

ZT0007 专题教程

作者：Eric2013

终极调试工具 EventRecorder 使用方法，各种 Link 通吃

销售QQ: 1295744630

销售旺旺: armfly

微信公众号: 安富莱电子

销售电话: 13638617262

邮箱: armfly@qq.com

公司网址: www.anfulai.cn

技术支持论坛: www.armbbs.cn

淘宝直销: anfulai.taobao.com



武汉安富莱电子有限公司

专业开发板、显示模块制造商

承接项目开发 (提供生产供货服务)

继前面的专题教程推出 SEGGER 的 RTT, JScope, Micrium 的 uC/Probe 之后, 再出一期终极调试方案 Event Recorder, 之所以叫终极解决方案, 是因为所有 Link 通吃, 支持时间测量, 功耗测量, printf 打印, RTX5 及其所有中间件调试信息展示。

- 1.1 重要提示 (必读)
- 1.2 Event Recorder 简介
- 1.3 创建工程模板和注意事项
- 1.4 Event Recorder 事件记录的实现
- 1.5 Event Recorder 实现 printf 重定向
- 1.6 Event Statistics 时间测量功能的实现
- 1.7 Event Statistics 功耗测量功能的实现
- 1.8 Event Recorder 对 RTX5 及其所有中间件的支持
- 1.9 JLINK 配置说明
- 1.10 STLINK 配置说明
- 1.11 CMSIS-DAP 配置说明
- 1.12 ULINK 配置说明
- 1.13 配套例子
- 1.14 总结

1.1 重要提示 (必读)

- ◆ 只要是 MDK 支持的调试下载器, 基本都支持 Event Recorder, 本教程测试了 JLINK, STLINK 和 CMSIS-DAP。
- ◆ 使用最新版本的 MDK5.25, 下载地址:
<https://www.armbbs.cn/forum.php?mod=viewthread&tid=85789> 。
- ◆ 使用 ARM_Compiler 软件包当前的最新版本 V1.4.0。详情看此贴:
<https://www.armbbs.cn/forum.php?mod=viewthread&tid=87175> 。
- ◆ 务必使用当前最新的 CMSIS 软件包 V5.3.0。详情本教程 1.3 小节末尾的说明。
- ◆ 如果大家的 MDK5.X 应用不是很熟练的话, 可以看论坛网友翻译的 MDK5.X 入门手册:
<https://www.armbbs.cn/forum.php?mod=viewthread&tid=31288> 。
- 如果觉得看手册上手慢的话, 可以直接看 KEIL 官方做的 MDK 入门系列视频, 带中文字幕:
<https://www.armbbs.cn/forum.php?mod=viewthread&tid=82667> 。
- ◆ 为了实现 Event Recorder 组件的最高性能, 最好将下载器的时钟速度设置到所支持的最大值, 另外, 根据需要加大 EventRecorderConf.h 文件中的缓冲大小, 默认可以缓冲 64 个消息 (动态更新的 FIFO

空间)。

- ◆ 此调试组件不需要用到 SWO 引脚,使用标准的下载接口即可。以我们的开发板为例,用到 VCC, GND, SWDIO, SWCLK 和 NRST。大家使用三线 JLINK-OB 也是没问题的, 仅需用到 GND, SWDIO 和 SWCLK。

1.2 Event Recorder 简介

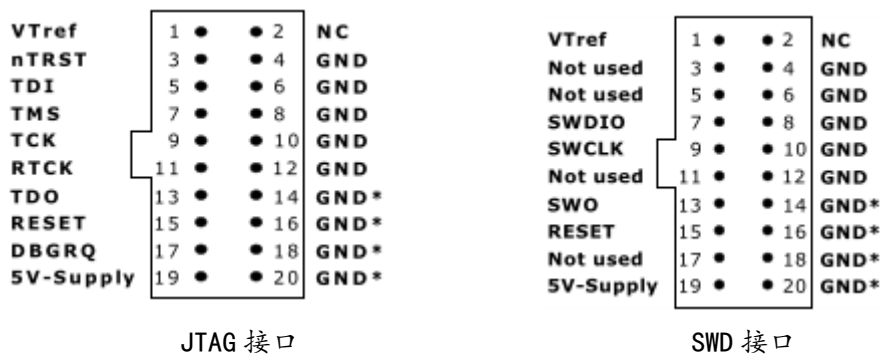
前面的专题教程中为大家讲解了使用 SEGGER 的 RTT 功能来替代串口打印, 比较方便。只是这种方法限制用户必须使用 JLINK 才可以。而使用 Event Recorder 的话, 无此限制, 各种 LINK 通吃。只要是 MDK 支持的即可。

Event Recorder 是 MDK 在 5.22 版本的时增加的功能, 到了 5.25 版本后, 这个功能就更加完善了, 增加了时间测量和功耗测量的功能。

此调试组件不需要用到 SWO 引脚,使用标准的下载接口即可。以我们的开发板为例,用到 VCC, GND, SWDIO, SWCLK 和 NRST。大家使用三线 JLINK-OB 也是没问题的, 仅需用到 GND, SWDIO 和 SWCLK。

● JTAG 接口和 SWD 接口区别

下图分别是 20pin 的标准 JTAG 引脚和 SWD (Serial Wire Debug) 引脚, 一般 SWD 接口仅需要 Vref, SWDIO, SWCLK, RESET 和 GND 五个引脚即可, SWO (Serial Wire Output) 引脚是可选的。有了 SWO 引脚才可以实现数据从芯片到电脑端的数据发送。



● 词条 SWV (Serial Wire Viewer)

SWV 是由仪器化跟踪宏单元 ITM (Instrumentation Trace Macrocell) 和 SWO 构成的。SWV 实现了一种从 MCU 内部获取信息的低成本方案, SWO 接口支持输出两种格式的跟踪数据, 但是任意时刻只能使用一种。两种格式的数据编码分别是 UART (串行) 和 Manchester (曼彻斯特)。当前 JLINK 仅支持 UART 编码, SWO 引脚可以根据不同的信息发送不同的数据包。当前 M3/M4 可以通过 SWO 引脚输出以下三种信息:

1. ITM 支持 printf 函数的 debug 调用 (工程需要做一下接口重定向即可)。ITM 有 32 个通道, 如果使用 MDK 的话, 通道 0 用于输出调试字符或者实现 printf 函数, 通道 31 用于 Event Viewer, 这就是为什么实现 Event Viewer 需要配置 SWV 的原因。

2. 数据观察点和跟踪 DWT (Data Watchpoint and Trace) 可用于变量的实时监测和 PC 程序计数器采样。
3. ITM 还附带了一个时间戳的功能：当一个新的跟踪数据包进入了 ITM 的 FIFO 时，ITM 就会把一个差分的时间戳数据包插入到跟踪数据流中。跟踪捕获设备在得到了这些时间戳后，就可以找出各跟踪数据之间的时间相关信息。另外，在时间戳计数器溢出时也会发送时间戳数据包。

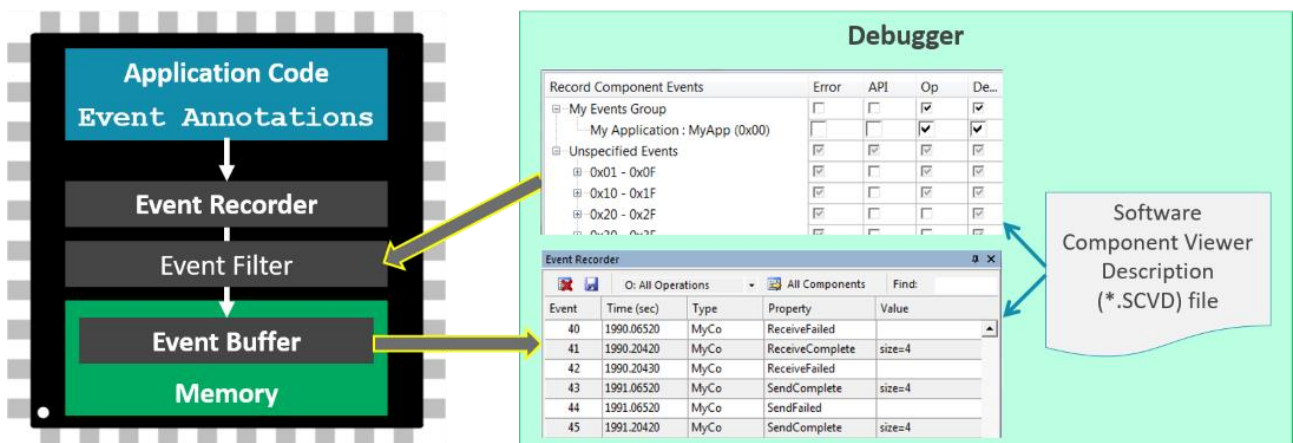
1.2.1 Event Recorder 的特色

Event Recorder 的特色主要有以下几点：

1. 提升应用程序动态执行期间的检测能力。
2. 支持的事件类型滤除机制，比如运行错误，API 调用，内部操作和操作信息的区分。
3. 可以在任务中，RTOS 内核中和中断服务程序中任意调用。
4. 对于带 ITM 功能的 Cortex-M3/M4/M7/M33 内核芯片，执行记录期间，全程无需开关中断操作。对于不带 ITM 功能的 Cortex-M0/M0+/M23，是需要开关中断的。
5. 支持 printf 重定向。
6. 各种 link 通吃，支持 SWD 接口或者 JTAG 接口方式的 JLINK、STLINK、ULINK 和 CMSIS-DAP。
7. 对于带 DWT 时钟周期计数器功能的 Cortex-M3/M4/M7/M33 内核芯片，创建时间戳时，可以有效降低系统负担，无需专用定时器来实现。
8. Event Recorder 执行时间具有时间确定性，即执行的时间是确定的，而且执行速度超快，因此，实际产品中的代码依然可以带有这部分，无需创建 debug 和 release 两种版本。
9. RTX5 及其所有中间件都支持 Event Recorder 调试。

1.2.2 Event Recorder 是如何工作的

首先来看下面这张图：



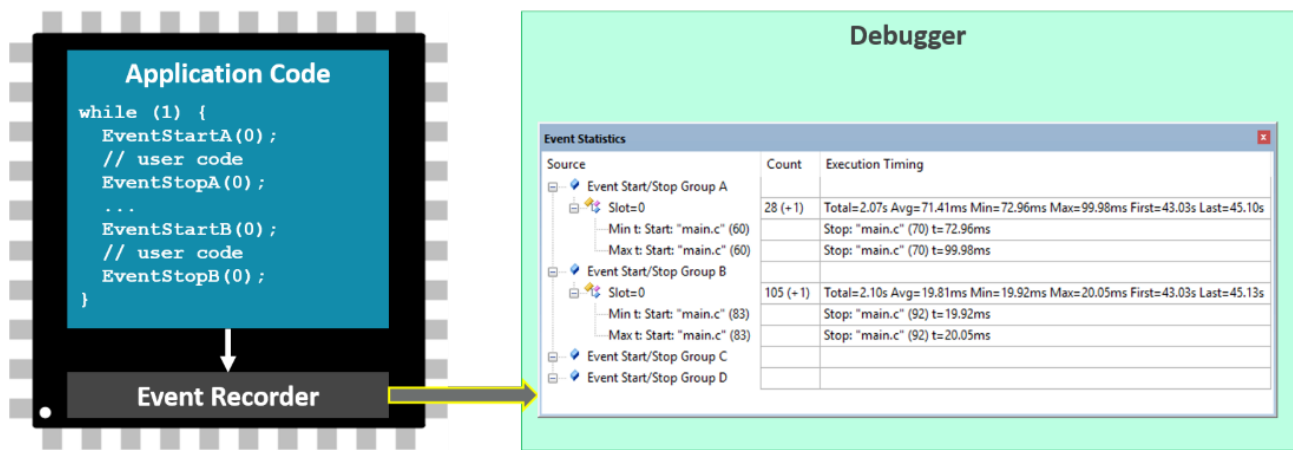
在截图的左下角有个 Memory 内存区，在这个内存区里面有一个缓冲 Event Buffer，其实就是一个

大数组。MDK 通过访问这个数组实现消息的图形化展示。为了正确的图形化展示，数组缓冲里面的数据就得有一定的数据格式。而这个数据格式就是通过左侧截图里面的 Event Recorder 和 Event Filter 来实现的。Event Recorder 的 API 实现数据记录和整理，Event Filter 的 API 实现数据的筛选，从而可以选择哪些数据可以在 MDK 的 Event Recorder 调试组件里面展示出来。

这就是 Event Recorder 的基本工作流程。

1.2.3 Event Statistics 时间测量功能

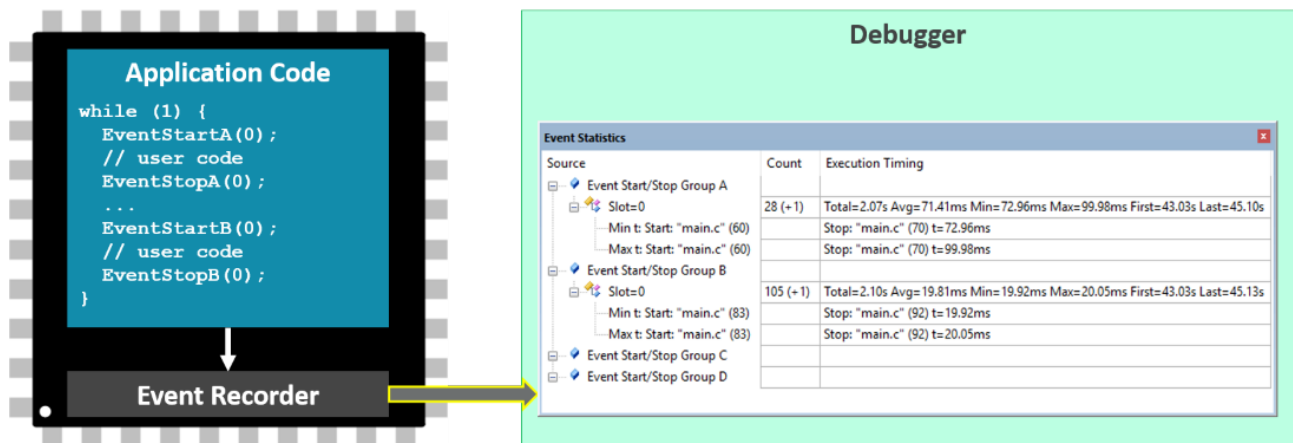
Event Statistics 提供的时间测量功能简单易用，在测试代码前后加上测量函数即可：



在本教程的 1.6 小节为大家详细进行了讲解。通过这个时间测量功能，用户可以方便测试代码的执行时间，从而根据需要，进行合理的优化，提高代码执行效率。

1.2.4 Event Statistics 功耗测量功能

Event Statistics 提供的功耗测量功能，当前只有 KEIL 的 ULINKplus 支持此功能，由于 ULINKplus 价格不便宜，一套 5000 多，大家作为了解即可，实际效果如下：



1.2.5 Event Recorder 的实现原理

每条 Event Recorder 消息是由 16 字节的数据组成, 32 位的 ID, 32 位的时间戳, 两个 32 位的数据, 共计 16 个字节。其中 32 位 ID 最重要, 格式如下:

ID	位	描述
Message number	Bit[0:7]	标识软件组件的事件消息信息, 其实就是消息 ID
Component number	Bit[8:15]	标识软件组件 (也用于消息筛选)
Level	Bit[16:17]	指定用于消息筛选的类别
保留	Bit[18:31]	设置为 0

Level 指定消息分类, 主要用于消息筛选:

Level	描述
EventLevelError = 0	运行错误
EventLevelAPI = 1	API 函数调用
EventLevelOp = 2	内部操作
EventLevelDetail = 3	更多详细的操作信息

Component number 指定事件消息所属的软件组件, 也可用于过滤:

Component number	描述
0x0 .. 0x3F (0 .. 63)	用于用户应用程序的软件组件
0x40 .. 0x7F (64 .. 127)	第三方中间件
0x80 .. 0xEE (128 .. 238)	MDK 中间件
0xEF (239)	用于 Event Statistics 的启动和停止事件
0xF0 .. 0xFC (240 .. 253)	RTOS 内核
0xFD (253)	进程间通信层 (用于多核系统)
0xFE (254)	printf 重定向
0xFF (255)	Event Recorder 消息

看了下 Event Recorder 的源码, 每条消息大体是一样的:

```
typedef struct {  
    uint32_t ts;           // Timestamp (32-bit, Toggle bit instead of MSB)  
    uint32_t val1;         // Value 1 (32-bit, Toggle bit instead of MSB)  
    uint32_t val2;         // Value 2 (32-bit, Toggle bit instead of MSB)  
    uint32_t info;         // Record Information  
                           // [ 7.. 0]: Message ID (8-bit)  
                           // [15.. 8]: Component ID (8-bit)  
                           // [18..16]: Data Length (1..8) / Event Context  
                           //      [19]: IRQ Flag  
                           // [23..20]: Sequence Number  
};
```

```
// [24]: First Record
// [25]: Last Record
// [26]: Locked Record
// [27]: Valid Record
// [28]: Timestamp MSB
// [29]: Value 1 MSB
// [30]: Value 2 MSB
// [31]: Toggle bit
} EventRecord_t;
```

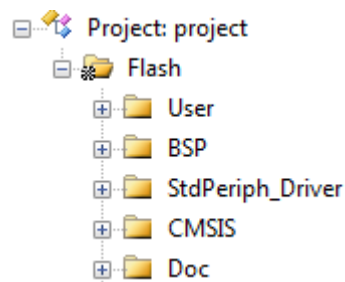
其中参数成员 info 最重要，也就是前面说的 32 位 ID，这里的说明与前面的说明稍有不同。这里是经过处理后，实际存储到 Event Recorder 缓冲里面的数据。

对于 Event Recorder，大家了解了这些知识点基本就够用了。

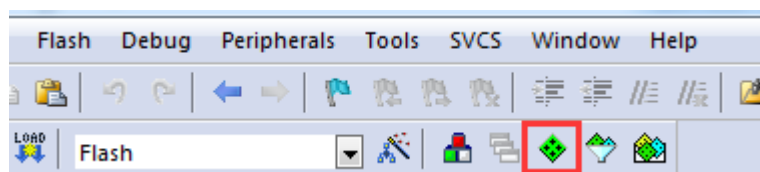
1.3 创建工程模板和注意事项

Event Recorder 工程的创建比较简单，这里分步为大家做个介绍。

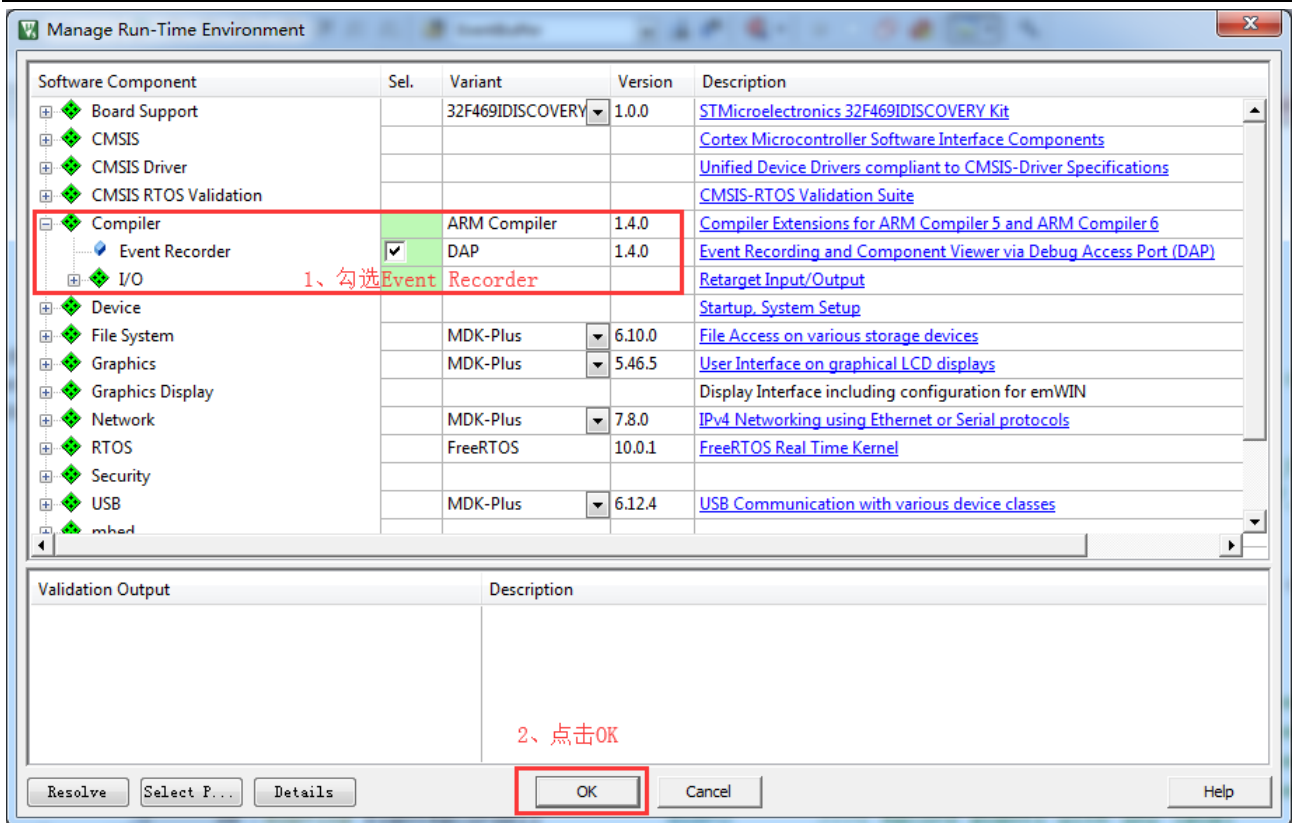
- ◆ 第 1 步:准备好一个使用 MDK5.25 创建的工程模板(或者其他任何 MDK4 或者低版本 MDK5 工程，只有能够使用 MDK5.25 打开并且编译正确即可)。



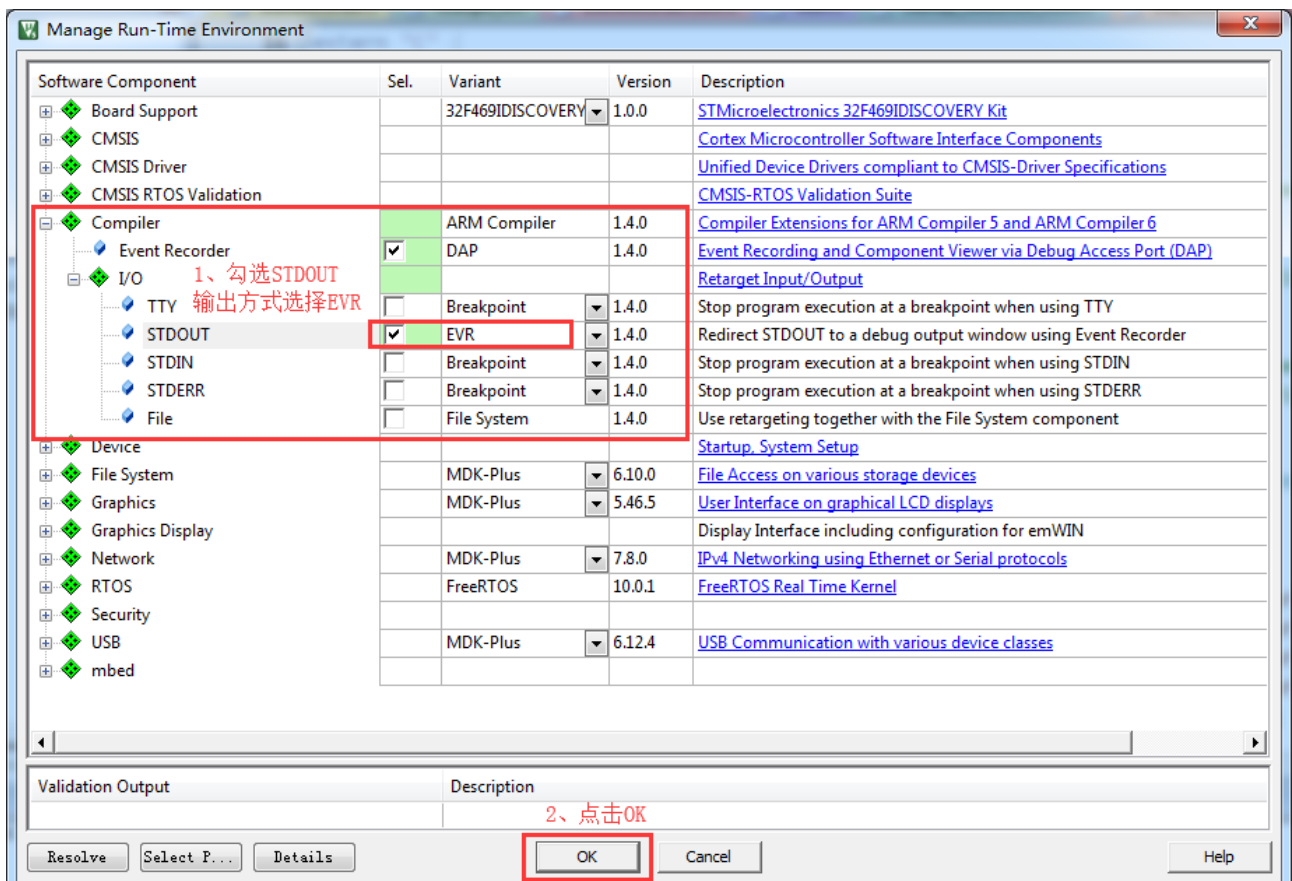
- ◆ 第 2 步: 安装 ARM_Compiler V1.4.0，详情见帖子：
<https://www.armbbs.cn/forum.php?mod=viewthread&tid=87175>。
- ◆ 第 3 步: 打开 MDK5.25 的 RTE 环境。



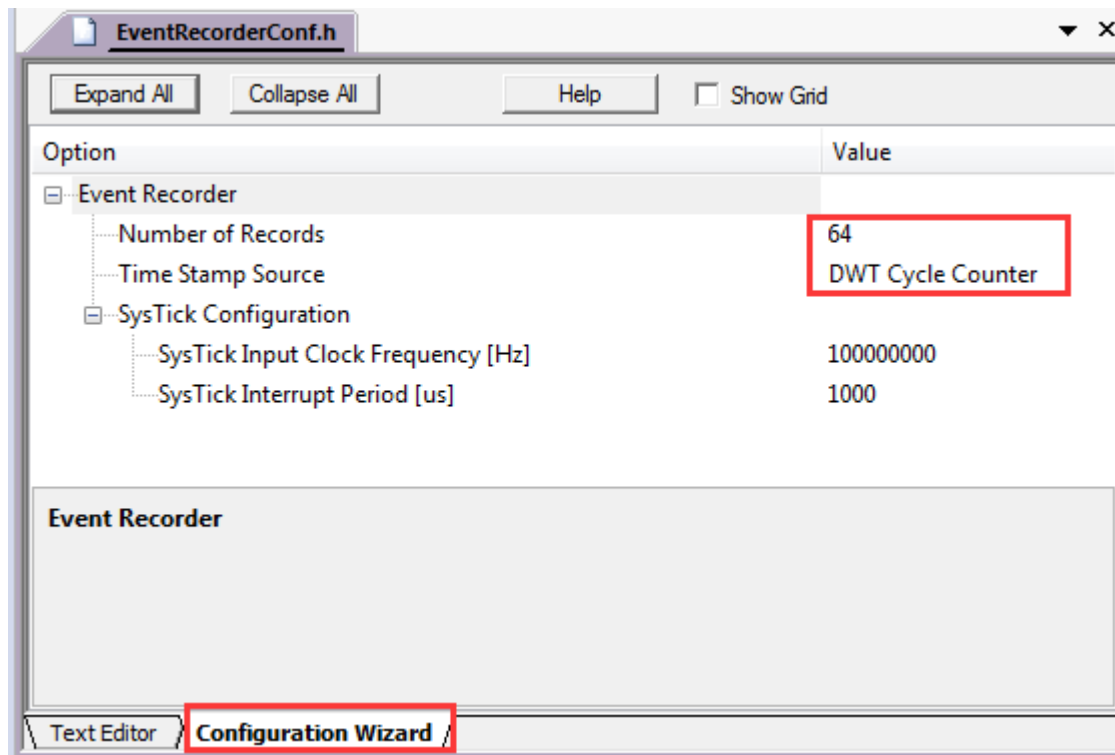
- ◆ 第 4 步: 通过 RTE 环境，为工程添加 Event Recorder 功能。



- ◆ 第 5 步：为了实现 printf 重定向，我们需要将 STDOUT 的输出方式改为 Event Recorder，即选项里面的 EVR。



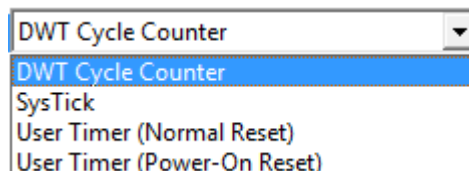
- ◆ 第 6 步：打开通过 RTE 环境为工程添加的文件 EventRecorderConf.h，配置如下：



这里主要设置方框里面的两个参数。

Number of Records：表示 Event Recorder 缓冲可以记录的消息条数。

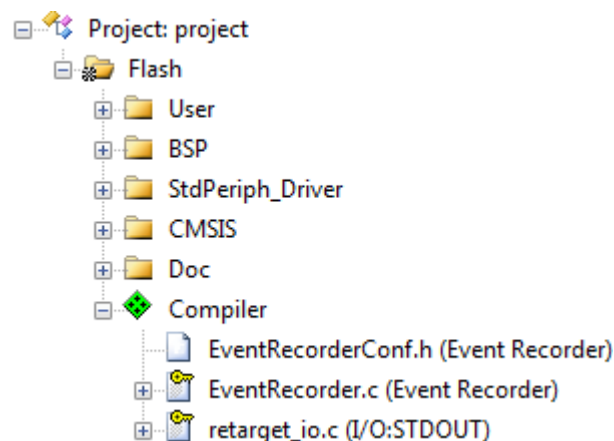
Time Stamp Source：表示时间戳来源，有如下四种可以选择，我们这里使用 DWT 时钟周期计数器。



由于选择的是 DWT，因此 EventRecorderCong.h 文件中的 SysTick Configuration 配置就不用管了。

=====

通过上面的 6 步就完成了 Event Recorder 功能的添加，效果如下：



添加完成后，还有非常重要的两点要特别注意：

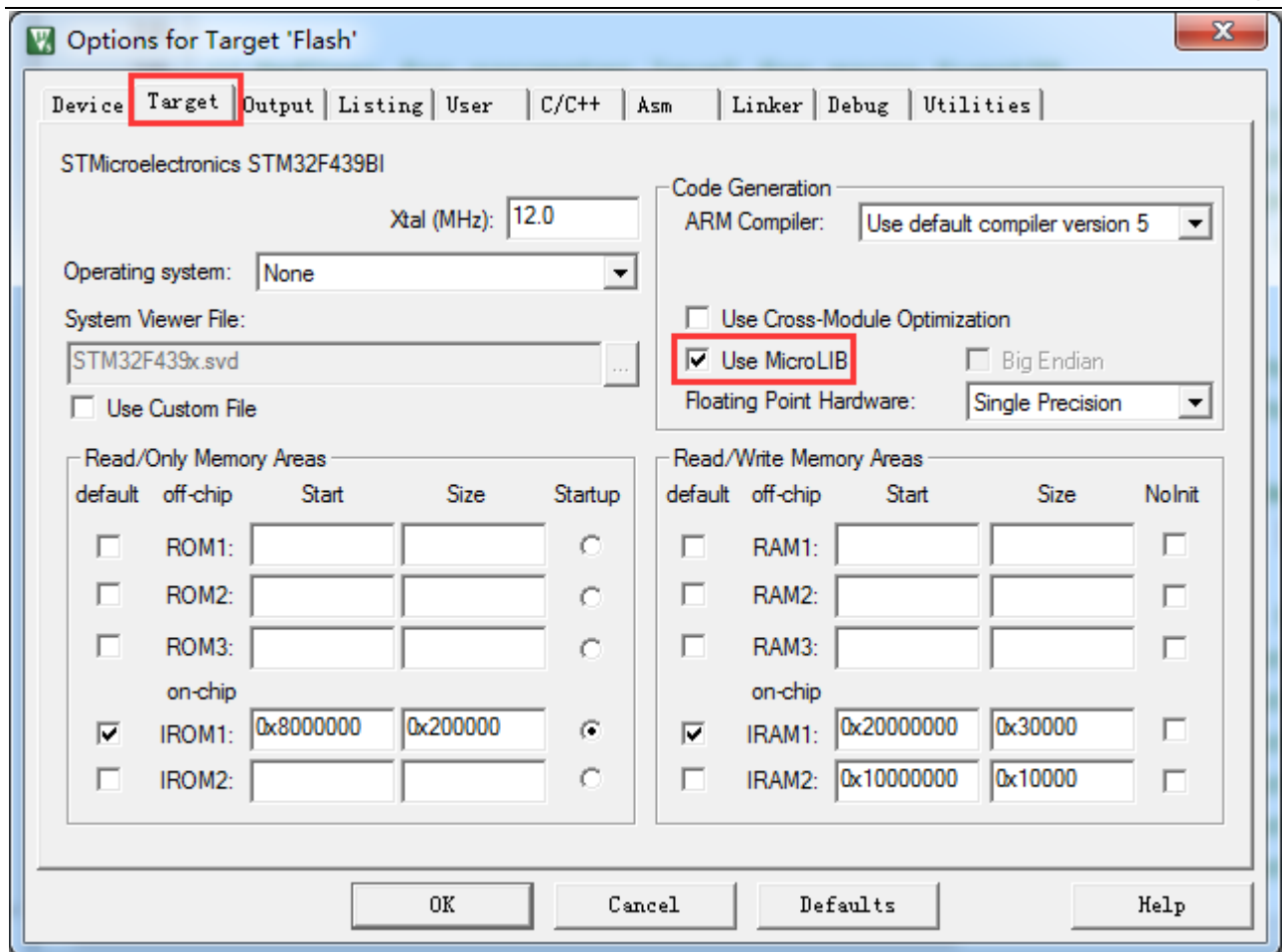
- ◆ 第 1 点：一定要使用当前最新的 CMSIS 软件包，当前是 V5.3.0。大家可以从这里下载：
<http://www.keil.com/dd2/pack/>。

ARM

➤ ARM V2M-MPS2 Board Support PACK for CoreLink SSE-200 -	BSP	DFP	1.0.3	↓
➤ Musca A1 Board Support PACK for CoreLink SSE-200 based - TrustZone	BSP	DFP	1.0.2	↓
➤ ARM V2M-MPS3 Board Support PACK for CoreLink SSE-200 -	BSP	DFP	1.0.0	↓
➤ ARM mbed Client for Cortex-M devices			1.1.0	↓
➤ ARM mbed Cryptographic and SSL/TLS library for Cortex-M devices	New		1.5.0	↓
➤ Bundle of FreeRTOS for Cortex-M and Cortex-A			10.0.1	↓
➤ CMSIS (Cortex Microcontroller Software Interface Standard)	BSP	DFP	5.3.0	↓
➤ CMSIS Drivers for external devices			2.2.0	↓

下载并导入到 MDK 后，需要大家更新自己现有工程 CMSIS 文件里面的头文件，可以直接将 CMSIS 文件夹中 Include 文件里面的所有文件全部删掉。替换为 MDK 安装目录如下路径里面的所有头文件：
ARM\PACK\ARM\CMSIS\5.3.0\CMSIS\Include。保证头文件都是最新的 5.3.0 版本。

- ◆ 第 2 点：由于使能了 printf 重定向，大家的工程里面一定不要再做重定向了，比如 fpuc, fgetc。另外当前选择了微库 MicroLib：



注意这两点后，就可以使用 Event Recorder 的功能了。

1.4 Event Recorder 事件记录的实现

Event Recorder 的使用也比较省事，这里也分步为大家进行说明：

◆ 第 1 步：初始化，仅需添加如下两行代码即可。

```
/* 初始化 EventRecorder 并开启 */
EventRecorderInitialize(EventRecordAll, 1U);
EventRecorderStart();
```

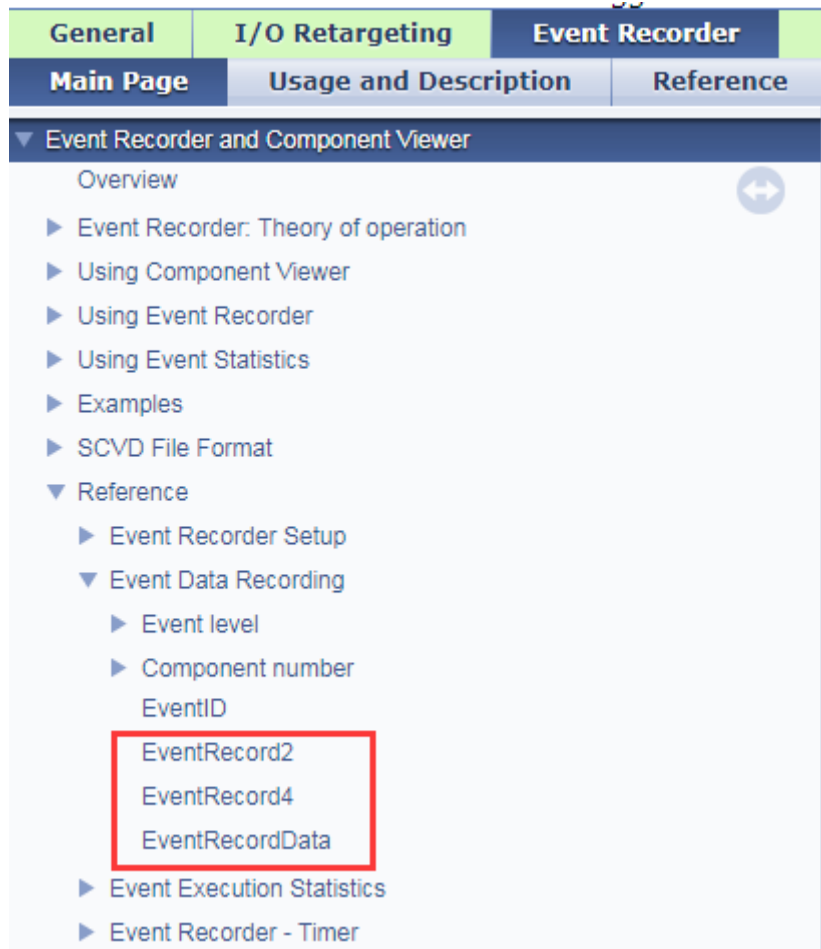
◆ 第 2 步：调用 Event Recorder 的 API 就可以使用了，主要有以下三个 API：

EventRecord2：可以发送两个 32 位数据。

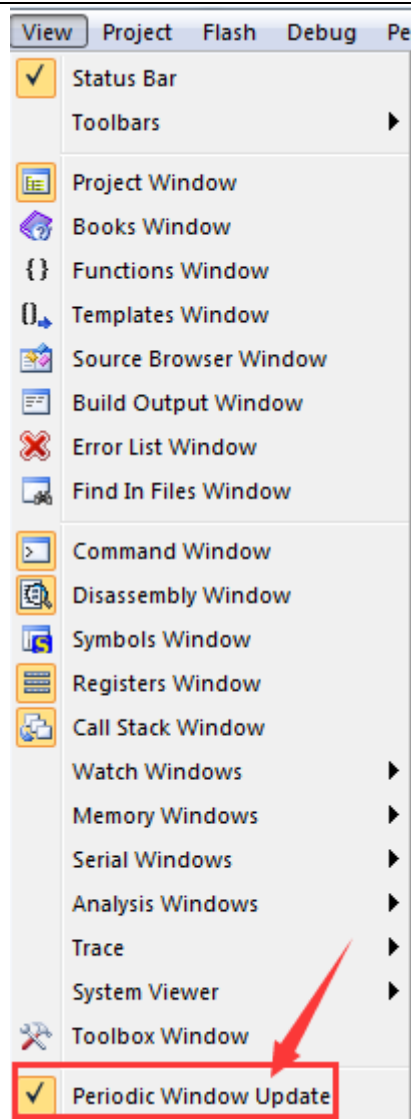
EventRecord4：可以发送四个 32 位数据。

EventRecordData：可以发送字符串。

显然这三个函数没有 printf 使用方便，所以对于这三个函数，大家做个简单的了解即可。教程配套例子里面有调用到这三个函数，可以操作熟悉下。这三个 API 的说明是在对应的 help 文档中，即 MDK 安装目录路径：/ARM/PACK/Keil/ARM_Compiler/1.4.0/Doc/General/html/index.html。



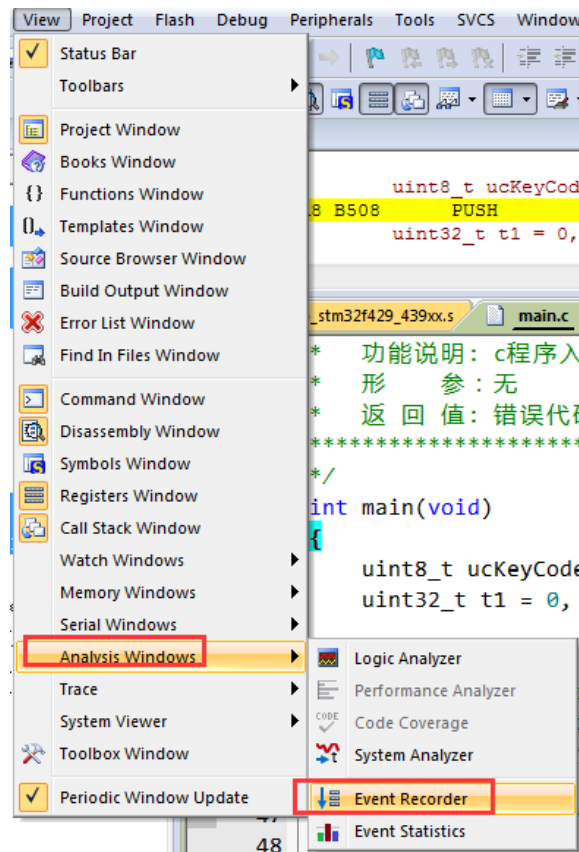
- ◆ 第 3 步：进入调试状态，选上周期更新：



点击全速运行：



然后将 Event Recorder 调试组件展示出来：



效果如下：

Event Recorder				
Enable Recorder: <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> Mark: <input type="text"/> All Operations <input type="text"/> Recording				
Event	Time (sec)	Component	Event Property	Value
0	228.51042139		id=0x0001	0x00000002,0x00000004
1	228.51042342		id=0x0002	0x00000002,0x00000004,0x00000006,0x00000008
2	228.51042642		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000
3	230.36042115		id=0x0001	0x00000003,0x00000006
4	230.36042329		id=0x0002	0x00000003,0x00000006,0x00000009,0x0000000C
5	230.36042629		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000
6	231.47042131		id=0x0001	0x00000004,0x00000008
7	231.47042333		id=0x0002	0x00000004,0x00000008,0x0000000C,0x00000010
8	231.47042634		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000
9	232.77042115		id=0x0001	0x00000005,0x0000000A
10	232.77042318		id=0x0002	0x00000005,0x0000000A,0x0000000F,0x00000014
11	232.77042619		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000

另外，这里有个知识点需要大家了解下，如果程序里面也调用了 Event Statistics 时间测量函数，那么也会在这个界面里面展示消息的，如何才能仅展示大家想看的功能呢？这就需要用到 Event Recorder 支持的筛选功能。使用这个功能需要大家先暂停全速运行，然后点击下面这个选项：

Event Recorder				
Enable Recorder: <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> Mark: <input type="text"/> All Operations <input type="text"/> Stopped with <input type="text"/>				
Event	Time (sec)	Component	Event Property	Value
0	228.51042139		id=0x0001	0x00000002,0x00000004
1	228.51042342		id=0x0002	0x00000002,0x00000004,0x00000006,0x00000008
2	228.51042642		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000
3	230.36042115		id=0x0001	0x00000003,0x00000006
4	230.36042329		id=0x0002	0x00000003,0x00000006,0x00000009,0x0000000C
5	230.36042629		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000
6	231.47042131		id=0x0001	0x00000004,0x00000008
7	231.47042333		id=0x0002	0x00000004,0x00000008,0x0000000C,0x00000010
8	231.47042634		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000
9	232.77042115		id=0x0001	0x00000005,0x0000000A
10	232.77042318		id=0x0002	0x00000005,0x0000000A,0x0000000F,0x00000014
11	232.77042619		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000

弹出的界面里面可以设置哪些选项显示，哪些选项不显示（勾上表示显示），我们这里取消 Event Statistics 的显示，设置完毕后记得点击 OK 按钮。

Show Event Levels

Record Component Events 取消

Event Statistics

EvStat : Start/Stop Statistics (0xEF)

STDIO

STDIO : C Standard I/O (0xFE)

Recorder Control

EvCtrl : Event Recorder Control (0xFF)

Unspecified Events

0x00 - 0x0F

0x10 - 0x1F

0x20 - 0x2F

0x30 - 0x3F

0x40 - 0x4F

0x50 - 0x5F

0x60 - 0x6F

0x70 - 0x7F

0x80 - 0x8F

0x90 - 0x9F

0xA0 - 0xAF

0xB0 - 0xBF

0xC0 - 0xCF

0xD0 - 0xDF

0xE0 - 0xEE

0xF0 - 0xFD

Error

API

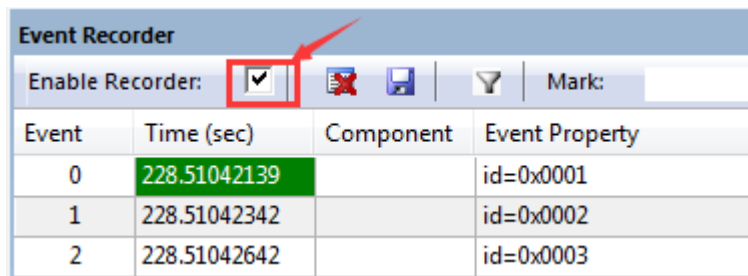
Op

Detail

OK

Cancel

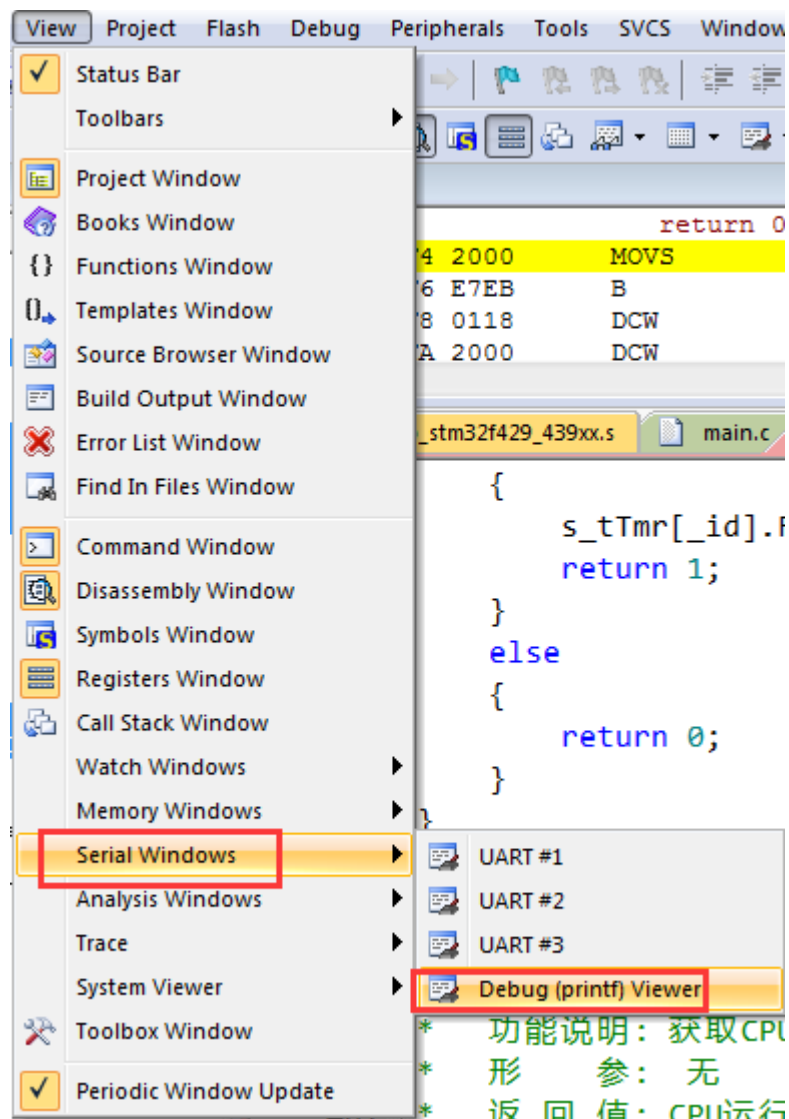
这就不展示 Event Statistics 的内容了。再次启动全速运行前，下面这个选项的对勾别忘了勾上。



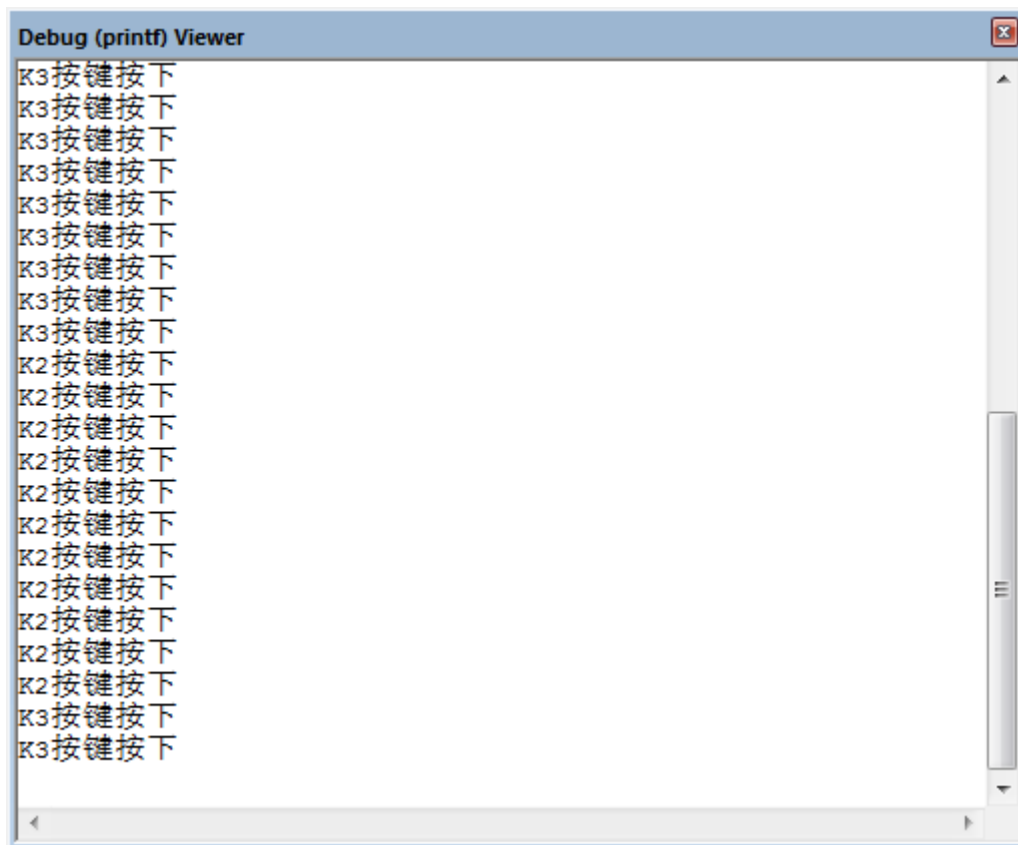
1.5 Event Recorder 实现 printf 重定向

实现 printf 输出需要用到 MDK 调试组件中的 Debug(printf) Viewer, 输出效果就跟大家使用串口调试软件一样，可以输出中文和英文。

MDK 的 printf 调试组件使用方法跟 1.4 小节中的说明一样，点击调试，选中周期运行，然后显示 Debug(printf) Viewer 调试组件：



效果如下：



另外，还有一个知识点需要给大家做个补充，使用 SWD 接口的 SWO 引脚也是可以做串口打印的，并且也是通过这个调试组件 Debug(printf) Viewer 进行输出。只是这种方式的性能没有 Event Viewer 强，而且要多占用一个 SWO 引脚。

关于 SWO 输出方式可以看此贴：<https://www.armbbs.cn/forum.php?mod=viewthread&tid=526>。

1.6 Event Statistics 时间测量功能的实现

时间测量功能简单易用，仅需一个起始函数，一个停止函数即可。当前支持 4 组，每组支持 16 路测量，也就是可以同时测量 64 路。

时间测量的 API 函数支持多任务和中断里面随意调用。

◆ 测量起始函数：EventStartG(slot) 或者 EventStartGv(slot, val1, val2)

- 函数中的字母 G 是表示分组 A, B, C, D，即实际调用函数为 EventStartA, EventStartB, EventStartC 和 EventStartD。
- 函数的第一个形参 slot 的范围是 0-15，也就是每个分组可以测试 16 路。
- 函数后面的两个形参 val1 和 val2 是 32 位变量，用户可以用这两个形参来传递变量数值给 Event Statistics 调试组件里面，方便图形化展示。简单的说，这两个变量仅仅起到一个传递变量数值

的作用。

◆ 测量停止函数：EventStopG (slot) 或者 EventStopGv (slot, val1, val2)

- 函数中的字母G是表示分组A,B,C,D,即实际调用函数为 EventStopA, EventStopB, EventStopC 和 EventStopD。
- 函数的第一个形参 slot 的范围是 0-15, 也就是每个分组可以测试 16 路。
- 函数后面的两个形参 val1 和 val2 是 32 位变量, 用户可以用这两个形参来传递变量数值给 Event Statistics 调试组件里面, 方便图形化展示。简单的说, 这两个变量仅仅起到一个传递变量数值的作用。

这里也分步为大家说明 Event Statistics 时间测量功能的使用方法。

◆ 第 1 步：初始化，仅需添加如下两行代码即可。

```
/* 初始化 EventRecorder 并开启 */  
EventRecorderInitialize(EventRecordAll, 1U);  
EventRecorderStart();
```

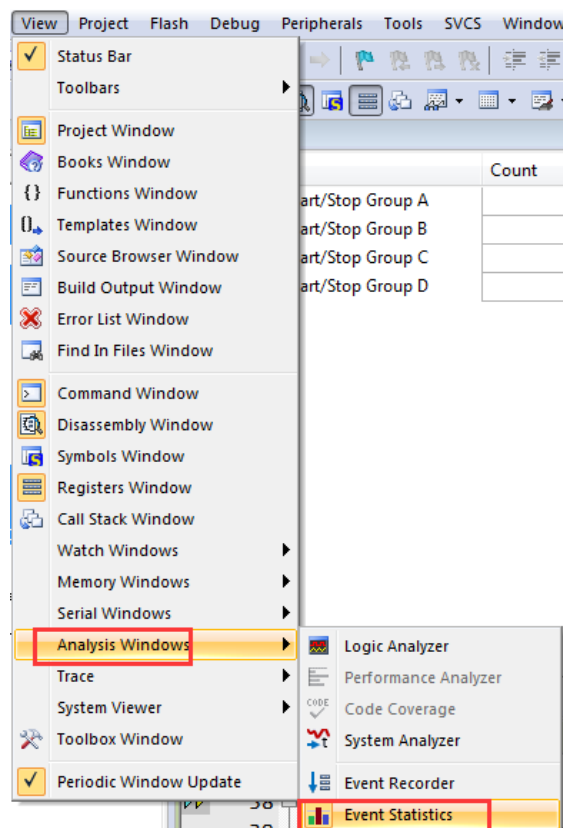
◆ 第 2 步：在要测量的代码前后加上起始和结束时间。

```
EventStartA(0);  
测量的代码部分  
EventStopA(0);
```

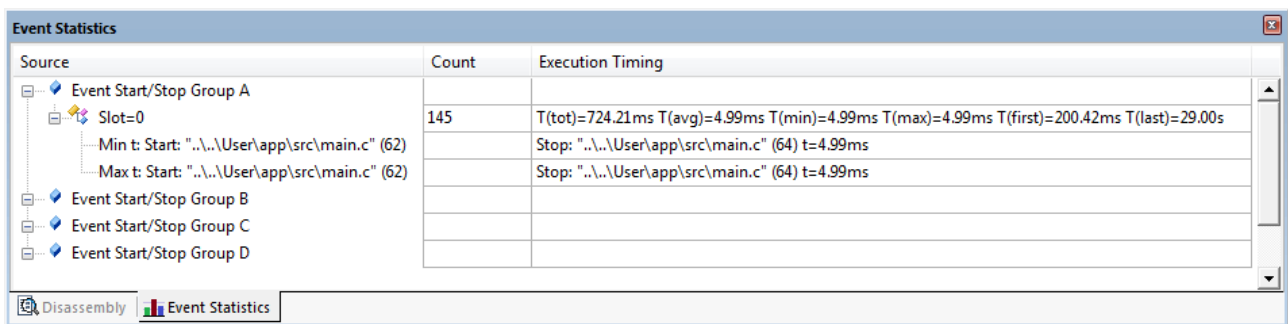
这里是用分组 A 的测量通道 0。

◆ 第 3 步：跟前面 1.4 小节讲解的一样，点击调试，选择周期更新选项，然后全速运行。

◆ 第 4 步：全速运行后，显示 Event Statistics 调试组件。

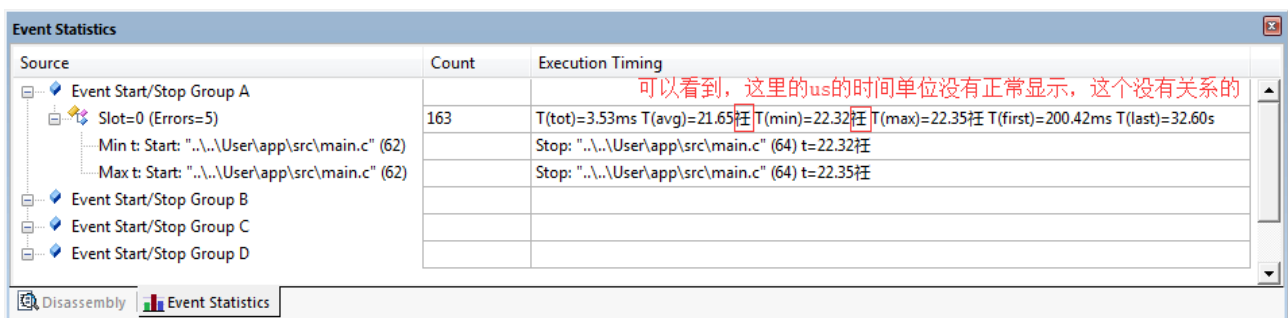


比如我这里简单的测试了一个 5ms 的延迟函数，效果如下（测量时间是动态更新的）：



Source	Count	Execution Timing
Event Start/Stop Group A		
Slot=0	145	T(tot)=724.21ms T(avg)=4.99ms T(min)=4.99ms T(max)=4.99ms T(first)=200.42ms T(last)=29.00s
Min t: Start: "..\..\User\app\src\main.c" (62)		Stop: "..\..\User\app\src\main.c" (64) t=4.99ms
Max t: Start: "..\..\User\app\src\main.c" (62)		Stop: "..\..\User\app\src\main.c" (64) t=4.99ms
Event Start/Stop Group B		
Event Start/Stop Group C		
Event Start/Stop Group D		

另外要注意一点，微妙的时间单位 us 可能无法正常显示，这个是没有关系的：



Source	Count	Execution Timing
Event Start/Stop Group A		
Slot=0 (Errors=5)	163	T(tot)=3.53ms T(avg)=21.65us T(min)=22.32us T(max)=22.35us T(first)=200.42ms T(last)=32.60s
Min t: Start: "..\..\User\app\src\main.c" (62)		Stop: "..\..\User\app\src\main.c" (64) t=22.32us
Max t: Start: "..\..\User\app\src\main.c" (62)		Stop: "..\..\User\app\src\main.c" (64) t=22.35us
Event Start/Stop Group B		
Event Start/Stop Group C		
Event Start/Stop Group D		

可以看到，这里的us的时间单位没有正常显示，这个没有关系的

1.7 Event Statistics 功耗测量功能的实现

当前仅 KEIL 自家的 ULINKplus 支持功耗测量功能，这款下载器不便宜，一套 5000 多，大家有个了解即可，我们这里就不做讲解了。

1.8 Event Recorder 对 RTX5 及其所有中间件的支持

后面做 RTX5 及其所有中间件的教程时会为大家做讲解，这里让大家看下效果：

◆ RTX5 组件和使用 Event Recorder 的效果：

RTX RTOS

Property	Value
System	
Kernel ID	RTX V5.3.0
Kernel State	osKernelRunning
Kernel Tick Count	19899
Kernel Tick Frequency	1000
Round Robin Tick Count	0
Round Robin Timeout	1
Global Dynamic Memory	Base: 0x20000000, Size: 10240, Used: 7208, Max used: 7208
Stack Overrun Check	Enabled
Stack Usage Watermark	Enabled
Default Thread Stack Size	1024
ISR FIFO Queue	Size: 16, Used: 0
Object Memory usage counters	
Threads	
Timers	
id: 0x20001B70	Running, Tick: 1
Mutexes	
id: 0x20001B98	Lock counter: 0
Event Flags	
id: 0x20001BC0	Flags: 0x00000000

System and Thread Viewer | RTX RTOS | Network | Event Recorder

Event Recorder

Enable Recorder: ☐ ☒ Mark: Errors

Event	Time (sec)	Component	Event Property	Value
317	7.31935551	RTX Kernel	KernelGetTickCount	count=7319
318	7.31935622	RTX Kernel	KernelGetTickCount	count=7319
319	7.31935699	RTX Mutex	MutexAcquire	mutex_id=0x20001B98, timeout=-1
320	7.31935760	RTX Mutex	MutexAcquired	mutex_id=0x20001B98, lock=1
321	7.31935830	RTX Thread	ThreadGetId	thread_id=0x20000010
322	7.31935891	RTX Thread	ThreadGetId	thread_id=0x20000010
323	7.31935974	RTX Kernel	KernelGetTickCount	count=7319
324	7.31936052	RTX Mutex	MutexRelease	mutex_id=0x20001B98
325	7.31936119	RTX Mutex	MutexReleased	mutex_id=0x20001B98, lock=0
326	7.31936202	RTX Mutex	MutexAcquire	mutex_id=0x20001B98, timeout=-1
327	7.43310043	RTX Mutex	MutexAcquire	mutex_id=0x20001B98, timeout=-1
328	7.43310111	RTX Mutex	MutexAcquired	mutex_id=0x20001B98, lock=1
329	7.43310173	RTX Thread	ThreadGetId	thread_id=0x20000010
330	7.43310237	RTX Thread	ThreadGetId	thread_id=0x20000010
331	7.43310296	RTX Mutex	MutexRelease	mutex_id=0x20001B98
332	7.43310355	RTX Mutex	MutexReleased	mutex_id=0x20001B98, lock=0
333	7.43310442	RTX Thread	ThreadDelay	ticks=5
334	7.43310519	RTX Thread	ThreadBlocked	thread_id=0x20000010, timeout=5
335	7.43310590	RTX Thread	ThreadSwitched	thread_id=0x20001068
336	7.43310742	RTX Thread	ThreadDelay	ticks=10
337	7.43310813	RTX Thread	ThreadBlocked	thread_id=0x20001068, timeout=10
338	7.43310862	RTX Thread	ThreadSwitched	thread_id=0x20002984
339	7.49637007	RTX Mutex	MutexReleased	mutex_id=0x20001B98, lock=0
340	7.49637101	RTX Mutex	MutexAcquire	mutex_id=0x20001B98, timeout=-1
341	7.49637170	RTX Mutex	MutexAcquired	mutex_id=0x20001B98, lock=1
342	7.66169407	RTX Thread	ThreadGetId	thread_id=0x20000010

Disassembly | System and Thread Viewer | Event Viewer | Event Recorder

◆ 网络调试组件效果展示:

Network

Property	Value
Library Version	IPv4/IPv6 Release
ETH interface	Link-Up
MAC address	1E-30-6C-A2-45-5E
IPv4 settings	
IP address	192.168.1.4
Network mask	255.255.255.0
Default gateway	192.168.1.1
Primary DNS server	192.168.1.1
Secondary DNS server	0.0.0.0
IPv6 settings	
UDP sockets	Used: 3, Available: 5
Socket 1	Opened
Socket 2	Opened
Socket 3	Opened
TCP sockets	Used: 1, Available: 6
Socket 1	Listen
Local Port	5900
Callback Function	bsd_cb_tcp
Options	Keep-alive: Off, Flow-ctrl: Off, Delay-ACK: Off

Disassembly | System and Thread Viewer | Event Viewer | Event Recorder | **Network**

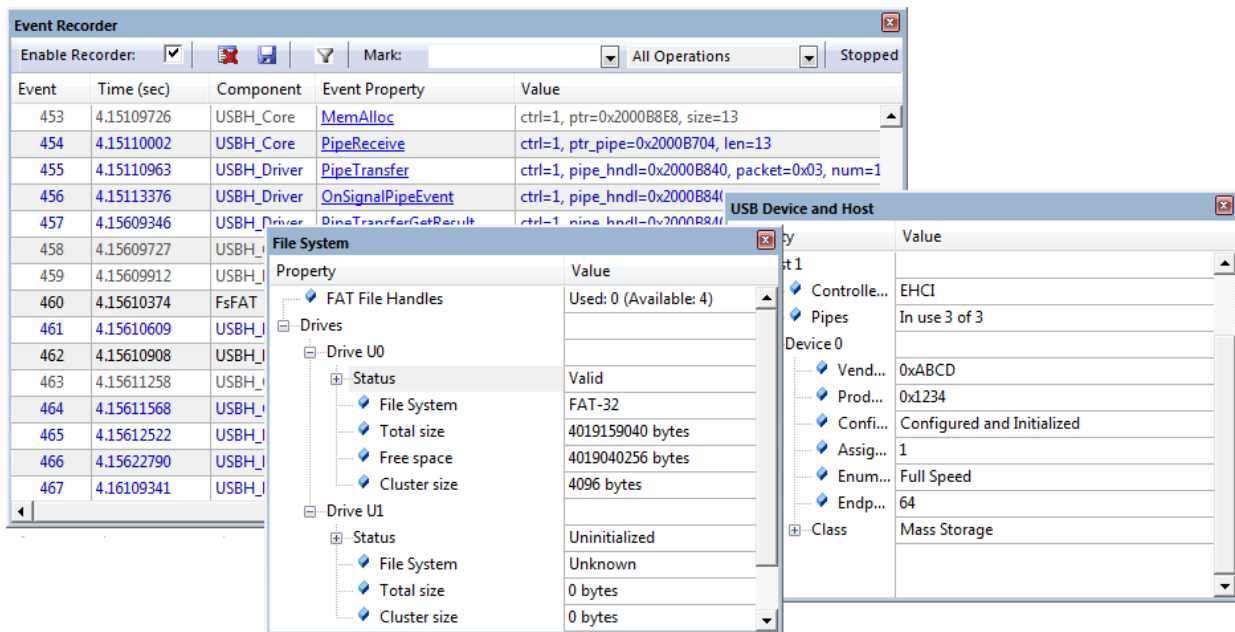
Event Recorder

Enable Recorder: ☐ Mark: Errors

Event	Time (sec)	Component	Event Property	Value
51	1.90092101	NetIP4	ShowFrameHeader	dst=255.255.255.255, src=0.0.0.0, proto=69, id=0x0001, frag=0x0000, l...
52	1.90092231	NetETH	SendFrame	len=342, ver=IPv4
53	1.90092287	NetETH	ShowFrameHeader	dst=FF-FF-FF-FF-FF-FF, src=1E-30-6C-A2-45-5E, proto=65535
54	1.90094019	NetMEM	FreeMemory	size=376 (used=384, blocks=1)
55	1.90094229	NetDHCP	NextState	state=REQUESTING
56	1.90094349	NetMEM	FreeMemory	size=384 (used=0, blocks=0)
57	1.90145066	NetMEM	AllocMemory	size=384 (used=384, blocks=1)
58	1.90147127	NetETH	ReceiveFrame	len=371
59	1.90147185	NetETH	ShowFrameHeader	dst=FF-FF-FF-FF-FF-FF, src=8-10-78-73-78-CB, proto=65535
60	1.90147305	NetIP4	ReceiveFrame	len=357
61	1.90147344	NetIP4	ShowFrameHeader	dst=255.255.255.255, src=192.168.1.1, proto=69, id=0x11C7, frag=0x0...
62	1.90147565	NetUDP	ReceiveFrame	len=337, ver=IPv4
63	1.90147611	NetUDP	ShowFrameHeader	dst_port=68, src_port=67, cksum=0xF103, len=337
64	1.90147695	NetUDP	MappedToSocket	sock=2
65	1.90147760	NetDHCP	ReceiveFrame	server=192.168.1.1, len=329
66	1.90147829	NetDHCP	ClientState	state=REQUESTING
67	1.90147899	NetDHCP	DhcpAckReceived	
68	1.90147940	NetDHCP	ShowAssignedAddress	ip=192.168.1.4
69	1.90148049	NetDHCP	ShowNetMask	mask=255.255.255.0
70	1.90148129	NetDHCP	ShowLeaseTime	time=86400
71	1.90148194	NetDHCP	ShowDnsServers	pri=192.168.1.1, sec=0.0.0.0
72	1.90148253	NetDHCP	ShowGatewayAddress	gateway=192.168.1.1
73	1.90148327	NetDHCP	NextState	state=BOUND

System and Thread Viewer | Event Viewer | Event Recorder

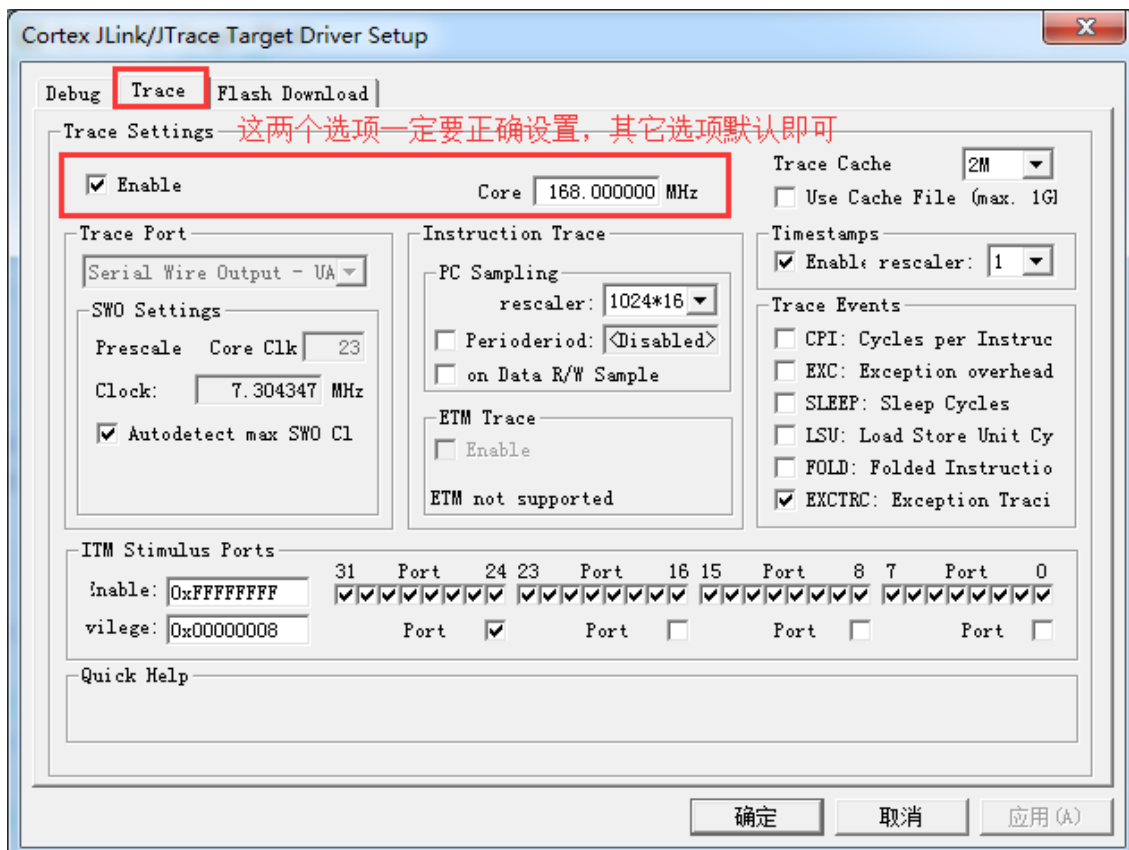
◆ 文件系统和 USB 协议栈的效果展示：



1.9 JLINK 配置说明

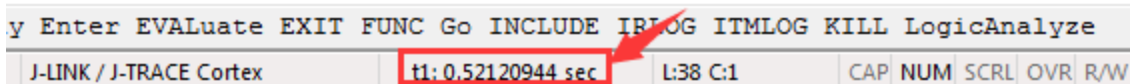
为了帮助大家更好的使用 JLINK，这里将 JLINK 配置中关键的几个地方做个说明。

- ◆ 下面这个地方最重要，一定要正确设置当前系统工作的主频，如果不正确，会导致 Event Statistics 的时间统计不正确。

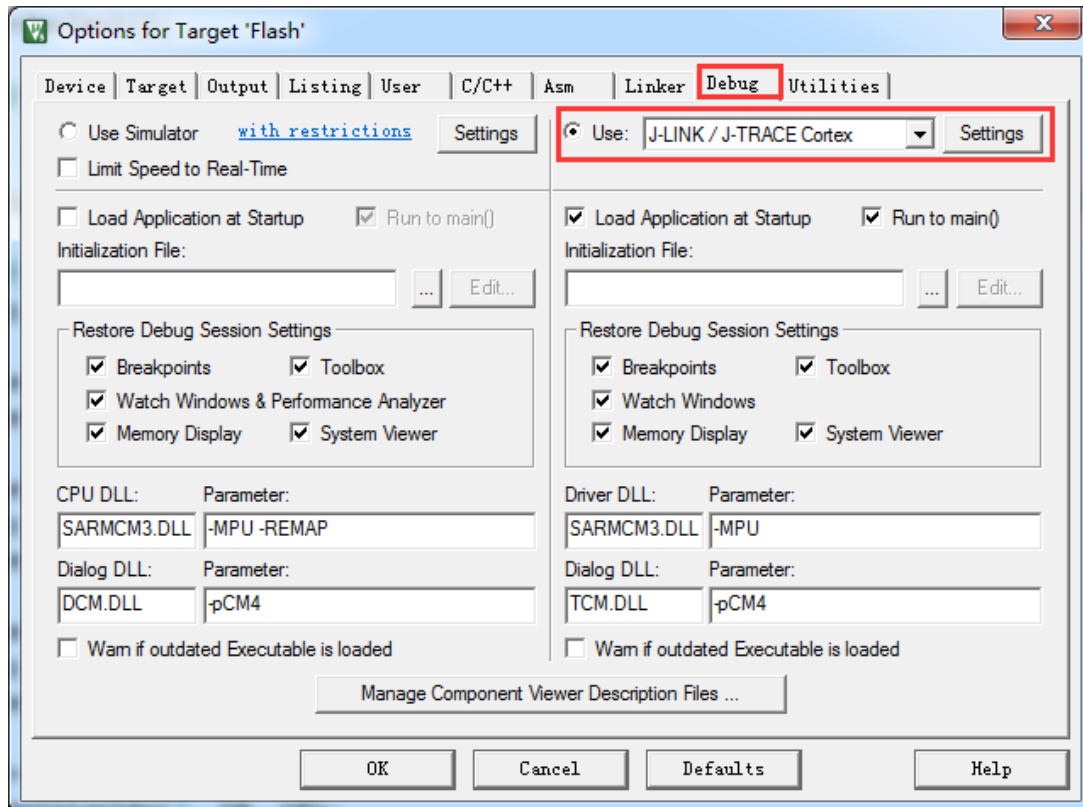


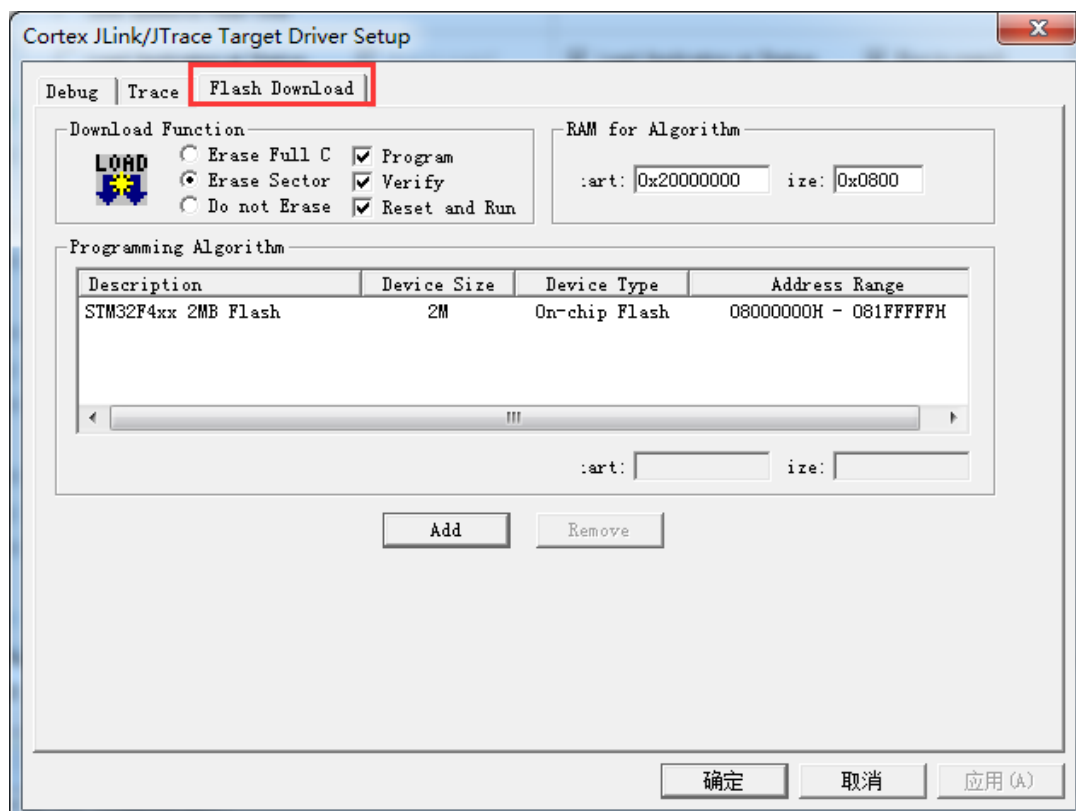
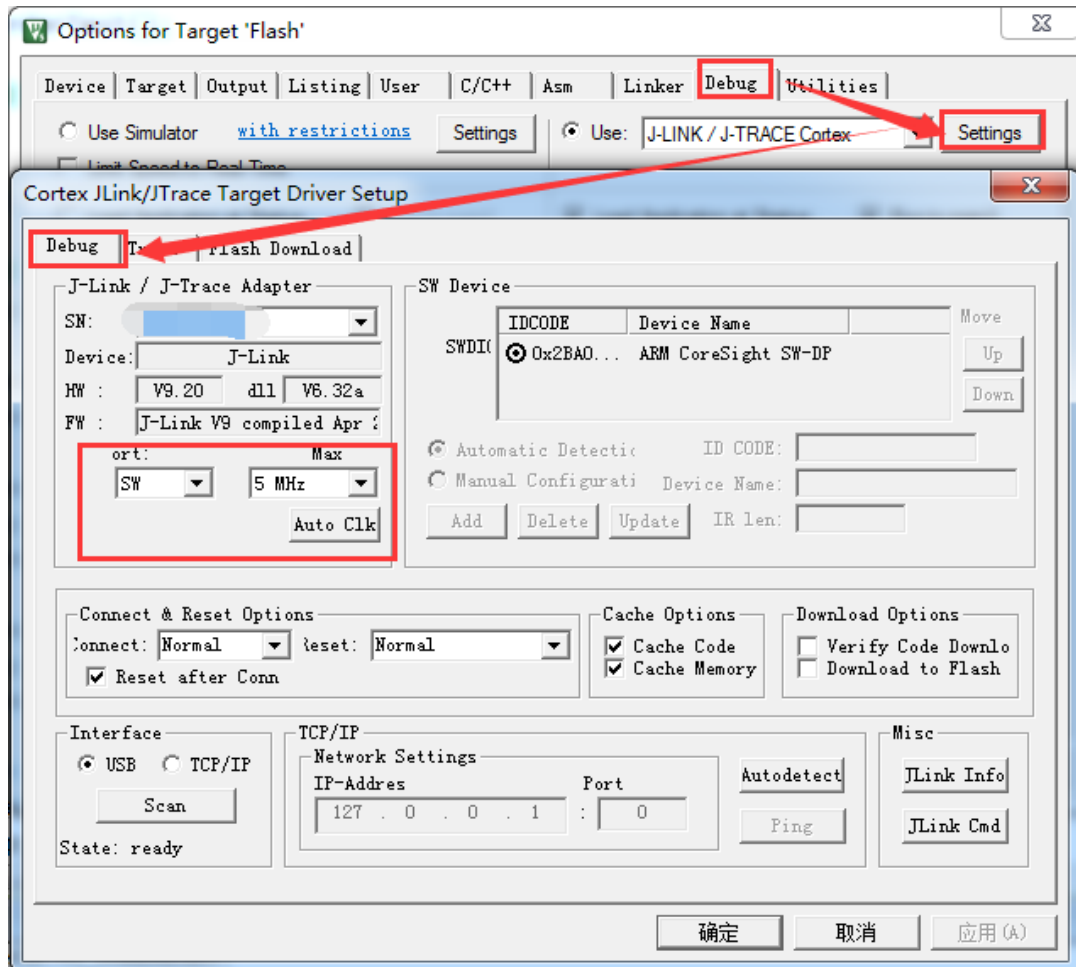
注：如果大家调试状态弹出 SWD 配置时钟超出范围的问题，可以考虑将上面截图中的 Enable 选项的对勾取消掉即可，但内核时钟一定要修改为芯片的主频：

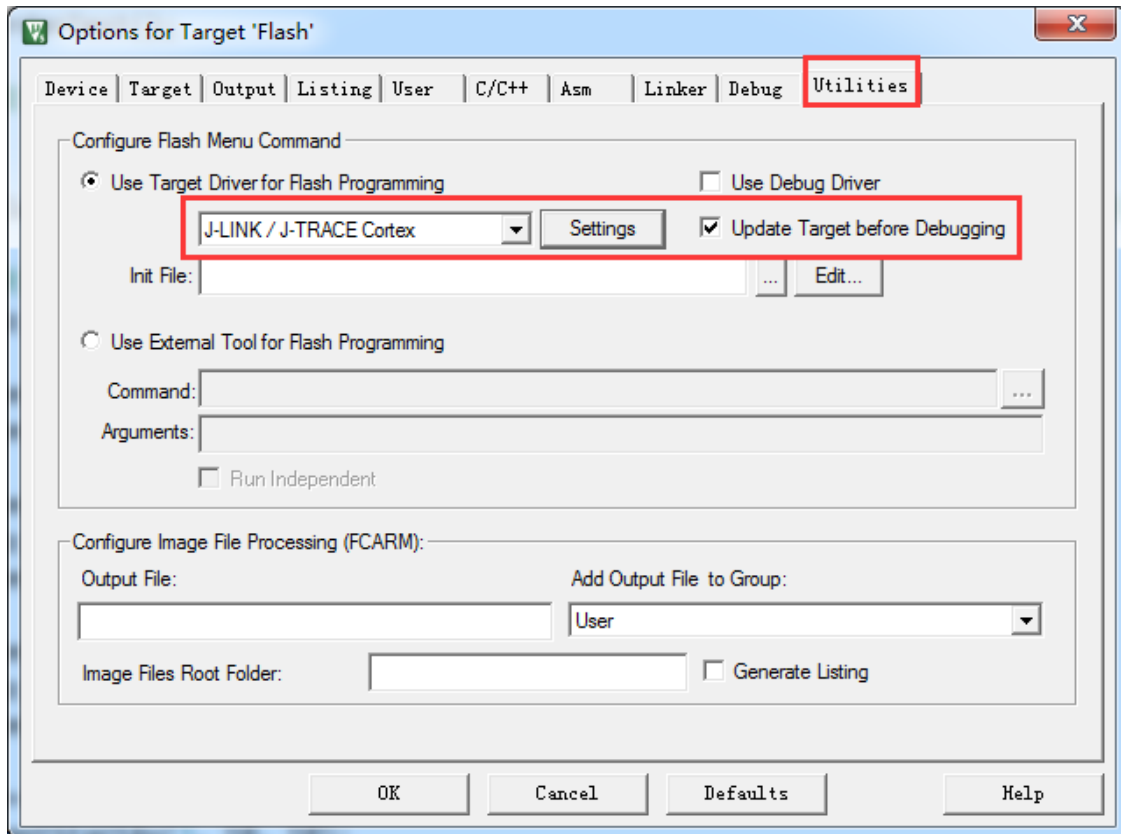
另外，进入调试状态后，右下角的时间是否正常更新都没有关系：



◆ 其它选项配置如下（只要大家的工程能够正常调试，配置就是没问题的）：



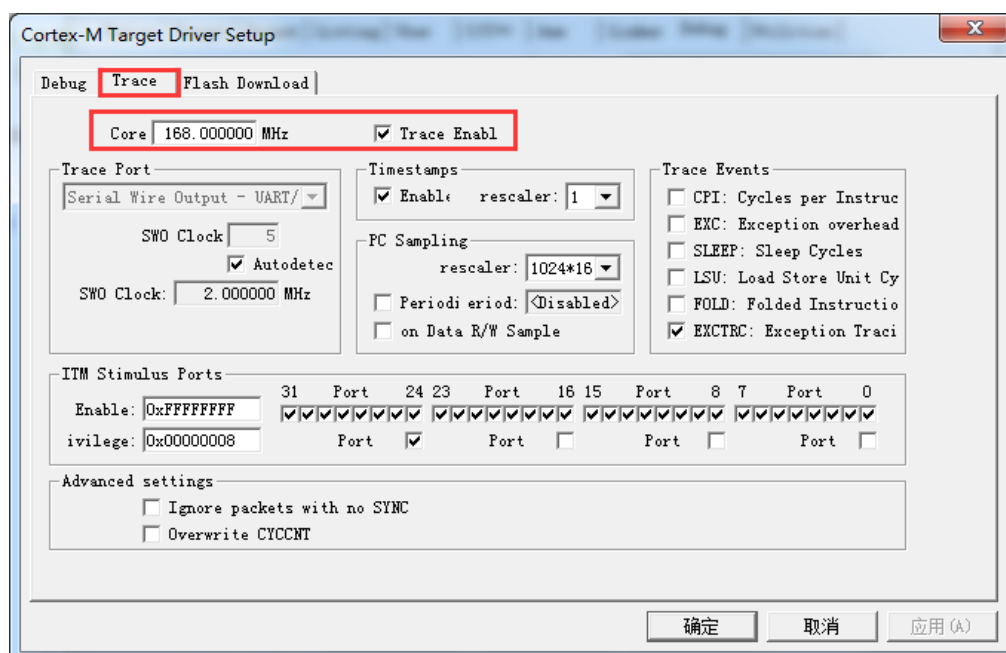




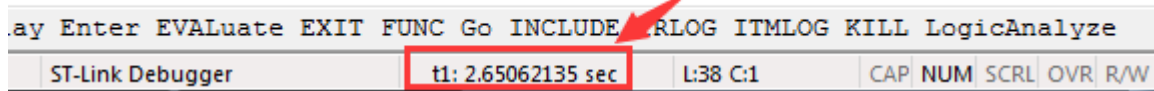
1.10 STLINK 配置说明

为了帮助大家更好的使用 STLINK，这里将 STLINK 配置中关键的几个地方做个说明。

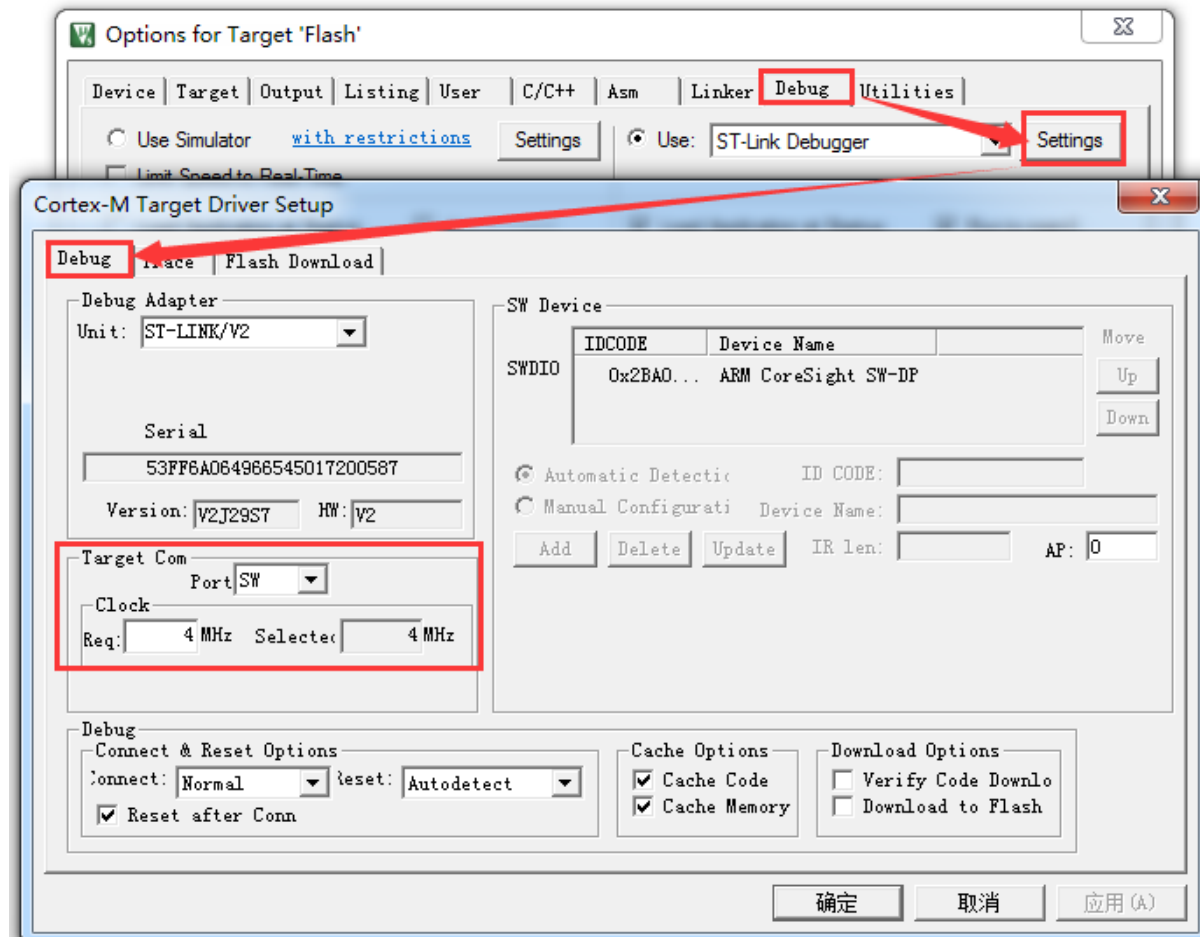
- ◆ 下面这个地方最重要，一定要正确设置当前系统工作的主频，如果不正确，会导致 Event Statistics 的时间统计是不正确的。

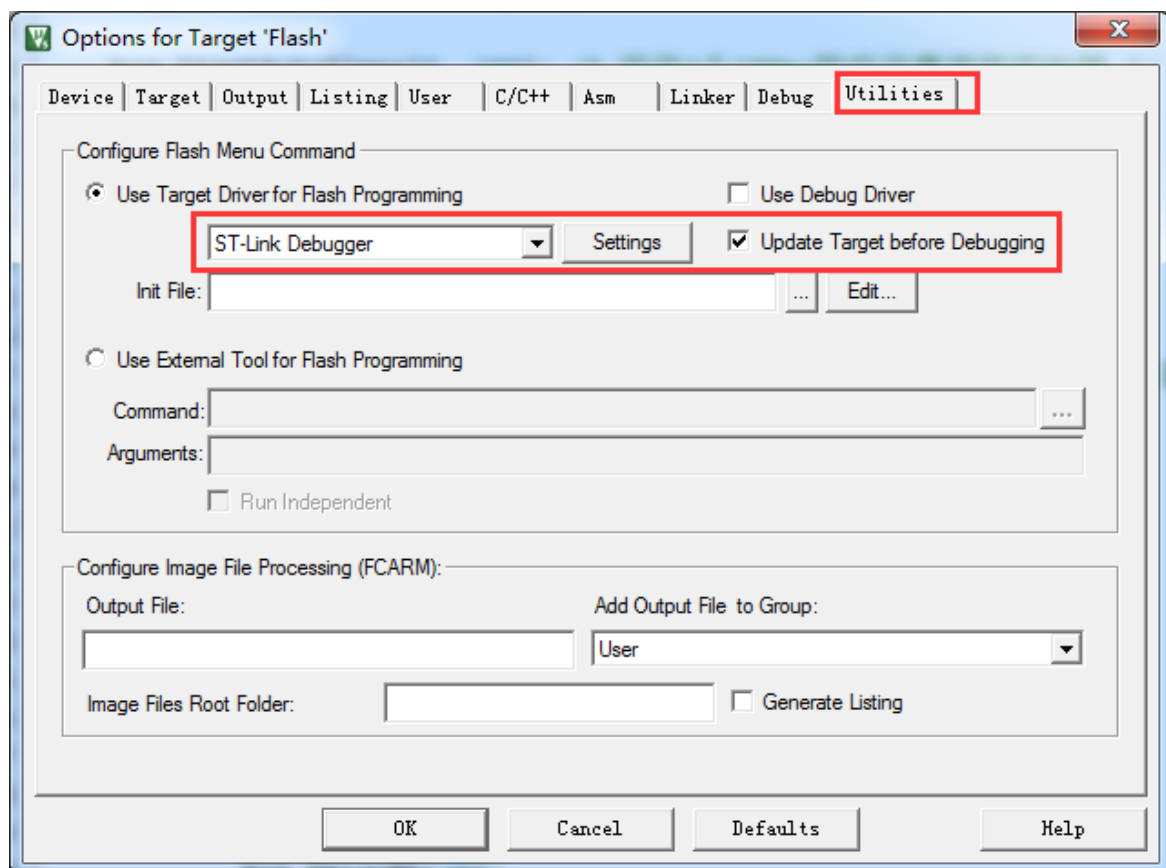
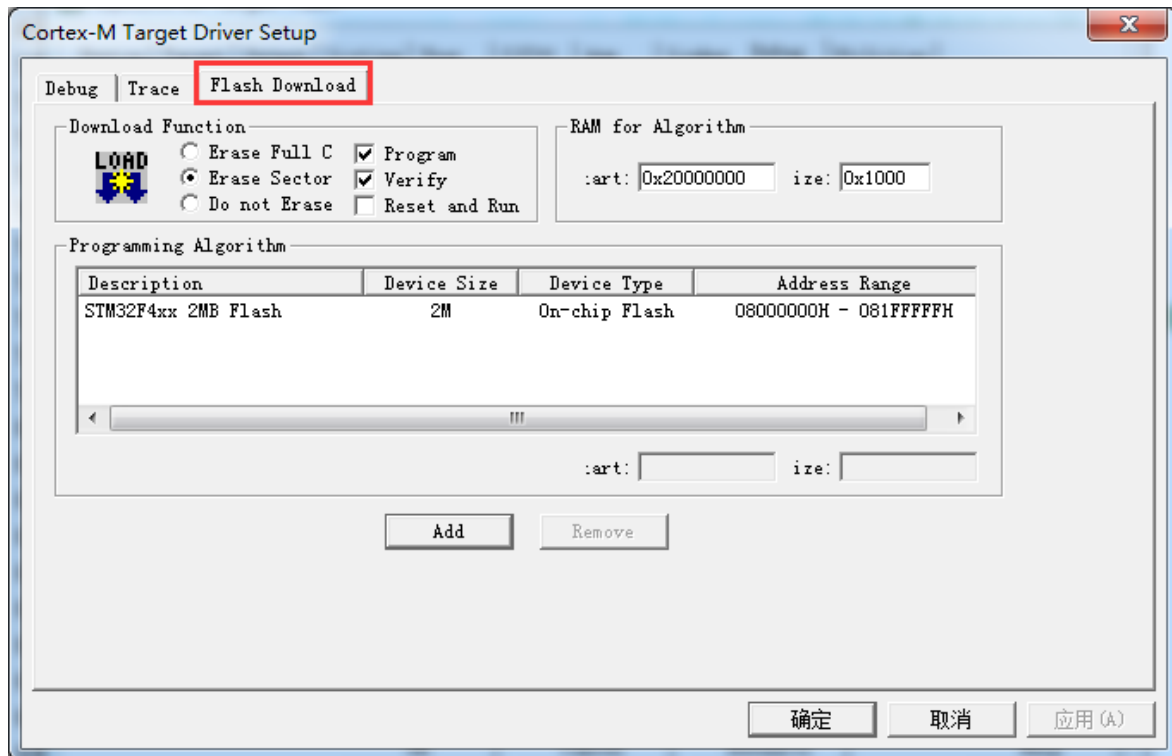


另外注意，进入调试状态后，右下角的时间是否正常更新都没有关系：



◆ 其它选项配置如下（只要大家的工程能够正常调试，配置就是没问题的）：

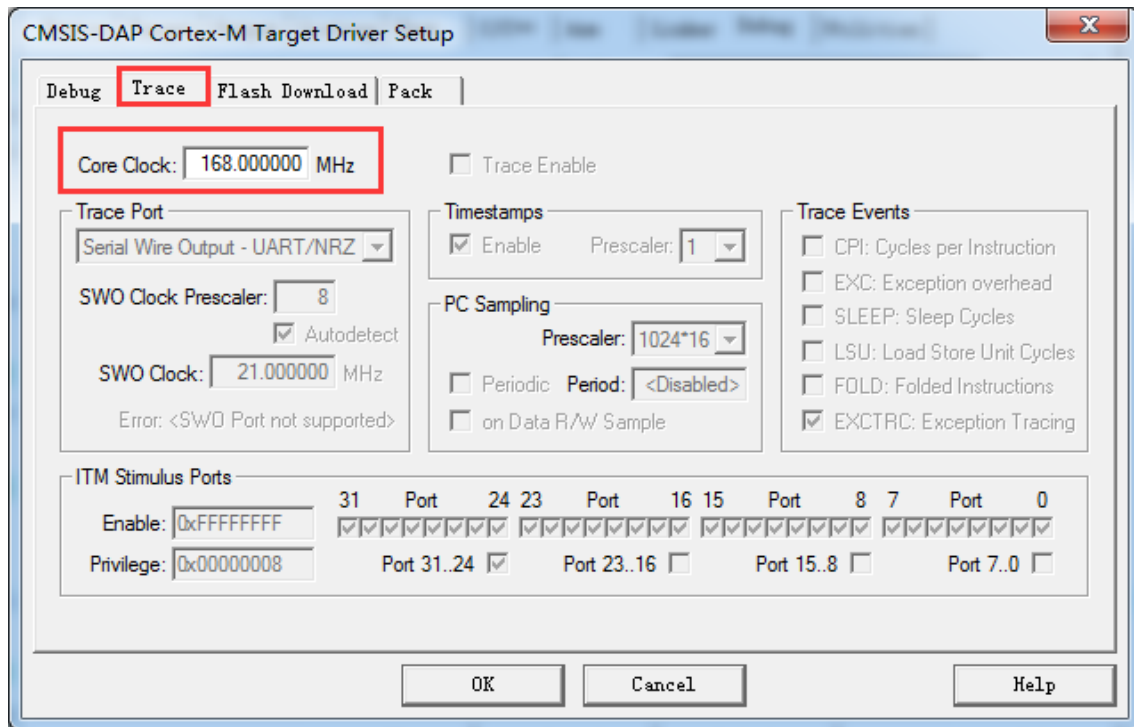




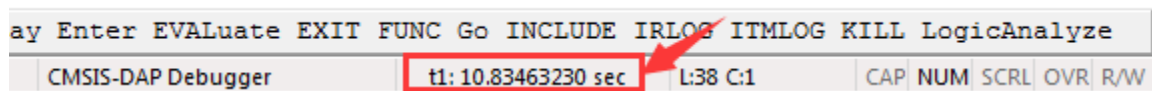
1.11 CMSIS-DAP 配置说明

为了帮助大家更好的使用 CMSIS-DAP，这里将 CMSIS-DAP 配置中关键的几个地方做个说明。

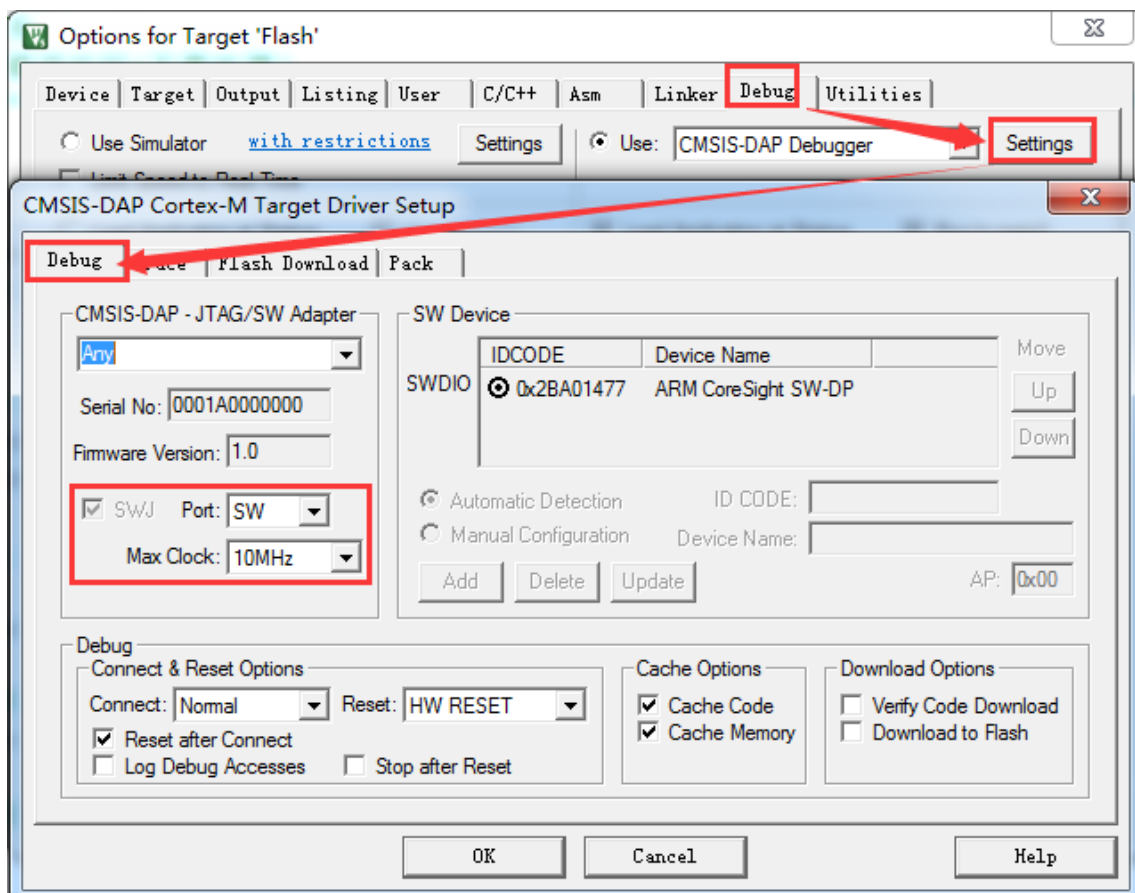
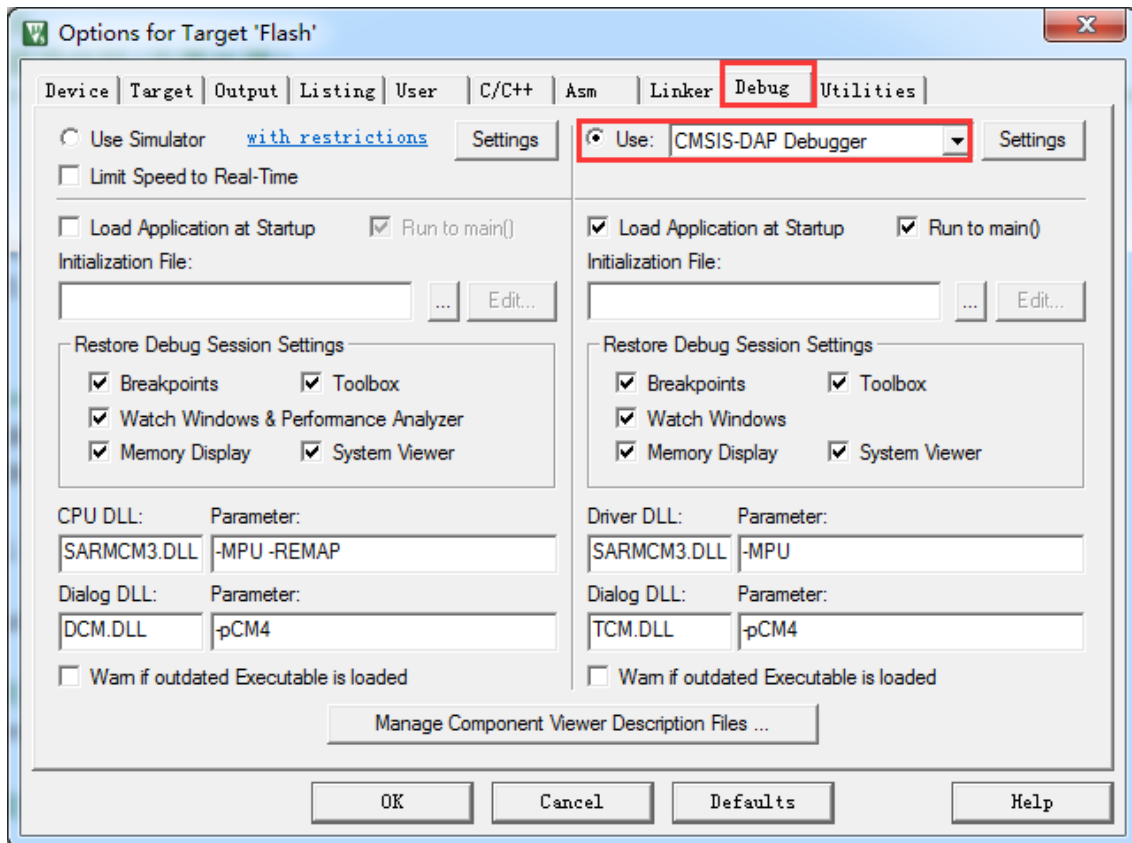
- ◆ 下面这个地方最重要，一定要正确设置当前系统工作的主频，如果不正确，会导致 Event Statistics 的时间统计不正确。

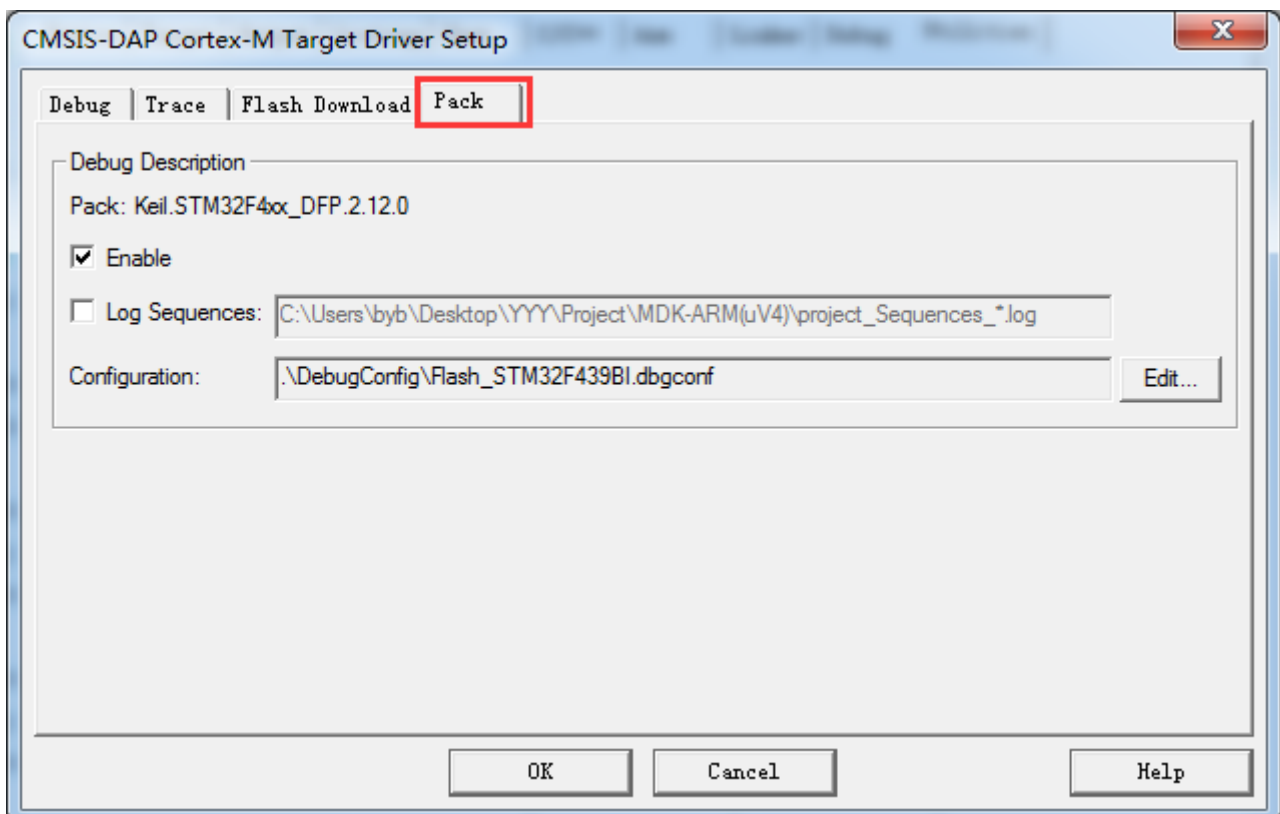
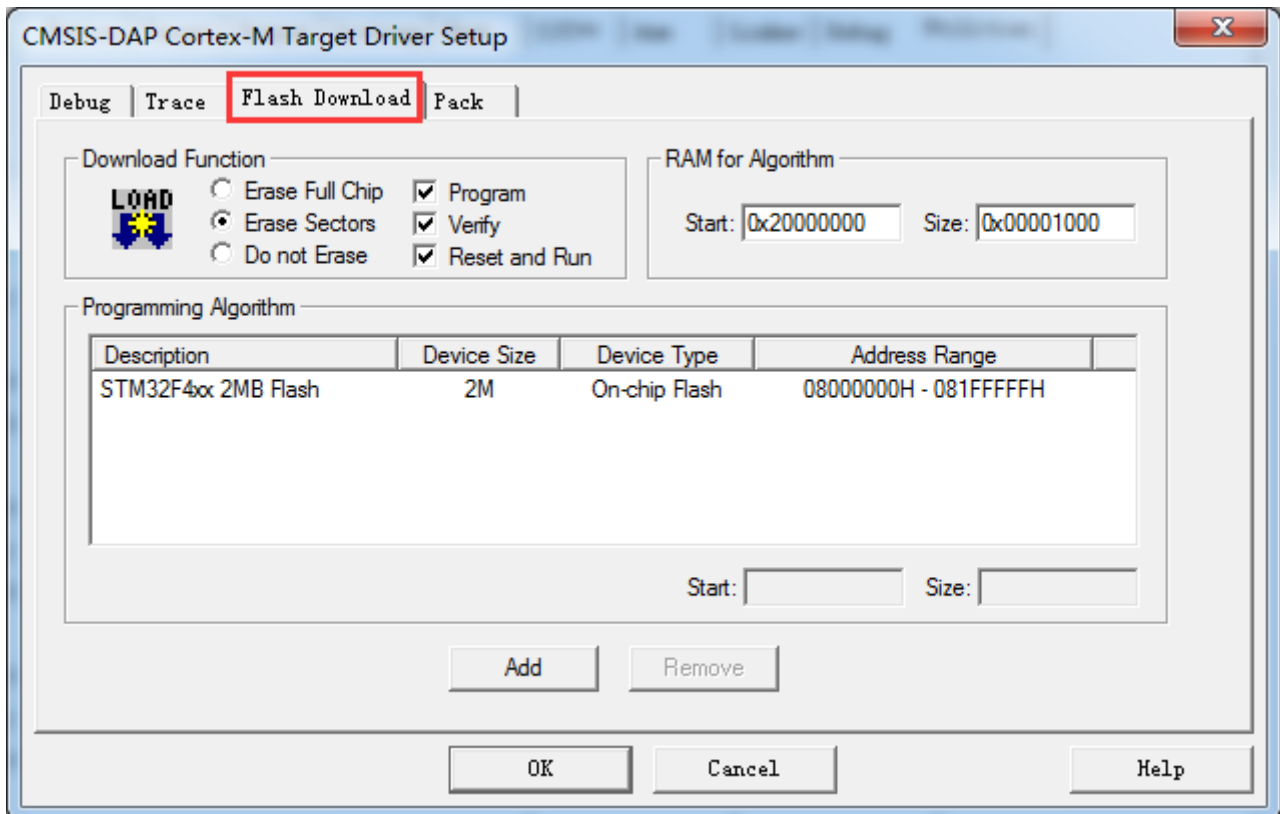


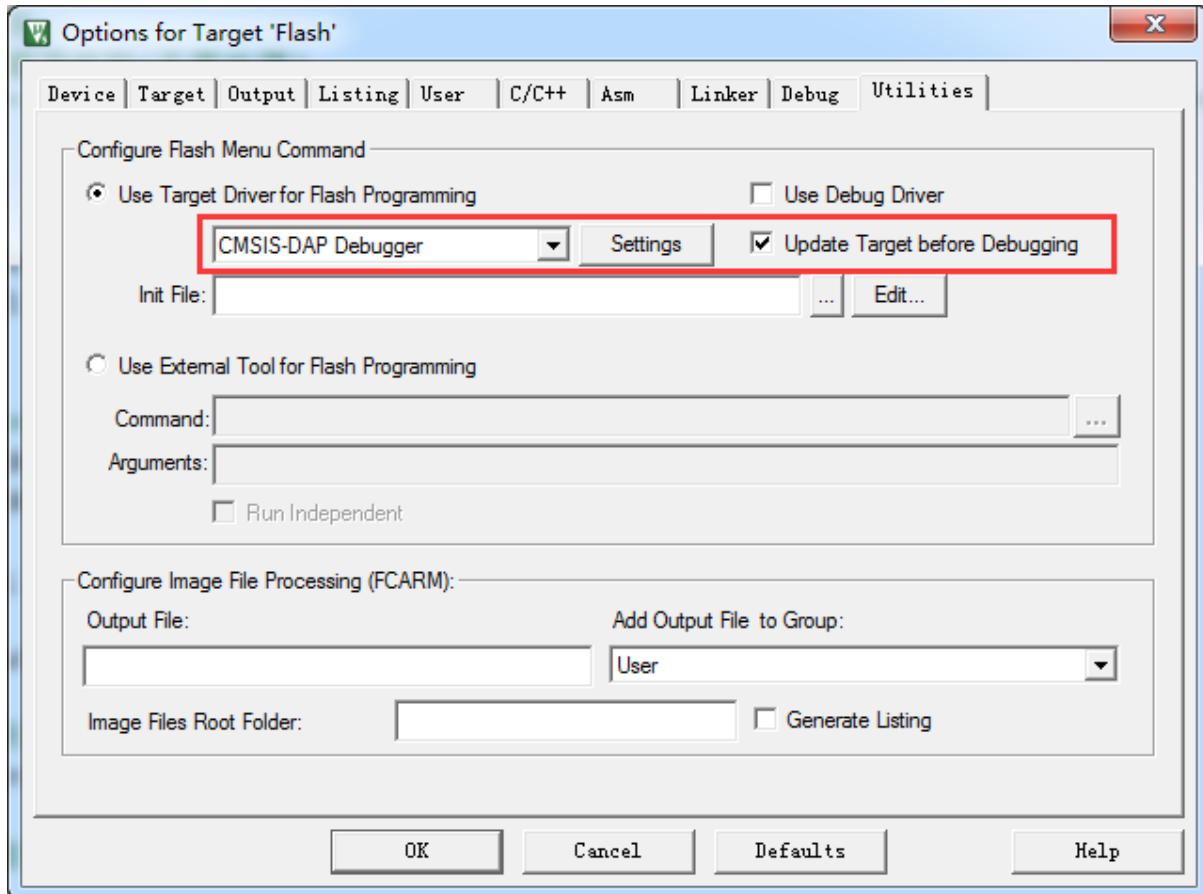
另外注意，进入调试状态后，右下角的时间是否正常更新都没有关系：



- ◆ 其它选项配置如下（只要大家的工程能够正常调试，配置就是没问题的）：







1.12 ULINK 配置说明

由于手头没有 ULINK，这里就不做讲解了。如果大家需要相关配置，按照前面小节三款 LINK 的配置照葫芦画瓢搞一下即可，或者在 MDK 安装目录的路径 ARM\Hlp 下有对应的文档说明：

- ? ulink2.chm
- ? ulinkme.chm
- ? ulinkplus.chm
- ? ulinkpro.chm

1.13 Event Recorder 狂暴模式

这个专门录制过一期视频教程，大家有需要可以看看：

视频教程第 11 期：STM32H7 的 GPIO 实战，深化非阻塞编程思想，移植驱动到全新器件上，开启 Event Recorder 狂暴模式：<https://www.armbbs.cn/forum.php?mod=viewthread&tid=111702>

1.14 配套例子

专题教程配套了三个例子，仅 MDK 版本。

- ◆ STM32-V4 开发板配套的例子是：V4-终极调试工具 Event Recorder。
- ◆ STM32-V5 开发板配套的例子是：V5-终极调试工具 Event Recorder。
- ◆ STM32-V6 开发板配套的例子是：V6-终极调试工具 Event Recorder。

具体代码实现也比较简单,以 V6 开发板为例,定义一个 TIM6 的中断,中断频率是 500Hz,通过 Event Statistics 测量中断的执行频率。代码如下:

```
#include "bsp.h"
#include "EventRecorder.h"

/* 定时器频率, 500Hz */
#define timerINTERRUPT_FREQUENCY    500

/* 中断优先级 */
#define timerHIGHEST_PRIORITY       10

/*
*****
*   函数名: vEventRecorderTest
*   功能说明: 创建定时器
*   形参: 无
*   返回值: 无
*****
*/
void vEventRecorderTest(void)
{
    bsp_SetTIMforInt(TIM6, timerINTERRUPT_FREQUENCY, timerHIGHEST_PRIORITY, 0);
    EventStartB(0);
}

/*
*****
*   函数名: TIM6_DAC_IRQHandler
*   功能说明: TIM6 中断服务程序。
*   形参: 无
*   返回值: 无
*****
*/
void TIM6_DAC_IRQHandler( void )
{
    if(TIM_GetITStatus(TIM6, TIM_IT_Update) != RESET)
    {
        EventStopB(0);
        EventStartB(0);
        TIM_ClearITPendingBit(TIM6, TIM_IT_Update);
    }
}
```


效果如下，测量的平均频率是 1.98ms，与我们设计的 500Hz 基本符合：

Event Statistics		
Source	Count	Execution Timing
Event Start/Stop Group A		
Event Start/Stop Group B		
Slot=0 (Errors=220)	17307 (+1)	T(tot)=34.34s T(avg)=1.98ms T(min)=1.99ms T(max)=8.00ms T(first)=429.38ms T(last)=34.61s
Min t: Start: "..\..\User\app\src\EventRecorderTest.c" (40)		Stop: "..\..\User\app\src\EventRecorderTest.c" (55) t=1.99ms
Max t: Start: "..\..\User\app\src\EventRecorderTest.c" (56)		Stop: "..\..\User\app\src\EventRecorderTest.c" (55) t=8.00ms
Event Start/Stop Group C		
Event Start/Stop Group D		

应用程序的设计如下：

```
#include "bsp.h"          /* 底层硬件驱动 */
#include "EventRecorder.h"

/*
*****
*                                     函数和变量
*****
*/
extern void vEventRecorderTest(void);
uint8_t s_ucBuf[10] = "armfly";

/*
*****
* 函数名: main
* 功能说明: c 程序入口
* 形参: 无
* 返回值: 错误代码(无需处理)
*****
*/
int main(void)
{
    uint8_t ucKeyCode;      /* 按键代码 */
    uint32_t t0 = 0, t1 = 0, t2 = 0, t3 = 0, t4 = 0;

    /* 初始化 EventRecorder 并开启 */
    EventRecorderInitialize(EventRecordAll, 1U);
    EventRecorderStart();

    bsp_Init();             /* 硬件初始化 */

    bsp_StartAutoTimer(0, 200); /* 启动 1 个 200ms 的自动重装的定时器 */

    /* 测量中断周期 */
    vEventRecorderTest();

    /* 进入主程序循环体 */
    while (1)
    {
        bsp_Idle();         /* 这个函数在 bsp.c 文件。用户可以修改这个函数实现 CPU 休眠和喂狗 */
    }
}
```

```
/* 判断定时器超时时间 */
if (bsp_CheckTimer(0))
{
    EventStartA(0);
    EventStopA(0);

    EventStartA(1);
    bsp_DelayMS(5);
    EventStopA(1);

    EventStartA(2);
    bsp_DelayMS(30);
    EventStopA(2);

    t0++;
    EventStartAv(3, t0, t0);
    bsp_DelayMS(30);
    EventStopAv(3, t0, t0);
}

/* 按键滤波和检测由后台 systick 中断服务程序实现，我们只需要调用 bsp_GetKey 读取键值即可。 */
ucKeyCode = bsp_GetKey(); /* 读取键值，无键按下时返回 KEY_NONE = 0 */
if (ucKeyCode != KEY_NONE)
{
    switch (ucKeyCode)
    {
        case KEY_DOWN_K1: /* K1 键按下 */
            t1 += 1;
            t2 += 2;
            EventRecord2(1+EventLevelAPI, t1, t2);
            t3 += 3;
            t4 += 4;
            EventRecord4(2+EventLevelOp, t1, t2, t3, t4);
            EventRecordData(3+EventLevelOp, s_ucBuf, sizeof(s_ucBuf));
            break;

        case KEY_DOWN_K2: /* K2 键按下 */
            printf("K2 按键按下\r\n");
            break;

        case KEY_DOWN_K3: /* K3 键按下 */
            printf("K3 按键按下\r\n");
            break;

        default:
            /* 其它的键值不处理 */
            break;
    }
}
}
```

应用程序里面主要实现了三个功能：

- ◆ 利用测量分组 A 实现 4 路时间的测量（第 1 路什么也没有测量，可以用来表示这两个函数本身执行占用的时间）。每 100ms 测量一次时间，效果如下：

Source	Count	Execution Timing
Event Start/Stop Group A		
Slot=0 (Errors=1)	79	T(tot)=122.32 T(avg)=1.55 T(min)=1.57 T(max)=1.78 T(first)=200.44ms T(last)=15.80s
Min t: Start: "..\..\User\app\src\main.c" (62)		Stop: "..\..\User\app\src\main.c" (63) t=1.57
Max t: Start: "..\..\User\app\src\main.c" (62)		Stop: "..\..\User\app\src\main.c" (63) t=1.78
Slot=1 (Errors=1)	79	T(tot)=389.00ms T(avg)=4.92ms T(min)=4.99ms T(max)=4.99ms T(first)=200.44ms T(last)=15.80s
Min t: Start: "..\..\User\app\src\main.c" (65)		Stop: "..\..\User\app\src\main.c" (67) t=4.99ms
Max t: Start: "..\..\User\app\src\main.c" (65)		Stop: "..\..\User\app\src\main.c" (67) t=4.99ms
Slot=2	79	T(tot)=2.37s T(avg)=30.00ms T(min)=30.00ms T(max)=30.00ms T(first)=205.43ms T(last)=15.81s
Min t: Start: "..\..\User\app\src\main.c" (69)		Stop: "..\..\User\app\src\main.c" (71) t=30.00ms
Max t: Start: "..\..\User\app\src\main.c" (69)		Stop: "..\..\User\app\src\main.c" (71) t=30.00ms
Slot=3	79	T(tot)=2.37s T(avg)=30.00ms T(min)=30.00ms T(max)=30.00ms T(first)=235.43ms T(last)=15.84s
Min t: Start: v1=26 v2=26		Stop: v1=26 v2=26 t=30.00ms
Max t: Start: v1=1 v2=1		Stop: v1=1 v2=1 t=30.00ms
Event Start/Stop Group B		
Slot=0 (Errors=94)	7974 (+1)	T(tot)=15.88s T(avg)=1.99ms T(min)=1.99ms T(max)=34.00ms T(first)=435.47 T(last)=15.95s
Min t: Start: "..\..\User\app\src\EventRecorderTest.c" (40)		Stop: "..\..\User\app\src\EventRecorderTest.c" (55) t=1.99ms
Max t: Start: "..\..\User\app\src\EventRecorderTest.c" (56)		Stop: "..\..\User\app\src\EventRecorderTest.c" (55) t=34.00ms
Event Start/Stop Group C		
Event Start/Stop Group D		

- ◆ 利用函数 EventRecord2, EventRecord4 和 EventRecordData 发送消息事件。按下按键 K1 进行更新, 效果如下:

Event	Time (sec)	Component	Event Property	Value
0	228.51042139		id=0x0001	0x00000002,0x00000004
1	228.51042342		id=0x0002	0x00000002,0x00000004,0x00000006,0x00000008
2	228.51042642		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000
3	230.36042115		id=0x0001	0x00000003,0x00000006
4	230.36042329		id=0x0002	0x00000003,0x00000006,0x00000009,0x0000000C
5	230.36042629		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000
6	231.47042131		id=0x0001	0x00000004,0x00000008
7	231.47042333		id=0x0002	0x00000004,0x00000008,0x0000000C,0x00000010
8	231.47042634		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000
9	232.77042115		id=0x0001	0x00000005,0x0000000A
10	232.77042318		id=0x0002	0x00000005,0x0000000A,0x0000000F,0x00000014
11	232.77042619		id=0x0003	0x666D7261,0x0000796C,0x00000000,0x00000000

- ◆ 基于 Event Recorder 的 printf 重定向。按下按键 K2 或者 K3 会打印消息, 效果如下:

Debug (printf) Viewer
K2 按键按下
K2 按键按下
K2 按键按下
K2 按键按下
K3 按键按下
K3 按键按下
K3 按键按下
K3 按键按下

1.15 总结

Event Recoder 还是非常实用的, 建议大家多使用几次, 熟练掌握。基本用上几次就上瘾, 离不开了, 的确是工程调试的利器。

1.16 相关专题教程汇总

【专题教程第 1 期】基于 STM32 的硬件 RGB888 接口实现 emWin 的快速刷新方案，32 位色或 24 色
<http://www.armbbs.cn/forum.php?mod=viewthread&tid=44512> 。

【专题教程第 2 期】uC/Probe 简易使用说明，含 MDK 和 IAR，支持 F103，F407 和 F429 开发板
<http://www.armbbs.cn/forum.php?mod=viewthread&tid=50280> 。

【专题教程第 3 期】开发板搭建 Web 服务器，利用花生壳让电脑和手机可以外网远程监控，极具可玩性
<http://www.armbbs.cn/forum.php?mod=viewthread&tid=56875> 。

【专题教程第 4 期】SEGGER 的 J-Scope 波形上位机软件，HSS 模式简单易用，无需额外资源，也不需要写目标板代码
<http://www.armbbs.cn/forum.php?mod=viewthread&tid=83097> 。

【专题教程第 5 期】工程调试利器 RTT 实时数据传输组件，替代串口调试，速度飞快，可以在中断和多任务中随意调用
<http://www.armbbs.cn/forum.php?mod=viewthread&tid=86177> 。

【专题教程第 6 期】SEGGER 的 J-Scope 波形上位机软件，RTT 模式波形上传速度可狂飙到 500KB/S 左右
<http://www.armbbs.cn/forum.php?mod=viewthread&tid=86881>。

【专题教程第 7 期】终极调试组件 Event Recorder，各种 Link 通吃，支持时间和功耗测量，printf 打印，RTX5 及中间件调试
<http://www.armbbs.cn/forum.php?mod=viewthread&tid=87176> 。

【专题教程第 8 期】基于 emWin 模拟器的 USB BULK 上位机开发，仅需 C 即可，简单易实现
<https://www.armbbs.cn/forum.php?mod=viewthread&tid=90026>。

【原创开源应用第 1 期】花式玩转网络摄像头之 TCP 上位机软件实现，高端大气上档次，速度 2MB/S，华丽丽的界面效果
<http://www.armbbs.cn/forum.php?mod=viewthread&tid=87016>。

【原创开源应用第 2 期】基于 RL-USB 和 RL-FlashFS 的完整 NAND 解决方案，稳定好用，可放心用于产品批量

<http://www.armbbs.cn/forum.php?mod=viewthread&tid=87118>。

【原创开源应用第 3 期】花式玩转网络摄像头之 VNC 远程桌面版本，稳定运行 2 年不死机，手机端和电脑端均可访问

<http://www.armbbs.cn/forum.php?mod=viewthread&tid=87362> 。

【原创开源应用第 4 期】给 ili9488，RA8875 等类显示屏的 emWin 底层增加 DMA 加速，提供 RTX，uCOS 和 FreeRTOS 版本

<http://www.armbbs.cn/forum.php?mod=viewthread&tid=87501>。

【原创开源应用第 5 期】基于 RL-USB+RL-FlashFS 的外挂 U 盘解决方案

<http://www.armbbs.cn/forum.php?mod=viewthread&tid=89202>。

【原创开源应用第 6 期】基于 SEGGER 的 FIND 小软件，快速检索局域网所有设备案例，非常实用

<http://www.armbbs.cn/forum.php?mod=viewthread&tid=89911>。



1.17 文档更新记录

版本	更改说明	作者	发布日期
V1.0	初版首发，制作于2018-06-13	白永斌 (Eric2013)	2018-06-15 (38页)
V1.1	1、更新文档的超链接。 2、修改增添相关专题教程的汇总章节。	白永斌 (Eric2013)	2023-03-10 (39页)