基于 VScode 移植 Goodix BLE keil 工程

一、环境准备

Vscodec 安装包：https://code.visualstudio.com/

BLE sdk 示例工程：https://www.goodix.com/zh/software_tool

GProgrammer 烧录工具：https://www.goodix.com/zh/software_tool/gprogrammer_ble

arm-none-eabi-gdb ：GNU 的官方工具链下载网站上有 arm-none 平台的工具链下载链接：

https://gnutoolchains.com/arm-eabi/，注意 gdb 需要 9.0 版本以上的

编译环境：ARMCC

VScodec 插件：

1.C/C++

2.C/C++ Extension Pack

3.C/C++ Themes

4.Chinese (Simplified) (简体中文) Language Pack for Visual Studio Code

5.Embedded IDE

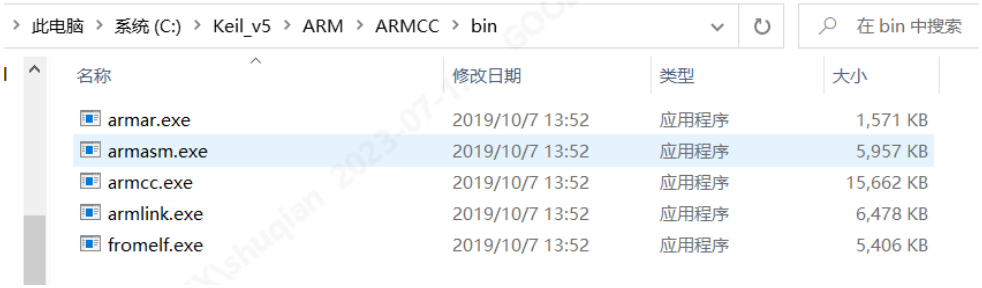
6.Cortex-Debug (ARM 调试插件)

7.vscode-icons

二、移植流程

1、配置工具链

如果你的电脑上已经安装 Keil MDK，那么你可以在 Keil 的安装目录下找到 ARM->ARMCC 文件夹，其对应的是我们开发 STM32F1 时常用到的 AC5 编译器（实际上已经落后），打开里面的 bin 目录可以找到如下图所展示的工具：



其中：

armar 主要用于从.a 和.lib 文件中提取信息

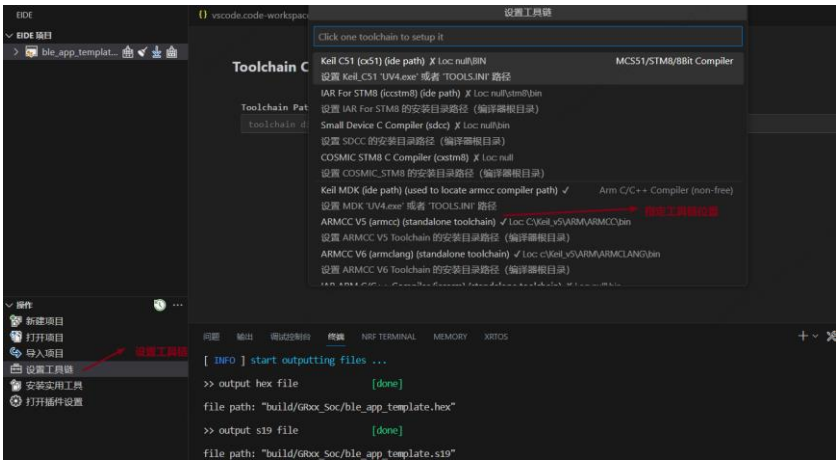
armasm 为汇编器，用于汇编程序

armcc 为编译器，用于编译源代码文件

armlink 为链接器，用于连接各程序段

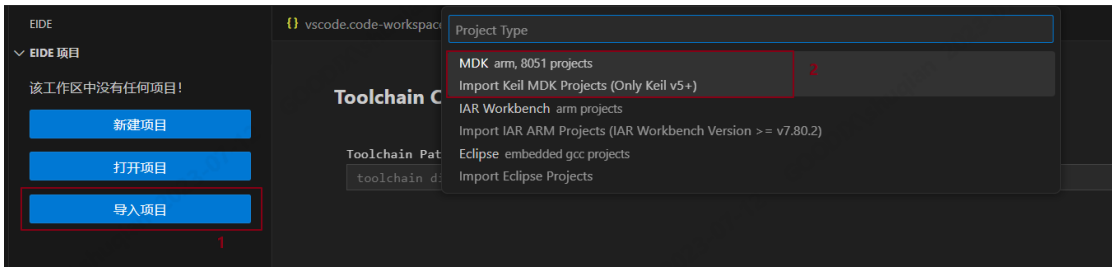
fromelf 可用于从 elf 文件中提取反汇编代码、生成二进制文件等

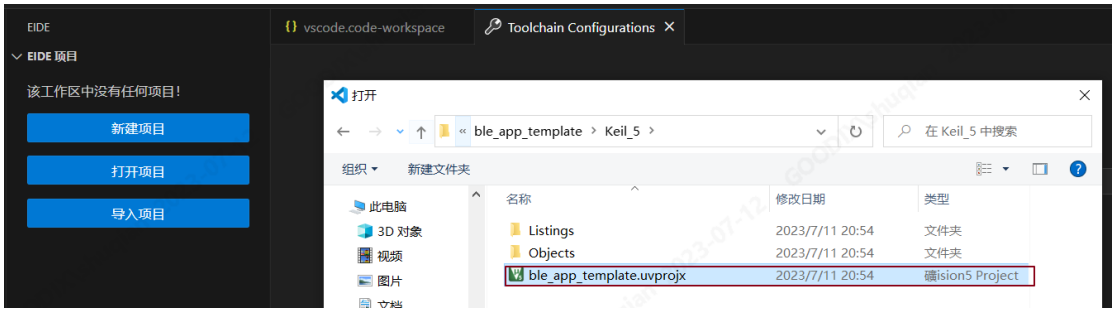
各个工具的具体用法可参照 ARM 官方说明。我们只需要知道可用的 ARM 编译器所在路径为 C:\Keil_v5\ARM\ARMCC\bin，再次配置编译工具链，设置 ARMCC 编译器所在路径如下图



2、直接导入 Keil5 工程

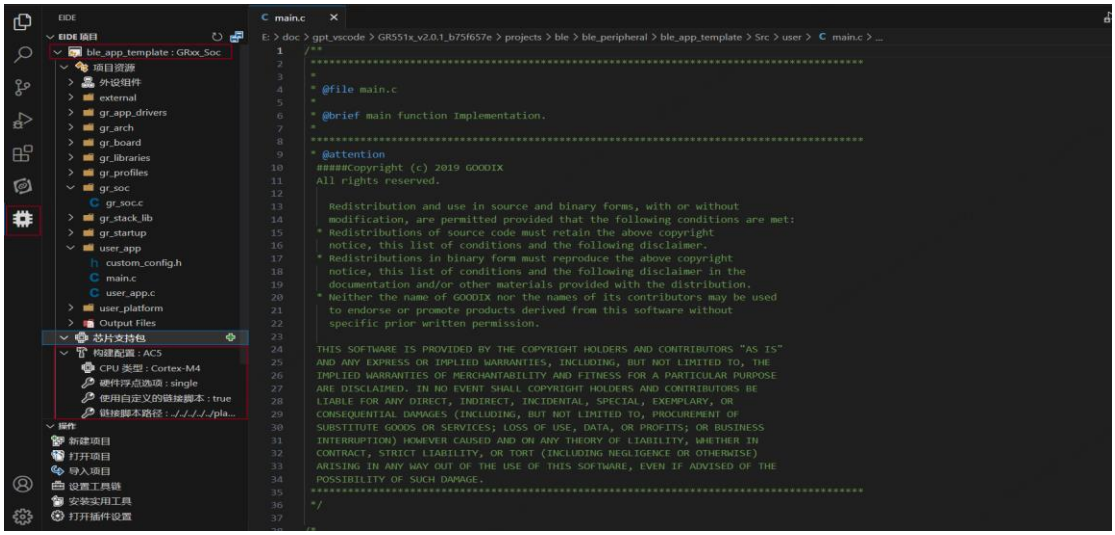
选择好编译工具链后可以直接打开已有的 Keil-MDK 工程，点击左侧的“导入项目”，并选择“MDK”，找到文件夹中的 Keil 工程后点击“Import”，此时右下角若有弹窗提示“是否与原有 Keil 项目共存于同一目录下”，可根据自身情况选择，选择完毕后切换入新工作区。



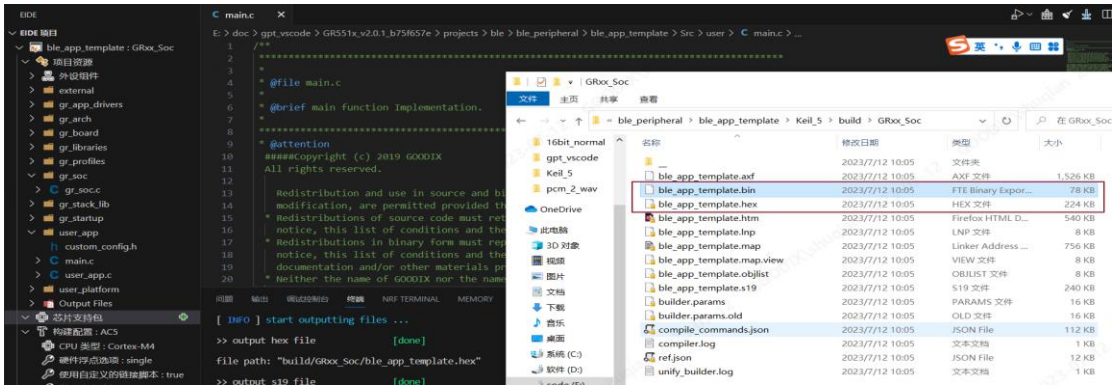


可以看到很多报错，不要急，点击右下角的“允许 EIDE 为此文件夹配置 IntelliSense”，报错消失。

切换至“EIDE”分页并查看工程配置，可以发现与 Keil 中的配置的一致，且跳转、自动补全功能都已经可以实现。



没有问题后按下快捷键 F7 成功编译，可以发现成功输出了 bin 文件与 hex 文件。

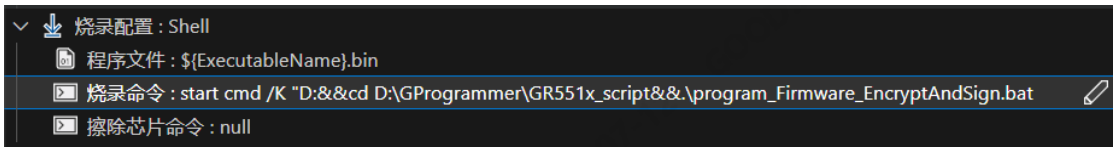


至此，通过 VScode 对 ARM 搭建工程进行编辑与编译的任务已经完成，下面介绍 VSCode 如何取代 Keil 对编写的 ARM 裸机代码进行烧录与调试。

三、烧录

因为没有找到我们芯片对应的 pack 包，没有找到 flash 相关的配置，所以这里使用的是 shell 的方式，用命令行调用 GProgrammer 这个工具进行烧录

- start cmd /K "D:&&cd D:\GProgrammer\GR551x_script&&.\program_Firmware_EncryptAndSign.bat "

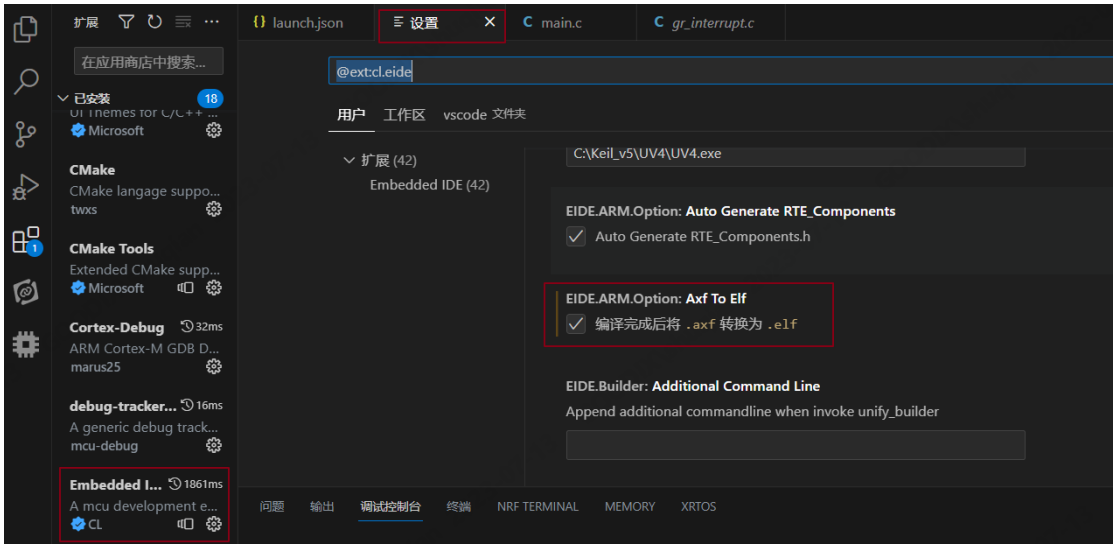


注意当前借助了我们自己的工具进行烧录，工具的安装目录下需要复制一份编译完成的 bin 文件，才能保证烧录正常，或者修改 bat 脚本中的固件路径保证固件正确下载烧录。

四、调试

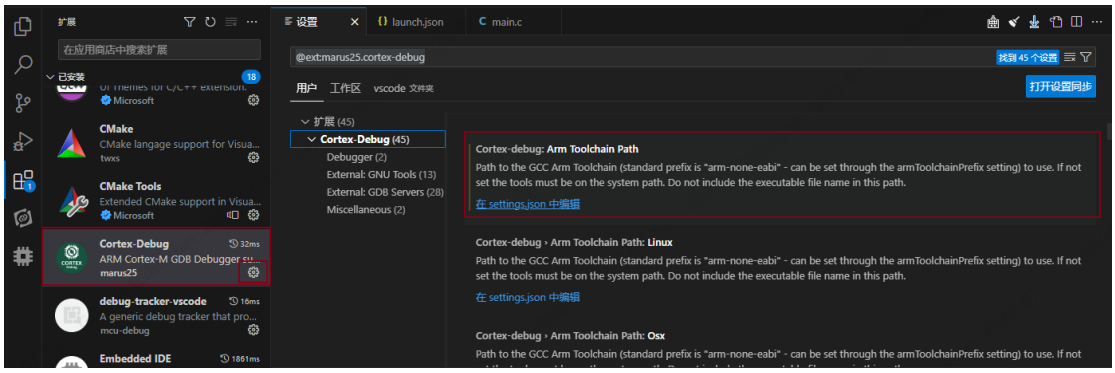
1、调试环境配置

- 首先需要准备 GDB 调试使用的文件类型，转到 EIDE 插件的设置界面，找到 Axf To Elf 选项，将其勾选。



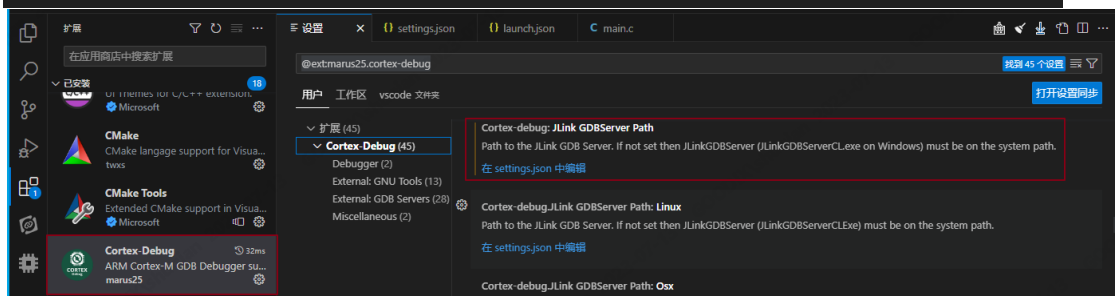
- 再打开 Cortex-Debug 插件的设置页面，搜索 Arm Toolchain Path 项，将 ARM Toolchian Path 的路径修改为 gcc-none-eabi 工具链中的 bin 文件夹所在路径，这里注意下，调试需要的 GDB 工具的版本需要 9.0 以上的，可以到官网下载最新版本。

```
"cortex-debug.armToolchainPath": "D:\\SysGCC\\bin",
```

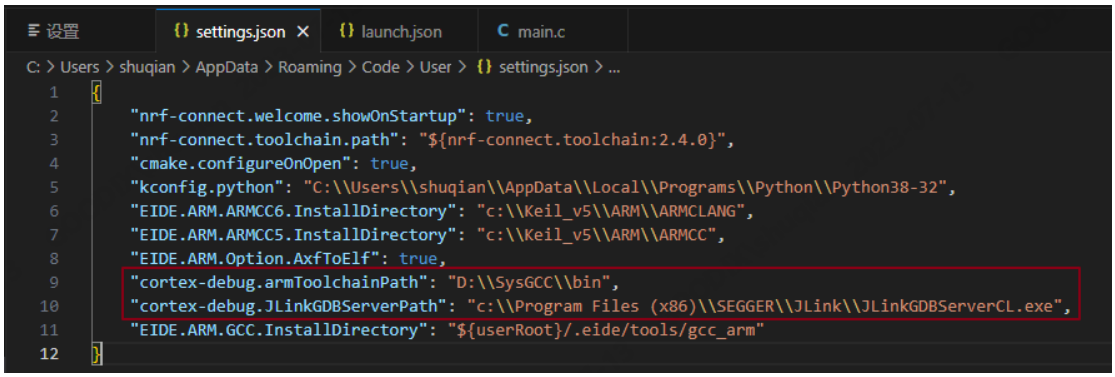


- 再搜索 jlink 关键字，将 JLink GDBServer Path 的路径修改为 Jlink GDBServer 所在目录

```
"cortex-debug.JLinkGDBServerPath": "c:\\Program Files (x86)\\SEGGER\\JLink\\JLinkGDBServerCL.exe",
```



修改完成后的 json 文件如下图所示。

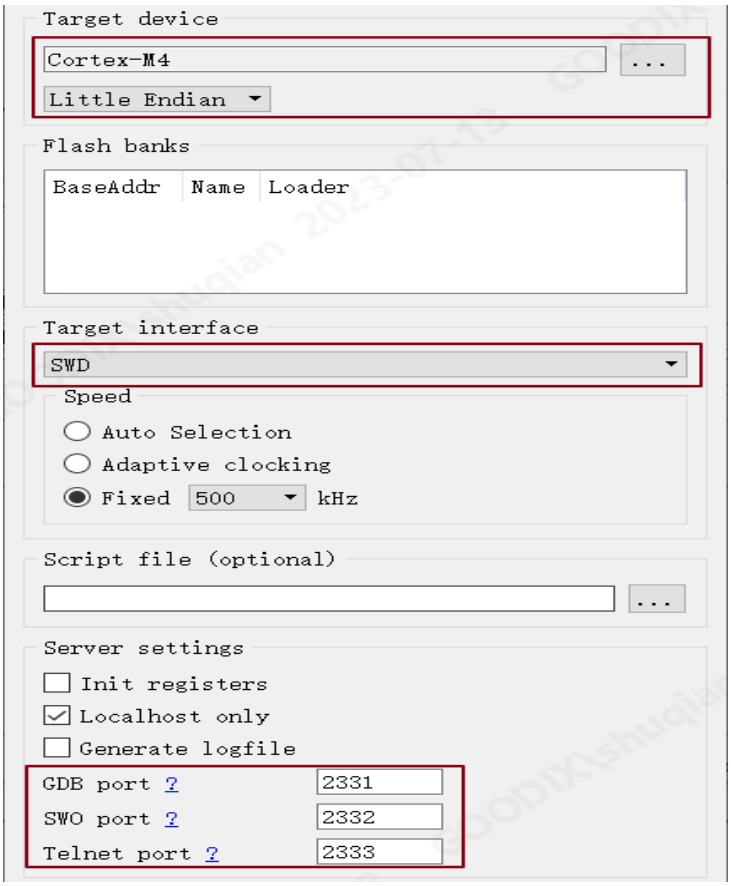


到这里工程的基本配置就完成了，配置完成后需要重新编译。

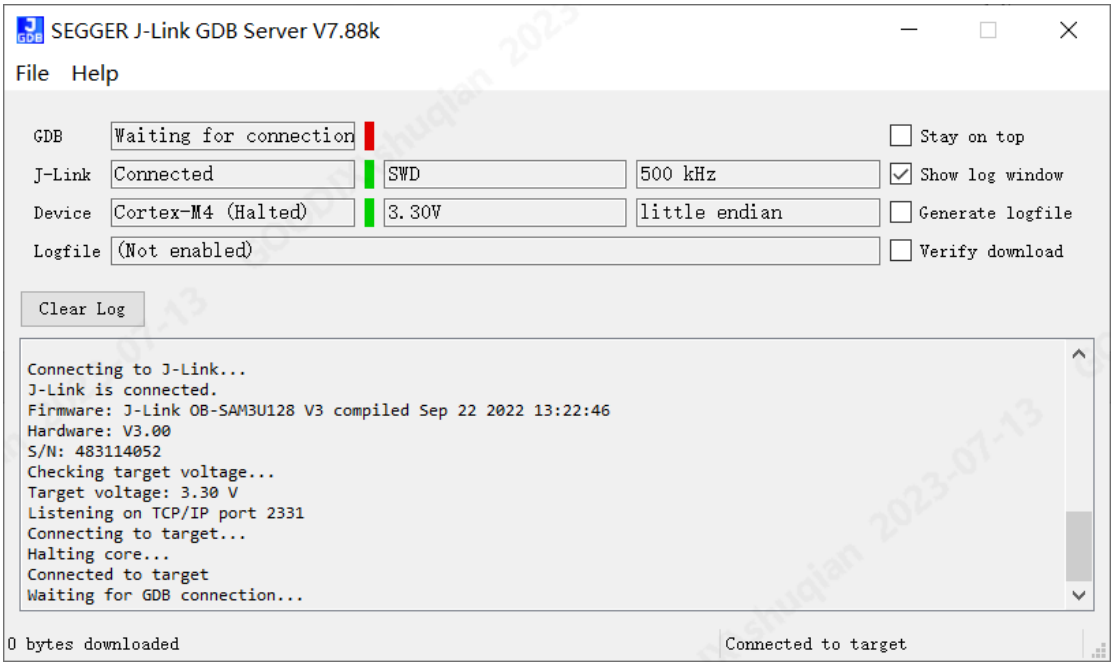
2、调试流程

- 打开 JLINK GDBServer，有两种方式可选

方法一：JLinkGDBServer.exe，双击后进入配置界面，可以按照下图进行配置。



配置完成后显示 J-Link 已连接，GDB Server Listening port: 2331，TCP 端口是 2331 处于监听状态，等待 GDB 的 TCP 连接。



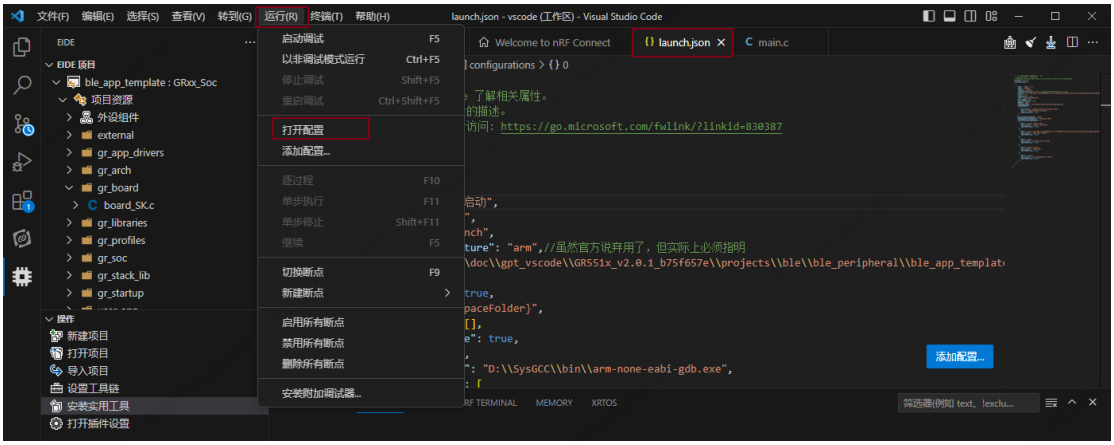
方法二：命令行，需将 JLinkGDBServer.exe 的路径保存到环境变量中，然后输入命令：

C:\Users\shuqian>JLinkGDBServer -device Cortex-M4 -if SWD -speed 500

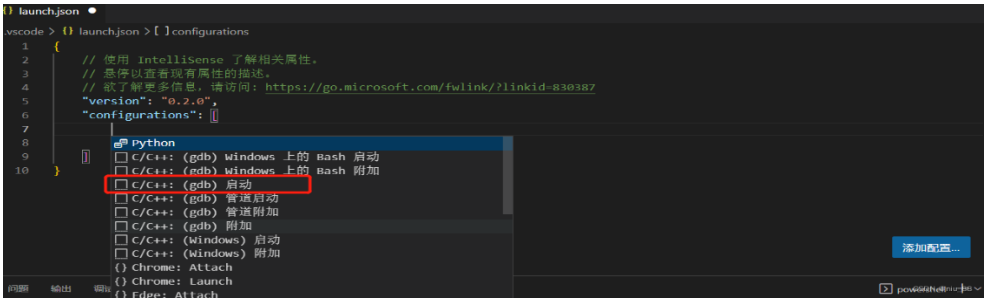
执行命令行后会弹出上图等待连接的窗口。

至此，GDB Server 已打开，tcp 端口 2331 处于监听状态，下一步是要通过 GDB 进行连接 GDB Server。

- vscode 配置 GDB，添加 gdb 设置文件 lanuch.json（很重要），通过 vscode 的菜单栏，**运行->添加配置**，则生成如下图文件

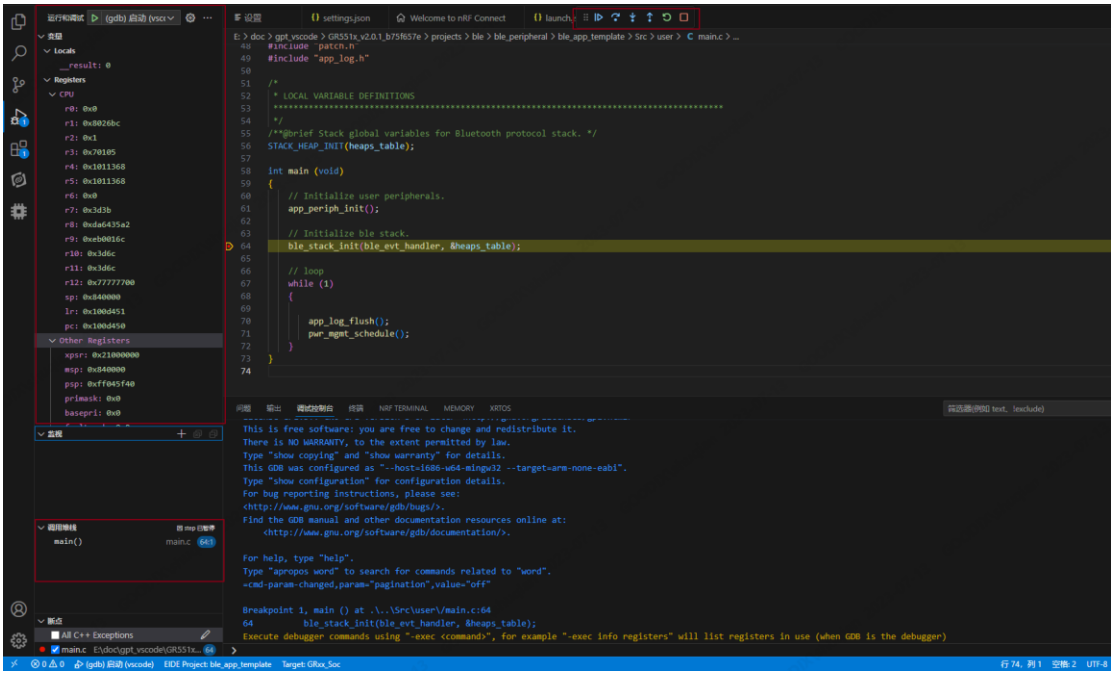


然后，选择下图中的**添加配置**，添加红色圈中的内容，



添加后，会自动生成一个基础版本的 gdb 配置，需要根据我们自己的项目进行修改，关于这个 json 文件的格式需要注意，具体有什么制约关系不是很清楚，我这里是在百度上找到的一个可以适用于我们板子在线调试的一个 launch.json 文件，如果只用它默认的模块配置的话，在线调试时会一直进不到 main 入口，怀疑是我们板子上的程序已经 run 起来了，于 GDB 调试有一些冲突，所以需要新增一些行为来保证调试的正常运行，customLaunchSetupCommands，这个是新增的一些行为。

- 配置完成后，F5 进行调试界面



五、参考文献

https://article.juejin.cn/post/7205489514361225273
[嵌入式开发工具]使用 VSCODE 代替 Keil 进行 C51 和 ARM 单片机开发与调试
https://www.dandelioncloud.cn/article/details/1545421980717776898
Visual Studio Code （VSCode） 之 C/C++ 调试配置详解
https://www.yht7.com/news/85534
使用 VSCode 和 VS2017 编译调试 STM32 程序的实现
https://blog.csdn.net/niu_88/article/details/127347197
vscode+jlink+GDBServer 在线调试
https://blog.csdn.net/studyingdda/article/details/126184241
VScode 配置 C 环境和导入 keil 工程