本教程以正点原子战舰 V4 开发板（实验 1，跑马灯实验）为例来说明 ST LINK 仿真器的驱动安装、下载和调试。其他开发板，请参考本教程的设置方法设置即可，都差不多。

1，ST LINK 仿真器驱动安装

安装仿真器驱动，以便我们使用相应的仿真器实现代码仿真调试。本节我们介绍 ST LINK 仿真器的驱动安装，ST LINK 的驱动我们已经放在开发板 A 盘→6，软件资料→1，软件→ST LINK 驱动及教程→ST-LINK 官方驱动.rar 里面，解压得到如图 1.1 所示内容：

图 1.1 ST LINK 驱动

如果我们是 64 位电脑系统，则双击：dpinst_amd64.exe 进行安装。如果是 32 位电脑系统，则双击：dpinst_x86.exe 进行安装。

安装完成后如图 1.2 所示：

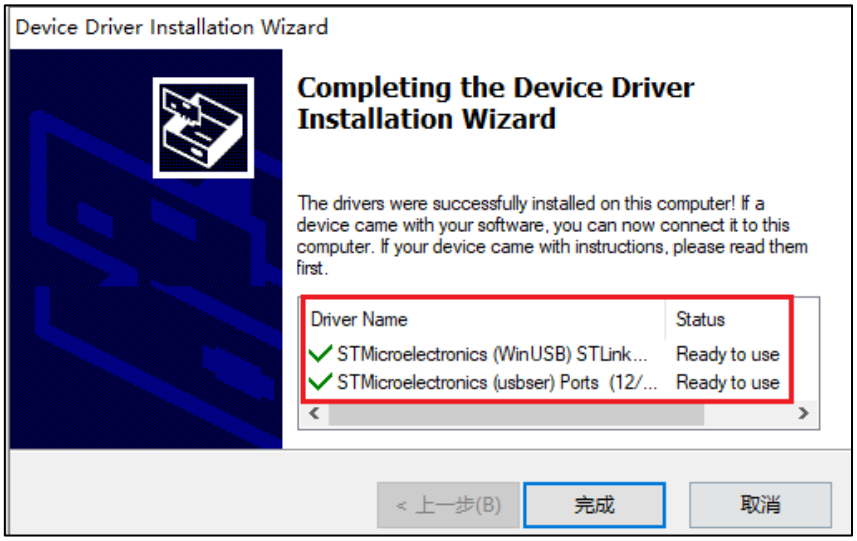

图 1.2 ST LINK 驱动安装成功

在 ST LINK 驱动安装完成以后，我们在电脑设备管理器里面可以看到会多出一个设备（此时 ST LINK 必须通过 USB 连接到电脑，ST LINK 红灯常亮），如图 1.3 所示：

图 1.3 设备管理器显示 ST LINK 设备

注意：

1，不同 Windows 版本设备名称和所在设备管理器栏目可能不一样，例如 WIN7 电脑插上 ST LINK 后显示的是：STMicroelectronics STLINK dongle。

2，如果设备名称旁边显示的是黄色的叹号，请直接点击设备名称，然后在弹出的界面点击更新设备驱动。

至此，ST Link 驱动已经安装完成。后续我们在 MDK 里面简单配置一下，即可支持 MDK 通过 ST LINK 仿真调试 STM32，这个我们后续再介绍。

2，使用 ST LINK 下载与调试程序

本节我们将给大家介绍如何使用仿真器给 STM32 下载代码，并调试代码。

这里我们以 ST LINK 为例给大家进行讲解，如果你用的是其他仿真器，基本上也是一样的用法，只是选择仿真器的时候，选择对应的型号即可。

ST LINK 与开发板连接如图 2.1 所示：

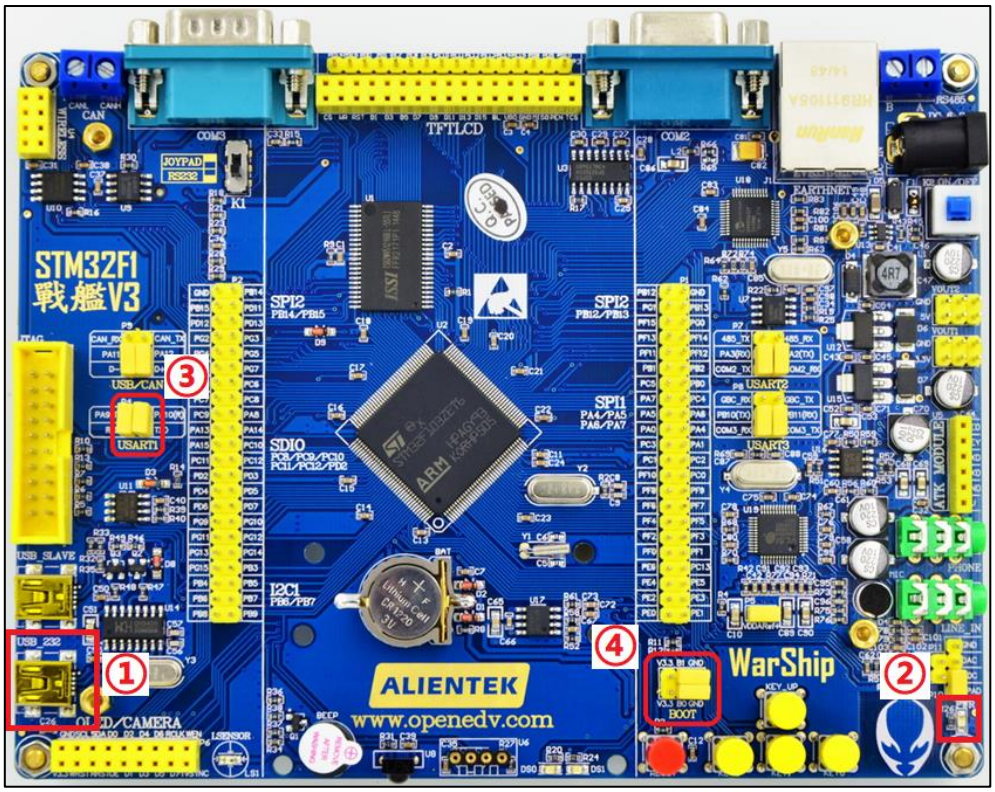

图 2.1 ST LINK 与开发板连接

① ST LINK 通过 USB 线连接电脑，且仿真器的红灯常亮（如果红灯闪烁，说明没有安装驱动，请参考上一节内容安装仿真器驱动）。

② ST LINK 通过 20P 灰排线连接开发板。

③ 确保开发板已经正常供电，蓝色电源灯亮起

④ B0，B1 都接 GND。

2.1 使用 ST LINK 下载程序

打开战舰 V4 A 盘→4，程序源码→1，标准例程-寄存器版本→实验 1 跑马灯实验 工程，并在 MDK IDE 主界面下，点击按钮，打开 Options for Target 选项卡，在 Debug 栏选择仿真工具为 use：ST-Link Debugger，如图 2.1.1 所示：

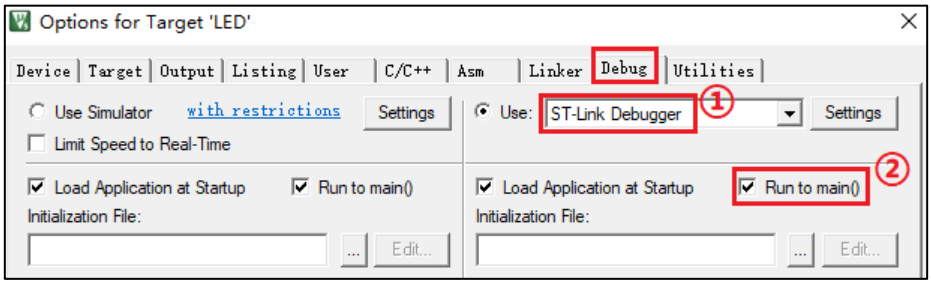

图 2.1.1 Debug 选项卡设置

① 选择使用 ST-Link Debugger 仿真器仿真调试代码。如果你使用的是其他仿真器，比如 DAP、JLINK、ULINK 等，请在这里选择对应的仿真器型号。

② 该选项选中后，只要点击仿真就会直接运行到 main 函数，如果没选择这个选项，则会先执行 startup_stm32f103xe.s 文件的 Reset_Handler，再跳到 main 函数。

然后我们点击 Settings，设置 ST LINK 的一些参数，如图 2.1.2 所示：

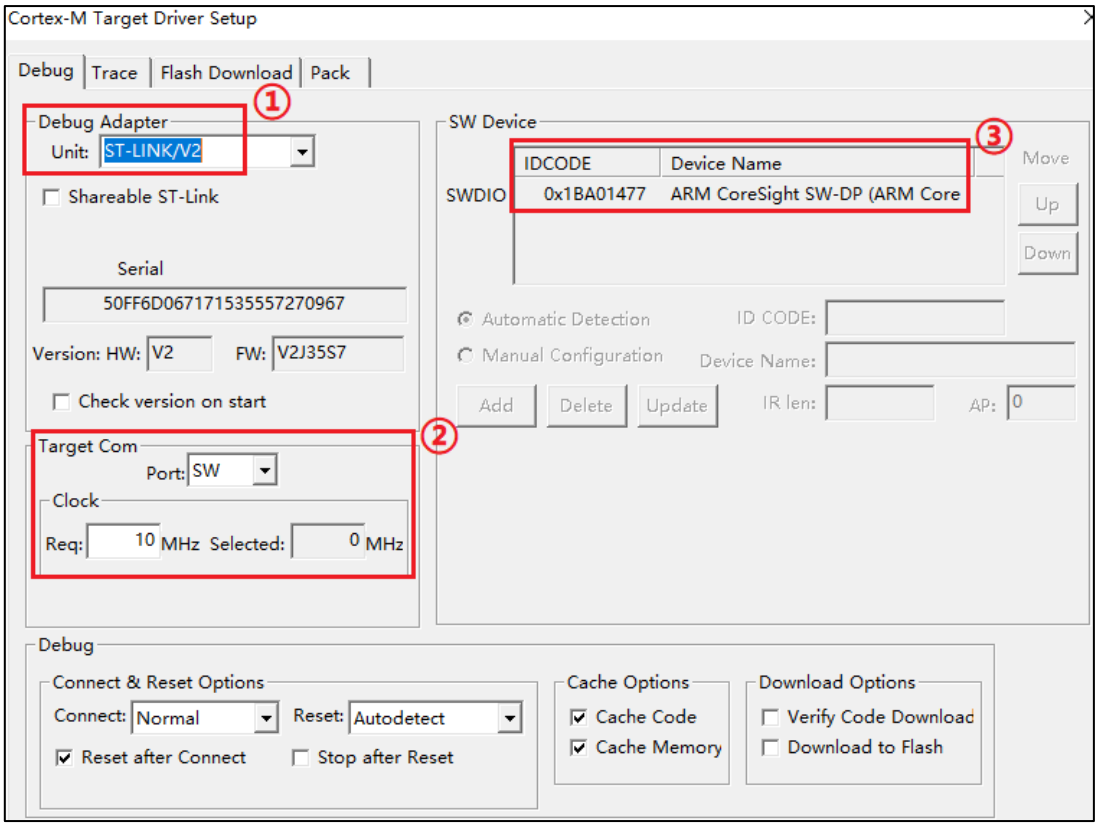

图 2.1.2 ST LINK 模式设置

① 表示 MDK 找到了 ST-LINK/V2 仿真器，如果这里显示为空，则表示没有仿真器被找到，请检查你的电脑是否接了仿真器？并安装了对应的驱动？

② 设置接口方式，这里选择 SW（比 JTAG 省 IO），通信速度设置为 10Mhz（实际上大概只有 4M 的速度，MDK 会自动匹配）。

③ 表示 MDK 通过仿真器的 SW 接口找到了目标芯片，ID 为：0x1BA01477。如果这里显示：No target connected，则表示没找到任何器件，请检查仿真器和开发板连接是否正常？开发板是否供电了？

其他部分使用默认设置，设置完成以后单击“确定”按钮，完成此部分设置，接下来我们还需要在 Utilities 选项卡里面设置下载时的目标编程器，如图 2.1.3 所示：

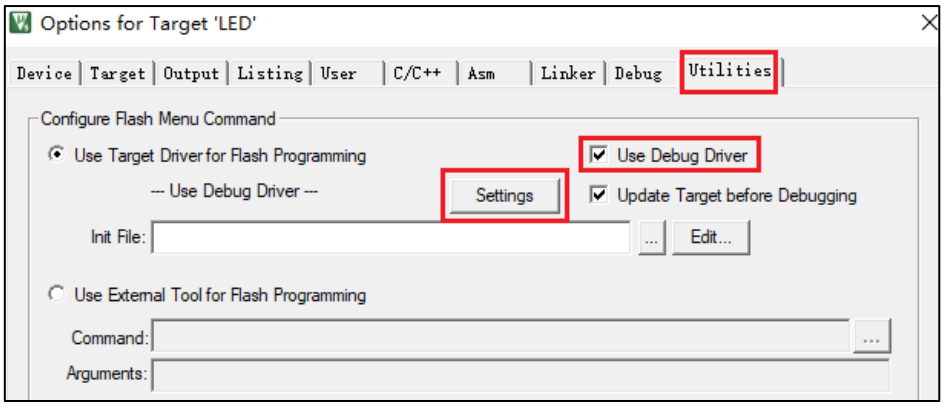

图 2.1.3 FLASH 编程器选择

上图中，我们直接勾选 Use Debug Driver，即和调试一样，选择 ST LINK 来给目标器件的 FLASH 编程，然后点击 Settings，进入 FLASH 算法设置，设置如图 2.1.4 所示：

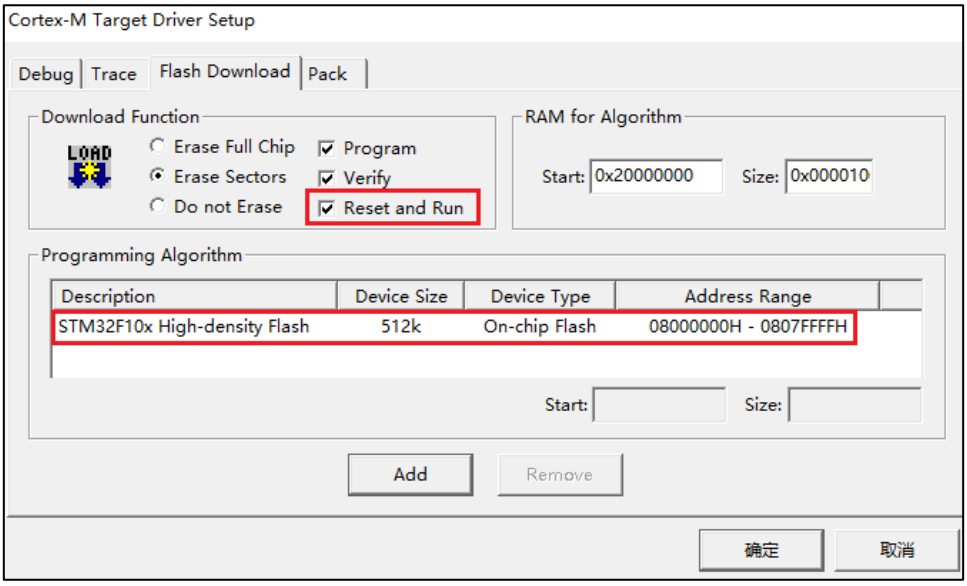

图 2.1.4 FLASH 算法设置

这里 MDK5 会根据我们新建工程时选择的目标器件，自动设置 flash 算法。我们使用的是 STM32F103ZET6，FLASH 容量为 512K 字节，所以 Programming Algorithm 里面默认会有 512K 型号的 STM32F10x High-density Flash 算法。另外，如果这里没有 flash 算法，大家可以点击 Add 按钮，自行添加即可。最后，选中 Reset and Run 选项，以实现在编程后自动运行，其他默认设置即可。

在设置完之后，点击“确定”，然后再点击“OK”，回到 IDE 界面，编译（可按 F7 快捷键）一下工程，编译完成以后（0 错误，0 警告），我们按  （快捷键：F8）这个按钮，就可以将代码通过仿真器下载到开发板上，在 IDE 下方的 Build Output 窗口就会提示相关信息，如图 2.1.5 所示：

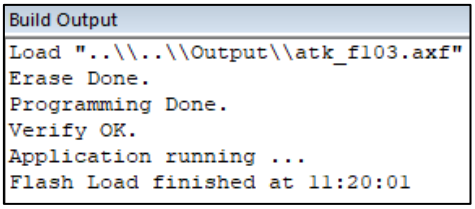

图 2.1.5 仿真器下载代码

下载完后，就可以看到 DS0 和 DS1 交替闪烁了，说明代码下载成功。

2.2 使用 ST LINK 仿真调试程序

在正常编译完例程以后（0 错误，0 警告），点击：（开始/停止仿真按钮），开始仿真（如果开发板的代码没被更新过，则会先更新代码（即下载代码），再仿真），如图 2.2.1 所示：

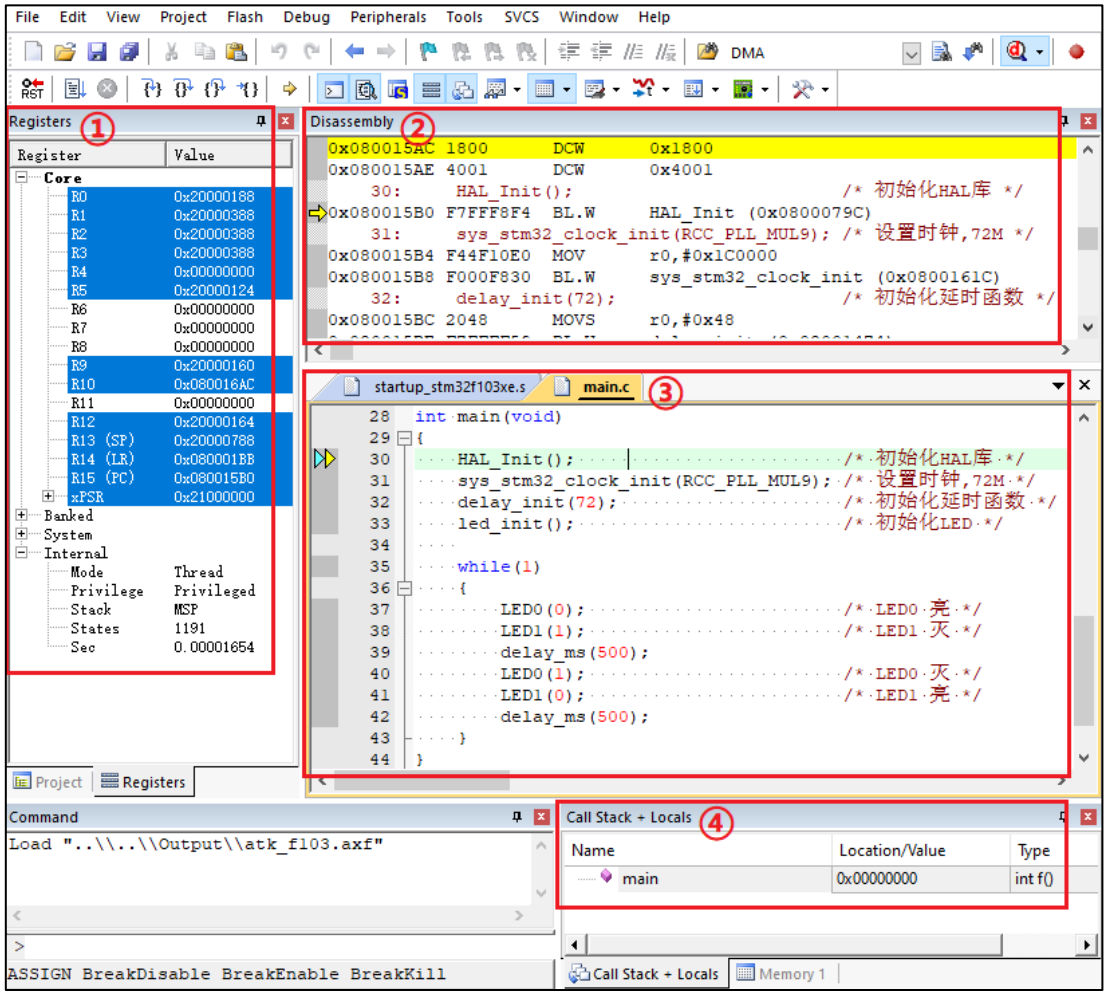

图 2.2.1 开始仿真

① Register：寄存器窗口，显示了 Cortex M3 内核寄存器 R0~R15 的值，还显示了内部的线程模式（处理者模式、线程模式）及特权级别（用户级、特权级），并且还显示了当前程序的运行时间（Sec），该选项卡一般用于查看程序运行时间，或者比较高级的 bug 查找（涉及到分析 R0~R14 数据是否异常了）。

② Disassembly：反汇编窗口，将 C 语言代码和汇编对比显示（指令存放地址，指令代码，指令，具体操作），方便从汇编级别查看程序运行状态，同样也属于比较高级别的 bug 查找。

③ 代码窗口，在左侧有黄绿色三角形，黄色的三角形表示将要执行的代码，绿色的三角形表示当前光标所在代码（C 代码 或 当前汇编行代码对应的 C 代码）。一般情况下，这两个三角形是同步的，只有在点击光标查看代码的时候，才可能不同步。

④ Call Stack + Locals：调用关系&局部变量窗口，通过该窗口可以查看函数调用关系，以及函数的局部变量，在仿真调试的时候，是非常有用的。

开始仿真的默认窗口，我们就给大家介绍这几个，实际上还有一些其他的窗口，比如Watch、Memory、外设寄存器等也是很常用的，可以根据实际使用选择调用合适的窗口来查看对应的数据。

图 2.2.1 中，还有一个很重要的工具条：Debug 工具条，其内容和作用如图 2.2.2 所示：

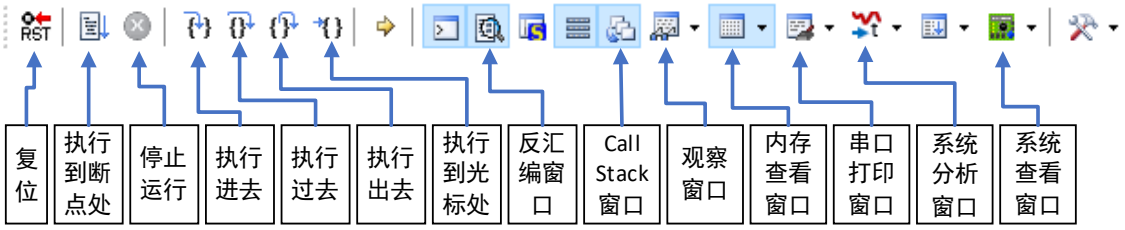

图 2.2.2 Debug 工具条

复位：其功能等同于硬件上按复位按钮。相当于实现了一次硬复位。按下该按钮之后，代码会重新从头开始执行。

执行到断点处：该按钮用来快速执行到断点处，有时候你并不需要观看每步是怎么执行的，而是想快速的执行到程序的某个地方看结果，这个按钮就可以实现这样的功能，前提是你在查看的地方设置了断点。

停止运行：此按钮在程序一直执行的时候会变为有效，通过按该按钮，就可以使程序停止下来，进入到单步调试状态。

执行进去：该按钮用来实现执行到某个函数里面去的功能，在没有函数的情况下，是等同于执行过去按钮的。

执行过去：在碰到有函数的地方，通过该按钮就可以单步执行过这个函数，而不进入这个函数单步执行。

执行出去：该按钮是在进入函数单步调试的时候，当不需要再执行该函数的剩余部分时，通过该按钮就可以一步执行完该函数的余部分，并跳出函数，回到函数被调用的位置。

执行到光标处：该按钮可以迅速的使程序运行到光标处，其实是挺像执行到断点处按钮功能，但是两者是有区别的，断点可以有多个，但是光标所在处只有一个。

反汇编窗口：通过该按钮，就可以查看汇编代码，这对分析程序很有用。

Call STACK 窗口：通过该按钮，显示调用关系&局部变量窗口，显示当前函数的调用关系和局部变量，方便查看，对分析程序非常有用。

观察窗口：MDK5 提供 2 个观察窗口（下拉选择），该按钮按下，会弹出一个显示变量的窗口，输入你所想要观察的变量/表达式，即可查看其值，是很常用的一个调试窗口。

内存查看窗口：MDK5 提供 4 个内存查看窗口（下拉选择），该按钮按下，会弹出一个内存查看窗口，可以在里面输入你要查看的内存地址，然后观察这一片内存的变化情况。是很常用的一个调试窗口

串口打印窗口：MDK5 提供 4 个串口打印窗口（下拉选择），该按钮按下，会弹出一个类似串口调试助手界面的窗口，用来显示从串口打印出来的内容。

系统分析窗口：该图标下面有 6 个选项（下拉选择），我们一般用第一个，也就是逻辑分析窗口(Logic Analyzer)，点击即可调出该窗口，通过 SETUP 按钮新建一些 IO 口，就可以观察这些 IO 口的电平变化情况，以多种形式显示出来，比较直观。

系统查看窗口：该按钮可以提供各种外设寄存器的查看窗口（通过下拉选择），选择对应外设，即可调出该外设的相关寄存器表，并显示这些寄存器的值，方便查看设置是否正确。

Debug 工具条上的其他几个按钮用的比较少，我们这里就不介绍了。以上介绍的是比较常用的，当然也不是每次都用得着这么多，具体看你程序调试的时候有没有必要观看这些东西，来决定要不要看。

在图 2.2.1 的基础上：关闭反汇编窗口（Disassembly）、添加观察窗口 1（Watch1）。然后调节一下窗口位置，然后将全局变量：g_fac_us （在 delay.c 里面定义）加入 Watch1 窗口（方法：双击 Enter expression 添加/直接拖动变量 t 到 Watch1 窗口即可），如图 2.2.3 所示：

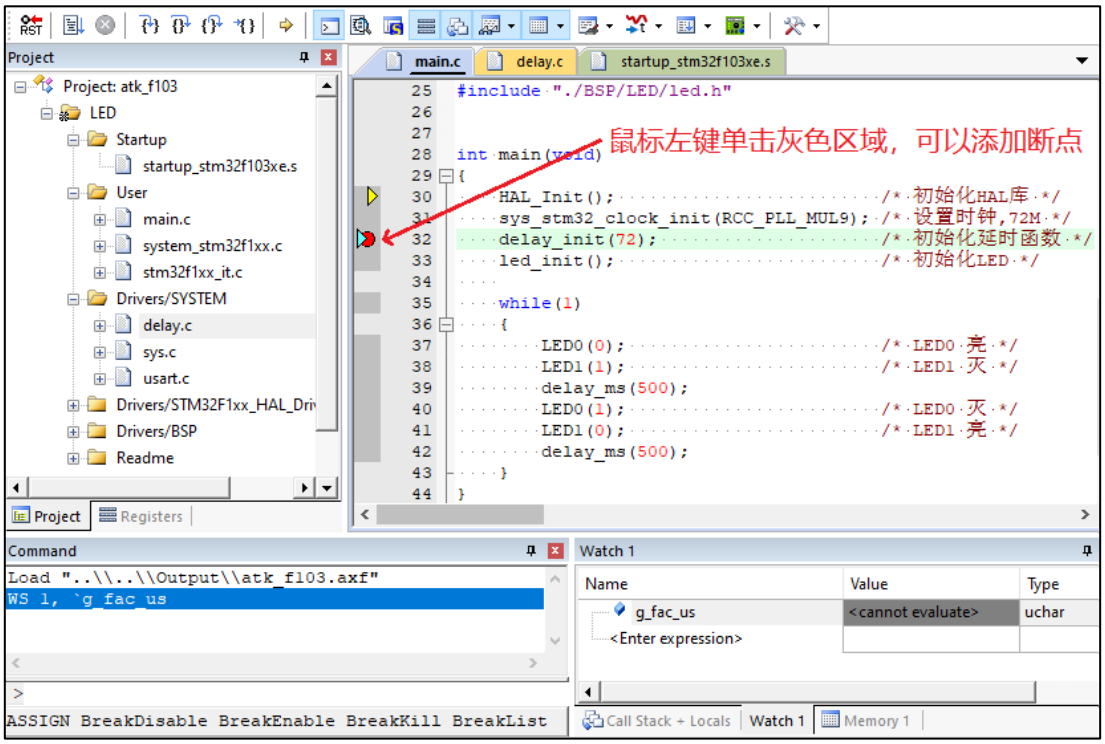

图 2.2.3 开始仿真

此时可以看到 Watch1 窗口的 g_fac_us 的值提示：无法计算，我们把鼠标光标放在第 32 行左侧的灰色区域，然后按下鼠标左键，即可放置一个断点（红色的实心点，也可以通过鼠标右键弹出菜单来加入），这样就在 delay_init 函数处放置一个断点，然后点击：，执行到该断点处，然后再点击：，执行进入 delay_init 函数，如图 2.2.4 所示：

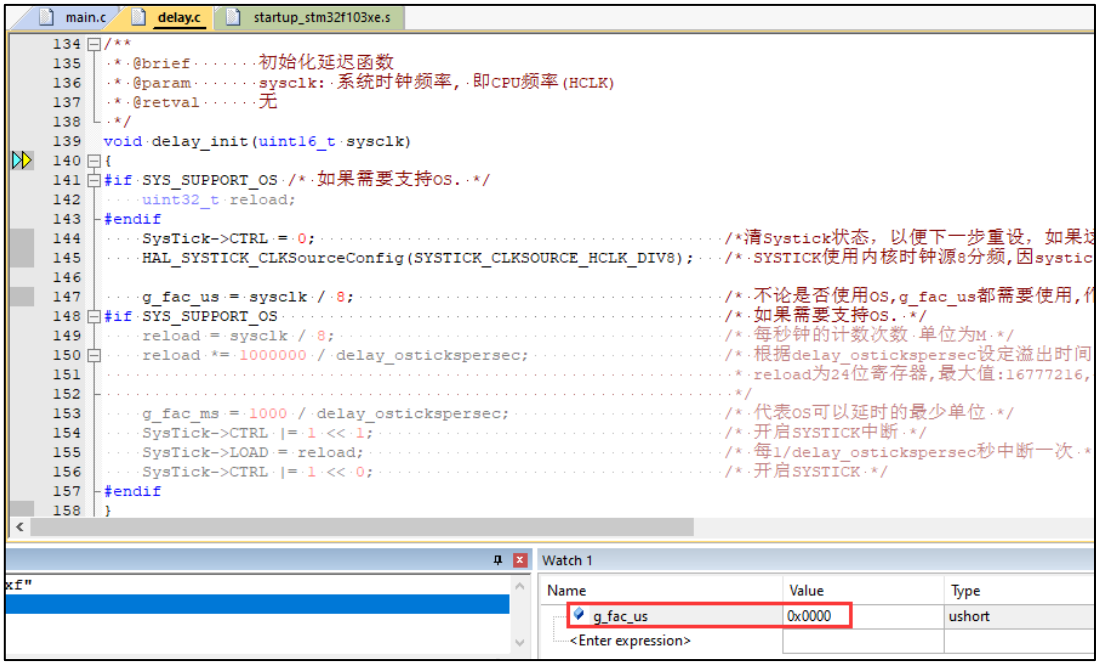

图 2.2.4 执行进入 delay_init 函数

此时，可以看到 g_fac_us 的值已经显示出来了，默认是 0（全局变量如果没有赋初值，

一般默认都是 0），然后继续单击：{}，单步运行代码到 g_fac_us=sysclk/8; 这一行，再把鼠标光标放在 sysclk 上停留一会，就可以看到 MDK 自动提示 sysclk（delay_init 的输入参数）的值为：0X0048，即 72，如图 2.2.5 所示：

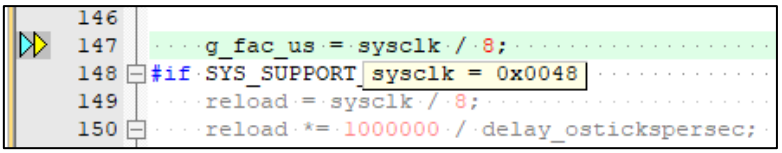

图 2.2.5 单步运行到 g_fac_us 赋值

然后再单步执行过去这一行，就可以在 Watch1 窗口中看到 g_fac_us 的值变成 9 了，如图 2.2.6 所示：

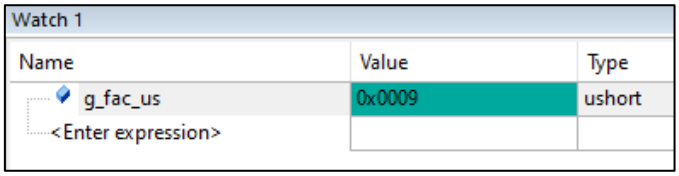

图 2.2.6 Watch1 窗口观看 g_fac_us 的值

由此可知，某些全局变量，我们在程序还没运行到其所在文件的时候，MDK 仿真时可能不会显示其值（如提示：cannot evaluate），当我们运行到其所在文件，并实际使用到的时候，此时就会显示其值出来了！

然后，我们再回到 main.c，在第一个 LED0(0)处放置一个断点，运行到断点处，如图 2.2.7 所示：

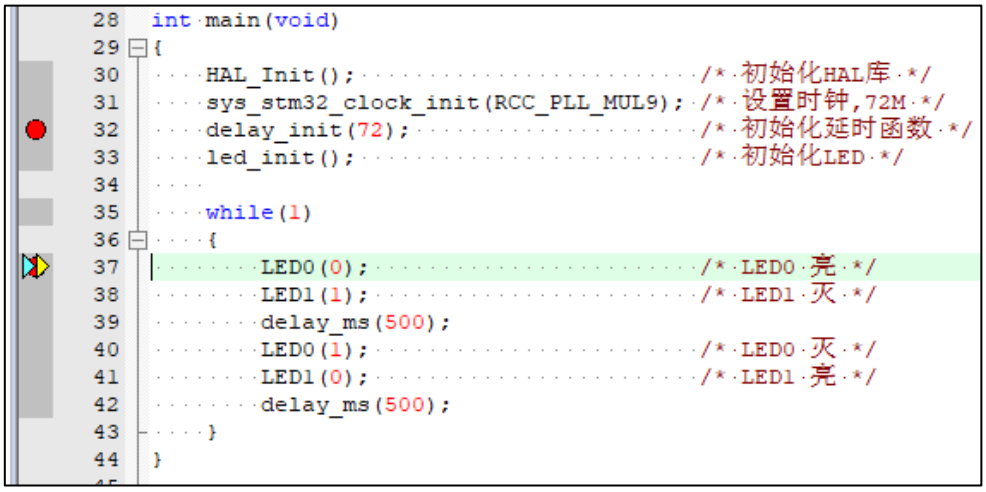

图 2.2.7 运行到 LED0(0)代码处

此时，我们点击：{}，执行过这一行代码，就可以看到开发板上的 DS0（红灯）亮起来了，以此继续点击{}，依次可以看到：DS0 灭→DS1 亮→DS0 亮→DS1 灭→DS0 灭→DS1 亮……,一直循环。这样如果全速运行，就可以看到 DS0，DS1 交替亮灭，不过全速运行的时候，一般是看不出 DS0 和 DS1 全亮的情况的，而通过仿真，我们就可以很容易知道有 DS0 和 DS1 同时亮的情况！

最后，我们在 delay.c 文件的 delay_us 函数，第二行处设置一个断点，然后运行到断点处，如图 2.2.8 所示：

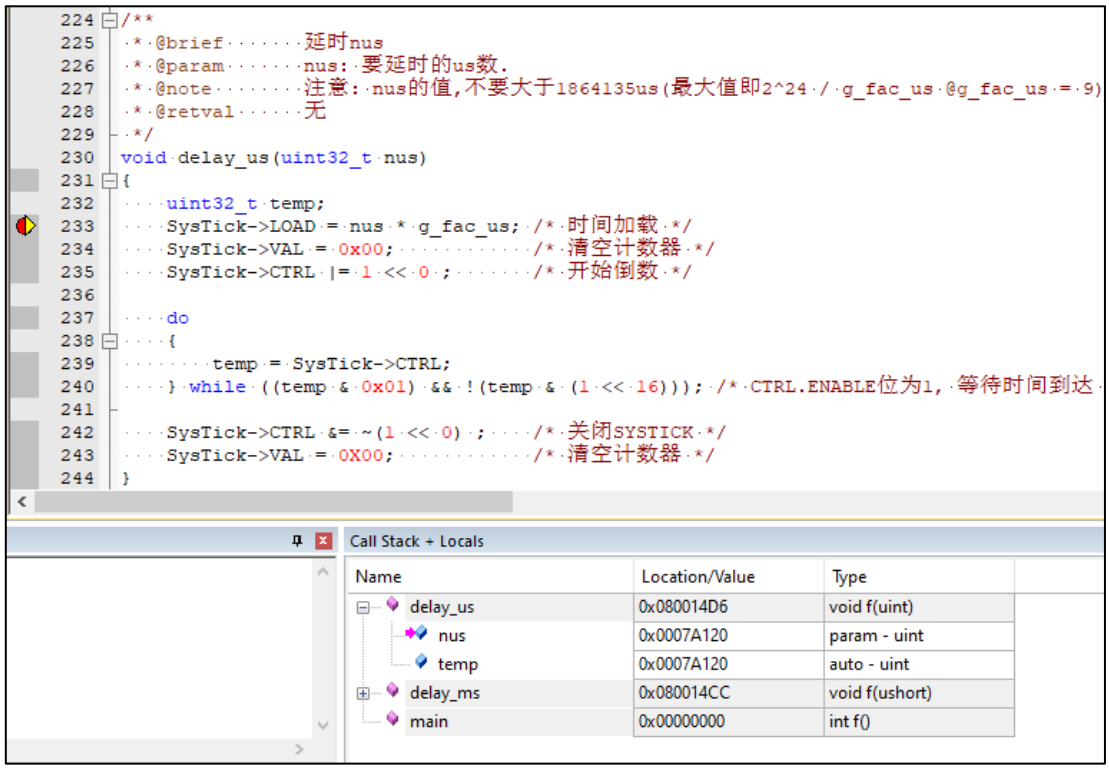

图 2.2.8 查看函数调用关系及局部变量

此时，我们可以从 Call Stack + Locals 窗口看到函数的调用关系，其原则是：**从下往上看**，即下一个函数调用了上一个函数，因此，其关系为：main 函数调用了 delay_ms 函数，然后 delay_ms 函数调用了 delay_us 函数。这样在一些复杂的代码里面（尤其是第三方代码），可以很容易捋出函数调用关系并查看其局部变量的值，有助于我们分析代码解决问题。

关于 ST LINK 的仿真调试，我们暂时就讲这么多。

2.3 仿真调试注意事项

1， 由于 MDK5.23 以后对中文支持不是很好，具体现象是：在仿真的时候，当有断点未清除时点击结束仿真，会出现如图 2.3.1 所示的报错：

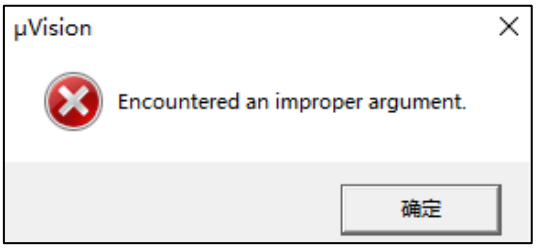

图 2.3.1 仿真结束时报错！

此时我们点击确定，是无法关闭 MDK 的，只能到电脑的任务管理器里面强制结束 MDK，才可以将其关闭，比较麻烦。

该错误就是由于 MDK5.23 以后的版本对中文支持不太好导致的，这里提供 2 个解决办法：**a，仿真结束前将所有设置的断点都清除掉**，可以使用 File 工具栏的：按钮，快速清除当前工程的所有断点，然后再结束仿真，就不会报错；**b，将工程路径改浅，并改成全英文路径**（比如，将源码拷贝到：E 盘→Source Code 文件夹下。注意：例程名字一般可以不用改英文，因为只要整个路径不超过 10 个汉字，一般就不会报错了，如果还报错就再减少汉字数量）。通过这两个方法，可以避免仿真

结束报错的问题。我们推荐大家使用第二种方法，因为这样就不用每次都全部清除所有断点，下回仿真又得重设的麻烦。

2， 关于 STM32 软件仿真，老版本的教程，我们给大家介绍过如何使用 MDK 进行 STM32 软件仿真，由于其限制较多（只支持部分 F1 型号），而且仿真器越来越便宜，硬件仿真更符合实际调试需求，调试效果更好。所以后续我们只介绍硬件仿真，不再推荐大家使用软件仿真了！

3， 仿真调试找 bug 是一个软件工程师必备的基本技能。MDK 提供了很多工具和窗口来辅助我们找问题，只要多使用，多练习，肯定就可以把仿真调试学好。这对我们后续的独立开发项目，非常有帮助。因此极力推荐大家多练习使用仿真器查找代码 bug，学会这个基本技能。

4， 调试代码不要浅尝辄止，要想尽办法找问题，具体的思路：先根据代码运行的实际现象分析问题，确定最可能出问题的地方，然后在相应的位置放置断点，查看变量，查看寄存器，分析运行状态和预期结果是否一致？从而找到问题原因，解决 bug。特别提醒：一定不要浅尝辄止，很多朋友只跟踪到最上一级函数，就说死机了，不会跟踪进去找问题！所以一定要一层层进入各种函数，越是底层（甚至汇编级别），越好找到问题原因。