VSCode 应用实例

在本章中，将通过具体实例向读者介绍如何使用 VSCode 完成项目的开发与调试。本章将根据具体实例讲述智能编辑代码、编译项目、调试功能等。

1 VSCode 简介

VSCode For ESSEMI （简称 VSCode）是由上海东软载波微电子有限公司自主研发的新一版本的集成开发环境，为用户提供了完整的设计、开发、调试、部署嵌入式应用程序的工具包。本产品是基于 Visual Studio Code 开发的扩展包，其本身对于运行环境要求很低，不需要安装环境，支持 Windows7 及以上版本、 Ubuntu16.04 及以上版本。本文档仅介绍基于 Visual Studio Code 扩展的相关功能，其他功能可参考Visual Studio Code 官网。

2 VSCode 安装

2.1 Windows 下安装 VSCode

运行 VSCode 的安装程序，进入安装界面，依次执行以下操作：

1. 打开 VSCode 安装软件，跳出"用户账户控制页面"，点击"是"。
2. 进入 VSCode 安装程序界面，单击"下一步"。
3. 选择安装目录，单击"下一步"。
4. 单击 "完成"。

2.2 Linux 下安装 VSCode

在Linux 下安装 VSCode 依次执行以下操作，桌面会生成 VSCode 快捷方式：

1. 复制 ESSEMI.tar.gz 及 Install.sh 到目标安装目录。
2. 在终端窗口运行 Install.sh 脚本。

注意，VSCode 在 Linux 环境下支持 ES32/ES8P 系列32位芯片开发，支持使用 JLink 进行仿真调试；不支持 ES7P 系列8位芯片编译调试。

3 构建项目

VSCode 支持通过多种方式进行项目的构建：打开文件夹、加载iDesigner/Keil项目、新建项目。本节将M950项目的ADC例程以打开文件夹的形式加载到工作区，通过对项目类型及芯片进行选择完成项目配置。具体操作步骤如下：

1. 点击主菜单栏中的"文件"一栏。
2. 点击"打开文件夹"。
3. 在"打开文件夹"对话框中选择项目文件夹"ADC"。
4. 单击"选择文件夹"按钮。
5. 在VSCode界面出现芯片类型选择列表，单击选择"Cortex-M"，如图所示。

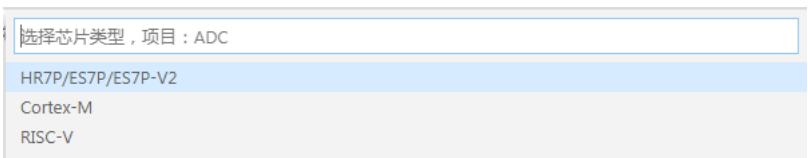

图3-1 "芯片类型"选择框

6. 在VSCode界面出现"Cortex-M"类型对应的芯片列表，如图所示，点击选择"ES32F3696LT"，则成功构建项目。

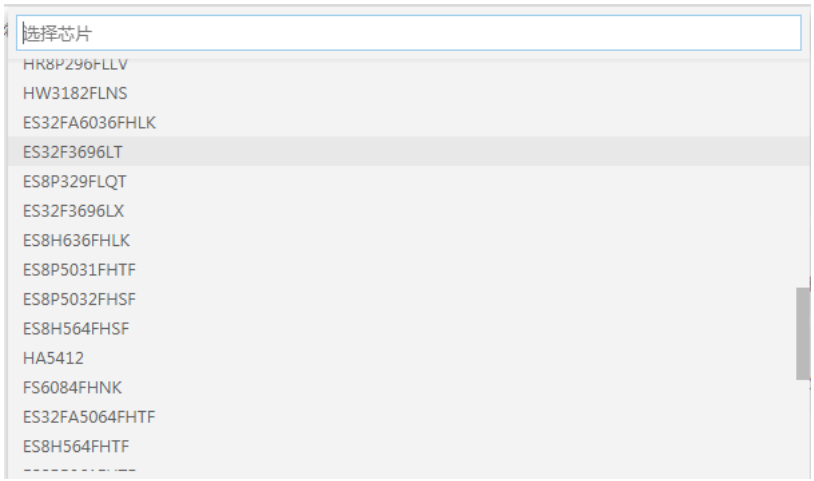

图3-2 芯片列表

注意：如果打开的文件夹目录在 Windows 资源管理器里存在 project.escfg 文件，则代表此项目已构建好，则不会执行5、6步骤。

4 编辑与智能编码辅助功能

VSCode 具有高效、强大的语言支持，功能、效率、准确度相较于 iDesigner 有很大的提升，给用户提供智能编码辅助功能。本节将详细介绍如何使用 VSCode 提供的代码补全、诊断提示、函数签名、查找引用等一系列智能编码辅助功能。

4.1 代码补全

"代码补全"功能会显示当前区域的有效成员，包括全局变量、局部变量、函数等。

在键入符号时，成员列表将自动显示，并匹配键入的内容，若有一个或多个匹配项，将自动选中第一个匹配项。选择列表中的某个成员，按"Enter"可以将该成员插入到代码中。按"Esc"可以关闭成员列表。

在 main.c 文件中输入已定义过的结构体变量 `timer_init_struct` ，并键入 "."，则补全列表将 `timer_init_struct` 的成员全部列出，如图4-1所示。通过上下快捷键选中 `mode` 成员，按 Enter 快捷键成功将此结构体成员变量插入到代码中。对于结构体类型的指针键入"->"则可以显示成员列表。

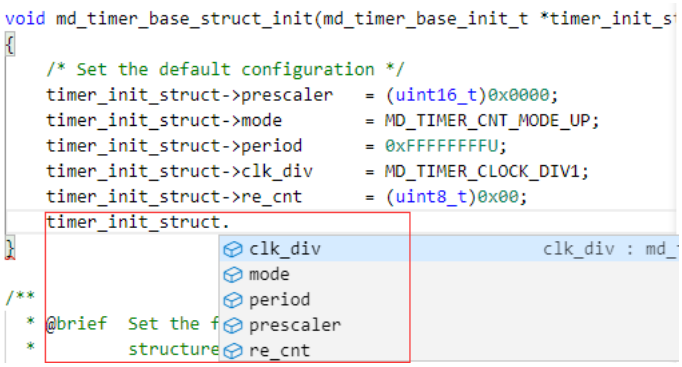

图4-1 代码补全

4.2 诊断提示

在"问题窗口"里可以查看到项目的全部错误信息列表，单击错误信息 `use of undeslcred identifiter 'timer'` ，则编辑界面跳转到错误位置 main.c 文件第67行，在图4-2中，可以看到在main.c文件第67行的错误地方标注红色曲线，将鼠标悬停在此处则会提示相关的诊断信息。

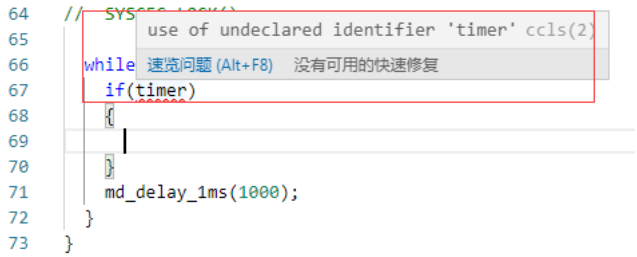

图4-2 诊断提示

4.3 悬停提示

"悬停提示"可为代码中的任意标识符显示完整的声明，通常为标识符的类型和说明。从列出成员框中选择成员时也会出现快速信息。

鼠标悬停在main.c文件第40行的变量 `config` 上，则显示了该变量的类型为 `md_adc_ich_conf_t` 类型的指针变量，如下图4-3所示。

```
37  void md_adc_insert_struct_init(md_adc_ich_conf_t *config)
38  {
39      md_adc_ich_conf_t *config NNEL_2;
40      config->idx     = MD_ADC_ICH_IDX_1;
41      config->samp    = MD_ADC_SAMPLETIME_4;
42      config->offset = 0;
43      config->nr      = MD_ADC_ICH_NR_1;
44      config->auto_m = DISABLE;
45
46      return;
47  }
```

图4-3 悬停提示

4.4 函数签名

在编辑代码时经常要引用已定义过的函数，为了能够快速直接地知道此函数的用法，包括其参数、返回值等，这就需要函数签名功能。此功能无需开发人员转到函数定义位置，只需直接编辑输入此函数名则会自动跳出此函数说明的悬浮框。

在main.c文件的代码编辑区域第60行输入已定义过的函数 `md_adc_struct_init()` ，则出现一个弹出框显示 `md_adc_struct_init` 函数的返回值类型和参数列表等。如下图4-4所示。根据函数签名提示可以看出 `md_adc_struct_init` 函数的参数为 指向 `md_addc_init_t` 的指针类型 ，且通过函数签名可以看出 `md_adc_struct_init` 函数无返回值。

```
56  while (1) {
57
58      md_delay_1ms(1000);
59
60      md_adc_struct_init()
61  }
```

md_adc_struct_init(md_adc_init_t *init) -> void

图4-4 函数签名

4.5 查找所有引用

"查找所有引用"功能可以找到项目中所有引用到所选符号的位置。

`md_delay_1ms` 函数在很多地方都被调用，可以通过"查找所有引用"功能直接在视图中清晰准确的知道左所有的引用地方，以及快速第跳转到指定的位置。在图4-5中，右击代码中第58行的函数 `md_delay_1ms(1000)` ，在快捷菜单中选择 "Find All References"，在侧边栏 "REFERENCES RESULTS" 窗口中显示所有引用函数 `md_delay_1ms(1000)` 的文件及结果数量。如下图4-5所示。

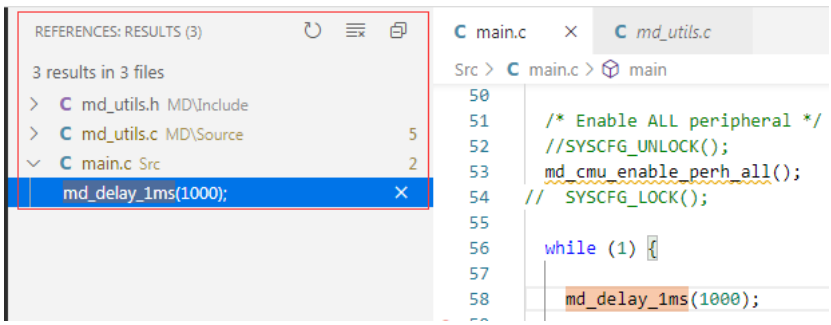

图4-5 查找引用结果

结果以文件对应查找到的符号展示，用户可以通过展开文件查看该文件中引用的位置，双击子项，代码编辑器将跳转到该项的位置。

4.6 显示文件所有符号

单击打开 main.c 文件，展开"大纲"一栏显示 main.c 文件中的所有符号，如图4-6所示，可以清楚直接地看到当前main.c文件中的所有符号信息，单击 main.c 文件中的符号 `adc_pin_init` 则光标跳转到此符号位置。

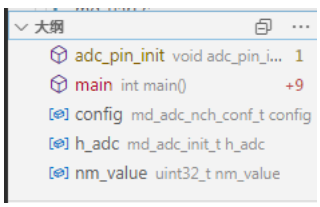

图4-6 "大纲"列表

4.7 CodeLens

在 md_flash.c 文件中，想要查看 在ADC项目中有哪些地方引用了`pragram_flash_lock`函数，则单击第58行代码上方的 "4 refs"，在编辑界面显示函数`pragram_flash_lock`在项目中的所有引用位置，单击任意引用位置可查看具体引用信息，其中 "4 refs"代表整个项目中有四个位置引用了此函数。如图4-7所示。

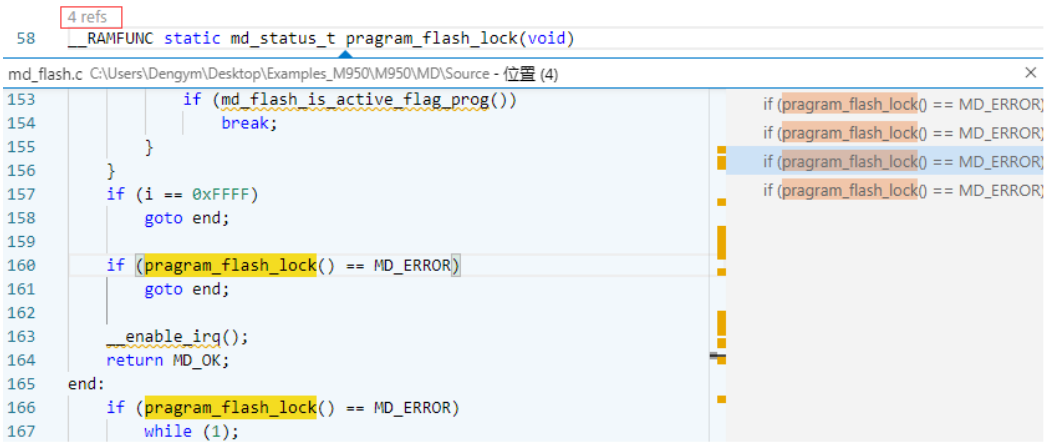

图4-7 CodeLens

5 编译项目

编辑完所有程序代码之后，需要编译项目，生成 Hex 文件方可使用。VSCode 提供了编译项目、重新编译项目、全部编译、全部重新编译以及编译文件五种编译操作。

右击本项目任意位置，从快捷菜单中选择"编译项目"，将编译该项目。如果程序存在错误，编译结束后，输出窗口将显示"编译出错"，切换到"问题"窗口查看详细信息，如图5-1所示。单击错误信息，在编辑窗口中定位到错误位置。

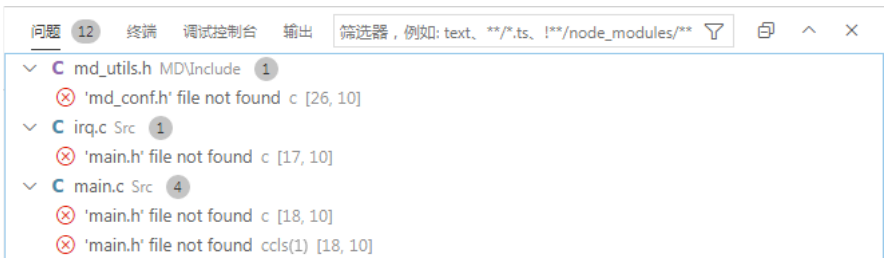

图5-1 问题列表

从详细信息中可以看出，错误原因是编译器无法找到源文件中包含的头文件。本项目的源文件中直接通过文件名引用了项目中的2个头文件，而在项目结构中，头文件位于子文件夹 Inc 中，与源文件并不在同一个目录中。

对于头文件与源文件不在同一个目录的情况，一般有以下两种修改方式：

1. 在源文件中包含头文件时指定头文件路径。
2. 在编译选项中设置头文件搜索路径。

注：**这两种方式均可以使用绝对路径或相对路径。**使用相对路径时需要注意的是，直接包含头文件时，使用的相对路径是相对于源文件本身的，而在编译选项中使用的是相对于项目文件夹的。

本项目中，使用编译选项设置搜索路径。右键单击项目任意位置，选择"属性"，在弹出的项目属性切换到"编译选项"页，在 "C Include files search directories" 一栏中单击"+"符号，输入相对路径 "Inc"，如图5-2所示。

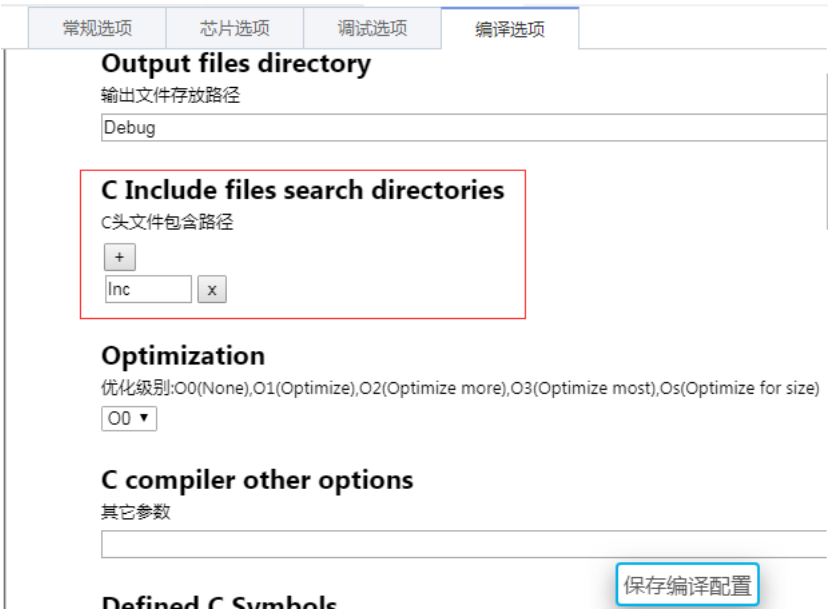

图5-2 编译选项页

之后重新再执行一次"编译项目"或"重新编译项目"操作，生成成功。编译成功后，输出窗口中将显示编译时间，如图5-3所示。

图5-3 输出信息

编译命令建议

当修改少量源文件的代码时，执行生成"编译项目"操作可以节省编译时间。添加、删除、重命名.h 或.inc文件后，建议开发人员不要使用"编译项目"功能，应该使用"重新编译项目"，否则编译结果可能不正确。当工作区下有多个项目时，用户可选择"全部"编译操作，对工作区下的所有项目全部进行编译。开发人员也可对通过"编译文件"对某个源文件进行编译操作。

5.1 优化选项

在图5-2中，可以看到一个选项 "Optimization"，在此可以设置优化级别。优化级别有以下5种：O0、O1、O2、O3、Os。其中，默认优化级别为O0，即为常规优化，不进行任何优化；O1优化会消耗较多的编译时间，它主要对代码的分支，常量以及表达式等进行优化；O2会尝试更多的寄存器级的优化以及指令集的优化，它会在编译期间占用更多的内存和编译时间；O3在O2的基础上进行更多的优化，例如使用伪寄存器网络，普通函数的内联，以及针对循环的更多优化；Os主要是对代码大小的优化，打开了大部分O2优化中不会增加程序大小的优化选项，并对程序代码的大小做更深层的优化，通常我们不需要这种优化。

6 调试项目

本节将详细介绍如何使用 VSCode 提供的变量监视、查看内存、断点、可视化、反汇编等调试功能。

6.1 启动调试

将调试工具与 PC 连接，右键单击项目任意位置，选择 "属性"菜单，将弹出的项目属性切换到"调试选项"页，在"调试工具"一栏下拉选择与PC所连的通信口对应的调试工具，更改设置以后点击"保存调试配置"按钮。如图6-1所示。

图6-1 调试选项窗口

打开 main.c 文件，单击当前活动页上方栏的"启动调试"(

)按钮，开始向芯片中下载配置字及程序，下载成功后， VSCode 界面将切换至调试模式。在调试模式下， VSCode 将禁止编辑功能，并启用调试工具栏，允许使用各种调试窗口，如图6-2所示。若下载失败， VSCode 将不会进入调试模式，而是在"输出"窗口提示相应的错误信息。

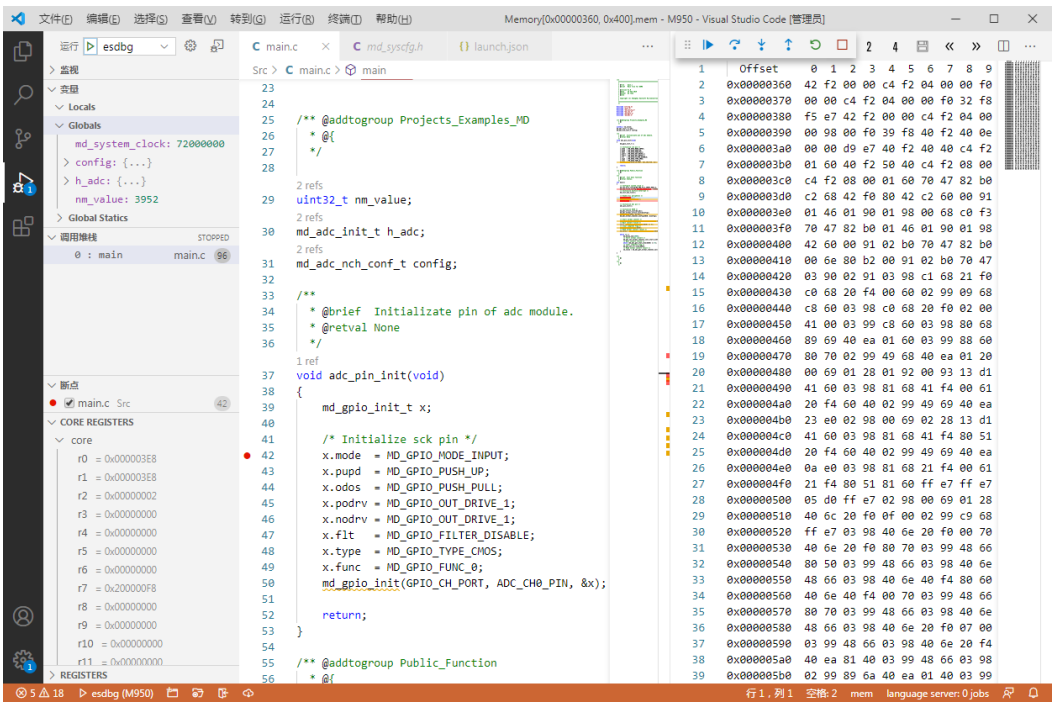

图6-2调试模式

6.2 变量监视窗口

在调试过程中，监视全局的结构体变量 `config` 可以通过以下三种方式：

- 鼠标悬停在变量 `config` 上，则出现此变量当前值的浮动提示框。
- 在"运行"视图的"变量"部分，展开 "Globals" ，可以看到变量 `config` 每个成员变量的值，如图6-3所示。双击成员名可以在出现的输入栏编辑此成员变量的值，用于模拟输入信号等。
- 打开"运行"视图，在"监视"一栏中点击"添加表达式"图标，在输入框输入 `config` 并回车，则在"监视"视图可以看到结构体变量 `config` 当前值，展开查看其每个成员当前对应的值，如图6-3所示。

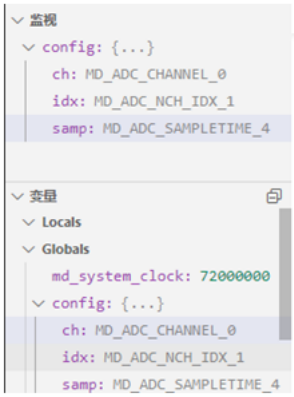

图6-3 监视窗口 图6-4 寄存器窗口

注意：用户可点击变量监视窗口的“切换十六进制显示”按钮（ H ），将数据格式切换为十六进制。

6.3 寄存器窗口

在调试过程中，可以通过寄存器窗口查看当前程序暂停时寄存器的值。其中，"CORE REGISTERS"（内核寄存器窗口）一栏显示了当前程序暂停时内核寄存器的值，可以通过“REGISTERS”（外设寄存器窗口）一栏查看芯片的特殊功能寄存器的状态。

当前程序停在`md_gpio.c`文件的`md_gpio_init`函数第46行，在"运行"视图中点击 “CORE REGISTERS” 一栏，在展开的列表中点击 “core” 即打开查看当前内核寄存器的值。如上图6-4 "CORE REGISTERS"一栏所示。内核寄存器`PC`即当前的PC值为`0X0000111C`。

程序暂停时，点击"运行"视图-> "REGISTERS"，即可查看当前程序状态下芯片的特殊功能寄存器的状态，对于支持位功能的寄存器CMU，点击 CMU 前的控件符号，则展开查看支持的位。如图6-4 "REGISTERS"一栏所示。

6.4 内存窗口

在`main.c`文件第89行的`for`循环中，对`int`类型数组`a`的部分元素进行了赋值。执行完循环后，并不知道`a`中哪些元素的值发生了变化，及是否符合预期值，而在变量窗口中需要频繁地翻页才能查看全部元素，此时可以通过内存窗口来观察`a`的变化。在编辑界面右击并从快捷菜单中选择"显示内存窗口"。通过在监视窗口添加`&a`可查看`a`在 RAM 中分配的地址为0x20000000，所以在内存窗口的地址中输入 0x20000000，按 "Enter" 键，在出现的"内存长度"输入栏中输入内存长度为120并回车，此时内存窗口的数据就是`a`的数据。其中以被红色方框框起的表示当前调试状态下发生改变的数据。在循环执行完成后，通过内存窗口可以很方便地看到`a`每次发生的改变，如图6-5所示。

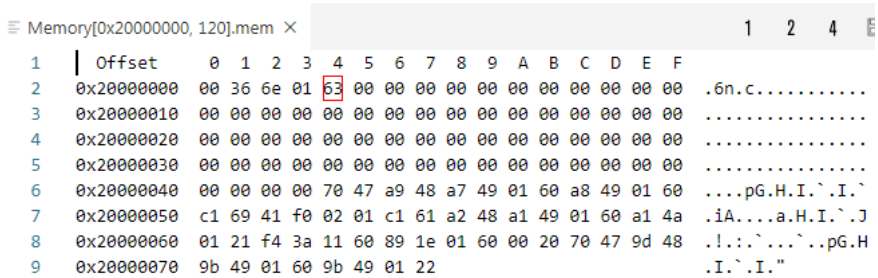

图6-5 内存窗口

注意：在内存窗口右上角可以切换数据显示格式，默认为"1"即以单字节显示，还可以切换"2"（以双字节显示）或切换为"4"（以四字节显示），单击对应的数字即可切换为相应的数据格式。

6.5 断点

在调试如图6-6所示程序时，希望能够监视循环中每次符合`if`条件时程序的运行状态。如果使用"单步调试"命令的话，可能会一直在`if`语句块之外单步运行，需要执行多次"单步调试"命令，才能进入一次`if`语句块。单击`if`语句块内第一行（即第91行）的左侧空白处，即可在改行设置一个 PC 断点，在行首出现的红色实心圈表示断点设置成功。然后点击"启动调试（F5）"按钮全速运行该段程序，当程序满足`if`条件时，就会在第91行处暂停，此时可以方便地查看`a`、`i`等变量的状态，或使用"单步跳过"命令执行`if`语句块内的代码。执行"继续"命令，程序会在下一次符合`if`条件时，再次暂停。

```
82  int main()
83  {
84      int i;
85
86      int temp[30];
87      init_str();
88      int a[30];
89      for (i = 0; i < 30; i++) {
90      if (i % 3 == 1) {
91        a[i] = i / 3 + 100 - i;
92        temp[i] = a[i];
```

图6-6 PC 断点

6.5.1 禁用与删除断点

如果某次调试时，暂时不希望某个断点起作用，可以右键单击断点，在弹出的快捷菜单中选择"禁用断点"，该断点将变为禁用状态（空心圈），而右键单击空心圈选择"启用断点"可以再次启用被禁用的断点。如果以后不再需要该断点，单击断点可以将其删除。

6.6 调用堆栈窗口

函数 `md_gpio_init` 在很多地方都被调用，在其中设置断点使得调试程序暂停时，通过"调用堆栈"视图可以方便地查看它是被何处代码调用的。点击侧边栏"运行"->"调用堆栈"可以展开堆栈窗口，它显示了当前的函数调用堆栈，栈顶为当前PC所在函数 `md_gpio_init`。双击它的上一级堆栈 `adc_pin_init`，编辑窗口就会跳转到 `adc_pin_init` 方法中调用 `md_gpio_init` 的位置，以绿色箭头作为标记。可以看到，当前调用 `md_gpio_init` 的是 `adc_pin_init` 中第50行代码，如图6-7所示。

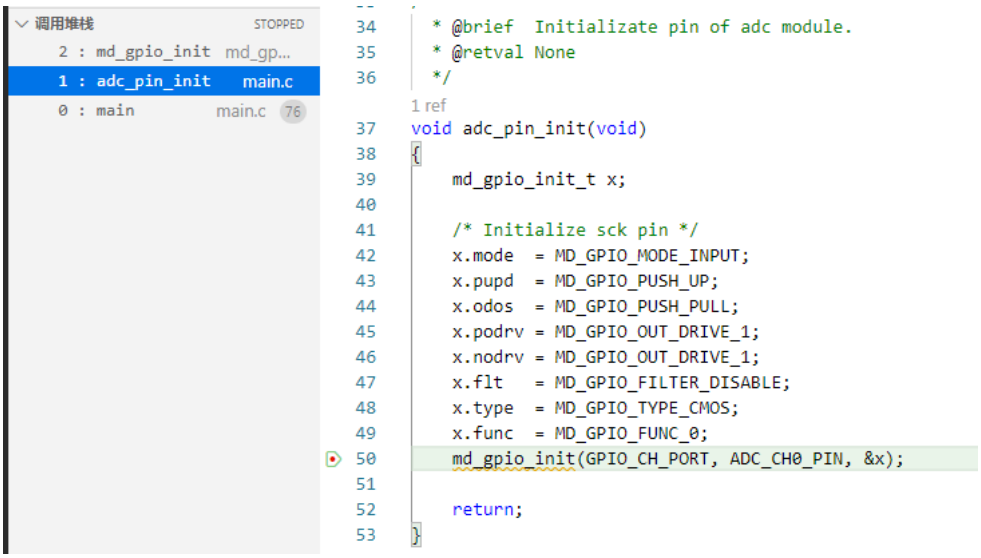

图6-7 "调用堆栈"窗口

6.7 调试反汇编

程序在 `main.c` 函数 第87行暂停时，点击编辑界面上方的"切换到反汇编单步（ ）"快捷按钮，执行"单步调试"命令时，界面将跳转到汇编调试窗口，芯片将执行一句反汇编指令。如图6-8所示。而在反汇编中执行"单步跳过"命令时，芯片将持续执行反汇编指令，直到堆栈级别小于或等于当前堆栈级别。这两种命令通常会在函数调用指令（CALL 、 LCALL、bl 等）处执行效果产生差别。

注意：可通过再次点击汇编按钮"切换到语句单步（

）"，退出汇编调试进入源代码调试状态。

在不需要执行反汇编调试的情况下，可以通过执行"显示反汇编代码"功能，直接打开汇编窗口。右键单击 `main.c` 文件中的第87行，选择"显示反汇编代码"，将会打开反汇编窗口，并跳转至改行代码对应的反汇编处，并且对此行代码进行蓝色高亮标注。如图6-8所示。

```
27   ;c:\Users\Dengym\Desktop\M950\ADC\Src\main.c
28   ;Line 44:   stu1.age = 16;
29   0x0000025E    movw r0, #68  ; 0x44
30   0x00000262    movt r0, #8192    ; 0x2000
31   0x00000266    movs r1, #16
32   0x00000268    str r1, [r0, #8]
33   ;Line 45:   stu1.group = 'A';
34   0x0000026A    movs r1, #65  ; 0x41
35   0x0000026C    strb r1, [r0, #12]
36   ;Line 46:   stu1.num = 1;

.....

19   ;Line 87:   init_str();
20   0x000002C2    str r0, [sp, #52] ; 0x34
21   0x000002C4    bl 0x25e <init_str>
22   ;Line 88:       for (i = 0; i < 30; i++) {
23   0x000002C8    ldr r0, [sp, #52] ; 0x34
24   0x000002CA    str r0, [sp, #296]    ; 0x128
25   0x000002CC    b.n 0x2ce <main+22>
```

图6-8 "调试反汇编"窗口

"`;Line 87: init_str();`"是之后一段反汇编程序所对应的源代码，87是源代码对应的行号。

在图6-8中，当程序运行到`0x000002C4`处时，执行"单步调试"命令，程序将会跳转到`0x0000025E`。而在`0x000002C4`处执行的是"单步跳过"命令时，程序将直接在`0x000002C8`处暂停。

6.8 可视化调试

VSCode为用户提供了对变量进行可视化监视的功能。其中，对于监视的变量存在以下三种不同情况：

- 若变量时基础变量，如`int`、`float`等，直接显示名称、类型、值。
- 若变量是数组类型，则以表格方式显示。若数组元素是基础变量，显示元素值；若数组元素是复杂变量，则以图表的形式显示元素类型。
- 若变量时指针、结构体、联合体，以图的方式显示，展开变量所有成员。若遇到指针成员，会尝试分析所有局部变量、全局变量，找到指针指向的真实变量，并将其显示展开。

注意：8位芯片不支持表达式计算，监视数组元素只支持常量下标。

在调试本项目时，希望能够更加直观地监视main.c文件中局部的结构体变量`student1`,可通过可视化调试功能以图的方式实时显示`student1`的成员变量。在调试处于中断状态下，打开命令面板（`Ctrl+Shift+P`），输入"`Debug Visualizer:New View`"命令，打开Debug Visualizer窗口，在输入框里输入`student1`并回车，即可在该窗口中以图的形式实时且直观地看到`student1`成员变量`name`、`num`、`float`以及结构体类型`birthday`的值，如下图6-9的左半图所示。其中，变量`birthday`为结构体类型，即也展开显示其成员变量`year`、`month`、`day`的值。

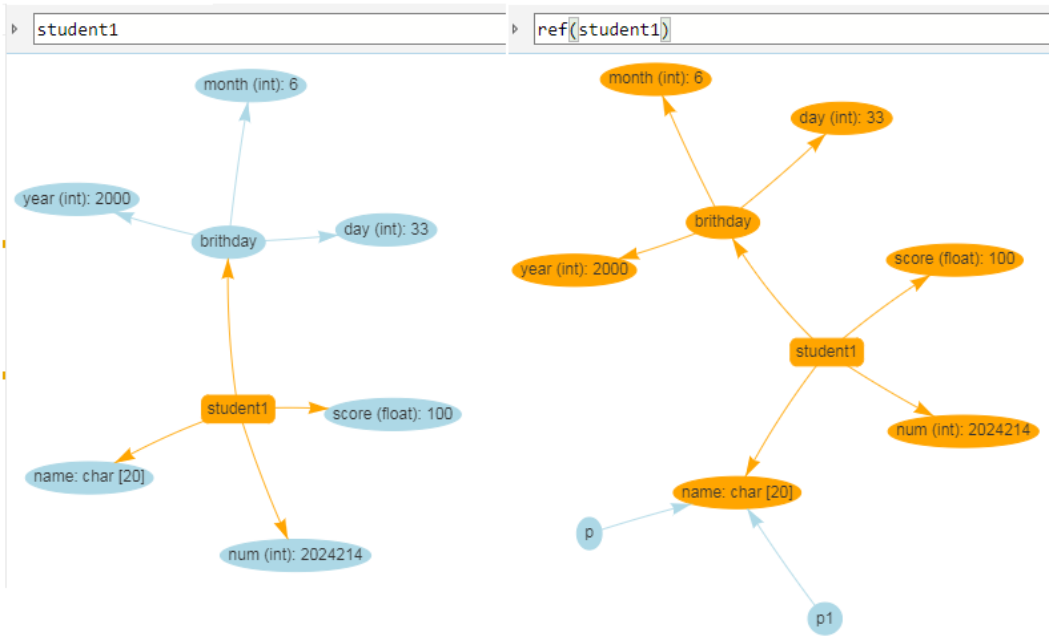

图6-9 可视化监视

有时为了能够更好地分析变量，需要知道所有指向该变量内存的指针变量有哪些。可以通过在输入栏输入`ref(变量名)`,即可查找到所有指向该变量内存的指针。如图6-9右半图所示，在输入栏输入`ref(student1)`并回车，即可看到指向`student1`的指针有`p1`、`p2`且都指向`student`的第一个成员变量`name`。