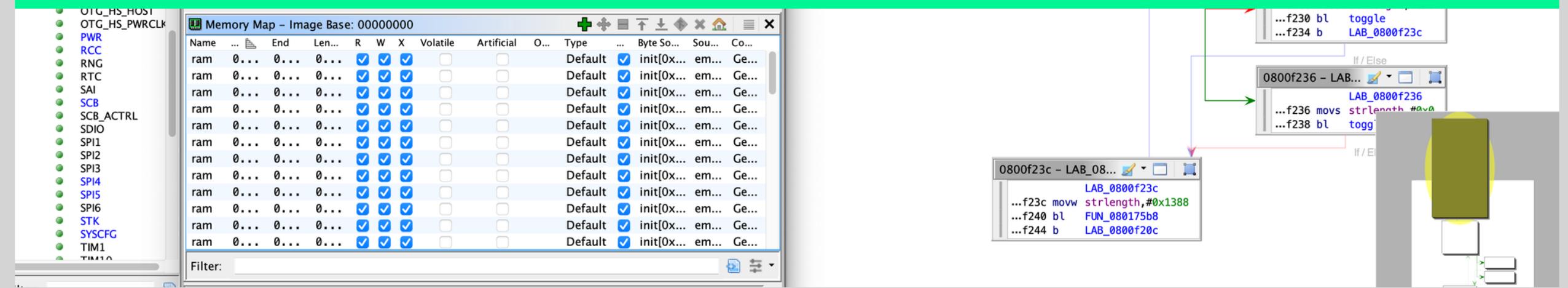


# Bare Metal Reverse Engineering





# Agenda

---

1. Bare Metal
2. Firmware Overview
3. ARM Assembly
4. Firmware Development
5. Basic Reverse Engineering
6. Advanced Reverse Engineering

# Whoami

---



Caleb Davis

Co-Founder of [SolaSec](#)

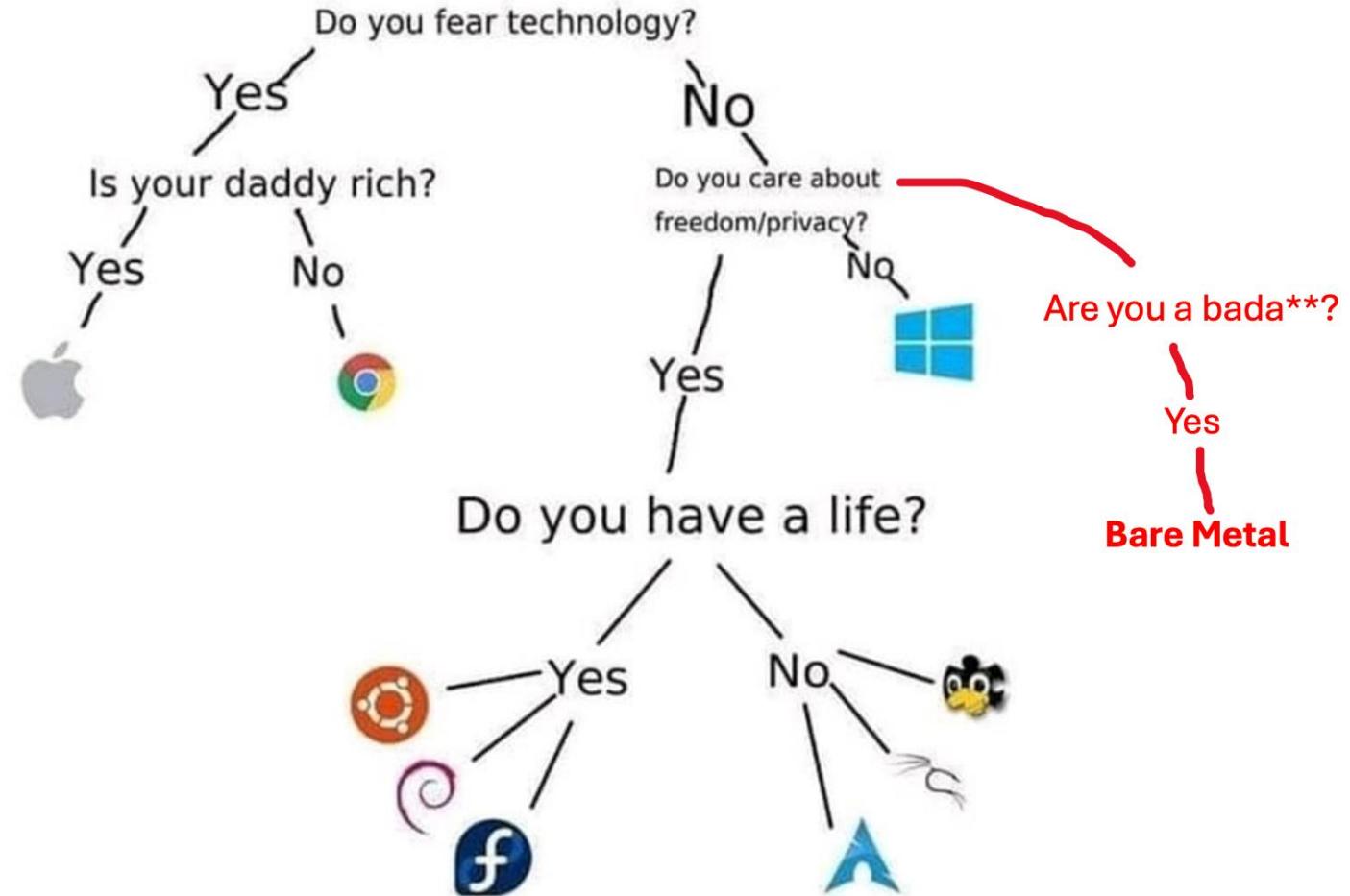
Penetration Tester | EE | Reverse Engineer

@So11Deo6loria

## Bare Metal

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

## Choosing an OS



Bare Metal Flow Diagram

# Bare Metal

---

## Key Considerations

### Minimal Privilege Separation

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

### Lightweight Scheduling, Minimal Abstraction

Real-Time Operating Systems (RTOS) provide multitasking 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 raw 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 optional 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.

```
1/* USER CODE BEGIN 1 */  
2  
3/* USER CODE END 1 */  
4  
5/* MCU Configuration-----  
6  
7/* Reset of all peripherals, Initializes the Flash interface and the Systick.  
8HAL_Init();  
9  
10/* USER CODE BEGIN Init */  
11  
12/* USER CODE END Init */  
13  
14/* Configure the system clock */  
15SystemClock_Config();  
16  
17/* USER CODE BEGIN SysInit */  
18  
19/* USER CODE END SysInit */  
20  
21/* Initialize all configured peripherals */  
22MX_GPIO_Init();  
23MX_CRC_Init();  
24MX_I2C3_Init();  
25MX_SPI5_Init();  
26MX_FMC_Init();  
27MX_LTDC_Init();  
28MX_DMA2D_Init();  
29MX_TouchGFX_Init();  
30/* USER CODE BEGIN 2 */  
31  
32/* USER CODE END 2 */  
33  
34/* Init scheduler */  
35osKernelInitialize();  
36  
37/* USER CODE BEGIN RTOS_MUTEX */  
38/* add mutexes, ... */  
39/* USER CODE END RTOS_MUTEX */  
40  
41/* USER CODE BEGIN RTOS_SEMAPHORES */  
42/* add semaphores, ... */  
43  
44i2C_SemaphoreHandle = osSemaphoreNew( 1, 0, &i2c_SemaphoreConfig );  
45  
46/* USER CODE END RTOS_SEMAPHORES */
```

```
99     uint8_t debugFlag = gDeviceStatus
100
101    // 4. Parse firmware version: "X.Y.Z"
102    uint16_t fwValue = 0;
103    int major = 0, minor = 0;
104    sscanf(gDeviceStatus.firmwareVersion,
105           "%d.%d", &major, &minor);
106
107    // 5. Format into global string
108    sprintf(deviceID, sizeof(deviceID),
109             "%s%02d", pinSuffix, baudSum, debugFlag);
110 }
111
112 /**
113  * @brief SPI4 Initialization Function
114  * @param None
115  * @retval None
116 */
117 static void MX_SPI4_Init (void)
118 {
119     /** SPI4 GPIO Configuration
120      PE4      -----> CS
121      PE2      -----> SPI4_SCK
122      PE5      -----> SPI4_MISO
123      PE6      -----> SPI4_MOSI
124     */
125     GPIO_InitTypeDef GPIO_InitStruct;
126
127     /* CS Configuration */
128     /*Configure GPIO pin Output Level*/
129     HAL_GPIO_WritePin (GPIOE, GPIO_PIN_4, GPIO_PIN_SET);
130
131     /*Configure GPIO CS pin: PE4 */
132     GPIO_InitStruct.Pin = GPIO_PIN_4;
133     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
134     GPIO_InitStruct.Pull = GPIO_NOPULL;
135     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
136     HAL_GPIO_Init (GPIOE, &GPIO_InitStruct);
137
138     /* SPI4 parameter configuration*/
139     hspi4.Instance = SPI4;
140     hspi4.Init.Mode = SPI_MODE_MASTER;
141     hspi4.Init.Direction = SPI_DIRECTION_2LINES;
142     hspi4.Init.DataSize = SPI_DATASIZE_8BIT;
143     hspi4.Init.CLKPolarity = SPI_POLARITY_LOW;
144     hspi4.Init.CLKPhase = SPI_PHASE_1EDGE;
145     hspi4.Init.NSS = SPI_NSS_HARD_OUTPUT;
```

# 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



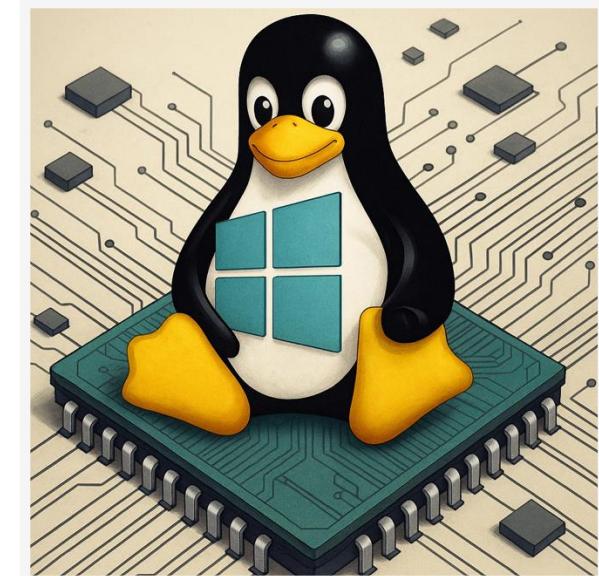
### Bare-Metal

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.

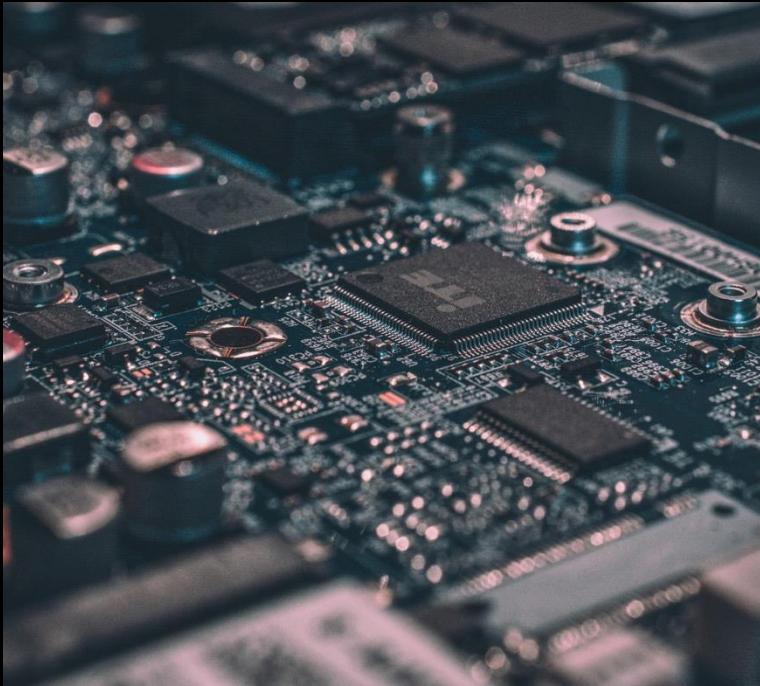


### Embedded OS

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

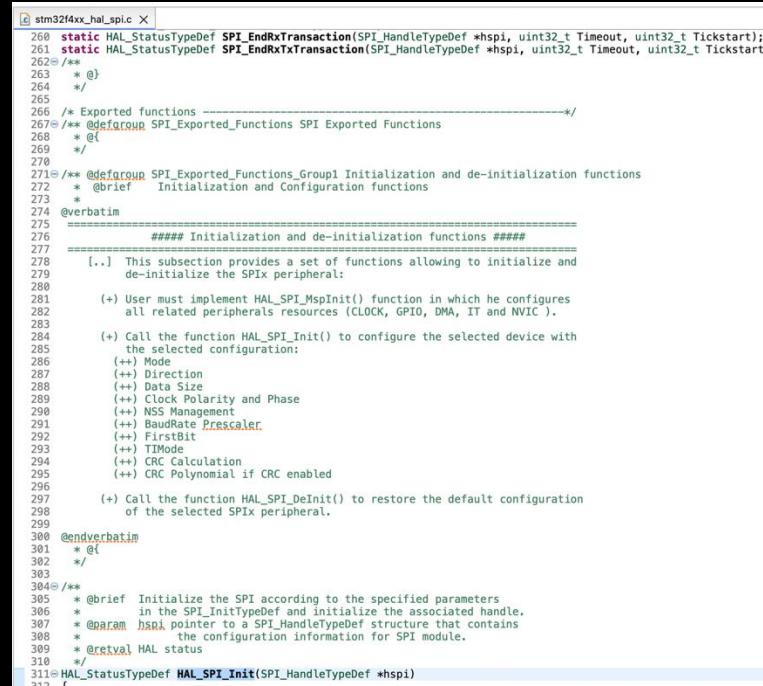
# Firmware Overview

# Key Software Components



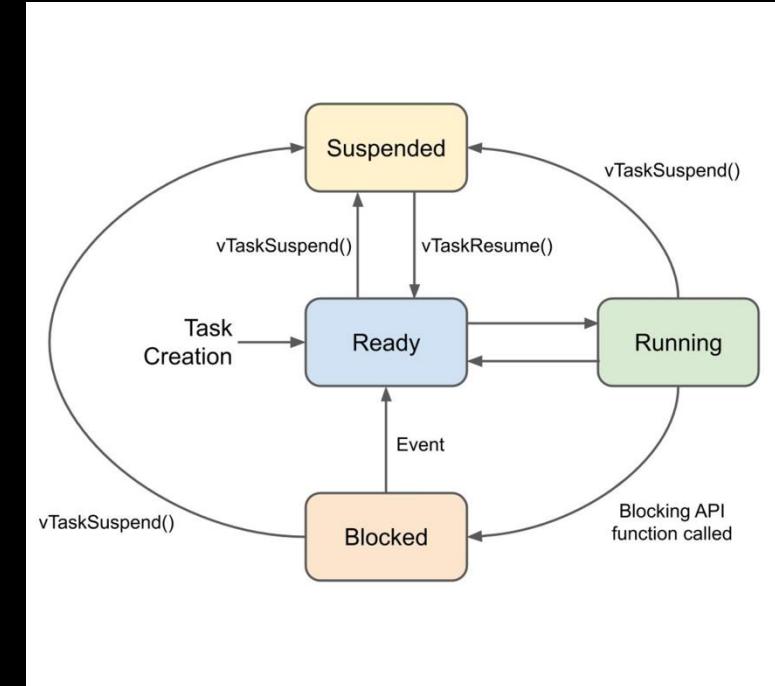
## Board Support Package (BSP)

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

intel. arm

Intel x86 / x86\_64

ARM



RISC-V



MIPS

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

```
#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>: strb  r3, [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   r3, #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?

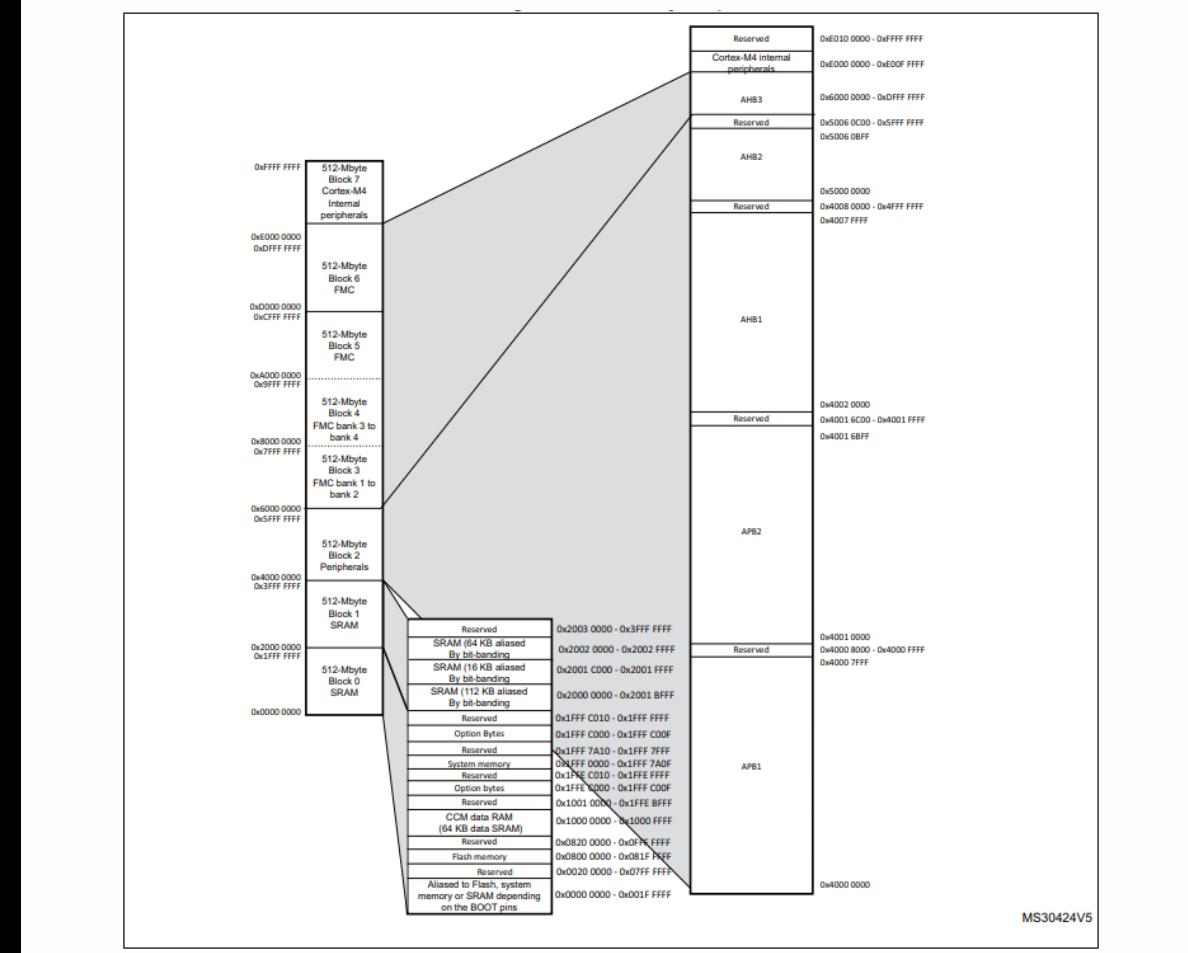
### Steps

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	<a href="#">Section 24.4.4: RNG register map on page 774</a>
0x5006 0400 - 0x5006 07FF	HASH		<a href="#">Section 25.4.9: HASH register map on page 798</a>
0x5006 0000 - 0x5006 03FF	CRYP		<a href="#">Section 23.6.13: CRYP register map on page 766</a>
0x5005 0000 - 0x5005 03FF	DCMI		<a href="#">Section 15.8.12: DCMI register map on page 481</a>
0x5000 0000 - 0x5003 FFFF	USB OTG FS		<a href="#">Section 34.16.6: OTG_FS register map on page 1329</a>
0x4004 0000 - 0x4007 FFFF	USB OTG HS	AHB1	<a href="#">Section 35.12.6: OTG_HS register map on page 1475</a>
0x4002 B000 - 0x4002 BBFF	DMA2D		<a href="#">Section 11.5: DMA2D registers on page 355</a>
0x4002 8000 - 0x4002 93FF	ETHERNET MAC		<a href="#">Section 33.8.5: Ethernet register maps on page 1239</a>
0x4002 6400 - 0x4002 67FF	DMA2		<a href="#">Section 10.5.11: DMA register map on page 338</a>
0x4002 6000 - 0x4002 63FF	DMA1		-
0x4002 4000 - 0x4002 4FFF	BKPSRAM		<a href="#">Section 3.9: Flash interface registers</a>
0x4002 3C00 - 0x4002 3FFF	Flash interface register		<a href="#">Section 7.3.24: RCC register map on page 267</a>
0x4002 3800 - 0x4002 3BFF	RCC		<a href="#">Section 4.4.4: CRC register map on page 116</a>
0x4002 3000 - 0x4002 33FF	CRC		<a href="#">Section 8.4.11: GPIO register map on page 290</a>
0x4002 2800 - 0x4002 2BFF	GPIOK		
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	Reset value
0x00	RCC_CR	0x0000 0000
0x04	RCC_PLLCFGR	0x0000 0000
0x08	RCC_CFGR	0x0000 0000
0x0C	RCC_CIR	0x0000 0000
0x10	RCC_AHB1RSTR	0x0000 0000
0x14	RCC_AHB2RSTR	0x0000 0000
0x18	RCC_AHB3RSTR	0x0000 0000
0x1C	RCC_APB1RSTR	0x0000 0000
0x20	RCC_APB2RSTR	0x0000 0000
0x24	RCC_SPIRST	0x0000 0000
0x28	RCC_SDIORST	0x0000 0000
0x2C	RCC_PWDGIRST	0x0000 0000
0x30	RCC_AHB1ENR	0x0000 0000

# Firmware Development

# The Hard Way

# RCC AHB1 Peripheral Clock Register

RCC AHB1 peripheral clock register (RCC_AHB1ENR)																	
Address offset: 0x30																	
Reset value: 0x0010 0000																	
Access: no wait state, word, half-word and byte access.																	
Reserved	OTGH S ULPIE N	OTGH SEN	ETHM ACPTP EN	ETHM ACRXE N	ETHM ACTXE N	ETHMA CEN	Res.	DMA2D EN	DMA2E N	DMA1E N	CCMDAT ARAMEN	Res.	BKPSR AMEN	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			CRCE N	Res.	GPIOK EN	GPIOJ EN	GPIOE N	GPIOH EN	GPIOG EN	GPIOF E N	GPIOEEN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN		
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	

# Firmware Development

---

The Hard Way

## GPIOC Peripheral Offset

ARM I	
0x4002 2800 - 0x4002 2BFF	GPIOK
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
0x4002 0C00 - 0x4002 0FFF	GPIOD
0x4002 0800 - 0x4002 0BFF	<b>GPIOC</b>
0x4002 0400 - 0x4002 07FF	GPIOB
0x4002 0000 - 0x4002 03FF	GPIOA

*Section 8.4.11: GPIO register map on page 290*

*Section 8.4.11: GPIO register map on page 290*

# 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 **PUPDR<sub>y</sub>[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															
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;
    }
}
```

Programmer

1 Lsh 2 = 4

	HEX	4	DEC	4	OCT	4	BIN	0100	WORD	QWORD	MS
A											
B											
C											
D											
E											
F											

Bitwise Bit shift

A	<<	>>	CE	⊗
B	(	)	%	÷
C	7	8	9	×
D	4	5	6	-
E	1	2	3	+
F	+/-	0	.	=

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

---

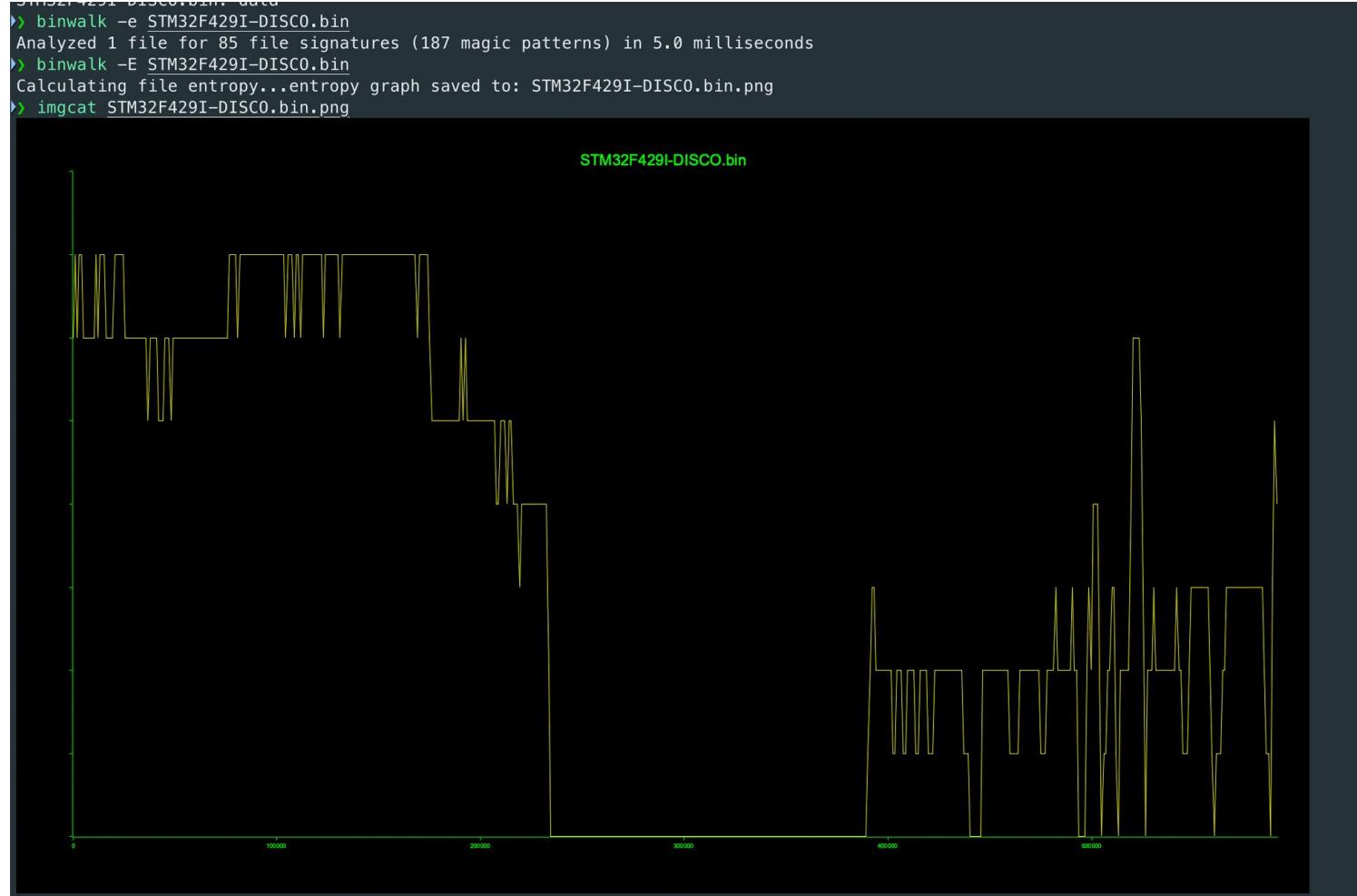
```
>> file STM32F429I-DISCO.bin
STM32F429I-DISCO.bin: data
```

file

# Basic Reverse Engineering

---

binwalk



# Basic Reverse Engineering

---

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

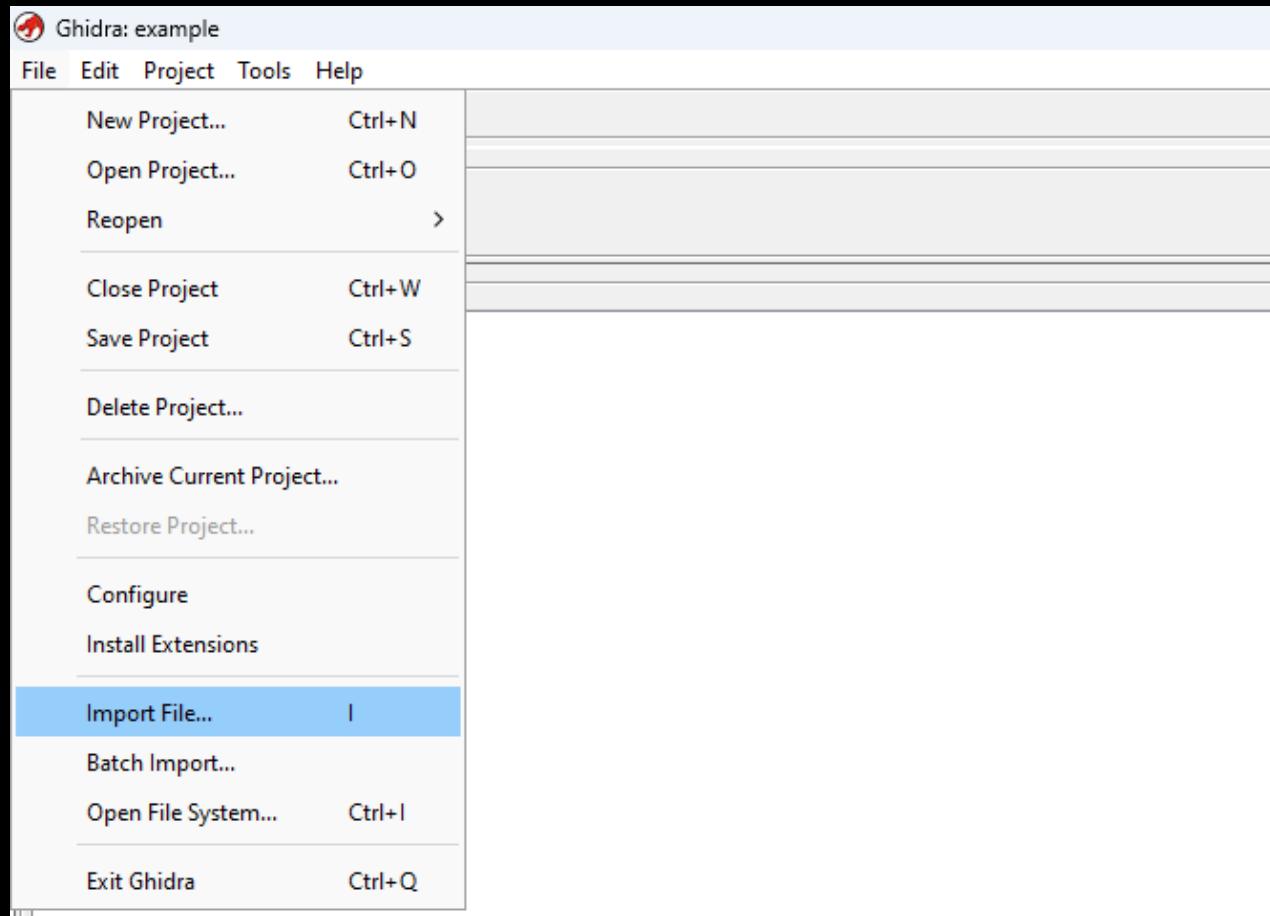
aeskeyfind

# Advanced Reverse Engineering

---

## Ghidra Basics

**File -> Import File**

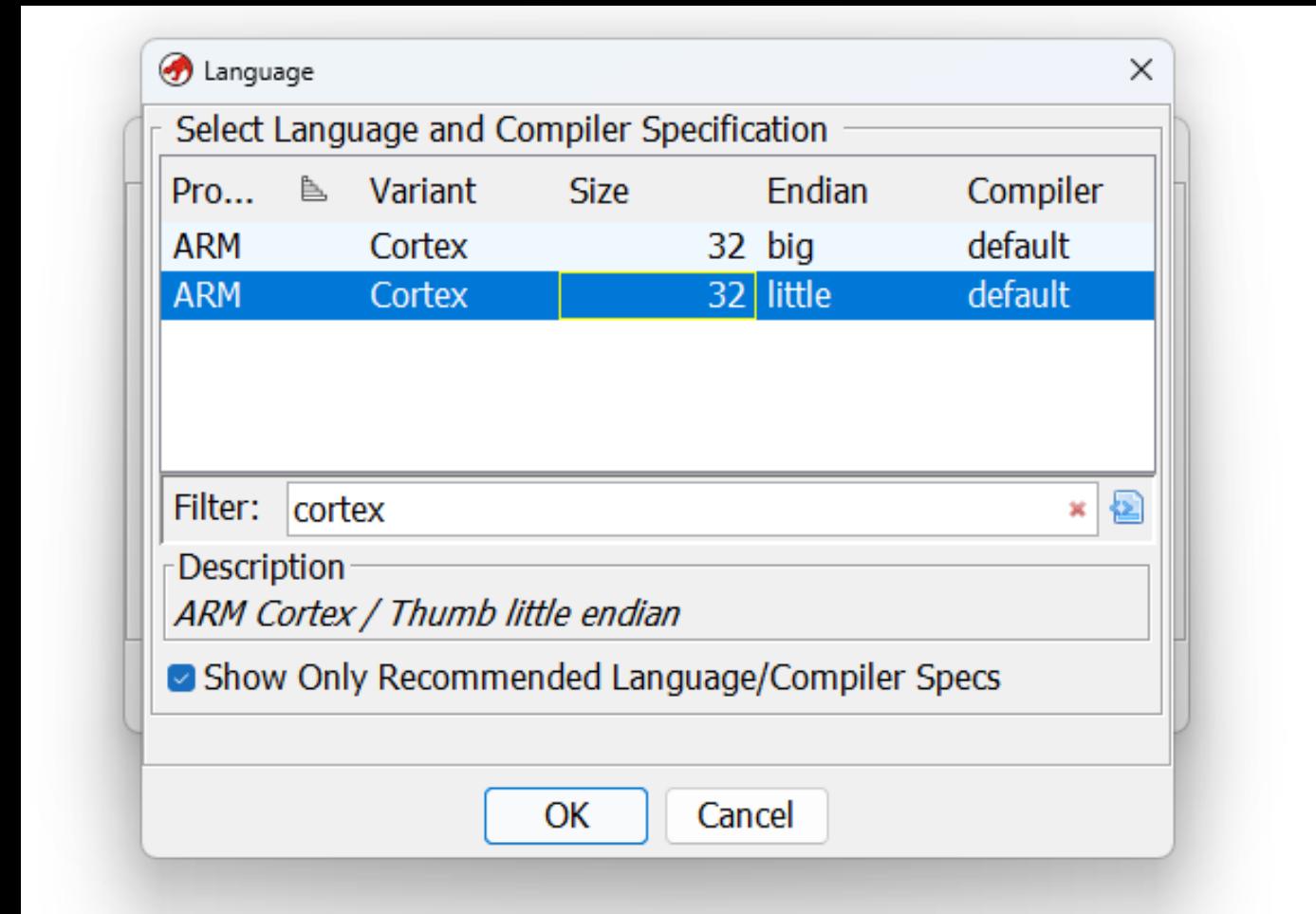


# Advanced Reverse Engineering

## Ghidra Basics

### Select Language and Compiler Specification

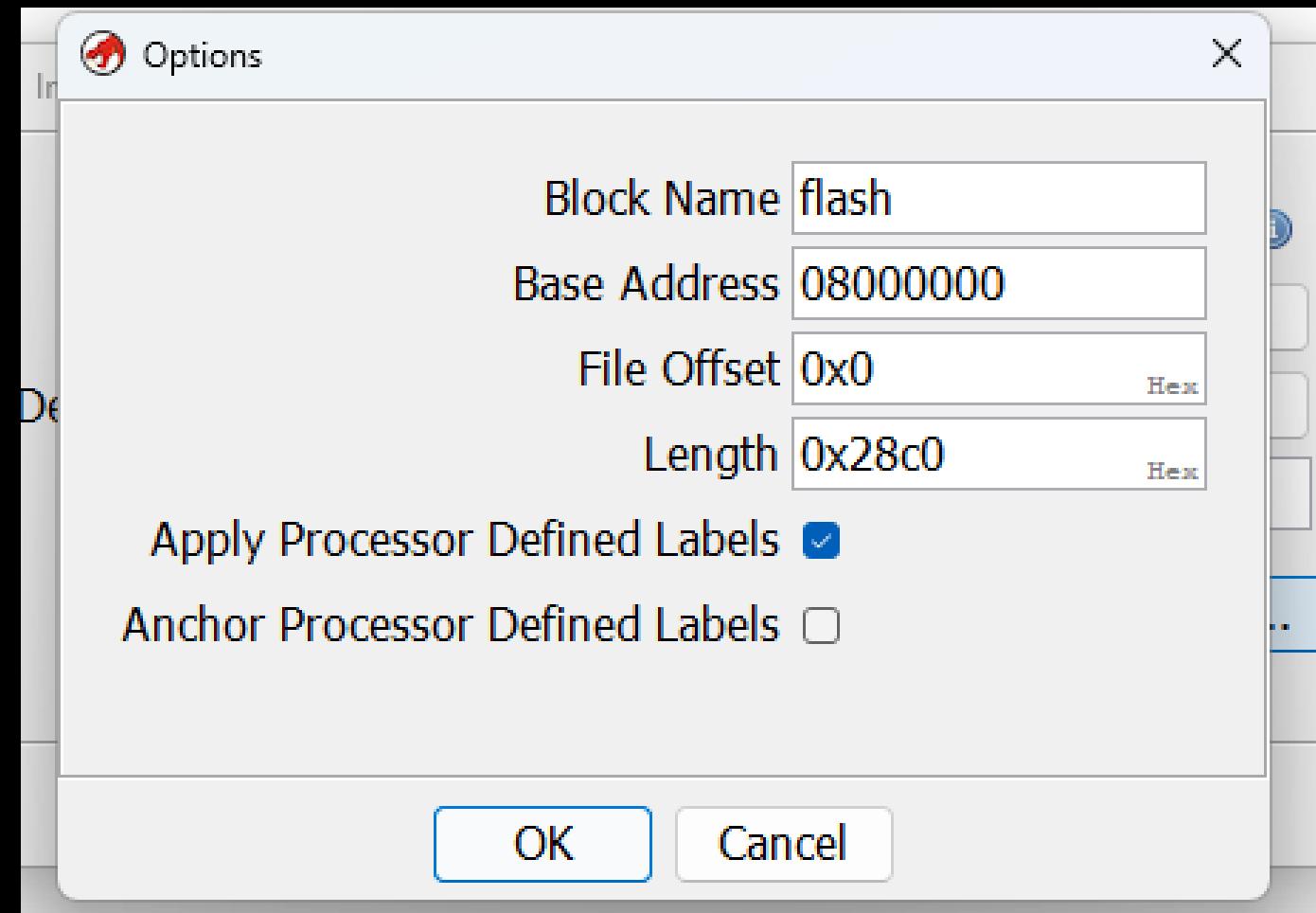
This information can typically be found in the datasheet and reference manual.



# Advanced Reverse Engineering

## Ghidra Basics

### Add Mirrored Flash Memory Block

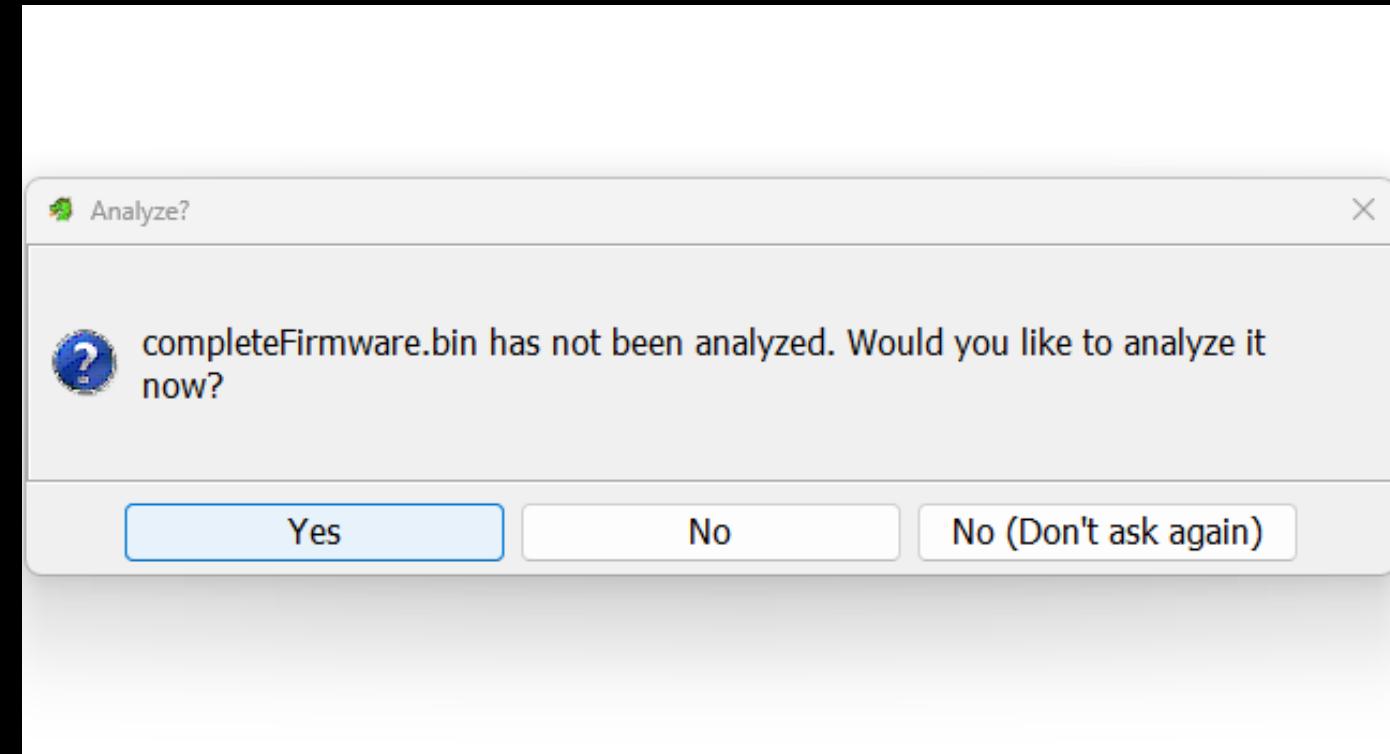


# Advanced Reverse Engineering

---

Ghidra Basics

**Do Not Analyze the  
Firmware  
Immediately**



# Advanced Reverse Engineering

---

## Ghidra Basics

### Basic Memory Map Configuration

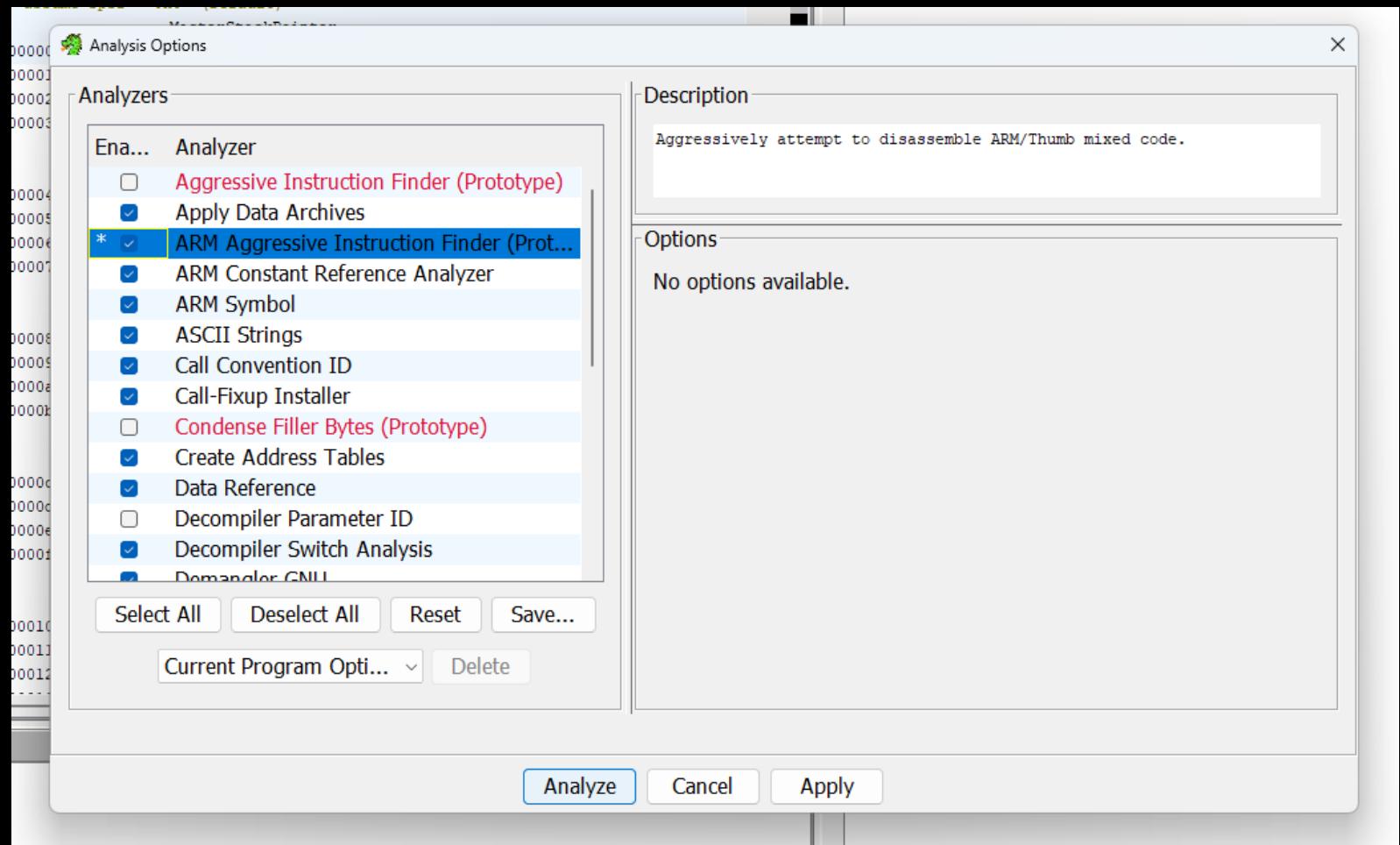
Name	Start	End	Length	R	...	X	Volat...	Artifi...	O...	Type	...	Byte Source	Source	Comment
flash_mirror	00000000	000028bf	0x28c0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Default	<input checked="" type="checkbox"/>	completeFirmware.bin[0x0, ...]		
flash	08000000	080028bf	0x28c0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Default	<input checked="" type="checkbox"/>	completeFirmware.bin[0x0, ...]	Binary Loader	
ram	20000000	2001bfff	0x1...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x1c000]		

# Advanced Reverse Engineering

## Ghidra Basics

### Analyse the Binary

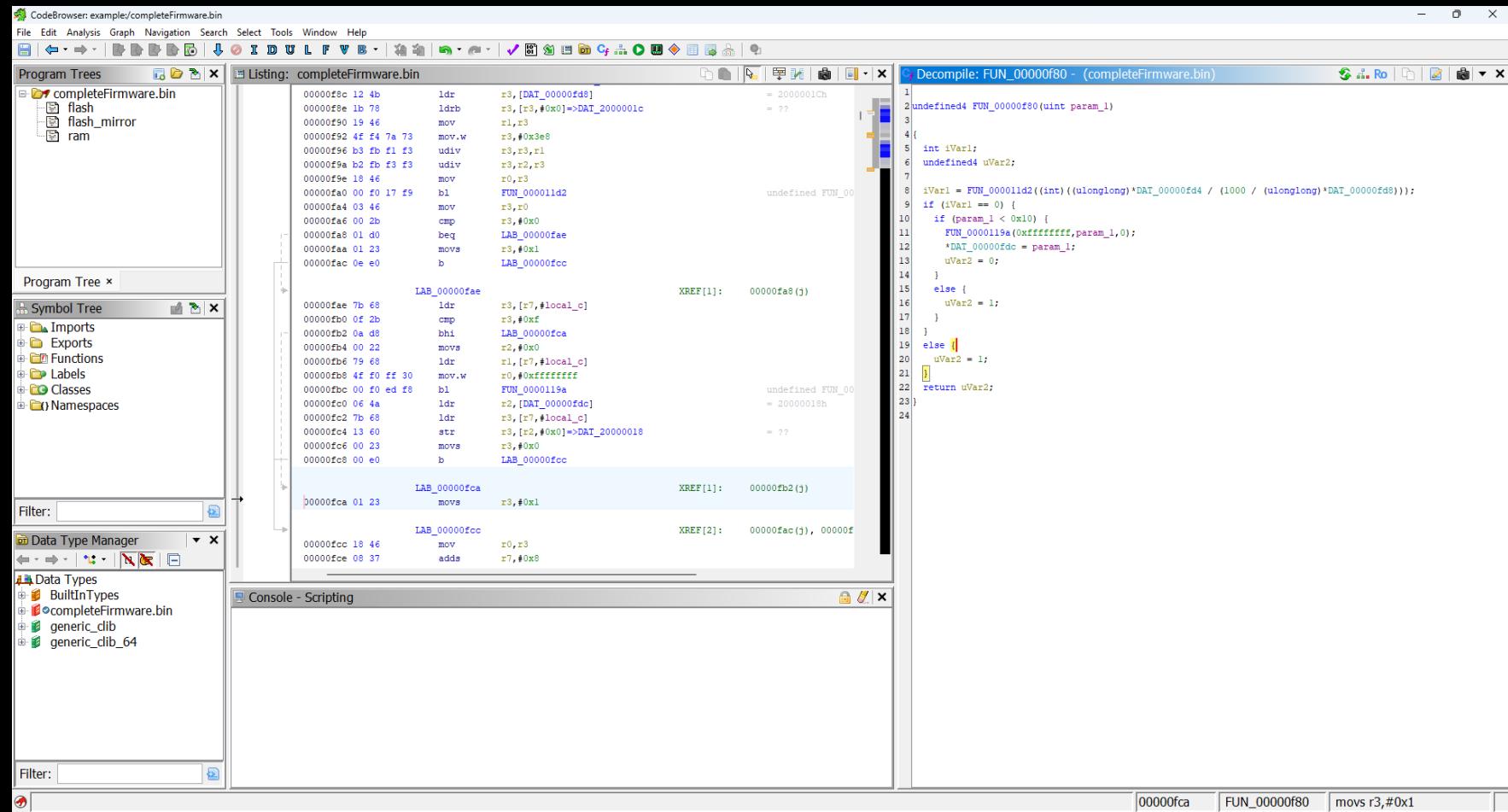
Ensure ARM Aggressive Instruction Finder is enabled



# Advanced Reverse Engineering

## Ghidra Basics

## Ghidra View Basics



# Advanced Reverse Engineering

## Ghidra Basics

### Renaming Functions (L)

The screenshot shows the Ghidra decompiler interface with the title bar "Decompile: FUN\_00000f80 - (completeFirmware.bin)". The main window displays assembly code:

```
1
2 undefined4 FUN_00000f80(uint param_1)
3
4 {
5     int iVar1;
6     undefined4 uVar2;
7
8     iVar1 = FUN_000011d2
9     if (iVar1 == 0) {
10        if (param_1 < 0x10
11            FUN_0000119a(0xf
12            *DAT_00000fd8 =
13            uVar2 = 0;
14        }
15        else {
16            uVar2 = 1;
17        }
18    }
19    else {
20        uVar2 = 1;
21    }
22    return uVar2;
23 }
```

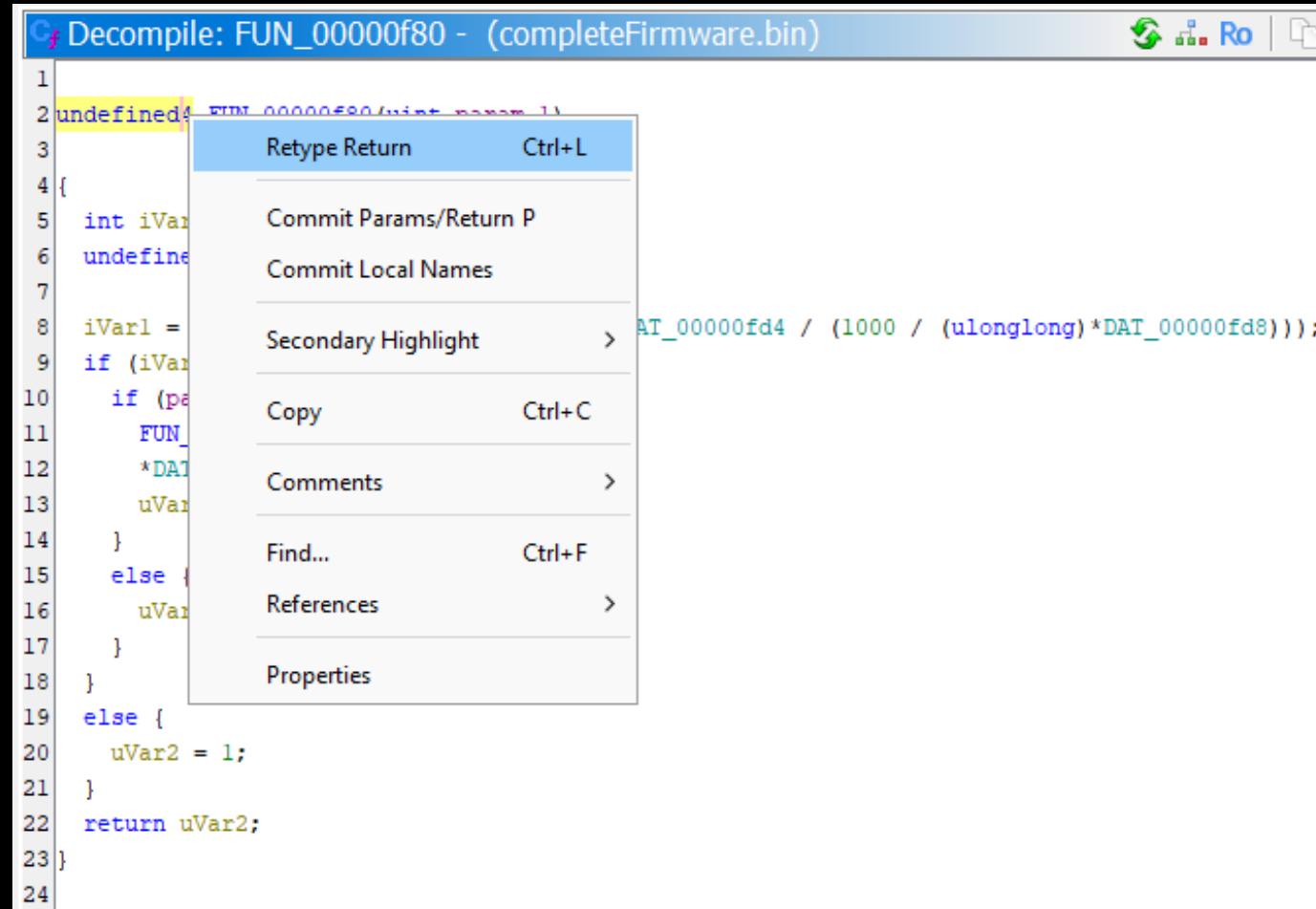
A context menu is open over the line "undefined4 FUN\_00000f80(uint param\_1)", with the "Rename Function" option highlighted.

- Edit Function Signature
- Rename Function
- Commit Params/Return P
- Commit Local Names
- Secondary Highlight
- Copy Ctrl+C
- Comments
- Find... Ctrl+F
- References
- Properties

# Advanced Reverse Engineering

# Ghidra Basics

# Retyping Variables (**CTRL+L**)



# Advanced Reverse Engineering

## Ghidra Basics

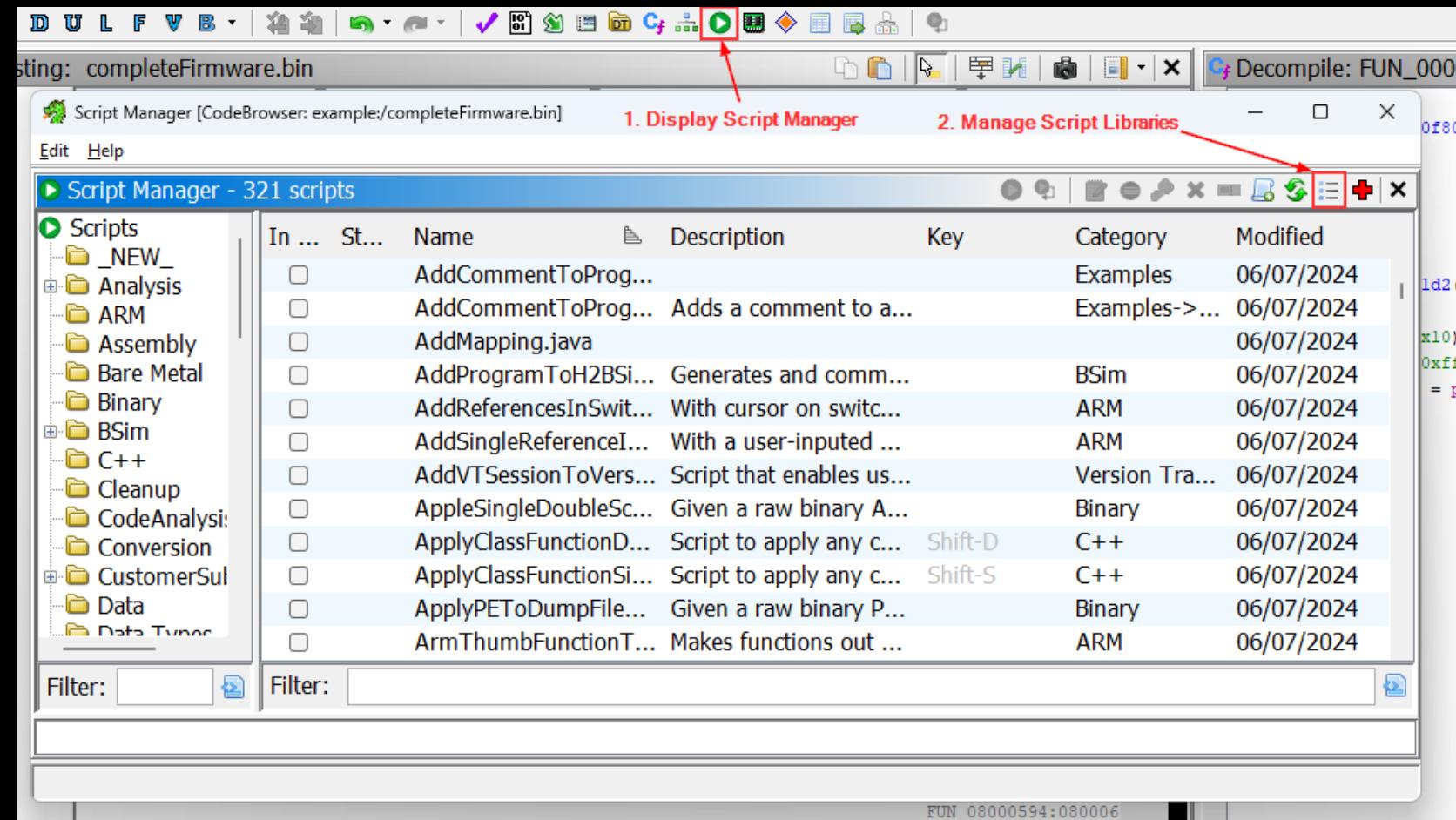
### Searching for Strings

example:/completeFirmware.bin]							
Items - [completeFirmware.bin, Minimum size = 5, Align = 1]							
Location	Label	Code Unit	String View	String Type	Length	Is Word	
08002844	s_SuperSecurePassword12...	ds "SuperSecurePassword123456789!@#...	"SuperSecurePassword123456789!@#...	string	39	true	A
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 "SuperSecurePassword123456789!@#...	"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

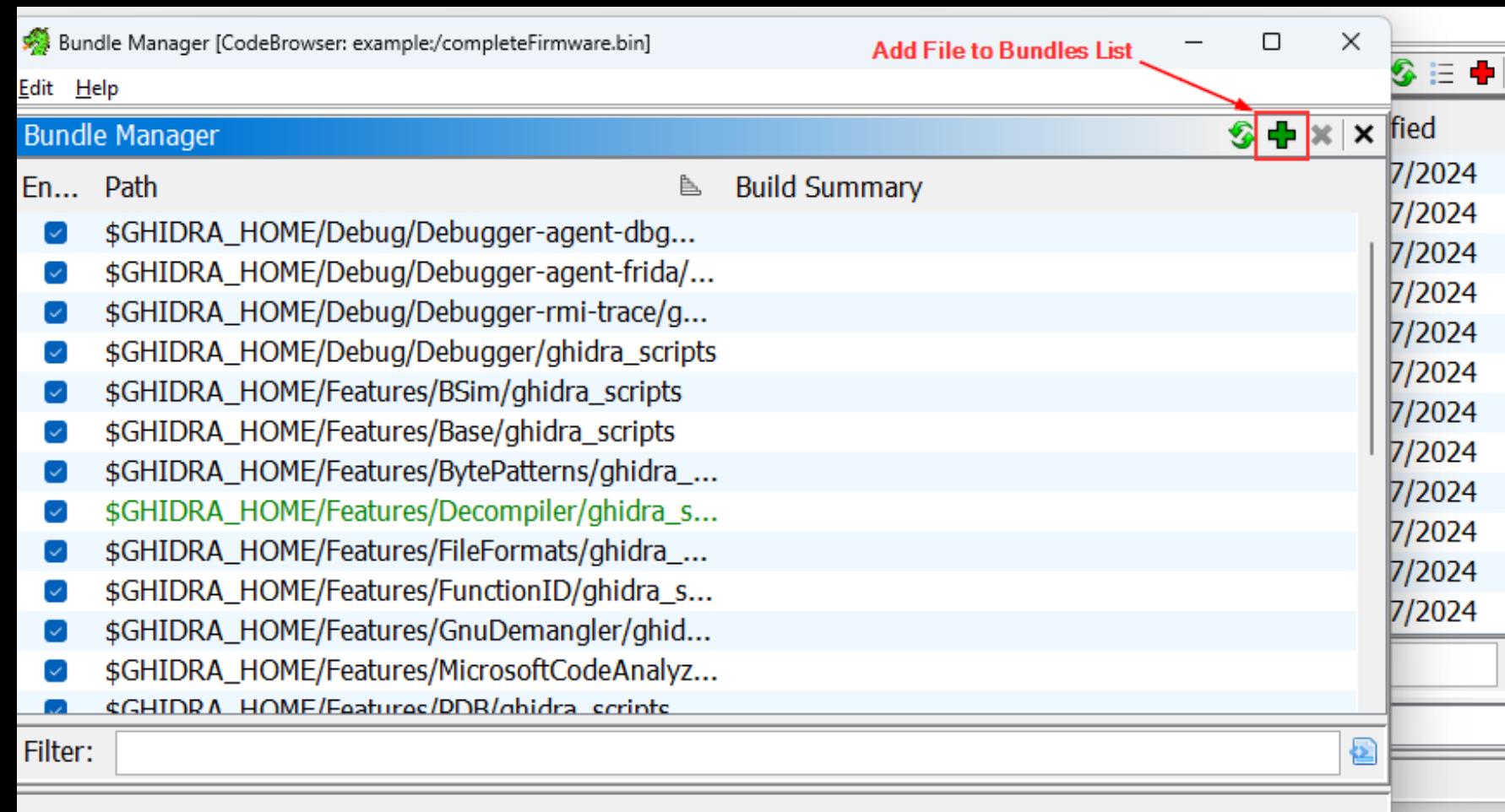


# Advanced Reverse Engineering

## 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.

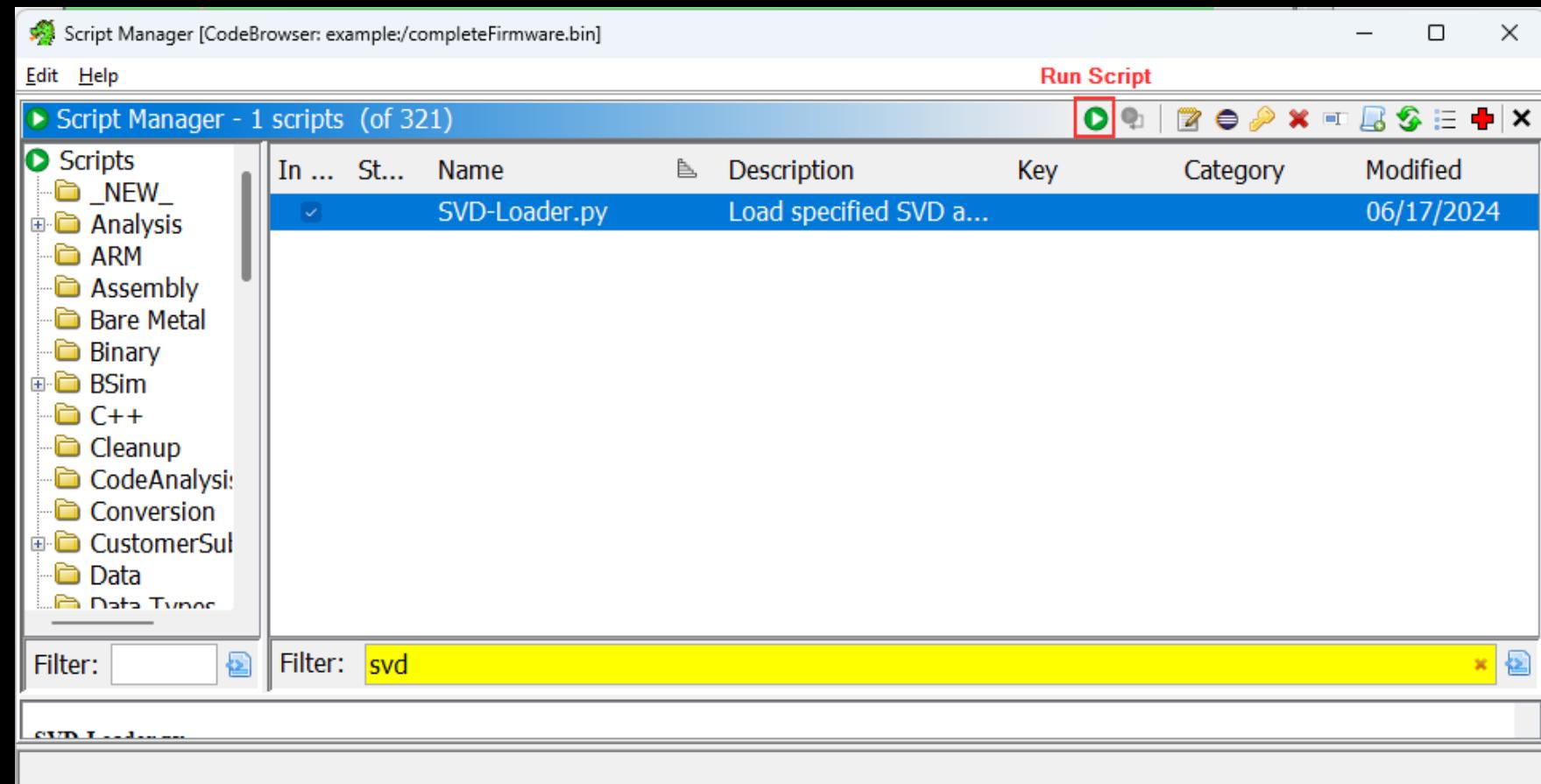
```
7  <description>SIM52R429</description>
8  <!-- details about the cpu embedded in the device -->
9  <cpu>
10   <name>CM4</name>
11   <revision>r1p0</revision>
12   <endian>little</endian>
13   <mpuPresent>false</mpuPresent>
14   <fpuPresent>false</fpuPresent>
15   <nvicPrioBits>3</nvicPrioBits>
16   <vendorSystickConfig>false</vendorSystickConfig>
17 </cpu>
18 <!--Bus Interface Properties-->
19 <!--Cortex-M4 is byte addressable-->
20 <addressUnitBits>8</addressUnitBits>
21 <!--the maximum data bit width accessible within a single transfer-->
22 <width>32</width>
23 <!--Register Default Properties-->
24 <size>0x20</size>
25 <resetValue>0x0</resetValue>
26 <resetMask>0xFFFFFFFF</resetMask>
27 <peripherals>
28   <peripheral>
29     <name>RNG</name>
30     <description>Random number generator</description>
31     <groupName>RNG</groupName>
32     <baseAddress>0x50060800</baseAddress>
```

A dedicated [CMSIS-SVD Repository](#) can be used to gather target SVD files.

# Advanced Reverse Engineering

## Ghidra Plugin: SVD Loader (BSP)

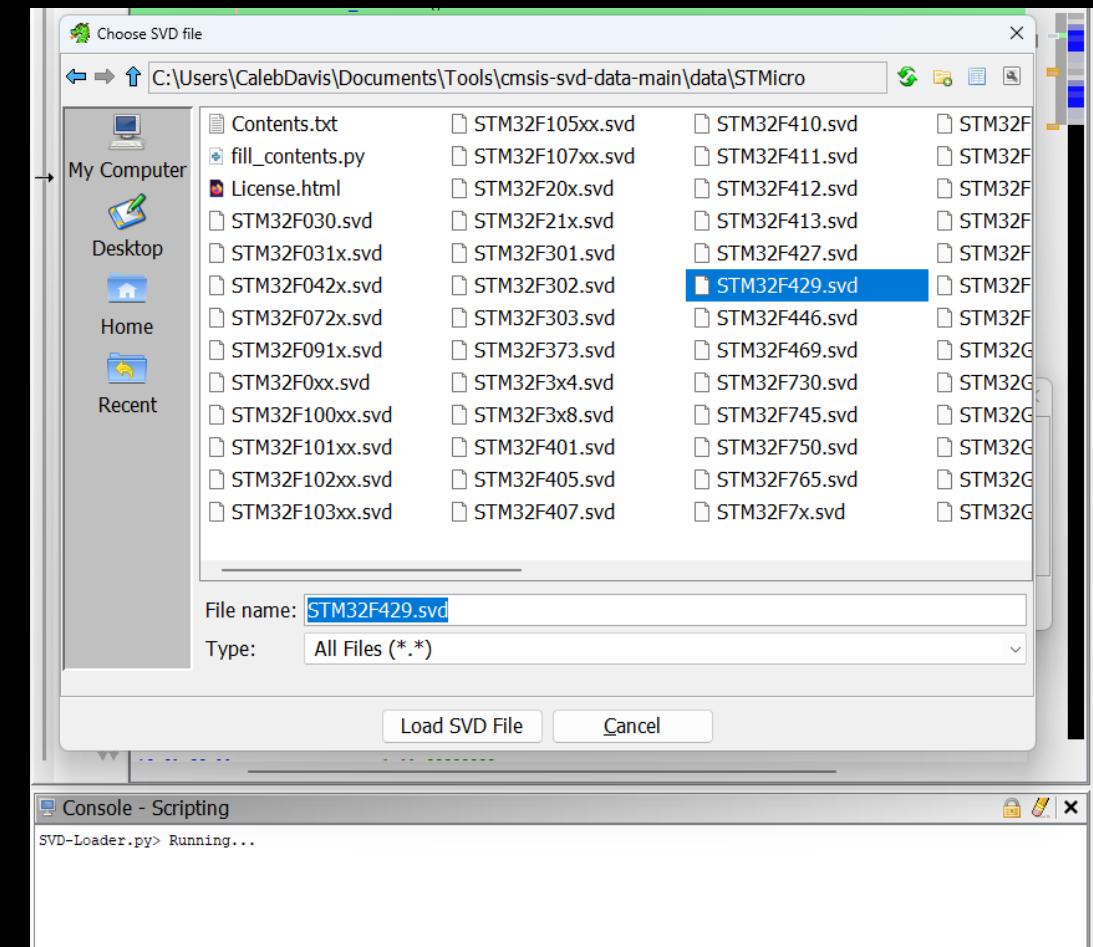
Run Script



# Advanced Reverse Engineering

Ghidra Plugin: SVD Loader (BSP)

**Choose SVD File Per Target Architecture**

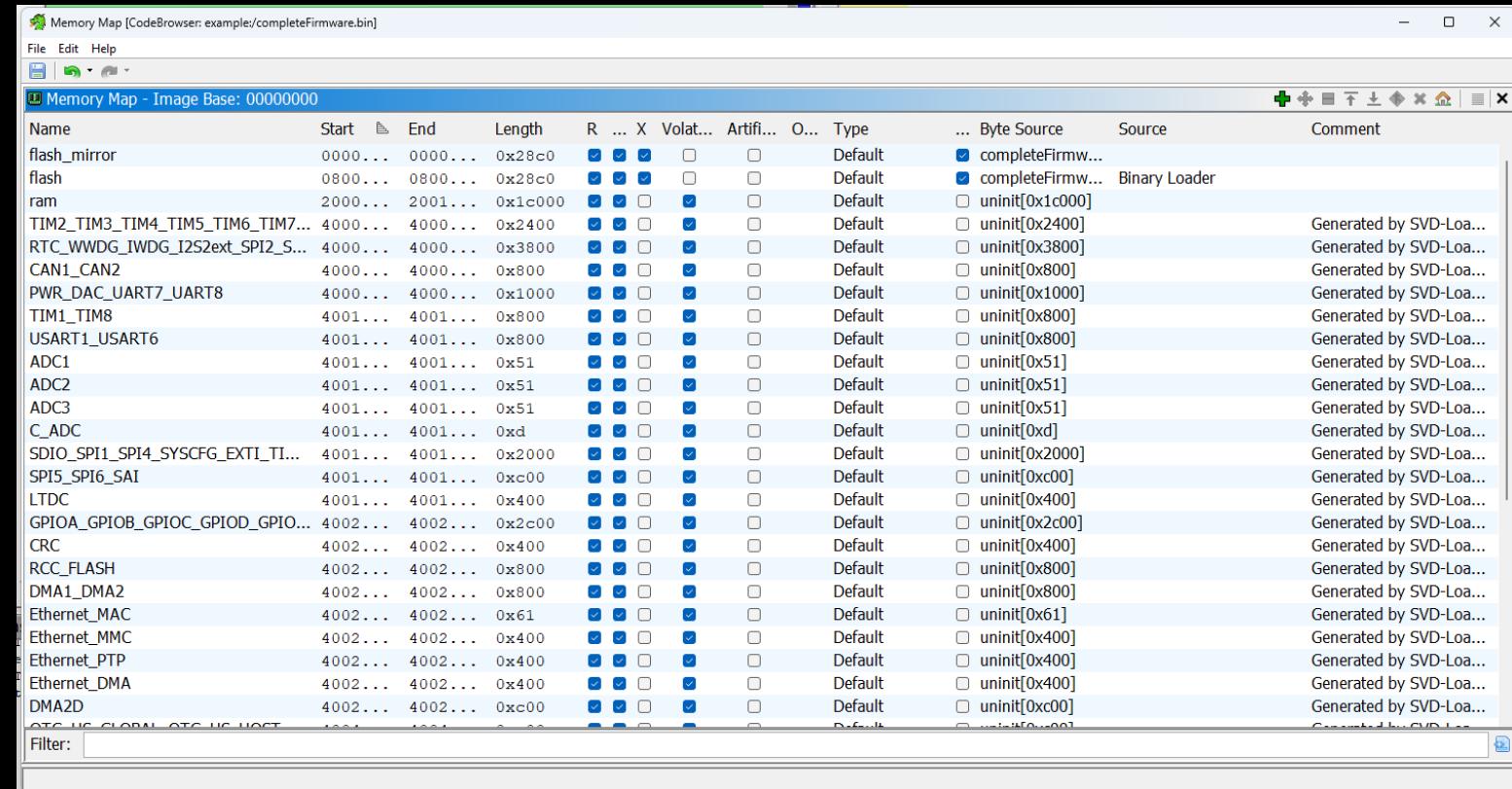


# Advanced Reverse Engineering

Ghidra Plugin: SVD Loader (BSP)

## Updated Memory Map

Added Peripheral Mappings



The screenshot shows the Ghidra Memory Map window with the title "Memory Map [CodeBrowser: example/completeFirmware.bin]". The window displays a table of memory regions with the following columns: Name, Start, End, Length, R, ... X, Volat..., Artifi..., O..., Type, ... Byte Source, Source, and Comment. The table lists numerous peripheral components such as flash\_mirror, flash, ram, TIM2, TIM3, TIM4, TIM5, TIM6, TIM7, RTC\_WWDG\_IWDG\_I2S2ext\_SPI2\_S..., CAN1\_CAN2, PWR\_DAC\_UART7\_UART8, TIM1\_TIM8, USART1\_USART6, ADC1, ADC2, ADC3, C\_ADC, SDIO\_SPI1\_SPI4\_SYSCFG EXTI\_TI..., SPI5\_SPI6\_SAI, LTDC, GPIOA\_GPIOB\_GPIOC\_GPIOD\_GPIO..., CRC, RCC\_FLASH, DMA1\_DMA2, Ethernet\_MAC, Ethernet\_MMC, Ethernet\_PTP, Ethernet\_DMA, DMA2D, and OTG\_HS\_GLOBAL\_OTG\_HS\_HOST. Many entries have their "Byte Source" field checked, indicating they were loaded from the SVD file. A filter bar at the bottom of the table allows for searching the data.

Name	Start	End	Length	R	...	X	Volat...	Artifi...	O...	Type	... Byte Source	Source	Comment
flash_mirror	0000...	0000...	0x28c0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Default	<input checked="" type="checkbox"/>	completeFirmw...	
flash	0800...	0800...	0x28c0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Default	<input checked="" type="checkbox"/>	completeFirmw...	Binary Loader
ram	2000...	2001...	0x1c000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x1c000]	
TIM2_TIM3_TIM4_TIM5_TIM6_TIM7...	4000...	4000...	0x2400	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x2400]	Generated by SVD-Lo...
RTC_WWDG_IWDG_I2S2ext_SPI2_S...	4000...	4000...	0x3800	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x3800]	Generated by SVD-Lo...
CAN1_CAN2	4000...	4000...	0x800	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x800]	Generated by SVD-Lo...
PWR_DAC_UART7_UART8	4000...	4000...	0x1000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x1000]	Generated by SVD-Lo...
TIM1_TIM8	4001...	4001...	0x800	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x800]	Generated by SVD-Lo...
USART1_USART6	4001...	4001...	0x800	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x800]	Generated by SVD-Lo...
ADC1	4001...	4001...	0x51	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x51]	Generated by SVD-Lo...
ADC2	4001...	4001...	0x51	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x51]	Generated by SVD-Lo...
ADC3	4001...	4001...	0x51	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x51]	Generated by SVD-Lo...
C_ADC	4001...	4001...	0xd	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0xd]	Generated by SVD-Lo...
SDIO_SPI1_SPI4_SYSCFG_EXTI_TI...	4001...	4001...	0x2000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x2000]	Generated by SVD-Lo...
SPI5_SPI6_SAI	4001...	4001...	0xc00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0xc00]	Generated by SVD-Lo...
LTDC	4001...	4001...	0x400	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x400]	Generated by SVD-Lo...
GPIOA_GPIOB_GPIOC_GPIOD_GPIO...	4002...	4002...	0x2c00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x2c00]	Generated by SVD-Lo...
CRC	4002...	4002...	0x400	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x400]	Generated by SVD-Lo...
RCC_FLASH	4002...	4002...	0x800	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x800]	Generated by SVD-Lo...
DMA1_DMA2	4002...	4002...	0x800	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x800]	Generated by SVD-Lo...
Ethernet_MAC	4002...	4002...	0x61	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x61]	Generated by SVD-Lo...
Ethernet_MMC	4002...	4002...	0x400	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x400]	Generated by SVD-Lo...
Ethernet_PTP	4002...	4002...	0x400	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x400]	Generated by SVD-Lo...
Ethernet_DMA	4002...	4002...	0x400	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x400]	Generated by SVD-Lo...
DMA2D	4002...	4002...	0xc00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0xc00]	Generated by SVD-Lo...
OTG_HS_GLOBAL_OTG_HS_HOST	4002...	4002...	0x200	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Default	<input type="checkbox"/>	uninit[0x200]	Generated by SVD-Lo...

# Advanced Reverse Engineering

Ghidra Plugin: SVD Loader (BSP)

## Type Casting

New data types can be used to cast structures

The screenshot shows the Ghidra decompiler interface. The assembly code window displays:

```
Cf Decompile: FUN_000004ec - (completeFirmware.bin)
1
2 uint FUN_000004ec(void)
3
4 {
5     uint uVar1;
6
7     PTR_RCC_00000520->AHB1ENR = PTR_RCC_00000520->AHB1ENR | 4;
8     *DAT_00000524 = *DAT_00000524 & 0xffffffffcf;
9     uVar1 = DAT_00000524[3];
10    DAT_00000524[3] = uVar1 | 0x10;
11    return uVar1 | 0x10;
12 }
```

Below the assembly window, a "Data Type Chooser Dialog" is open. The input field contains "RCC \*".

OK Cancel

# Advanced Reverse Engineering

Ghidra Plugin: SVD Loader (BSP)

## Code Comparison

```
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;
}
```

```
C:\f\Decompile: initPC2 - (completeFirmware.bin)
1
2 uint initPC2(void)
3
4 {
5     uint pc2_pupdr_value;
6
7     PTR_RCC_00000520->AHB1ENR = PTR_RCC_00000520->AHB1ENR | 0b000000100;
8     PTR_GPIOC_00000524->MODER = PTR_GPIOC_00000524->MODER & 0xffffffffcf;
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

```
⌚ > cd /mnt/c/Users/CalebDavis/Documents/Tools/typeLoader > on ⚖ p 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

⌚ > cd /mnt/c/Users/CalebDavis/Documents/Tools/typeLoader > on ⚖ p 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)

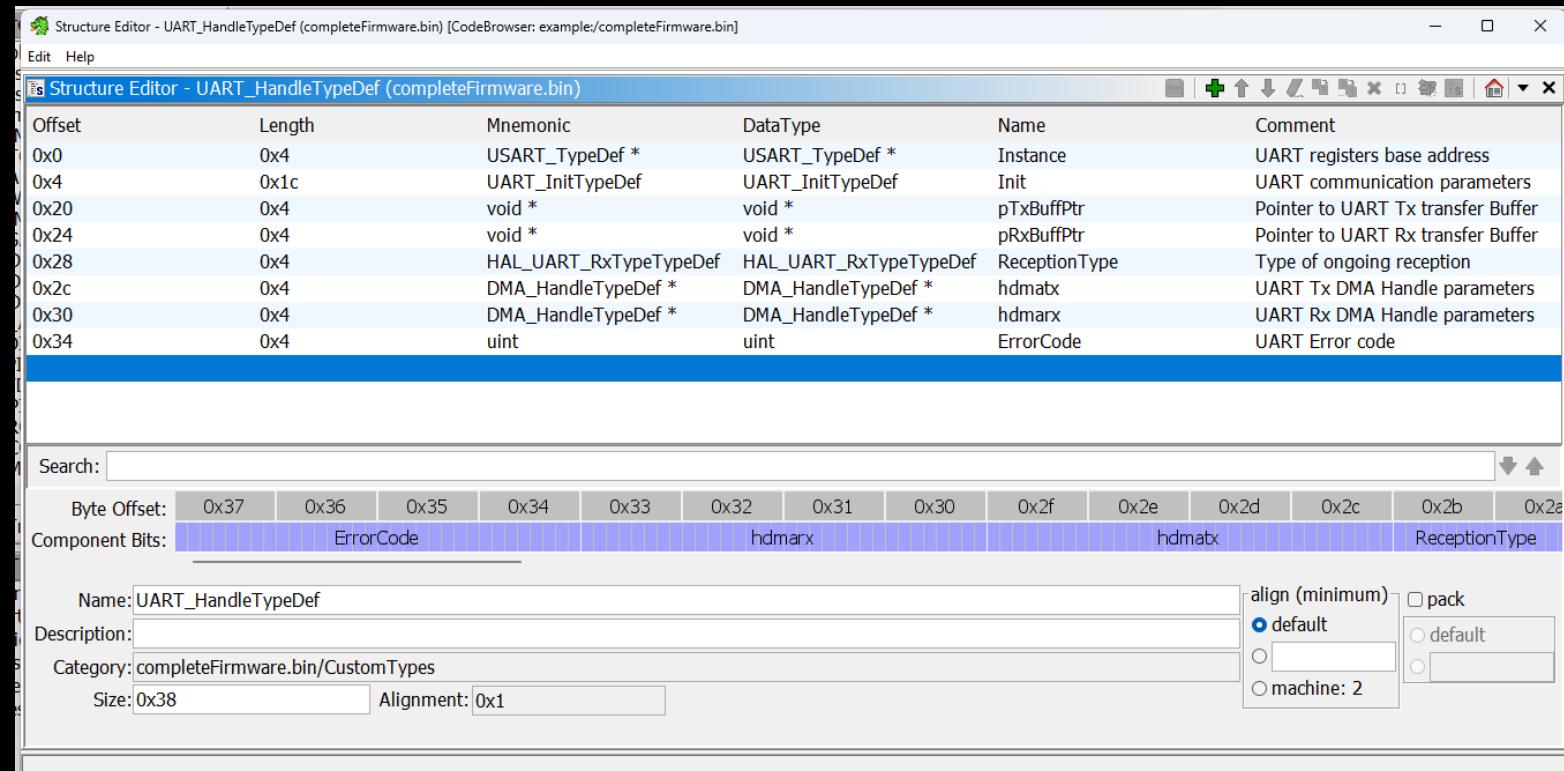
### TypeLoader Output

```
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
        USART_InitTypeDef
        USART_HandleTypeDef
```

# Advanced Reverse Engineering

Ghidra Plugin: Type Loader (HAL)

## Custom HAL Data Types



# Advanced Reverse Engineering

Ghidra Plugin: Type Loader (HAL)

## Code Comparison

```
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();
    }
}
```

```
5     int errorCheck;
6
7     uartHandle->Instance = DAT_080008c0;
8     (uartHandle->Init).BaudRate = 115200;
9     (uartHandle->Init).WordLength = 0;
10    (uartHandle->Init).StopBits = 0;
11    (uartHandle->Init).Parity = 0;
12    (uartHandle->Init).Mode = 0b00001100;
13    (uartHandle->Init).HwFlowCtl = 0;
14    (uartHandle->Init).OverSampling = 0;
15    errorCheck = uartInit(uartHandle);
16    if (errorCheck != 0) {
17        Error_Handler();
18    }
19    return;
```

# Advanced Reverse Engineering

---

Ghidra Plugin: BSim (RTOS / HAL)

## BSim by the NSA

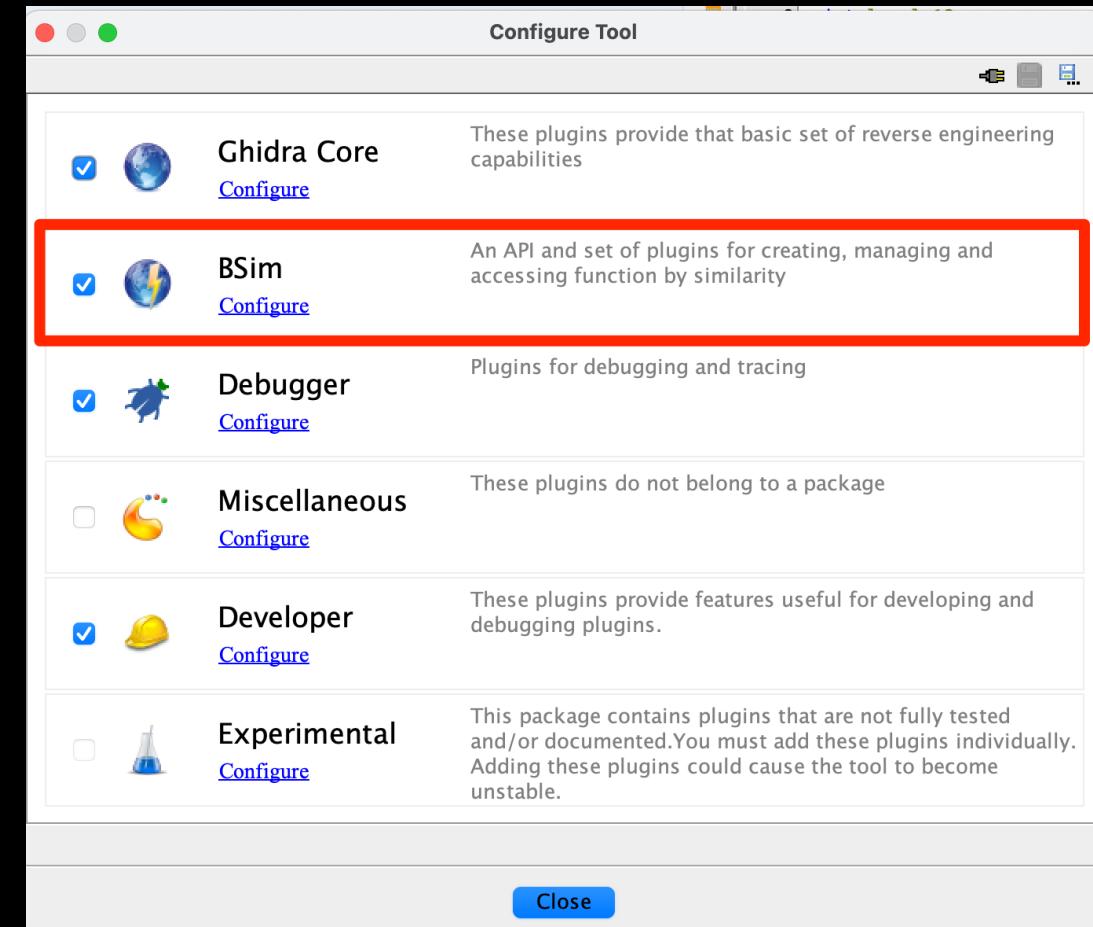
BSim is Ghidra's built-in plugin designed to accelerate reverse engineering of stripped or firmware binaries. It enables rapid identification and annotation of functions by performing fuzzy matching against known libraries, even when compiler options or exact versions are unknown.

# Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

## Enabling BSim

File -> Configure -> Enable BSim



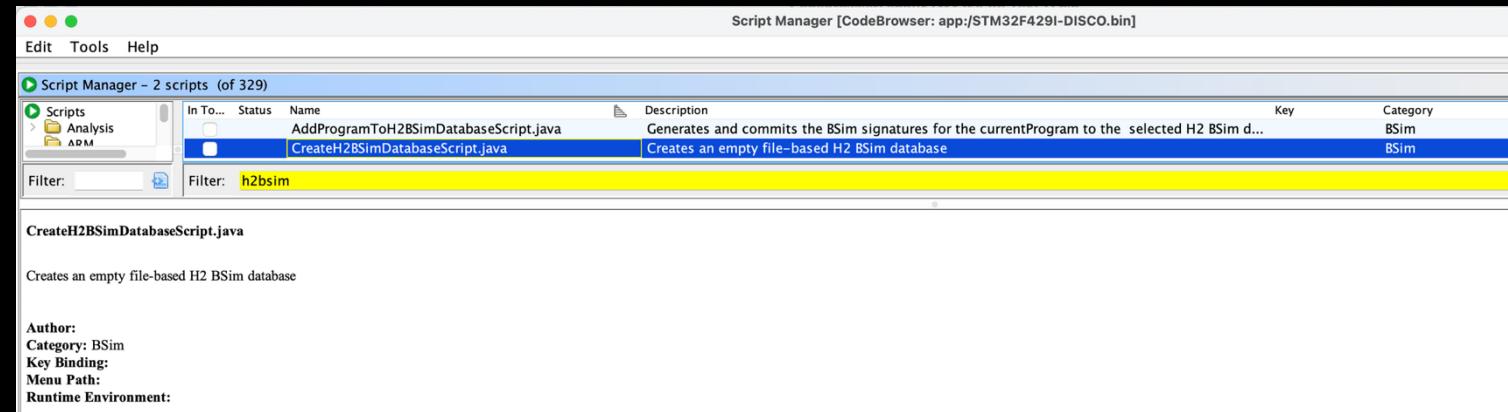
# Advanced Reverse Engineering

---

Ghidra Plugin: BSim (RTOS / HAL)

## Create an H2 BSim Database

CreateH2BSimDatabaseScript.java

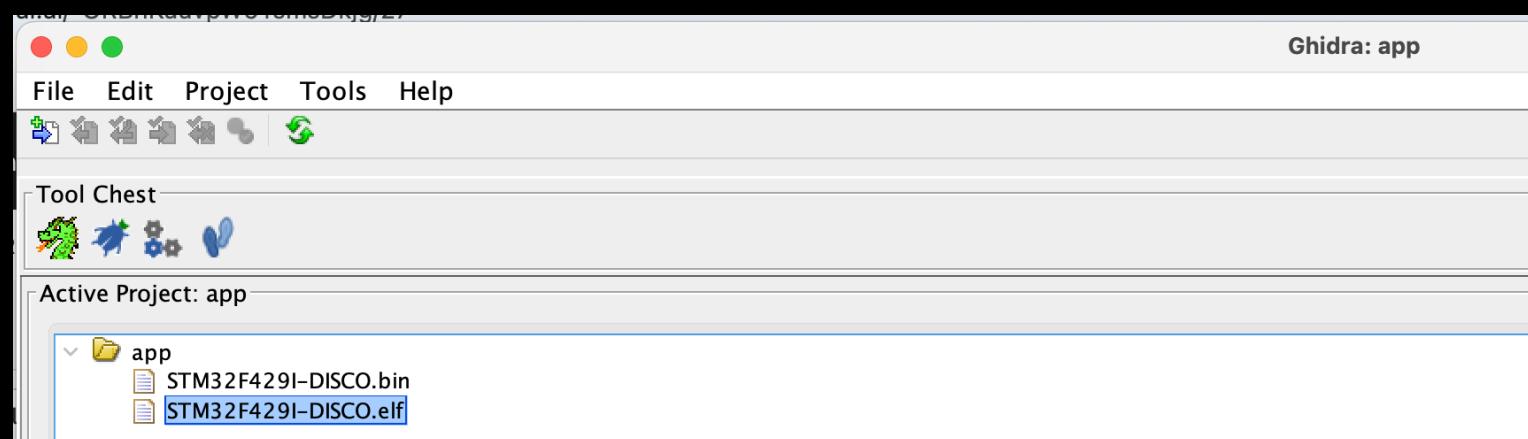


# Advanced Reverse Engineering

---

Ghidra Plugin: BSim (RTOS / HAL)

**Build and RE a  
Dedicated  
Binary (.elf)**



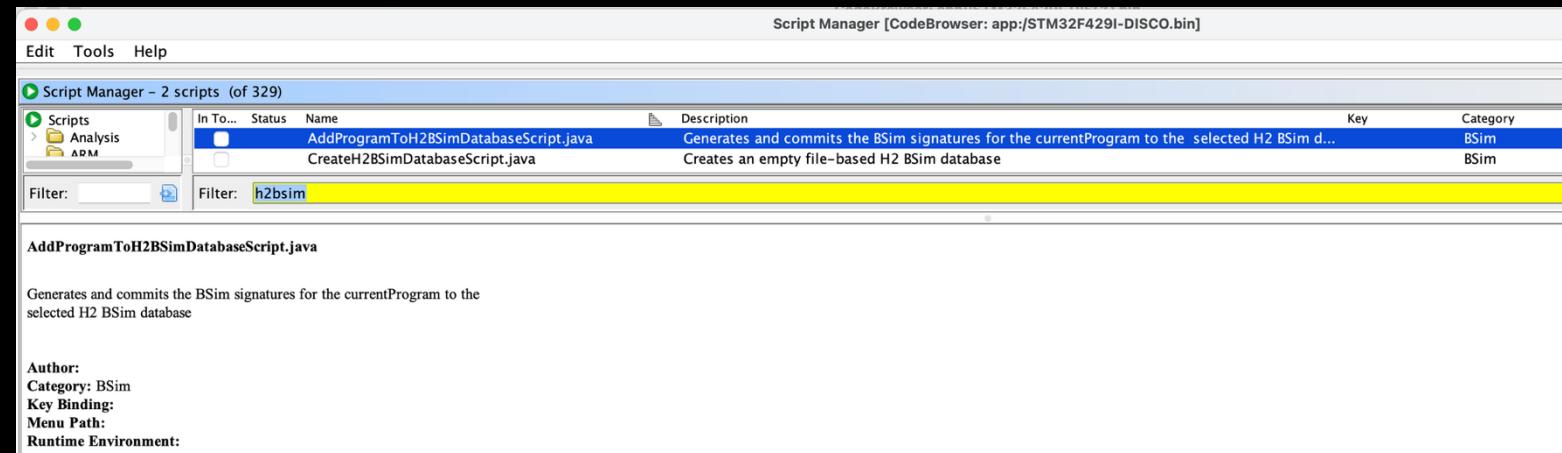
# Advanced Reverse Engineering

---

Ghidra Plugin: BSim (RTOS / HAL)

## Add Reversed Program to H2 BSim Database

AddProgramToH2BSimDatabaseScript.java

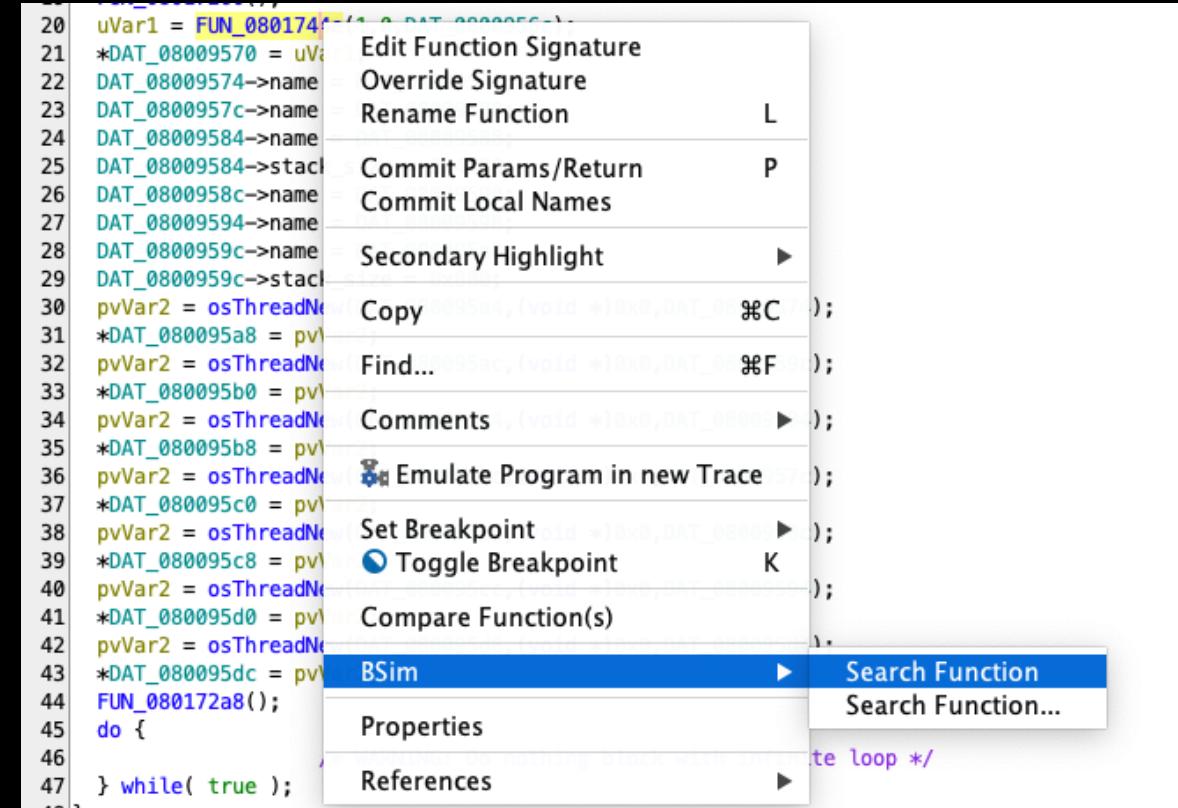


# Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

## Search Function

Right Click Function -> BSim -> Search Function

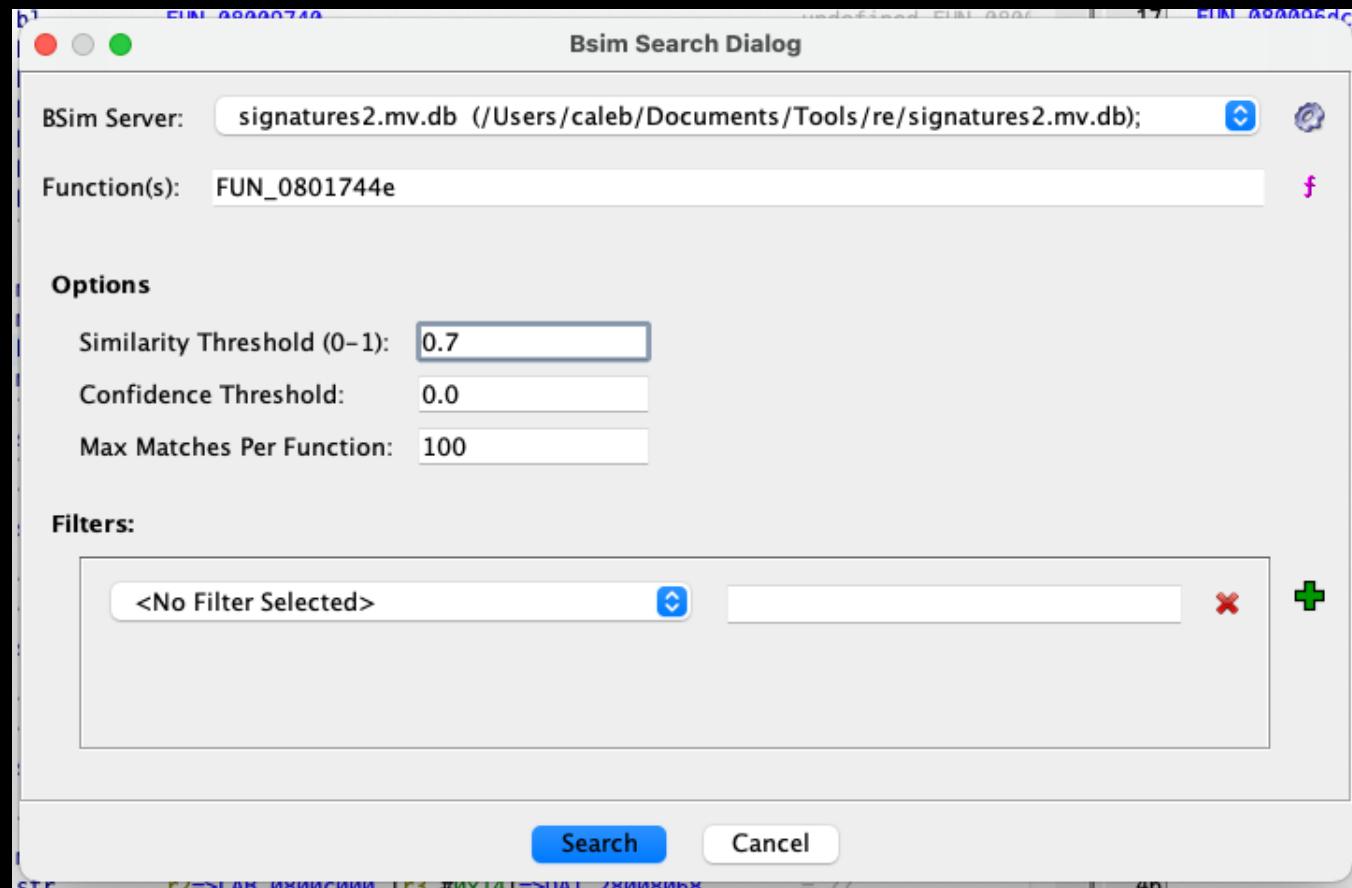


# Advanced Reverse Engineering

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

The screenshot shows the Ghidra interface with the BSim Search Results window open. The search results table lists 13 function matches for 'FUN\_0801744e' with similarity scores ranging from 0.219 to 0.882. The decompiled code for 'FUN\_0801744e' is shown in the right pane, featuring assembly-like syntax with annotations for parameters and local variables. A context menu is open over the decompiled code, with the 'Apply Signature and Data Types' option highlighted.

Status	Similarity	Conf...	Function Name	Matching Function Name	Exe Name	Ingest Date	Location
0.882	106.851	FUN_0801744e	osSemaphoreNew	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.438	33.381	FUN_0801744e	osMessageQueueNew	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.374	24.421	FUN_0801744e	osThreadNew	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.380	23.030	FUN_0801744e	osMessageQueueGet	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.379	23.030	FUN_0801744e	osMessageQueuePut	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.371	14.063	FUN_0801744e	osDelay	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.337	11.757	FUN_0801744e	osKernelInitialize	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.289	11.697	FUN_0801744e	osSemaphoreAcquire	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.300	9.573	FUN_0801744e	osKernelStart	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.248	6.985	FUN_0801744e	xQueueGenericSendF...	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.216	3.862	FUN_0801744e	vPortValidateInterrupt	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.204	2.349	FUN_0801744e	osSemaphoreRelease	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	
0.219	1.172	FUN_0801744e	xQueueGiveFromISR	STM32F429I-DISCO.elf	Jun 17, 2025	0801744e	

Executables - 1 results

Exe Name	Ingest Date	Function Count	Confidence
STM32F429I-DISCO.elf	Jun 17, 2025	1	106.856

Decompile: FUN\_0801744e ~ (STM32F429I-DISCO.bin)

```
1 int FUN_0801744e(uint param_1,uint param_2,undefined4 *param_3)
2 {
3     bool bVar1;
4     int iVar2;
5     uint uVar3;
6
7     iVar2 = param_1;
8     uVar3 = param_2;
9     if (iVar2 & 0x1f) {
10        iVar2 = -1;
11        if (param_3 == (undefined4 *)0x0) {
12            local_10 = 0;
13        }
14        else if ((param_3[2] == 0) || ((uint)param_3[3] < 0x50)) {
15            if ((param_3[2] == 0) && (param_3[3] == 0)) {
16                local_10 = 0;
17            }
18        }
19        else {
20            local_10 = 1;
21        }
22        if (local_10 != -1) {
23            if (param_1 == 1) {
24                if (local_10 == 1) {
25                    local_c = FUN_08017b94(1,0,0,param_3[2],3);
26                }
27            }
28            else {
29                local_c = FUN_08017c0e(1,0,3);
30            }
31            if (((local_c != 0) && (param_2 != 0)) && (iVar2 = FUN_08017e28(local_c,0,0,0), iVar2 != 0)) {
32                FUN_08018770(local_c);
33            }
34            local_c = 0;
35        }
36    }
37 }
```

# Advanced Reverse Engineering

Ghidra Plugin: BSim (RTOS / HAL)

## Code Comparison

The image shows a screenshot of the Ghidra software interface, specifically the 'Code Comparison' feature. It displays two windows side-by-side, each showing assembly code for the same function, 'osThreadNew', from different sources.

**Left Window (cmsis\_os2.c X):**

```
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->prio != osPriorityNone) {
465                 prio = (UBaseType_t)attr->prio;
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 {
```

**Right Window (Decompile: osThreadNew - (STM32F429I-DISCO.bin)):**

```
1  osThreadId_t osThreadNew(osThreadFunc_t func,void *argument,osThreadAttr_t *attr)
2
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

---

