



life.augmented

# 使用STM32L5的OTFDEC和ICACHE功能实时 高效运行片外Flash中的加密代码

MCD China Team

1 简介

5 示例运行演示

2 示例代码介绍

6 总结

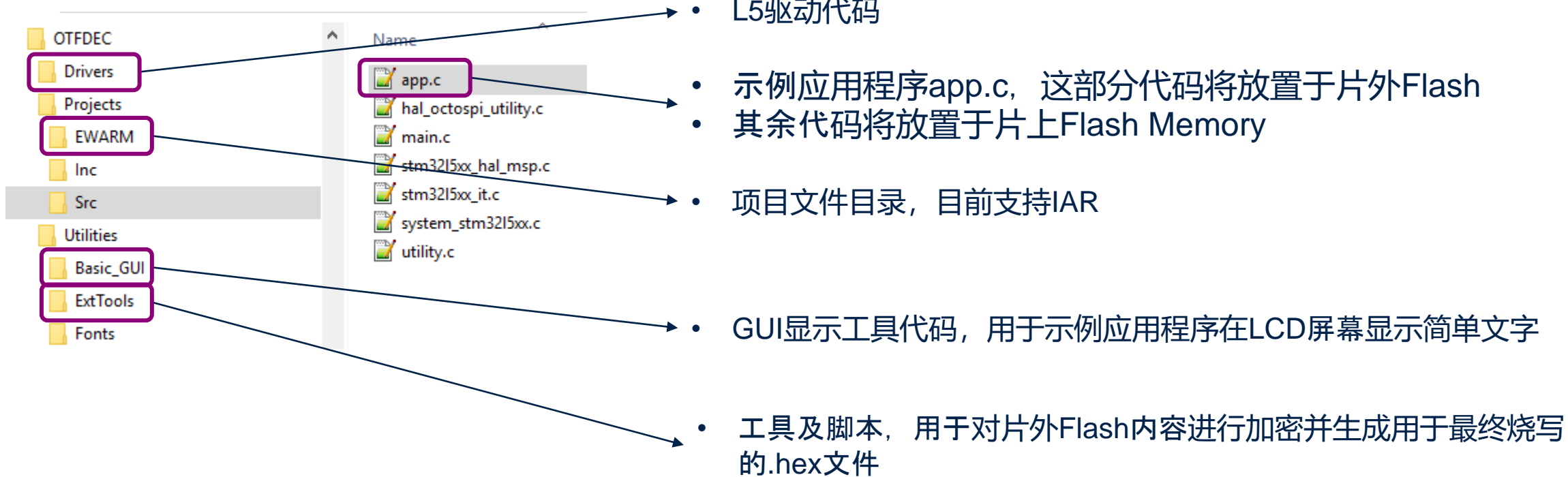
3 OTFDEC与ICACHE配置

4 使用openssl加密片外Flash代码内容

- 目标
  - 保护外置Flash中的代码和数据的机密性
  - 同时提高外部Flash中代码的运行效率
- 利用STM32L5的新特性
  - **OTFDEC** →
    - **OnTheFlyDEC**rypt硬件可以**实时**解密外置OSPI Flash中的数据和代码，从而CPU能够直接运行片外Flash上的加密代码
    - 外部Flash的代码和数据能够以加密的形式存放在外部Flash中，达到保护代码/数据的功能
  - **ICACHE** →
    - 总线矩阵前的8KB 指令与数据Cache
    - 支持将外部Memory地址Remap到Code Address，从而提高放置在外Flash中的代码运行效率
- 示例
  - 基于在STM32L562-DK板，一部分代码运行在片外OSPI Flash
  - 通过PC端软件openssl对片外Flash代码binary进行加密后写入片外Flash
  - 初始化过程将配置OTFDEC和ICACHE，实时解密运行片外Flash中的app代码

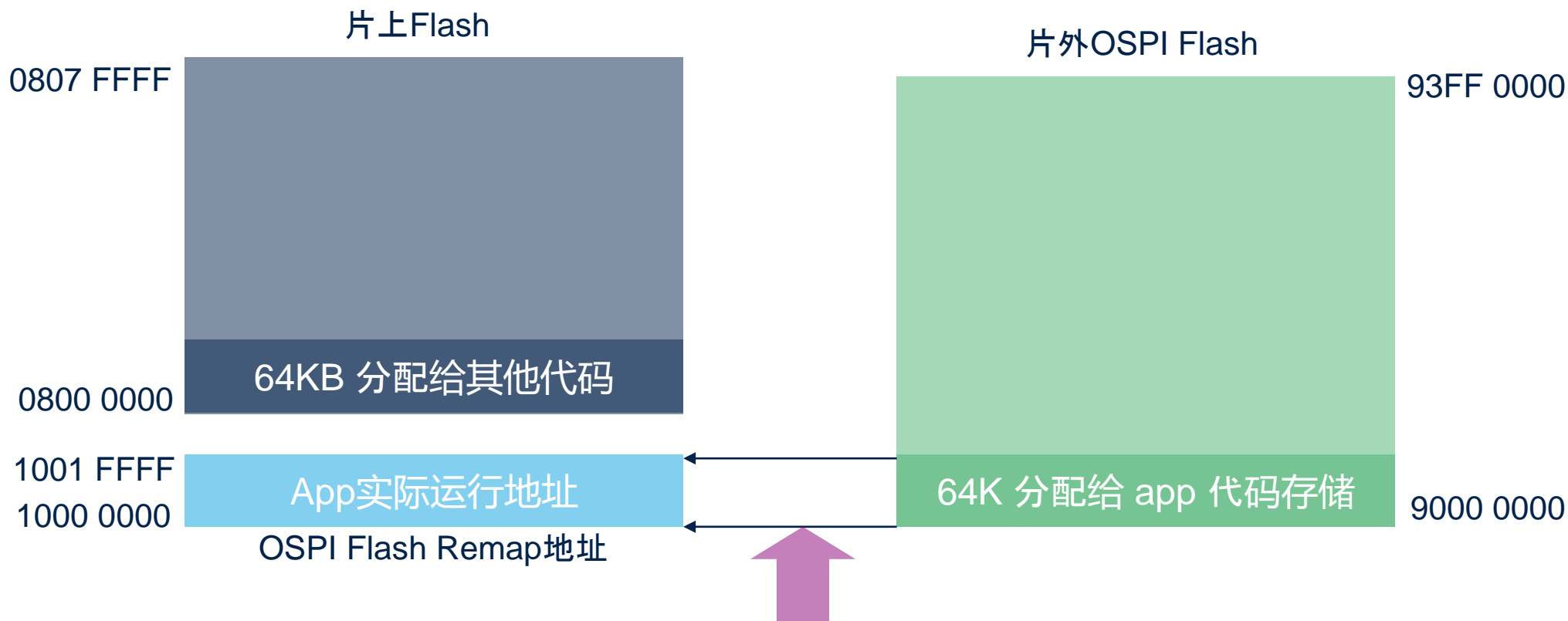
# 示例代码介绍

## • 示例代码结构



# 示例代码介绍

- 示例代码Flash存储布局



CPU对0x10000000 (Code Address) 地址访问会通过ICache remap到0x90000000地址, 从而运行于片外Flash上的代码得以通过ICACHE加速

# 示例代码介绍

## • 示例代码linker file（Flash部分）

```
define symbol __ICFEDIT_intvec_start__ = 0x08000000;  
/*-Memory Regions-*/  
define symbol __ICFEDIT_region_ROM_start__ = 0x08000000;  
define symbol __ICFEDIT_region_ROM_end__ = 0x0801FFFF;
```

```
define symbol __ICFEDIT_region_OSPI1_REMAP_start__ = 0x10000000;  
define symbol __ICFEDIT_region_OSPI1_REMAP_end__ = 0x1001FFFF;
```

这里定义OSPI remap地址，与前页说明对应

```
define memory mem with size = 4G;  
define region ROM_region = mem:[from __ICFEDIT_region_ROM_start__ to __ICFEDIT_region_ROM_end__];  
define region RAM_region = mem:[from __ICFEDIT_region_RAM_start__ to __ICFEDIT_region_RAM_end__];  
define region OSPI1_REMAP_region = mem:[from __ICFEDIT_region_OSPI1_REMAP_start__ to __ICFEDIT_region_OSPI1_REMAP_end__];
```

```
define block APP { readonly object app.o };
```

```
initialize by copy { readwrite };  
do not initialize { section .noinit };  
do not initialize { section .bss object app.o };
```

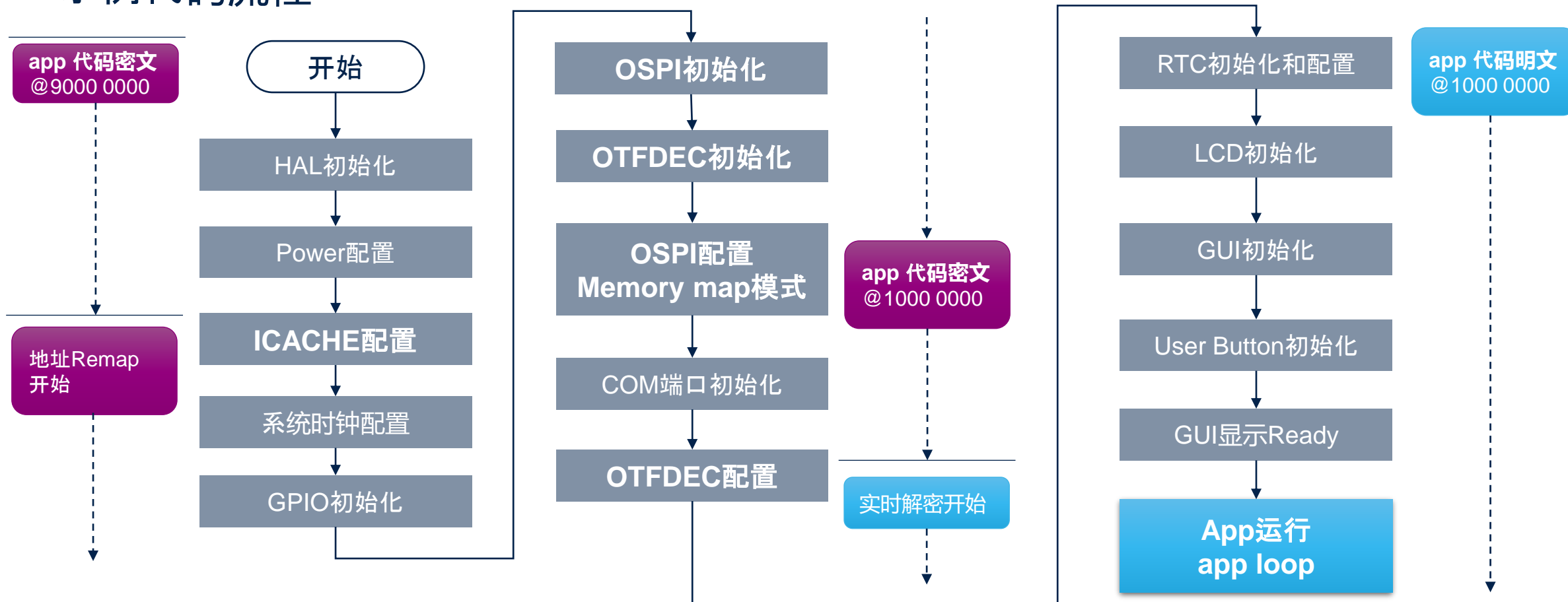
不初始化app部分的bss段，避免在10000000地址remap配置之前访问该地址段

```
place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };
```

```
place in ROM_region { readonly };  
place in OSPI1_REMAP_region { block APP };
```

app代码将被放置于OSPI remap地址区域

## • 示例代码流程



- ICache配置

- Memory remap

```
ICACHE_RegionConfigTypeDef sRegionConfig;  
sRegionConfig.BaseAddress = 0x10000000;  
sRegionConfig.RemapAddress = 0x90000000;  
sRegionConfig.Size = ICACHE_REGIONSIZE_2MB;  
sRegionConfig.OutputBurstType = ICACHE_OUTPUT_BURST_INCR;  
sRegionConfig.TrafficRoute = ICACHE_MASTER2_PORT;
```

BaseAddress 设置为1000 0000 → Code Address

RemapAddress 设置为9000 0000 → OSPI Flash Address  
当访问1000 0000地址时，ICACHE会自动向总线发送9000 0000地址访问请求，  
所以实际访问的地址是OSPI Flash的地址

```
//HAL ICACHE Disable();  
if (HAL_ICACHE_EnableRemapRegion(0, &sRegionConfig) != HAL_OK)  
{  
    while(1);  
}
```

这里开始region 0的remap。  
无论Icache本身是否使能，只要使能了remap region，  
都可以通过1000 0000地址访问9000 0000地址内容

- Icache使能

```
/* Enable Instruction cache (default 2 ways set associative cache) */  
if (HAL_ICACHE_ConfigAssociativityMode(ICACHE_2WAYS) != HAL_OK)  
{  
    while(1);  
}  
  
if (HAL_ICACHE_Enable() != HAL_OK)  
{  
    /* Initialization Error */  
    while(1);  
}
```

可以选择1-way或者2-way模式，然后使能Icache，加速代码运行



# OTFDEC配置

- 使用OTFDEC之前， OSPI需要配置为memory map模式
- 设置了ICache memory remap之后9000 0000 OSPI Flash的内容已经可以通过1000 0000地址访问，但是读取的数据都是密文，需要正确配置OTFDEC region后才能访问到解密后的明文数据
- OTFDEC region配置（1）
  - 设置region模式为指令和数据访问（我们的app代码包含指令与数据）

```
/* Enable all interruptions */
__HAL_OTFDEC_ENABLE_IT(&hotfdec1, OTFDEC_ALL_INT);

/* Set OTFDEC Mode */
if (HAL_OTFDEC_RegionSetMode(&hotfdec1, OTFDEC_REGION1, OTFDEC_REG_MODE_INSTRUCTION_OR_DATA_ACCESSES) != HAL_OK)
{
    Error_Handler();
}
```

- OTFDEC region配置 (2)
  - 配置region密钥Key → 即AES CTR解密使用的Key

```
uint32_t Key[4]={0x00010203, 0x22222222, 0x33333333, 0x44444444};

/* Set OTFDEC Key */
if (HAL_OTFDEC_RegionSetKey(&shotfdecl, OTFDEC_REGION1, Key) != HAL_OK)
{
    Error_Handler();
}
```

- 配置region其他参数 → Nonce, 地址, 版本, RegionID构成AES CTR解密使用的IV

```
OTFDEC_RegionConfigTypeDef Config = {0};
/* Configure then activate OTFDEC decryption */
Config.Nonce[1]    = 0x0A0B0C0D;
Config.Nonce[0]    = 0x0E0F0102;
Config.StartAddress = 0x90000000;
Config.EndAddress   = 0x9001FFFF;
Config.Version      = 0xA5E6;
if (HAL_OTFDEC_RegionConfig(&shotfdecl, OTFDEC_REGION1, &Config, OTFDEC_REG_CONFIGR_LOCK_ENABLE) != HAL_OK)
{
    Error_Handler();
}
```

这里的起始和结束地址是OSPI Flash的地址, 而不是remap之后的地址

Region ID也是IV的一部分

# 使用openssl加密片外Flash代码内容

- 对片外Flash代码进行加密前，首先要了解OTFDEC配置参数与加密参数的对照关系
- OTFDEC的解密使用的是AES128 CTR模式，那么这里的AES CTR用到的Key与IV和前面代码中region的几个配置参数是如何对应的呢？

- Key

Words number			
W3	W2	W1	W0
Bits number			
127:96	95:64	63:32	31:0
Description			
Key 3	Key 2	Key 1	Key 0

- IV

Words number						
W3	W2	W1		W0		
Bits number						
127:96	95:64	63:48	47:32	31:30	29:28	27:0
Description						
Nonce 1	Nonce 0	Not used	Region FW version	Not used	Region ID	External memory start @ (modulo 128-bit)

# 使用openssl加密片外Flash代码内容

- AES128 CTR 的KEY和IV 与代码中OTFDEC配置数据的对应关系

- AES128 Key

bit 127	bit 96	bit 95	bit 64	bit 63	bit 32	bit 31	bit 0
---------	--------	--------	--------	--------	--------	--------	-------

- 代码中的 KEY定义:

- uint32\_t Key[4]={0x00010203, 0x22222222, 0x33333333, 0x44444444};

bit 127	Key[3]	bit 96	bit 95	Key[2]	bit 64	bit 63	Key[1]	bit 32	bit 31	Key[0]	bit 0
4	4	4	4	4	4	4	4	4	4	4	4
3	3	3	3	3	3	3	3	3	3	3	3
2	2	2	2	2	2	2	2	2	2	2	2
0	0	0	0	1	0	2	0	3			

- IV

bit 127	bit 96	bit 95	bit 64	bit 63	bit 32	bit 31	bit 0
---------	--------	--------	--------	--------	--------	--------	-------

- 代码中有关IV的参数 :

- Nonce[1]=0x0A0B0C0D; Nonce[0]=0x0E0F0102; StartAddress=0x90000000; Version=0xA5E6;  
OTFDEC\_REGION1→Region ID: 0

bit 127	Nonce[1]	bit 96	bit 95	Nonce[0]	bit 64	bit 63	0000 Version	bit 32	bit 31	0 RID SAddr	bit 0
0	A	0	B	0	C	0	D		0	9	0
0	E	0	F	0	1	0	2		0	0	0
0	0	0	0	A	5	E	6		0	0	0

# 使用openssl加密片外Flash代码内容

- 片外Flash代码数据加密过程

- AES128 CTR块加密数据也是以16字节（128bit）为单位，分块进行块加密，字节顺序如下

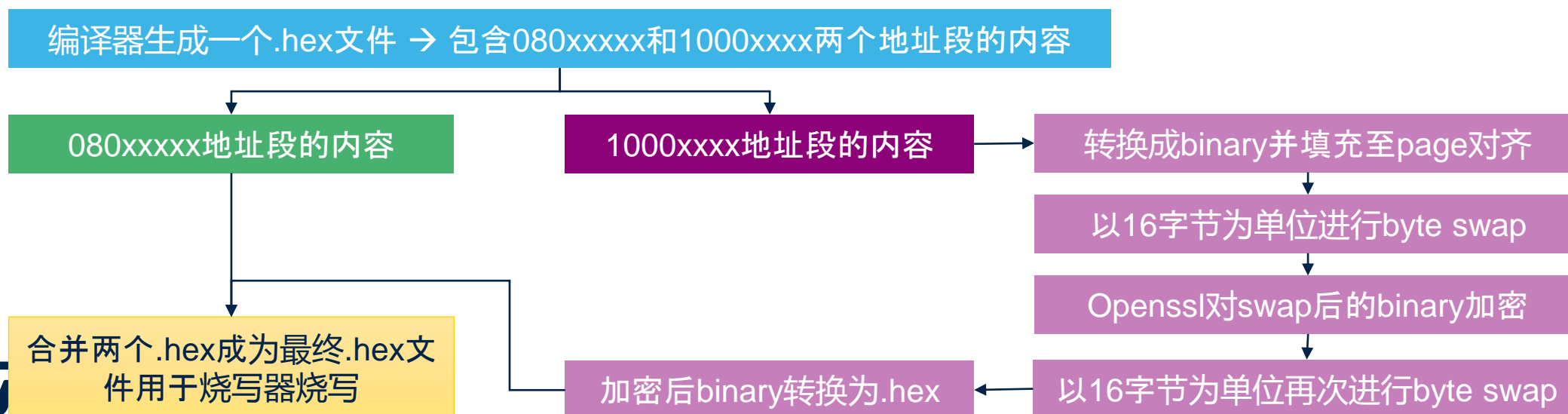


- 而编译器生成的binary文件是从低地址到高地址排列的，所以需要进行预处理和后处理

- 每16字节组的字节顺序转换



- 在本示例中我们采用如下流程进行加密（以下过程都在 **Utilities\ExtTools\PostBuild.bat** 脚本中完成）



# 使用openssl加密片外Flash代码内容

- PostBuild.bat脚本说明

- 通过xxd工具进行预处理 (16字节为单位byte swap)

- `%XXD% -e -g 16 %padextbinname% > %tempfile%`
    - `%XXD% -r %tempfile% > %padextbinname%.pre`

- 通过openssl进行加密

- AES CTR Key与代码的对应

- 代码： `uint32_t Key[4]={0x00010203, 0x22222222, 0x33333333, 0x44444444};`
      - 脚本： `SET key128="44444444333333332222222200010203"`

- AES CTR IV 与代码的对应


- 代码： `Nonce[1]=0x0A0B0C0D; Nonce[0]=0x0E0F0102; StartAddress=0x90000000; Version=0xA5E6; OTFDEC_REGION1→Region ID: 0`
      - 脚本： `SET iv="0A0B0C0D0E0F01020000A5E609000000"`
    - `%OPENSSL% enc -aes-128-ctr -nosalt -e -in %padextbinname%.pre -out %extbinname%.pad.enc.bin -K %key128% -iv %iv%`

- 通过xxd工具进行后处理(16字节为单位byte swap)

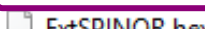
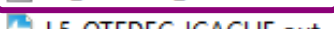
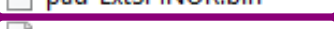
- `%XXD% -e -g 16 %extbinname%.pad.enc.bin > %tempfile%`
    - `%XXD% -r %tempfile% > %extbinname%.pad.enc.post.bin`

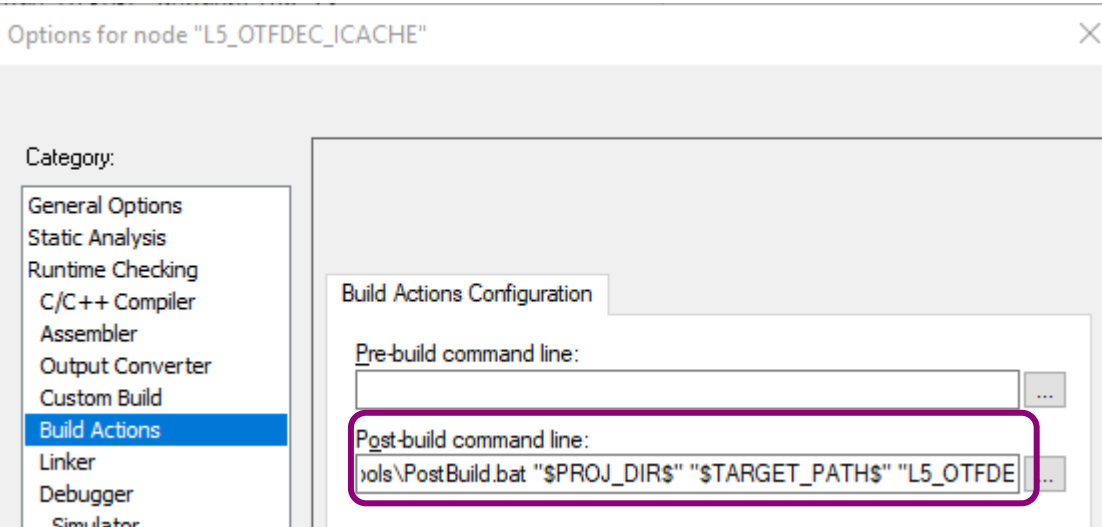
# 示例运行演示

- 编译生成待烧写文件

- 打开IAR工程文件  L5\_OTFDEC\_ICACHE.eww
- 工程除了正常的编译链接过程，还会执行一个PostBuild.bat脚本
- 该脚本将直接完成.hex拆分、外部Flash代码内容加密、合成一个.hex文件用于烧写等操作
- 最终在Exe目录下会看到以下几个文件
  - 除AllInOne.hex以外，其他文件仅供参考

Name

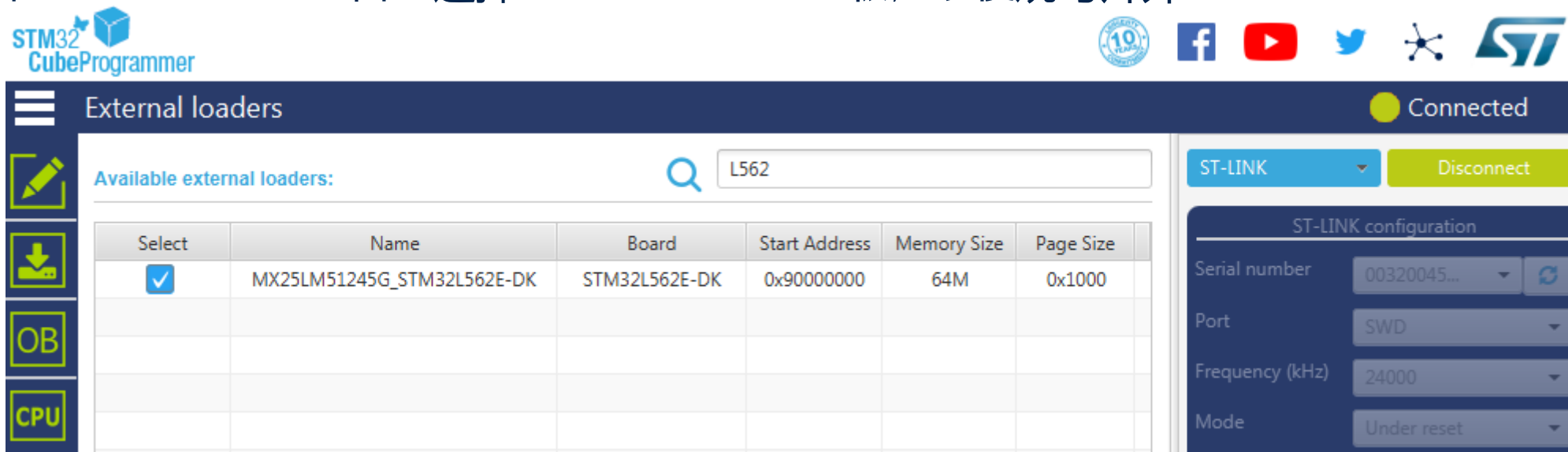
 AllInOne.hex	→ 用于烧写的最终hex文件
 ExtSPINOR.bin	→ 原始明文binary
 ExtSPINOR.bin.pad.enc.bin	→ 填充+预处理+加密后的binary
 ExtSPINOR.bin.pad.enc.post.bin	→ 加密+后处理的binary
 ExtSPINOR.hex	
 L5_OTFDEC_ICACHE.hex	→ 编译器生成的.hex文件
 L5_OTFDEC_ICACHE.out	
 log.txt	
 OnChipFlash.hex	→ 片上Flash代码.hex文件
 pad-ExtSPINOR.bin	→ 填充后的明文binary
 pad-ExtSPINOR.bin.pre	→ 填充后的明文binary经过16字节为单位的byte swap预处理的binary
 progetspi-enc.hex	
 progetspi-plain.hex	→ 片外Flash代码加密后的.hex文件



# 示例运行演示

- 烧写AllInOne.hex

- 在PC上打开STM32CubeProgrammer (v2.4.0或以上版本)，使用Under Reset模式连接L5-DK板
- 在External loaders窗口选择STM32L562E-DK板，以便烧写片外Flash





# 示例运行演示

- 烧写AllInOne.hex（续）
  - 烧写前保持以下Option bytes内容
    - TZEN = 0
    - RDP = Level 0 (0xAA)
    - NSBootAddr = 0x08000000
  - 在Erasing & Programming窗口选择AllInOne.hex文件进行烧写
    - 选择Verify programming
    - 烧写完成后会看到烧写了0x08000000和0x90000000两段地址的内容

**Download**

File path: ebug\Exe\AllInOne.hex [Browse]

Start address: [ ]

☐ Skip flash erase before programming

☒ Verify programming

☐ Run after programming

[Start Programming]

**Automatic Mode**

☐ Full chip erase

**Erase flash memory** | **Erase external memory**

[Erase selected sectors] | [Full chip erase]

Select	Index	Start Address	Size
<input type="checkbox"/>	0	0x08000000	2K
<input type="checkbox"/>	1	0x08000800	2K
<input type="checkbox"/>	2	0x08001000	2K
<input type="checkbox"/>	3	0x08001800	2K
<input type="checkbox"/>	4	0x08002000	2K
<input type="checkbox"/>	5	0x08002800	2K
<input type="checkbox"/>	6	0x08003000	2K

**Log** | Verbosity level: 1 | 2 | 3

```
14:54:45 : Memory Programming ...
14:54:45 : Opening and parsing file: AllInOne.hex
14:54:45 : File : AllInOne.hex
14:54:45 : Size : 164500 Bytes
14:54:45 : Address : 0x08000000
14:54:45 : Erasing memory corresponding to segment 0:
14:54:45 : Erasing internal memory sectors [0 16]
14:54:45 : Erasing memory corresponding to segment 1:
14:54:46 : Erasing external memory sectors [0 1]
14:54:46 : Download in Progress:
14:54:54 : File download complete
14:54:54 : Time elapsed during download operation: 00:00:09.176
14:54:54 : Verifying ...
14:54:54 : Read progress:
14:54:54 : Download verified successfully
14:55:29 : Read File: C:\MMS\MMS-
MCD\training\LS\LAT\OTFDEC\Projects\EWARM\Debug\Exe\AllInOne.hex
14:55:29 : Number of segments: 2
14:55:29 : segment[0]: address= 0x8000000, size= 0x8294
14:55:29 : segment[1]: address= 0x90000000, size= 0x20000
```

# 示例运行演示

- 查看外部Flash的内容

- 打开Memory & File edition窗口，查看**0x90000000**地址的内容，可以看到与加密后的binary的内容一致

Device memory

+

Address

0x90000000

Size

0x400

Data width

8-bit

Read

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
0x90000000	91	44	A4	C9	76	B9	C4	30	59	A1	68	29	BA	9C	B5	9B	.D#Év'Ä0Yjh)°.µ.
0x90000010	BB	FD	C5	6C	8B	6E	33	BD	38	D5	FF	6F	D1	03	C3	DF	»ýÄl.n3½8ÖÿoÑ.Äß
0x90000020	5F	EF	AA	06	C7	EF	B4	27	7B	93	07	4F	36	D3	AE	8B	_ïª.Çï´'{..06Ó®.
0x90000030	E4	C9	00	A8	08	9F	E1	49	E3	AC	8A	AD	95	4D	60	32	äÉ.``.áIä~...M`2
0x90000040	14	6B	DD	7C	20	04	BF	B4	D2	6E	18	90	D4	B2	81	52	.kÝ  .¿´òn..Ô².R
0x90000050	6D	84	8F	F5	53	F6	5A	2D	CF	A5	F7	0C	05	6A	2D	5D	m...öSöZ-İ¥÷...j-]

```
$ xxd -g 1 -l 128 ExtSPINOR.bin.pad.enc.post.bin
00000000: 91 44 a4 c9 76 b9 c4 30 59 a1 68 29 ba 9c b5 9b .D..v..0Y.h)....
00000010: bb fd c5 6c 8b 6e 33 bd 38 d5 ff 6f d1 03 c3 df ...l.n3.8..o....
00000020: 5f ef aa 06 c7 ef b4 27 7b 93 07 4f 36 d3 ae 8b _.....' {...06...
00000030: e4 c9 00 a8 08 9f e1 49 e3 ac 8a ad 95 4d 60 32 .....I.....M`2
00000040: 14 6b dd 7c 20 04 bf b4 d2 6e 18 90 d4 b2 81 52 .k.| ....n.....R
00000050: 6d 84 8f f5 53 f6 5a 2d cf a5 f7 0c 05 6a 2d 5d m...S.Z-.....j-]
```

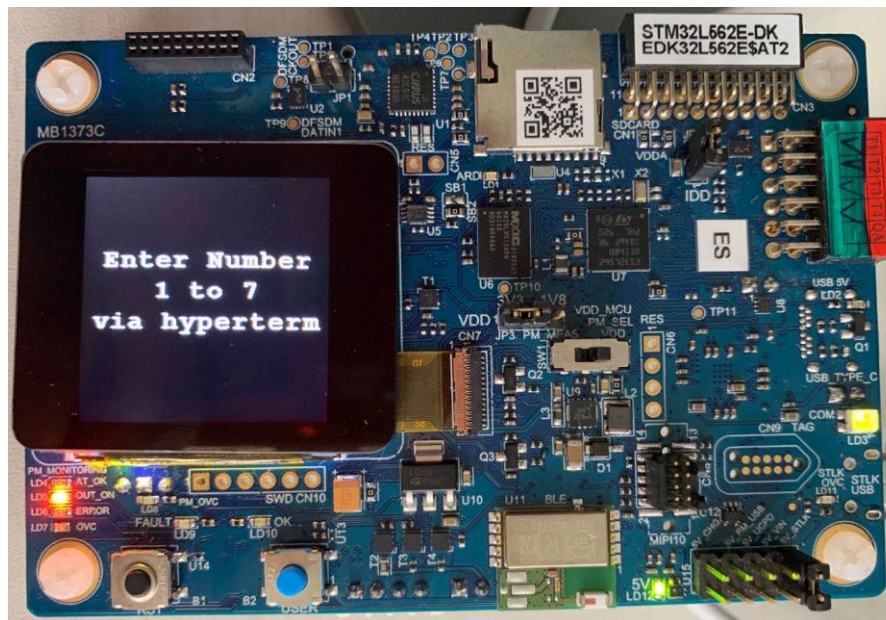
# 示例运行演示

- 运行示例程序

- 打开PC端的串口接收软件，例如Teraterm
- 复位L5-DK板之后可以看到屏幕显示和串口打印，说明程序已经在运行
- 此时可以通过串口输入数字，尝试打开/关闭 ICache等功能

```
COM Init done.
Power on from reset
BSP_LCD_Init 0
BSP_PB_Init 0
GUI_DisplayString
Initialization done, enter application...

=====
MAIN MENU
=====
Disable ICACHE -----1
Enable ICACHE -----2
Get ICACHE state -----3
Enter Standby -----4
Force SMPS bypass mode -----5
Disable SMPS bypass mode -----6
Get SMPS bypass state mode -----7
=====
Please enter your choice:
█
```



# 示例运行演示

- 观察解密后的片外Flash内容
  - 在PC上打开STM32CubeProgrammer (v2.4.0或以上版本)，使用Hot plug模式连接L5-DK板
  - 打开Memory & File edition窗口，查看0x10000000地址的内容，可以看到与原始明文binary的内容一致

Device memory

+

Address

0x10000000

Size

0x400

Data width

8-bit

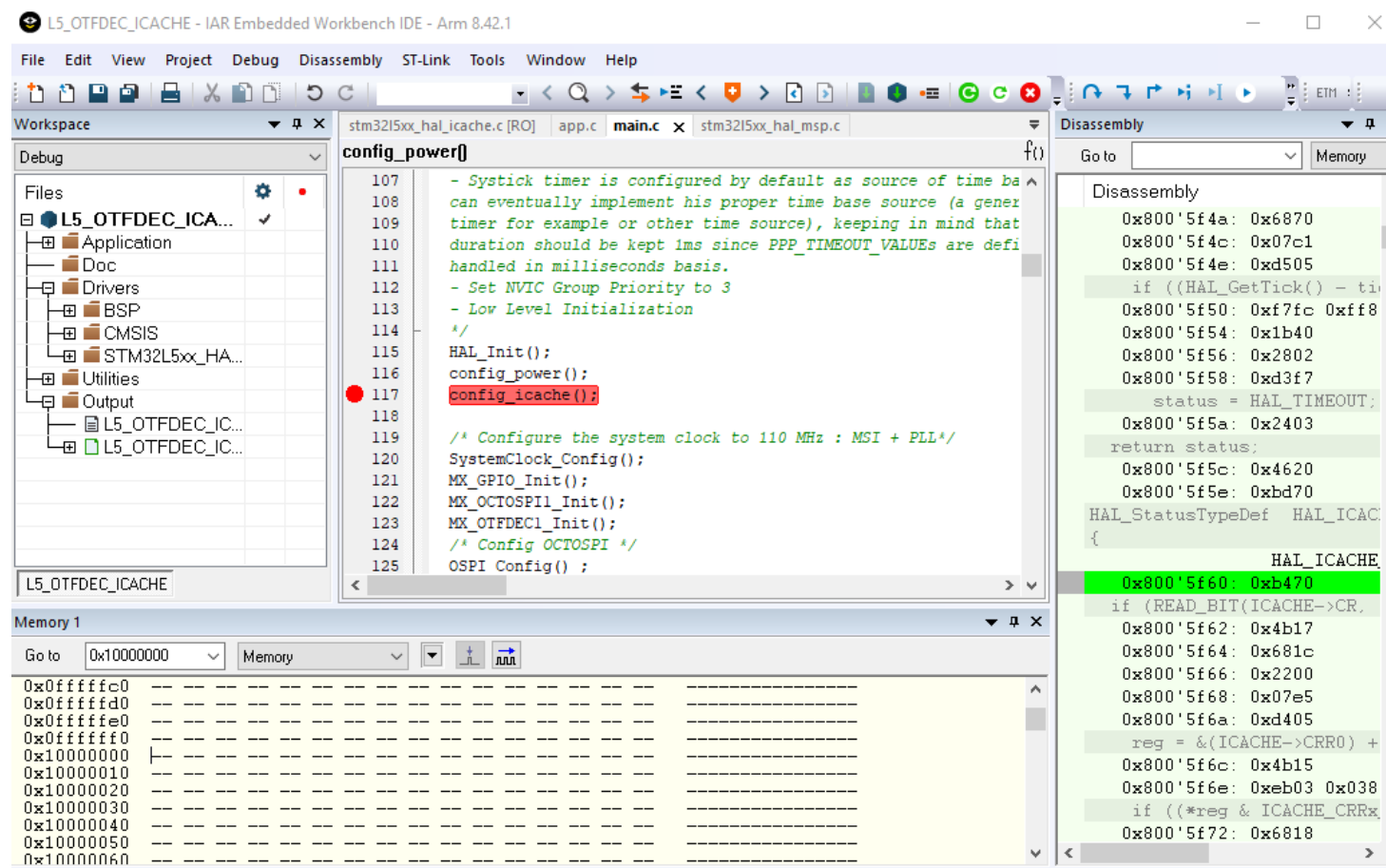
Read

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
0x10000000	44	49	43	00	45	49	43	00	52	49	43	00	53	54	42	59	DIC.EIC.RIC.STBY
0x10000010	00	00	00	00	53	4D	42	50	00	00	00	00	53	4E	42	50	....SMBP....SNBP
0x10000020	00	00	00	00	52	53	42	50	00	00	00	00	0D	0A	47	65	....RSBP.....Ge
0x10000030	74	20	53	4D	50	53	20	66	6F	72	63	65	20	62	79	70	t SMPS force byp
0x10000040	61	73	73	20	6D	6F	64	65	20	73	74	61	74	65	3A	20	ass mode state:
0x10000050	25	73	0D	0A	00	00	00	00	0D	0A	3D	3D	3D	3D	3D	3D	%s.....=====

```
$ xxd -g 1 -l 128 ExtSPINOR.bin
00000000: 44 49 43 00 45 49 43 00 52 49 43 00 53 54 42 59  DIC.EIC.RIC.STBY
00000010: 00 00 00 00 53 4d 42 50 00 00 00 00 53 4e 42 50  ....SMBP....SNBP
00000020: 00 00 00 00 52 53 42 50 00 00 00 00 0d 0a 47 65  ....RSBP.....Ge
00000030: 74 20 53 4d 50 53 20 66 6f 72 63 65 20 62 79 70  t SMPS force byp
00000040: 61 73 73 20 6d 6f 64 65 20 73 74 61 74 65 3a 20  ass mode state:
00000050: 25 73 0d 0a 00 00 00 00 0d 0a 3d 3d 3d 3d 3d 3d  %s.....=====
```

# 示例运行演示

- 通过编译器调试窗口观察不同阶段片外Flash代码地址访问情况 (1)
  - 运行到ICACHE配置之前
  - 0x10000000地址段无法访问





# 示例运行演示

- 通过编译器调试窗口观察不同阶段片外Flash代码地址访问情况 (2)
  - ICACHE和OSPI memory map配置完成后
  - 0x10000000和0x90000000地址内容都可以访问，但都是密文数据

The screenshot shows the IAR Embedded Workbench IDE with the source code of `main.c` on the left and the disassembly of the `OSPI_MemoryMap` function on the right. The source code includes comments for configuring the system clock to 110 MHz and activating memory mapping. The disassembly shows the instructions for setting the baud rate and word length for the COM port, and the memory mapping for the OSPI.

```
main()
{
    117 config_icache();
    118
    119 /* Configure the system clock to 110 MHz : MSI + PLL*/
    120 SystemClock_Config();
    121 MX_GPIO_Init();
    122 MX_OTFDEC1_Init();
    123 MX_OTFDEC2_Init();
    124 /* Config OCTOSPI */
    125 OSPI_Config();
    126 /* Activate memory mapping */
    127 OSPI_MemoryMap();
    128
    129 #if USE_UART_DEBUG
    130 /* Configure COM port */
    131 COM_Init.BaudRate = 115200;
    132 COM_Init.WordLength = COM_WORDLENGTH_8B;
    133 COM_Init.StopBits = COM_STOPBITS_1;
    134 COM_Init.Parity = COM_PARITY_NONE;
    135 COM_Init.HwFlowCtl = COM_HWCONTROL_NONE;
}
```

The disassembly of `OSPI_MemoryMap` shows the instructions for setting the baud rate and word length for the COM port, and the memory mapping for the OSPI.

```
Disassembly
0x800'5b64: 0xf7fe 0xfe2
0x800'5b68: 0xb100
0x800'5b6a: 0xe7fe
0x800'5b6c: 0x4885
0x800'5b6e: 0xf8c6 0x008
0x800'5b72: 0xf106 0x008
0x800'5b76: 0xf000 0xfa5
0x800'5b7a: 0xb100
0x800'5b7c: 0xe7fe
0x800'5b7e: 0xf000 0xfb4
OSPI_MemoryMap()
0x800'5b82: 0xf000 0xfcb
COM_Init.BaudRate = 115
0x800'5b86: 0xf44f 0x30e
COM_Init.WordLength = COM
0x800'5b8a: 0x2100
0x800'5b8c: 0x9007
0x800'5b8e: 0x9108
COM_Init.StopBits = COM
0x800'5b90: 0x2200
0x800'5b92: 0x9209
COM_Init.HwFlowCtl = COM
0x800'5b94: 0xf8ad 0x202
if (BSP_COM_Init(COM1, &O
0x800'5b98: 0xa907
```

The screenshot shows the IAR Embedded Workbench IDE with the source code of `main.c` on the left and the memory dump of the OSPI memory map on the right. The source code includes comments for configuring the system clock to 110 MHz and activating memory mapping. The memory dump shows the data stored in the OSPI memory map, which is encrypted.

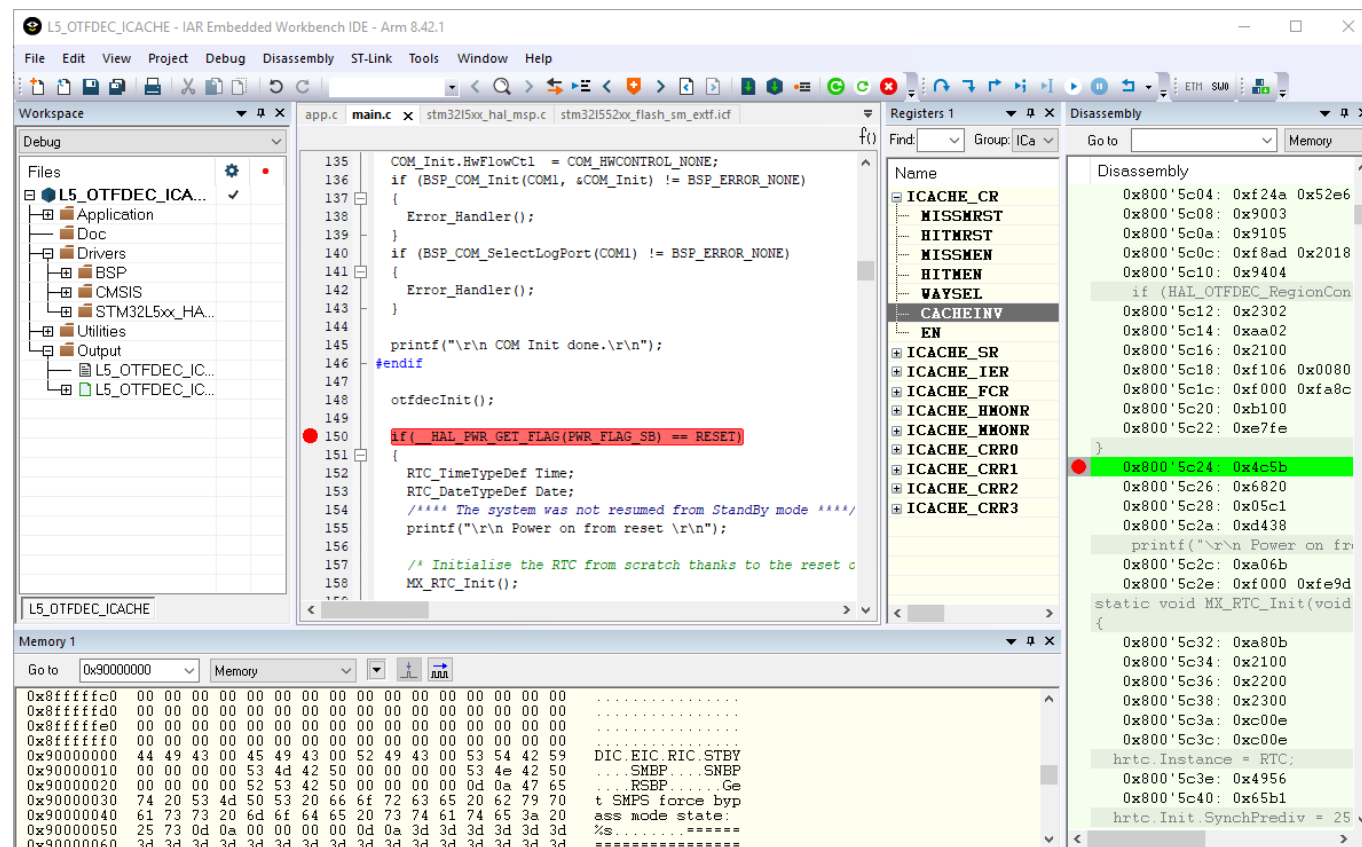
```
main()
{
    117 config_icache();
    118
    119 /* Configure the system clock to 110 MHz : MS
    120 SystemClock_Config();
    121 MX_GPIO_Init();
    122 MX_OTFDEC1_Init();
    123 MX_OTFDEC2_Init();
    124 /* Config OCTOSPI */
    125 OSPI_Config();
    126 /* Activate memory mapping */
    127 OSPI_MemoryMap();
    128
    129 #if USE_UART_DEBUG
    130 /* Configure COM port */
    131 COM_Init.BaudRate = 115200;
    132 COM_Init.WordLength = COM_WORDLENGTH_8B;
    133 COM_Init.StopBits = COM_STOPBITS_1;
    134 COM_Init.Parity = COM_PARITY_NONE;
    135 COM_Init.HwFlowCtl = COM_HWCONTROL_NONE;
}
```

The memory dump shows the data stored in the OSPI memory map, which is encrypted.

```
Memory 1
Go to 0x90000000 Memory
0x90000000 91 44 a4 c9 76 b9 c4 30 59 a1 68 29 ba 9c b5 9b .D.v..0Y.h)...
0x90000001 bb fd c5 6c 8b 6e 33 bd 38 d5 ff 6f d1 03 c3 df ...l.n3.8..o...
0x90000002 5f ef aa 06 c7 ef b4 27 7b 93 07 4f 36 d3 ae 8b ...{..06...
0x90000003 e4 c9 00 a8 08 9f e1 49 e3 ac 8a ad 95 4d 60 32 ...I...M'2...
0x90000004 14 6b dd 7c 20 04 bf b4 d2 6e 18 90 d4 b2 81 52 .k.|...n...R
0x90000005 6d 84 8f f5 53 f6 5a 2d cf a5 f7 0c 05 6a 2d 5d m...S.Z-...j-]
0x90000006 ac 78 91 54 9d bc 72 85 e4 15 5a c5 51 a0 fd 32 .x.T.r...Z.Q..2
0x90000007 9c 00 9a 0e 5a e6 8f 80 2b b9 e6 32 c3 8a f4 75 ...Z...+...2...u
0x90000008 56 b1 35 2b 7a f1 b3 22 3c a5 a7 d4 46 38 cf 2e V.5+z...<...F8..
0x90000009 75 83 25 c9 e2 0d 43 0c c6 de 84 d5 83 94 55 be u.%...C...U..
0x9000000a ca cd d9 fa h1 96 h1 h3 14 a6 nf 04 f7 78 ah de
```

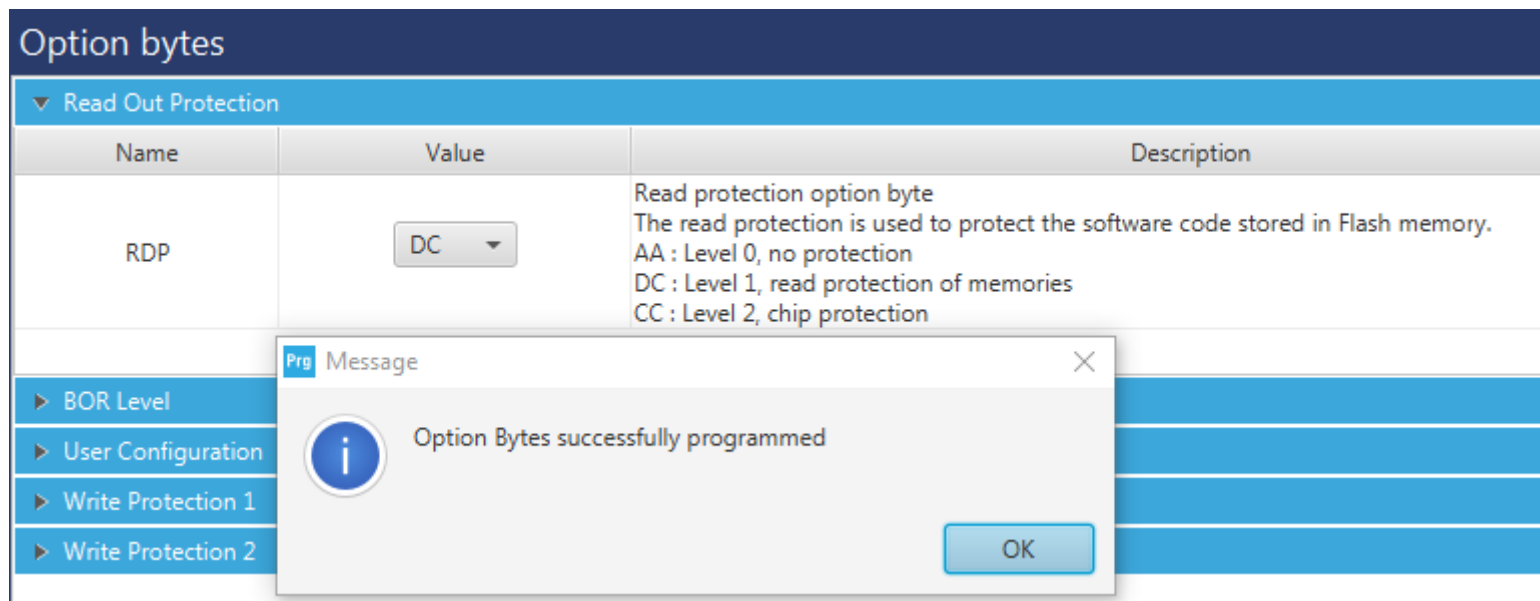
# 示例运行演示

- 通过编译器调试窗口观察不同阶段片外Flash代码地址访问情况 (3)
  - OTFDEC配置后可以在0x90000000地址读取到解密后的数据内容
  - 如果之前读取过0x10000000地址内容, 那么此时依旧会看到加密的内容, 需要做一次Cache Invalidate之后即可看到同样的明文内容
  - 只需要向ICACHE\_CR寄存器的CACHEINV bit写1



# 示例运行演示

- 在RDP为Level1的情况下，再次尝试观察解密后的片外Flash内容
  - 打开STM32CubeProgrammer的Option bytes窗口，设置RDP为Level 1



- 给L5-DK板断电后再上电，直到看到屏幕上的显示
- 再次用STM32CubeProgrammer以Hot plug模式连接DK板



# 示例运行演示

- 在RDP为Level1的情况下，再次尝试观察解密后的片外Flash内容（续）
  - 打开Memory & File edition窗口，查看0x10000000地址的内容，此时只能读到全0

Memory & File edition

Device memory +

Address: 0x10000000 Size: 0x400 Data width: 8-bit Read

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
0x10000000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x10000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x10000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x10000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x10000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x10000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x10000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x10000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x10000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x10000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x100000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x100000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x100000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x100000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

ST-LINK configuration

Serial number: 00320045313...  
Port: SWD  
Frequency (kHz): 24000  
Mode: Hot plug  
Access port: 0  
Reset mode: Hardware reset  
Shared: Disabled  
External loader: MX25LM51245G\_STM32L56...  
Target voltage: 3.27 V  
Firmware version: V3J3M2  
Firmware upgrade

- 当代码及其数据需要放置在片外Flash的时候，可以利用STM32L5的OTFDEC功能保护片外Flash上的代码和数据的机密性
  - 代码及数据以加密形式存放于片外Flash，OTFDEC能够对片外Flash访问的内容进行实时解密，从而允许CPU执行存储于片外Flash的加密代码
  - 当RDP设置为Level 1的时候，无法通过Debug端口读取解密后的片外Flash内容
- 结合ICACHE的remap功能，片外Flash加密代码依旧可以通过ICACHE进行访问加速，从而在保护机密性的同时提高片外代码的运行效率
- 示例程序给出了通过openssl加密片外Flash binary的例子，几个重点内容包括
  - 代码中配置给OTFDEC的加密Key和IV数据与openssl使用的Key和IV数据的对应
    - 注意大小端及顺序
  - Flash内容的binary数据的预处理和加密后binary数据的后处理
    - 被加密数据及加密后数据都需要经过以16字节为单位的byte swap的处理

# 重要通知 - 请仔细阅读

- 意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。
- 买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。
- ST 不对任何知识产权进行任何明示或默示的授权或许可。
- 转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。
- ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。
- 本文档中的信息取代本文档所有早期版本中提供的信息。

# Thank you

© STMicroelectronics - All rights reserved.

The STMicroelectronics corporate logo is a registered trademark of the STMicroelectronics group of companies. All other names are the property of their respective owners.



life.augmented