GR5xx GCC User Manual

Version: 1.0

Release Date: 2023-12-27

Shenzhen Goodix Technology Co., Ltd.

Trademarks and Permissions

GOODiX and other Goodix trademarks are trademarks of Shenzhen Goodix Technology Co., Ltd. All other trademarks and trade names mentioned in this document are the property of their respective holders.

Disclaimer

Information contained in this document is intended for your convenience only and is subject to change without prior notice. It is your responsibility to ensure its application complies with technical specifications.

Shenzhen Goodix Technology Co., Ltd. (hereafter referred to as “Goodix”) makes no representation or guarantee for this information, express or implied, oral or written, statutory or otherwise, including but not limited to representation or guarantee for its application, quality, performance, merchantability or fitness for a particular purpose. Goodix shall assume no responsibility for this information and relevant consequences arising out of the use of such information.

Without written consent of Goodix, it is prohibited to use Goodix products as critical components in any life support system. Under the protection of Goodix intellectual property rights, no license may be transferred implicitly or by any other means.

Shenzhen Goodix Technology Co., Ltd.

Headquarters: Floor 12-13, Phase B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828 Zip Code: 518000

Website: www.goodix.com

Preface

Purpose

This document introduces the methods to establish integrated development environments (IDEs) for cross compilation of GR5xx System-on-Chips (SoCs) in command-line interface with GNU Compiler Collection (GCC) and makefiles on Linux and Windows, to help users quickly get started with secondary development of GR5xx Software Development Kit (SDK) applications.

Audience

This document is intended for:

- Device user
- Developer
- Test engineer
- Hobbyist developer
- Technical writer

Release Notes

This document is the initial release of *GR5xx GCC User Manual*, corresponding to Bluetooth Low Energy GR5xx SoC series.

Revision History

Version	Date	Description
1.0	2023-12-27	Initial release

Contents

Preface I

1 Introduction 1

2 Setting up Compiling Environment 2

2.1 Preparation 2
2.2 Installation 3
2.2.1 On Linux 3
2.2.1.1 Downloading GCC 3
2.2.1.2 Installing GCC 3
2.2.1.3 Setting Environment Variables (On Linux) 3
2.2.1.4 Testing GCC Installation Result (On Linux) 4
2.2.1.5 Installing Python (On Linux) 4
2.2.1.6 Installing J-Link (On Linux) 5
2.2.2 On Windows 5
2.2.2.1 Downloading MSYS and GCC 5
2.2.2.2 Installing MSYS and GCC 5
2.2.2.3 Setting Environment Variables (On Windows) 6
2.2.2.4 Testing GCC Installation Result (On Windows) 7
2.2.2.5 Installing Python (On Windows) 7
2.2.2.6 Installing J-Link (On Windows) 8
2.3 Development Board Connection and Testing 8
2.4 Compiling SDK Application Example Projects 9
2.4.1 Makefile 9
2.4.2 Generating Makefile 10
2.4.3 Modifying Makefile Configuration 10
2.4.4 Executing make Compilation 12
2.5 Downloading Program 12
2.6 Building a New Application Project 13

1 Introduction

GNU Compiler Collection (GCC) is an open-source, cross-platform compiler system developed by the GNU Project running on Linux and Windows. The arm-none-eabi-gcc cross compiler is based on GCC, and supports the instruction sets of ARM CPUs, making it an ideal choice for GR5xx System-on-Chips (SoCs).

In software development, make is a build automation tool that automatically compiles and links the project source files based on makefiles. Makefiles specify the rules of compiling and linking multiple project source files with compilers, and enable users to call and to execute system commands.

This document introduces the approaches for building the development environment for GR5xx SoCs with GCC and makefiles in Ubuntu, a Linux distribution, and on Windows. The document also provides users with examples.

Before getting started, you can refer to the following documents.

Table 1-1 Reference documents

Name	Description
Developer guide of the specific GR5xx SoC	Introduces GR5xx Software Development Kit (SDK) and how to develop and debug applications based on the SDK.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at https://www.segger.com/downloads/jlink/UM08001_JLink.pdf.
Bluetooth Core Spec	Offers official Bluetooth standards and core specification from Bluetooth SIG.
Bluetooth GATT Spec	Provides details about Bluetooth profiles and services. Available at https://www.bluetooth.com/specifications/gatt.
GCC	Provides more information about GCC. Available at https://launchpad.net/gcc-arm-embedded.
GNU make	Provides a makefile developing guide. Available at https://www.gnu.org/software/make/manual/make.html.

2 Setting up Compiling Environment

This chapter introduces environment setup for GR5xx cross compilation and development on Linux OS (Ubuntu) and Windows.

2.1 Preparation

Before setting up the compiling environment, users shall get the following items ready.

- **Software preparation**

Table 2-1 Software preparation (Linux)

Name	Description
Ubuntu (Linux)	Ubuntu 16.04 LTS or later LTS versions (both 32-bit and 64-bit versions are acceptable)
gcc-arm-none-eabi	Used for GR5xx cross compilation of the target executable code Version: *gcc-arm-none-eabi-5_4-2016q3-linux.tar.bz2* Available at https://launchpad.net/gcc-arm-embedded/+download.
Python	Used to build environment for script execution for GR5xx application projects Version: Python 3.0 or later versions Available at https://www.python.org/downloads.
J-Link	Version: J-Link Software and Documentation pack for Linux, DEB installer Available at https://www.segger.com/downloads/jlink/. **Note:** ◦ Choose the versions that are compatible with your Ubuntu operating systems. ◦ Use J-Link 6.10a or later versions.

Note:

◦ Ubuntu 16.04 LTS or later LTS versions are recommended. As GCC operates on Ubuntu, it is recommended to install the GCC version and the Ubuntu version as recommended in this document, to simplify compilation and development.

◦ If you choose other Linux distributions, environment-related problems may occur.

Table 2-2 Software preparation (Windows)

Item	Description
MSYS	MSYS is a lightweight GNU development environment running on Windows, providing functionalities of Unix tools including bash, make, mkdir, and grep. Version: MSYS-1.0.11 Available at https://nchc.dl.sourceforge.net/project/mingw/MSYS/Base/msys-core/msys-1.0.11/MSYS-1.0.11.exe?viasf=1.

Item	Description
gcc-arm-none-eabi	Used for GR5xx cross compilation of the target executable code Version: gcc-arm-none-eabi-9-2020-q2-update-win32 Available at https://developer.arm.com/-/media/Files/downloads/gnu-rm/9-2020q2/gcc-arm-none-eabi-9-2020-q2-update-win32.zip.
Python	Used to build environment for script execution for GR5xx application projects Version: Python 3.0 or later versions Available at https://www.python.org/downloads.
J-Link	Provides the support library for J-Link hardware driver and software operations. Version: J-Link Software and Documentation pack for Windows Available at https://www.segger.com/downloads/jlink/. **Note:** Use J-Link 6.10a or later versions.

- **Hardware preparation**

Table 2-3 Hardware preparation

Name	Description
Development board	Starter Kit Board (SK Board) of the corresponding SoC
Connection cable	USB Type-C cable (Micro USB 2.0 cable for GR551x SoCs)

2.2 Installation

Before compiling ARM programs, users shall install the cross compiler, gcc-arm-none-eabi.

2.2.1 On Linux

This section expounds the steps to install GCC, Python, and J-Link on Linux.

2.2.1.1 Downloading GCC

Get the installation package (*gcc-arm-none-eabi-5_4-2016q3-linux.tar.bz2*).

This installation package is designed for 32-bit Linux. If you need a 64-bit version, download 64-bit Linux Tarball.

2.2.1.2 Installing GCC

The installation package is compilation-free. Extract the file to the correct directory.

Run the following command to extract the installation package.

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

2.2.1.3 Setting Environment Variables (On Linux)

Add the actual path of gcc-arm-none-eabi to the PATH environment variable, based on the practice. Examples are provided below.

- Root users

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

- Non-root users

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

2.2.1.4 Testing GCC Installation Result (On Linux)

After GCC installation, test whether GCC is installed successfully by running the following command.

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

When the following information is printed on the terminal, GCC is successfully installed.

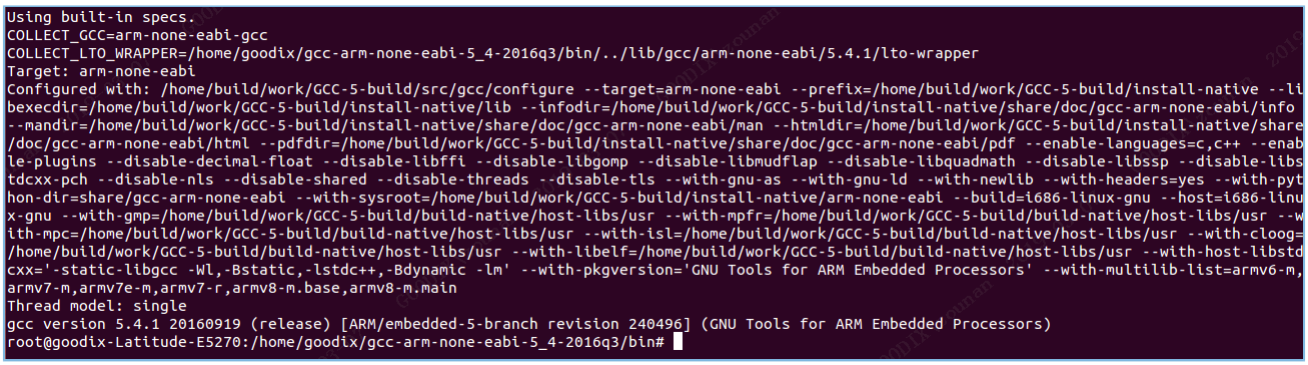

Figure 2-1 GCC installation result

Note:

- The Ubuntu compiler arm-none-eabi-gcc provides a universal version for both 32-bit or 64-bit versions.
- When users run `arm-none-eabi-*` commands on some Ubuntu LTS releases, if the third-party library ia32-libs is absent, an error message will be displayed: “no such file or directory”. This can be solved by running the following commands:

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

2.2.1.5 Installing Python (On Linux)

1. Download the installation package of Python 3. Choose a version that is compatible with your Ubuntu system.

   Enter the following command to install Python 3:

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


2. Run Python.

```
python
```

3. If the installation is successful, the version information of Python is shown as follows.

```
$ python
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 2-2 Python installation result

Note:

Python 3.6.7 is installed as an example. When it prints Python 3.6.7 on the terminal, the installation is successful.

2.2.1.6 Installing J-Link (On Linux)

Download J-Link for Linux on the official website of SEGGER.

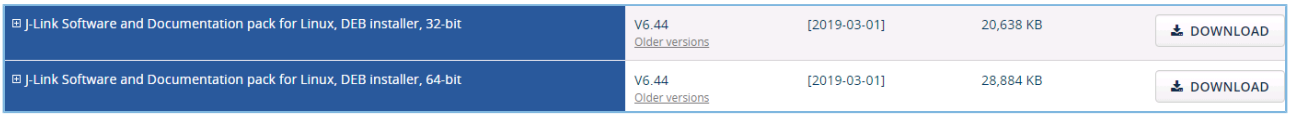

Figure 2-3 Downloading J-Link for Linux from SEGGER official website

Install the DEB package of J-Link for Linux on Ubuntu.

Note:

- The J-Link version should be compatible with the Ubuntu version.
- Install J-Link 6.10a or later versions.
- After installation, run the `JLinkExe` command in the command-line interface, and J-Link is ready. If not, check whether the environment variable has been successfully added.

2.2.2 On Windows

This section expounds the steps to install MSYS, GCC, Python, and J-Link on Windows.

2.2.2.1 Downloading MSYS and GCC

Click to get the executable package of MSYS: *MSYS-1.0.11.exe*.

Click to get the installation package of GCC: *gcc-arm-none-eabi-9-2020-q2-update-win32.zip*.

2.2.2.2 Installing MSYS and GCC

1. Double-click *MSYS-1.0.11.exe* to install MSYS. It shows as follows after installation completes.

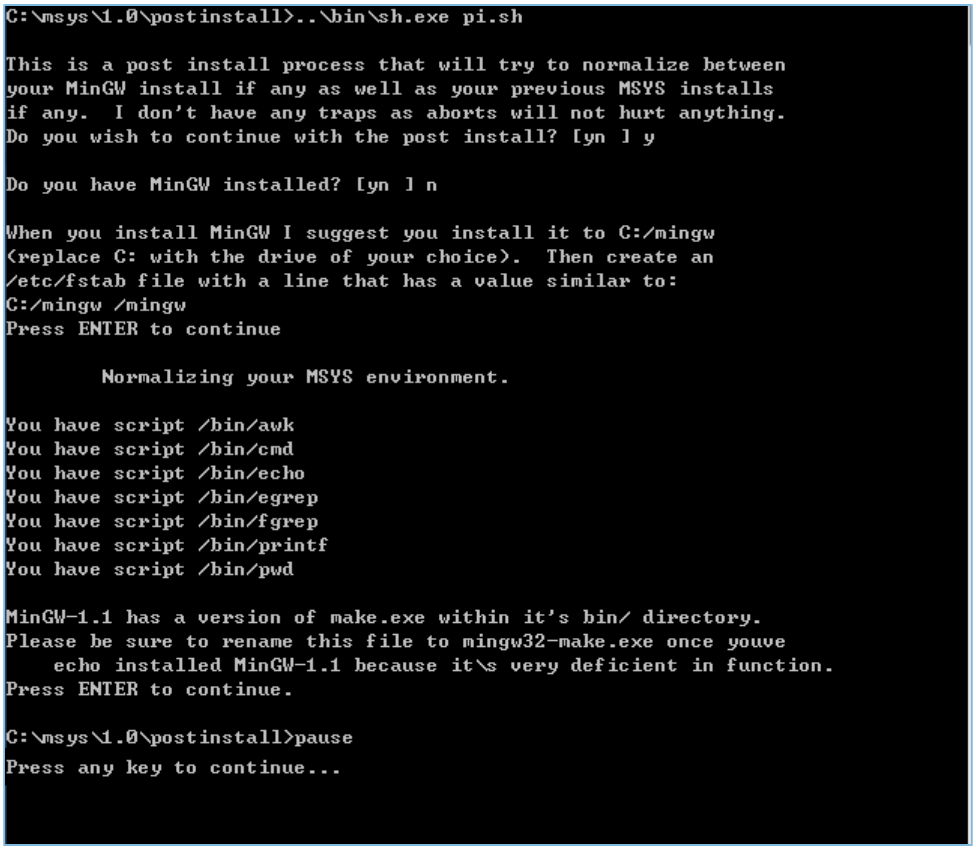

Figure 2-4 MSYS successful installation

2. Extract the installation-free package *gcc-arm-none-eabi-9-2020-q2-update-win32.zip* to the suitable path that you see fit.

2.2.2.3 Setting Environment Variables (On Windows)

Add the paths of the two pieces of software as environment variables by entering **Advanced system settings** in **Control Panel** > **System**. Click **Advanced** and choose **Path** in the **System variables** list. Click **Edit**.

- MSYS path: <MSYS_*installation path*>\bin
- GCC path: <GCC Win32_*installation path*>\bin

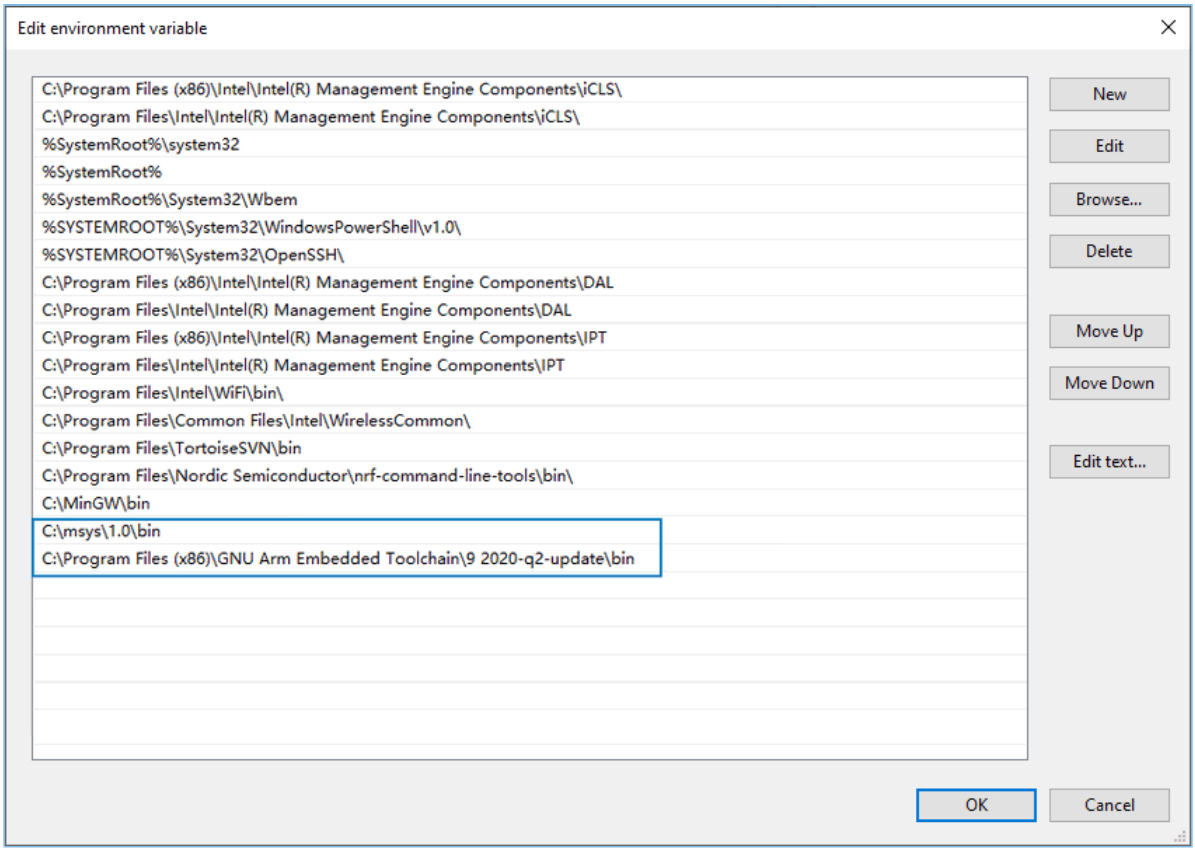

Figure 2-5 Setting environment variables on Windows 10

2.2.2.4 Testing GCC Installation Result (On Windows)

After GCC installation, enter `make -v` to view the version information of the make tool (provided by MSYS).

```
$ 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.
```

Figure 2-6 To view version information of GNU Make

Enter `arm-none-eabi-gcc -v` to view the version information of GCC.

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

Figure 2-7 To view GCC version information

2.2.2.5 Installing Python (On Windows)

1. Download the installation package of Python 3. Choose a version that is compatible with your Windows OS.
2. Install Python by following steps of the installation wizard.

3. Set the environment variables.

After installation completes, enter `python` in **Command Prompt** to view the version information of Python.

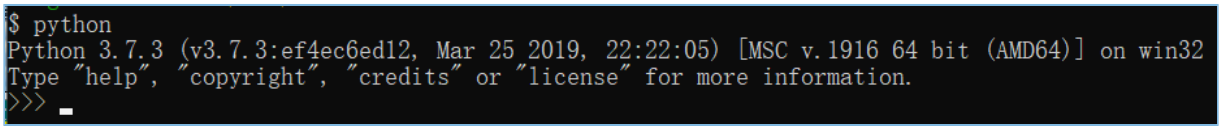

Figure 2-8 To view Python version information

2.2.2.6 Installing J-Link (On Windows)

Download the J-Link for Windows on the official website of SEGGER.

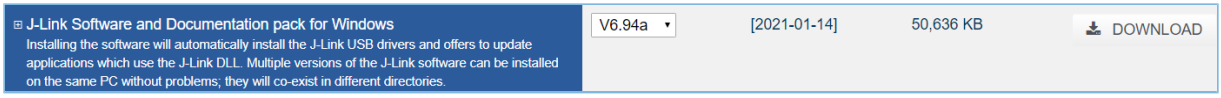

Figure 2-9 Downloading J-Link for Windows from SEGGER official website

Double-click the downloaded installation package *JLink_Windows_Version.exe* and install J-Link by following steps in the installation wizard.

Note:

- Install J-Link 6.10a or later versions.
- *Version* shows the J-Link version number.

2.3 Development Board Connection and Testing

After the installation completes, users can connect the development board to the PC and perform tests.

- **On Linux**

  Ensure J-Link path is included in the environment variables. Enter the following commands on the terminal in sequence (texts after # are command annotations):

```
JLinkExe        #Launch J-Link tools.
connect         #Connect to the development board with the connect command. Before running
 the command, make sure the development board is accessible.
CORTEX-M4       #Define the model of the CPU core. If the model can be identified by J-Link
 tools, press Enter.
S               #Choose the debug interface for hardware connection debugging. S stands for
 Serial Wire Debug (SWD).
4000            #Specify the data rate of SWD (unit: kHz), which is set to 4,000 kHz here.
```

  When it shows **Cortex-M4 identified**, the PC is successfully connected with the SK Board through J-Link.

```
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>
```

Figure 2-10 Successful connection through J-Link

- **On Windows**

  When the SK Board is connected to a Windows PC, open **Device Manager** to check whether J-Link is shown in the **Ports (COM & LPT)** list. J-Link is successfully connected if it is in the list; if J-Link is not detected, check whether the J-Link driver is installed successfully, and reinstall the J-Link driver in the latest version if needed.

2.4 Compiling SDK Application Example Projects

This section introduces the creation, application, and compilation of makefiles by taking the application example project, ble_app_template, as an example. The steps below are applicable for operations on both Linux and Windows PCs.

Note:

SDK_Folder is the root directory of GR5xx SDK.

2.4.1 Makefile

At present, GR5xx SDK provides example projects with scripting tools to generate makefiles.

Makefiles define the compilation rules of Make, which enable the execution of GCC commands (to compile and to link) and OS commands. A makefile contains a set of directives, including properties of a compiler, sequence of file compilation, rules for compiling and linking, and dependencies between targets and between targets and source files. Executable files are generated by executing the make command.

2.4.2 Generating Makefile

By default, the application example projects in a GR5xx SDK are compiled in Keil µVision5 IDE. If users wish to compile application example projects (except ble_app_template) by using GCC toolchain, you can choose the scripting tool, *keil2makefile.py*, to convert Keil project files **.uvprojx* to makefiles and generate .lds files.

Instructions for using *keil2makefile.py*:

1. By default, the tool file *keil2makefile.py* is under the directory SDK_`Folder\build\gcc`. (Note: It is not *keil2make.py*.)
2. The *keil2makefile.py* script and the **.uvprojx* file shall be under the same directory while in use, to ensure the paths of source files and header files that makefiles refer to after file conversion are correct.
3. Copy the *keil2makefile.py* file to the Keil_5 directory of the target application project. For ble_app_template, copy the script to the directory: SDK_`Folder\projects\ble\ble_peripheral\ble_app_template\keil_5`.
4. Leave the command-line interface and enter the target path. Run the command provided as follows. The project ble_app_template is used as an example in this chapter. You can see the command, makefile, and .lds files as follows:

```
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 ...
```

Figure 2-11 Generated makefile and .lds file

5. After successfully converting the files to makefiles and .lds files, put the makefiles under the GCC directory, which is of the same level with Keil_5. Users can access the makefiles under the GCC directory.

2.4.3 Modifying Makefile Configuration

A set of default parameters for compiling and linking are provided in “Common Configuration Area” in the newly converted makefiles. Users can modify the compilation parameters, based on the practice of a project. Modify parameters with caution, to avoid failures in compiling projects.

In addition to the makefiles, users can create custom *.c* files or *.h* files by taking the approaches provided below (applicable to both Linux and Windows).


```
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  \
```

Figure 2-12 Creating *.c* files

```
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  \
```

Figure 2-13 Creating *.h* files

2.4.4 Executing make Compilation

1. Enter the directory path of the makefiles of the target example project. For ble_app_template, find the makefiles in SDK_Folder\projects\ble\ble_peripheral\ble_app_template\GCC.
2. Enter the directory of makefiles with the command line of the system. Enter the `make` command to start automatic compilation.

```
make
```

When information similar to the following (details vary for different projects) is printed in the command-line interface, the compilation is successful.

```
 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
```

Figure 2-14 Interface for successful compilation

When the compilation is finished, *$(project_name).bin* and *$(project_name).hex*, as well as folders lst and obj, which store the process files of compilation, are generated under the out directory.

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

Figure 2-15 Files created by `make` command on Linux

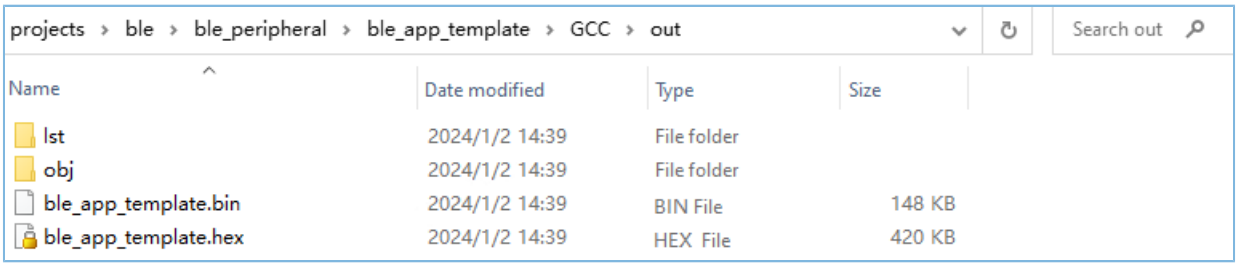

Figure 2-16 Files created by `make` command on Windows

2.5 Downloading Program

Users can download programs with GProgrammer (a GUI programming tool supported both on Linux and Windows PCs) under the GCC directory.

To install GProgrammer on Windows, see details in *GProgrammer User Manual*.

To install GProgrammer on Linux, follow the steps below.

1. Extract *GProgrammer_linux_x64_version.tar.bz2*, the portable GProgrammer package, to a specified location. The extracted files are shown in the figure below.

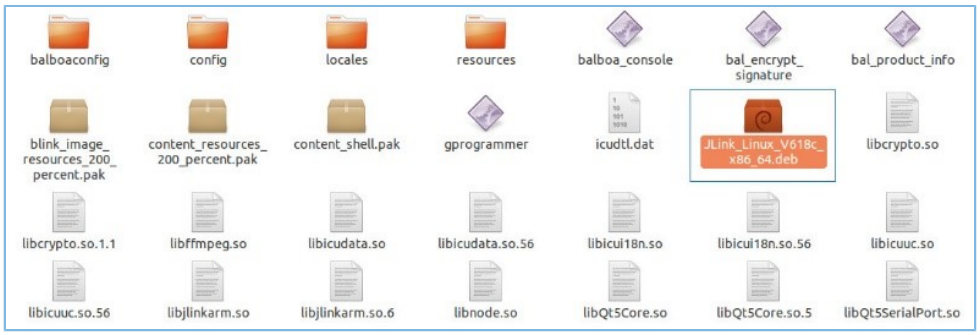

Figure 2-17 Files extracted from *GProgrammer_linux_x64_version.tar.bz2*

Note:

version in the package name indicates the current version number of GProgrammer.

2. Add the directory of the extracted GProgrammer for Linux as an environment variable.
3. If you need to install J-Link, double-click *JLink_Linux_V618c_x86_64.deb* and then start installing GProgrammer.
4. On the Linux-based PC, use the `cd` command to enter the location for file extraction. Enter `sudo ./gprogrammer` and input the password as prompted. Then, GProgrammer is started.

So far, the cross compilation environment for GR5xx application projects on Linux/Windows is successfully set up. Users can modify, compile, download, and test the example projects of GR5xx SDK.

2.6 Building a New Application Project

Users can follow the instructions below to develop new applications with GR5xx:

1. Users are free to build the underlying framework of application projects based on their own programming habits. Two examples of building a new underlying framework of application projects are provided below.
   - Subtraction build model: Find an application project similar to the one to be dealt with under `SDK_Folder\projects`. Rename the folder with the name of the target application project and update the directory; update the Keil project files; keep the files that the project needs for reference; remove files that are not useful. Generate initial makefiles for the new application project with the scripting tool *keil2makefile.py*.
   - Addition build model: Refer to the directory structure of the template application projects, and build the directory structure for the new application projects; copy the existing makefiles (such as the makefiles of the ble_app_template project); keep the common configurations in makefiles; invalidate the settings of the source files and the header files, and make settings based on future demands.
2. Develop source code for new application projects based on demands. Users can add, delete, or modify source files or header files.

3. Modify the references of source files and header files in makefiles, based on the dependencies of the new project files.
4. Modify parameters of compiling and linking based on demands.
5. Run the `make` command to perform cross compilation, and to generate .hex/.bin files. Users can download the .hex/.bin files to an SK Board for test and verification.