GR551x GCC User Manual

Version: 1.7

Release Date: 2020-12-23

Shenzhen Goodix Technology Co., Ltd.

Shenzhen Goodix Technology Co., Ltd.

Headquarters: 2F. & 13F., Tower B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828 FAX: +86-755-33338099

Website: www.goodix.com

Preface

Purpose

This document introduces the methods to establish integrated development environments (IDEs) for cross compilation of GR551x System on Chips (SoCs) in command-line interface with GNU Compiler Collection (GCC) and makefiles on Linux operating system, to help users quickly get started with secondary development of GR551x SDK applications.

Audience

This document is intended for:

- GR551x user
- GR551x developer
- GR551x tester
- Technical writer

Release Notes

This document is the fifth release of *GR551x GCC User Manual*, corresponding to GR551x SoC series.

Revision History

Version	Date	Description
1.0	2019-12-08	Initial release
1.3	2020-03-16	Updated the Python version number in “Section 2.3 Install Python”.
1.5	2020-05-30	Added descriptions on GProgrammer for Linux in “Section 2.7 Download Program”.
1.6	2020-06-30	Updated the document version based on SDK changes.
1.7	2020-12-23	Deleted the FAQ "Why does 'Conflicting CPU architectures' occur when establishing link?".

Contents

1 Introduction

GNU Compiler Collection (GCC) is an open-source, cross-platform compiler system developed by the GNU Project running on Linux operating system. 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 GR551x SoCs.

In software development, Make is a build automation tool that automatically compiles and links the project source files based on the 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 GR551x SoCs with GCC and makefiles in Ubuntu, a Linux distribution. The document also provides users with examples.

Before you get started, it is recommended to refer to the following documents.

Table 1-1 Reference documents

Name	Description
GR551x Developer Guide	Introduces the software/hardware and quick start guide of GR551x SoCs.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at www.segger.com/downloads/jlink/UM08001_JLink.pdf.
Bluetooth Core Spec v5.1	Offers official Bluetooth standards and core specification (v5.1) from Bluetooth SIG. Available at https://www.bluetooth.com/specifications/bluetooth-core-specification/.
Bluetooth GATT Spec	Provides details about Bluetooth profiles and services. Available at 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 Set up Compiling Environment

This chapter introduces environment setup for GR551x cross compilation and development on Linux OS (Ubuntu).

To set up the environment for GR551x cross compilation and development, the following tools are required:

- gcc-arm-none-eabi cross compiler: to cross-compile the executable object code of GR551x
- Python: to set up the environment for the script execution of GR551x application projects
- J-Link: to debug and program GR551x firmware

2.1 Preparation

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

Table 2-1 Software preparation

Name	Description
Ubuntu	Ubuntu 16.04 LTS or later LTS versions (both 32-bit and 64-bit versions are acceptable)
gcc-arm-none-eabi	Version: *gcc-arm-none-eabi-5_4-2016q3-linux.tar.bz2* Available at https://launchpad.net/gcc-arm-embedded/+download.
Python	Version: Python 3.0 or later versions Available at www.python.org/downloads.
J-Link	Version: J-Link Software and Documentation pack for Linux, DEB installer Available at 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.

2.2 Install GCC

Before compiling ARM programs on Linux, users shall install the cross compiler, gcc-arm-none-eabi. This chapter describes the steps of installing GCC.

2.2.1 Download GCC

Click https://launchpad.net/gcc-arm-embedded/+download and 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, click https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads and download 64-bit Linux Tarball.

2.2.2 Install

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
```

2.2.3 Set Environment Variables

Add the 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

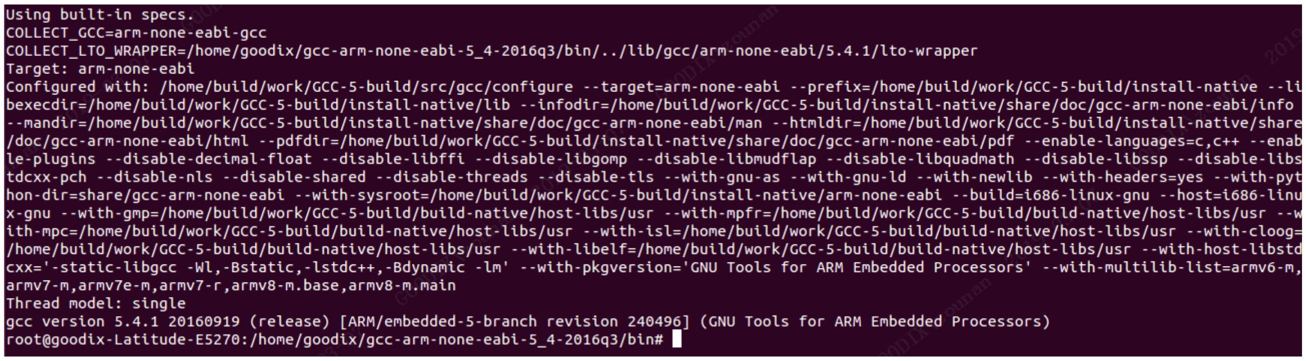
```
echo "export PATH=$PATH:/home/goodix/gcc-arm-none-eabi-5_4-2016q3/bin" >> ~/.bashrc
source ~/.bashrc
```

2.2.4 Test GCC Installation Result

After installation, test whether the installation is successful by running the following command.

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

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



```
Using built-in specs.
COLLECT_GCC=arm-none-eabi-gcc
COLLECT_LTO_WRAPPER=/home/goodix/gcc-arm-none-eabi-5_4-2016q3/bin/../lib/gcc/arm-none-eabi/5.4.1/lto-wrapper
Target: arm-none-eabi
Configured with: /home/build/work/GCC-5-build/src/gcc/configure --target=arm-none-eabi --prefix=/home/build/work/GCC-5-build/install-native --li
bexecdir=/home/build/work/GCC-5-build/install-native/lib --infodir=/home/build/work/GCC-5-build/install-native/share/doc/gcc-arm-none-eabi/info
--mandir=/home/build/work/GCC-5-build/install-native/share/doc/gcc-arm-none-eabi/man --htmldir=/home/build/work/GCC-5-build/install-native/share
/doc/gcc-arm-none-eabi/html --pdfdir=/home/build/work/GCC-5-build/install-native/share/doc/gcc-arm-none-eabi/pdf --enable-languages=c,c++ --enab
le-plugins --disable-decimal-float --disable-libffi --disable-libgomp --disable-libmudflap --disable-libquadmath --disable-libssp --disable-libs
tdcxx-pch --disable-nls --disable-shared --disable-threads --disable-tls --with-gnu-as --with-gnu-ld --with-newlib --with-headers=yes --with-pyt
hon-dir=share/gcc-arm-none-eabi --with-sysroot=/home/build/work/GCC-5-build/install-native/arm-none-eabi --build=i686-linux-gnu --host=i686-linu
x-gnu --with-gmp=/home/build/work/GCC-5-build/build-native/host-libs/usr --with-mpfr=/home/build/work/GCC-5-build/build-native/host-libs/usr --w
ith-mpc=/home/build/work/GCC-5-build/build-native/host-libs/usr --with-isl=/home/build/work/GCC-5-build/build-native/host-libs/usr --with-cloog=
/home/build/work/GCC-5-build/build-native/host-libs/usr --with-libelf=/home/build/work/GCC-5-build/build-native/host-libs/usr --with-host-libstd
cxx='-static-libgcc -Wl,-Bstatic,-lstdc++,-Bdynamic -lm' --with-pkgversion='GNU Tools for ARM Embedded Processors' --with-multilib-list=armv6-m,
armv7-m,armv7e-m,armv7-r,armv8-m.base,armv8-m.main
Thread model: single
gcc version 5.4.1 20160919 (release) [ARM/embedded-5-branch revision 240496] (GNU Tools for ARM Embedded Processors)
root@goodix-Latitude-E5270:/home/goodix/gcc-arm-none-eabi-5_4-2016q3/bin#
```

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
  ```

2.3 Install Python

1. Click www.python.org/downloads to download the installation package of Python 3. Choose a version that is compatible with your Ubuntu system.

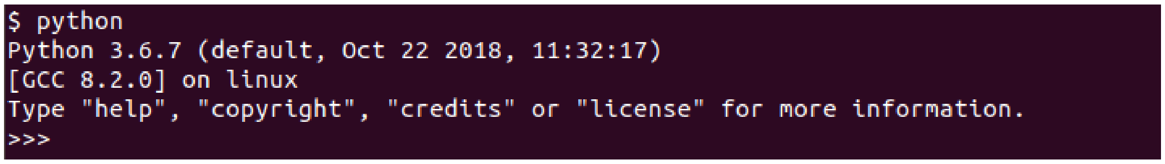
   Enter the following command to install Python 3:

```
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.
>>>
```

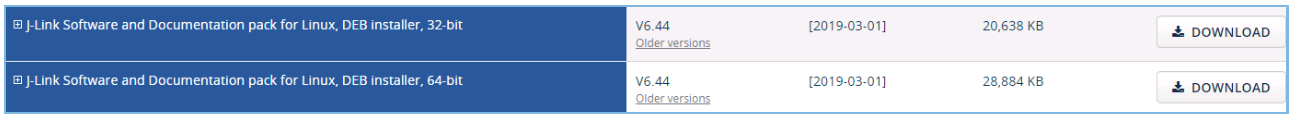
Figure 2-2 Python installation result

Note:

Python 3.7.6 is installed as an example. When it prints Python 3.7.6 on the Terminal, the installation is successful.

2.4 Install J-Link

Click www.segger.com/downloads/jlink/ and download the J-Link for Linux on the official website of SEGGAR.



⊞ J-Link Software and Documentation pack for Linux, DEB installer, 32-bit	V6.44 Older versions	[2019-03-01]	20,638 KB	DOWNLOAD
⊞ J-Link Software and Documentation pack for Linux, DEB installer, 64-bit	V6.44 Older versions	[2019-03-01]	28,884 KB	DOWNLOAD

Figure 2-3 Downloading J-Link from SEGGAR 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.

2.5 Connect to and Test Development Board

After the J-Link is successfully installed, users can connect the development board to the PC and perform tests.

2.5.1 Connect to Development Board

Connect the GR5515 Starter Kit Board (GR5515 SK Board) to a PC with a Micro USB 2.0 cable.

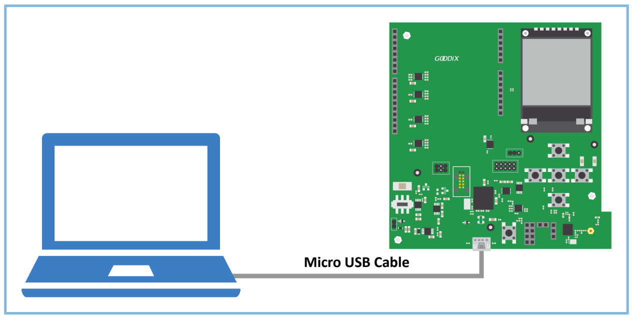


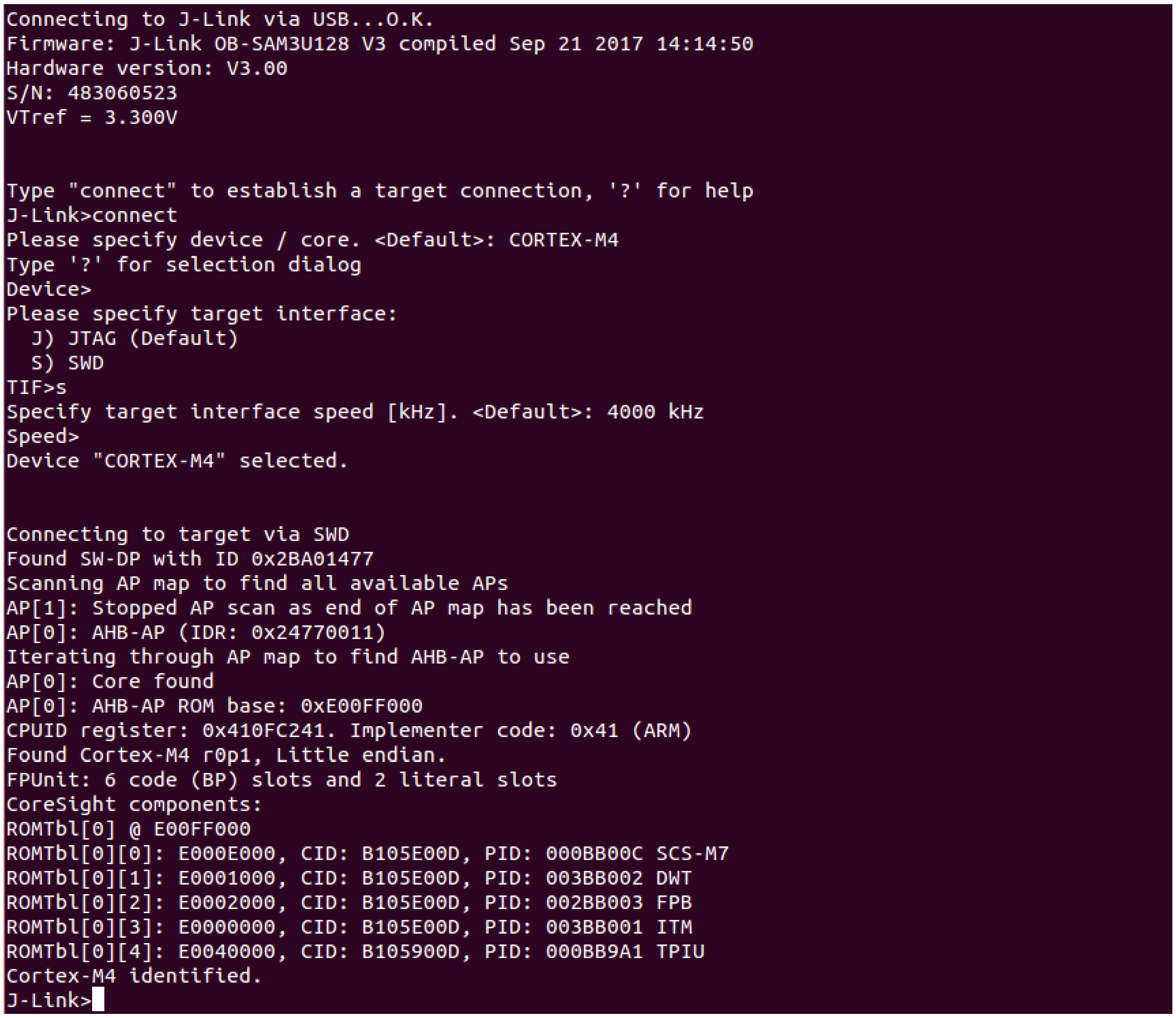
Figure 2-4 Hardware connection

2.5.2 Connect to Development Board Through J-Link

Ensure J-Link is included in the environment variables. Enter the following commands on the Terminal (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 be 4,000 kHz
 here.
```

When it shows `Cortex-M4 identified`, the PC is successfully connected with the GR5515 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 2-5 Successful connection through J-Link

2.6 Compile SDK Application Example Projects

This chapter introduces the creation, application, and compilation of makefiles by taking the application example project, ble_app_hrs, as an example.

Note:

SDK_Folder is the root directory of GR551x SDK.

2.6.1 Makefile

At present, GR551x SDK provides the makefiles for the example projects ble_app_hrs by default, to which users can refer during application tests and verifications. Makefiles for other example projects are created by using scripting tools.

The following is a reference path for the makefiles of ble_app_hrs example project:

SDK_Folder\projects\ble\ble_peripheral\ble_app_hrs\make_gcc.

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.

2.6.2 Generate Makefile

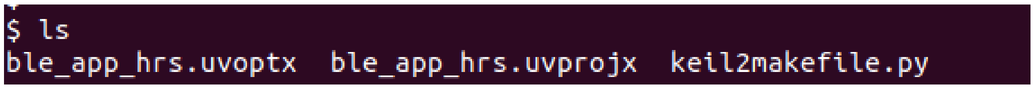
By default, the application example projects in a GR551x SDK are compiled in Keil µVision5 IDE. If users wish to compile application example projects (except ble_app_hrs) by using GCC toolchain, they can choose the scripting tool, *keil2makefile.py*, to convert Keil project files **.uvprojx* to makefiles.

Instructions for using *keil2makefile.py*:

1. By default, the tool file *keil2makefile.py* is under the directory SDK_Folder\tools\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 under the Keil_5 directory of the target application project. For ble_app_hrs, copy the script to the directory: SDK_Folder\projects\ble\ble_peripheral\ble_app_hrs\keil_5.



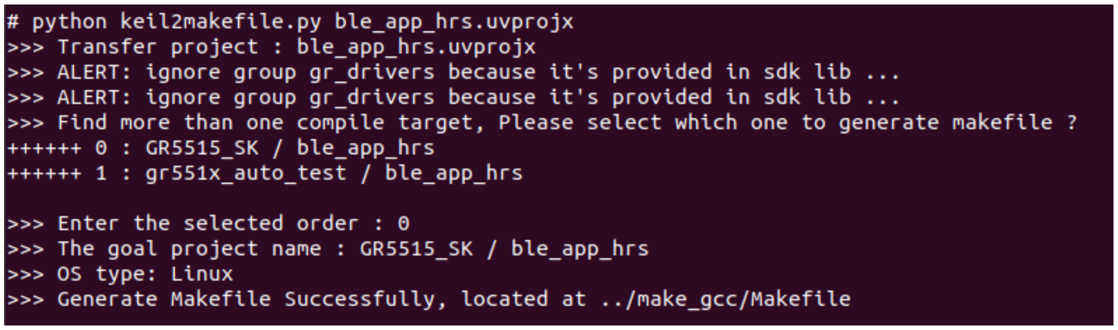
```
$ ls
ble_app_hrs.uvoptx  ble_app_hrs.uvprojx  keil2makefile.py
```

Figure 2-6 Copying *keil2makefile.py* to the Keil_5 directory

4. Leave the command-line interface and enter the target path. Run the following command. "$(project_name)" indicates the file name of the Keil project (for example, ble_app_hrs), which also applies to the content below.

```
python keil2makefile.py $(project_name).uvprojx [-C/-D]
```

5. If more than one build targets exist in the Keil source files, the scripting tools remind users to choose the target that they wish to build during conversion. In general, choose the first target (excluding test targets).



```
# python keil2makefile.py ble_app_hrs.uvprojx
>>> Transfer project : ble_app_hrs.uvprojx
>>> ALERT: ignore group gr_drivers because it's provided in sdk lib ...
>>> ALERT: ignore group gr_drivers because it's provided in sdk lib ...
>>> Find more than one compile target, Please select which one to generate makefile ?
++++++ 0 : GR5515_SK / ble_app_hrs
++++++ 1 : gr551x_auto_test / ble_app_hrs

>>> Enter the selected order : 0
>>> The goal project name : GR5515_SK / ble_app_hrs
>>> OS type: Linux
>>> Generate Makefile Successfully, located at ../make_gcc/Makefile
```

Figure 2-7 Choosing the target to be built

6. After successfully converting the files to makefiles, put the makefiles under the make_gcc directory, which is of the same level with Keil_5. Users can access the makefiles under the make_gcc directory.

2.6.3 Modify Makefile Configuration

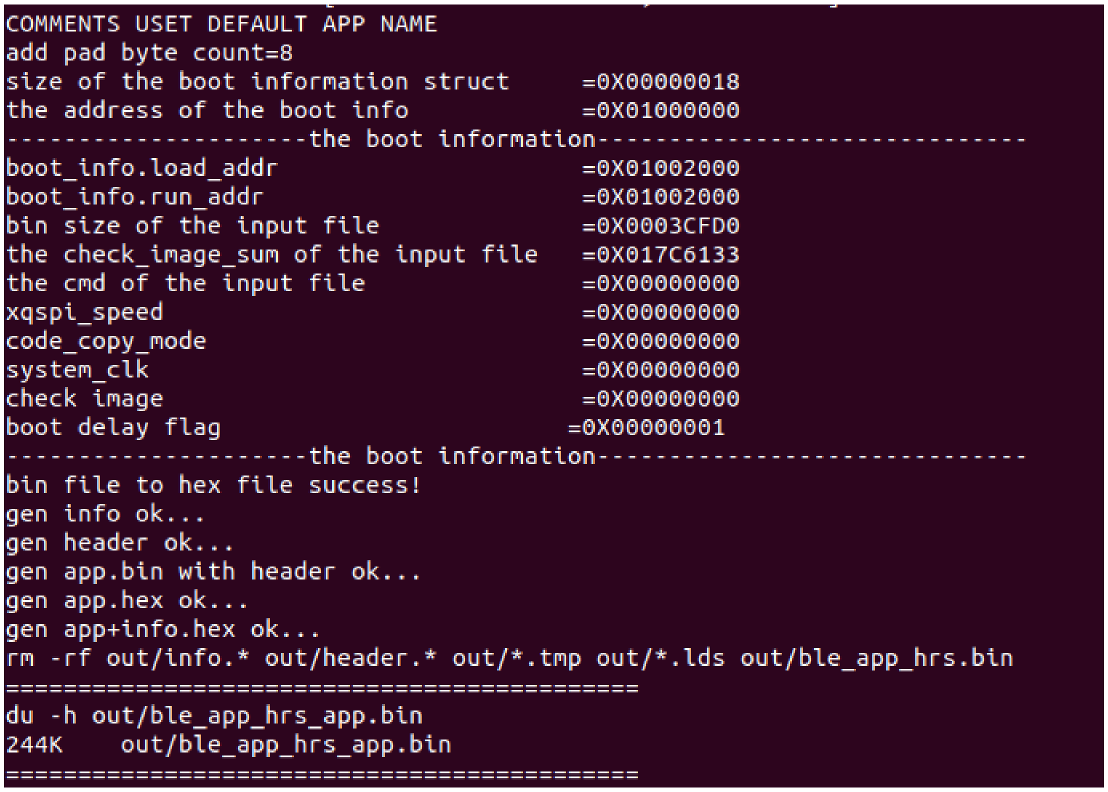
A set of default parameters for compiling and linking with makefiles are provided in GR551x application projects. Users can modify the compilation parameters, based on the practice of a project. Modify parameters with caution, to avoid failures in compiling projects.

2.6.4 Execute Make Compilation

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

```
make
```

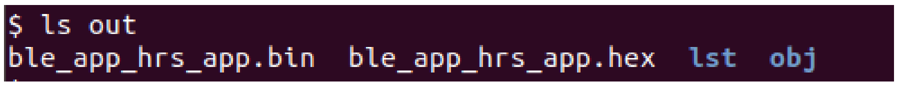
When information similar to the following (details vary for different projects) is printed in the command-line interface, the compilation is successful.



```
COMMENTS USET DEFAULT APP NAME
add pad byte count=8
size of the boot information struct     =0X00000018
the address of the boot info            =0X01000000
---------------------the boot information------------------------------
boot_info.load_addr                     =0X01002000
boot_info.run_addr                      =0X01002000
bin size of the input file              =0X0003CFD0
the check_image_sum of the input file   =0X017C6133
the cmd of the input file               =0X00000000
xqspi_speed                             =0X00000000
code_copy_mode                          =0X00000000
system_clk                              =0X00000000
check image                             =0X00000000
boot delay flag                        =0X00000001
---------------------the boot information------------------------------
bin file to hex file success!
gen info ok...
gen header ok...
gen app.bin with header ok...
gen app.hex ok...
gen app+info.hex ok...
rm -rf out/info.* out/header.* out/*.tmp out/*.lds out/ble_app_hrs.bin
============================================
du -h out/ble_app_hrs_app.bin
244K    out/ble_app_hrs_app.bin
============================================
```

Figure 2-8 Interface for successful compilation

When the compilation is finished, *$(project_name)_app.bin* and *$(project_name)_app.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_hrs_app.bin  ble_app_hrs_app.hex  lst  obj
```

Figure 2-9 Files created by `make` command

2.7 Download Program

Users can download programs with GProgrammer for Linux, a GUI tool under the make_gcc directory.

Before download, make sure GProgrammer for Linux is installed, and its directory is included in the environment variables. Follow the steps below to install GProgrammer.

1. Extract *GProgrammer_linux_x64_1.2.6.tar.bz2*, the package for GProgrammer portable version, to a specified location. The extracted files are shown in the figure below.

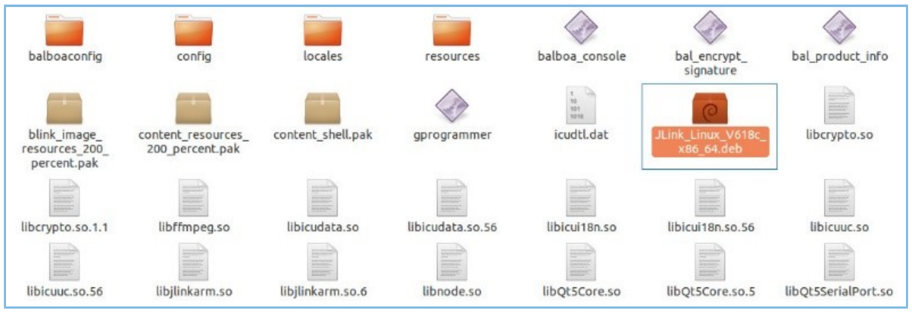


Figure 2-10 Files extracted from *GProgrammer_linux_x64_1.2.6.tar.bz2*

2. If you need to install J-Link, double-click *JLink_Linux_V618c_x86_64.deb* and then start installing GProgrammer.

3. 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 GR551x application projects on Linux is successfully set up. Users can modify, compile, download, and test the example projects of GR551x SDK.

2.8 Build a New Application Project

Users can follow the instructions below to develop new applications with GR551x:

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_hrs 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 the GR5515 SK Board for test and verification.

3 FAQ

This chapter describes possible problems, reasons, and solutions when using GCC examples.

3.1 Why does an Error Occur When Running ble_tools.gcc?

- Description

  An error occurs when I am running ble_tools.gcc after the `make` command is executed.

- Analysis

  The user has no permission for the operation.

- Solution

  Run the `chmod +x ble_tools.gcc` command to grant the user with permission for the operation.