GR5xx GCC用户手册

版本： **1.2**

发布日期： **2025-07-11**

深圳市汇顶科技股份有限公司

商标声明

GOODiX 和其他汇顶商标均为深圳市汇顶科技股份有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人持有。

免责声明

本文档中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。

深圳市汇顶科技股份有限公司（以下简称“GOODIX”）对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。GOODIX对因这些信息及使用这些信息而引起的后果不承担任何责任。

未经GOODIX书面批准，不得将GOODIX的产品用作生命维持系统中的关键组件。在GOODIX知识产权保护下，不得暗中或以其他方式转让任何许可证。

深圳市汇顶科技股份有限公司

总部地址：深圳市福田区梅康路1号汇顶科技总部大厦26楼

电话：+86-755-33338828　　邮编：518000

网址：www.goodix.com

前言

编写目的

本文档介绍了在Linux和Windows环境下，使用GCC + Makefile的方式构建的命令行交叉编译开发环境，旨在帮助用户快速进行GR5xx SDK应用的二次开发。

读者对象

本文适用于以下读者：

- 芯片用户
- 开发人员
- 测试人员
- 技术支持工程师

版本说明

本文档为第3次发布，对应的产品为低功耗蓝牙GR5xx系列。

修订记录

版本	日期	修订内容
1.0	2023-12-27	首次发布
1.1	2025-06-06	新增“使用J-Link Commander下载固件”。
1.2	2025-07-11	“使用J-Link Commander下载固件”章节新增“运行固件”描述。

目录

1 简介

GCC（GNU Compiler Collection）是由GNU开发的一套开源编译器集，也是跨平台软件的编译器。GCC支持Linux和Windows操作系统。arm-none-eabi-gcc交叉编译器基于GCC，支持ARM系列CPU的指令集，适合GR5xx芯片使用。

在软件开发中，make是一个自动化构建工具，它根据Makefile文件的内容完成工程源代码文件的自动编译和链接。Makefile定义了使用编译器进行众多工程源文件编译、链接的规则，同时也可以调用执行系统命令。

本文主要介绍在Linux发行版Ubuntu以及Windows开发环境下，使用GCC + Makefile构建GR5xx应用开发环境的方法，并给出示例供用户参考。

在进行操作前，可参考以下文档。

表 1-1 文档参考

名称	描述
对应芯片开发者指南	介绍GR5xx SDK以及基于SDK的应用开发和调试
J-Link用户指南	J-Link使用说明：https://www.segger.com/downloads/jlink/UM08001_JLink.pdf
Bluetooth Core Spec	Bluetooth官方标准核心规范
Bluetooth GATT Spec	Bluetooth Profile和Service的详细信息查看地址：https://www.bluetooth.com/specifications/gatt/
GCC	GCC用户手册：https://launchpad.net/gcc-arm-embedded
Makefile	Makefile开发指南：https://www.gnu.org/software/make/manual/make.html

2 构建编译环境

本章节介绍基于Linux（Ubuntu）和Windows构建GR5xx的交叉编译及开发环境。

2.1 准备工作

在构建编译环境之前，用户需要完成如下准备。

- 软件准备

表 2-1 Linux软件准备

名称	描述
Ubuntu	推荐使用Ubuntu 16.04及之后的LTS版本（32 bit或64 bit均可）
gcc-arm-none-eabi	用于GR5xx可执行目标代码的交叉编译。 下载版本：gcc-arm-none-eabi-5_4-2016q3-linux.tar.bz2 下载地址：https://launchpad.net/gcc-arm-embedded/+download
Python	用于GR5xx应用工程的脚本执行环境的构建。 版本：python 3.x + 下载地址：https://www.python.org/downloads/
J-Link	提供J-Link硬件工具的驱动程序和软件操作支持库。 版本：J-Link Software and Documentation pack for Linux, DEB installer 下载地址：https://www.segger.com/downloads/jlink/ **说明：** ◦ 选择和用户所用的Ubuntu系统兼容的版本。 ◦ 使用J-Link 6.10a及之后的版本。

说明:

◦ Linux发行版本建议使用Ubuntu 16.04及之后的LTS版本。由于GCC软件的运行依赖于Ubuntu的环境，建议将本文推荐的GCC版本和Ubuntu版本配合使用。

◦ 如果选用其他Linux发行版本，可能会产生环境依赖问题。

表 2-2 Windows软件准备

名称	描述
MSYS	MSYS是Windows平台的一个小型GNU环境，提供了bash、make、mkdir、grep等传统依赖UNIX系统运行的工具。 下载版本：MSYS-1.0.11 下载地址：https://nchc.dl.sourceforge.net/project/mingw/MSYS/Base/msys-core/msys-1.0.11/MSYS-1.0.11.exe?viasf=1
gcc-arm-none-eabi	用于GR5xx可执行目标代码的交叉编译。

名称	描述
	下载版本：gcc-arm-none-eabi-9-2020-q2-update-win32.zip 下载地址：https://developer.arm.com/-/media/Files/downloads/gnu-rm/9-2020q2/gcc-arm-none-eabi-9-2020-q2-update-win32.zip
Python	用于GR5xx应用工程的脚本执行环境的构建。 版本：python 3.x + 下载地址：https://www.python.org/downloads/
J-Link	提供J-Link硬件工具的驱动程序和软件操作支持库。 版本：J-Link Software and Documentation pack for Windows 下载地址：https://www.segger.com/downloads/jlink/ **说明：** 使用J-Link 6.10a及之后的版本。

- 硬件准备

表 2-3 硬件准备

名称	描述
开发板	对应芯片Starter Kit开发板（以下简称“开发板”）
连接线	USB Type-C（GR551x系列使用Micro USB 2.0连接线）

2.2 安装

编译ARM程序前，用户需要安装交叉编译器gcc-arm-none-eabi。

2.2.1 Linux

本章节将详细介绍Linux下GCC和Python等的安装步骤。

2.2.1.1 下载GCC

访问网址https://launchpad.net/gcc-arm-embedded/+download，下载版本为*gcc-arm-none-eabi-5_4-2016q3-linux.tar.bz2*的安装包。

该安装包基于32-bit架构构建。如用户需要64-bit版本，可从https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads进行下载，Linux 64-bit Tarball版本。

2.2.1.2 安装GCC

安装包为免编译版本，将其解压到合适的目录位置。

使用以下命令解压安装包：

```
tar xf gcc-arm-none-eabi-5_4-2016q3-linux.tar.bz2
```

2.2.1.3 设置环境变量

用户可按照实际安装路径在环境变量PATH中增加.bin文件路径。如下所示：

- Root用户

```
echo “export PATH=$PATH:/home/goodix/gcc-arm-none-eabi-5_4-2016q3/bin” >> /etc/
bash.bashrc
source /etc/bash.bashrc
```

- 非Root用户

```
echo “export PATH=$PATH:/home/goodix/gcc-arm-none-eabi-5_4-2016q3/bin” >> ~/.bashrc
source ~/.bashrc
```

2.2.1.4 测试GCC安装结果

GCC安装完成后，可通过以下命令测试GCC是否安装成功。

```
arm-none-eabi-gcc -v
```

在Terminal上打印出以下信息，表示GCC安装成功。

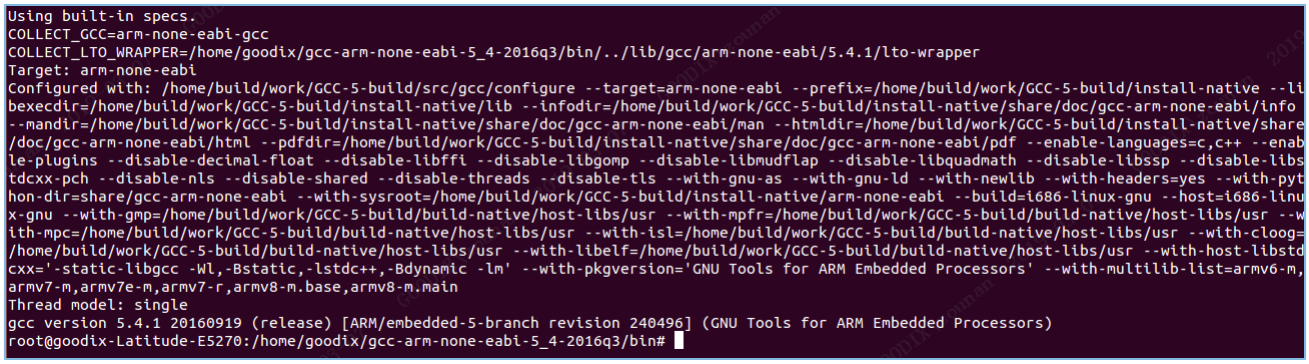

图 2-1 GCC安装结果显示

说明:

- Ubuntu版的arm-none-eabi-gcc编译器自身不区分32-bit / 64-bit。
- 部分Ubuntu发行版在执行`arm-none-eabi-*`系列命令时，可能出现错误提示“no such file or directory”，原因是缺少三方依赖库ia32-libs，执行如下安装命令即可：

  ```
  sudo apt-get install lib32ncurses5
  sudo apt-get install lib32z1
  ```

2.2.1.5 安装Python

1. 访问网址：https://www.python.org/downloads，下载并安装Python3。要求下载的Python版本和用户使用的Ubuntu系统兼容。

   输入以下命令安装Python3：

```
sudo apt-get install python3
```

2. 运行Python。

```
python
```

3. 如果Python安装成功，将查看到Python的版本信息。

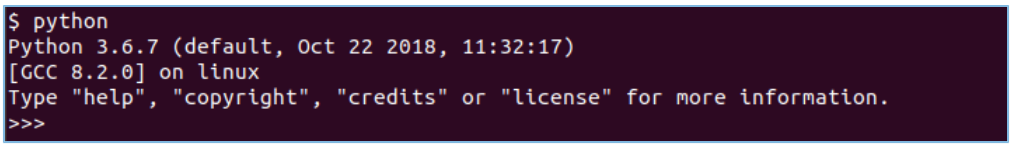

图 2-2 Python安装结果显示

说明:

本示例安装的Python版本为3.6.7。在Terminal上打印出：Python 3.6.7，表示Python安装成功。

2.2.1.6 安装J-Link

访问网址：https://www.segger.com/downloads/jlink/，在Segger官网下载J-Link for Linux。

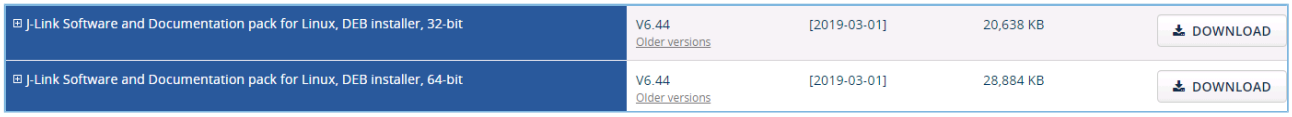

图 2-3 Segger官网J-Link下载

在Ubuntu下安装J-Link for Linux的DEB包。

说明:

- J-Link版本需和用户的Ubuntu系统兼容。
- J-Link使用6.10a及以上版本。
- 安装完成后，在命令行执行`JLinkExe`命令，正常情况下可开始使用J-Link。如果安装J-Link后不能正常使用，请检查环境变量是否添加成功。

2.2.2 Windows

本章节将详细介绍Windows下GCC和Python等的安装步骤。

2.2.2.1 下载MSYS和GCC

MSYS下载：访问https://nchc.dl.sourceforge.net/project/mingw/MSYS/Base/msys-core/msys-1.0.11/MSYS-1.0.11.exe?viasf=1，下载的*MSYS-1.0.11.exe*为可执行安装包。

GCC下载：访问https://developer.arm.com/-/media/Files/downloads/gnu-rm/9-2020q2/gcc-arm-none-eabi-9-2020-q2-update-win32.zip，下载版本为*gcc-arm-none-eabi-9-2020-q2-update-win32.zip*的软件包。

2.2.2.2 安装MSYS和GCC

1. 双击*MSYS-1.0.11.exe*完成MSYS工具安装。安装成功的界面如下所示：

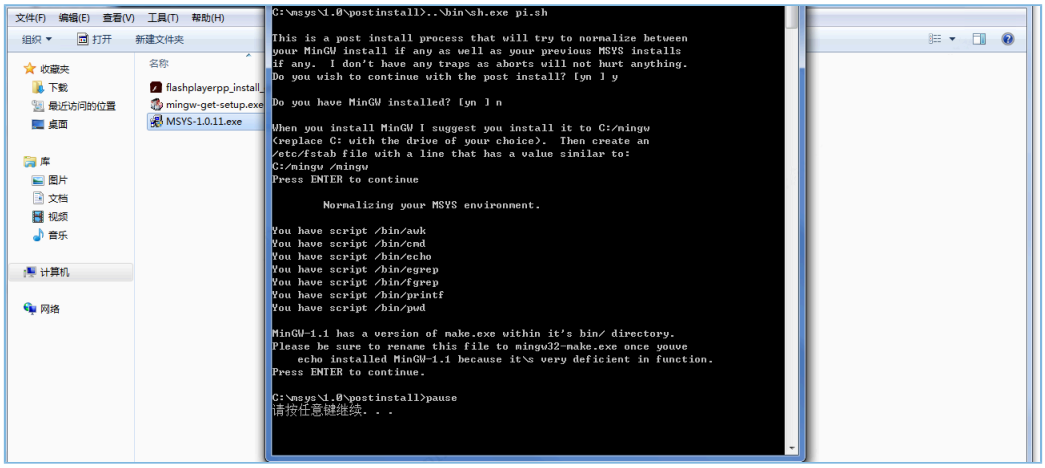

图 2-4 完成MSYS安装

2. *gcc-arm-none-eabi-9-2020-q2-update-win32.zip*为GCC免安装版本软件包，将其解压到合适的目录位置。

2.2.2.3 设置环境变量

在Windows高级设置的环境变量设置下增加以下软件的路径。

- MSYS路径：`<MSYS_安装路径>\bin`
- GCC路径：`<GCC Win32 安装路径>\bin`

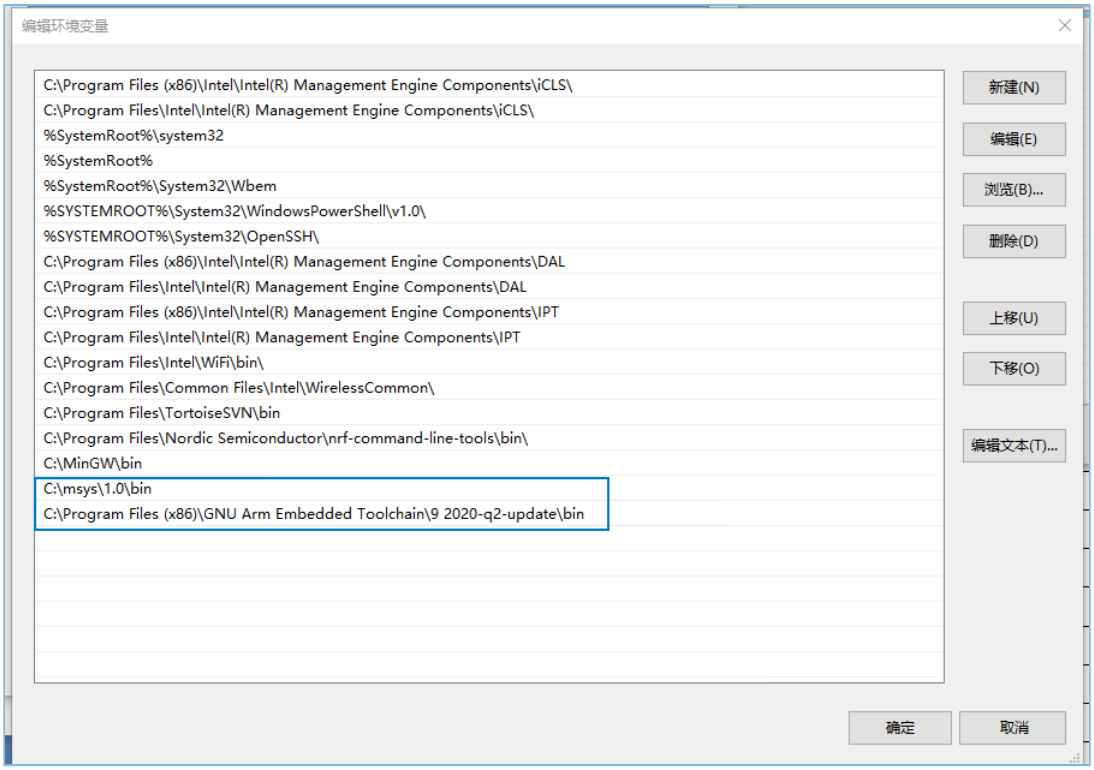

图 2-5 设置Windows系统环境变量

2.2.2.4 测试GCC安装结果

GCC安装完成后，输入`make -v`命令查看Make工具（MSYS提供）版本信息。

```
$ make -v
GNU Make 3.81
Copyright (C) 2006  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
```

图 2-6 查看Make工具版本信息

输入`arm-none-eabi-gcc -v`命令查看GCC工具版本信息。

```
Thread model: single
gcc version 9.3.1 20200408 (release) (GNU Arm Embedded Toolchain 9-2020-q2-update)
```

图 2-7 查看GCC工具版本信息

2.2.2.5 安装Python

1. 访问网址：https://www.python.org/downloads，下载并安装Python3。要求下载的Python版本和用户使用的Windows系统兼容。
2. 根据安装向导进行安装。
3. 设置环境变量。

安装成功后在Windows命令窗口输入`python`命令，可查看Python版本信息。

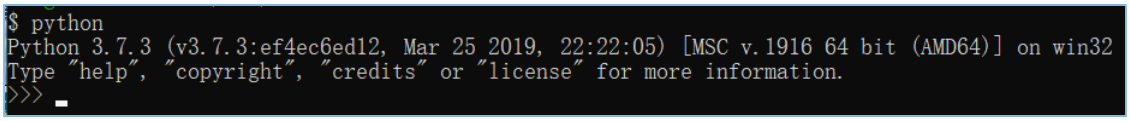

图 2-8 查看Python版本信息

2.2.2.6 安装J-Link

访问网址：https://www.segger.com/downloads/jlink/，在Segger官网下载J-Link for Windows。

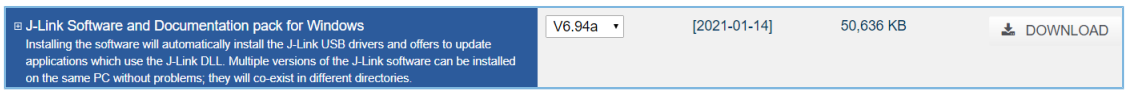

图 2-9 Segger官网J-Link下载

下载完成后双击*JLink_Windows_Version.exe*安装程序，根据安装向导选择合适路径进行安装。

说明:

- J-Link使用6.10a及以上版本。
- Version表示安装的J-Link版本号。

2.3 开发板连接测试

在完成安装后，即可连接开发板测试。

- **Linux**

  确认J-Link已加入环境变量，在计算机终端界面依次键入命令（#之后的文本为命令注释）：

```
JLinkExe          # 命令行调用启动Jlink工具
connect           # 使用connect命令连接开发板，执行本命令前请保证已接入开发板
CORTEX-M4         # 指定CPU内核型号，如J-Link工具正常识别,可以直接回车
S                 # 选择硬件链接调试接口，S代表SWD
```

```
4000                # 指定SWD通讯速率，单位kHz,这里设置为4000
```

界面出现“Cortex-M4 identified”时，表示PC通过J-Link成功连接了开发板。

```
Connecting to J-Link via USB...O.K.
Firmware: J-Link OB-SAM3U128 V3 compiled Sep 21 2017 14:14:50
Hardware version: V3.00
S/N: 483060523
VTref = 3.300V


Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: CORTEX-M4
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
  S) SWD
TIF>s
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "CORTEX-M4" selected.


Connecting to target via SWD
Found SW-DP with ID 0x2BA01477
Scanning AP map to find all available APs
AP[1]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x24770011)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FF000
CPUID register: 0x410FC241. Implementer code: 0x41 (ARM)
Found Cortex-M4 r0p1, Little endian.
FPUnit: 6 code (BP) slots and 2 literal slots
CoreSight components:
ROMTbl[0] @ E00FF000
ROMTbl[0][0]: E000E000, CID: B105E00D, PID: 000BB00C SCS-M7
ROMTbl[0][1]: E0001000, CID: B105E00D, PID: 003BB002 DWT
ROMTbl[0][2]: E0002000, CID: B105E00D, PID: 002BB003 FPB
ROMTbl[0][3]: E0000000, CID: B105E00D, PID: 003BB001 ITM
ROMTbl[0][4]: E0040000, CID: B105900D, PID: 000BB9A1 TPIU
Cortex-M4 identified.
J-Link>
```

图 2-10 J-Link连接成功

- **Windows**

  Windows平台下，开发板和PC连接后，打开PC的设备管理器，检查“设备管理器 > 端口（COM和LPT）”列表中是否有“J-Link”。若出现则J-Link连接成功，若未正确检测到J-Link设备，则需要检查是否正确安装了J-Link驱动，可以尝试重新安装最新版本的J-Link驱动。

2.4 编译SDK应用示例工程

本节通过应用示例工程ble_app_template，对Makefile的生成、使用和编译等进行介绍。以下流程通用于Linux和Windows环境。

说明:

SDK_Folder为GR5xx SDK的根目录。

2.4.1 Makefile文件

目前在GR5xx SDK中为示例工程提供了脚本工具生成Makefile文件。

Makefile：make编译规则文件，用于执行GCC命令（编译、链接）和各种操作系统命令，以及定义一系列规则，比如编译器属性、文件编译顺序、编译及链接规则、目标依赖关系等。通过执行make操作，生成可执行文件。

2.4.2 生成Makefile文件

GR5xx SDK开发包中的应用示例工程，默认使用Keil uVision5 IDE环境进行编译构建，如果用户希望使用GCC工具链编译构建ble_app_template之外的应用示例工程，可使用*keil2makefile.py*脚本工具，将Keil的工程文件* .uvprojx转换为Makefile，并生成lds文件。

*keil2makefile.py*使用说明如下：

1. *keil2makefile.py*工具文件默认位于`SDK_Folder\build\gcc`目录下。
2. 为保证转换后的Makefile引用的源文件和头文件路径正确，已约束*keil2makefile.py*脚本在使用时需要与*.uvprojx文件位于同一个目录。
3. 将*keil2makefile.py*文件拷贝到目标应用工程的Keil_5目录下。以ble_app_template为例，将脚本拷贝至`SDK_Folder\projects\ble\ble_peripheral\ble_app_template\keil_5`。
4. 从命令行切换到目标路径，执行如下命令。以ble_app_template为例，命令行和生成的makefile及lds文件如下图所示：

```
python keil2makefile.py ble_app_template.uvprojx
```

```
>python keil2makefile.py ble_app_template.uvprojx
>>> Transfer project : ble_app_template.uvprojx
>>> The goal project name : GRxx_Soc / ble_app_template
>>> OS type: Windows
>>> Generate Makefile Successfully, located at ../GCC/Makefile
>>> Generate lds file starting ...
    FLASH (rx) : ORIGIN = 0x00202000, LENGTH = (0x00800000 -0x00002000)

>>> Generate lds file finish ...
```

图 2-11 成功生成Makefile及lds文件

5. 转换成功后的Makefile和lds文件，放置于与Keil_5目录平行的GCC目录，用户可进入此目录进行查看。

2.4.3 修改Makefile配置项

生成的Makefile在文件内“Common Configuration Area”部分提供了编译和链接的一组默认参数，用户可以根据项目的具体情况，对编译参数按需进行修改，但建议谨慎修改，以避免引起工程编译失败。

用户可以在已生成Makefile文件基础上增加自定义.c和.h文件。Linux和Windows下增加.c文件和.h文件如下所示：

```
PRJ_C_SRC_FILES:=   \
../../../../../platform/soc/common/gr_system.c  \
../../../../../platform/soc/common/gr_interrupt.c  \
../../../../../platform/soc/common/gr_platform.c  \
../../../../../platform/soc/src/gr_soc.c  \
../../../../../platform/boards/board_SK.c  \
../../../../../drivers/src/app_dma.c  \
../../../../../drivers/src/app_gpiote.c  \
../../../../../drivers/src/app_io.c  \
../../../../../drivers/src/app_pwr_mgmt.c  \
../../../../../drivers/src/app_uart.c  \
../../../../../drivers/src/app_spi.c  \
../../../../../drivers/src/app_uart_dma.c  \
../../../../../components/libraries/utility/utility.c  \
../../../../../components/libraries/sensorsim/sensorsim.c  \
../../../../../components/libraries/app_timer/app_timer.c  \
../../../../../components/libraries/app_log/app_log.c  \
../../../../../components/libraries/app_assert/app_assert.c  \
../../../../../components/libraries/app_error/app_error.c  \
../../../../../components/libraries/ring_buffer/ring_buffer.c  \
../../../../../components/libraries/pmu_calibration/pmu_calibration.c  \
../../../../../components/libraries/dfu_port/dfu_port.c  \
../../../../../components/libraries/hci_uart/hci_uart.c  \
../../../../../components/libraries/app_key/app_key.c  \
../../../../../components/libraries/app_key/app_key_core.c  \
../../../../../components/libraries/fault_trace/fault_trace.c  \
../../../../../components/libraries/app_error/cortex_backtrace.c  \
../../../../../components/profiles/common/ble_prf_utils.c  \
../../../../../components/profiles/bas/bas.c  \
../../../../../components/profiles/dis/dis.c  \
../../../../../components/profiles/hrs/hrs.c  \
../../../../../components/profiles/otas/otas.c  \
../../../../../components/profiles/lns/lns.c  \
../../../../../external/segger_rtt/SEGGER_RTT.c  \
../Src/platform/user_periph_setup.c  \
../Src/user/main.c  \
../Src/user/user_app.c  \
```

图 2-12 增加.c文件

```
PRJ_C_INCLUDE_PATH :=   \
../Src/config  \
../Src/platform  \
../Src/user  \
../Src/config  \
../../../../../components/boards  \
../../../../../components/libraries/app_alarm  \
../../../../../components/libraries/app_assert  \
../../../../../components/libraries/app_error  \
../../../../../components/libraries/app_key  \
../../../../../components/libraries/app_log  \
../../../../../components/libraries/app_queue  \
../../../../../components/libraries/app_timer  \
../../../../../components/libraries/at_cmd  \
../../../../../components/libraries/dfu_master  \
../../../../../components/libraries/dfu_port  \
../../../../../components/libraries/gui  \
../../../../../components/libraries/gui/gui_config  \
../../../../../components/libraries/hal_flash  \
../../../../../components/libraries/fault_trace  \
../../../../../components/libraries/hci_uart  \
../../../../../components/libraries/pmu_calibration  \
../../../../../components/libraries/ring_buffer  \
../../../../../components/libraries/sensorsim  \
../../../../../components/libraries/utility  \
../../../../../components/patch/ind  \
../../../../../components/profiles/ams_c  \
../../../../../components/profiles/ancs_c  \
../../../../../components/profiles/ans  \
../../../../../components/profiles/ans_c  \
../../../../../components/profiles/bas  \
../../../../../components/profiles/bas_c  \
../../../../../components/profiles/bcs  \
../../../../../components/profiles/bps  \
../../../../../components/profiles/common  \
../../../../../components/profiles/cscs  \
../../../../../components/profiles/cts  \
../../../../../components/profiles/cts_c  \
../../../../../components/profiles/dis  \
```

图 2-13 增加.h文件

2.4.4 执行make编译

1. 进入目标示例工程的Makefile文件放置目录，以ble_app_template为例，位于：

   `SDK_Folder\projects\ble\ble_peripheral\ble_app_template\GCC`

2. 通过系统的命令行工具进入Makefile目录，输入`make`命令即可自动编译：

```
make
```

如果命令行打印输出以下类似的信息（不同工程具体信息不相同），表示编译成功：

```
 arm-none-eabi-gcc  ../../../../../components/sdk/ble.c
 arm-none-eabi-gcc  ../../../../../components/profiles/common/ble_prf_utils.c
 arm-none-eabi-gcc  ../../../../../external/segger_rtt/SEGGER_RTT.c
 arm-none-eabi-gcc  ../Src/platform/user_periph_setup.c
 arm-none-eabi-gcc  ../Src/user/main.c
 arm-none-eabi-gcc  ../Src/user/user_app.c
 arm-none-eabi-gcc  ../../../../../platform/arch/arm/cortex-m/gcc/startup_gr55xx.s
compile .elf file ...
 arm-none-eabi-gcc  out/obj/gr_system.o
compile binary file ...
 arm-none-eabi-objcopy  out/lst/ble_app_template.elf
compile hex file ...
 arm-none-eabi-objcopy  out/lst/ble_app_template.elf
```

图 2-14 编译成功显示

编译构建成功后，在out目录下输出*$(project_name).bin*和*$(project_name).hex*文件以及存放编译过程文件的文件夹lst和obj。

```
ls out
ble_app_template.bin  ble_app_template.hex  lst  obj
```

图 2-15 Linux下make输出文件

图 2-16 Windows下make输出文件

2.5 下载固件

2.5.1 使用GProgrmmer下载固件

在Linux和Windows平台上，可在GCC目录下使用GProgrammer图形化界面烧写工具进行固件下载。

1. 安装GProgrammer

   GProgrammer（Windows）的安装步骤可参考《GProgrammer用户手册》。

   GProgrammer（Linux）安装步骤如下：

   (1) 选择安装目录，解压GProgrammer for Linux的绿色版安装文件*GProgrammer_linux_x64_version.tar.bz2*，解压后的文件目录如图 2-17所示。

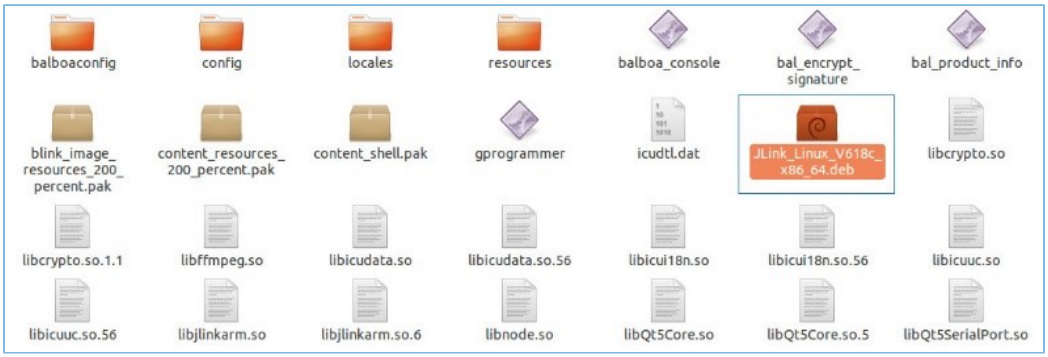

图 2-17 解压后的文件目录

说明:

version表示GProgrammer当前版本号。

---

   (2) 解压后需要将GProgrammer（Linux）所在的安装路径添加到环境变量中。

   (3) 若用户之前没有安装J-Link驱动，请双击目录下的*JLink_Linux_V618c_x86_64.deb*完成JLink驱动的安装，然后再启动软件安装。

(4) 打开终端，使用cd命令进入到解压目录，输入命令sudo ./gprogrammer，根据提示输入密码后，即可启动GProgrammer软件。

2. 下载固件

使用GProgrammer（图形化界面）下载固件的详细操作，请参考《GProgrammer用户手册》。

另外，GProgrammer还支持命令行操作。SDK的Makefile 已内置了GProgrammer命令行下载功能。用户只需将GProgrammer安装路径添加到 Windows/Linux环境变量中，然后执行make flash命令，即可下载固件。

```
$ make flash
Writing out/ble_app_template.bin to the GR55xx-SK board
GR5xxx_console.exe eraseall 1 0
eraseall success.
GR5xxx_console.exe program out/ble_app_template.bin 'y' 0x200000 1024 1 0
Programming 10%
Programming 20%
Programming 30%
Programming 40%
Programming 50%
Programming 60%
Programming 70%
Programming 80%
Programming 90%
Programming 100%
program success.
```

图 2-18 使用“make flash”命令下载固件

2.5.2 使用J-Link Commander下载固件

在Windows、Linux和Mac平台上，可在GCC目录下使用J-Link Commander工具进行固件下载。

使用J-Link Commander下载固件的步骤如下：

1. 在J-Link的安装目录下，找到文件*JLinkDevices.xml*，并在该文件中添加以下内容：

```
<Device>
  <ChipInfo Vendor="Goodix" Name="GR551x" Core="JLINK_CORE_CORTEX_M4"
 WorkRAMAddr="0x30000000" WorkRAMSize="0x40000"/>
  <FlashBankInfo Name="Internal Flash" LoaderType="FLASH_ALGO_TYPE_OPEN" Loader="Devices/
Goodix/GR5xxx_16MB_Flash_Jflash.FLM" MaxSize="0x100000" BaseAddr="0x200000"/>
 </Device>
 <Device>
  <ChipInfo Vendor="Goodix" Name="GR5526" Core="JLINK_CORE_CORTEX_M4"
 WorkRAMAddr="0x20000000" WorkRAMSize="0x80000"/>
  <FlashBankInfo Name="Internal Flash" LoaderType="FLASH_ALGO_TYPE_OPEN" Loader="Devices/
Goodix/GR5xxx_16MB_Flash_Jflash.FLM" MaxSize="0x100000" BaseAddr="0x200000"/>
 </Device>
 <Device>
  <ChipInfo Vendor="Goodix" Name="GR5x25" Core="JLINK_CORE_CORTEX_M4"
 WorkRAMAddr="0x20000000" WorkRAMSize="0x40000"/>
  <FlashBankInfo Name="Internal Flash" LoaderType="FLASH_ALGO_TYPE_OPEN" Loader="Devices/
Goodix/GR5xxx_16MB_Flash_Jflash.FLM" MaxSize="0x100000" BaseAddr="0x200000"/>
```

```
 </Device>
 <Device>
  <ChipInfo Vendor="Goodix" Name="GR533x" Core="JLINK_CORE_CORTEX_M4"
 WorkRAMAddr="0x20000000" WorkRAMSize="0x18000"/>
  <FlashBankInfo Name="Internal Flash" LoaderType="FLASH_ALGO_TYPE_OPEN" Loader="Devices/
Goodix/GR5xxx_16MB_Flash_Jflash.FLM" MaxSize="0x80000" BaseAddr="0x200000"/>
 </Device>
 <Device>
  <ChipInfo Vendor="Goodix" Name="GR5405" Core="JLINK_CORE_CORTEX_M4"
 WorkRAMAddr="0x20000000" WorkRAMSize="0x18000"/>
  <FlashBankInfo Name="Internal Flash" LoaderType="FLASH_ALGO_TYPE_OPEN" Loader="Devices/
Goodix/GR5xxx_16MB_Flash_Jflash.FLM" MaxSize="0x80000" BaseAddr="0x200000"/>
 </Device>
```

2. 将下载算法文件SDK_Folder\build\gcc\GR5xxx_16MB_Flash_Jflash.FLM添加到JLink_Folder\JLink\Devices\Goodix目录。

3. 打开J-Link Commander工具，键入“connect”，然后输入“?”，进行芯片选择，如图 2-19所示。例如，若目标芯片为GR5405，则在“Target device settings”中选择“GR5405”，如图 2-20所示。

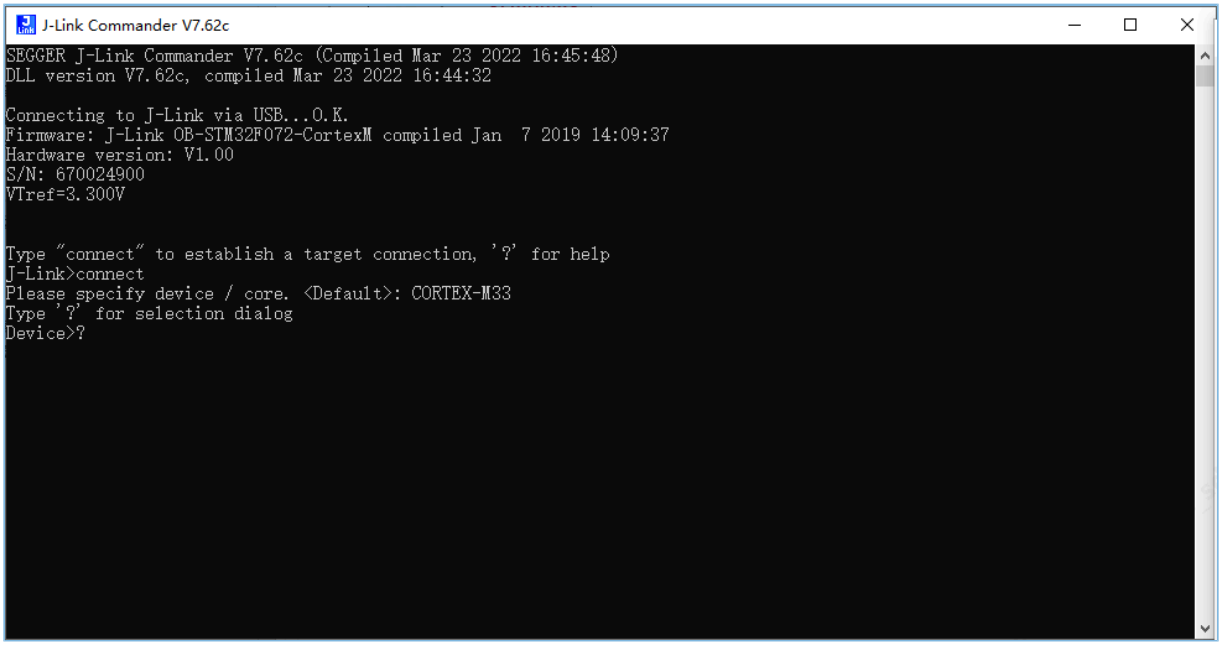

图 2-19 J-Link Commander界面

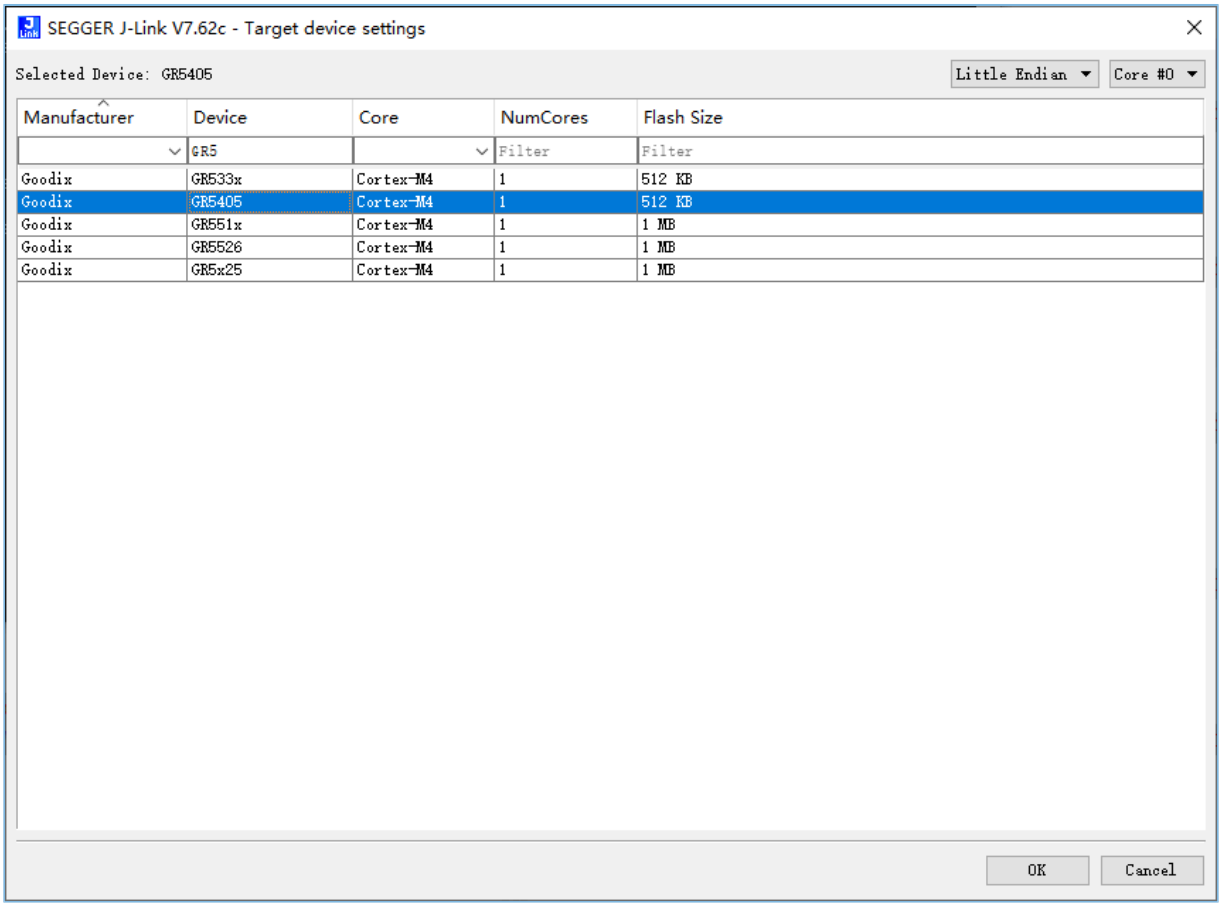

图 2-20 芯片选择界面

4. 使用`loadfile`命令下载固件。

```
loadfile xxx.hex
```

或

```
loadfile xxx.bin 0x00202000
```

说明:

- 上述两种命令，任选其中一种使用。
- “0x00202000”为*custom_config.h*中配置的“APP_CODE_LOAD_ADDR”宏的值。

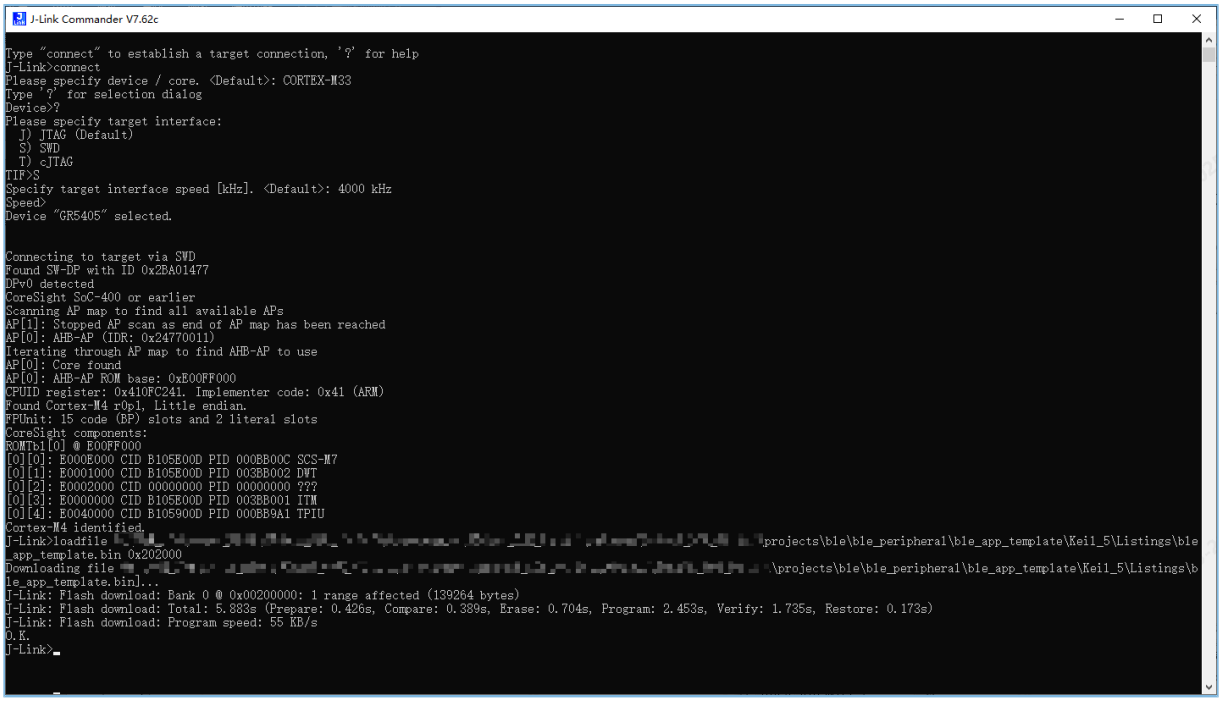

图 2-21 使用“loadfile”命令下载固件

5. 运行固件。

   固件下载完成后，可按下开发板上的复位按键，或在“J-Link Command”界面依次键入`r`、`g`命令，复位设备并运行固件。

图 2-22 使用“r”、“g”命令运行固件

2.6 构建新应用工程

如果用户需基于GR5xx开发新的应用，可参考以下步骤：

1. 用户可以根据编程习惯自由构建应用工程基础框架，也可选用如下方式构建新应用工程的基础框架。

   - 减法构建模式：在`SDK_Folder\projects`示例工程中寻找需求近似的应用工程，目录名修改为目标应用工程名，并更新keil工程文件名，保留工程需要继续引用的文件，移除不再使用的文件。使用*keil2makefile.py*脚本工具，生成新应用工程的Makefile初始文件。
   - 加法构建模式：参考模板应用工程的目录结构，构建新应用工程的目录结构，复制已存在的Makefile文件（如ble_app_template工程下），保留Makefile公共配置部分，移除源文件、头文件设置，后续再按工程实际进行设置。

2. 按项目需要对新应用工程进行源码开发，可增加、删除、修改源文件或头文件。
3. 根据新工程的文件依赖关系，修改Makefile中包含的源文件和头文件引用。
4. 根据实际项目需要，修改编译和链接等参数。
5. 执行make命令进行交叉编译生成*.hex/.bin*文件。用户可将*.hex/.bin*文件下载至开发板进行测试验证。