FDMTEK
中科本原

数字信号处理器及其解决方案提供商

CodeCanvas 使用手册

版本 V1.2

青岛本原微电子有限公司

目录

1 CodeCanvas 概述

CodeCanvas 是针对中科本原公司 RV 系列处理器的简单易用集成开发环境(IDE)。该 IDE 基于 Eclipse™，采用最新一代成熟的代码生成工具，提供汇编、C/C++语言的编辑、编译和调试功能。

CodeCanvas 还为开发人员提供驱动库和高性能软件库的高度集成插件支持。此类支持包括外设的驱动器支持、针对目标核专门优化的高性能函数库等。CodeCanvas 为用户提供了易于使用的开发工具，例如工具链、调试器、Flash 烧录工具。直观的 IDE 将引导开发人员完成应用开发流程的每个步骤，熟悉的工具和界面使入门变得简单。

1.1 优势和特点

- 无需安装，解压即用
- 基于 Eclipse 的集成开发环境(IDE)
- 出色的开发与调试支持
- 出色的代码生成工具，包括编译器、汇编器、链接器和加载器
- 成熟、极致优化的高性能函数库
- 自带外设使用例程，轻松上手

1.2 系统要求

Windows 10 Professional/Enterprise/64 位。

建议使用最低为 2 GHz 的单核处理器或最低 3.3 GHz 的双核处理器。

存储器(RAM)空间不低于 1 GB，建议采用 4 GB 存储器。

要求硬盘(HDD)空间不低于 2GB。

1.3 安装使用

本软件为免安装版本，用户将安装包解压至本地目录中即可使用，无需执行额外的安装步骤。

注意事项：

1. 操作系统使用 Windows 10 Professional/Enterprise64 位。

2. 软件解压路径不要有中文、空格或特殊字符。

2 创建工程

2.1 创建工程

有三个创建工程的入口。第一次运行 Eclipse 时，可在左侧工程浏览窗口中点击“CC Project”链接（如图 2-1）打开创建工程向导；也可依次选择菜单栏中的“File”“New”“CC Project”（如图 2-2）；还可点击工具栏“New”右侧的倒三角并选择“CC Project”（如图 2-3）。

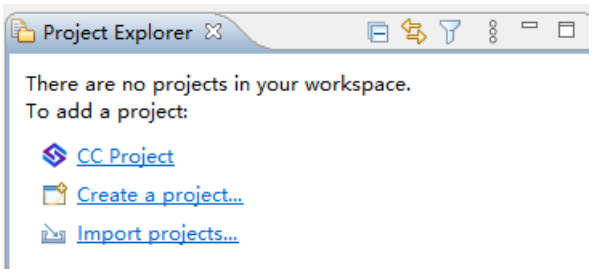


图 2-1 “CC Project”链接

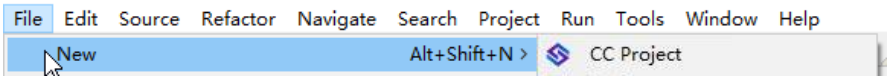


图 2-2 创建工程

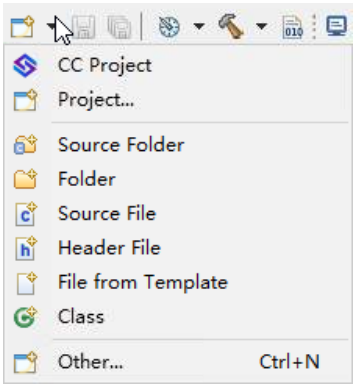


图 2-3 创建工程

如图 2-4，在弹出界面输入工程名，输出类型，工程语言，工具链类型，产品型号，然后点击“Finish”按钮，IDE 将自动创建对应配置的工程。如果勾选了“with driver support”，生成工程时会在“driverlib”文件夹下生成外设使用相关代码。

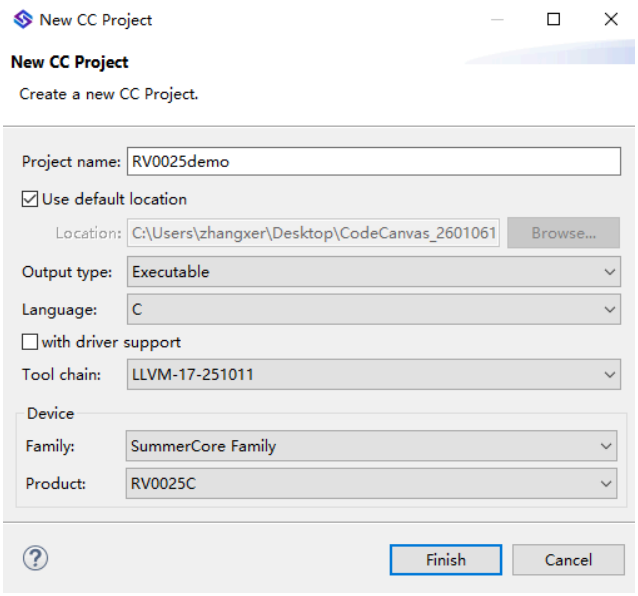


图 2-4 创建工程

新创建的工程将在工程浏览窗口看到，如图 2-5。

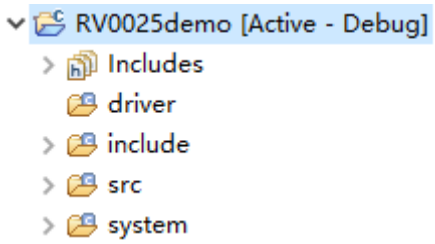


图 2-5 新工程

2.2 导入第三方工程

首先在第三方中构建（build）目标工程，保障可以生成目标文件，然后在 CodeCanvas 中执行如下操作：

点击菜单栏的“File”=>“Import”，选择“Code Canvas”=>“C Project from CCS”，进入导入 CCS 工程页面，在该页面选择待导入的 CCS 工程路径，以及是否需要外设驱动库支持。

- “CCS Workspace”：设置要导入的第三方工程路径或 workspace 路径，设置后会在“Available Projects”下展示指定目录中可用的 CCS 工程。
- “SDK Directory”：SDK 包含了本原芯片的函数库、Flash API 库、外设驱动以及例程，此处请填写安装到本地电脑上的 SDK 安装路径。
- “With driver support”：当勾选时，在本原工程中会自动添加驱动库支持。

- “Available Projects”：当选择“CCS Workspace”后，会自动识别出第三方工程，勾选要导入的工程。

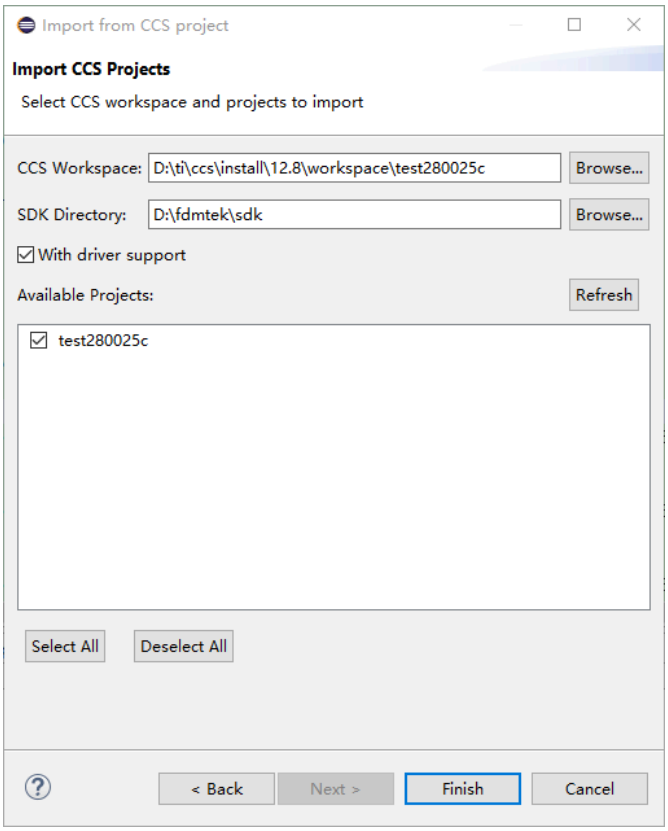


图 2-6 导入工程

填写上述信息后，点击“Finish”按钮，开始进行第三方工程导入，之后会显示转换进度，如图所示。

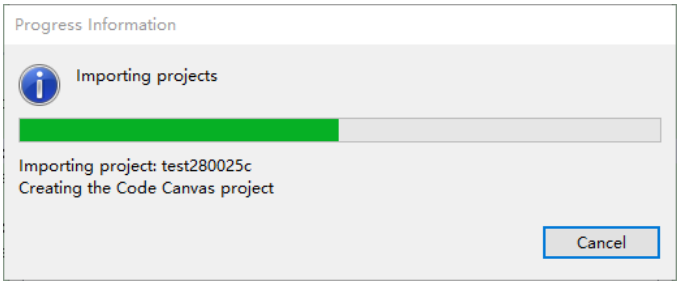


图 2-7 导入工程

当运行结束后，IDE 完成了第三方工程的导入，以及自动转换为 CodeCanvas 工程的工作。下图为转换后的工程结构。

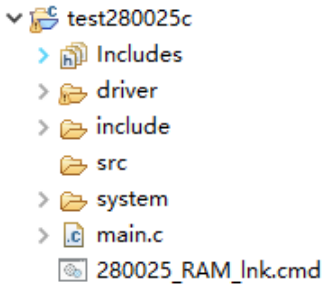


图 2-8 RV0025 工程

不支持转换的文件会在“Console”视图中以红色字体输出，予以提示。

3 编译工程

CodeCanvas 使用 Make 进行工程编译管理，在构建工程时，默认情况下会自动生成 makefile 文件，开发人员无需担心文件的添加、修改和删除，Make 采用增量编译，只对上一次编译后发生了改动的文件进行编译，未发生改动的文件不会被编译，节省了编译时间。

3.1 编译配置

步骤一：先选中工程，然后点击鼠标右键，选择“Properties”。

步骤二：在弹出界面的左侧导航列表中，依次选择“C/C++ Build”、“Settings”。在右侧的“Tool Settings”选项卡，可以对编译器、汇编器、链接器的相关配置进行修改，如图 3-1。

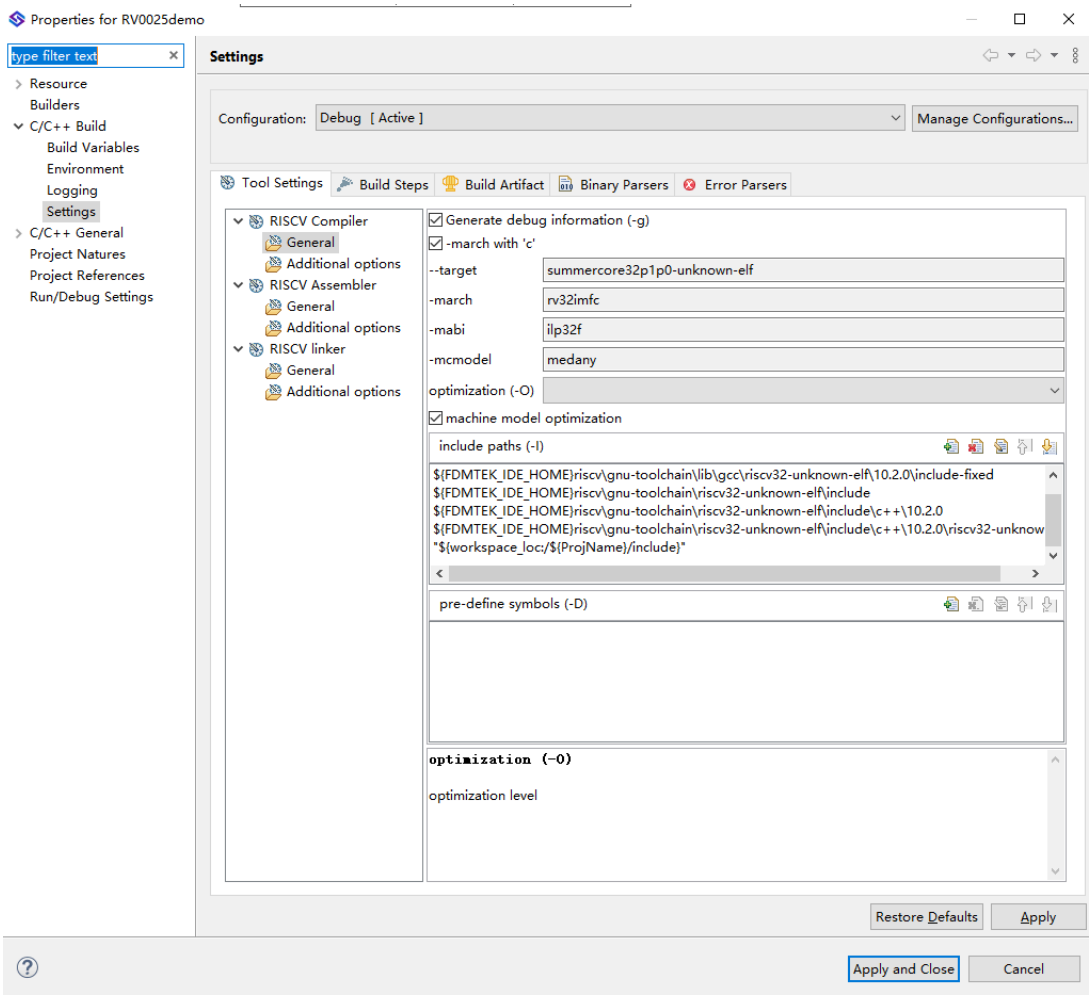


图 3-1 编译配置

如果希望在编译过程中链接静态库，需要在链接器中进行配置。右键选择要配置的工程，依次点击“Properties”=>“C/C++ Build”=>“Settings”=>“RISCV linker”=>“General”，打开链接器配置界面，如图 3-2 所示。在“Library search path”中配置静态库所在路径，在“Libraries”中配置静态库名字。注意静态库名称通常为 libxxx.a，配置时只写 xxx；例如要链接 libc.a 库，这里只填写 c。

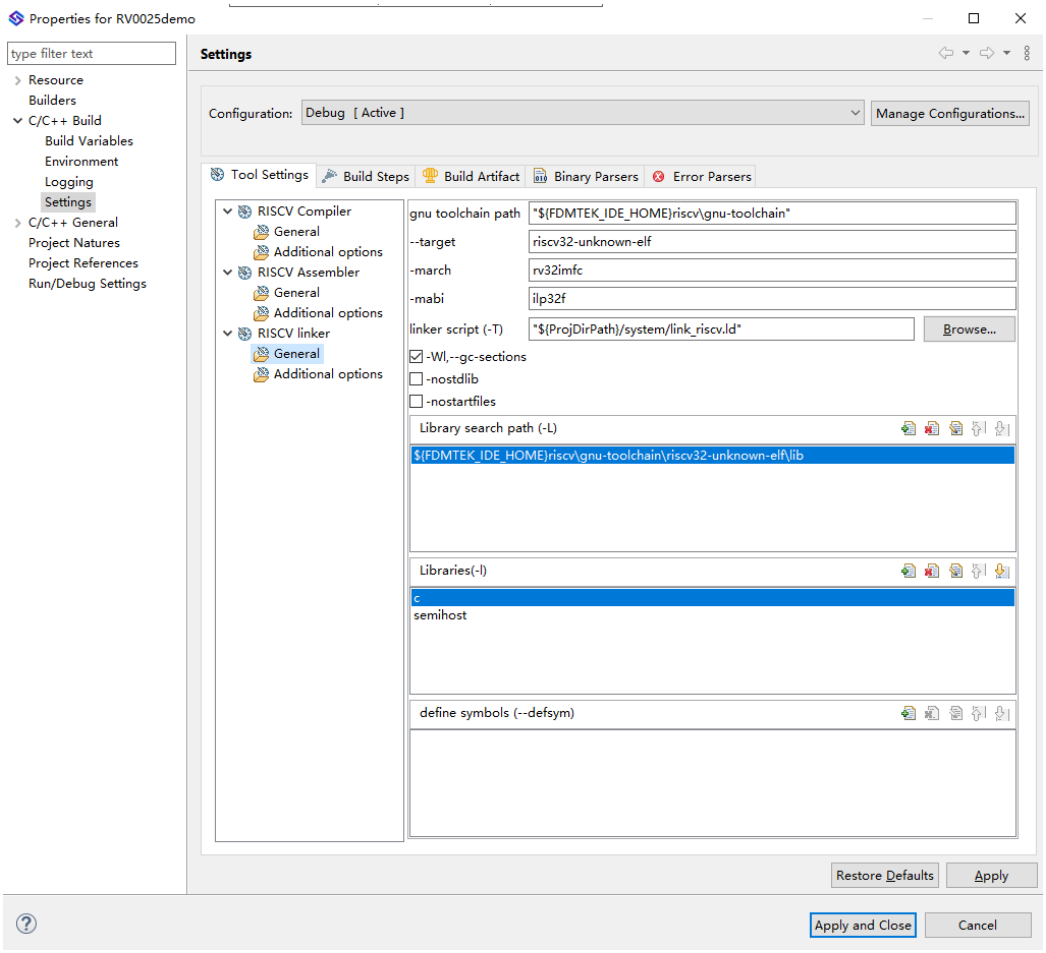


图 3-2 静态库链接配置

3.2 编译（Build）

IDE 有“Debug”和“Release”两种编译方案，“Debug”方案生成的目标文件包含调试信息，适合用于代码开发过程中的调试。“Release”方案生成的目标文件不含调试信息，因此体积相对小一些，适合用于发布目标文件。用户可以通过工具栏 图标右侧的三角形选择编译方案。

编译工程有两种方法：选中工程，点击鼠标右键，然后选择“Build Project”（如图 3-3 所示）；或点击工具栏中的“Build”图标。

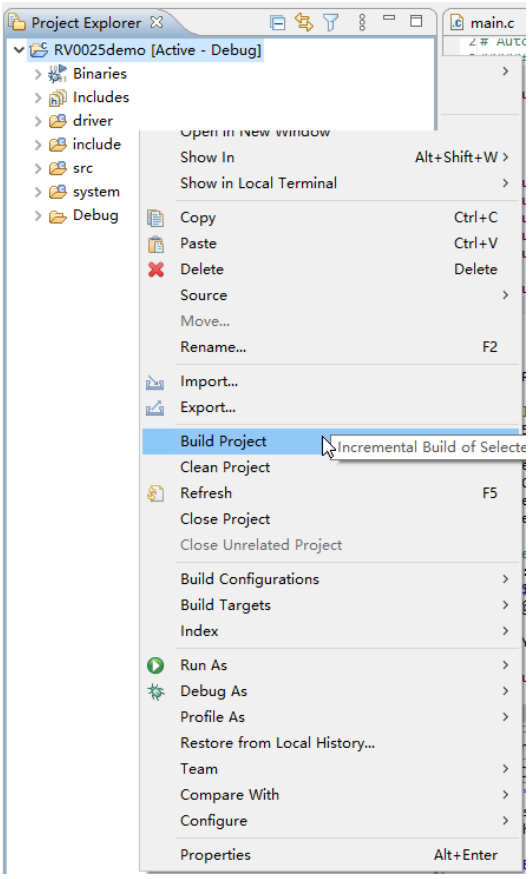


图 3-3 编译

如果编译成功，可以在工程浏览框中发现工程下多了 Binaries 目录，展开目录可以看到生成的目标文件。

3.3 清空（Clean）

如果要删除之前编译生成的所有目标代码文件，可以使用清空功能，有如下两种方法。

第一种：在工程浏览框中选中目标工程，然后点击鼠标右键，选择“Clean Project”，如图 3-4 所示。

第二种：在菜单栏，选择“Project”，然后选择“Clean…”，再在弹出的对话框中选择目标工程，然后点击下方的“Clean”按钮，如图 3-5 所示。

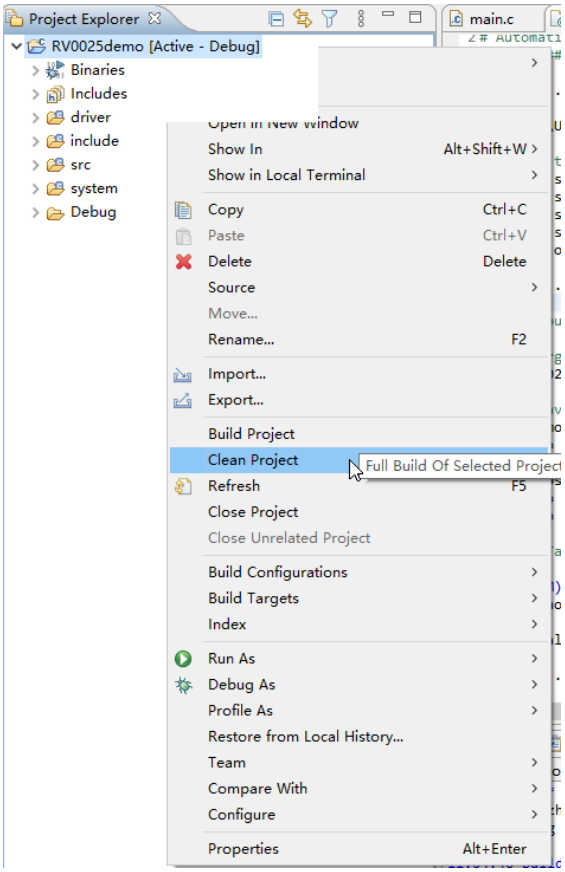

Project Explorer
main.c
RV0025demo [Active - Debug]
Binaries
Includes
driver
include
src
system
Debug
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Source
Move...
Rename... F2
Import...
Export...
Build Project
Clean Project
Full Build Of Selected Project
Refresh F5
Close Project
Close Unrelated Project
Build Configurations
Build Targets
Index
Run As
Debug As
Profile As
Restore from Local History...
Team
Compare With
Configure
Properties Alt+Enter

图 3-4 清除

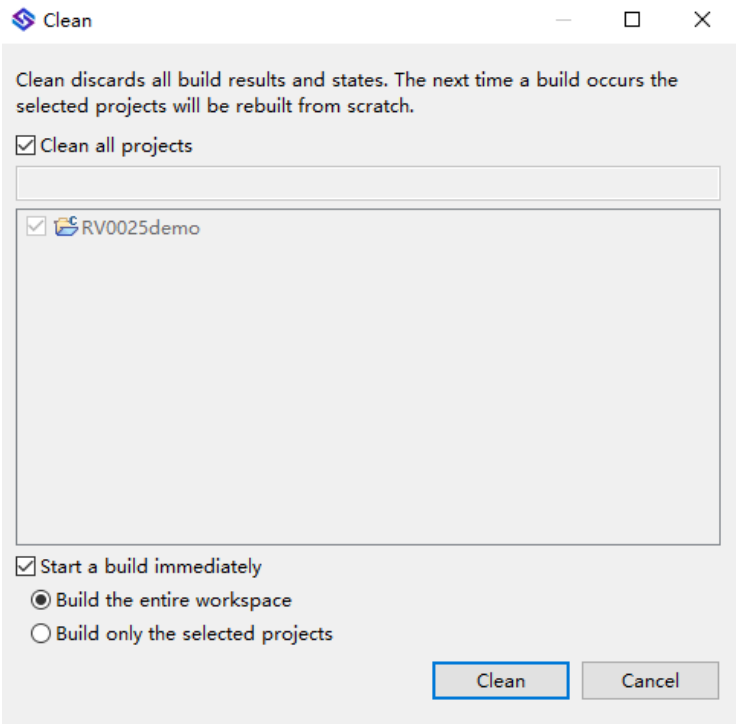

Clean
Clean discards all build results and states. The next time a build occurs the selected projects will be rebuilt from scratch.
Clean all projects
RV0025demo
Start a build immediately
Build the entire workspace
Build only the selected projects
Clean
Cancel

图 3-5 清除

4 调试

有三种进入调试的方法，在使用前两种方法前，请确保目标工程已经生成了目标文件。下面依次讲解。

第一种方法：在工程浏览框中选择目标工程，然后点击鼠标右键，依次选择“Debug As”、“Debug Configurations”。

第二种方法：点击工具栏 图标右侧的三角形，然后选择“Debug Configurations”。

第三种方法：右键点击工程，选择“Debug As”=>“Application with GDB and OpenOCD”。

4.1 创建调试配置

在前面弹出的窗口左侧列表中，选中“Application with GDB and OpenOCD”，点击鼠标右键并选择“New Configuration”。在右侧详情页面中可对配置进行重命名，如图 4-1 所示。

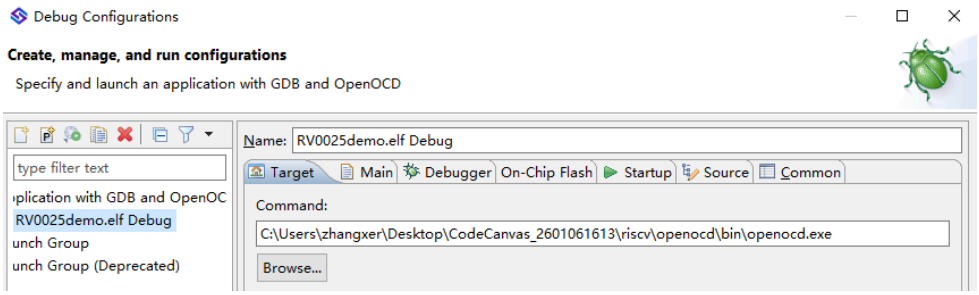


图 4-1 调试配置

4.2 配置 Target

CodeCanvas 会自动选择自带的 OpenOCD。如果没有特殊原因，开发人员无需修改该配置。

关于“Target（processor）”和“Board”选项，前者对应的 OpenOCD 配置文件仅包含核（core）配置，后者还包含外设配置。

如非特殊原因，不要修改该配置。

4.3 配置 Flash

在调试配置界面的“On-Chip Flash”选项卡下，可以对 Flash 进行配置，如图 4-2 所示。具体配置参见第 6 章。

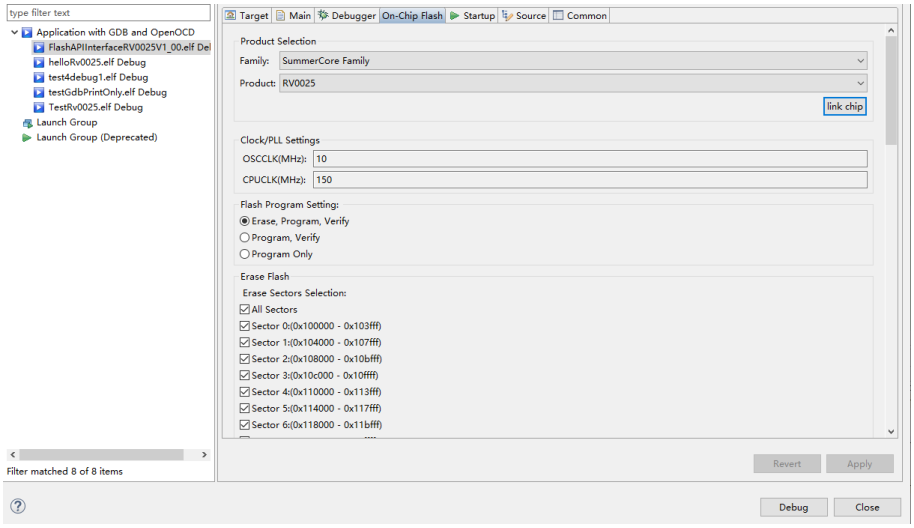


图 4-2 “On-Chip Flash”界面

启动调试时，IDE 会先检查目标芯片的 DCSM 状态。

如果目标芯片当前已解锁，则跳过解锁操作，继续执行后续调试流程；如果目标芯片已锁定，则使用当前“Debug Configuration”中“On-Chip Flash”保存的“CSMPSWD”密码进行解锁。解锁成功后继续加载被调试程序；解锁失败时，会给出提示并终止本次调试。

4.4 设置被调试程序

“Main”选项卡的默认内容会因调试配置的创建方式而异。使用“Debug As”创建配置时，IDE 会自动选择被调试工程及其目标文件；使用其他方式创建配置时，需要点击“Browse”按钮手动指定，如图 4-3 所示。

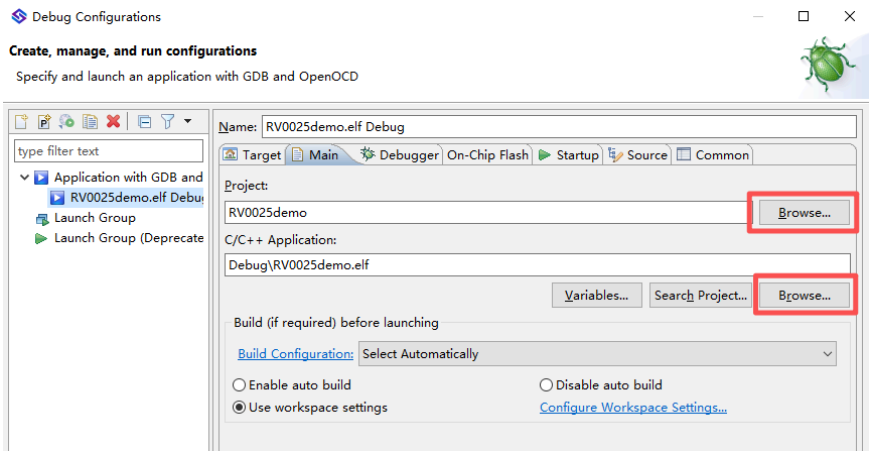


图 4-3 设置被调试程序

如果被调试程序是 Flash 程序，IDE 会使用“C/C++ Application”中指定的 ELF 文件自动烧录；完整流程和配置说明请参见第 6.3 节“Flash 程序自动烧录”。

4.5 配置 GDB

在“Debugger”选项卡，可以设置调试器和 JTAG 信息，如果没有特殊原因，无需修改，使用默认值即可，如图 4-4。

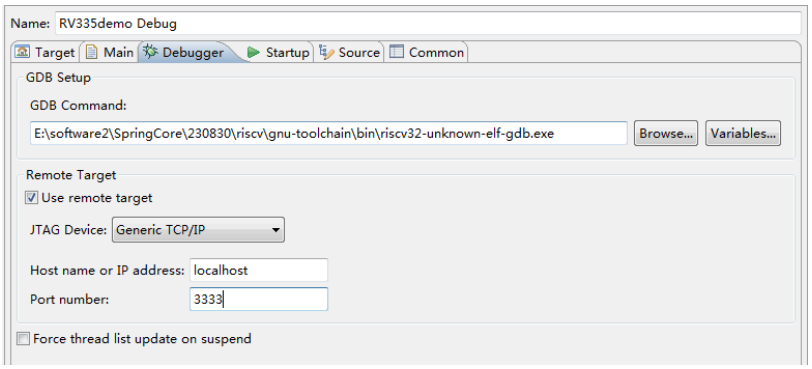


图 4-4 配置 GDB

4.6 调试动作

4.6.1 恢复（Resume）

恢复挂起的 core，让其全速运行。如图 4-5 所示。

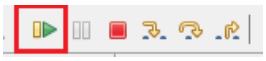


图 4-5 恢复

4.6.2 挂起（Suspend）

让全速运行的 core 挂起，暂停运行，可通过“Resume”继续全速运行。如图 4-6 所示。

图 4-6 挂起

4.6.3 终止（Terminate）

终止程序运行。如图 4-7 所示。

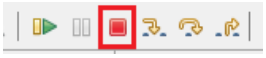


图 4-7 终止

4.6.4 步入（Step Into）

单步调试，如果遇到函数调用，则会进入函数内部。如图 4-8 所示。

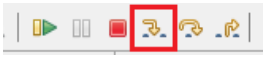


图 4-8 步入

4.6.5 步过（Step Over）

单步调试，如果遇到函数调用，将其视为一条普通的 C 语句，直接运行，不会进入函数内部。如图 4-9 所示。

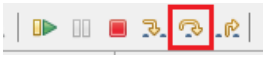


图 4-9 步过

4.6.6 返回（Step Return）

从当前位置全速运行，直到函数调用者（caller）处。如图 4-10 所示。

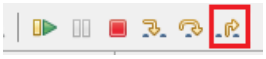


图 4-10 返回

4.6.7 重新开始（Restart）

该功能的效果为重新加载被调试程序，外设寄存器不会修改/重置。如图 4-11 所示。

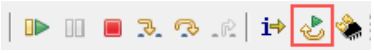


图 4-11 重新开始

4.6.8 复位（Reset）

该功能的效果为重置整个芯片，相当于芯片重新上电，但调试状态不会断开，如图 4-12 所示。

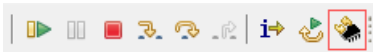


图 4-12 复位

4.7 断点

“断点”是指在程序执行过程中设置的一个特殊的位置，用于暂停程序的执行，以便用户可以检查程序的状态、变量的值和执行路径。当程序执行到断点位置时，程序会停止执行，进入调试模式。在调试模式下，用户可以逐行执行代码，观察变量的值，查看函数的调用栈，以及进行其他调试操作。通过断点，用户可以逐步跟踪程序执行流程，观察变量值的变化，判断程序逻辑是否正确，从而定位和修复问题。断点是调试中的重要工具，对于复杂的程序调试和错误排查非常有帮助。

在 CodeCanvas 中，断点相关操作主要包括创建、启用、禁用和删除断点等。根据地址空间的不同可分为软件断点和硬件断点。用户不需要手动区分断点类型，IDE 会在调试状态下根据断点的地址自动判断创建软件断点或硬件断点。

4.7.1 创建断点

在编辑器中，双击行号即可在当前位置设置断点，创建断点后，编辑器在断点位置显示蓝色断点图标，如图 4-13、图 4-14 所示。

```
3  int add(int x, int y){
4      int ret = x + y;
5      return ret;
6  }
7
```

图 4-13 创建软件断点

```
19 void flash_test_func2(void){
20     g_counter+=2;
21     g_result = g_counter+2;
22 }
```

图 4-14 创建硬件断点

在非调试状态下，由于 IDE 无法判断断点对应的实际运行地址，创建的断点默认为软件断点。在调试状态下，IDE 会根据断点对应的运行地址自动判断断点类型。该判断逻辑同样适用于反汇编视图。

硬件断点使用芯片内部的断点资源，不同型号芯片支持的硬件断点数量不同。

当已启用的硬件断点达到当前芯片支持的最大数量后，继续创建或启用硬件断点时，IDE 会提示硬件断点数量已达到上限，超出数量的硬件断点将保持禁用状态。可以先禁用或删除启用状态的硬件断点，释放硬件断点资源后再启用或创建硬件断点。

软件断点不占用硬件断点数量。

4.7.2 启用、禁用断点

在“Breakpoints”视图下展示所有断点。默认情况下，打开“Debug”透视图后，断点视图在右侧展示；如果没有，可以通过菜单栏“Window”=> “Show View”=> “Breakpoints”调出。

启用和禁用断点有两种方式：

第一种方式是在断点视图中，通过勾选或取消勾选断点前的选择框可以启用或禁用相应断点。如图 4-15 所示。

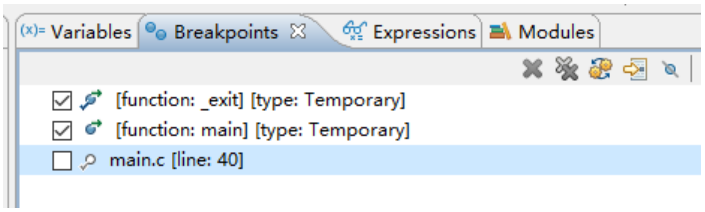


图 4-15 启用/禁用断点

第二种方式是在代码断点处右击，选择“Enable Breakpoint”或“Disable Breakpoint”，如图 4-16 所示。

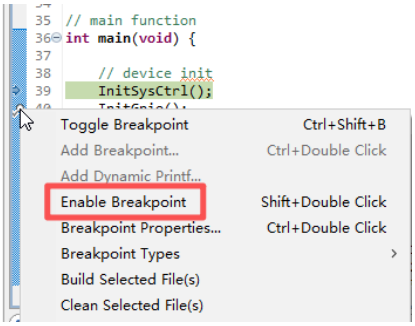


图 4-16 启用/禁用断点

对于软件断点，启用和禁用操作按普通断点流程执行。

对于硬件断点，启用时会受到当前芯片支持的最大硬件断点数量限制。已启用的硬件断点数量未达到芯片上限时，可以正常启用；达到芯片上限时，IDE 会弹出提示，并将该断点加入断点视图且保持禁用状态。禁用已启用的硬件断点后，会释放一个硬件断

点资源；此前因数量超限而处于禁用状态的硬件断点不会自动启用，如需启用，需由用户手动操作。

4.7.3 删除断点

在断点视图中，选中要删除的断点后，点击工具栏的“Remove Selected Breakpoint”图标，即可删除相应断点。用户也可在编辑器中双击已有断点位置的行号，删除该位置断点。

在断点视图中，点击工具栏的“Remove All Breakpoints”图标，即可删除所有已创建的断点。

对于硬件断点，如果删除的是已启用状态的硬件断点，会释放一个硬件断点可用数量；如果删除的是禁用状态的硬件断点，则不影响当前已启用硬件断点数量。

4.7.4 断点创建规则

CodeCanvas 支持自动硬件断点功能。在调试状态下用户创建断点时，IDE 会根据断点所在内存地址自动判断断点类型，无需用户手动选择软件断点还是硬件断点。

表 4-1 断点创建规则

当前状态	断点所在位置	断点类型
非“Debug”状态	无法判断地址空间	默认创建软件断点
“Debug”状态	Flash 地址空间	自动创建硬件断点
“Debug”状态	非 Flash 地址空间	创建软件断点

其中非“Debug”状态下创建的断点，虽然默认是软件断点，但是在进入“Debug”状态后，会再次根据内存地址空间进行转换。

4.8 调试视图

调试视图在通过 CodeCanvas 调试程序时起到了关键作用。它为用户提供了一个集中管理和查看程序执行过程中的关键信息的界面。通过调试视图，用户可以同时查看变量的值、调用栈、内存与寄存器信息等，从而帮助定位和解决程序中的错误。

常用的调试视图主要包括调用栈视图、反汇编视图、变量视图、表达式视图、内存视图和寄存器视图等。在“Debug”透视图下，可以通过菜单栏“Window”=> “Show View”调出所需要的调试视图。

4.8.1 调用栈视图

调用栈视图用于显示程序的调用栈信息。调用栈是一个记录函数调用关系的数据结构，用于追踪程序执行过程中的函数调用和返回。在调试过程中，通过查看调用栈视图，用户可以了解当前程序执行到哪个函数，以及该函数是被哪个函数调用的。

以 main 函数调用 add 函数为例，调用栈视图如图 4-17 所示。越处于外层的函数，在视图中的位置越在下方。其他视图默认展示当前代码所在函数的上下文信息，通过点击对应的函数，可以改变其他视图的展示信息对应当前选中的函数。

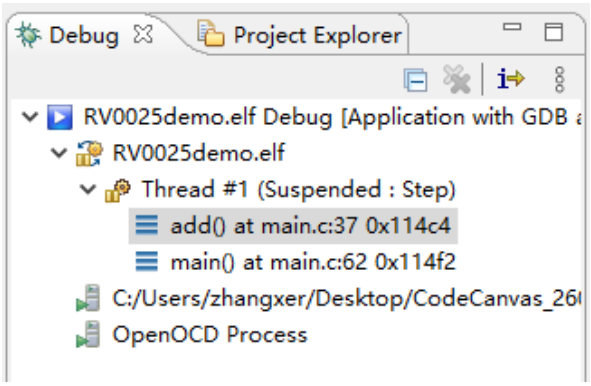


图 4-17 栈视图

4.8.2 反汇编（Disassembly）视图

反汇编用于显示程序的汇编指令。通过查看反汇编视图，用户可以了解程序的低级执行过程，分析程序的逻辑和性能。

可以通过菜单栏“Window”=>“Show View”=>“Disassembly”打开反汇编视图，如图 4-18 所示。

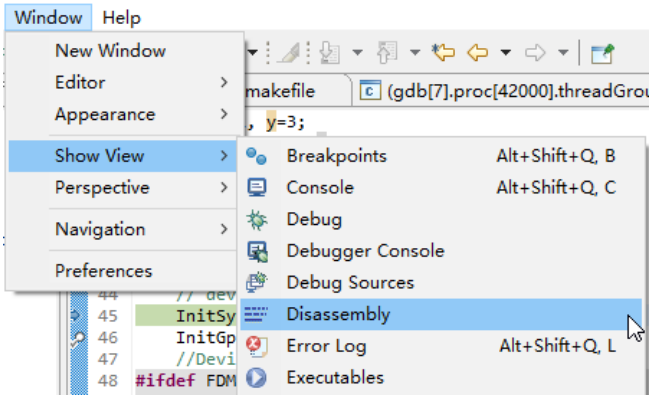


图 4-18 反汇编视图开启

如图 4-19 所示，反汇编视图主要包括以下几个部分：

1.内存地址：每条汇编指令的内存地址显示在左侧，方便用户定位到具体的指令。

2.汇编指令：每条汇编指令显示为文本形式，表示程序中的机器码指令。

3.符号名称：如果程序中的某个地址对应于已知的符号（如函数名、变量名），则反汇编视图可能会显示符号名称，方便用户理解指令所在的上下文。

4.源文件信息：标记当前汇编代码所对应的源文件内容，方便用户理解当前程序的功能。

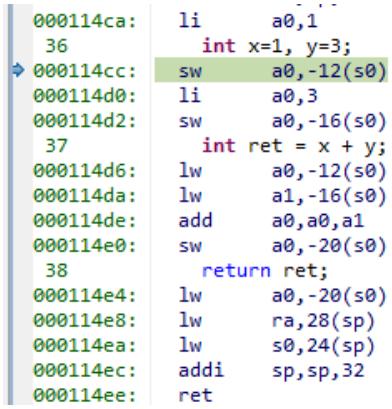


图 4-19 反汇编视图

反汇编视图的工具栏包含一些常用功能方便用户更好的使用反汇编相关功能，如图 4-20 所示。

1.地址跳转：在文本框输入地址后敲击回车键，可以展示特定地址的反汇编信息。

2.返回当前位置：在用户查看其他位置反汇编信息后，点击“Home”按钮，可以使反汇编视图快速回到程序当前运行位置。

3.链接源文件：切换是否要在反汇编视图中展示源文件内容。

图 4-20 反汇编视图工具栏

4.8.3 变量（Variables）视图

变量视图用于显示程序中各个变量的当前值。通过变量视图，用户可以实时监测、查看和修改变量的值，帮助调试和分析程序的状态。如图 4-21，在变量视图中，显示以下信息：

1.变量名称：每个变量在变量视图中以其名称显示，方便用户识别和查找。

2.变量类型：变量视图还显示每个变量的数据类型，例如整数、字符、数组等。

3.变量值：变量视图显示每个变量的当前值。这些值可以是整数、浮点数、字符串等数据类型的实际值。

如果希望修改变量的值，可以点击对应的变量值，输入新值后敲击回车即可完成对变量值的修改。

点击选中变量可以展示多种格式的变量值。点击工具栏上 按钮，可以修改数据展示的默认格式和修改变量视图的布局方式。

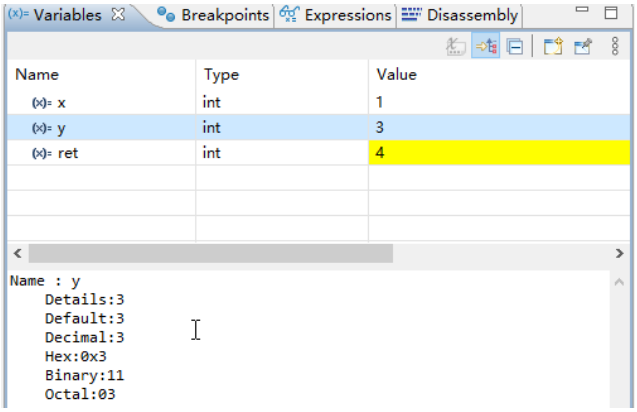


图 4-21 变量视图

4.8.4 表达式（Expressions）视图

表达式视图用于评估和监视表达式的值。通过表达式视图，用户可以输入和计算特定的表达式，并实时查看表达式的值，帮助调试和分析程序的状态。它对于动态计算和跟踪表达式的值非常有用，提供了更深入的调试和分析能力。

表达式视图如图 4-22 所示，包含三列，与变量视图类似，分别是表达式，类型，值。

通过点击“Add New Expression”添加新的表达式。表达式可以是变量、常量、数学运算、函数调用、寄存器或内存等；输入完成后，敲击回车即可展示表达式的信息。当表达式的值发生变化时，会以黄色背景展示。

常见表达式示例如下：

g_value

g_array[0]

g_point.x

g_complex.points[0].x

(volatile int32_t)0x20000100

其中，固定地址表达式需要按照“*(类型*)地址”的格式输入，例如：`*(int32_t*)0x20000100`。

如果希望修改表达式的值，只需要点击对应表达式值，输入新之后敲击回车，即可完成对表达式值的修改。注意，涉及到数学运算的表达式无法修改值。

与变量视图类似，点击工具栏上 按钮，可以修改数据展示的默认格式和视图的布局方式。

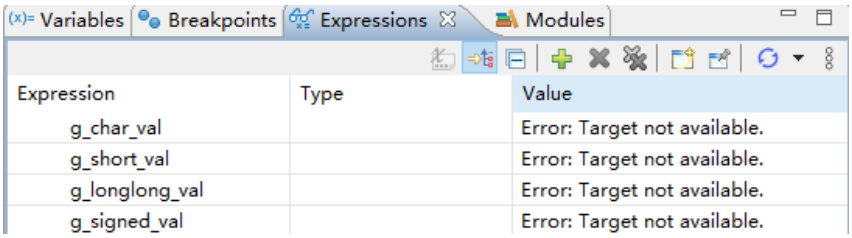


图 4-22 表达式视图

表达式视图支持“Live Refresh”实时刷新功能和单次刷新功能。进入“Debug”状态后，表达式视图工具栏中的“Live Refresh”图标按钮变为使能状态，默认处于关闭状态。该按钮由图标主体和右侧下拉小三角组成，如图 4-23 所示。

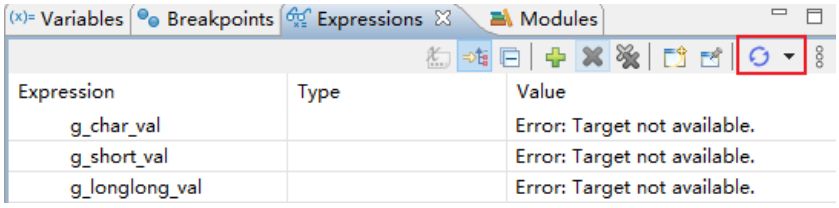


图 4-23 “Live Refresh”图标

点击图标主体可开启或关闭实时刷新。开启后，按钮会切换为高亮状态，如图 4-24 所示，表达式视图会按照设定的刷新间隔周期性更新支持实时刷新的表达式值；再次点击图标主体可关闭实时刷新，并恢复普通图标状态。

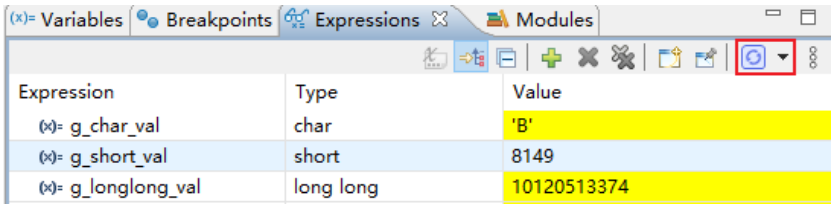


图 4-24 “Live Refresh”开启状态

点击右侧下拉小三角可打开刷新间隔菜单，支持选择 300ms、500ms、800ms、1000ms、2000ms 等刷新间隔，默认刷新间隔为 500ms，如图 4-25 所示。用户可在刷新过程中切换刷新间隔，切换后后续刷新会按照新的间隔执行。

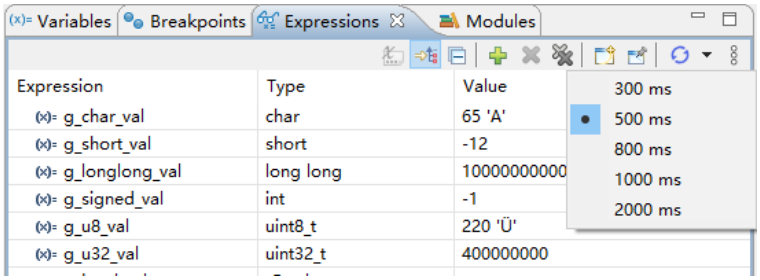


图 4-25 “Live Refresh”刷新间隔下拉菜单

“Live Refresh”开启后，对于可以解析为固定地址和固定大小的表达式，可在目标程序“Running”状态下持续刷新显示，例如全局变量、静态变量、数组元素、结构体字段、结构体数组字段以及明确类型的固定地址表达式等。该功能适用于程序运行过程中需要持续观察变量变化的场景。

对于数组和结构体表达式，用户可以展开节点查看具体元素或字段值。对于大于 4KB 的数组，表达式视图会按范围分段显示，例如[0..99]、[100..199]，继续展开范围节点后可查看具体数组元素，避免一次性展开大量数据导致界面卡顿。

“Live Refresh”支持对部分标量表达式进行写入，包括标量变量、数组标量元素、结构体标量字段以及结构体数组元素中的标量字段。数组整体和结构体整体不支持直接写入。

需要注意的是，局部变量、函数调用表达式、普通指针解引用、-> 指针成员访问、bitfield 等表达式不进入“Live Refresh”实时刷新路径，仍按照 CDT 原生表达式视图方式处理。

4.8.5 内存（Memory）视图

内存视图帮助开发人员在调试程序时查看和修改内存中的数据。如图 4-26 所示，内存数据作为一组内存监视器(Monitor)显示。每个监视器都代表由其位置（称为基地址）指定的内存部分。每个内存监视器都可以采用不同的预定义数据格式（内存呈示）显示。调试器支持五种呈示类型，即十六进制（缺省）、ASCII、有符号整数、无符号整数和十六进制整数。创建监视器时，将自动显示缺省呈示。

通过左侧“Monitors”旁的绿色加号 可以新建一个内存监视器。在弹出的对话框中输入要监看的内存基地址，点击“OK”后即可新建一个内存监视器以展示内存数据。

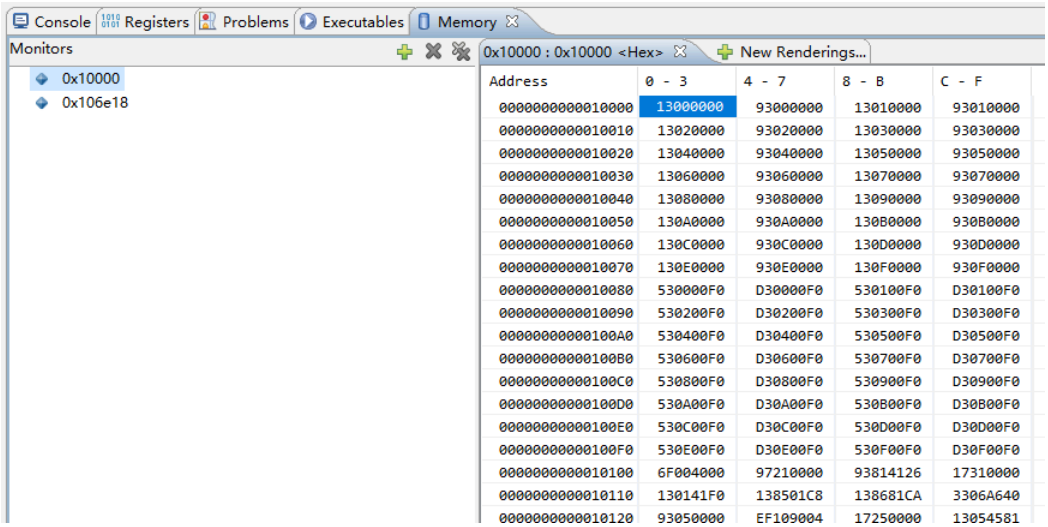


图 4-26 内存视图

通过点击上方的“New Renderings”可以选择展示其他格式的内存数据。如图 4-27 所示，选择所需格式后点击“Add Rendering(s)”按钮即可添加新的展示格式。

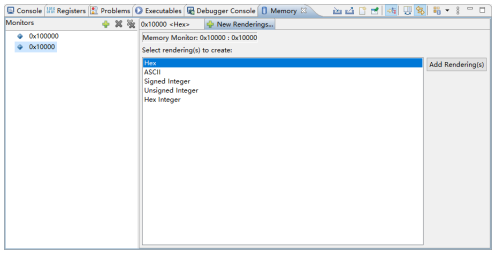


图 4-27 数据格式

在内存视图中点击右键，可以弹出上下文菜单。其中，可以通过“Format”调整内存每行数据量格式。

内存视图还支持内存数据导出和恢复功能。在进入调试状态后，内存视图工具栏会显示“Dump Memory”和“Restore Memory”功能按钮，如图 4-28 所示；在非调试状态下，由于未连接有效调试会话，相关按钮不可用。

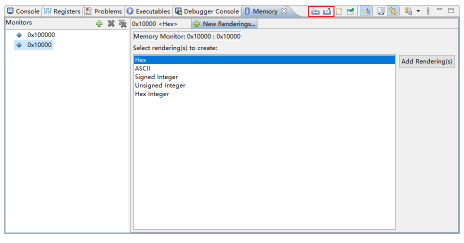


图 4-28 “Dump Memory”和“Restore Memory”按钮

“Dump Memory”用于将目标设备指定地址范围内的内存数据导出到本地文件。点击“Dump Memory”按钮后，会弹出“Dump Memory”配置窗口，如图 4-29 所示。用户

需要在窗口中设置输出格式、输出文件路径、起始地址和导出字节长度。其中，“Output type”用于选择导出文件格式，支持 verilog（hex text）、binary、ihex、srec、tekhex；“File name”用于指定导出文件保存位置；“Address”用于指定导出的起始内存地址，默认值为当前“Memory”视图正在浏览的地址；“Byte length”用于指定需要导出的字节长度。配置完成后点击“OK”，IDE 会将指定地址范围内的数据导出到本地文件。

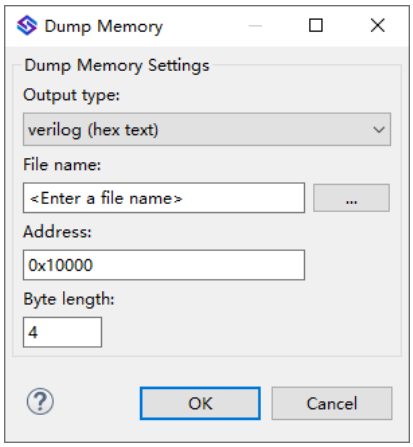


图 4-29 “Dump Memory”配置窗口

“Restore Memory”用于将本地文件中的数据恢复到目标设备内存中。点击“Restore Memory”按钮后，会弹出“Restore Memory”配置窗口，如图 4-30 所示。用户需要在窗口中设置输入格式、输入文件路径和目标地址。其中，“Input type”用于选择输入文件格式，支持 binary、ihex、srec；“File name”用于选择需要恢复的本地文件；“Address”用于指定数据写入的目标内存地址。使用 binary 格式进行恢复时，由于 binary 文件本身不包含地址信息，因此需要用户手动输入目标“Address”。使用 ihex 或 srec 格式进行恢复时，文件内部通常包含地址信息，IDE 会解析文件中的地址并作为默认恢复地址；用户也可以根据需要修改“Address”，将数据恢复到指定的目标内存位置。

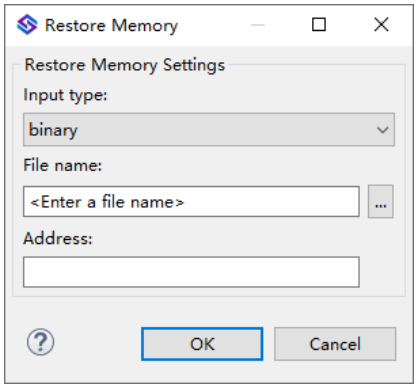


图 4-30 “Restore Memory”配置窗口

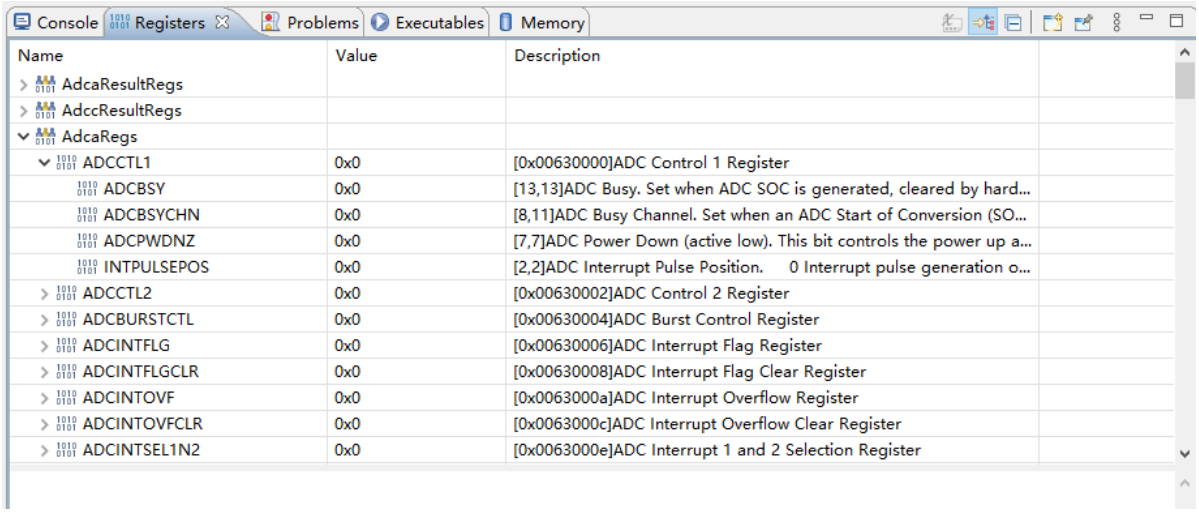
内存导出和恢复功能可用于保存调试现场、分析内存数据、对比程序运行前后的内存变化，以及将已保存的内存数据重新写入目标设备，便于问题复现和调试验证。

4.8.6 寄存器（Registers）视图

寄存器视图为调试提供了一个查看和修改内核寄存器的界面。在寄存器视图中，开发人员可以查看 CPU 或外设寄存器及其当前值。通过查看寄存器值，开发人员可以了解程序的当前执行状态，包括变量值、函数调用参数和返回值等。

寄存器视图如图 4-31 所示。共三列，包括寄存器名(或寄存器组名，寄存器位域名)，寄存器值和描述信息，点击选中寄存器可以在下方展示多种格式的值。发生变化的寄存器用黄色背景标注。如果要修改寄存器的值，可以点击相应寄存器的值，输入新值，敲击回车后即可完成对寄存器值的修改。

与变量和表达式视图类似，点击工具栏上 按钮，可以修改数据展示的默认格式和视图的布局方式。



Name	Value	Description
AdcaResultRegs		
AdccResultRegs		
AdcaRegs		
ADCCTL1	0x0	[0x00630000]ADC Control 1 Register
ADCBSY	0x0	[13,13]ADC Busy. Set when ADC SOC is generated, cleared by hard...
ADCBSYCHN	0x0	[8,11]ADC Busy Channel. Set when an ADC Start of Conversion (SO...
ADCPWDNZ	0x0	[7,7]ADC Power Down (active low). This bit controls the power up a...
INTPULSEPOS	0x0	[2,2]ADC Interrupt Pulse Position. 0 Interrupt pulse generation o...
ADCCTL2	0x0	[0x00630002]ADC Control 2 Register
ADCBURSTCTL	0x0	[0x00630004]ADC Burst Control Register
ADCINTFLG	0x0	[0x00630006]ADC Interrupt Flag Register
ADCINTFLGCLR	0x0	[0x00630008]ADC Interrupt Flag Clear Register
ADCINTOVF	0x0	[0x0063000a]ADC Interrupt Overflow Register
ADCINTOVFCLR	0x0	[0x0063000c]ADC Interrupt Overflow Clear Register
ADCINTSEL1N2	0x0	[0x0063000e]ADC Interrupt 1 and 2 Selection Register

图 4-31 寄存器视图

5 代码编辑

5.1 编辑器（Editor）

开发人员可以通过在“工程浏览”视图中，双击目标文件，在编辑器中会显示目标文件内容，开发人员可以在此对文件进行编辑，如图 5-1。

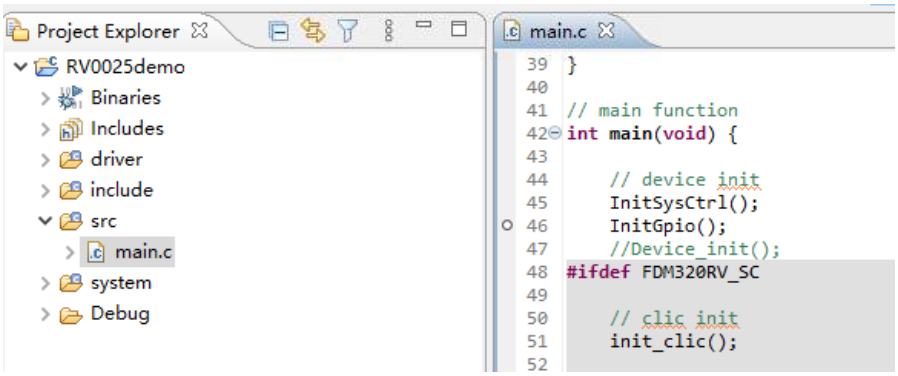


图 5-1 编辑器

5.1.1 字体

点击菜单栏选项“Window”，然后选择“Preferences”，在弹出的对话框左侧列表中，依次选择“General”、“Appearance”、“Colors and Fonts”，再在右侧的列表中，依次选择“Basic”、“Text Font”，然后点击右侧的“Edit…”按钮，如图 5-2。

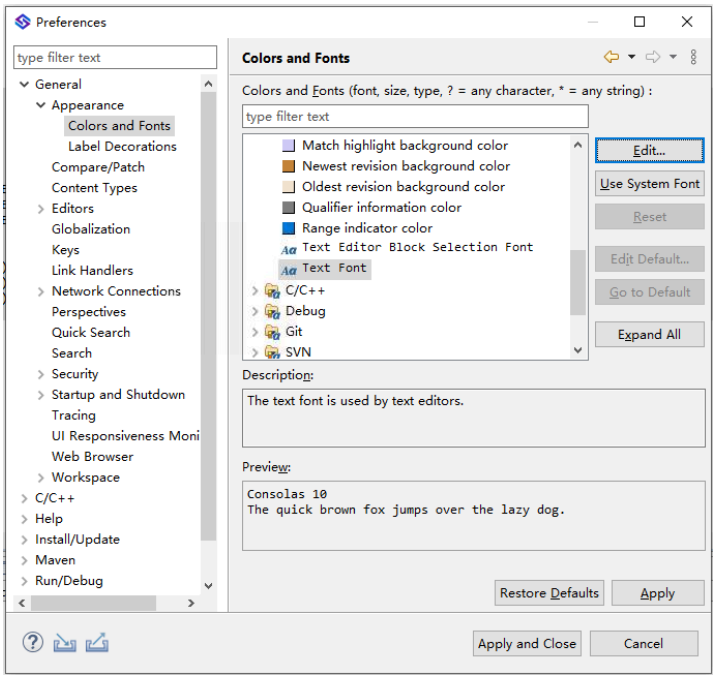


图 5-2 字体

在弹出界面可以对字体、字形和大小进行设置，如图 5-3，最后点击图 5-2 右下角的“Apply”按钮来应用修改。

图 5-3 设置

5.1.2 行号

点击菜单栏选项“Window”，然后选择“Preferences”。在弹出的对话框左侧列表中，依次选择“General”“Editors”“Text Editors”，然后在右侧配置项中勾选“Show line

numbers”以显示行号，取消勾选则不显示行号，如图 5-4 所示。最后点击下方的“Apply and Close”按钮应用修改。

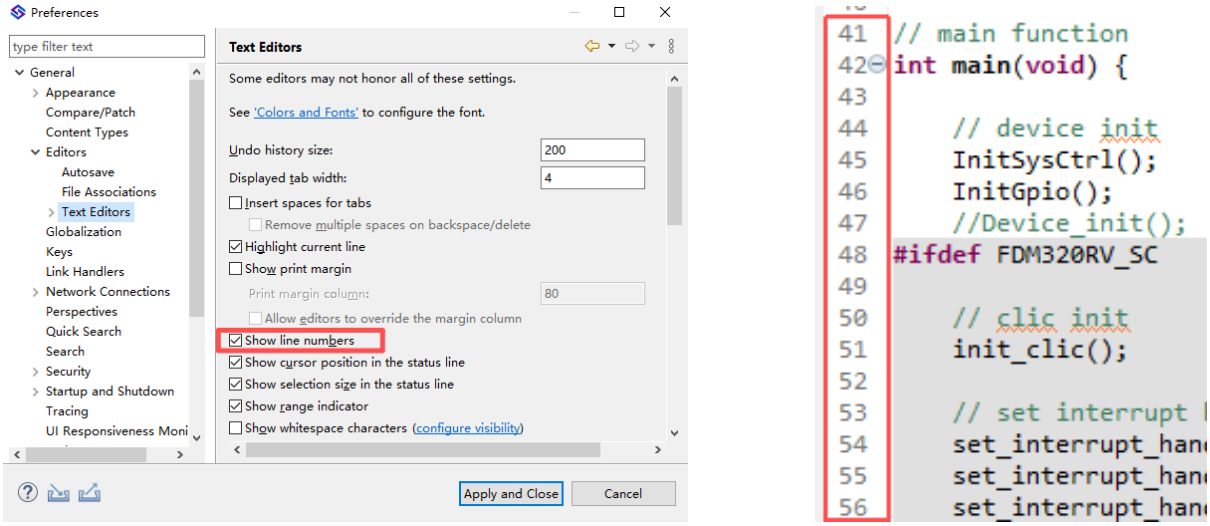


图 5-4 行号设置

5.1.3 背景色

点击菜单栏选项“Window”，然后选择“Preferences”，在弹出的对话框左侧列表中，依次选择“General”、“Editors”、“Text Editors”，然后在右侧配置项中，找到“Appearance color options”列表，在列表中选中“Background color”，然后点击右侧“Color：”旁边的颜色进行修改，如图 5-5，最后点击下方的“Apply and Close”按钮应用修改。

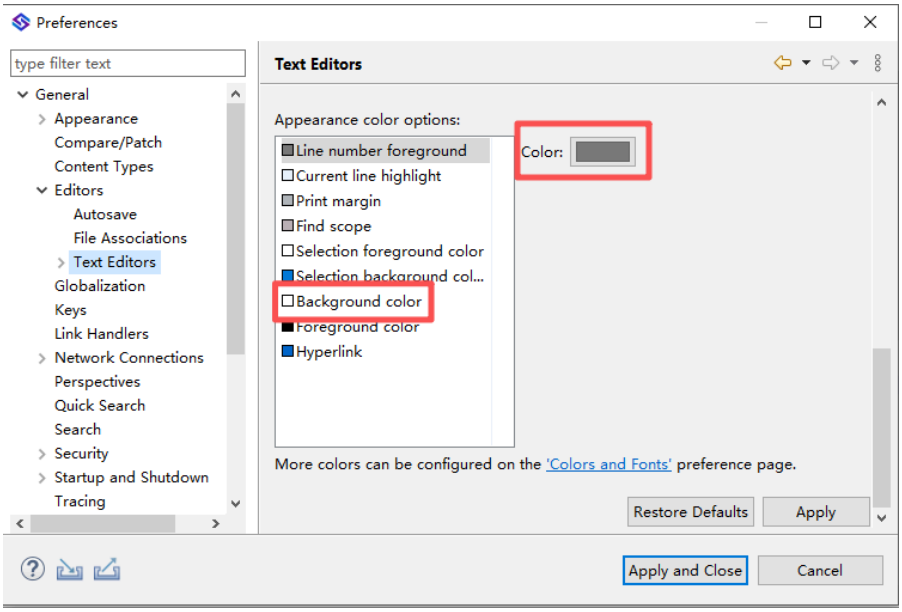


图 5-5 背景设置

5.1.4 空白字符

点击菜单栏选项“Window”，然后选择“Preferences”，在弹出的对话框左侧列表中，依次选择“General”、“Editors”、“Text Editors”，然后在右侧配置项中，勾选“Show whitespace characters”则显示空白字符（如图 5-6），反之则不显示，最后点击下方的“Apply and Close”按钮应用修改，如图 5-7。

```
41  // main function
42  int main(void) {
43      
44      // device init
45      InitSysCtrl();
46      InitGpio();
47      //Device_init();
48  #ifdef FDM320RV_SC
49  
50      // clic init
51      init_clic();
```

图 5-6 空白字符

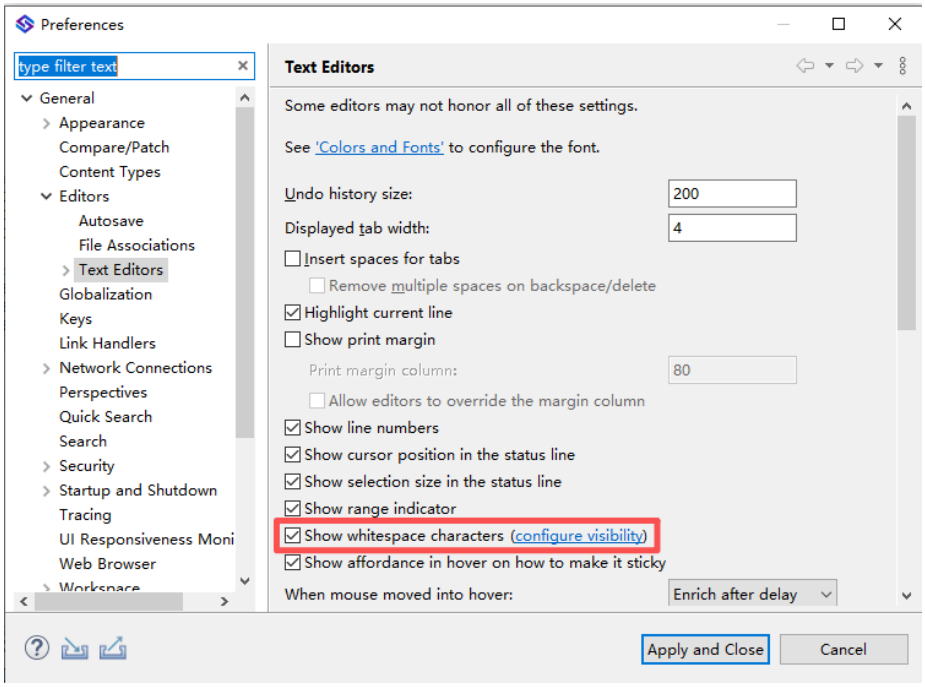


图 5-7 空白字符

5.2 透视图（Perspective）

Eclipse 的透视图提供了一种灵活的方式来组织和管理工作空间，以适应不同的开发任务和工作流程。通过自定义透视图的布局、工具栏和快捷键，可以提高开发效率并提供更好的用户体验。

透视图在 eclipse 的右上角展示，开发人员可以通过点击这些透视图图标来实现透视图切换，如图 5-8，也可以点击左侧的图标，在弹框中选择其他透视图。

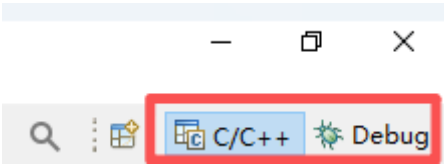


图 5-8 透视图

5.3 开发视图

5.3.1 工程浏览器（Project Explorer）

开发人员可以在工程浏览器中查看工程及工程内的文件，并执行文件的添加、修改、删除和重命名等操作；也可使用 Ctrl+C 复制文件，使用 Ctrl+V 粘贴文件。

通过点击图标（如图 5-9），开启编辑器中文件在工程浏览器中的自动定位。

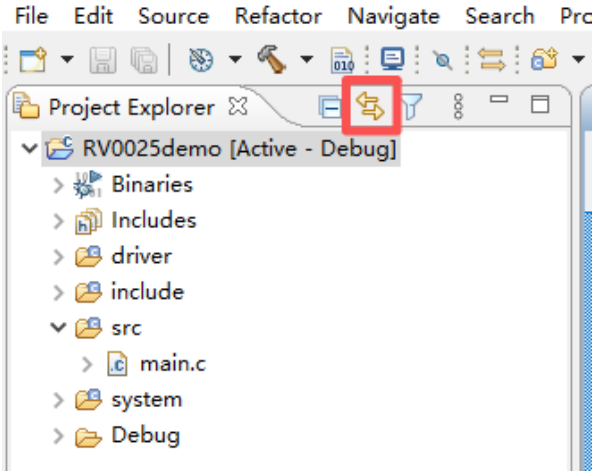


图 5-9 工程浏览器

5.3.2 大纲（Outline）视图

在打开 C 文件后，在右侧的大纲视图中，会显示当前文件中定义的宏、全局变量、函数，开发人员通过点击大纲视图中的元素，可以快速定位到元素位置，如图 5-10。

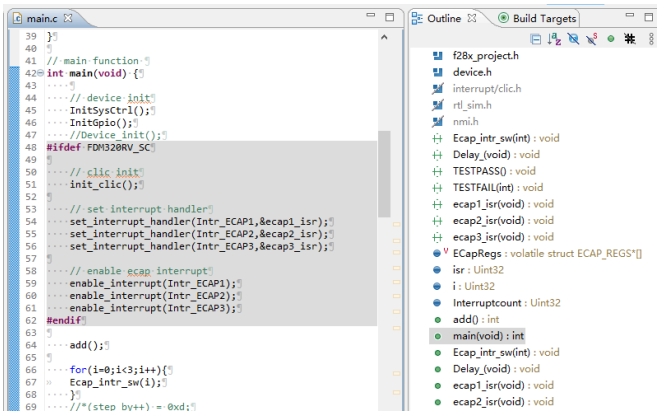

main.c
Outline
Build Targets
f28x_project.h
device.h
interrupt/clic.h
rtl_sim.h
nmi.h
Ecap_intr_sw(int) : void
Delay_(void) : void
TESTPASS() : void
TESTFAIL(int) : void
ecap1_isr(void) : void
ecap2_isr(void) : void
ecap3_isr(void) : void
ECapRegs : volatile struct ECAP_REGS*[]
isr : Uint32
i : Uint32
Interruptcount : Uint32
add() : int
main(void) : int
Ecap_intr_sw(int) : void
Delay_(void) : void
ecap1_isr(void) : void
ecap2_isr(void) : void

图 5-10 大纲视图

6 On-Chip Flash

IDE 提供 Flash 配置工具，通过“On-Chip Flash”工具实现包括擦除、烧录、验证以及安全域等相关操作。

6.1 On-Chip Flash 操作入口

IDE 提供两种操作的入口，分别是

- 通过菜单栏 => “Tools” => “On-Chip Flash”打开。
- 通过“Debug Configuration” => “On-Chip Flash”选项卡打开。

通过“Debug Configuration”打开的“On-Chip Flash”界面，如果之前有保存过信息，则会自动显示之前存储的信息；如果是第一次，则显示默认配置信息。这些存储信息包括：“Flash Program Setting”、“Erase Sections Selection”、“Select Program”和“CSMPSWD”。

下面以 RV0025 产品为例，讲解使用方法。在“Family”项中选择“SummerCore Family”，在“Product”项中选择“RV0025”。在 IDE 已连接仿真器和 RV0025 板卡的前提下，点击“Link Chip”进入“On-Chip Flash”配置界面，如下图。

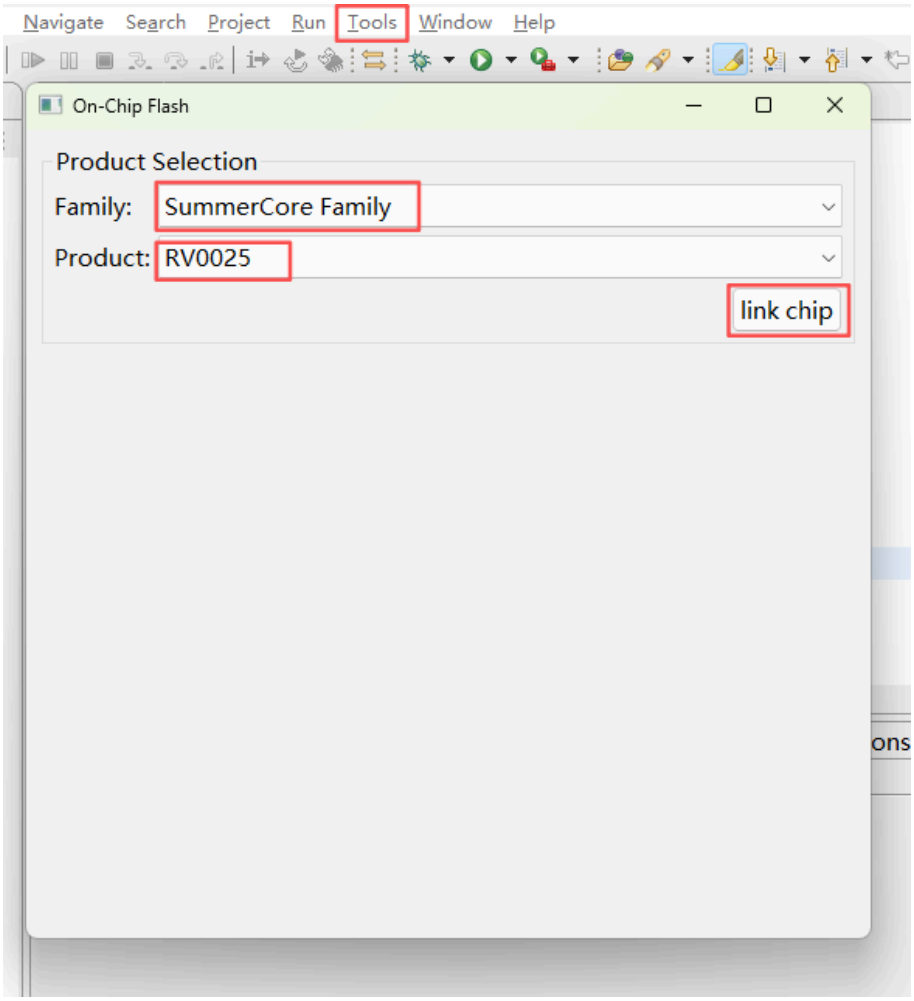


图 6-1 “On-Chip Flash”入口

注意：“Debug”状态下，“On-Chip Flash”会被禁用！

6.2 Flash Program Setting 使用

如图，进入“On-Chip Flash”界面后，可以选择 Flash 烧录选项。

“Flash Program Setting”用于设置调试 Flash 程序时的 Flash 操作方式：

- “Erase, Program, Verify”：先擦除选中的 Flash 扇区，再烧录程序并校验；
- “Program, Verify”：不执行擦除，直接烧录程序并校验；
- “Program Only”：只烧录程序，不执行擦除和校验。

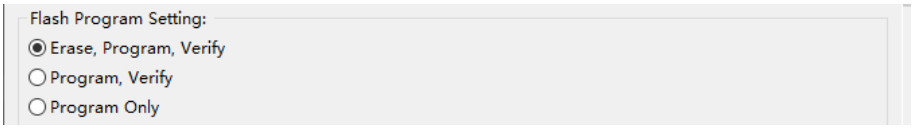


图 6-2 烧录配置

6.3 Flash 程序自动烧录

启动调试时，IDE 会检查“Main”页签中“C/C++ Application”指定的 ELF 文件。

如果 ELF 文件不包含映射到 Flash 区域的 section，则将其视为 RAM 程序处理，跳过 Flash 烧录，继续加载程序并进入调试。

如果 ELF 文件包含映射到 Flash 区域的 section，则 IDE 会提取需要写入 Flash 的数据，并按照“Flash Program Setting”和“Erase Sectors Selection”中的配置自动执行 Flash 相关操作。

Flash 操作成功后，IDE 会继续加载调试信息并进入调试；如果擦除、烧录或校验失败，IDE 会给出错误提示并终止本次调试。

6.4 Erase Flash 使用

如图所示，通过“Erase Flash”选项配置可以擦除 Flash 扇区。提供两种擦除方式：

- 全擦：勾选“All Sectors”，自动选中所有扇区；
- 部分擦：勾选特定“Sectors”。

配置要擦除的扇区后，点击“Erase Flash”按钮即可擦除扇区；擦除成功后会弹出“Success”对话框。

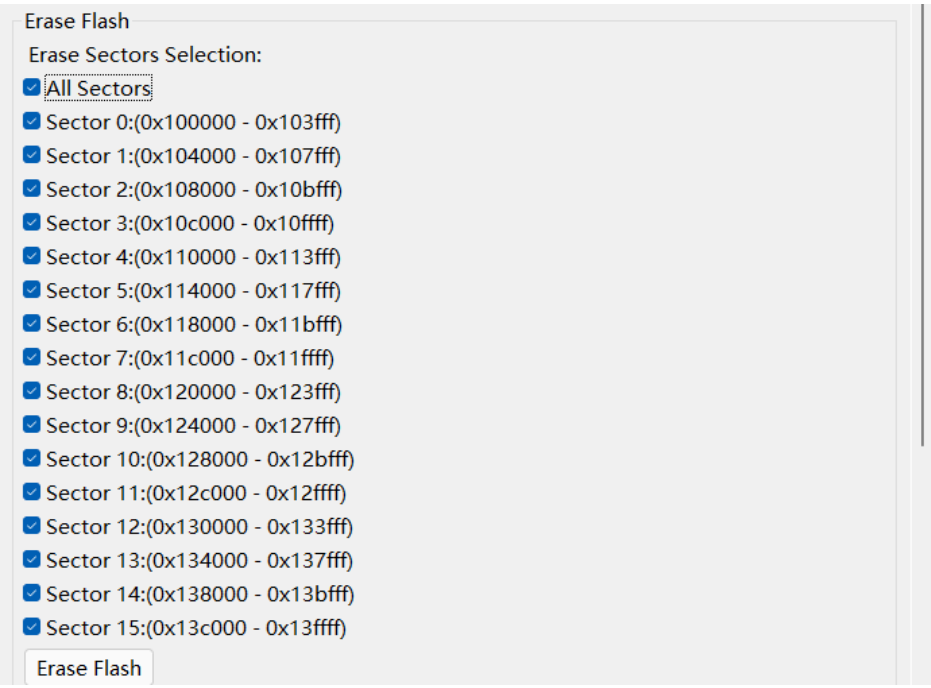


图 6-3 全擦配置

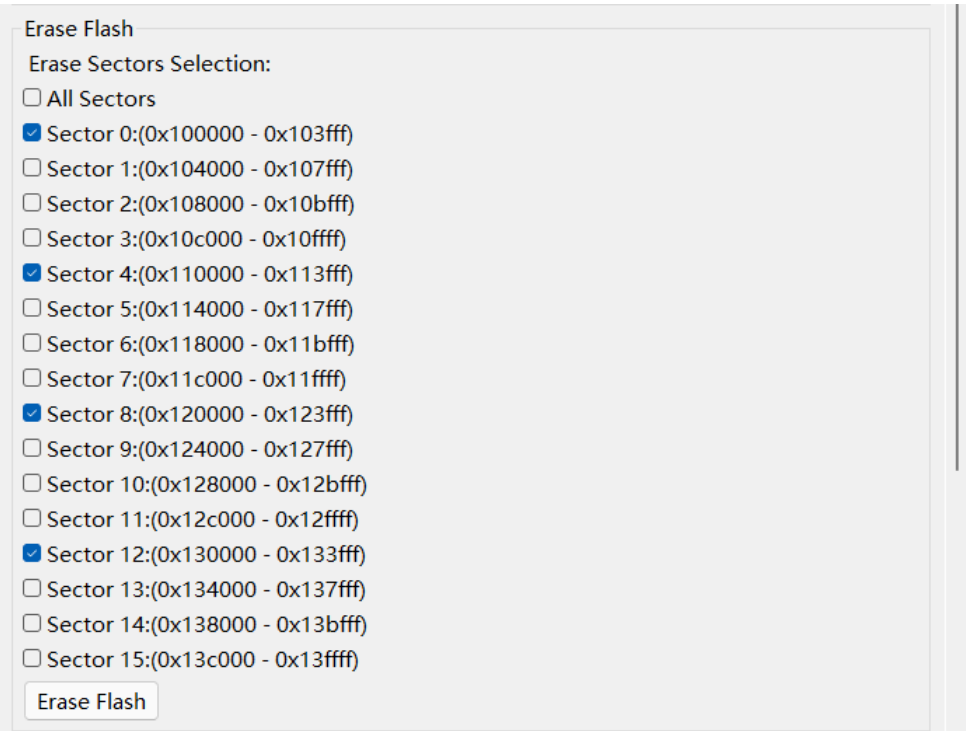


图 6-4 部分擦配置

6.5 Select Program 使用

如图所示，此区域结合“Flash Program Setting”选项的配置，可以对 Flash 进行烧录和验证：

- 通过“Select”按钮选择程序，填写要烧录的地址；
- 当前已勾选“Erase, Program, Verify”：
  - 点击“Program Flash”执行先擦除、后烧录指定文件，并自动进行烧录正确性验证。
- 当前已勾选“Program, Verify”：
  - 点击“Program Flash”烧录指定文件，并自动进行烧录正确性验证。
- 当前已勾选“Program Only”：
  - 点击“Program Flash”烧录指定文件。
- 当前已勾选“Load Ram Only”：
  - 只下载 ram 部分，映射到 flash 区域的部分不会烧录。
- “Erase, Program, Verify”和“Program, Verify”模式会在烧录后自动验证；“Program Only”模式不会自动验证。如需校验，可在烧录完成后点击“Select Program”选项中的“Verify”按钮手动执行。

注意：所选程序为 ELF 文件时，无需填写烧录地址；启动“Debug”时的 Flash 自动烧录不读取此处选择的文件，而是使用“Debug Configuration”的“Main”页签中“C/C++ Application”指定的 ELF 文件。

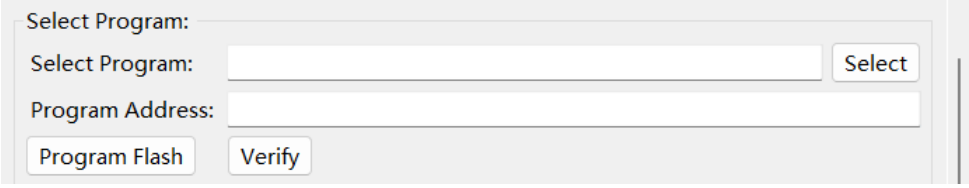


图 6-5 烧录与验证

6.6 Checksum 计算

在“Checksum”选项中点击“Calculate Checksum”，会自动读取图中各项存储空间数据，计算已有数据的校验和。

图 6-6 校验和计算

6.7 Security Setting 安全配置

“Security Setting”选项包含对 Zone1 和 Zone2 的 LINKPOINTER、GPREG、OTPSECLOCK、OTPBOOTCTRL/GPREG3、CSMPSWD、GRABSECT/RAM 的配置。

注意：flash 每 64 位计算对应 ecc 值，对于有 ecc 保护的 flash 区域，同一地址空间，用户只能烧录一次差异值，重复烧录不同数据会出现弹窗报错。

6.7.1 Zone Selection 切换分区

如图所示，在“Zone Selection”选项下勾选“Zone1”将显示 Zone1 的安全配置界面，勾选“Zone2”将切换到 Zone2 的安全配置界面。

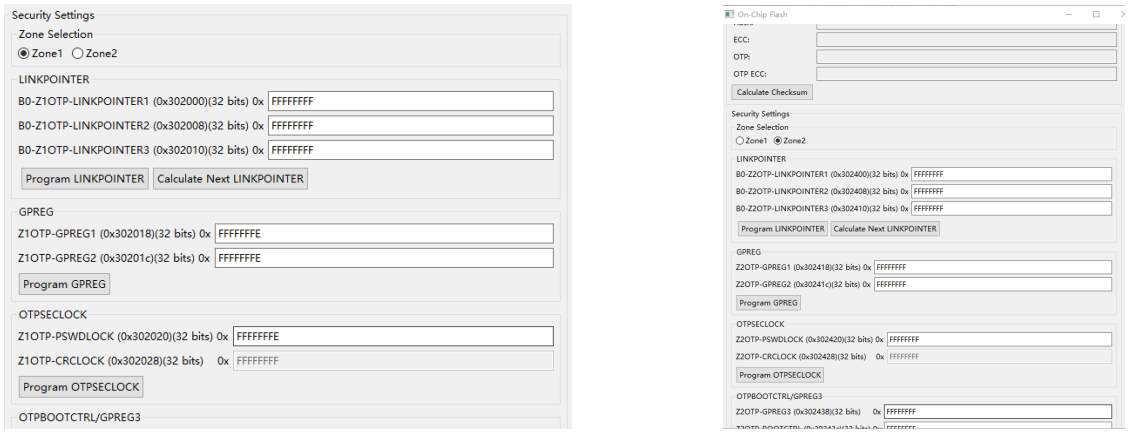


图 6-7 “Zone Selection”切换分区

6.7.2 LINKPOINTER 计算与烧录

如图，点击“Program LINKPOINTER”按钮，可将当前输入的 LINKPOINTER 值烧录到指定位置。由于 LINKPOINTER 不受 ECC 保护，工具会将该值存储到 3 个不同位置，并校验输入的 3 个 LINKPOINTER 值是否相同；如不相同则报错。点击“Calculate Next LINKPOINTER”将计算下一个区选择块对应的 LINKPOINTER。当计算结果为 0 时，表示所有区选择块已分配完毕，工具自动停止计算。

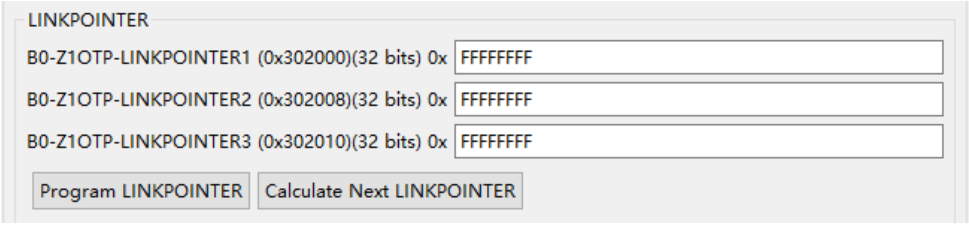


图 6-8 LINKPOINTER 计算与烧录

6.7.3 GPREG 烧录

GPREG 对应的 BootROM 名称为 Z1-OTP-BOOTPIN-CONFIG，输入值并点击“Program GPREG”进行烧录。

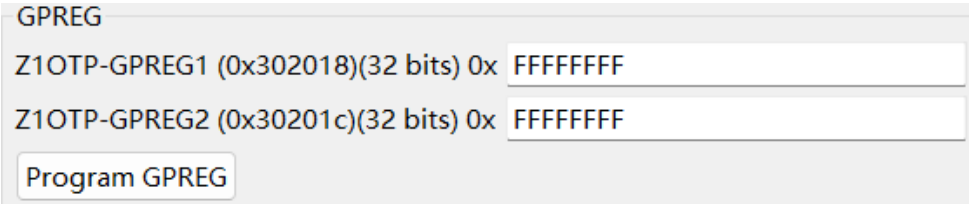


图 6-9 GPREG 烧录

6.7.4 OTPSECLOCK 烧录

OTPSECLOCK 为 OTP 安全锁，如下图，PSWDLOCK[3:0]为实际有效值：

- PSWDLOCK[3:0]=1111：OTP 中的 CSM 密码区域未受保护，可通过调试器以及任意位置运行的代码读取。
- PSWDLOCK[3:0]=其他值：OTP 中的 CSM 密码区域受保护，未解锁 Zone1 时无法读取。

点击“Program OTPSECLOCK”烧录。

例如，若希望 OTP 中的 CSM 密码区不可见，可向 Z1OTP-PSWDLOCK 或 Z2OTP-PSWDLOCK 烧录 0xFFFFFFF0。

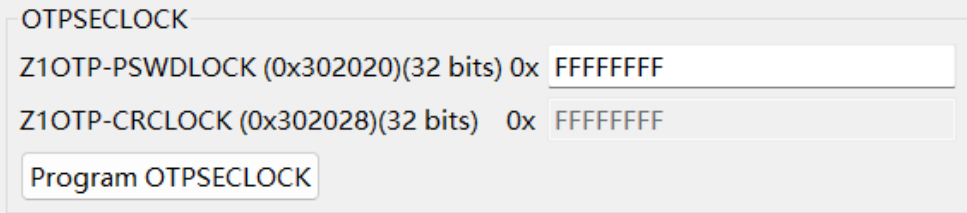


图 6-10 OTPSECLOCK 烧录

6.7.5 OTPBOOTCTRL 烧录

输入 GPREG3 值、BOOTCTRL 值，点击“Program OTPBOOTCTRL”，可将值烧录到对应地址。

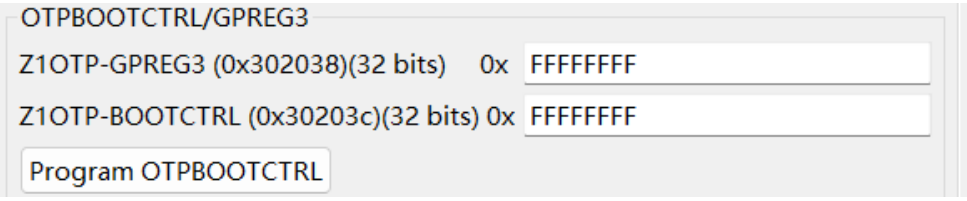


图 6-11 OTPBOOTCTRL 烧录

6.7.6 CSMPSWD 烧录与上锁

安全区可设置密码进行上锁。如图所示，输入密码值（默认全 F），点击“Program Password”烧录；板卡断电重新上电后，密码配置生效。点击“Lock”可将安全区上锁，使用设定的密码点击“Unlock”可解锁。

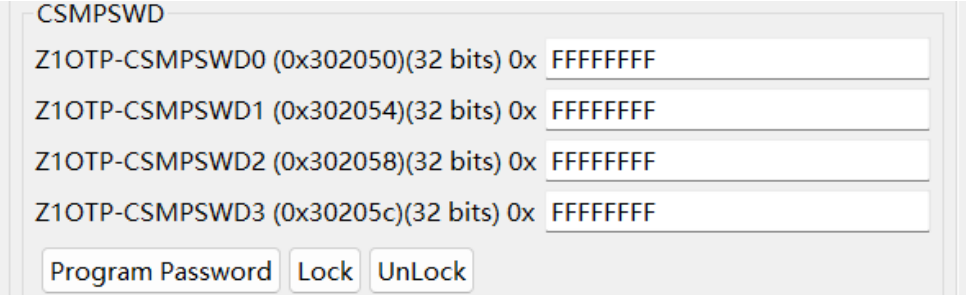


图 6-12 CSMPSWD 烧录

在“Debug Configuration”的“On-Chip Flash”页签中填写 CSMPSWD 解锁密码。该配置与当前“Debug Configuration”绑定并保存。

启动调试时，如果 IDE 检测到目标芯片处于锁定状态，将读取已保存的 CSMPSWD 解锁密码并尝试解锁 DCSM。解锁成功后继续调试；解锁失败时提示错误并终止本次调试。

此处填写的 CSMPSWD 仅用于启动调试时解锁 DCSM；如需重新设置芯片密码，应在“On-Chip Flash”界面中通过“Program Password”将新密码烧录到 OTP 区域。

6.7.7 GRABSECT/RAM 烧录

GRABSECT/RAM 包括以下两项配置：

- GRABSECT 分配 Flash 扇区的访问权限、安全区归属；
- GRABRAM 分配安全 RAM 的访问权限、安全区归属。

分别输入值并点击“Program GRABSECT/RAM”，可将值烧录到对应地址。

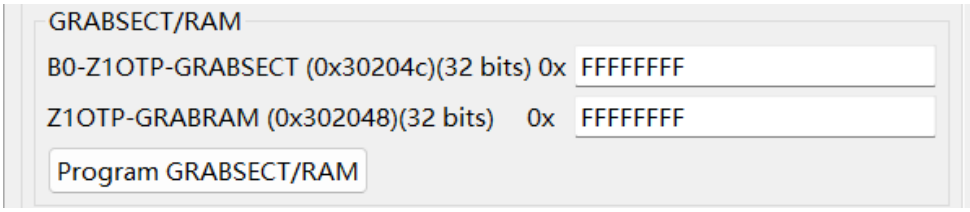


图 6-13 GRABSECT/RAM 烧录

7 其他工具

7.1 Elf2bin

Elf2bin 工具用于将 ELF 文件转换为 boot 使用的 bin 格式文件。可通过 IDE 菜单栏中的“Tools”=>“Elf2BinTool”打开该工具，界面如图所示。

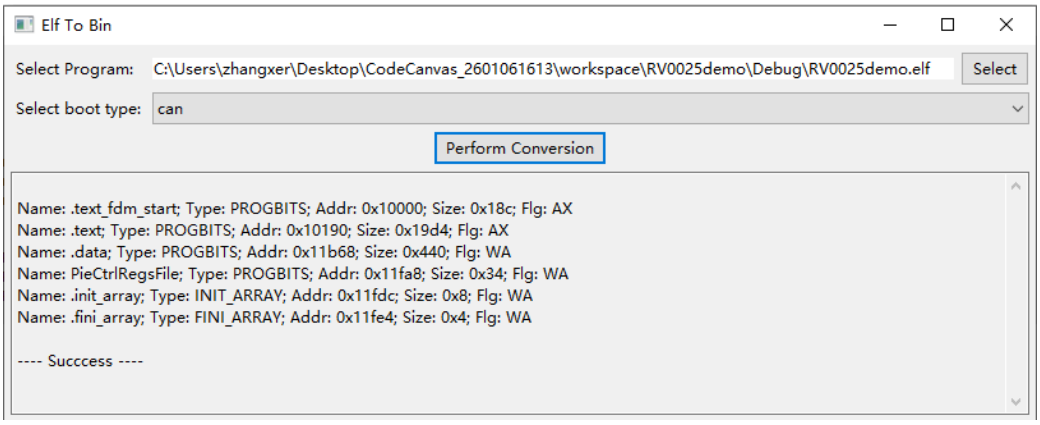
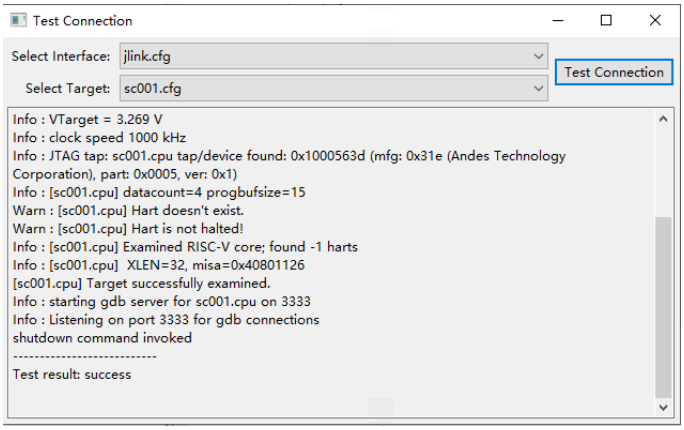


图 7-1 转换工具

首先点击“Select”按钮选择要转换的 ELF 文件，然后选择 boot 类型，接着点击“Perform Conversion”按钮，即可在下方文本框中获取转换结果。

7.2 连接测试

连接测试工具用于检查 IDE 与仿真器、板卡之间的连接是否可用。可通过 IDE 菜单栏中的“Tools”=>“TestConnection”打开该工具。首先选择仿真器型号和连接目标型号；以 RV0025 产品为例，“Interface”选择 jlink.cfg，“Target”选择 sc001.cfg。然后点击“TestConnection”，连接测试结果将显示在下方文本框中。


Test Connection
Select Interface:
jlink.cfg
Select Target:
sc001.cfg
Test Connection
Info : VTarget = 3.269 V
Info : clock speed 1000 kHz
Info : JTAG tap: sc001.cpu tap/device found: 0x1000563d (mfg: 0x31e (Andes Technology
Corporation), part: 0x0005, ver: 0x1)
Info : [sc001.cpu] datacount=4 progbufsize=15
Warn : [sc001.cpu] Hart doesn't exist.
Warn : [sc001.cpu] Hart is not halted!
Info : [sc001.cpu] Examined RISC-V core; found -1 harts
Info : [sc001.cpu] XLEN=32, misa=0x40801126
[sc001.cpu] Target successfully examined.
Info : starting gdb server for sc001.cpu on 3333
Info : Listening on port 3333 for gdb connections
shutdown command invoked

Test result: success

图 7-2 连接测试

8 历史版本

表 8-1 历史版本

版本号	描述	日期
V1.0	初稿	2026.01.06
V1.1	增加第三方导入操作说明	2026.07.16
V1.2	新增 6.3 节“Flash 程序自动烧录”，补充自动解锁及 Flash 程序调试时自动烧录的说明	2026.08.19