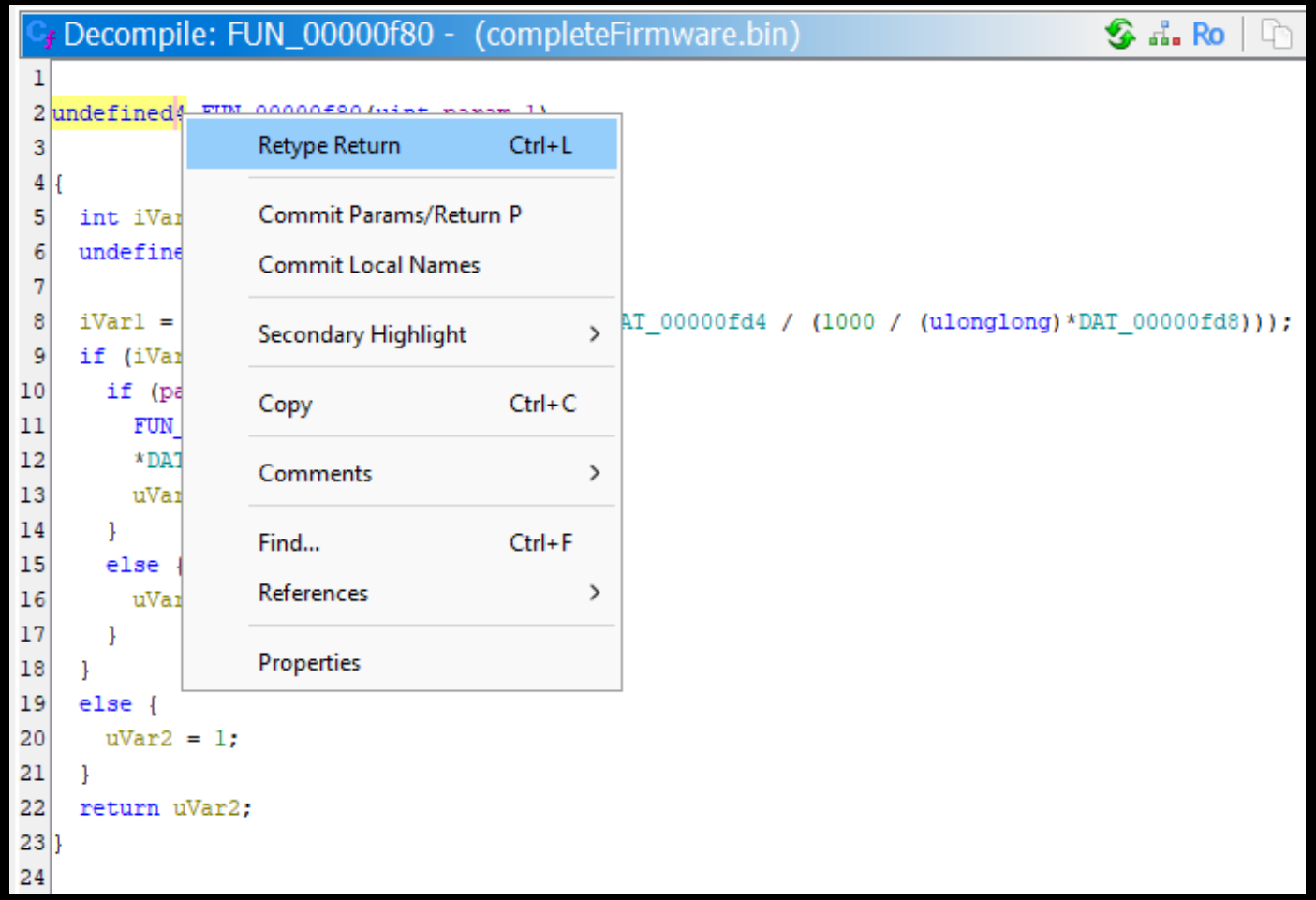
Renaming Functions (L)



Advanced Reverse Engineering

Ghidra Basics

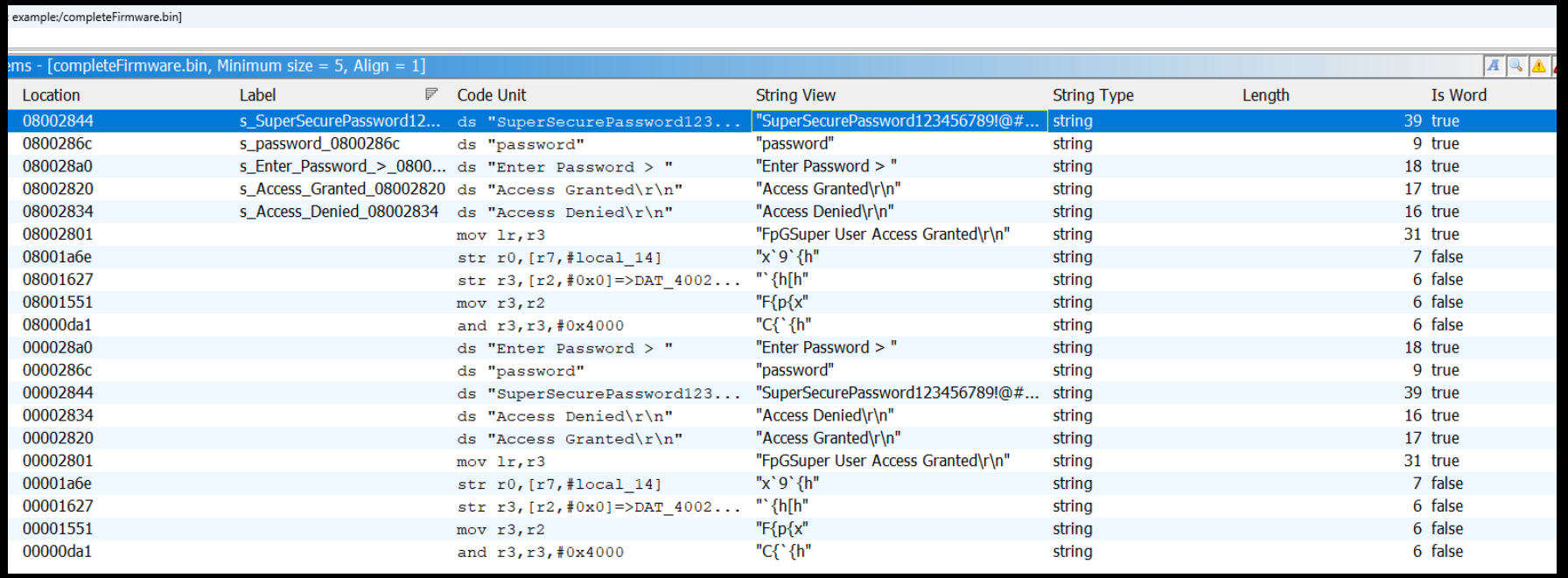
Retyping Variables (CTRL+L)



Advanced Reverse Engineering

Ghidra Basics

Searching for Strings


example:/completeFirmware.bin]

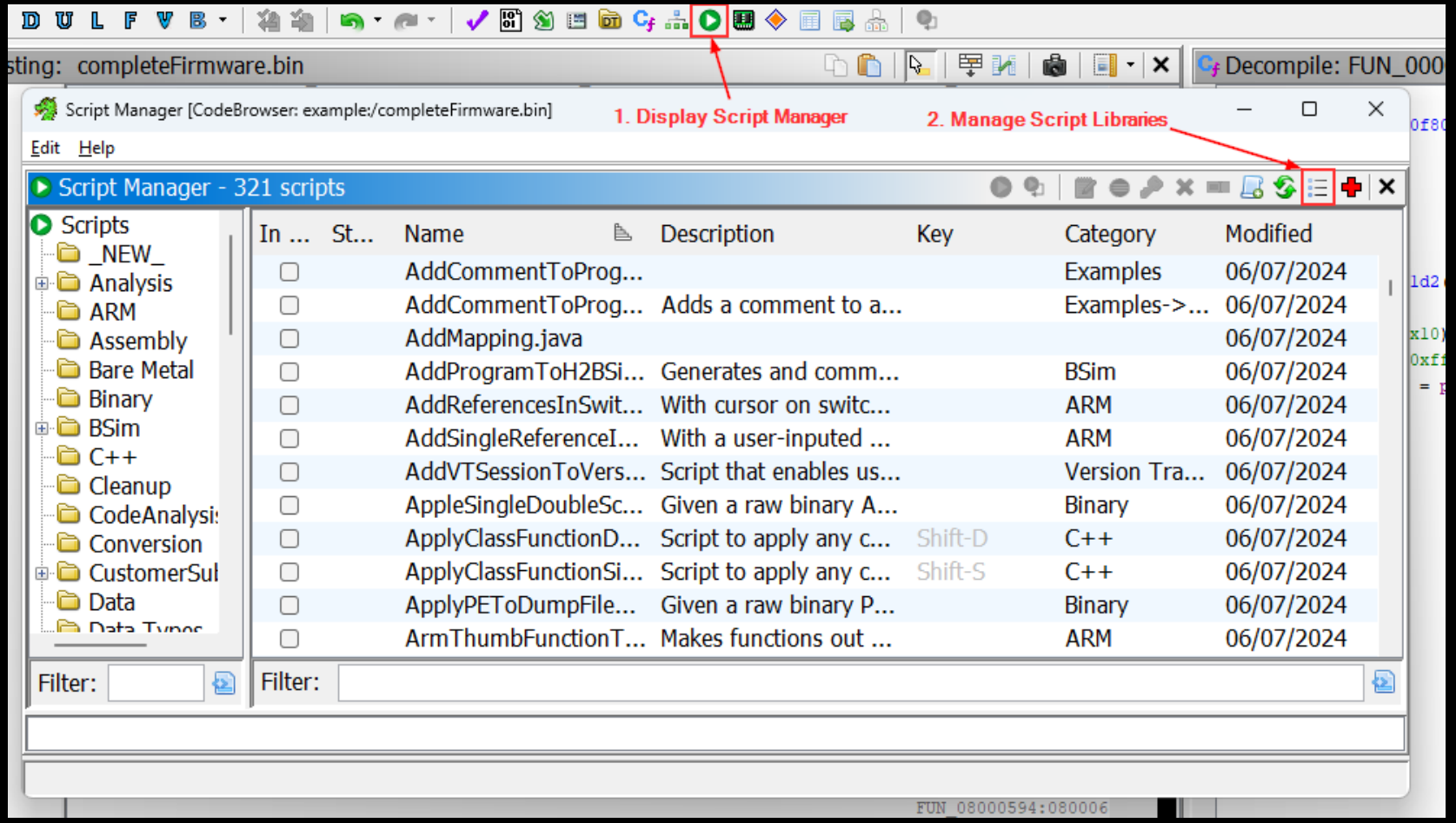
ems - [completeFirmware.bin, Minimum size = 5, Align = 1]

Location	Label	Code Unit	String View	String Type	Length	Is Word
08002844	s_SuperSecurePassword12...	ds "SuperSecurePassword123...	"SuperSecurePassword123456789!@#...	string	39	true
0800286c	s_password_0800286c	ds "password"	"password"	string	9	true
080028a0	s_Enter_Password_>_0800...	ds "Enter Password > "	"Enter Password > "	string	18	true
08002820	s_Access_Granted_08002820	ds "Access Granted\r\n"	"Access Granted\r\n"	string	17	true
08002834	s_Access_Denied_08002834	ds "Access Denied\r\n"	"Access Denied\r\n"	string	16	true
08002801		mov lr,r3	"FpGSuper User Access Granted\r\n"	string	31	true
08001a6e		str r0,[r7,#local_14]	"x`9`{h"	string	7	false
08001627		str r3,[r2,#0x0]=>DAT_4002...	"`{h[h"	string	6	false
08001551		mov r3,r2	"F{p{x"	string	6	false
08000da1		and r3,r3,#0x4000	"C{`{h"	string	6	false
000028a0		ds "Enter Password > "	"Enter Password > "	string	18	true
0000286c		ds "password"	"password"	string	9	true
00002844		ds "SuperSecurePassword123...	"SuperSecurePassword123456789!@#...	string	39	true
00002834		ds "Access Denied\r\n"	"Access Denied\r\n"	string	16	true
00002820		ds "Access Granted\r\n"	"Access Granted\r\n"	string	17	true
00002801		mov lr,r3	"FpGSuper User Access Granted\r\n"	string	31	true
00001a6e		str r0,[r7,#local_14]	"x`9`{h"	string	7	false
00001627		str r3,[r2,#0x0]=>DAT_4002...	"`{h[h"	string	6	false
00001551		mov r3,r2	"F{p{x"	string	6	false
00000da1		and r3,r3,#0x4000	"C{`{h"	string	6	false

Advanced Reverse Engineering

Ghidra Plugins

Manage Script Libraries

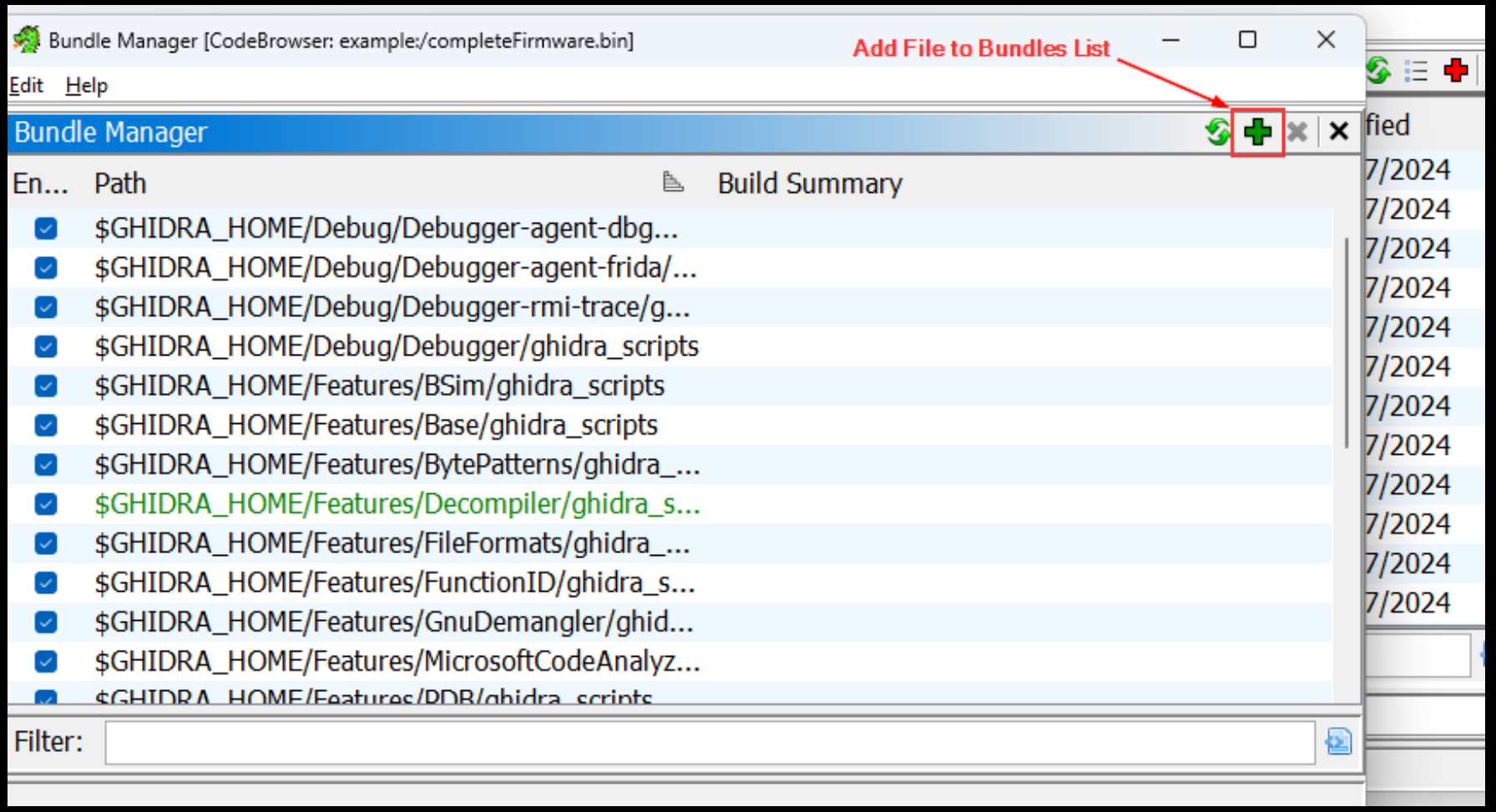


Advanced Reverse Engineering

Ghidra Plugins

Add File to Bundles List

Select source code directory, bundle, or bnd script



Advanced Reverse Engineering

Ghidra Plugin: SVD Loader (BSP)

SVD-Loader-Ghidra by leveldown-security

Programmatically processes an SVD (System View Description) file, which is an XML-based representation of a microcontroller's peripheral and register configuration.

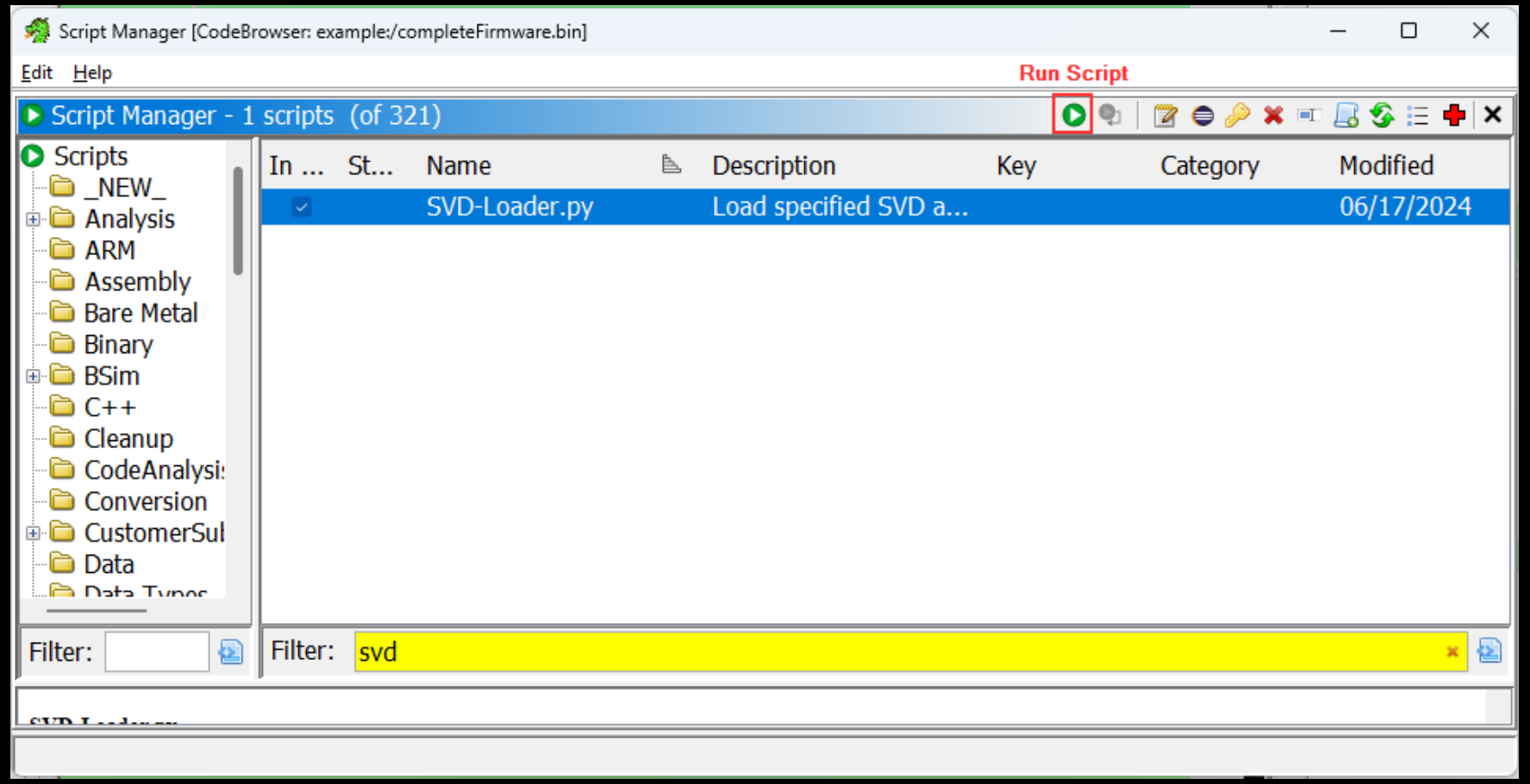
```
<description>STM32F429</description>
<!-- details about the cpu embedded in the device -->
<cpu>
  <name>CM4</name>
  <revision>r1p0</revision>
  <endian>little</endian>
  <mpuPresent>false</mpuPresent>
  <fpuPresent>false</fpuPresent>
  <nvicPrioBits>3</nvicPrioBits>
  <vendorSystickConfig>false</vendorSystickConfig>
</cpu>
<!--Bus Interface Properties-->
<!--Cortex-M4 is byte addressable-->
<addressUnitBits>8</addressUnitBits>
<!--the maximum data bit width accessible within a single transfer-->
<width>32</width>
<!--Register Default Properties-->
<size>0x20</size>
<resetValue>0x0</resetValue>
<resetMask>0xFFFFFFFF</resetMask>
<peripherals>
  <peripheral>
    <name>RNG</name>
    <description>Random number generator</description>
    <groupName>RNG</groupName>
    <baseAddress>0x50060800</baseAddress>
```

A dedicated CMSIS-SVD Repository can be used to gather target SVD files.

Advanced Reverse Engineering

Ghidra Plugin: SVD Loader (BSP)

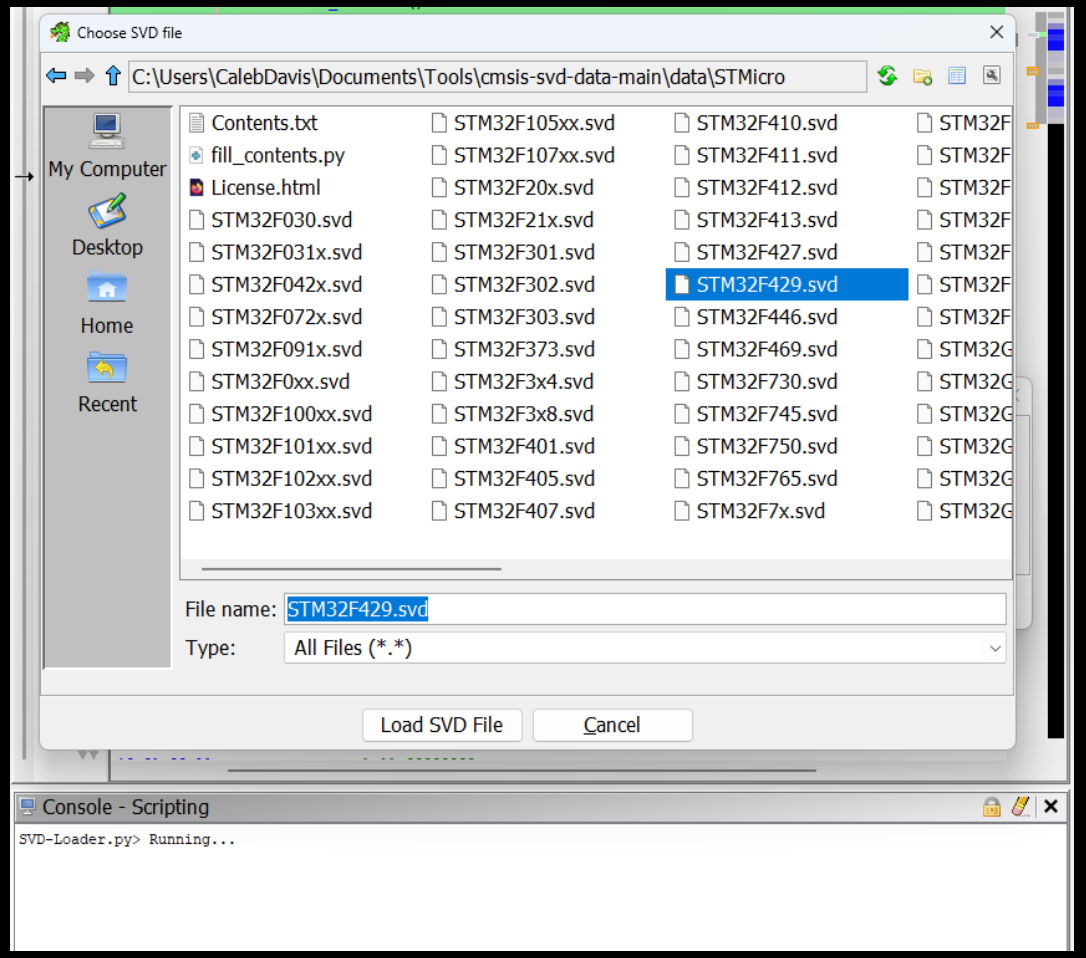
Run Script



Advanced Reverse Engineering

Ghidra Plugin: SVD Loader (BSP)

Choose SVD File Per Target Architecture

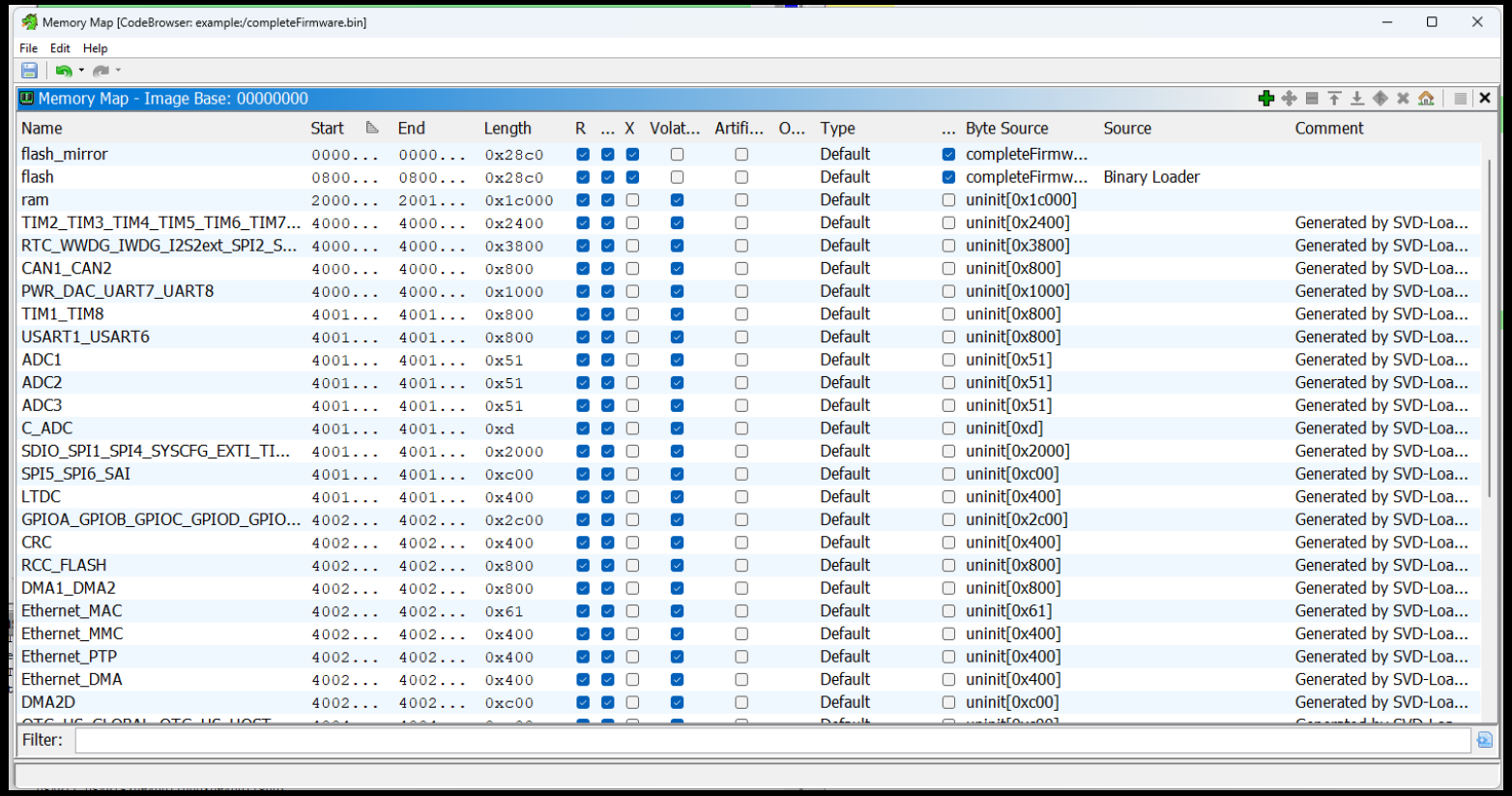


Advanced Reverse Engineering

Ghidra Plugin: SVD Loader (BSP)

Updated Memory Map

Added Peripheral Mappings



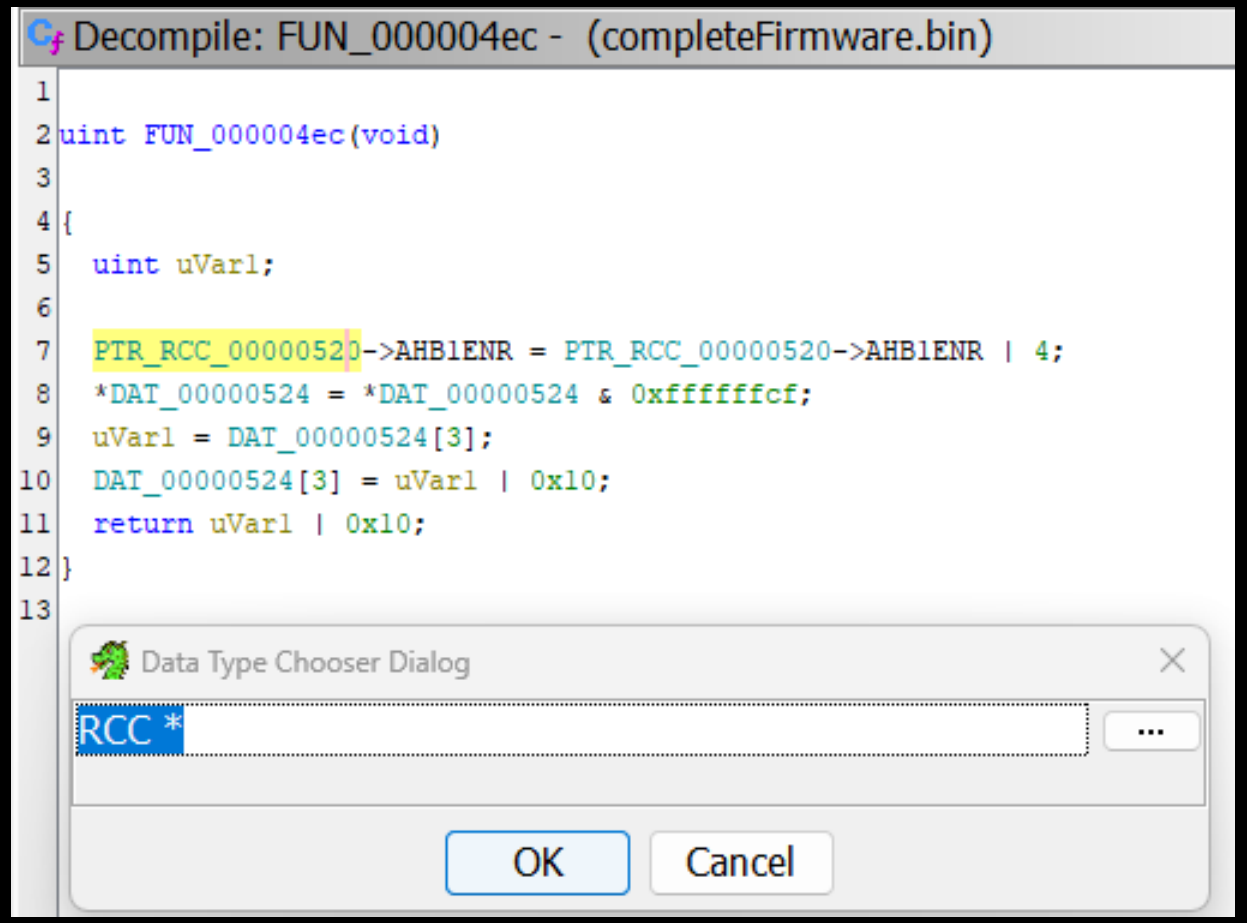
Name	Start	End	Length	R	...	X	Volat...	Artifi...	O...	Type	...	Byte Source	Source	Comment
flash_mirror	0000...	0000...	0x28c0	☑	☑	☑	☐	☐		Default	☑	completeFirmw...		
flash	0800...	0800...	0x28c0	☑	☑	☑	☐	☐		Default	☑	completeFirmw...	Binary Loader	
ram	2000...	2001...	0x1c000	☑	☑	☐	☑	☐		Default	☐	uninit[0x1c000]		
TIM2_TIM3_TIM4_TIM5_TIM6_TIM7...	4000...	4000...	0x2400	☑	☑	☐	☑	☐		Default	☐	uninit[0x2400]		Generated by SVD-Loa...
RTC_WWDG_IWDG_I2S2ext_SPI2_S...	4000...	4000...	0x3800	☑	☑	☐	☑	☐		Default	☐	uninit[0x3800]		Generated by SVD-Loa...
CAN1_CAN2	4000...	4000...	0x800	☑	☑	☐	☑	☐		Default	☐	uninit[0x800]		Generated by SVD-Loa...
PWR_DAC_UART7_UART8	4000...	4000...	0x1000	☑	☑	☐	☑	☐		Default	☐	uninit[0x1000]		Generated by SVD-Loa...
TIM1_TIM8	4001...	4001...	0x800	☑	☑	☐	☑	☐		Default	☐	uninit[0x800]		Generated by SVD-Loa...
USART1_USART6	4001...	4001...	0x800	☑	☑	☐	☑	☐		Default	☐	uninit[0x800]		Generated by SVD-Loa...
ADC1	4001...	4001...	0x51	☑	☑	☐	☑	☐		Default	☐	uninit[0x51]		Generated by SVD-Loa...
ADC2	4001...	4001...	0x51	☑	☑	☐	☑	☐		Default	☐	uninit[0x51]		Generated by SVD-Loa...
ADC3	4001...	4001...	0x51	☑	☑	☐	☑	☐		Default	☐	uninit[0x51]		Generated by SVD-Loa...
C_ADC	4001...	4001...	0xd	☑	☑	☐	☑	☐		Default	☐	uninit[0xd]		Generated by SVD-Loa...
SDIO_SPI1_SPI4_SYSCFG_EXTI_TI...	4001...	4001...	0x2000	☑	☑	☐	☑	☐		Default	☐	uninit[0x2000]		Generated by SVD-Loa...
SPI5_SPI6_SAI	4001...	4001...	0xc00	☑	☑	☐	☑	☐		Default	☐	uninit[0xc00]		Generated by SVD-Loa...
LTDC	4001...	4001...	0x400	☑	☑	☐	☑	☐		Default	☐	uninit[0x400]		Generated by SVD-Loa...
GPIOA_GPIOB_GPIOC_GPIOD_GPIO...	4002...	4002...	0x2c00	☑	☑	☐	☑	☐		Default	☐	uninit[0x2c00]		Generated by SVD-Loa...
CRC	4002...	4002...	0x400	☑	☑	☐	☑	☐		Default	☐	uninit[0x400]		Generated by SVD-Loa...
RCC_FLASH	4002...	4002...	0x800	☑	☑	☐	☑	☐		Default	☐	uninit[0x800]		Generated by SVD-Loa...
DMA1_DMA2	4002...	4002...	0x800	☑	☑	☐	☑	☐		Default	☐	uninit[0x800]		Generated by SVD-Loa...
Ethernet_MAC	4002...	4002...	0x61	☑	☑	☐	☑	☐		Default	☐	uninit[0x61]		Generated by SVD-Loa...
Ethernet_MMC	4002...	4002...	0x400	☑	☑	☐	☑	☐		Default	☐	uninit[0x400]		Generated by SVD-Loa...
Ethernet_PTP	4002...	4002...	0x400	☑	☑	☐	☑	☐		Default	☐	uninit[0x400]		Generated by SVD-Loa...
Ethernet_DMA	4002...	4002...	0x400	☑	☑	☐	☑	☐		Default	☐	uninit[0x400]		Generated by SVD-Loa...
DMA2D	4002...	4002...	0xc00	☑	☑	☐	☑	☐		Default	☐	uninit[0xc00]		Generated by SVD-Loa...

Advanced Reverse Engineering

Ghidra Plugin: SVD Loader (BSP)

Type Casting

New data types can be used to cast structures



Advanced Reverse Engineering

Ghidra Plugin: SVD Loader (BSP)

Code Comparison

```c
static uint8_t pc2InitThHardWay( void )
{
        // Initialize GPIOC Peripheral Clock (0x1 for bit 2 in RCC_AHB1ENR)
        RCC->AHB1ENR |= (1<<2);

        // Set port Input mode for PC2 (0x00 for bit 5 and 4 in GPIOC_MODER)
        GPIOC->MODER &= ~(0x00000030);

        // Reset Pull-Up Value for PC2 (0x00 for bit 5 and 4 in GPIOC_PUPDR)
        GPIOC->PUPDR &= ~(0x00000030);

        // Set port Pull-up for PC2 (0x01 for bit 5 and 4 in GPIOC_PUPDR)
        GPIOC->PUPDR |= 0x00000010;
}
```

Decompile: initPC2 - (completeFirmware.bin)

```c
1
2  uint initPC2(void)
3
4  {
5    uint pc2_pupdr_value;
6
7    PTR_RCC_00000520->AHB1ENR = PTR_RCC_00000520->AHB1ENR | 0b00000100;
8    PTR_GPIOC_00000524->MODER = PTR_GPIOC_00000524->MODER & 0xffffffcf;
9    pc2_pupdr_value = PTR_GPIOC_00000524->PUPDR | 0x10;
10   PTR_GPIOC_00000524->PUPDR = pc2_pupdr_value;
11   return pc2_pupdr_value;
12 }
13
```

Advanced Reverse Engineering

Ghidra Plugin: Type Loader (HAL)

typeLoader by So11Deo6loria

Simplifies the process of applying custom data types to memory regions or variables in a binary. It programmatically parses external type definitions, such as structs and enums, and integrates them into the reverse engineering workspace, mapping these types to relevant sections of memory or code.

Advanced Reverse Engineering

Ghidra Plugin: Type Loader (HAL)

Type Extractor

Automate tool to parse HAL header files into JSON structure

```
 /mnt/c/Users/CalebDavis/Documents/Tools/typeLoader  on  main !1 ?3 ····· took 4s ⧖ at 04:37:46 PM ⊙
❯ python typeExtractor.py -d /mnt/c/Users/CalebDavis/Documents/CodeStuff/basicFirmwareExample/Drivers/STM32F4xx_HAL_Driver/Inc/ -v ST -o stm32Test.json
Processing Directory: /mnt/c/Users/CalebDavis/Documents/CodeStuff/basicFirmwareExample/Drivers/STM32F4xx_HAL_Driver/Inc/
Processing Filename: Legacy
Processing Filename: stm32f4xx_hal.h
Processing Filename: stm32f4xx_hal.h
Processing Filename: stm32f4xx_hal_cortex.h
Processing Filename: stm32f4xx_hal_cortex.h
Processing Filename: stm32f4xx_hal_def.h
Processing Filename: stm32f4xx_hal_def.h
```

```
 /mnt/c/Users/CalebDavis/Documents/Tools/typeLoader  on  main !1 ?4 ····· at 04:38:28 PM ⊙
❯ head -n 10 stm32Test.json
{
    "enum": {
        "HAL_TickFreqTypeDef": {
            "HAL_TICK_FREQ_10HZ         = 100U": {
                "value": null,
                "description": null
            },
            "HAL_TICK_FREQ_100HZ        = 10U": {
                "value": null,
                "description": null
```

Advanced Reverse Engineering

Ghidra Plugin: Type Loader (HAL)

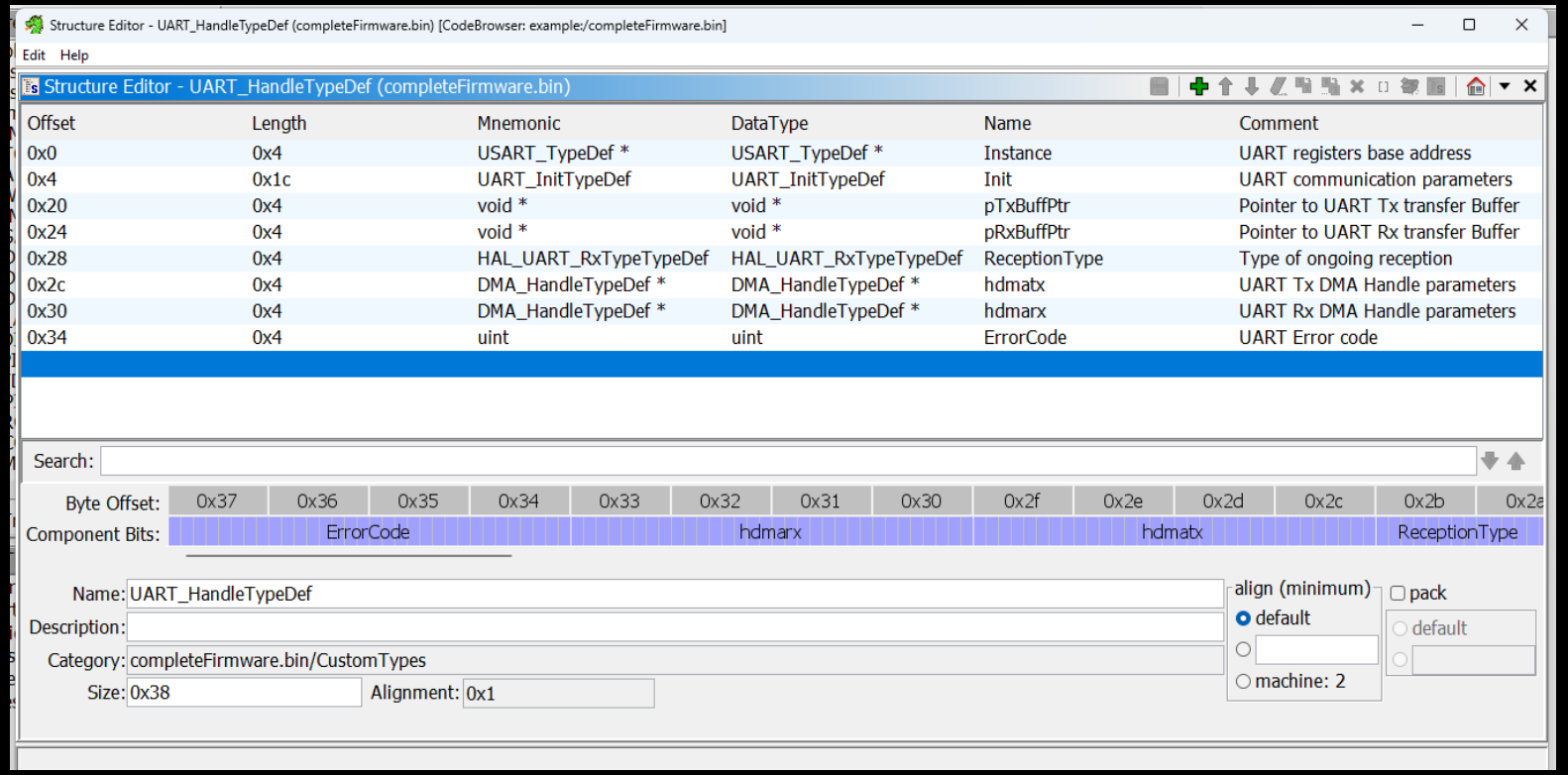
TypeLoader Output

```
typeLoader.py> Running...
Initialization Complete
Error: Custom data type 'uint8_t' not found for pointer 'uint8_t *'.
Error: Custom data type 'uint8_t' not found for pointer 'uint8_t *'.
Created Data Types:
        Enums:
                HAL_StatusTypeDef
                HAL_LockTypeDef
                HAL_UART_StateTypeDef
                GPIO_PinState
                HAL_DMA_StateTypeDef
                HAL_LockTypeDef
                HAL_DMA_StateTypeDef
                HAL_LockTypeDef
                HAL_UART_StateTypeDef
                HAL_UART_StateTypeDef
        Structs:
                GPIO_InitTypeDef
                GPIO_TypeDef
                DMA_Stream_TypeDef
                DMA_TypeDef
                DMA_InitTypeDef
                DMA_HandleTypeDef
                HAL_UART_RxTypeTypeDef
                USART_TypeDef
                UART_InitTypeDef
                UART_HandleTypeDef
```

Advanced Reverse Engineering

Ghidra Plugin: Type Loader (HAL)

Custom HAL Data Types



Advanced Reverse Engineering

Ghidra Plugin: Type Loader (HAL)

Code Comparison

```c
static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```c
  int errorCheck;

  uartHandle->Instance = DAT_080008c0;
  (uartHandle->Init).BaudRate = 115200;
  (uartHandle->Init).WordLength = 0;
  (uartHandle->Init).StopBits = 0;
  (uartHandle->Init).Parity = 0;
  (uartHandle->Init).Mode = 0b00001100;
  (uartHandle->Init).HwFlowCtl = 0;
  (uartHandle->Init).OverSampling = 0;
  errorCheck = uartInit(uartHandle);
  if (errorCheck != 0) {
    Error_Handler();
  }
  return;
```

Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

BSim by the NSA

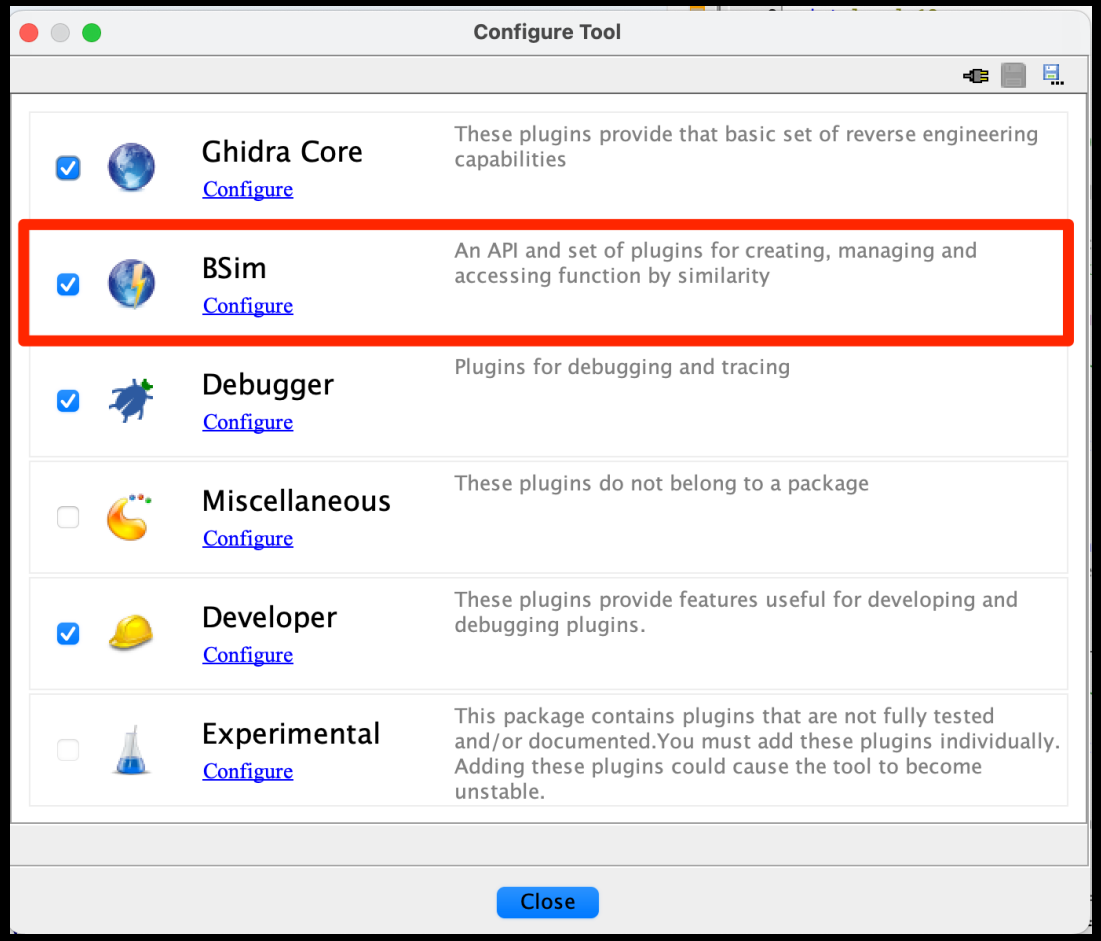
BSim is Ghidra’s built-in plugin designed to accelerate reverse engineering of stripped or firmware binaries. It enables rapid identification and annotation of functions by performing fuzzy matching against known libraries, even when compiler options or exact versions are unknown.

Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

Enabling BSim

File -> Configure -> Enable BSim

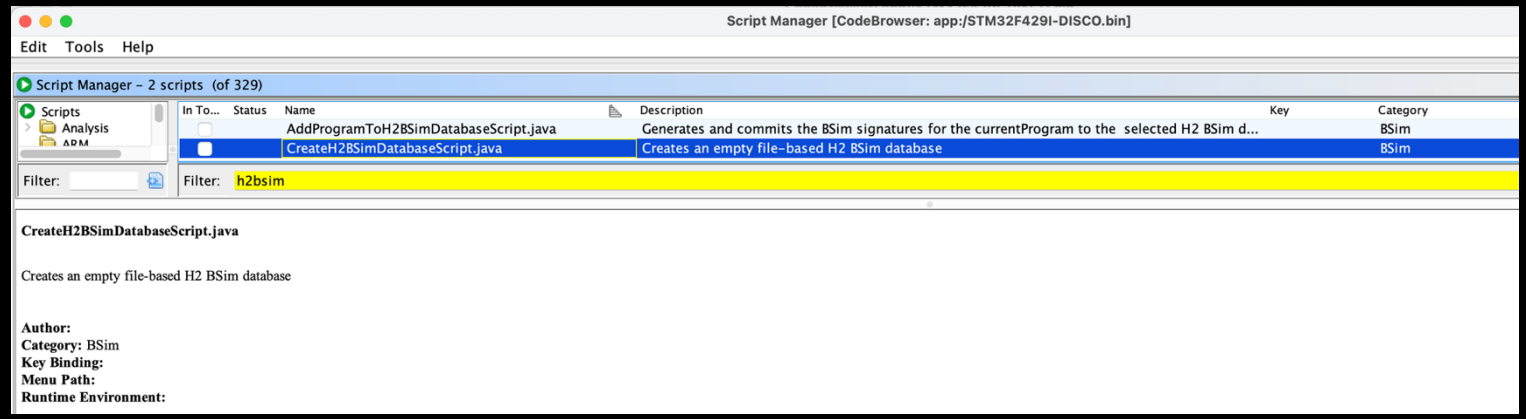


Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

Create an H2 BSim Database

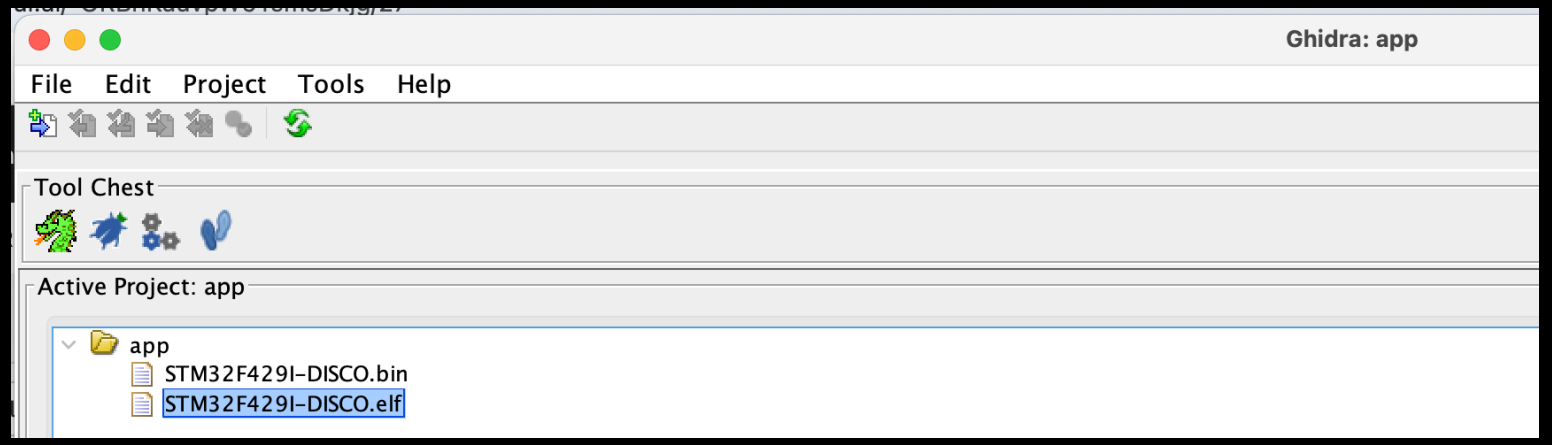
CreateH2BSimDatabaseScript.java



Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

Build and RE a Dedicated Binary (.elf)

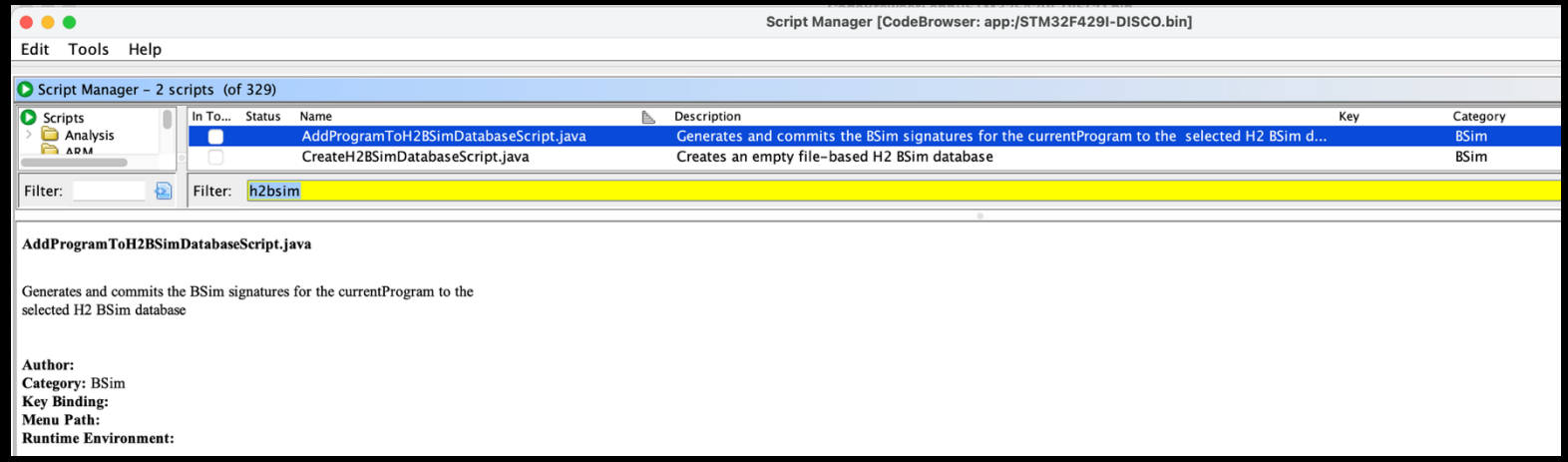


Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

Add Reversed Program to H2 BSim Database

AddProgramToH2BSimDatabaseScript.java

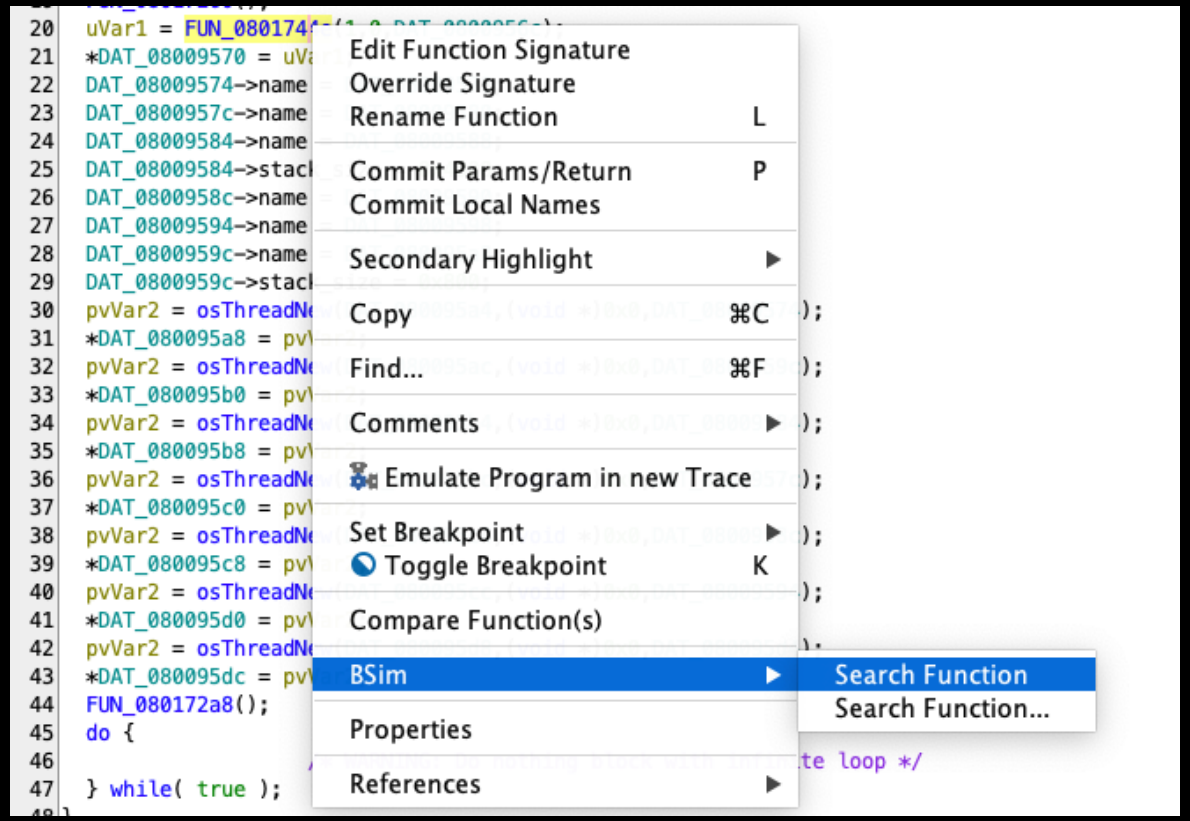


Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

Search Function

Right Click Function -> BSim -> Search Function

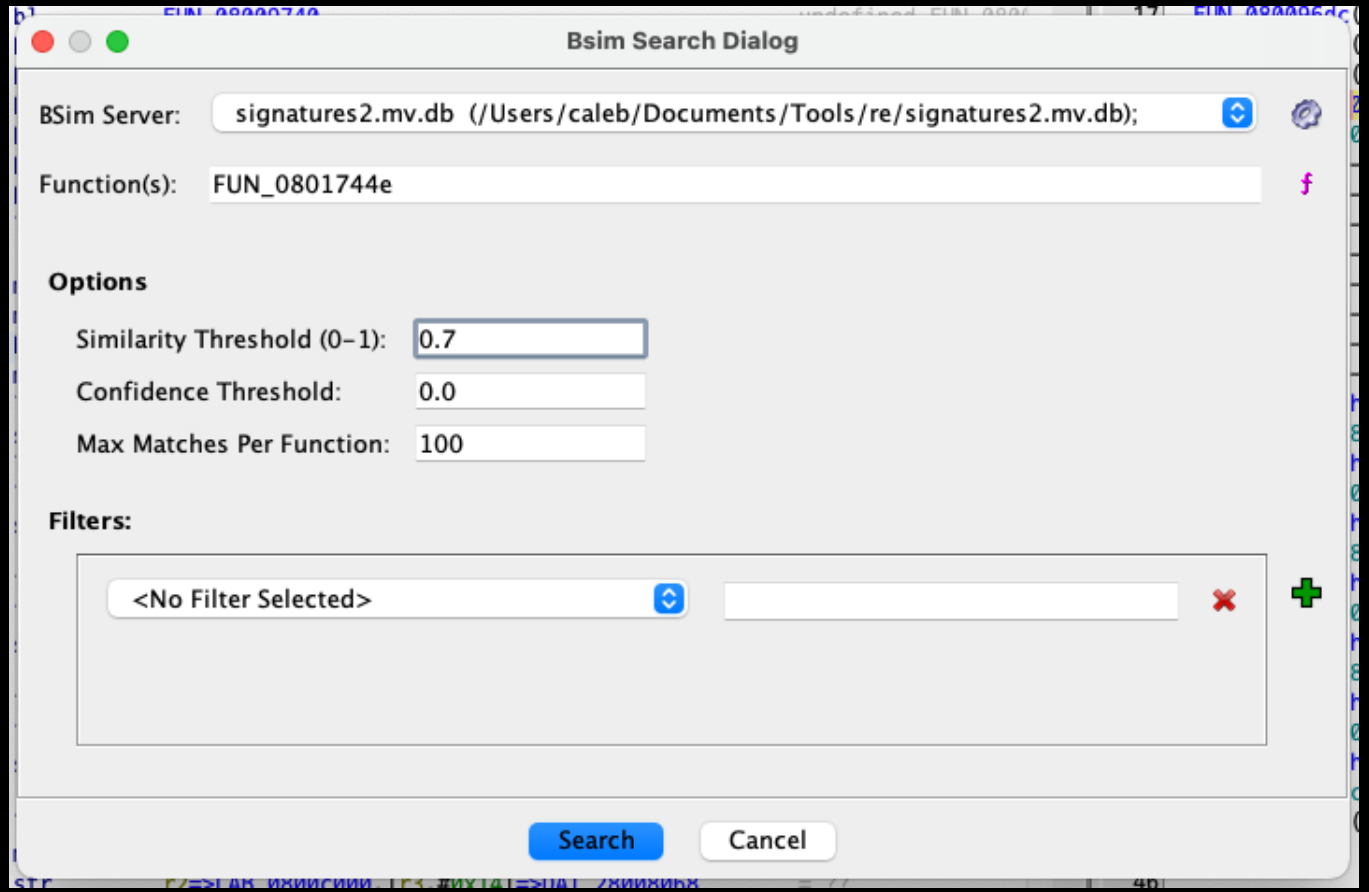


Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

Set Search Options

Similarity / Confidence

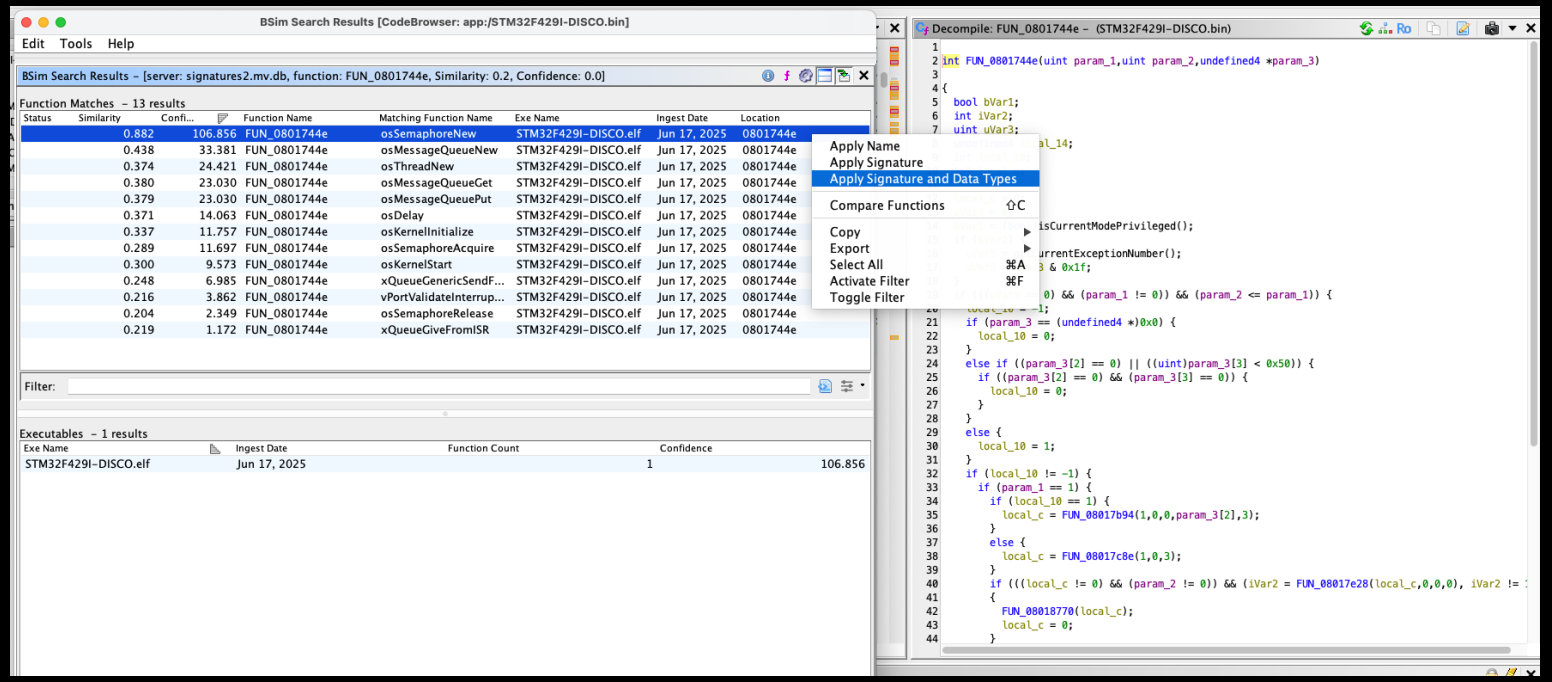


Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

Apply Signature and Data Types

Right Click -> Apply Signature and Data Types



Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

Code Comparison

cmsis_os2.c

```
444 osThreadId_t osThreadNew (osThreadFunc_t func, void *argument, const osThreadAttr_t *attr) {
445   const char *name;
446   uint32_t stack;
447   TaskHandle_t hTask;
448   UBaseType_t prio;
449   int32_t mem;
450
451   hTask = NULL;
452
453   if (!IS_IRQ() && (func != NULL)) {
454     stack = configMINIMAL_STACK_SIZE;
455     prio  = (UBaseType_t)osPriorityNormal;
456
457     name = NULL;
458     mem  = -1;
459
460     if (attr != NULL) {
461       if (attr->name != NULL) {
462         name = attr->name;
463       }
464       if (attr->priority != osPriorityNone) {
465         prio = (UBaseType_t)attr->priority;
466       }
467
468       if ((prio < osPriorityIdle) || (prio > osPriorityISR) || ((attr->attr_bits & osThreadJoinable) == osTh
469         return (NULL);
470       }
471
472       if (attr->stack_size > 0U) {
473         /* In FreeRTOS stack is not in bytes, but in sizeof(StackType_t) which is 4 on ARM ports.       */
474         /* Stack size should be therefore 4 byte aligned in order to avoid division caused side effects */
475         stack = attr->stack_size / sizeof(StackType_t);
476       }
477
478       if ((attr->cb_mem    != NULL) && (attr->cb_size    >= sizeof(StaticTask_t)) &&
479           (attr->stack_mem != NULL) && (attr->stack_size >  0U)) {
480         mem = 1;
481       }
482       else {
```

Decompile: osThreadNew - (STM32F429I-DISCO.bin)

```
1
2 osThreadId_t osThreadNew(osThreadFunc_t func,void *argument,osThreadAttr_t *attr)
3
4 {
5   bool bVar1;
6   BaseType_t BVar2;
7   TaskHandle_t local_20;
8   uint local_1c;
9   int local_18;
10  osPriority_t local_14;
11  uint local_10;
12  char *local_c;
13
14  local_20 = (TaskHandle_t)0x0;
15  local_1c = 0;
16  bVar1 = (bool)isCurrentModePrivileged();
17  if (bVar1) {
18    local_1c = getCurrentExceptionNumber();
19    local_1c = local_1c & 0x1f;
20  }
21  if ((local_1c == 0) && (func != (osThreadFunc_t)0x0)) {
22    local_10 = 0x80;
23    local_14 = osPriorityNormal;
24    local_c = (char *)0x0;
25    local_18 = -1;
26    if (attr == (osThreadAttr_t *)0x0) {
27      local_18 = 0;
28    }
29    else {
30      if (attr->name != (char *)0x0) {
31        local_c = attr->name;
```

Q&A