# OPL1000

ULTRA-LOW POWER 2.4GHZ WI-FI + BLUETOOTH SMART SOC

# SDK 入门手册

OPULINKS

# 版本纪录

| 日期 | 版本 | 更新内容 |
|---|---|---|
| 2018-05-11 | 0.1 | ● 初版 |
| 2018-05-17 | 0.2 | ● 加入章节 3.3 来介绍如何下载编译完的 M3 bin 档案 |
| 2018-05-31 | 0.3 | ● 更正 typo 错误<br>● 更新章节 3.3 |
| 2018-06-05 | 0.4 | ● 更新章节 3.1, 3.2，由于 hello_world 项目设定已改变 |
| 2018-08-02 | 0.5 | ● 更新章节 3 图片 |
| 2018-08-15 | 0.6 | ● 更新章节 2.1，加入 OTA image 文件介绍<br>● 更新章节 2.2，加入 OTA 描述 |
| 2019-07-10 | 0.7 | ● 将 fw_binary 文件夹改成 fw_pack |
| 2020-02-20 | 1.0 | ● 新增章节 2.2.2，来演示 hello_world<br>● 新增章节 2.2.3，介绍用 GCC 编译工程<br>● 移除章节 3.3 |

# 目录

# 图目录

# 表目录

# 1. 介绍

## 1.1. 文档应用范围

本文档介绍了基于 OPL1000 DEVKIT 上开发 OPL1000 应用程序的流程和方法。通过阅读此文用户可以快速理解 OPL1000 用户应用程序开发的原理和过程。

## 1.2. 缩略语

| Abbr. | Explanation |
|---|---|
| APP | APPlication 应用程序 |
| APS | Application Sub-system 应用子系统，在本文中亦指 M3 MCU |
| AT | Attention 终端命令指令集 |
| DevKit | Development Kit 开发工具板 |
| EVB | Evaluation Board 评估板 |
| FW | FirmWare 固件，处理器上运行的嵌入式软件 |
| ICE | In-Circuit Emulator 在线仿真调试工具 |
| OTA | Over-the-Air Technology 空间下载技术 |
| RX | Receive 接收 |
| SWD | Serial Wire Debug 串行线调试 |
| TX | Transmit 发送 |

## 1.3. 参考文献

[1] Download 工具使用指南 OPL1000-patch-download-tool-user-guide.pdf

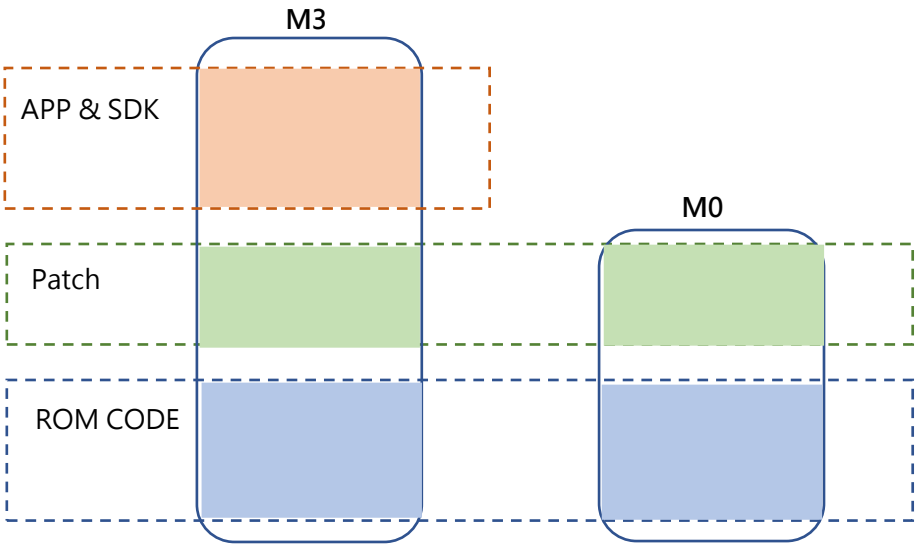
[2] DEVKIT 快速使用指南 OPL1000-DEVKIT-getting-start-guide.pdf

# 2. OPL1000 APP 开发流程

## 2.1. APP 与 ROM Code 和 Patch 关系

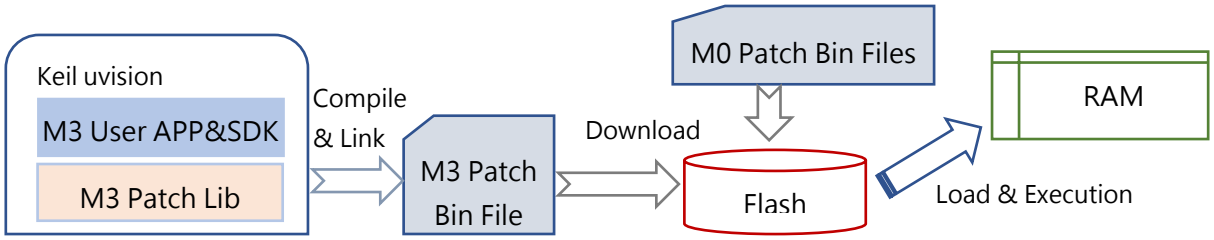
OPL1000 包含两个 MCU，ARM Cortex M3 和 Cortex M0。所谓 OPL1000 APP 开发是指在 OPL1000 的 M3 MCU 上开发用户的应用程序。OPL1000 的原初 M3、M0 固件以 ROM CODE 的方式包含在芯片中。除此之外由于功能扩展和修复 Bug，OPL1000 也提供了 M3 和 M0 的固件补丁。因此用户 App 的开发是基于 ROM Code 和固件补丁基础上完成的。它们之间的关系可以用图 Figure 1 表示。

Figure 1: 用户 APP 和 ROM CODE，Patch 之间的关系




M0 的 Patch 以二进制文件的方式由 Opulinks 给出。M3 Patch 以 lib 库文件的方式提供，用户的 APP 和 Opulinks 提供的 SDK 源码作为一个 Keil C 工程进行编译，因此生成的 M3 bin 文件包含 M3 的固件补丁，SDK 和用户 App 应用程序。最终 M0 的 Patch Bin 文件和 M3 的 Bin 文件合并后下载到片外 Flash 中，OPL1000 芯片上电后，将 Flash 中的 M3/M0 Bin 文件载入到 RAM 中执行。整个过程可以用图 Figure 2 表示。

Figure 2: 用户 APP 和 Patch 编译、载入过程（不带 OTA 功能）




OPL1000 支持两种类型的 Bin 文件。一种称为“Pure Bin”文件，即在 Figure 2 所示过程。用户的应用程序编译出 M3 bin 文件之后和 M0 Bin 文件合并在一起，构成 opl1000.bin。该文件存放在 Flash 0x0000 位置，不包含 OTA loader，也不支持 OTA（空中下载）功能。另一种称为“OTA Image”文件。它是在“Pure Bin”文件的基础上，增加了 OTA Loader，Bin Header 等信息，构成可以支持 OTA 空中下载的固件文件。使用 download tool 生成的文件名为 opl1000_ota.bin。

为了解 OTA Image 文件的结构，我们先需要了解 OPL1000 空中升级固件的工作过程。当 OPL1000 的固件支持空中下载升级时，在 Flash 中维护着两套固件文件信息，如 Table 1 所示，第一个 OTA bin 文件和第二个 OTA Bin 文件分别放在 0x00005000 和 0x0003E000 位置，最大尺寸为 228K Bytes。0x00003000 和 0x00004000 位置分别放置第一个和第二个 OTA bin 的 Header 信息。Header 信息包括 OTA Bin 固件的芯片类型，版本信息，固件校验和，固件尺寸大小，Header 校验和等信息。第一个 OTA bin 文件和第二个 OTA bin 使用乒乓切换方式进行升级。例如当前执行的固件对应于第一个 OTA Bin 文件时，那么通过空中下载得到的固件（以及它的 Header 信息）就放在第二个 OTA Bin 文件位置（固件放在 0x0003E000，header 放在 0x00004000）。如果当前正在执行的固件对应于第二个 OTA Bin 文件时，那么新下载的升级固件就需要放在第一个 OTA Bin 文件位置（固件放在 0x00005000，header 放在 0x00003000）。

Table 1: Flash 中存放 OTA Image 地址映射

| 编号 | 地址 | 大小 | 内容 |
|---|---|---|---|
| 1 | 0x00000000 ~ 0x00003000 | 12K Bytes | OTA loader (Boot Agent) OTA 固件载入程序 |

| | | | |
|---|---|---|---|
| 2 | 0x00003000 ~ 0x00004000 | 4K Bytes | 第一个 OTA Bin 文件的 Header。只使用了 64 字节，其余部分用 0xFF 填充。 |
| 3 | 0x00004000 ~ 0x00005000 | 4K Bytes | 第二个 OTA Bin 文件的 Header。格式同上。 |
| 4 | 0x00005000 ~ 0x0003E000 | 228K Bytes | 第一个 OTA Bin 文件，即前文所说的 opl1000.bin 文件。它和第二个 OTA Bin 文件使用乒乓切换方式升级。 |
| 5 | 0x0003E000 ~ 0x00077000 | 228K Bytes | 第二个 OTA Bin 文件，即前文所说的 opl1000.bin 文件。它和第一个 OTA Bin 文件使用乒乓切换方式升级。 |

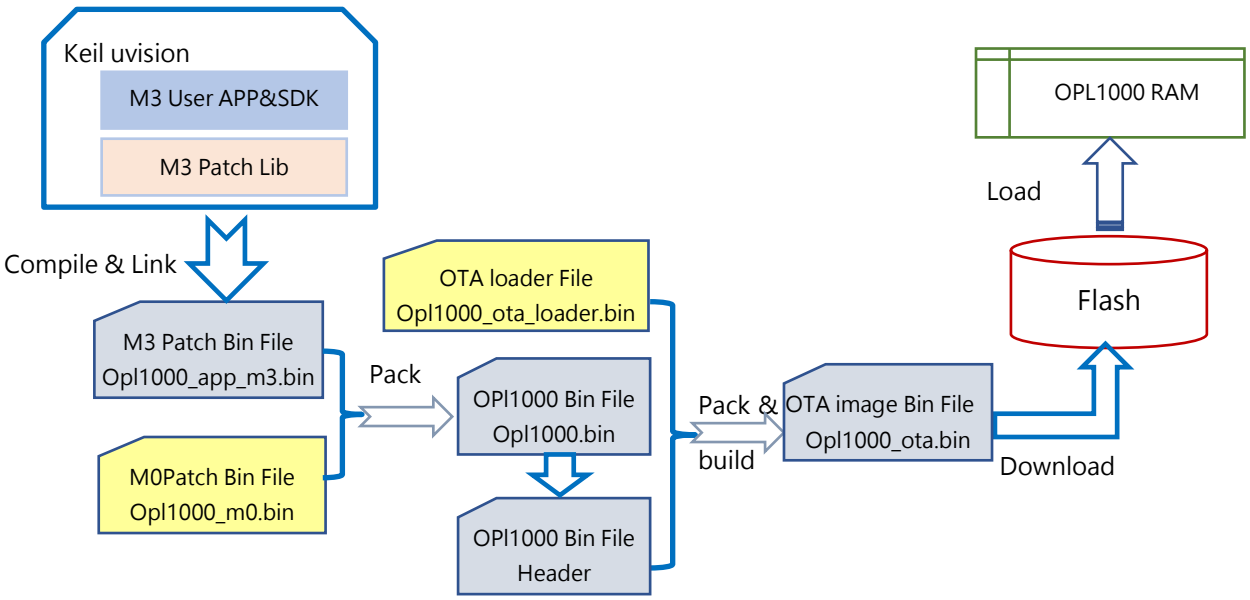
与 Table1 对应 opl1000_ota.bin 文件的构成如 Table 2 所示。它的前三个区域和 Table1 相同。由于 opl1000_ota.bin 只包含一个 Bin 文件，因此它没有第 5 个区段，即第二个 OTA bin 文件数据。并且第四个区段的大小由固件尺寸决定。假定 Flash 中已经存在一个 OTA image，例如对应于 Table1 中的第一个 OTA Bin 文件。那么新下载的 opl1000_ota.bin 文件第 2,4 区段内容将填充 Flash Table1 中的第 3 和 5 区段。

Table 2: OPL1000 OTA Image 构成

| 编号 | 地址 | 大小 | 内容 |
|---|---|---|---|
| 1 | 0x00000000 ~ 0x00003000 | 12K Bytes | OTA loader (Boot Agent) OTA 固件载入程序 |
| 2 | 0x00003000 ~ 0x00004000 | 4K Bytes | OTA Bin 文件的 Header。只使用了 64 字节，其余部分用 0xFF 填充。 |
| 3 | 0x00004000 ~ 0x00005000 | 4K Bytes | 第二个 OTA Bin 文件的 Header，没有被使用，固定用 0xFF 填充。 |
| 4 | 0x00005000 ~ 0x00005000+N | N Bytes | OTA Bin 文件，即前文所说的 opl1000.bin 文件。 |

注意 opl1000 固件支持空中下载有两个前提，一个是具备 opl1000_ota.bin 结构，另一个是 OTA Bin 文件本身支持通过 BLE 或者 WIFI 获取固件功能。对于支持 OTA 功能的用户 APP 编译，下载过程如 Figure 3 所示，它包括两次 Pack 合并功能。一次是将用户 APP bin 文件和 M0 bin 文件合并为 opl1000.bin。第二次是将 opl1000.bin，OTA loader 以及 opl1000.bin 的 Header 信息合并在一起，构成 opl1000_ota.bin 文件。

Figure 3: 用户 APP 和 Patch 编译、载入过程（支持 OTA 功能）




## 2.2. APP 开发流程

### 2.2.1. APP 开发模式

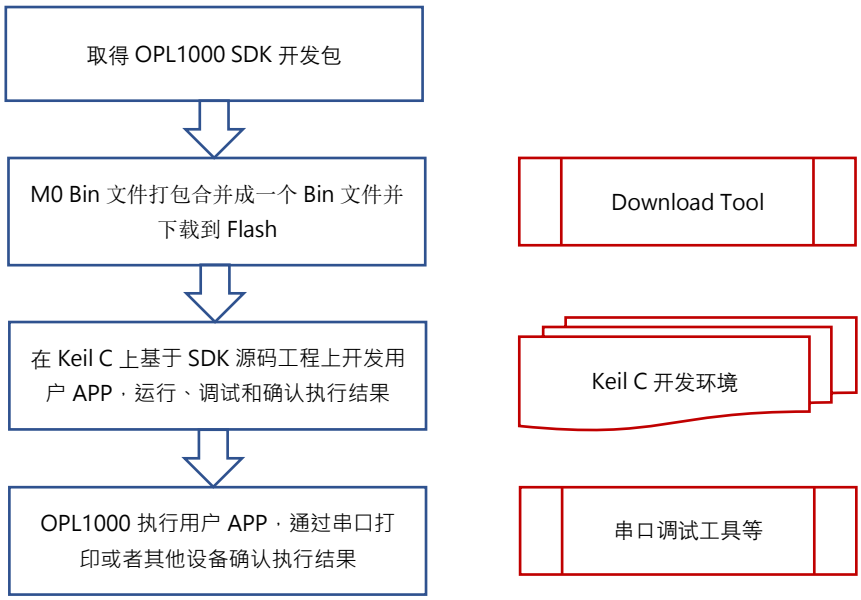
用户可以使用两种模式来开发应用程序。

**模式一：IDE 在线调试开发模式**

开发包括 4 个步骤，整个过程如图 Figure 4 所示。

1. 从原厂拿到 OPL1000 SDK 软件包。
2. 将 SDK 软件包的 FW_Pack 目录下 M0 Bin 文件按照参考文献[1]的 download 工具使用指南说明文档打包成一个 Bin 文件并下载到 Flash 中。注意 M0 bin 文件在 SDK 开发包没有更新的情况下只需要向 SPI flash 下载一次即可。
3. 在 Keil uVision 上开发用户 APP（基于 SDK 示例源码工程）。
4. 在 Keil C 环境下，在线仿真运行、调试，通过串口确认执行结果，注意此时并没有把代码下载到 flash 中，代码仅仅在 RAM 中执行。

Figure 4: IDE 在线调试开发模式

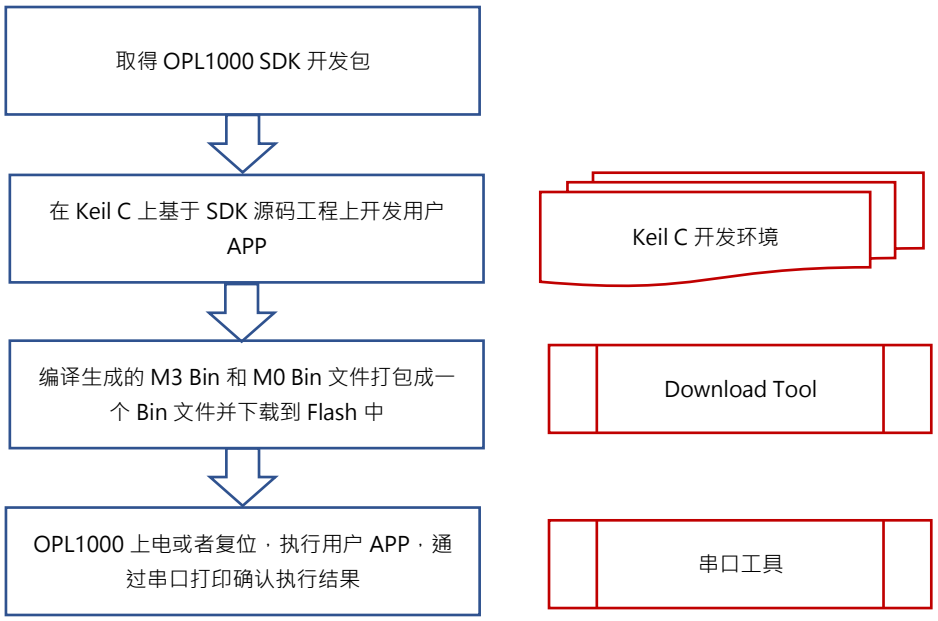



**模式二：串口调试开发模式**

开发包括 5 个步骤，整个过程如图 Figure 5 所示。

1. 从原厂拿到 OPL1000 SDK 软件包。
2. 在 Keil uvision 上开发用户 APP（基于 SDK 源码工程）。
3. 取得编译完成的 M3 Bin 文件
4. 将 SDK 软件包的 FW_Pack 目录下 M0 Bin 文件按照参考文献[1]的 download 工具使用指南说明文档打包成一个 Bin 文件并下载到 Flash 中。
5. OPL1000 上电或者复位，执行用户 APP，通过串口 log 信息确认执行结果。

Figure 5: 串口调试模式



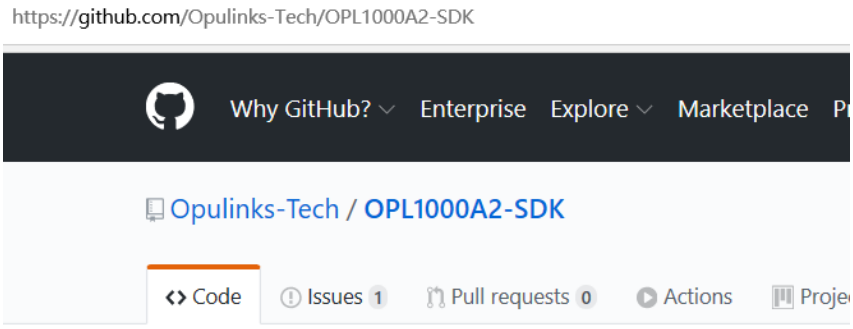

在实际开发过程中用户可以结合自己的开发需求在两种开发模式之间灵活切换。

### 2.2.2. APP 开发过程演示

本章节以现有的工程文件 hello_world 为例，演示 APP 开发的基本过程如下：

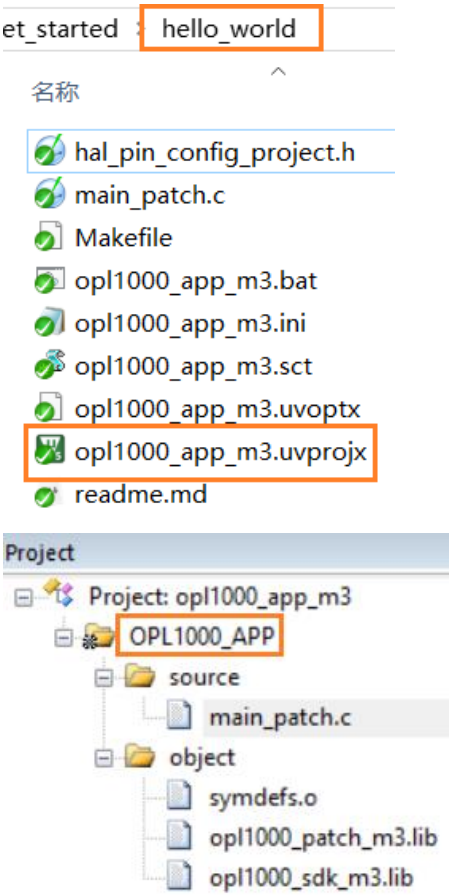
1. 在 github 上获取最新版的 SDK repo.

Figure 6: github 上的 SDK repo



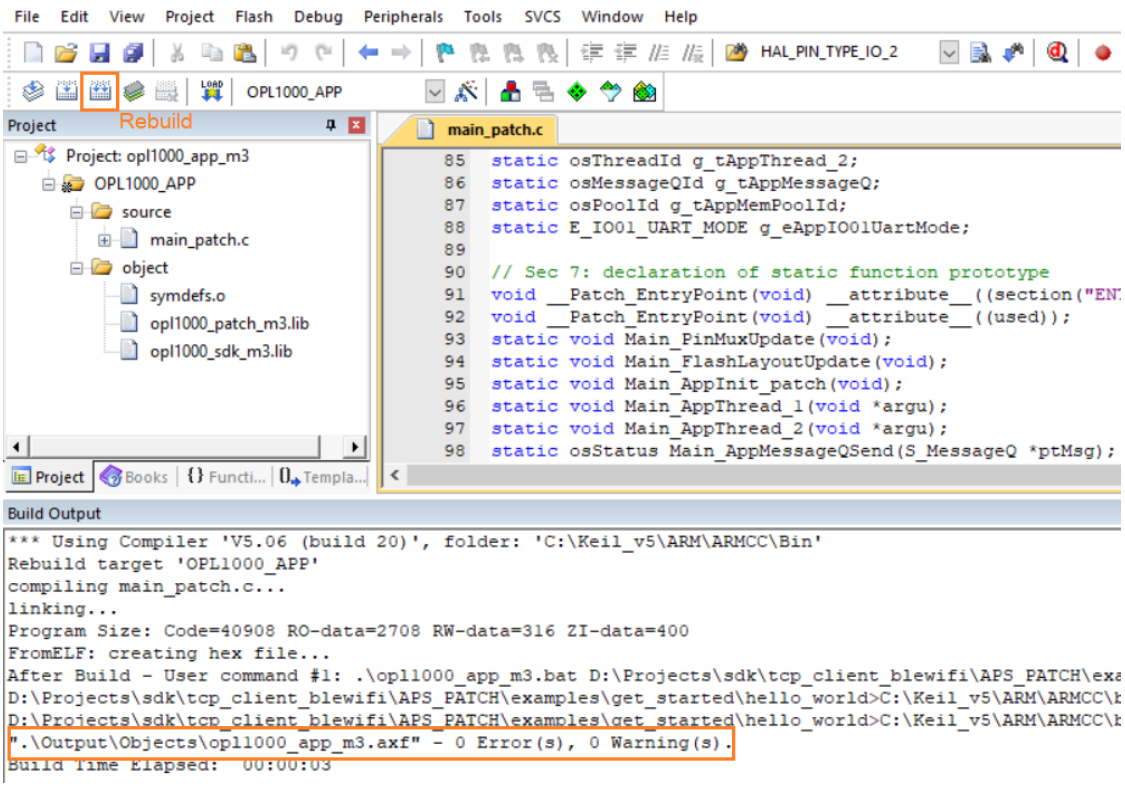
2. 下载好 repo 后，切换到 SDK \APS_PATCH\examples\get_started\hello_world 目录下，打开 opl1000_app_m3.uvprojx 工程文件

Figure 7: hello_world 工程文件和目录

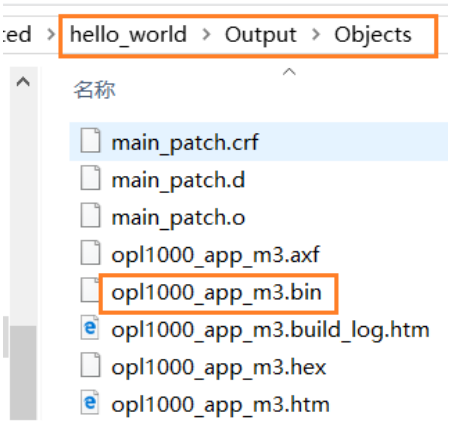



3. 在 Keil C 界面下，点击“rebuild”按钮，编译 hello_world 工程。

Figure 8: 编译 hello_world 工程

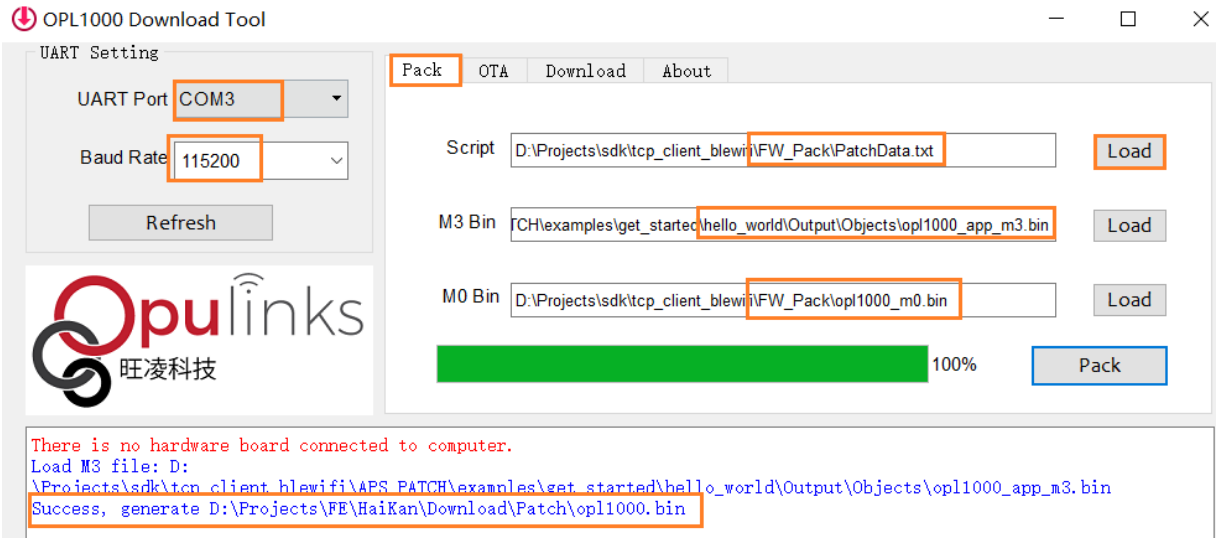



编译成功后在\Output\Objects 目录下生成 opl1000_app_m3.bin 文件如下：



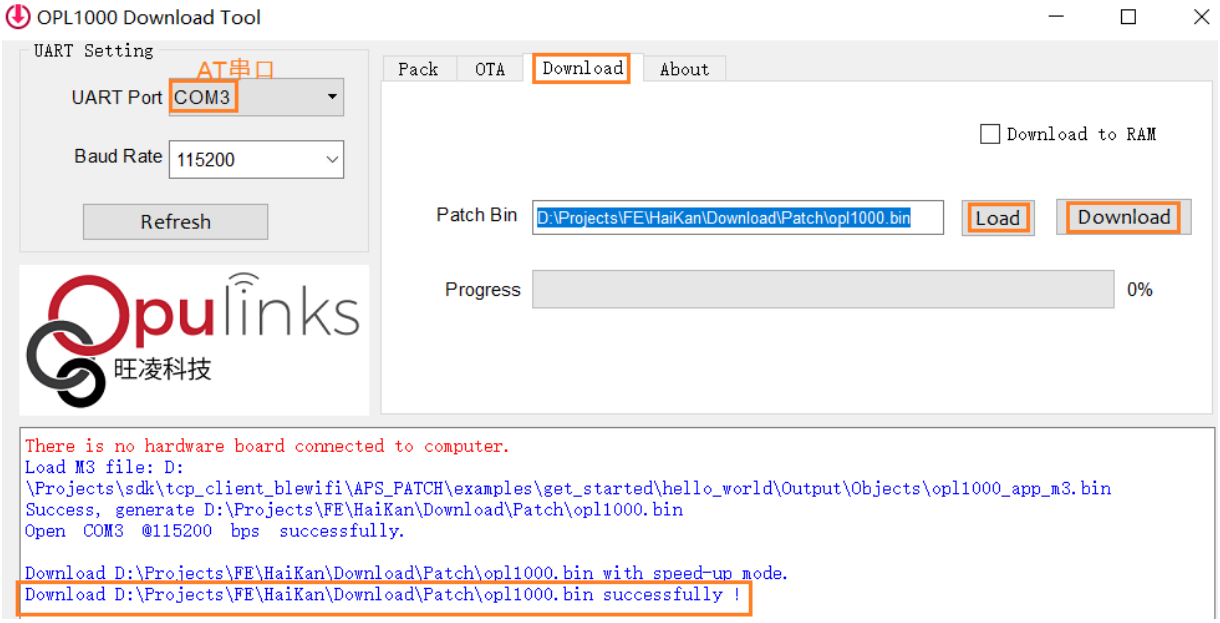
4. 在编译好后，打开 OPL1000 download tool 把上面生成的 opl1000_app_m3.bin 和 m0.bin 打包（pack）成 OPL1000 固件。关于如何下载并使用 OPL1000 download tool 的更多信息，请参考 OPL1000-patch-download-tool-user-guide.docx

Figure 9: pack M3 固件




5. 在 pack 好 OPL1000 固件后，切换到 Download 界面。点击“Load”按钮，选择上面 pack 好 OPL1000 固件（默认情况下已经选中它了），点击”Download”按钮开始下载【注意：在按下”Download”按钮后，请在 5 秒内复位一下开发板（按一下复位按钮）】

Figure 10: 下载 OPL1000 固件




6. 下载完成后，打开 APS 串口，查看在串口上打印的 log 信息如下：

Figure 11: 串口输出

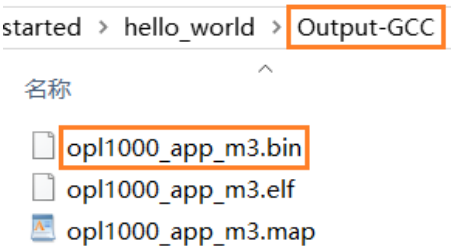
### 2.2.3. 用 GCC 编译工程

OPL1000 SDK 下的工程还提供了 Makefile，从而方便用户使用 GCC 编译它们。关于搭建 GNU ARM GCC 验证环境的更多信息，请参考 OPL1000-SDK-Development-guide.docx 的第四章。这里还是以 hello_world 为例，演示如何使用它。

在 DOS 窗口下，切换到 hello_world 工程所在目录，也就是 Makefile 所在目录，输入 make 命令如下

Figure 12: 用 GCC 编译工程


编译成功后在\ Output-GCC 目录下生成 opl1000_app_m3.bin 文件如下：



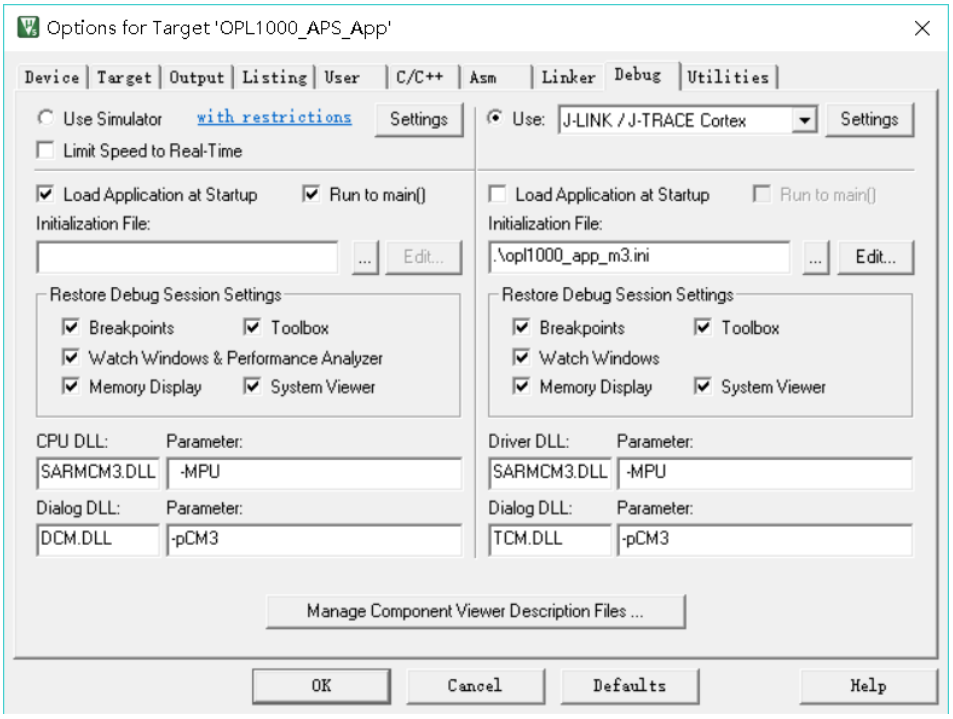
# 3. 使用 KEIL 调试应用程序

用户应用程序可以在 DEVKIT 板进行开发和调试。DEVKIT 板的使用请参考文献[2] DEVKIT 快速使用指南。

## 3.1. Keil 工程配置

DevKit 板使用 USB 供电，将 J-link 仿真器和 DevKit 板正确连接。打开工程：*SDK\APS_PATCH\examples\get_started\hello_world*。选择 Options for Target 按钮，弹出设置对话框，选择 Debug 界面，出现下图所示界面。

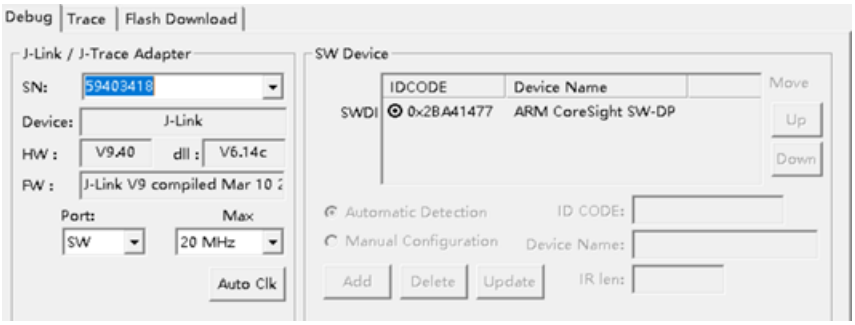
Figure 13: Options for Target 设置对话框




调试接口选择 JLINK / J-TRACE Cortex 接口 ，如果用户使用其他仿真器请选择对应的仿真接口类型，例如使用 CMSIS DAP 仿真器则需要选择 CMSIS-DAP Debugger 接口。选择正确后进入 Settings 按钮， 出

现 Figure 14 所示结果。在 SW Device 显示的是设备 ID 编码，表示设备连接和工作正常，如果 SW Device 为空则需要检查设备接线，确保接线正确。

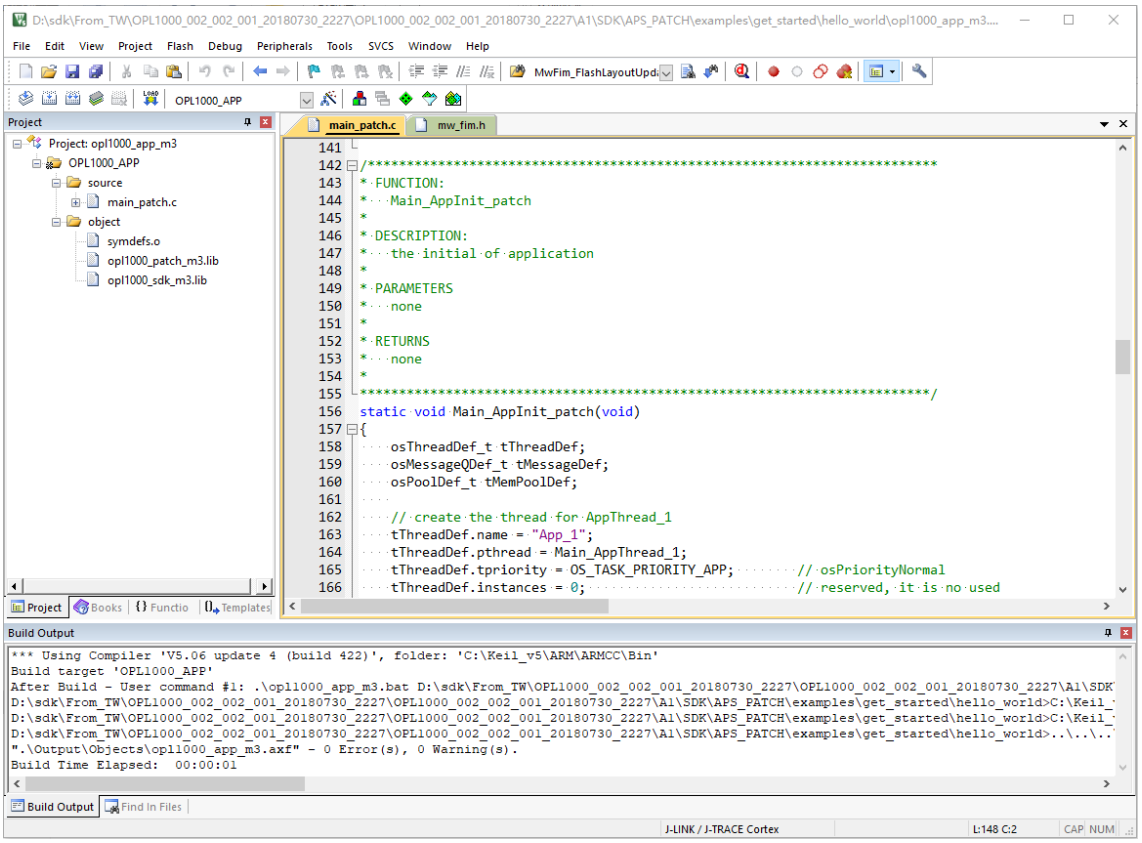
Figure 14: J-link ICE 仿真器正确识别




## 3.2. 应用程序在线调试

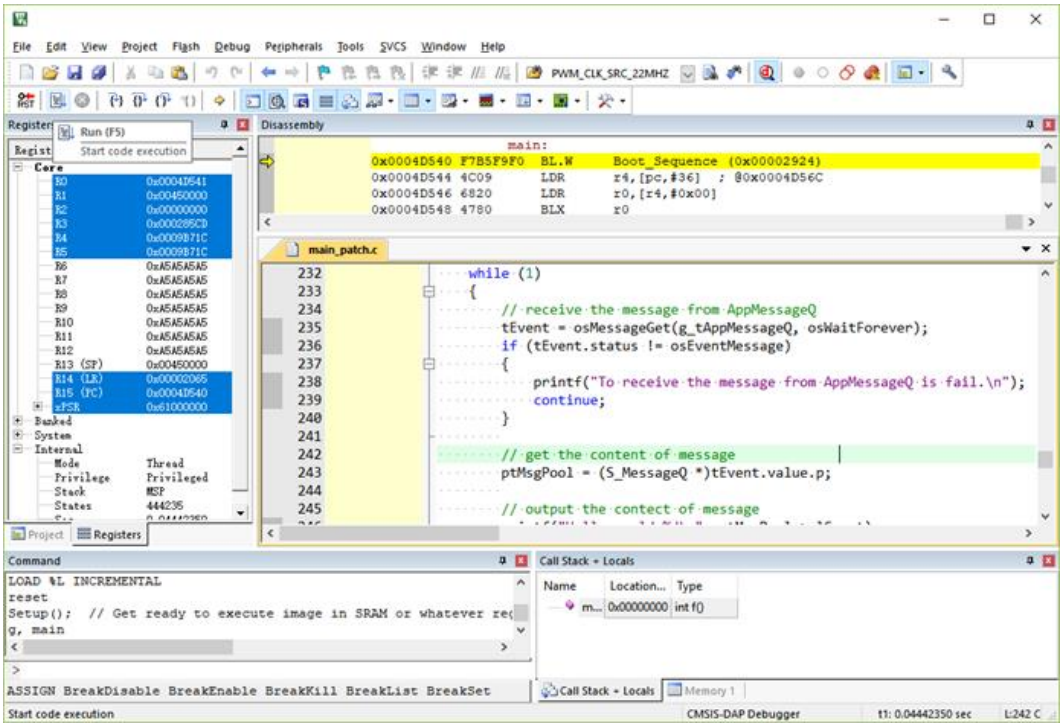
Keil 软件正确识别到 OPL1000 设备后，build 示例工程，得到 Figure 15 画面，表示编译正确完成。

Figure 15: Keil 编译 hello_world 示例代码



编译完成后，点击 Debug 按钮，正确进入 debugger 环境，如图 Figure 16 所示。

Figure 16: 使用 Keil C 在线调试




在确保串口接线正常的情况下，使用串口工具打开 APS 串口，同时在 keil 里面点击全速运行 ，此时 APS 串口打印如图 Figure 17 所示 log 信息，表明程序在 RAM 中执行正常 。

Figure 17: APS 串口打印确认程序执行结果

```
The init of MW_FIM is done.

[SVN REV] SVN_REVISION:1655

[Lib] SDK version info: 2221
[Lib] Compile time: 2018/07/30 11:47:02
wifiMac Task create successful
Supplicant task is created successfully!
controller_queue creates successful!
controller_queue_ble creates successful!
controller_task_create successful!
LE Task create successful
[SYS_COMMON]: OTP not support yet, read STA default mac addres.
WIFI sta cfg REQ idx=1, value=0!
There is no any OTA image.
Sw patch is changed successfully.
Hello world 1
Hello world 2
Hello world 3
Hello world 4
Hello world 5
Hello world 6
Hello world 7
Hello world 8
Hello world 9
Hello world 10
```

# CONTACT

sales@Opulinks.com