



## **PAN1070 DebugProtect 测试说明文档**

PAN-CLT-VER-A0, Rev 0.1

PANCHIP

PanchipMicroelectronics

[www.panchip.com](http://www.panchip.com)

## 修订历史

版本	修订日期	描述
V0.1	2023-10-20	初始版本创建
V0.2	2023-10-23	新增 3.5 应用笔记小节

PANCHIP

## 目录

第 1 章 测试目的 .....	4
第 2 章 测试内容 .....	5
2.1 测试内容 .....	5
2.2 环境准备 .....	5
2.2.1 软件环境 .....	5
2.2.1.1 待测代码 .....	5
2.2.1.2 软件工具 .....	5
2.2.2 硬件环境 .....	5
第 3 章 测试流程 .....	6
3.1 环境配置 .....	6
3.1.1 测试程序编译烧录 .....	6
3.1.2 硬件接线 .....	6
3.2 Debug Protect 工作流程 .....	6
3.3 测试程序初始化 .....	6
3.4 基本功能验证 .....	7
3.4.1 生成 Debug Key .....	7
3.4.2 将 Lock Key 写入 eFuse 并使能 Debug Protect 机制 .....	8
3.4.3 将 Unlock Key 载入解锁寄存器以使能 Debug 功能 .....	9
3.4.4 将 Unlock Key 从解锁寄存器中清除以禁止 Debug 功能 .....	11
3.5 应用笔记 .....	12
第 4 章 使用注意事项 .....	14

## 第1章 测试目的

1. PAN1070 Debug Protect 机制基本功能测试。
2. 通过测试，给出 Debug Protect 机制的使用说明文档。

PANCHIP

## 第2章 测试内容

### 2.1 测试内容

1. 生成 Debug Key（包括 16 字节的 Debug Unlock Key，以及 8 字节的 Debug Lock Key）
2. 将 8 字节 Debug Lock Key 写入 eFuse，并使能 Debug Protect 功能
3. 通过软件的方式将 16 字节 Debug Unlock Key 载入相关寄存器，以使能 Debug 功能
4. 通过软件的方式清空保存 16 字节 Debug Unlock Key 的寄存器，以禁止 Debug 功能

### 2.2 环境准备

#### 2.2.1 软件环境

##### 2.2.1.1 待测代码

测试工程文件：

<PAN1070-DK>\03\_MCU\mcu\_samples\DebugProtect\keil\DebugProtect.uvprojx

测试源文件目录：

<PAN1070-DK>\03\_MCU\mcu\_samples\sample\DebugProtect\src

##### 2.2.1.2 软件工具

- 1、SecureCRT（用于显示 PC 与 Test Board 的交互过程，打印 log 等）

#### 2.2.2 硬件环境

- 1、PAN1070 COB Test Board 1 块：

- a) UART0（测试交互接口，TX：P16，RX：P17，波特率：921600）
- b) SWD（用来调试和烧录程序，SWDCLK: P00，SWDIO: P01）

- 2、JLink（SWD 调试与烧录工具）

## 第3章 测试流程

### 3.1 环境配置

#### 3.1.1 测试程序编译烧录

打开测试工程，确保可以编译通过。

#### 3.1.2 硬件接线

测试过程中无需额外接线。

### 3.2 Debug Protect 工作流程

Debug Protect 机制是用来保护 SoC 数据的一种方法，当使能此机制后，默认将无法正常通过 SWD 接口与芯片通信：SWD 写数据将不会生效（除解锁保护的寄存器区域外），读数据将强制返回全 0；只有当向解锁保护的寄存器区域写入正确的 key 之后，SWD 通信才会恢复正常。

为使能 Debug Protect 机制，需做如下操作：

1. 生成 Debug Key，其包含两种 Key：
  - a) Debug Unlock Key，用来解锁 Debug 保护机制的密钥，长度 16 字节，可以为任意随机值
  - b) Debug Lock Key，用来使能 Debug 保护机制的密钥，长度 8 字节，其由 Debug Unlock Key 经过特定规则转换而来
2. 在 ROM 权限下，将 8 字节的 Debug Lock Key 写入 eFuse（地址：0x10 ~ 0x17）
3. 在 ROM 权限下，烧录 eFuse 0x1B 地址的 BIT2，以使能 Debug Protect 功能
4. （经过上述操作后，复位 SoC，之后将无法正常使用 SWD 读写数据）
5. 将 16 字节的 Debug Unlock Key 写入解锁保护的寄存器区域（位于 eFuse Ctrl 寄存器中，地址：0x40080068 ~ 0x40080077）
6. （经过上述操作后，将可以重新正常使用 SWD 功能）

### 3.3 测试程序初始化

硬件连线完成并烧录测试程序后，Test Board 上电，观察 Debug Port 是否正常打印测试主菜单。

```
CPU @ 48000000Hz
```

```
eFuse content after reset:
```

```
Debug Key in eFuse:
```

```
    Addr:Data      Addr:Data      Addr:Data      Addr:Data
    0x10:0x00(OK)  0x11:0x00(OK)  0x12:0x00(OK)  0x13:0x00(OK)
    0x14:0x00(OK)  0x15:0x00(OK)  0x16:0x00(OK)  0x17:0x00(OK)
```

```
Secure Enable Control Flag in eFuse:
```

```
0x1B:0x00
```

```
+-----+
|                                     |
|                               PAN1070 Debug Protect Sample Code. |
|                                     |
| Press key to start specific testcase: |
|                                     |
| Input '0'   Testcase 0: Generate Debug Key. |
| Input '1'   Testcase 1: Write Generated Debug Key to eFuse. |
| Input '2'   Testcase 2: Apply debug key to enable SWD Debug. |
| Input '3'   Testcase 3: Clear debug key to disable SWD Debug. |
|                                     |
+-----+
```

## 3.4 基本功能验证

### 3.4.1 生成 Debug Key

在主菜单下，输入 ‘0’ 命令，生成 Debug Key:

测试目的:

生成 Debug Protect 机制需要的两种 Debug Key。

测试现象:

```
0
Debug Key in Registers: 0x0a320bfa 0x32a24f95 0x29fe9e58 0x18e2403f
Debug Key in eFuse: 0xfa 0x0a 0xb0 0x0a 0x58 0x9e 0x01 0xe2
```

```
+-----+
|                                     |
|                               PAN1070 Debug Protect Sample Code. |
|                                     |
| Press key to start specific testcase: |
|                                     |
| Input '0'   Testcase 0: Generate Debug Key. |
| Input '1'   Testcase 1: Write Generated Debug Key to eFuse. |
| Input '2'   Testcase 2: Apply debug key to enable SWD Debug. |
| Input '3'   Testcase 3: Clear debug key to disable SWD Debug. |
|                                     |
+-----+
```

测试分析:

成功生成了两种 Debug Key，其中:

Debug Unlock Key 为: Debug Key in Registers: 0x0a320bfa 0x32a24f95 0x29fe9e58 0x18e2403f

Debug Lock Key 为: Debug Key in eFuse: 0xfa 0x0a 0xb0 0x0a 0x58 0x9e 0x01 0xe2

**注意:** 本例程为演示方便，实际上每次生成的 Debug Key 均为相同的值。

### 3.4.2 将 Lock Key 写入 eFuse 并使能 Debug Protect 机制

在主菜单下，输入 ‘1’ 命令，将前一步生成的 Lock Key 写入 eFuse 并使能 Debug Protect 机制：

测试目的：

确认 Debug Lock Key 可以成功写入 eFuse 并使能 Debug Protect 机制。

测试现象：

```

1
eFuse content after write before reset:
Debug key in eFuse:
  Addr:Data      Addr:Data      Addr:Data      Addr:Data
  0x10:0xfa(OK)  0x11:0x0a(OK)  0x12:0xb0(OK)  0x13:0x0a(OK)
  0x14:0x58(OK)  0x15:0x9e(OK)  0x16:0x01(OK)  0x17:0xe2(OK)

Secure Enable Control Flag in eFuse:
  0x1B:0x04

Trigger reset to apply the new configuration in eFuse..

CPU @ 48000000Hz

eFuse content after reset:
Debug key in eFuse:
  Addr:Data      Addr:Data      Addr:Data      Addr:Data
  0x10:0x00(N/A)  0x11:0x00(N/A)  0x12:0x00(N/A)  0x13:0x00(N/A)
  0x14:0x00(N/A)  0x15:0x00(N/A)  0x16:0x00(N/A)  0x17:0x00(N/A)

Secure Enable Control Flag in eFuse:
  0x1B:0x04
  
```

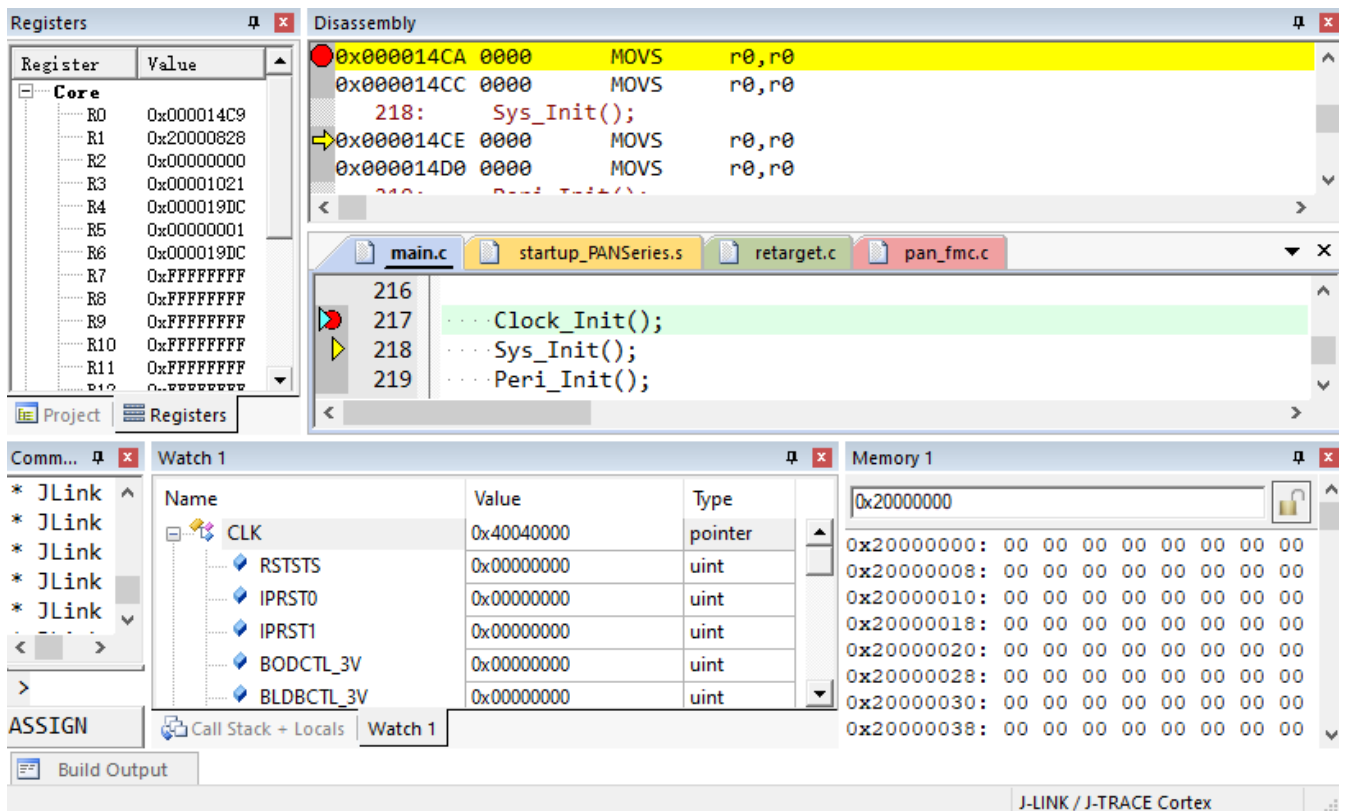
```

+-----+
|                PAN1070 Debug Protect Sample Code.                |
+-----+
| Press key to start specific testcase:                             |
| Input '0'   Testcase 0: Generate Debug Key.                       |
| Input '1'   Testcase 1: Write Generated Debug Key to eFuse.       |
| Input '2'   Testcase 2: Apply debug key to enable SWD Debug.     |
| Input '3'   Testcase 3: Clear debug key to disable SWD Debug.    |
+-----+
  
```

此时点击 Keil 的 Debug 按钮，进入 Debug 模式，可以看到此时已经无法正常 Debug：

- 反汇编界面已经无法正常显示反汇编程序（变为全 0）
- Watch 界面已经无法正常显示各个寄存器和变量的值（变为全 0）
- Memory 界面已经无法正常显示 SRAM 中的值（变为全 0）





### 测试分析:

eFuse 写入后，立刻将 Debug Lock Key 与 Debug Protect 使能控制位从 eFuse 中读出来，可以发现均成功写入。

稍后软件触发芯片复位，上电后立刻重新将上述两块 eFuse 区域读出来，可以看到 Debug Key 已经无法正常读出（返回全 0），而 Debug Protect 使能控制位仍可成功读出，其 BIT2 被置位 1，表示 Debug Protect 机制已经正常使能。

这时候打开 Keil 的 Debug 界面，可以看到虽然程序可以全速执行，但单步调试功能无法正常使用，且各类 Watch 机制观察到的数据均变为全 0。

### 3.4.3 将 Unlock Key 载入解锁寄存器以使能 Debug 功能

在调试模式的全速运行状态下，先输入 ‘0’ 命令，重新生成 Debug Key，然后再输入 ‘2’ 命令，将 Unlock Key 载入芯片解锁寄存器：

#### 测试目的:

由于前一步操作中有复位，保存在全局变量中的 Debug Key 会丢失，因此需要重新生成 Debug Key；然后将 Debug Unlock Key 载入芯片解锁寄存器，以使能 Debug 功能。

#### 测试现象:

在调试界面下全速执行，并重新输入 ‘0’ 命令以生成 Debug Key（本例程为演示方便，每次生成的 Debug Key 均相同）：

```
0
Debug Key in Registers: 0x0a320bfa 0x32a24f95 0x29fe9e58 0x18e2403f
Debug Key in eFuse: 0xfa 0x0a 0xb0 0x0a 0x58 0x9e 0x01 0xe2
```

```
+-----+
|                PAN1070 Debug Protect Sample Code.                |
+-----+
| Press key to start specific testcase:                             |
|                                                                    |
| Input '0'   Testcase 0: Generate Debug Key.                       |
| Input '1'   Testcase 1: Write Generated Debug Key to eFuse.      |
| Input '2'   Testcase 2: Apply debug key to enable SWD Debug.     |
| Input '3'   Testcase 3: Clear debug key to disable SWD Debug.    |
+-----+
```

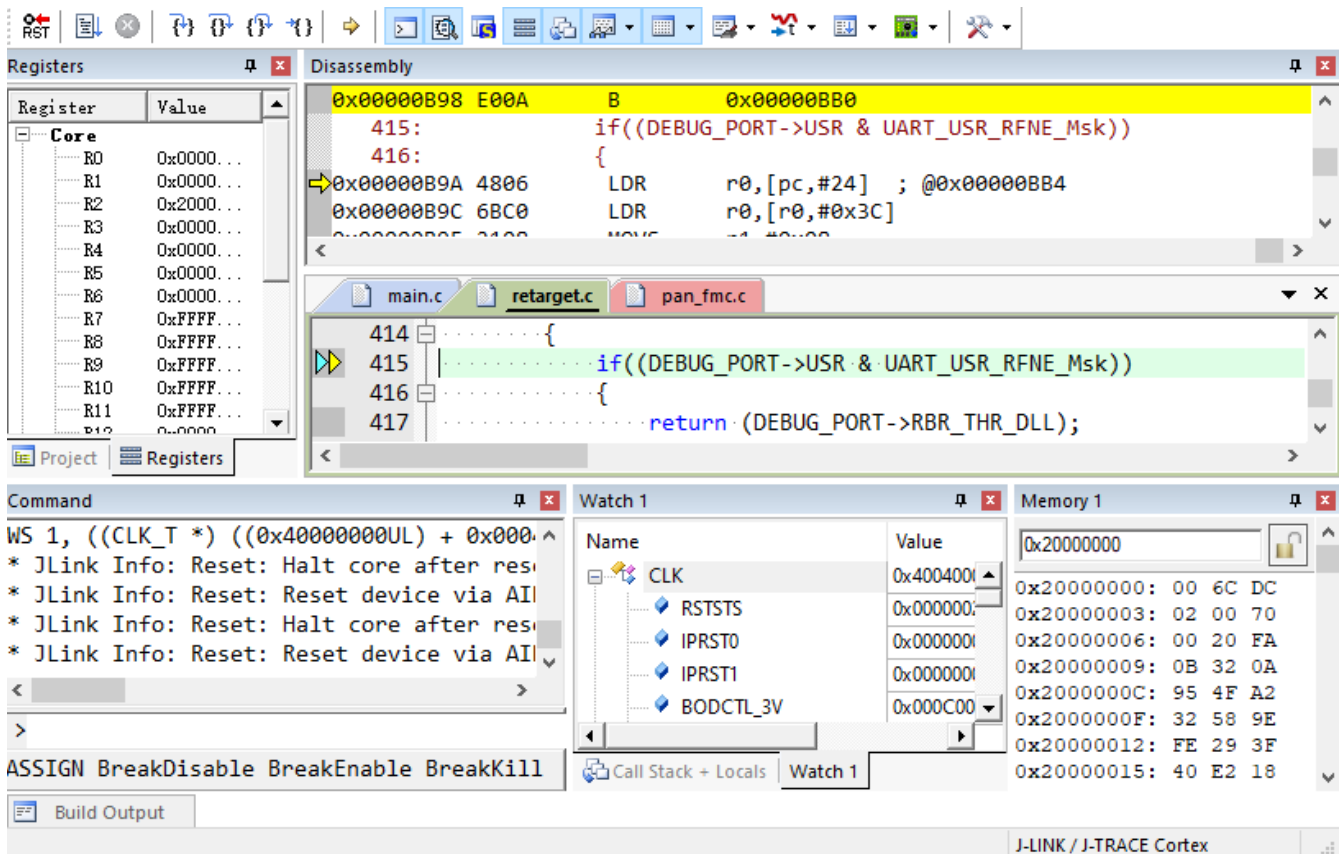
然后再输入 ‘2’ 命令以将生成的 Unlock Key 载入芯片的解锁寄存器中：

```
2
Debug Key Applied to Enable SWD Debug: 0x0a320bfa 0x32a24f95 0x29fe9e58 0x18
e2403f
```

```
+-----+
|                PAN1070 Debug Protect Sample Code.                |
+-----+
| Press key to start specific testcase:                             |
|                                                                    |
| Input '0'   Testcase 0: Generate Debug Key.                       |
| Input '1'   Testcase 1: Write Generated Debug Key to eFuse.      |
| Input '2'   Testcase 2: Apply debug key to enable SWD Debug.     |
| Input '3'   Testcase 3: Clear debug key to disable SWD Debug.    |
+-----+
```

成功后再观察 Keil 的 Debug 界面，点击一次单步按钮，可以发现各种 Debug 功能均已恢复正常：

- 反汇编界面重新正常显示反汇编程序
- Watch 界面重新正常显示各个寄存器和变量的值
- Memory 界面重新正常显示 SRAM 中的值



**测试分析:**

将 Debug Unlock Key 载入芯片解锁寄存器后，可以看到 SWD Debug 功能恢复正常。

**3.4.4 将 Unlock Key 从解锁寄存器中清除以禁止 Debug 功能**

在调试模式的全速运行状态下，输入 ‘3’ 命令，将之前载入芯片解锁寄存器的 Unlock Key 清除：

**测试目的:**

将前一步载入芯片解锁寄存器的 Unlock Key 清除，以重新禁止 Debug 功能。

**测试现象:**

在调试界面下全速执行，输入 ‘3’ 命令：

```
3
Debug key cleared to 0 to Disable SWD Debug
```

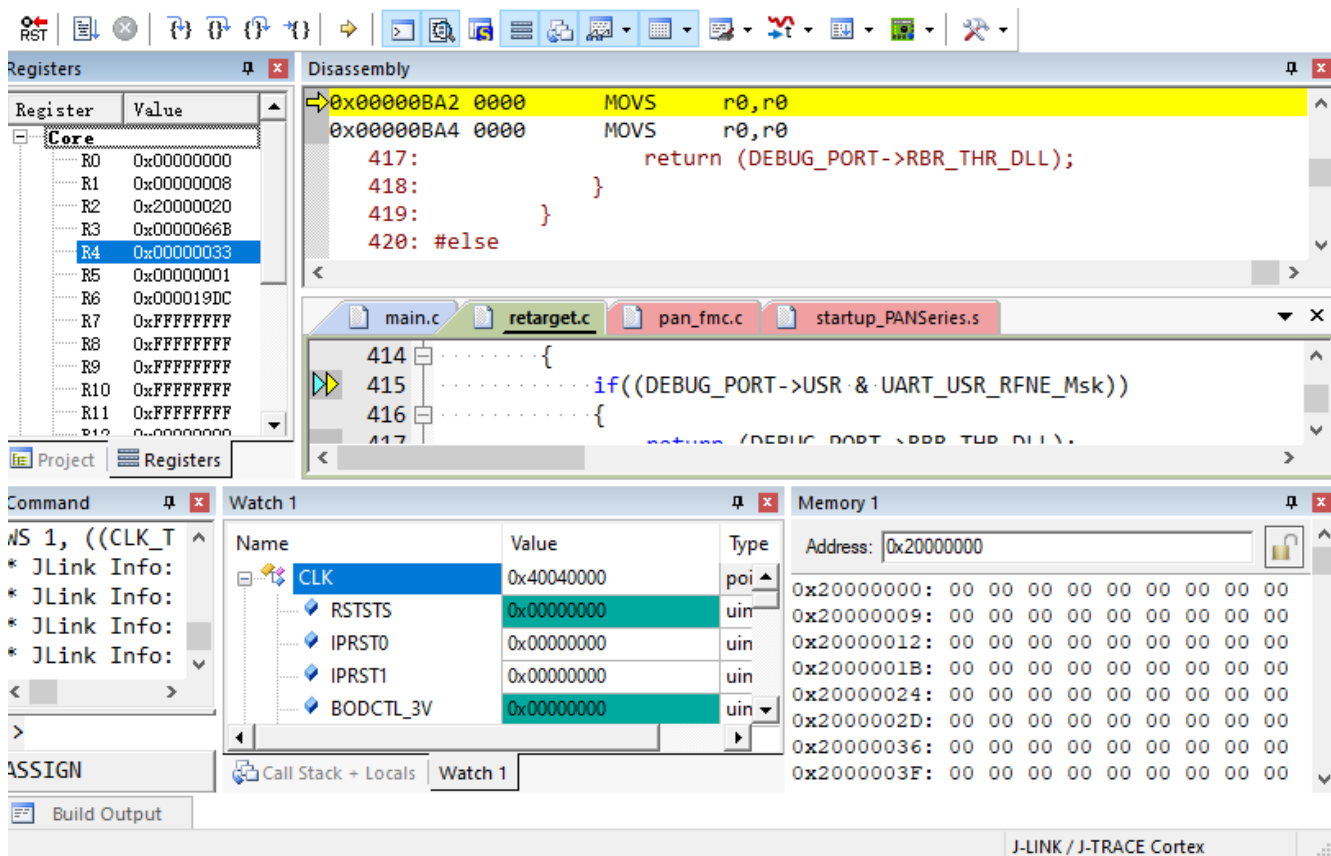
```

+-----+
|                PAN1070 Debug Protect sample Code.                |
+-----+
| Press key to start specific testcase:                             |
|                                                                    |
| Input '0'   Testcase 0: Generate Debug Key.                     |
| Input '1'   Testcase 1: Write Generated Debug Key to eFuse.     |
| Input '2'   Testcase 2: Apply debug key to enable SWD Debug.   |
| Input '3'   Testcase 3: Clear debug key to disable SWD Debug.  |
+-----+

```

然后在 Keil Debug 界面点击单步运行，可以发现此时又无法正常 Debug:

- 反汇编界面已经无法正常显示反汇编程序（变为全 0）
- Watch 界面已经无法正常显示各个寄存器和变量的值（变为全 0）
- Memory 界面已经无法正常显示 SRAM 中的值（变为全 0）



### 测试分析:

将 Debug Unlock Key 从芯片解锁寄存器清除后，可以看到 SWD Debug 功能又无法正常使用。

## 3.5 应用笔记

当我们将 Debug Lock Key 写入 eFuse 并使能 Debug Protect 机制后，所有通过 SWD 实现的将会均无法正常使用，包括 SWD 调试与烧录。

为使 SWD 功能恢复正常，需向 SoC 特定寄存器中写入 Debug Unlock Key。在前面的测试流程中，演示了在程序中预留 Debug 解锁命令，通过执行此命令解锁 Debug 功能，但是此方法使用起来不是很方便。这里我们演示另一种方法，通过 JLink Script 钩子脚本的方式，在 Keil 界面中每次点击 Download 按钮下载程序的时候，或点击 Debug 按钮调试程序的时候，均会自动执行此钩子脚本，将 Debug Unlock Key 载入 SoC 中，使能 SWD 功能。

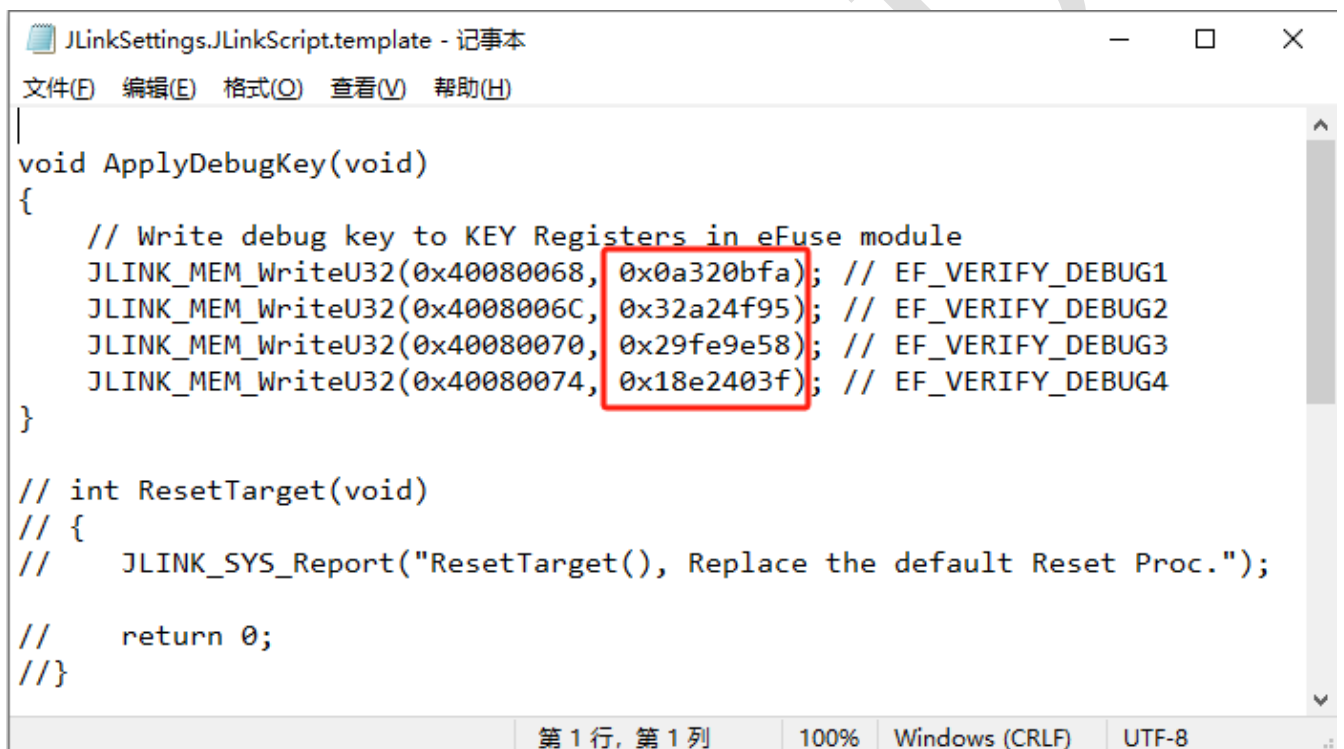
JLink Script 钩子脚本位于<PAN1070-DK>\03\_MCU\mcu\_samples\sample\DebugProtect\keil 目录下，名为 **JLinkSettings.JLinkScript.template**

为使此脚本生效，我们将其重命名为 **JLinkSettings.JLinkScript**（即直接删除此脚本文件名中的尾部.template 字样）

之后我们点击 Keil Download 按钮或者 Debug 按钮，即可正常烧录或调试程序了。

另外需要注意：

1. 上述 JLink Script 脚本文件中，里面的 Debug Unlock Key 是本例程演示的 Key，在实际使用过程中要用当前 SoC 中实际烧录的 Lock Key 对应的 Unlock Key 替换掉文件中默认的 Key:



```
JLinkSettings.JLinkScript.template - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

void ApplyDebugKey(void)
{
    // Write debug key to KEY Registers in eFuse module
    JLINK_MEM_WriteU32(0x40080068, 0x0a320bfa); // EF_VERIFY_DEBUG1
    JLINK_MEM_WriteU32(0x4008006C, 0x32a24f95); // EF_VERIFY_DEBUG2
    JLINK_MEM_WriteU32(0x40080070, 0x29fe9e58); // EF_VERIFY_DEBUG3
    JLINK_MEM_WriteU32(0x40080074, 0x18e2403f); // EF_VERIFY_DEBUG4
}

// int ResetTarget(void)
// {
//     JLINK_SYS_Report("ResetTarget(), Replace the default Reset Proc.");
//
//     return 0;
// }
```

2. 当我们需要在已使能 Debug Protect 机制的 SoC 上烧录或调试程序时，只需将 JLink Script 脚本拷贝到 Keil 工程目录下（与.uvprojx 工程文件在同一目录下）即可。



## 第4章 使用注意事项

- 1、使能 Debug Protect 机制需要写 eFuse，而 eFuse 是一种 otp (One-Time-Program) 存储器，其任意 bit 写为 1 后将会无法恢复为 0，这一点需要特别注意。
- 2、为防止 eFuse 被误操作，默认情况下对 eFuse (除 0x7C~0x7F 以外的地址) 进行写操作是不会成功的；当需要写 eFuse 中与 Debug Protect 有关的区域时，需先配置 eFuse 控制寄存器，提升操作权限：

```
EFUSE->EF_FLASH_PERMISSION |= EFUSE_FLASH_PERMISSION_CTRL_Msk;
```

之后再对 eFuse 进行写操作即可成功；成功后建议重新将上述控制寄存器清掉：

```
EFUSE->EF_FLASH_PERMISSION &= ~EFUSE_FLASH_PERMISSION_CTRL_Msk;
```

- 3、在用户产品量产阶段，可以使用 Panchip 提供的量产烧录工具 PANLINK 进行写 eFuse 操作 (包括 Debug Key 与 Debug Protect 机制使能控制位)。
- 4、每颗 PAN1070 SoC 的 eFuse 中均会保存一组独一无二的芯片 ID 信息 (UID)，在终端产品中，用户可在自己的产品量产阶段为每一颗 SoC 生成一组 Debug Key，保存在自己的量产数据库中，调试的时候可通过芯片 UID 查找到对应的 Debug Key，然后将其载入 SoC 中以重新使能 SWD Debug 功能。