BC35-G&BC28&BC95 R2.0-OpenCPU 快速开发指导

LPWA 模块系列

版本：BC35-G&BC28&BC95 R2.0-OpenCPU_快速开发指导_V1.0

日期：2019-05-13

状态：受控文件

上海移远通信技术股份有限公司始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司
上海市徐汇区虹梅路 1801 号宏业大厦 7 楼　　邮编：200233
电话：+86 21 51086236　　邮箱：info@quectel.com

或联系我司当地办事处，详情请登录：
http://www.quectel.com/cn/support/sales.htm

如需技术支持或反馈我司技术文档中的问题，可随时登陆如下网址：
http://www.quectel.com/cn/support/technical.htm
或发送邮件至：support@quectel.com

前言

上海移远通信技术股份有限公司提供该文档内容用以支持其客户的产品设计。客户须按照文档中提供的规范、参数来设计其产品。由于客户操作不当而造成的人身伤害或财产损失，本公司不承担任何责任。在未声明前，上海移远通信技术股份有限公司有权对该文档进行更新。

版权申明

本文档版权属于上海移远通信技术股份有限公司，任何人未经我司允许而复制转载该文档将承担法律责任。

版权所有 ©上海移远通信技术股份有限公司 2019，保留一切权利。
Copyright © Quectel Wireless Solutions Co., Ltd. 2019.

文档历史

修订记录

版本	日期	作者	变更表述
1.0	2019-05-13	吴丁园/ 王成钧	初始版本

目录

表格索引

图表索引

1 概述

本文档介绍了如何使用 OpenCPU SDK 软件包来快速开始 BC28、BC35-G、BC95 R2.0 模块的 OpenCPU 项目开发。此外，本文还介绍了编译环境搭建、开发工具安装、应用程序的设计和编程的注意事项。

2 OpenCPU 相关文档

表 1：OpenCPU 相关文档

文档名	介绍
Quectel_BC28-OpenCPU_硬件设计手册	文档定义了 BC28-OpenCPU 模块及其与客户应用连接的空中接口和硬件接口。
Quectel_BC35-G-OpenCPU_硬件设计手册	文档定义了 BC35-G-OpenCPU 模块及其与客户应用连接的空中接口和硬件接口。
Quectel_BC95 R2.0-OpenCPU_硬件设计手册	文档定义了 BC95 R2.0-OpenCPU 模块及其与客户应用连接的空中接口和硬件接口。

3 开发准备

在使用 OpenCPU 之前，需要确认是否具备本章节如下所列的软件和硬件组件。

3.1. 主机系统

开发系统须为以下任意一种主机操作系统：

- Windows 7 32 bit 或 64 bit
- Windows 10 32 bit 或 64 bit

3.2. 编译器

- GCC 编译器
- SCons 自动化构建工具
- DOS 命令行

3.3. 编程语言

开发者必须具备基本的 C 语言编程知识，并建议具备一定的多任务操作系统经验。

3.4. 模块硬件

- 移远通信 BC28、BC35-G 和 BC95 R2.0 模块
- 移远通信 BC28-TE-B、BC35-G-TE-B 和 BC95 R2.0-TE-B 开发板
- 其他配件，例如 USB 线等

如需上述硬件资源，可联系移远通信技术支持。

3.5. OpenCPU SDK

- OpenCPU SDK 软件包

您可以联系移远通信技术支持获取 OpenCPU SDK 软件包。

- App 下载工具

您可以联系移远通信技术支持以获取相关工具。

3.6. 开发环境工具包

编译环境及开发工具安装包

您可以联系移远通信获取。

备注

您可以通过邮箱 support@quectel.com 联系移远通信技术支持。

4 OpenCPU SDK 说明

4.1. OpenCPU SDK 软件包

BC35-G、BC28、BC95 R2.0 的 OpenCPU SDK 软件包分别为 *BC35-G_OpenCPU_NB2_SDK*、*BC28_OpenCPU_NB2_SDK*、和 *BC95_R2.0_OpenCPU_NB2_SDK*。解压缩 SDK 软件包以后，即可查看其目录结构。以 *BC35-G_OpenCPU_NB2_SDK_V1.1* 为例，其典型的目录层次结构如下图所示。

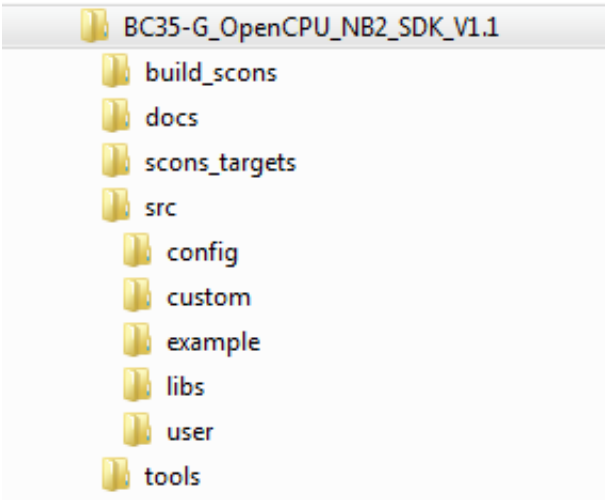

图 1：BC35-G_OpenCPU_NB2_SDK_V1.1 目录层次结构

表 2：BC35-G_OpenCPU_NB2_SDK_V1.1 目录说明

目录名	描述
BC35-G_OpenCPU_NB2_SDK_V1.1	OpenCPU 的根目录
build_scons	输出编译的目录； 可以找到编译生成的 bin 文件。
docs	存放 OpenCPU 项目相关的文件
scons_targets	编译的 target 脚本文件
config	工程的配置文件目录

custom	此目录被设计为系统的配置目录； 在子目录 *custom\private*\中，可根据需要重新配置应用程序，例如创建任务、任务的堆栈大小以及客户可定义自己的版本号等；
example	此目录存放示例代码； 每个示例文件实现了一个独立的功能； 每个示例文件都可以编译成一个可执行的 bin 文件。
include	存放 QI 开头 API 接口的头文件
iot_lib	存放电信物联网开放平台（或华为 OceanConnect 平台）功能的库文件
ont_lib	存放 OneNET 平台功能的库文件
user	此目录被设计为项目的根目录； 用户可以在此目录下创建属于自己的项目结构
tools	存放工程的工具脚本文件目录

备注

由于 BC28、BC35-G 和 BC95 R2.0 的 SDK 软件包目录结构完全相同，本文档后续章节将以 *OpenCPU_NB2_SDK_V1.1* 代替具体模块的 SDK 固件包根目录。

4.2. 创建用户项目

一般情况下，目录 *OpenCPU_NB2_SDK_V1.1\src\user*被设计为项目的根目录。在这个目录中有 *private* 文件夹、*public* 文件夹以及 *SConscript* 文件。其中 *private* 文件夹目录下存放工程配置的源文件，即后缀名为.c 的文件（后称为“c 文件”），客户可以将工程设计的 c 文件存放至该文件夹下。*public* 文件夹中存放工程的公共头文件，客户可以将工程需要的头文件放至 *public* 文件夹中。*SConscript* 文件为编译的脚本，客户不需要更改脚本中的内容。

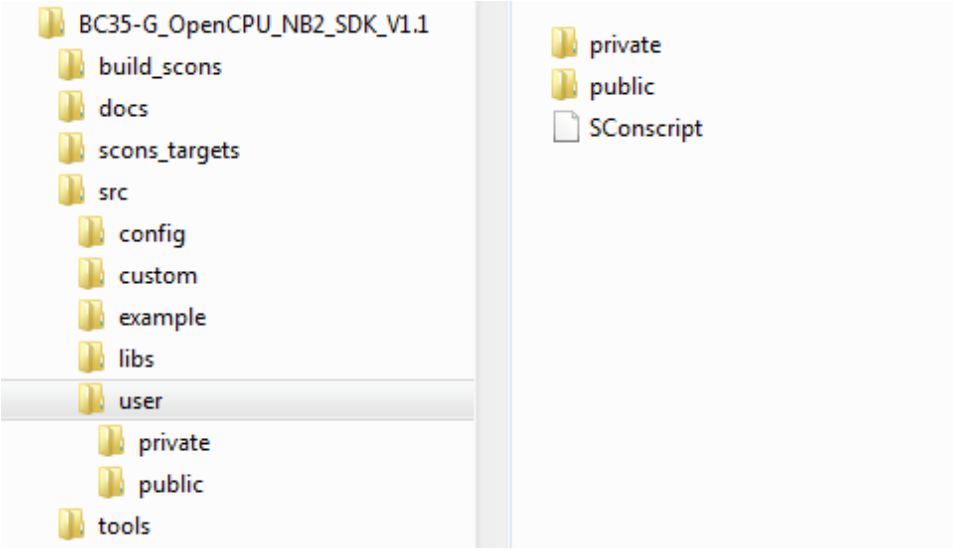

图 2：用户目录

在 *OpenCPU_NB2_SDK_V1.1\src\user*路径下的 *private* 和 *public* 文件夹中，可以添加其他模块文件和子目录，具体新建文件的方法可查阅***章节 9***。

5 编译环境及开发工具安装

OpenCPU 采用的是 Python 脚本写的 SCons 自动化构建工具。从构建角度说，其与 GNU Make 是同一类工具。它是一种改进并跨平台的 GNU Make 替代工具，其集成功能类似于 Autoconf 和 Automake。请使用管理员身份安装编译环境的相关软件。

5.1. 编译环境

5.1.1. GCC 编译器

GCC 编译软件 *gcc-arm-none-eabi-4_9-2015q2-20150609-win32.exe* 可从开发环境工具包中获取。如下图所示，右击安装包文件，以管理员身份运行，开始安装。

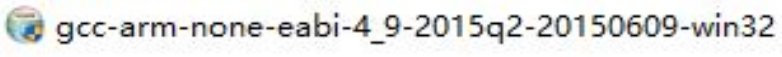

图 3：gcc-arm-none-eabi 安装文件

安装GCC编译器时，请参考下列图示的流程进行安装。

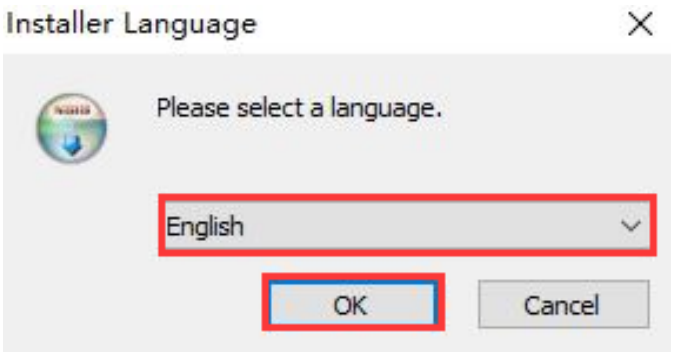

图 4：选择安装语言

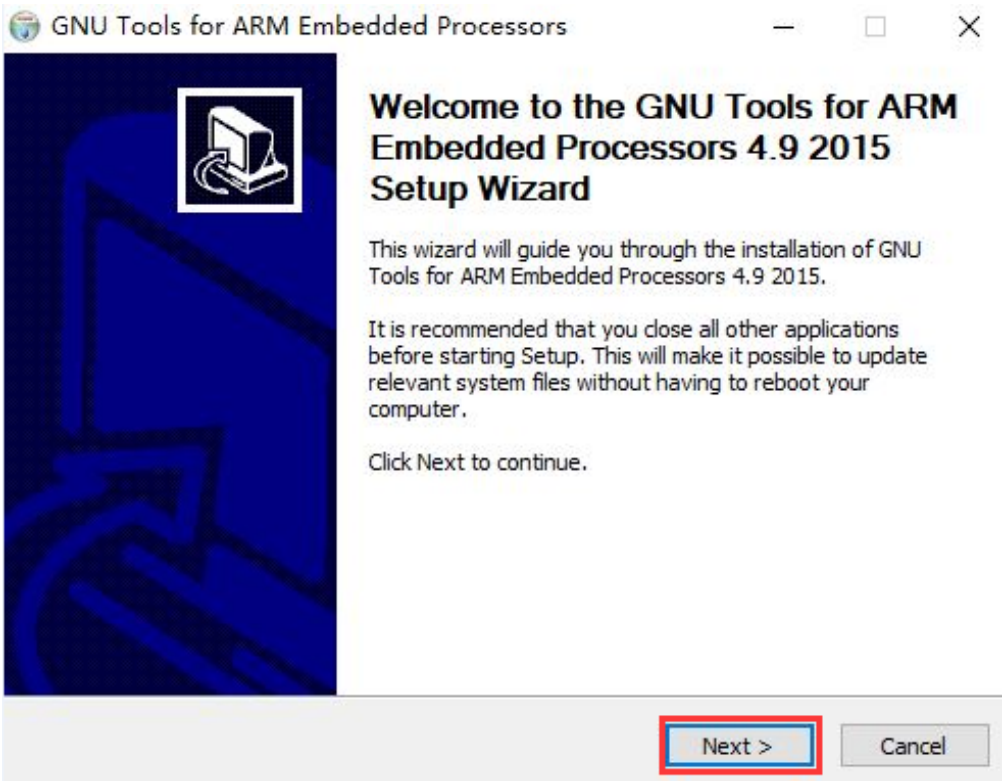

图 5：GCC 安装向导

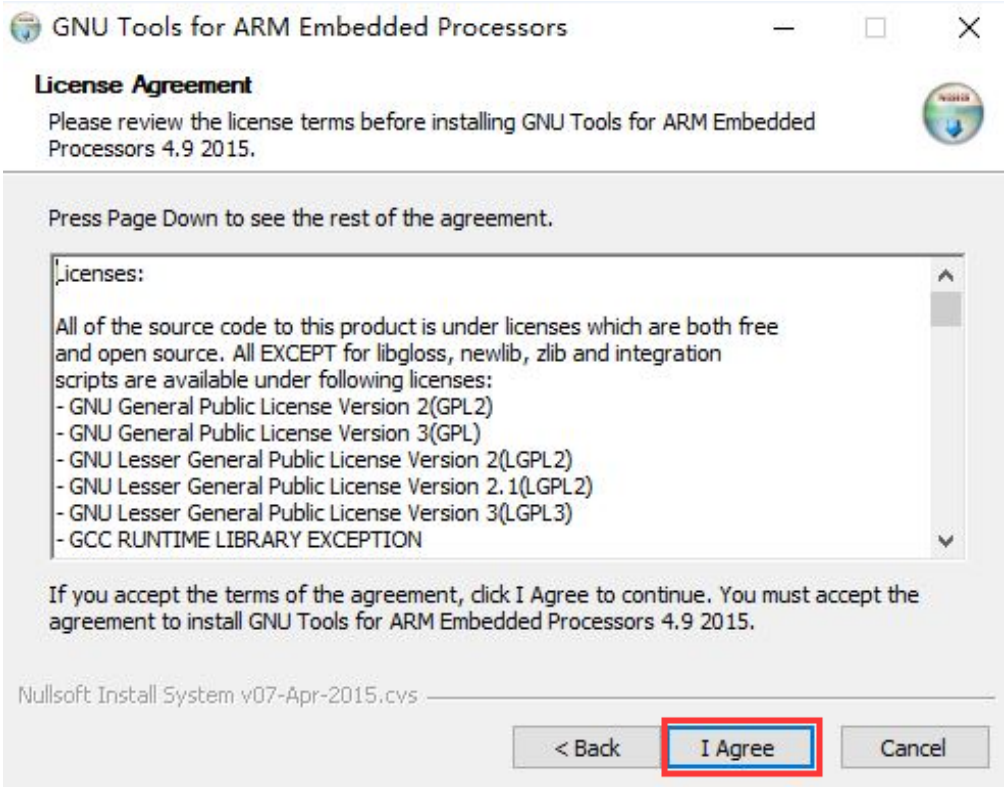

图 6：同意 GCC 安装许可协议

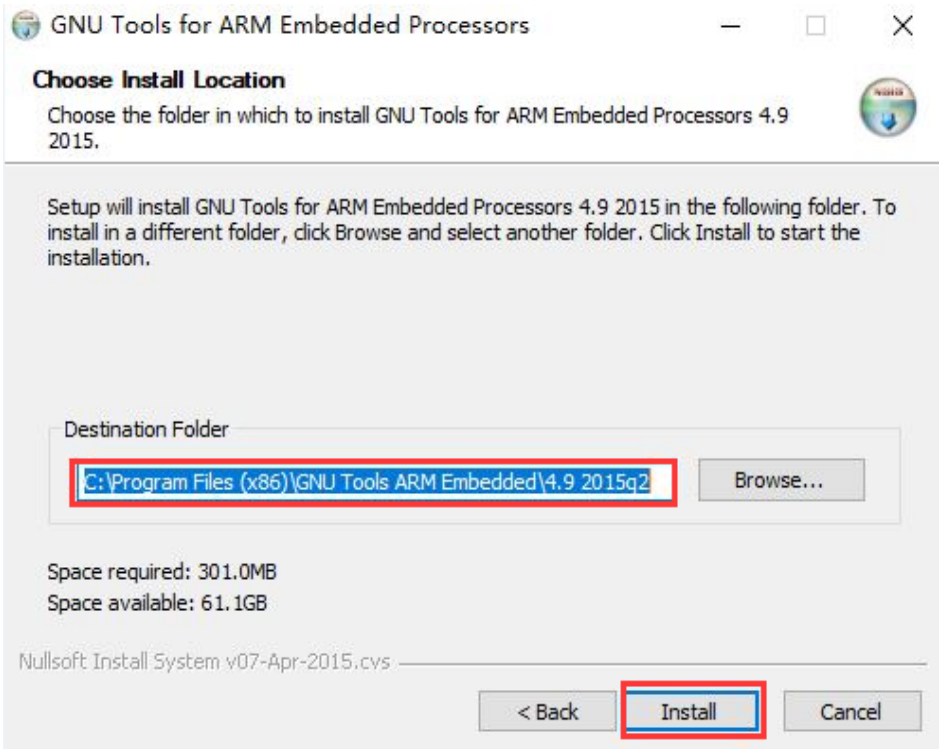

图 7：GCC 默认安装路径

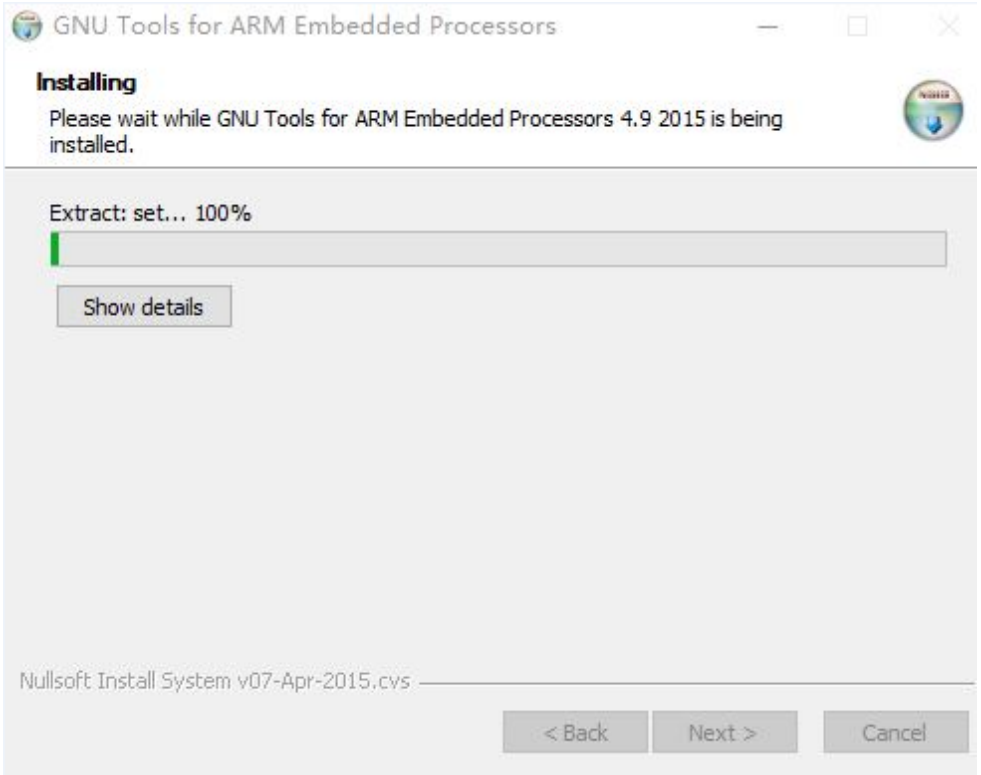

图 8：GCC 正在安装

安装进度条完成后，需要勾选下图中的“**Add path to environment variable**”选项，若忘记勾选，可通过右击“**计算机**”进入“**属性**”→“**高级系统设置**”→“**高级**”→“**环境变量**”，在“**系统变量**”中将 GCC 编译器的安装路径（默认路径为 *C:\Program Files (x86)\GNU Tools ARM Embedded\4.9 2015q2\bin*）手动添加到变量路径中。

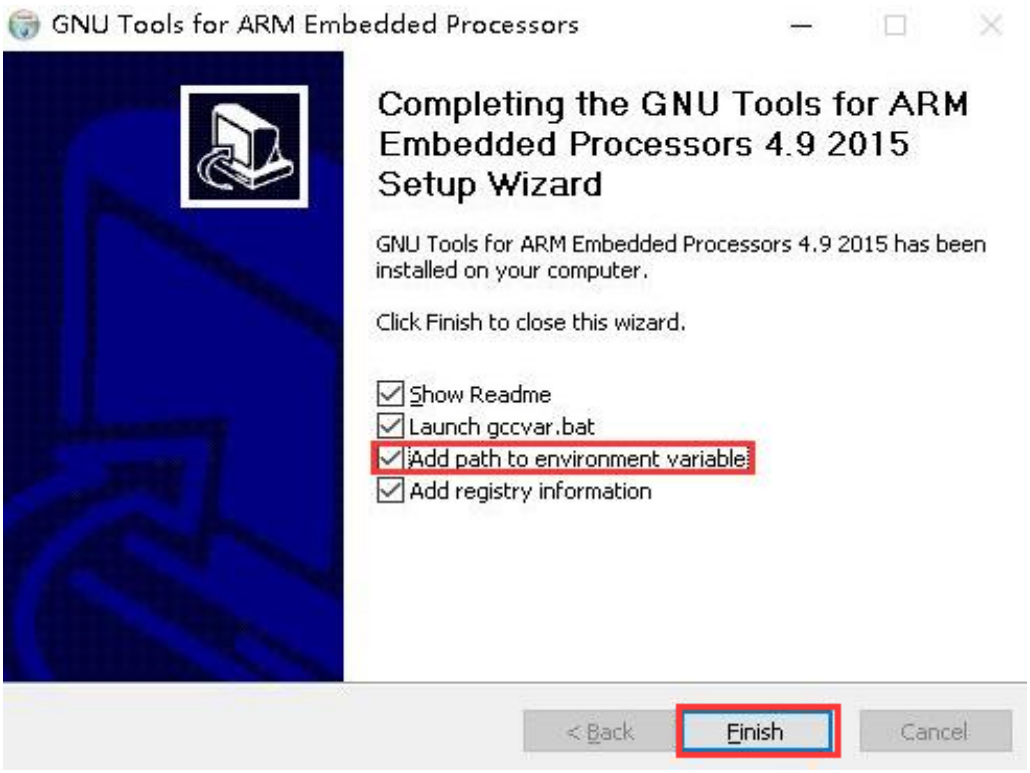

图 9：GCC 添加环境变量

5.1.2. 自动化构建工具 SCons

安装 SCons 工具前，需要先安装 Python 脚本解释工具。SCons 构建工具是采用 Python 脚本语言编写的，推荐使用 Python 2.7.10 版本，其他 Python 版本可能会出现不兼容或者编译报错等问题。

5.1.2.1.安装 Python

Python 脚本安装文件可直接从开发环境工具包中获取，如下图。双击安装文件 *python-2.7.10.msi* 开始安装。

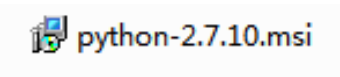

图 10：Python 安装文件

安装 Python 脚本时，请参考下列图示的流程安装。

图 11：选择 Python 安装用户权限

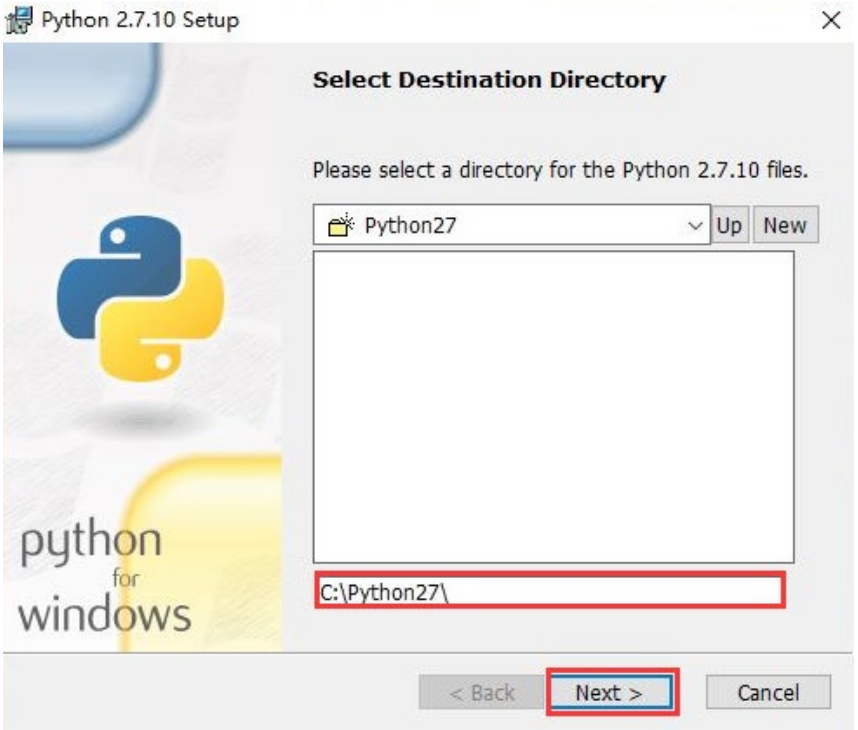

图 12：Python 安装路径

进行到此步骤时，需将 Python 的路径添加到的环境变量中，可按照如下图中的数字提示步骤进行添加。

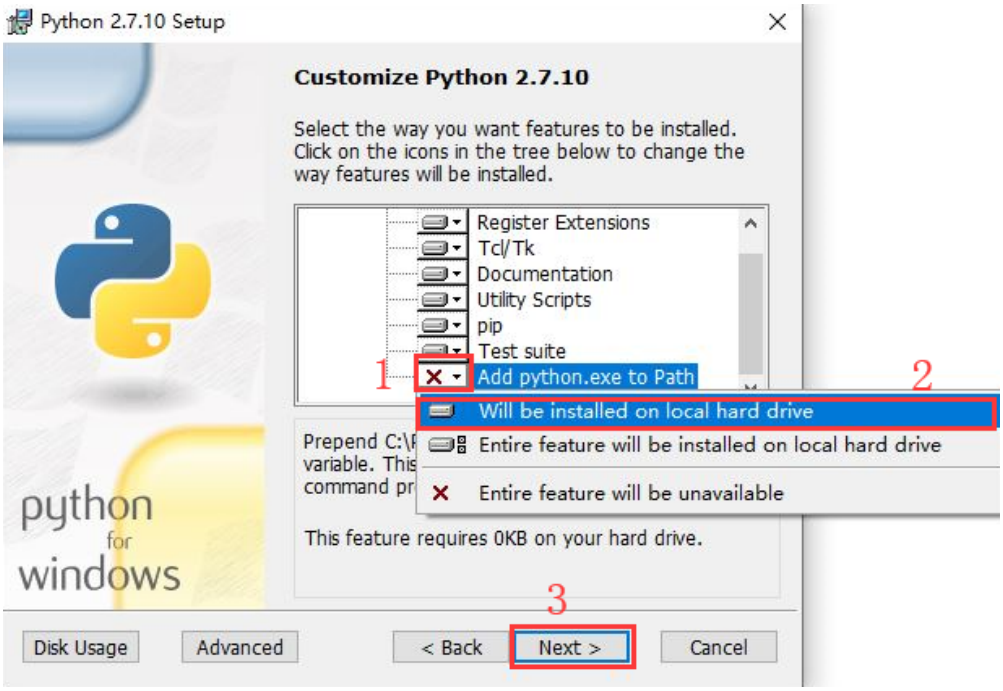

图 13：Python 添加环境变量

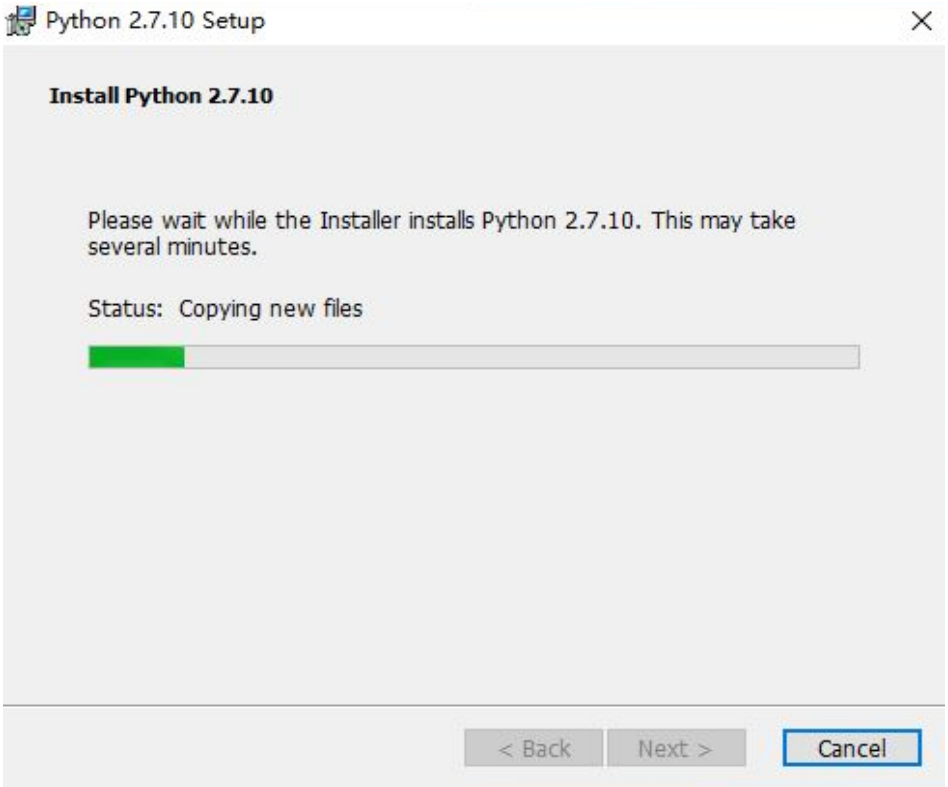

图 14：Python 正在安装

图 15：Python 安装完成

5.1.2.2.安装 pywin32 库

由于 Python 脚本默认不能直接访问 Windows 系统的 API 库，需要安装 pywin32 库才可访问 Windows API 资源。pywin32 库安装文件可直接从开发环境工具包中获取，如下图所示，双击 *pywin32-220.win32-py2.7.exe* 开始安装。

图 16：pywin32 库安装文件

安装 pywin32 库时，请参考下列图示的流程进行安装。

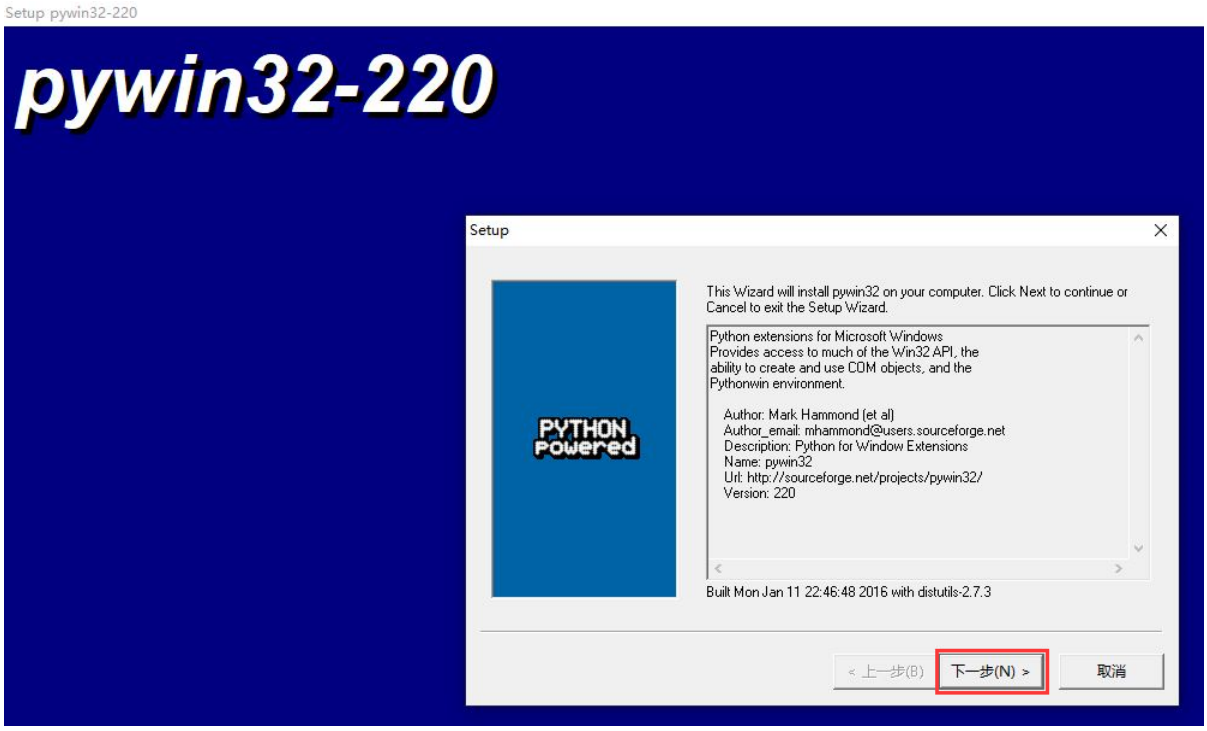

图 17：pywin32 库开始安装

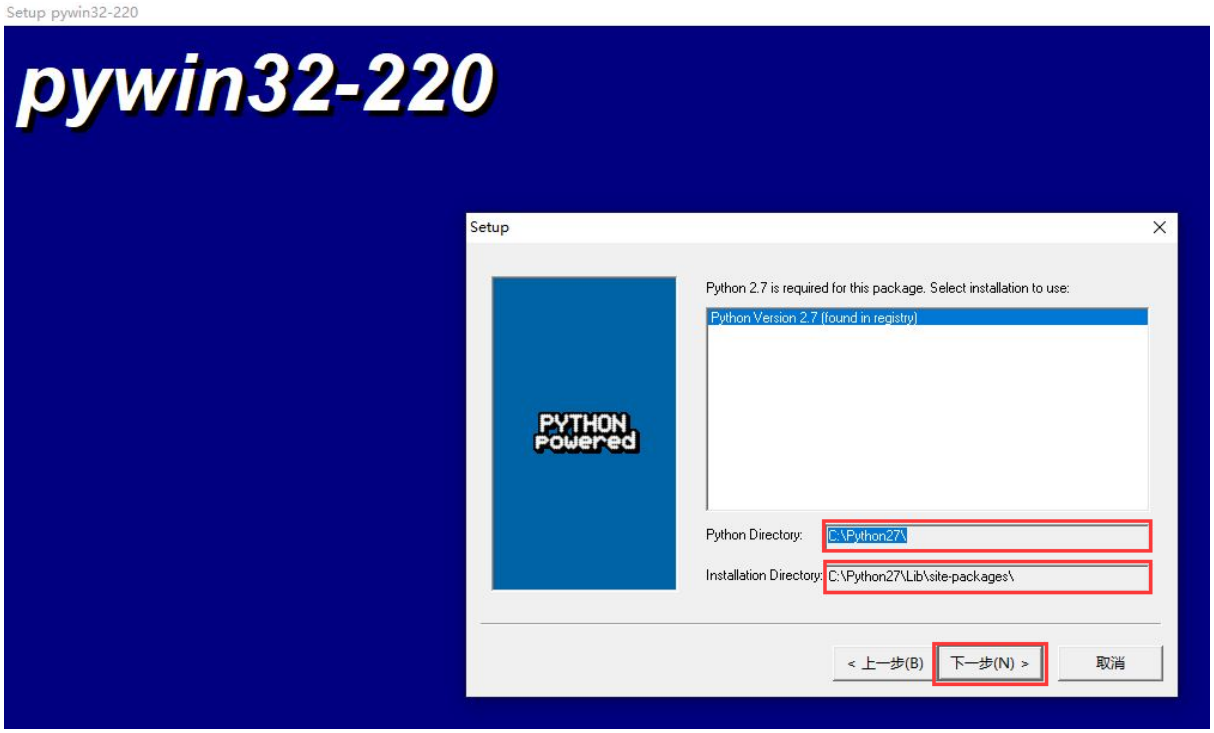

图 18：pywin32 库安装路径

图 19：pywin32 库正在安装

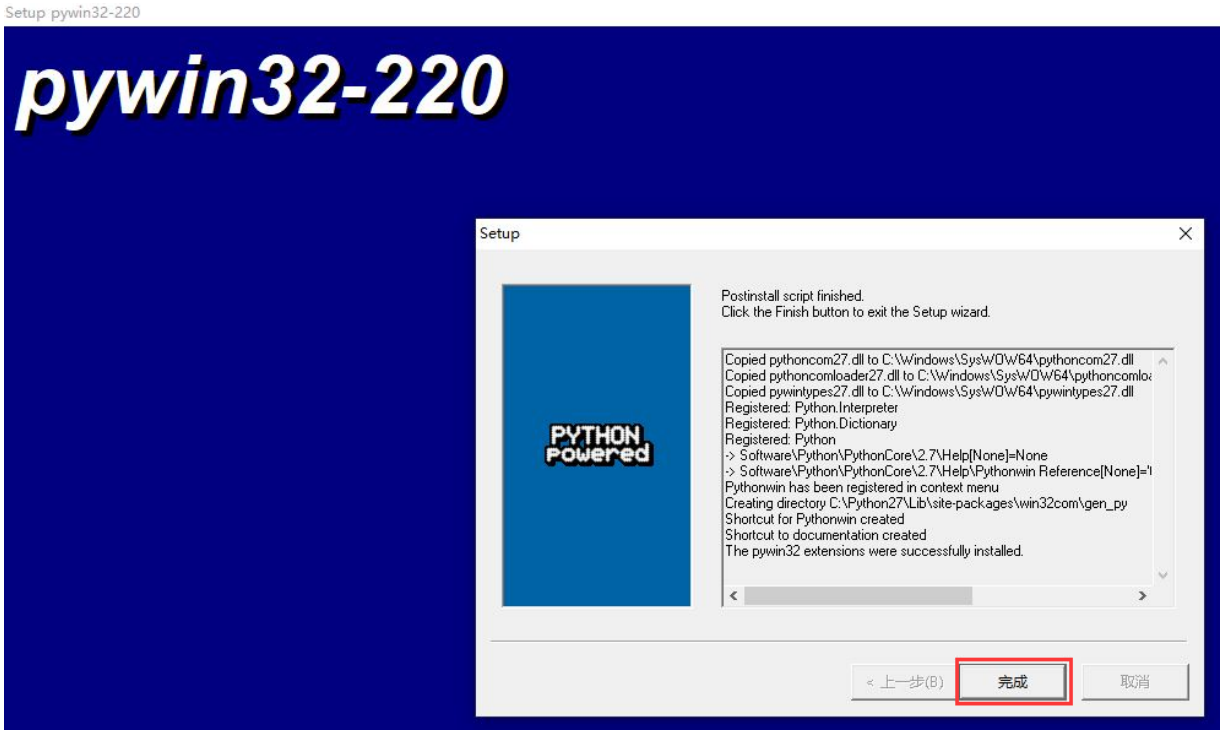

图 20：pywin32 库安装完成

5.1.2.3.安装 SCons 工具

SCons 工具安装文件可直接从开发环境工具包中获取，如下图所示，双击安装 *scons-2.4.0-setup.exe* 开始安装。

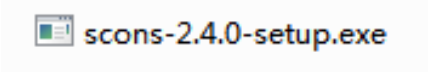

图 21：SCons 安装文件

安装 SCons 工具时，请参考下列图示的流程进行安装。

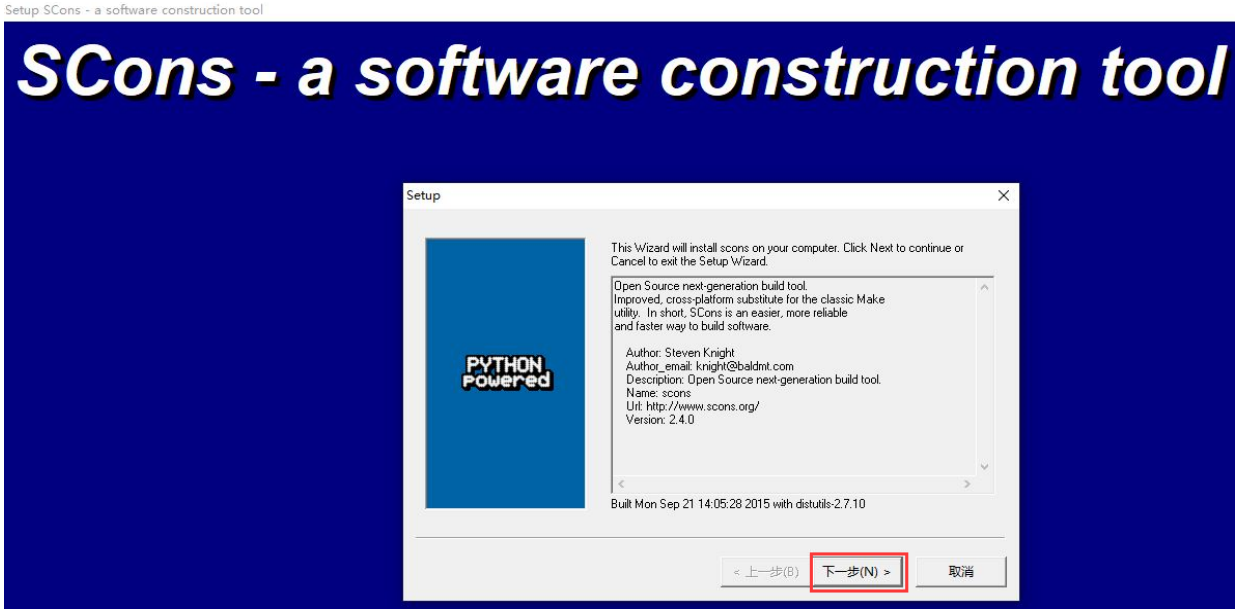

图 22：SCons 开始安装

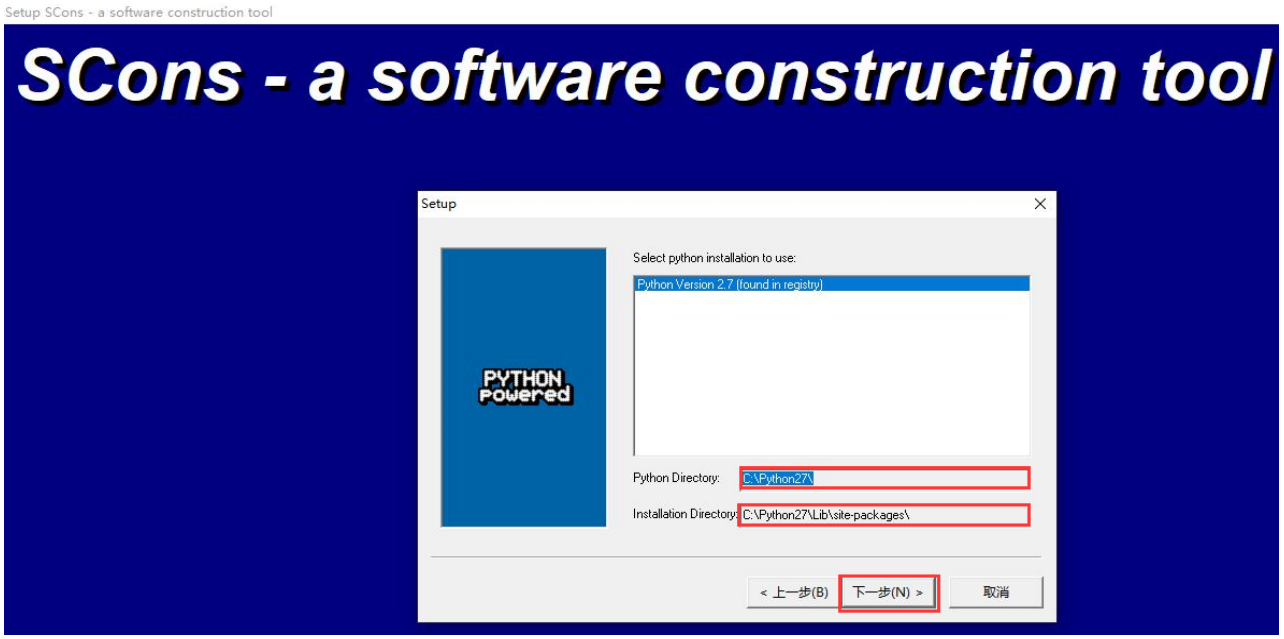

图 23：SCons 安装路径

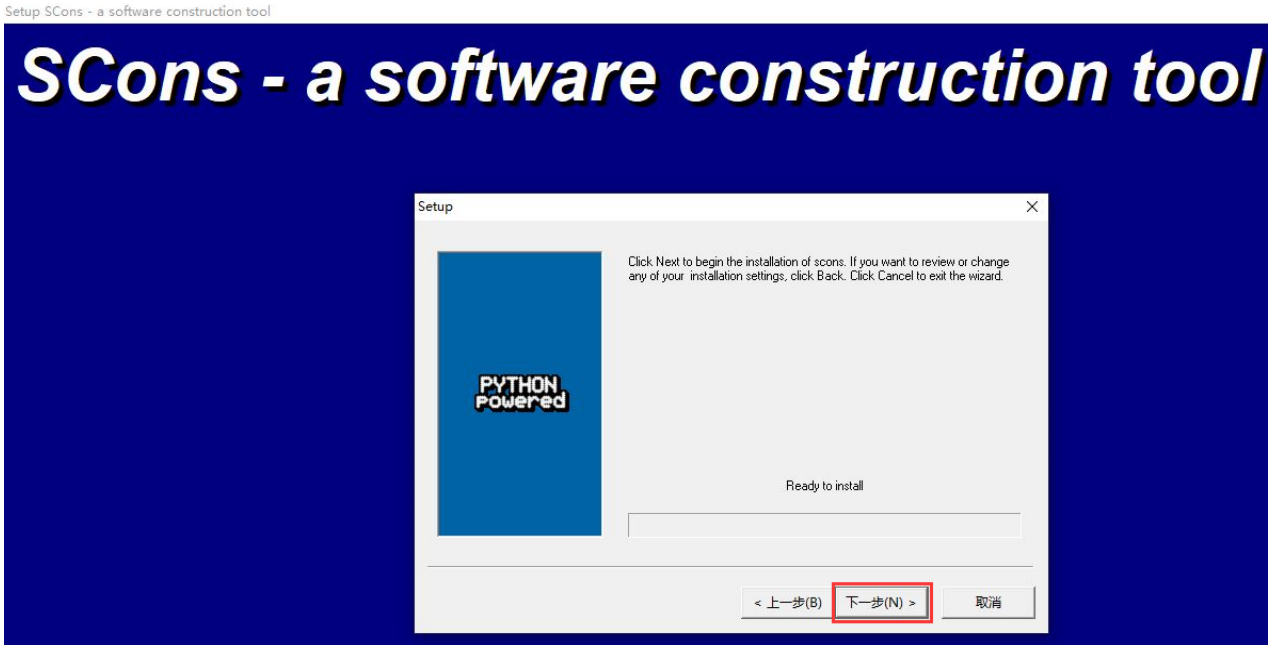

图 24：SCons 正在安装

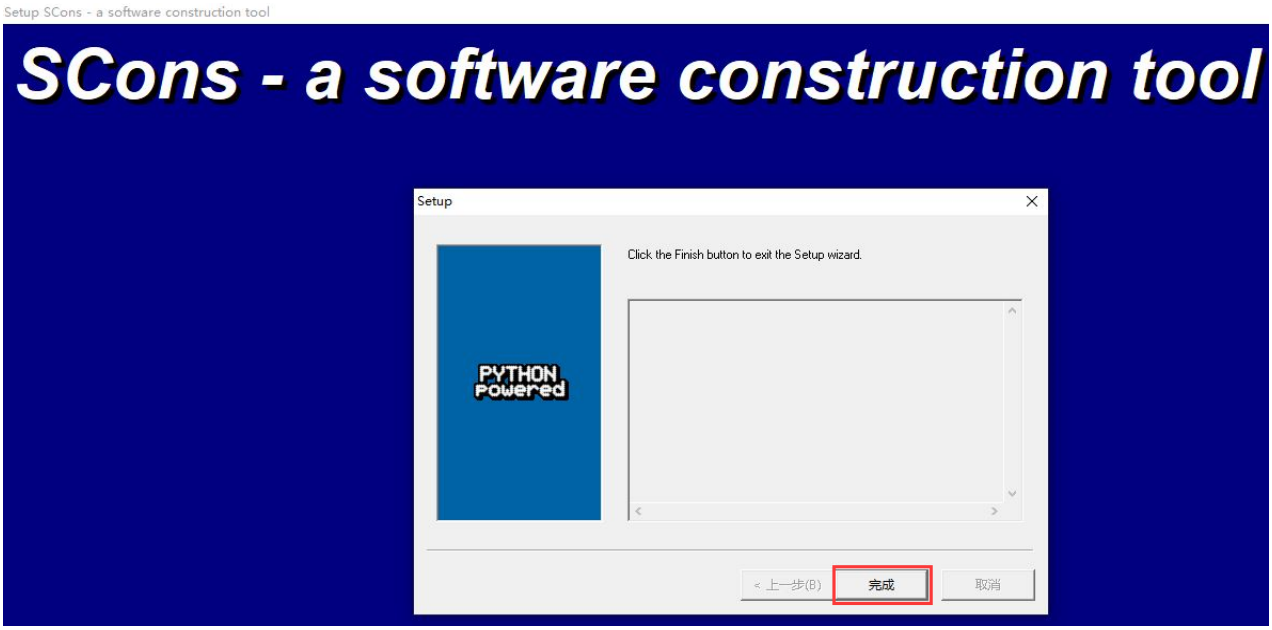

图 25：SCons 安装完成

5.2. 开发工具

5.2.1. 安装更新包工具 UpdatePackage

UpdatePackage 工具可用来更新 fwpkg 固件中的内容。

UpdatePackage 工具的安装文件可直接从开发环境工具包中获取，如下图，双击 *UpdatePackage-3.22.0.14.msi* 安装文件开始安装

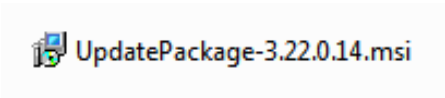

图 26：UpdatePackage 安装文件

安装 UpdatePackage 工具时，请参考下列图示的流程进行安装。

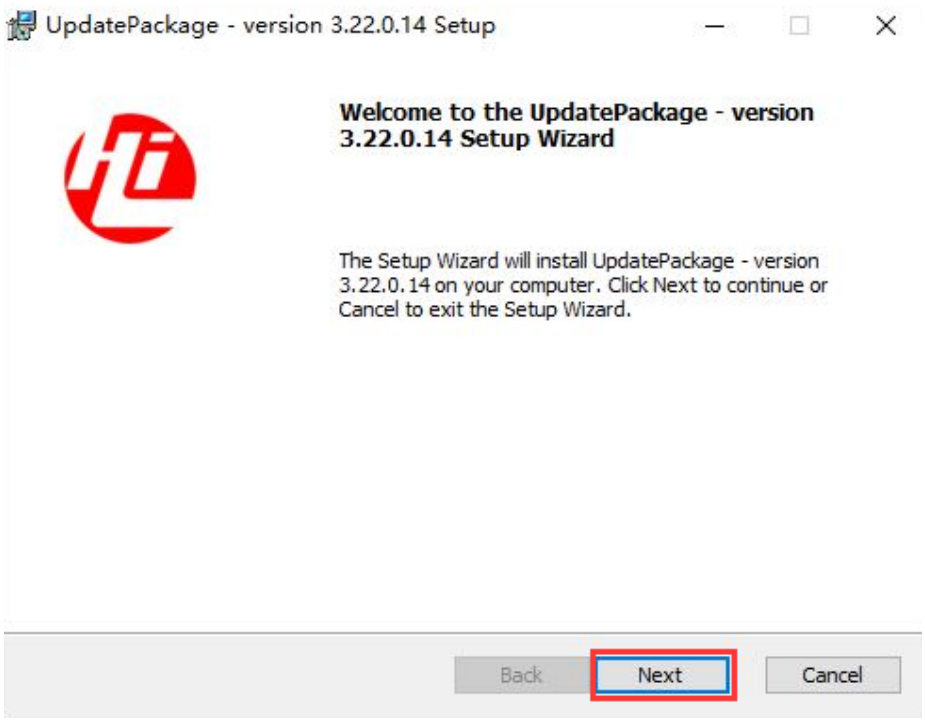

图 27：UpdatePackage 安装向导

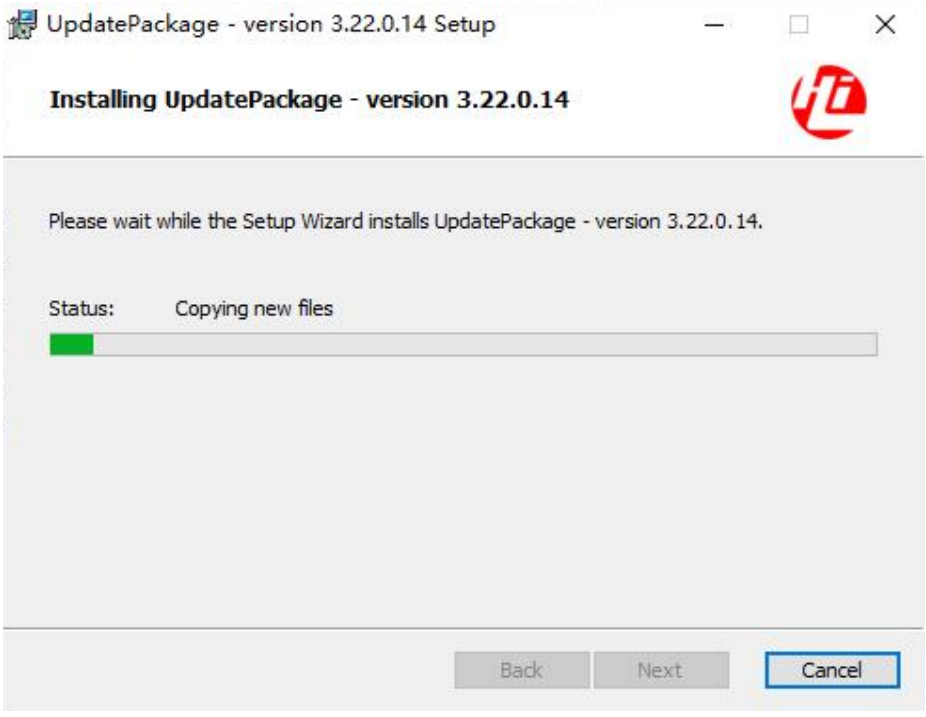

图 28：UpdatePackage 正在安装

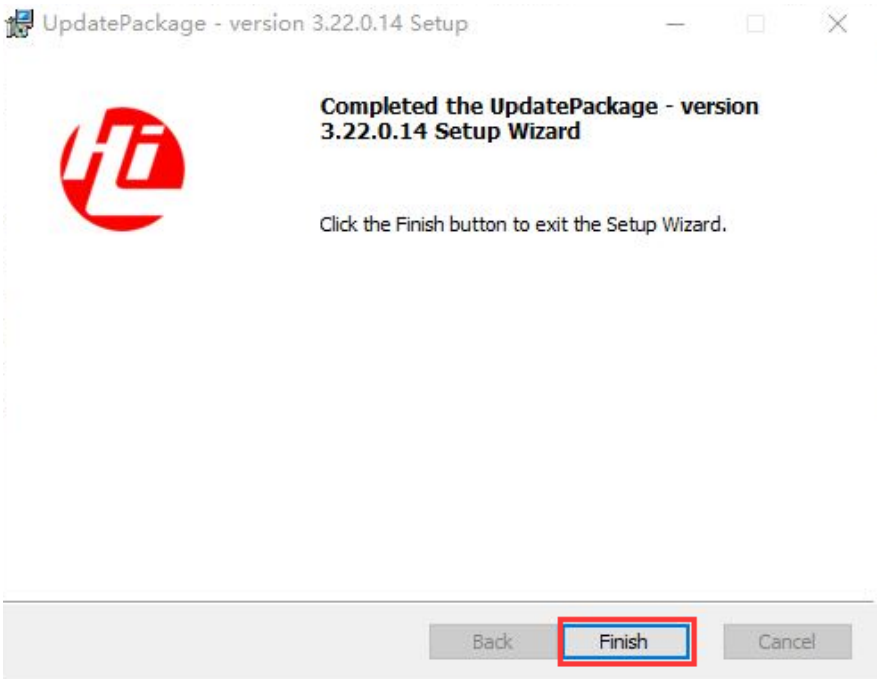

图 29：UpdatePackage 安装完成

5.2.2. 安装下载工具 UEUpdater

若使用的操作系统为 Windows 7，在使用 UEUpdater 工具之前需安装 Microsoft .NET Framework（4.7.2 或更高版本），请参考*章节 5.2.2.1*；若使用的操作系统是 Windows 10，则无需安装，请直接跳至*章节 5.2.2.2*。

5.2.2.1.安装 Microsoft .NET Framework

Microsoft .NET Framework 的安装文件可直接从开发环境工具包中获取，如下图，双击安装文件 *microsoft .NET framework 4.7.2.exe* 开始安装。

图 30：Microsoft .NET Framework 安装文件

安装 Microsoft .NET Framework 时，请参考下列图示的流程进行安装。

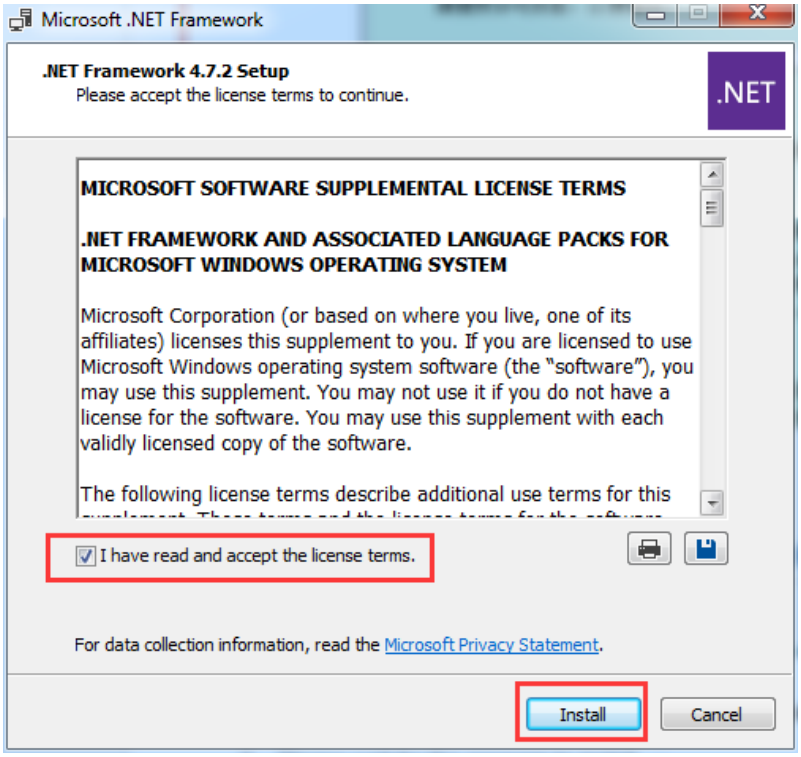

图 31：Microsoft .NET Framework 安装的许可协议

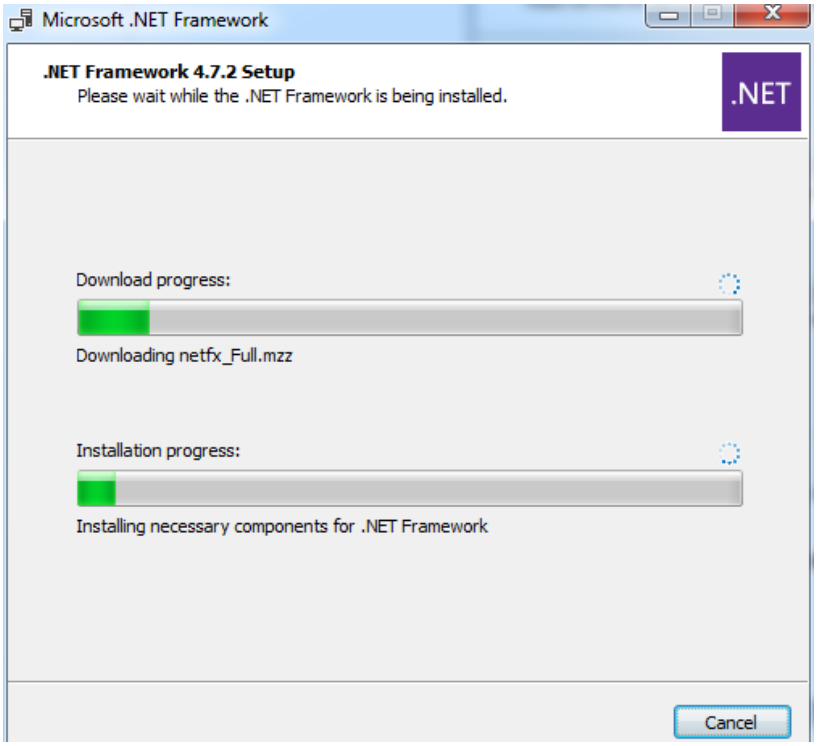

图 32：Microsoft .NET Framework 正在安装

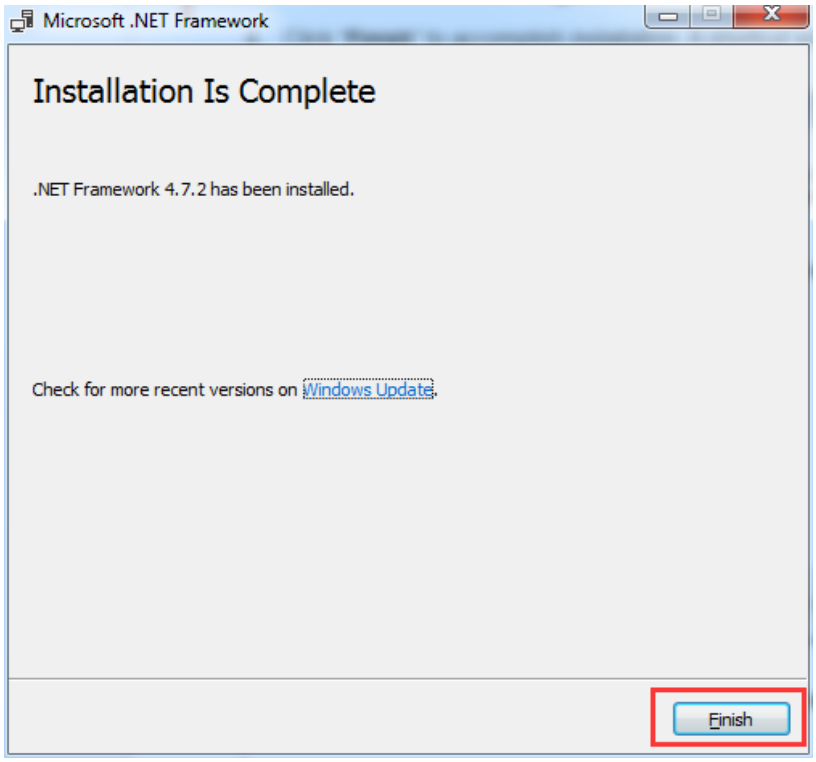

图 33：Microsoft .NET Framework 安装完成

5.2.2.2.安装下载工具 UEUpdater

UEUpdater 工具可将 fwpkg 文件下载至 BC28、BC35-G 和 BC95 R2.0 模块中。

UEUpdater 工具的安装文件可直接从开发环境工具包中获取，如下图所示，双击安装文件 *UEUpdaterUI-3.22.0.14.msi* 开始安装。

图 34：UEUpdater 安装文件

安装 UEUpdater 时，请参考下列图示的流程进行安装。

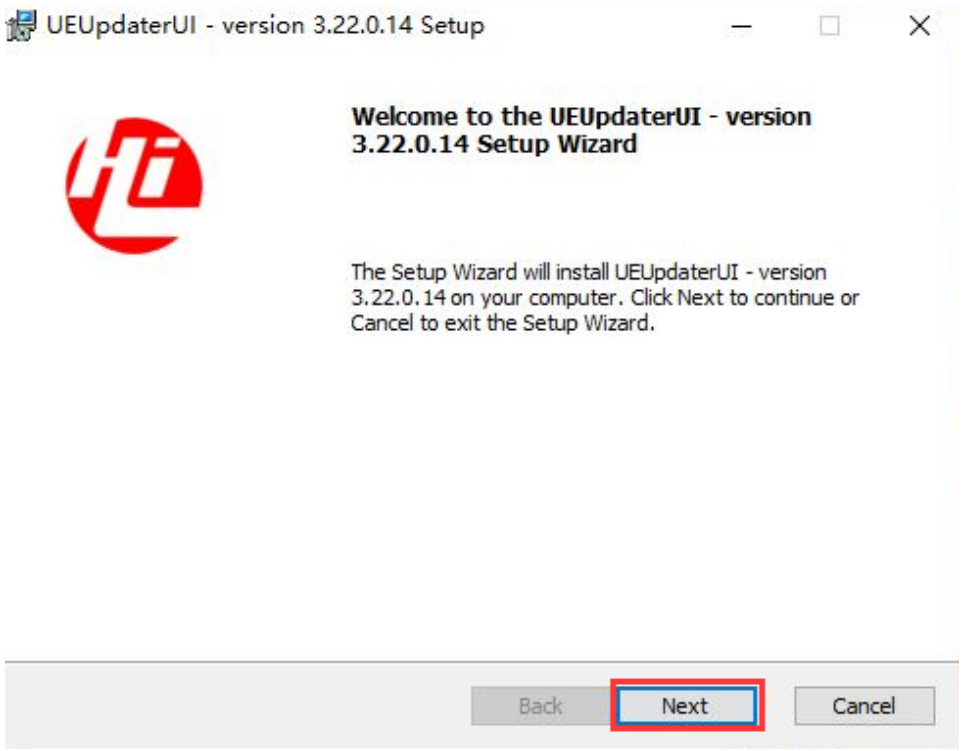

图 35：UEUpdater 开始安装

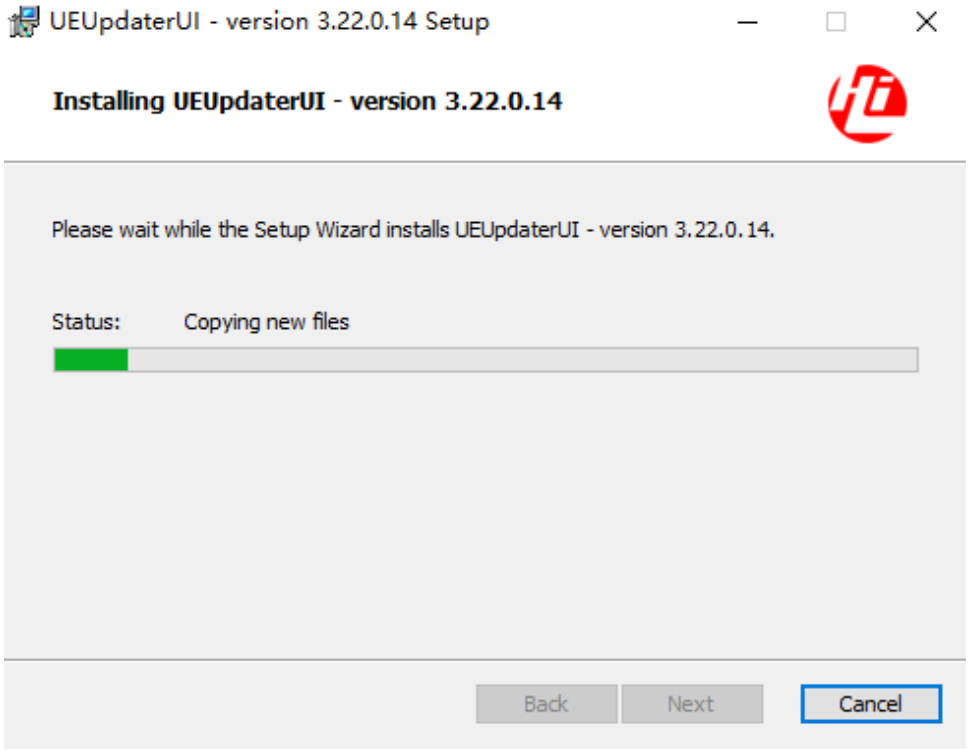

图 36：UEUpdater 正在安装

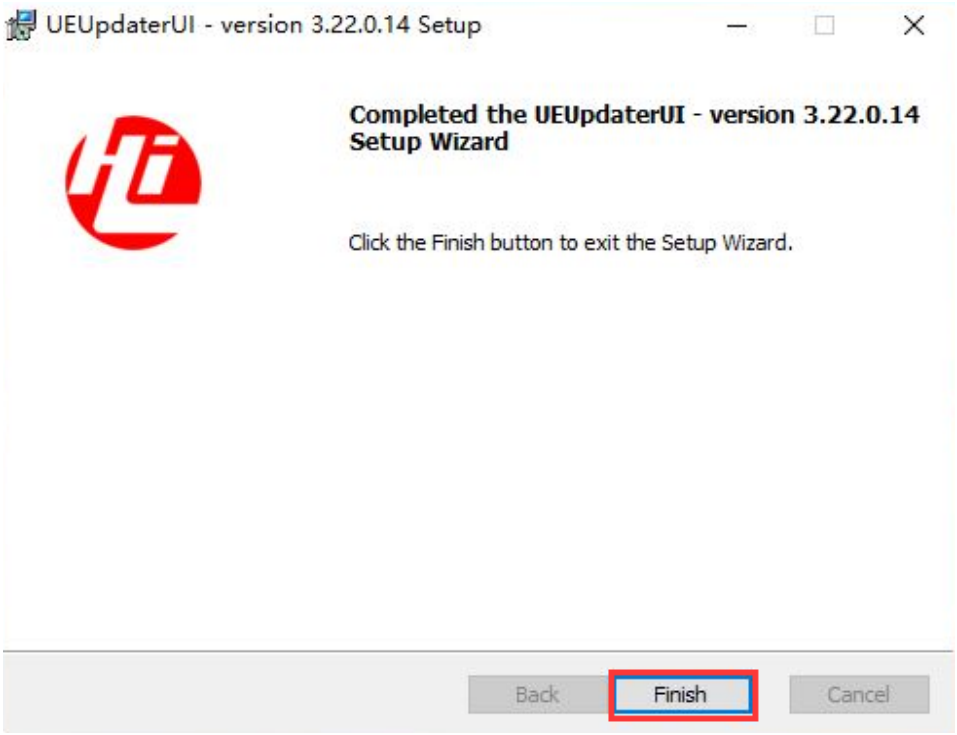

图 37：UEUpdater 安装完成

5.2.3. 安装抓包工具 UEMonitor

UEMonitor 工具被用来抓取 BC28、BC35-G、和 BC95 R2.0 模块的 UEMonitor Log，以便定位可能出现的问题，从而帮助解决问题。

UEMonitor 工具的安装文件可直接从开发环境工具包中获取，如下图，双击 *UEMonitor-3.22.0.14.msi* 开始安装。

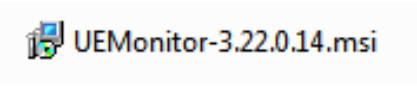

图 38：UEMonitor 安装文件

安装 UEMonitor 时，请参考下列图示的流程安装。

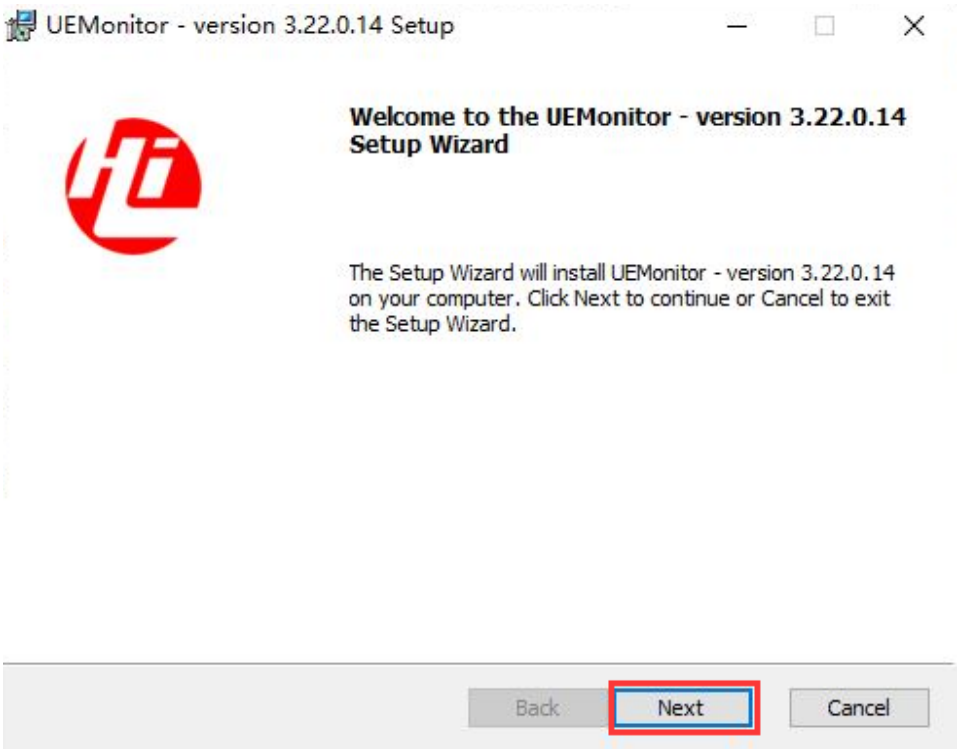

图 39：UEMonitor 开始安装

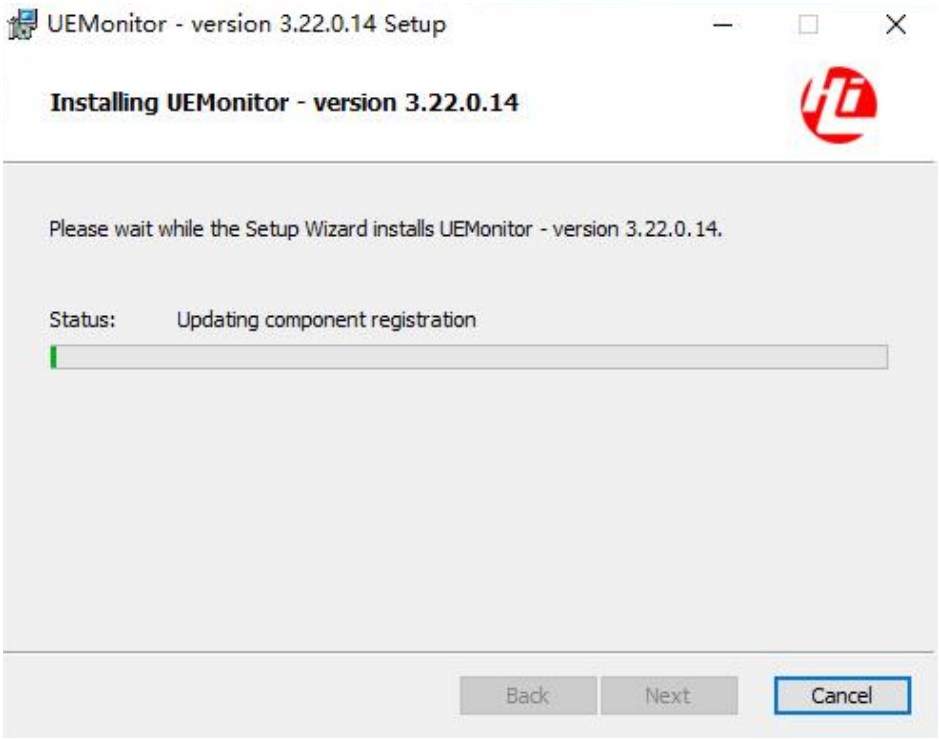

图 40：UEMonitor 正在安装

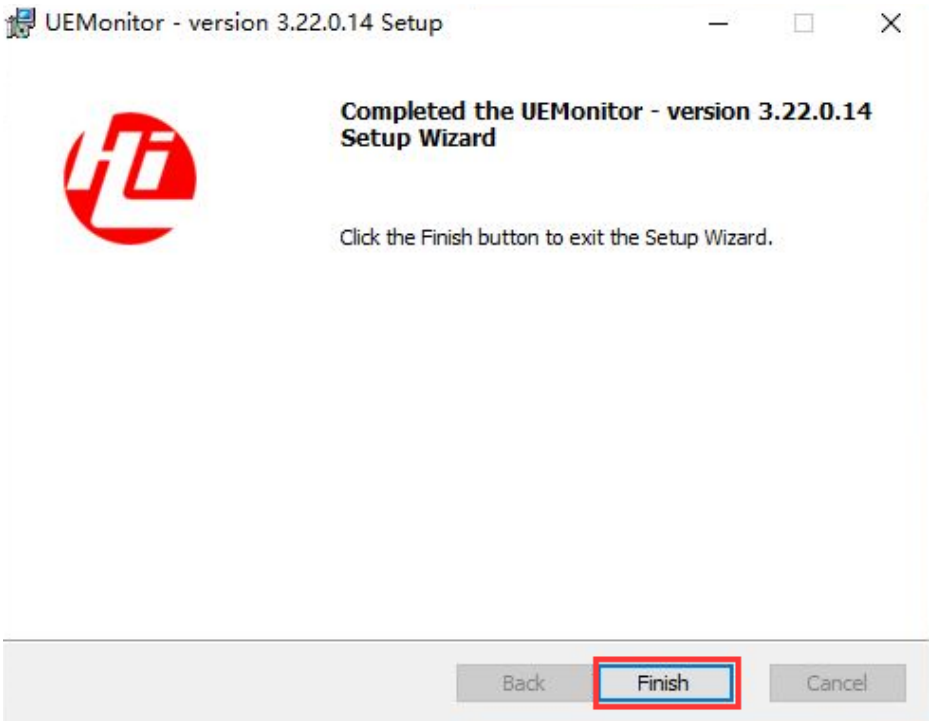

图 41：UEMonitor 安装完成

备注

UEMonitor 的使用方法请参考*文档 [1]*。

6 编译及固件合成

本章将介绍如何在命令行中编译 SDK。

6.1. 编译 bin 文件

6.1.1. 定义库文件

SDK 中提供了 2 个库文件分别对应电信物联网开放平台（或华为 OceanConnect 平台）和移动 OneNET 平台，只有使用了对应平台的库文件才能使用平台的相关功能。可在路径 *OpenCPU_NB2_SDK_V1.1\src* 下的 *SConscript* 脚本文件中定义使用具体的平台库文件，*SConscript* 脚本文件可通过系统自带的记事本程序打开。

如下例 *SConscript* 文件定义使能了电信物联网开放平台（或华为 OceanConnect 平台），并禁用了移动 OneNET 平台的功能 [1)]。

```
#SConscript文件中：
if "_QUECTEL_OPEN_CPU_" in env['CPPDEFINES']:
   env.Append( CPPDEFINES=["_QUECTEL_IOT_OPEN_"])          #Enable iot function
   #env.Append( CPPDEFINES=["_QUECTEL_ONET_OPEN_"])        #Disable onenet function
```

备注

1. [1)] 不能同时通过宏定义使能电信物联网开放平台（或华为 OceanConnect 平台）和移动 OneNET 平台。
2. “#”被用来注释控制移动 OneNET 平台的功能的宏“env.Append(CPPDEFINES=["_QUECTEL_ONET_OPEN_"])”，起到禁用的作用。
3. OpenCPU SDK 必须位于全英文的路径下，否则可能会导致编译失败。
4. 若 *OpenCPU_NB2_SDK_V1.1* 目录下的 *scons.bat* 文件不显示后缀名“.bat”，这是 windows 系统中文件夹选项设置的“**隐藏已知文件类型的扩展名**”导致的，可直接打开 *scons*。

6.1.2. 定义对应模块

BC95 R2.0 与 BC35-G 的引脚基本一致，但 BC28 的引脚与前两者有差异，因此在编译前需要定义对应的模块。SDK 中分别用 opencpu_bc35g 和 opencpu_bc28 控制编译对应的模块型号的 *application.bin* 文件（后文称“bin 文件”）：

- opencpu_bc35g 对应 BC35-G 和 BC95 R2.0

- opencpu_bc28 对应 BC28。

控制编译模块型号的宏的具体定义在 *scons.bat* 文件中，可用记事本打开，命令行如下，*scons.bat* 文件的具体路径为 *OpenCPU_NB2_SDK_V1.1\scons.bat*。

```
#scons.bat 文件中：
C:\Python27\Scripts\scons.bat    -c    target=opencpu_bc35g

C:\Python27\Scripts\scons.bat          target=opencpu_bc35g
```

备注

上方的命令定义了编译 BC35-G 模块的 bin 文件。若要更改成编译 BC28 模块的 bin 文件，只需将命令中所有 opencpu_bc35g 都更改为 opencpu_bc28 即可；若要编译 BC95 R2.0 模块的 bin 文件，仅需与 BC35-G 模块的定义保持一致即可。

6.1.3. 执行编译

在 OpenCPU 中，双击 SDK 目录下的*命令提示符*，再通过执行如下命令即可编译 SDK 代码。

```
scons.bat -c
scons.bat
```

6.1.4. 编译输出

编译期间一些编译处理信息将被输出在命令行中，所有的警告和错误都会直接显示在命令提示符窗口中，用户可以参考命令提示符窗口快速排查代码错误。

编译结束后，若没有任何的报错信息，将会生成 bin 文件，根据 *scons.bat* 文件中定义编译的、针对不同模块的 bin 文件，路径分别为：

- **BC28**：*OpenCPU_NB2_SDK_V1.1\build_scons\BC28*
- **BC35-G/BC95 R2.0**：*OpenCPU_NB2_SDK_V1.1\build_scons\BC35G*

成功编译后命令窗口如下图所示。

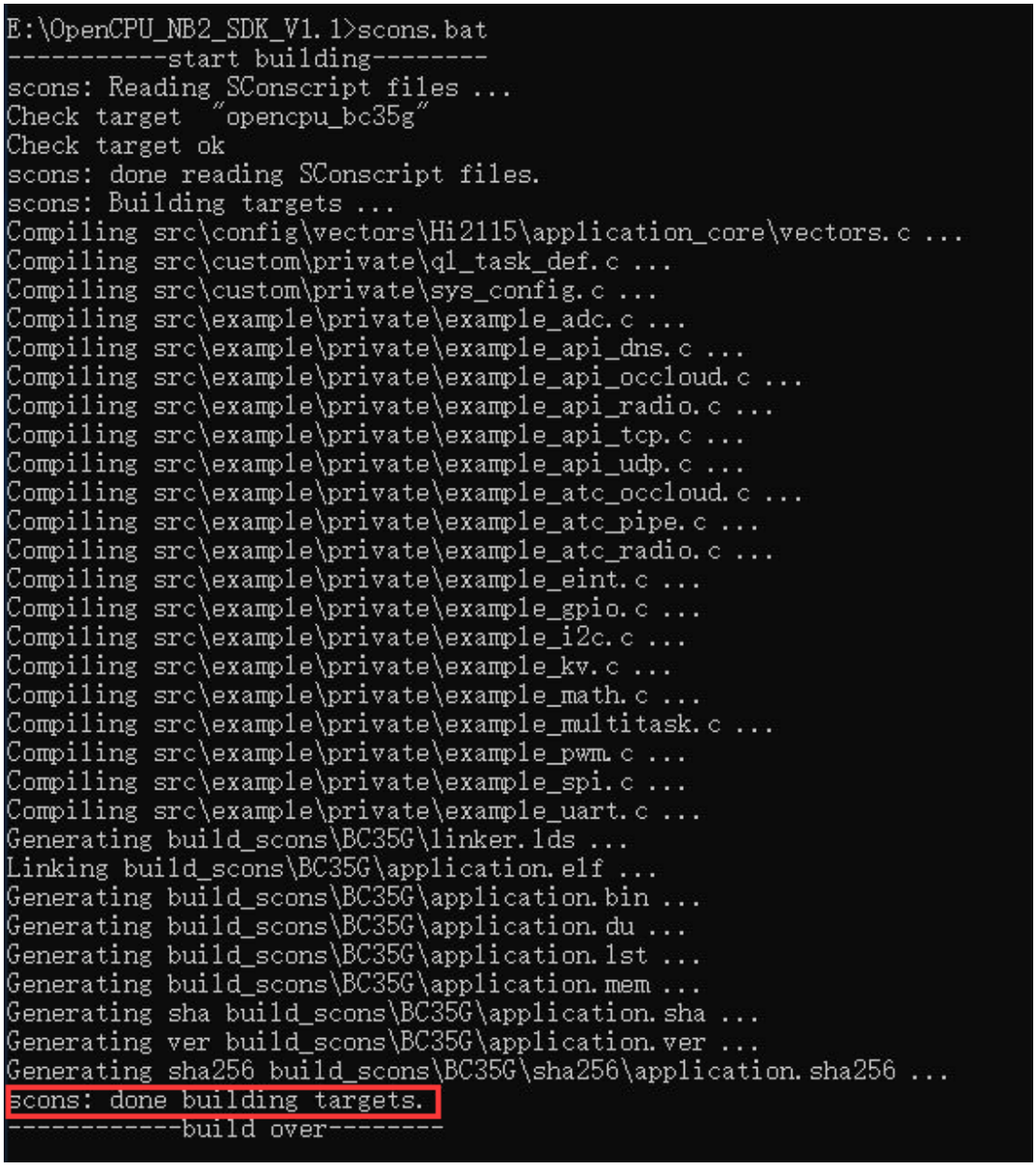

图 42：命令窗口编译结果

6.2. 合成 fwpkg 固件

bin 文件编译完成后，开始合成 fwpkg 固件。下面以合成 BC35-G 模块的 fwpkg 固件为例：

步骤一：在 SDK 主目录下新建一个 fwpkg 文件夹，将标准版本固件（以 BC35GJBR01A05 为例）放入到 fwpkg 文件夹中

步骤二 [1]：双击 SDK 目录下的*命令提示符*，再输入如下合入 fwpkg 命令以生成 OpenCPU fwpkg 固件包 *BC35GJBR01A05_OCN.fwpkg*：

```
"C:\Program Files (x86)\Neul\UpdatePackage\UpdatePackage.exe" updateApplication --in .\fwpkg\BC35GJBR01A05.fwpkg --folder .\build_scons\BC35G --out BC35GJBR01A05_OCN
```

合成后，将 SDK 目录下合成之后的 OpenCPU fwpkg 固件下载至模块中。

备注

1. 请联系移远通信技术支持获取标准版本固件。
2. [1] 步骤二中的合入命令解释如下：
   - 命令引号中为 UpdatePackage 工具路径，为默认安装路径，若安装时更改了 UpdatePackage 工具安装路径，使用此命令会出现合入失败的报错信息，此时需要在命令中将路径更改为对应的安装路径。
   - 系统路径后的“updateApplication”为执行关键字；空格后的“--in ”后跟待合入的基础标准固件路径和固件文件名；空格后的“--folder ”后跟编译生成的 bin 文件路径；再空格隔开的“--out ”后跟自定义的合入后的固件文件名，例如“BC35GJBR01A05_OCN”，固件将在当前路径下生成。
   - 若编译的是 BC28 模块的 bin 文件，则“--folder ”后为“.\build_scons\BC28”
   - 合入命令中不可换行。

6.3. SDK 版本、标准版本说明

SDK 编译生成 bin 文件后，需一个对应基线的标准固件包作为合成 OpenCPU fwpkg 固件的基础固件包，合入后的固件包才是最终 OpenCPU 中使用的应用层 fwpkg 固件包。

BC28、BC35-G 和 BC95 R2.0 模块都有着各自的标准固件包，用户在开发时要注意使用的模块型号，确保获取的标准固件包与模块型号对应，以及基线版本需保持一致。例如，BC35-G 模块使用 OpenCPU 方案开发时，OpenCPU_NB2_SDK_V1.1 版本的 SDK 是 B300SP5 基线，合入的固件就需要使用 BC35-G 的并且基线版本为 B300SP5 的标准固件（可通过命令 **AT+CGMR** 查询基线版本），因此，用作合入的基础固件包是 *BC35GJBR01A05.fwpkg*，合入之后的版本客户可根据实际的功能定义固件名称。

备注

若 SDK 的基线版本号与合入的标准固件包的基线版本号不一致，合入之后的 OpenCPU fwpkg 固件包下载到模块中应用时，会导致模块一直重启。

7 下载

7.1. TE-B

如果您使用的是移远通信的 TE-B 套件，请通过 TE-B 上的主串口（Ch A）下载合入 bin 文件之后的 fwpkg 固件。

7.2. 用户设备

如果模块已焊接到用户设备主板上，请通过模块的主串口下载合入 bin 文件之后的 fwpkg 固件。

7.3. 量产

为提高客户生产效率，移远通信提供了专用夹具和下载工具，可以同时给多个模块下载合入 bin 文件之后的 fwpkg 固件。如有需要，请咨询移远通信技术支持。

8 调试

在 OpenCPU 应用中，通过串口打印跟踪 Log 是主要的调试方法。

在使用 OpenCPU 方案时，BC28、BC35-G 和 BC95 R2.0 模块提供三个串口：UART1（主串口）、UART2（调试串口）和 UART3（BC28 定义为串口 3、BC35-G 和 BC95 R2.0 定义为辅助串口）。在程序中，客户可以调用函数 APP_DEBUG 来输出 Log 至 UART1 端口，此函数也可以输出一些的应用调试消息。

如果模块出现异常重启、Dump 或者网络业务异常问题，可以通过上述的 UART2 串口来抓取 UEMonitor Log，具体操作流程请参考*文档 [1]*。

9 快速编程

本章节将以 BC35-G 模块，使用电信物联网开放平台（或华为 OceanConnect 平台）的情况为例，演示如何通过控制 GPIO 的引脚电平来驱动 LED 灯的亮、灭，让用户对 OpenCPU SDK 方案达成初步了解。BC95 R2.0、BC28 和 BC35-G 的代码设计相同。

用户可以新建一个 *main.c* 文件放至路径 *OpenCPU_NB2_SDK_V1.1\src\user\private* 中，从而实现通过控制 GPIO 引脚电平来控制 LED 灯的亮与灭。

因为每一个 example 目录下例程的主函数 main_task 名称与系统任务中定义的是一致的，这会导致编译报错，所以在新建 *main.c* 文件时，需要同时禁用 example 目录下的例程，可通过用“#”注释 *OpenCPU_NB2_SDK_V1.1\src*\路径下 *SConscript* 文件中的例程宏“_QUECTEL_OPEN_EXAMPLE_”来禁用，如下例所示。

```
#SConscript文件中：
if "_QUECTEL_OPEN_CPU_" in env['CPPDEFINES']:
   env.Append( CPPDEFINES=["_QUECTEL_IOT_OPEN_"])              #Enable iot function
   #env.Append( CPPDEFINES=["_QUECTEL_ONET_OPEN_"])            #Disable onenet function
   #env.Append( CPPDEFINES=["_QUECTEL_OPEN_EXAMPLE_"])        #Disable example
```

9.1. GPIO 控制

9.1.1. 确认需求的头文件

要确认需要哪些头文件（.h 文件），用户首先需要了解这个应用程序的需求。例如 GPIO 控制 LED，这个应用程序的基本需求是：通过周期性地更改 GPIO 电平状态来实现 LED 闪烁：

- 首先，用户需要控制一个 GPIO；相关的 API 和变量的定义在 *ql_gpio.h* 中，所以需要 *ql_gpio.h*。
- 其次，“周期性”意味着需要一个计时器；相关定义在 *cmsis_os2.h* 中，由于该头文件的使用率较高，所以 *cmsis_os2.h* 头文件中的定义已经放在 *ql_common.h* 中，而且 *ql_common.h* 还包含了数据相关的头文件定义，因此客户只需包含 *ql_common.h* 文件即可。
- 另外，若 GPIO 引脚位于 VDD_IO_L1 电压域中，需要打开 VDD_IO_L1 电压域才可使用，因此需要用到相关的 *ql_io_bank.h*。但是这里使用的 NETLIGHT GPIO 引脚不位于 VDD_IO_L1 电压域，所以无需进行打开 VDD_IO_L1 电压域的步骤，因此不需要 *ql_io_bank.h* 头文件。查询 GPIO 引脚是否位于 VDD_IO_L1 电压域请参考对应模块的 OpenCPU 硬件设计手册。

- 最后，应用程序中需要处理计时器和主串口的消息，因此 *ql_common.h* 和 *ql_uart.h* 头文件是必须的。并且定义任务函数在头文件 *example.h* 中，因此也是必须的。此外，还需要打印一些日志信息来调试程序，相关的头文件是 *ql_app_debug.h*，也是必须的。API 函数的所有返回值都在 *ql_error.h* 中定义。

因此，用户需要包括的头文件如下：

```
#include "ql_common.h"
#include "ql_uart.h"
#include "ql_app_debug.h"
#include "ql_gpio.h"
#include "example.h"
```

9.1.2. 配置主串口

步骤一：在程序中初始化置 GPIO 引脚。

在使用 APP_DEBUG 接口打印信息到 UART1 时，首先需要确保 UART1 已经被初始化。在 UART1 还未被初始化之前，可以用函数 QDEBUG_TRACE 来打印相关信息，该接口通过 UART2 输出 Log 信息，可以用 UEMonitor 查看打印的信息。UART1 初始化流程如下：

```
// Define uart1_config varible
ql_uart_config uart1_config;

// Initialize UART1
if (ql_uart_init(UART_PORT1) != QL_RET_OK)
{
    QDEBUG_TRACE("uart port1 init error");
}
else
{
    QDEBUG_TRACE("uart port1 init succ");
}

// Open UART1
If (ql_uart_open(UART_PORT1, &uart1_config, uart1_recieve_handle) != QL_RET_OK )
{
    QDEBUG_TRACE("uart port1 open error");
}
else
{
    QDEBUG_TRACE("uart port1 open succ");
}
```

步骤二：在程序中配置 UART1 的回调函数

由于 UART1 仅仅用于输出 Log 信息，所以这里 UART1 中的回调函数对接收到的消息不做任何处理，若以后需要处理 UART1 接收到的消息，可参考 *example* 目录中的例程设计添加相关代码。由于编译的警告程度较高，所以不使用参数需要用 void 强制转换，否则编译会做出错误判断。如下为 UART1 回调函数的程序。

```
// UART1 callback function
static void uart1_recieve_handle(uint8 *buffer, uint32 len)
{
    (void)buffer;
    (void)len;
}
```

9.1.3. 控制 GPIO

在 BC35-G-TE-B 上，NETLIGHT 引脚已经连接到一个 LED 上，如下图中的红色标记，这意味着用户可以控制 NETLIGHT 来实现 LED 闪烁。

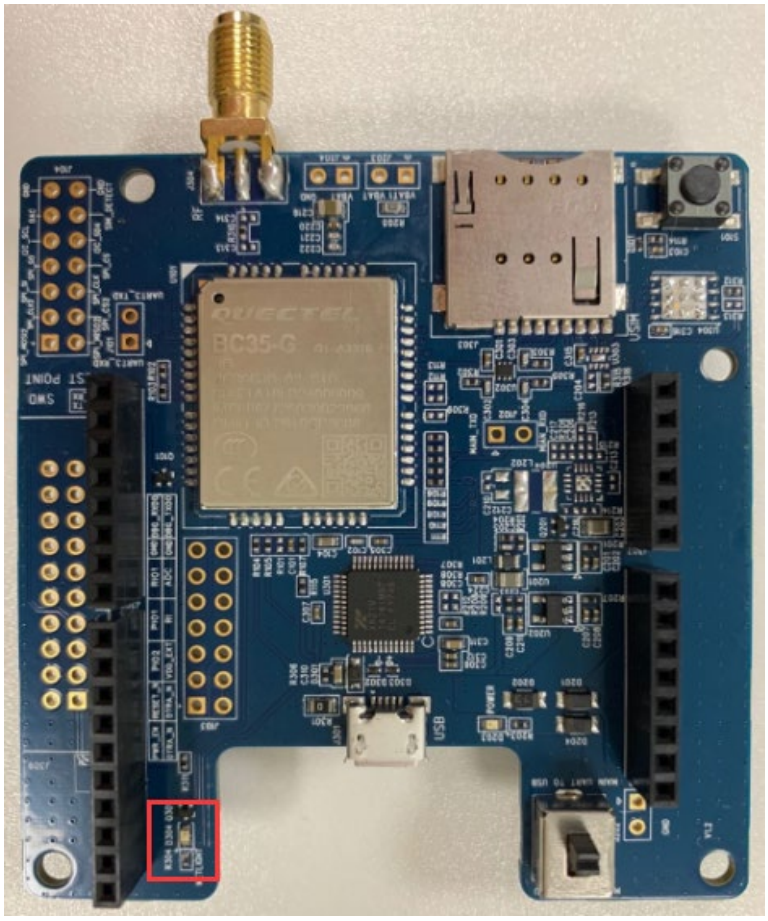

图 43：BC35-G-TE-B 的 LED 指示灯

步骤一：在使用 GPIO 引脚之前，需要确定使用的 GPIO 引脚是否位于 VDD_IO_L1 电压域中。若在 VDD_IO_L1 电压域中，须先打开此电压域才能使用（由于 NETLIGHT GPIO 引脚不位于 VDD_IO_L1 电压域，所以不需要打开 VDD_IO_L1 电压域）。

需要时，可通过调用函数 ql_io_bank_open 来打开 VDD_IO_L1 电压域，具体流程如下：

```
// Open L1 voltage domain
if (ql_io_bank_open (IO_BANK_L1,VDD_IO_LEVEL_3V0) != QL_RET_OK )
{
    APP_DEBUG ("<-- IO_BANK_L1 open error ->\r\n");
}
else
{
    APP_DEBUG ("<-- IO_BANK_L1 open succ->\r\n");
}
```

步骤二：在程序中配置 GPIO 引脚。

```
// Define GPIO pin
static Enum_PinName m_gpioPin = PINNAME_NETLIGHT;
```

步骤三：按照如下方式初始化 GPIO。

- “输入/输出”状态初始化为“输出”
- “初始电平”状态初始化为“低电平”
- “上下拉状态”状态初始化为“下拉”

```
// Initialize GPIO
ret = ql_gpio_init (m_gpioPin, PINDIRECTION_OUT, PINLEVEL_LOW);
if (QL_RET_OK == ret)
{
    APP_DEBUG ("<-- Initialize NETLIGHT GPIO succ -->\r\n");
}
else
{
    APP_DEBUG ("<-- Initialize NETLIGHT GPIO error, cause=%d -->\r\n", ret);
}

ret = ql_gpio_pull_config (m_gpioPin, PIN_PULL_DOWN);
if (QL_RET_OK == ret)
{
    APP_DEBUG ("<-- Set NETLIGHT GPIO pull down succ -->\r\n");
}
```

```
else
{
    APP_DEBUG ("<-- Set NETLIGHT GPIO pull down error, cause=%d -->\r\n", ret);
}
```

9.1.4. 时间和 LED 灯控制

接下来，用户需要启动一个计时器，并定期更改 GPIO 口的电平从而实现 LED 闪烁。

定义一个 500ms 超时的计时器，以控制 LED 亮 500ms、暗 500ms。

步骤一：定义一个计时器和计时器中断处理程序。

```
// Define a timer and the handler
static osTimerId_t   timer_handle;
static uint32   m_nInterval = 500;   //500ms
static void Callback_OnTimer(void);
```

步骤二：注册一个计时器并开始。

```
// Register and start timer
timer_handle = osTimerNew ((osTimerFunc_t)Callback_OnTimer, osTimerPeriodic, NULL, NULL);
osTimerStart (timer_handle, m_nInterval);
```

步骤三：实现计时器的中断处理程序。

```
static void Callback_OnTimer(void)
{
    Enum_PinLevel gpioLvl = ql_gpio_get_level(m_gpioPin);
    if (PINLEVEL_LOW == gpioLvl)
    {
        // Set GPIO to high level, then LED is light
        ql_gpio_set_level(m_gpioPin, PINLEVEL_HIGH);
        APP_DEBUG ("<-- Set GPIO to high level -->\r\n");
    }
    else
    {
        // Set GPIO to low level, then LED is dark
        ql_gpio_set_level(m_gpioPin, PINLEVEL_LOW);
        APP_DEBUG ("<-- Set GPIO to low level -->\r\n");
    }
}
```

至此，所有的编程工作均已完成，完整代码如下：

```
#include "ql_common.h"
#include "ql_string.h"
#include "ql_uart.h"
#include "ql_app_debug.h"
#include "ql_gpio.h"
#include "example.h"
static osTimerId_t    timer_handle;
static uint32    m_nInterval = 500;//500ms

// Define GPIO pin
static Enum_PinName    m_gpioPin = PINNAME_NETLIGHT;
static void uart1_recieve_handle(uint8 *buffer,uint32 len)
{
    (void)buffer;
    (void)len;
}
// Define a timer and the handler
static void Callback_OnTimer(void)
{
    Enum_PinLevel gpioLvl = ql_gpio_get_level(m_gpioPin);
    if (PINLEVEL_LOW == gpioLvl)
    {
        // Set GPIO to high level, then LED is light
        ql_gpio_set_level(m_gpioPin, PINLEVEL_HIGH);
        APP_DEBUG ("<-- Set NETLIGHT GPIO to high level -->\r\n");
    }
    else
    {
        // Set GPIO to low level, then LED is dark
        ql_gpio_set_level(m_gpioPin, PINLEVEL_LOW);
        APP_DEBUG ("<-- Set NETLIGHT GPIO to low level -->\r\n");
    }
}

/*****************************************************************************
* Main Task
*****************************************************************************/
void main_task( void *unused )
{
    (void) unused;
    // Define uart1_config varible
    ql_uart_config uart1_config;
```

```
    QL_RET ret = 0;

    ql_wait_for_at_init();    //wait for modem ok

    // Initialize UART1
    if (ql_uart_init(UART_PORT1) != QL_RET_OK)
    {
        QDEBUG_TRACE("uart port1 init error");
    }
    else
    {
        QDEBUG_TRACE("uart port1 init succ");
    }
    uart1_config.baudrate=9600;
    uart1_config.data_bits=QL_UART_DATA_BITS_8;
    uart1_config.parity=QL_UART_PARITY_NONE;
    uart1_config.stopbits=QL_UART_STOP_BITS_1;
    // Open UART1
    if (ql_uart_open(UART_PORT1, &uart1_config, uart1_recieve_handle) != QL_RET_OK )
    {
        QDEBUG_TRACE("uart port1 open error");
    }
    else
    {
        QDEBUG_TRACE("uart port1 open succ");
    }

    APP_DEBUG ("\r\n <-- OpenCPU: LED Blinking by NETLIGH -->\r\n");

    // Initialize GPIO
    ret = ql_gpio_init (m_gpioPin, PINDIRECTION_OUT, PINLEVEL_LOW);
    if (QL_RET_OK == ret)
    {
        APP_DEBUG ("\r\n<-- Initialize NETLIGHT GPIO succ -->\r\n");
    }
    else
    {
        APP_DEBUG ("<-- Initialize NETLIGHT GPIO error, cause=%d -->\r\n", ret);
    }
    ret = ql_gpio_pull_config (m_gpioPin, PIN_PULL_DOWN);
    if (QL_RET_OK == ret)
    {
        APP_DEBUG ("<-- Set NETLIGHT GPIO pull down succ -->\r\n");
    }
```

```
    else
    {
        APP_DEBUG ("<-- Set NETLIGHT GPIO pull down error, cause=%d -->\r\n", ret);
    }

    timer_handle = osTimerNew((osTimerFunc_t)Callback_OnTimer, osTimerPeriodic, NULL, NULL);
    osTimerStart(timer_handle, m_nInterval);
    while(true)
    {
        osDelay(10);
    }
}
```

9.1.5. 编译以及下载固件

用户可以将上述完整的代码复制到之前新建的、路径为 *OpenCPU_NB2_SDK_V1.1\ src\user\private* 的 *main.c* 文件中并编译。编译的时候需确保用“*#*”注释禁用 *OpenCPU_NB2_SDK_V1.1\src*路径下 *SConscript* 文件中定义例程宏和移动 OneNET 平台功能，并且使能电信物联网开放平台（或华为 OceanConnect 平台），可参考如下：

```
if "_QUECTEL_OPEN_CPU_" in env['CPPDEFINES']:
    env.Append( CPPDEFINES=["_QUECTEL_IOT_OPEN_"])            #Enable iot function
    #env.Append( CPPDEFINES=["_QUECTEL_ONET_OPEN_"])          #Disable onenet function
    #env.Append( CPPDEFINES=["_QUECTEL_OPEN_EXAMPLE_"])      #Disable example function
```

将生成的 bin 文件，合入到标准固件中，例如 *BC35GJBR01A05.fwpkg*，合入之后的固件为 *BC35GJBR01A05_OCN.fwpkg*，将合成之后的固件下载到 BC35-G-TE-B 运行。

当应用程序运行时，可以看到 BC35-G-TE-B 上的 LED 在 500ms 的时间内闪烁，同时可以看到主串口输出以下调试信息。

```
[2019-04-27_20:34:21]<-- IO_BANK_L1 open succ -->
[2019-04-27_20:34:21]
[2019-04-27_20:34:21] <-- OpenCPU: LED Blinking by NETLIGH -->
[2019-04-27_20:34:21]
[2019-04-27_20:34:21]<-- Initialize NETLIGHT GPIO succ -->
[2019-04-27_20:34:21]<-- Set NETLIGHT GPIO pull down succ -->
[2019-04-27_20:34:22]<-- Set NETLIGHT GPIO to low level -->
[2019-04-27_20:34:22]<-- Set NETLIGHT GPIO to high level -->
[2019-04-27_20:34:23]<-- Set NETLIGHT GPIO to low level -->
[2019-04-27_20:34:23]<-- Set NETLIGHT GPIO to high level -->
[2019-04-27_20:34:24]<-- Set NETLIGHT GPIO to low level -->
[2019-04-27_20:34:24]<-- Set NETLIGHT GPIO to high level -->
[2019-04-27_20:34:25]<-- Set NETLIGHT GPIO to low level -->
```

10 注意事项

10.1. Deep Sleep 模式

由于 BC28、BC35-G 和 BC95 R2.0 模块内部有 3 个核，分别是协议核（P 核）、安全核（S 核）以及应用核（A 核），当模块的三个核都不工作的时候，模块才进入 Deep Sleep 模式 [1]，模块的功耗才会达到最低（具体功耗可参考对应模块的 OpenCPU 硬件设计手册中 PSM 功耗）。

备注

[1] Deep Sleep 模式与 OpenCPU 硬件设计手册的模块工作模式中的 PSM 模式概念不同，PSM 模式指模块与基站完全没有了信令连接，即协议核进入休眠状态，而 Deep Sleep 模式下协议核、安全核及应用核均进入休眠状态。

10.2. 串口

BC28、BC35-G 和 BC95 R2.0 模块提供三个串行端口：

- UART1 为主串口，若 UART1 的波特率设置高于 57600bps，模块将禁止进入 Deep Sleep 模式。
- UART2 为调试串口，用来抓取的 UEMonitor Log，是分析问题的重要途径，UART2 无需客户配置，只需调用函数 QDEBUG_TRACE 就可以将相关信息输出至 UART2。
- UART3 在 BC28 模块定义为串口 3、BC35-G 和 BC95 R2.0 模块定义为辅助串口，UART3 的用法与 UART1 一致，具体可参考 example 目录中的代码设计。

由于 UART1 和 UART2 是分析模块问题的重要途径，若客户也需要使用 UART3 端口，建议客户在设计硬件时将 UART1、UART2 和 UART3 端口预留测试点，以便遇到问题时进行调试，帮助快速定位并解决问题。

10.3. PWM

在 OpenCPU 中，提供两个 GPIO 引脚（NETLIGHT、GPIO3）可以复用为 PWM：

- 当模块需要 PWM 功能时，由于模块进入 Idle 状态会影响 PWM 波形正常输出，所以为了保证 PWM 波形正常输出，需要调用函数 ql_add_stop_clocksveto 来禁止模块进入 Idle 状态，此时 A 核一直处于工作状态，模块的功耗与 Deep Sleep 模式下对比，会明显增高。
- 当模块不需要 PWM 功能时，需要调用 ql_pwm_uninit 函数释放使用的 PWM 引脚并调用 ql_remove_stop_clocksveto 函数重新让模块进入 Idle 状态，否则模块不会进入 Idle 模式，因此也无法进入 Deep Sleep 模式，进而影响模块的功耗。

10.4. Example 例程

在 OpenCPU SDK 中路径 *OpenCPU_NB2_SDK_V1.1\src\example\private* 下，提供了相关配套的例程，每个例程都可以单独编译成 bin 文件，每个例程都通过单独的宏进行控制。

例如，若只需要编译 ATC_PIPE 例程 [1]，编译时，可将路径 *OpenCPU_NB2_SDK_V1.1\src*\下的 *SConscript* 文件用记事本打开，使能电信物联网开放平台（或华为 OceanConnect 平台）和 example 功能，注释禁用移动 OneNET 平台功能 [2]，再定义使能 ATC_PIPE 的宏，并将其他的 example 宏注释禁用掉。若前面有新建的 *main.c* 文件，那么需要禁用 *main.c* 文件内容，可通过在 *main.c* 文件内容中注释或利用相关宏的控制来禁用，可参考如下：

```
if "_QUECTEL_OPEN_CPU_" in env['CPPDEFINES']:
    env.Append( CPPDEFINES=["_QUECTEL_IOT_OPEN_"])          #Enable iot function
    #env.Append( CPPDEFINES=["_QUECTEL_ONET_OPEN_"])        #Disable onenet function
    env.Append( CPPDEFINES=["_QUECTEL_OPEN_EXAMPLE_"])      #Enable example function

if "_QUECTEL_OPEN_EXAMPLE_" in env['CPPDEFINES']:
    #env.Append( CPPDEFINES=["__EXAMPLE_UART__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_GPIO__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_EINT__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_SPI__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_I2C__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_ADC__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_KV__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_PWM__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_MULTITASK__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_MATH__"])
    env.Append( CPPDEFINES=["__EXAMPLE_ATC_PIPE__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_ATC_RADIO__"])
    #env.Append( CPPDEFINES=["__EXAMPLE_ATC_UDP__"])
```

```
#env.Append( CPPDEFINES=["__EXAMPLE_ATC_TCP__"])
#env.Append( CPPDEFINES=["__EXAMPLE_ATC_OCCLOUD__"])
#env.Append( CPPDEFINES=["__EXAMPLE_ATC_ONENET__"])
#env.Append( CPPDEFINES=["__EXAMPLE_API_RADIO__"])
#env.Append( CPPDEFINES=["__EXAMPLE_API_UDP__"])
#env.Append( CPPDEFINES=["__EXAMPLE_API_TCP__"])
#env.Append( CPPDEFINES=["__EXAMPLE_API_OCCLOUD__"])
#env.Append( CPPDEFINES=["__EXAMPLE_API_ONENET__"])
#env.Append( CPPDEFINES=["__EXAMPLE_API_DNS__"])
```

备注

1. [1)] OpenCPU 中 ATC_PIPE 用来实现通过虚拟串口的方式发送 AT 命令。客户可以直接通过 ATC_PIPE 的例程发送标准的 AT 命令
2. [2)] 由于不能同时通过宏定义使能电信物联网开放平台（或华为 OceanConnect 平台）和移动 OneNET 平台，此例中禁用了移动 OneNET 平台。

11 附录 A 参考文档

表 3：参考文档

序号	文档名称	备注
[1]	Quectel_BC35-G&BC28_UEMonitor_User_Guide	文档介绍了如何抓取 UEMonitor Log。
[2]	Quectel_BC35-G&BC28&BC95 R2.0_AT_Commands_Manual	介绍 BC35-G、BC28 以及 BC95 R2.0 模块的 AT 命令的定义和使用。
[3]	Quectel_BC35-G&BC28&BC95 R2.0-OpenCPU_User_Guide	介绍 OpenCPU 平台及项目相关的 API 接口。