目 录

1. 实验目的 1

2. 系统总述 2

3. 实验思路与结果 3

4. 讨论题 6

5. 感想与收获 7

# 实验目的

了解BootLoader的工作原理。

能够利用TI的相关例程进行BootLoader及应用APP的操作。

最终能够将自己的APP项目修改为能够被BootLoader调用的应用APP。

# 系统总述

LM Flash Programmer\_1613，是一款免费的闪存编程实用程序，与Texas Instruments Stellaris或Tiva C系列微控制器一起使用。通过bootloader下载更新固件需要用到LMFlashProgrammer。

TivaWare\_C\_Series-2.2.0.295，TI TivaWare C系列评估、开发和演示软件

软件包安装完成后例程位于SW-TM4C-2.1.4.178\examples\boards\ek-tm4c1294xl\下。与bootloader相关的有五个目录，分别对应串口更新与网口更新两个部分，对应串口的是boot\_serial，boot\_demo1和boot\_demo2。其中boot\_serial是bootloader本身，boot\_demo1和boot\_demo2是两个跑马灯例程，一个闪LED1，一个闪LED2，主要就是演示下载不同固件用的。

|  |  |  |
| --- | --- | --- |
| 名称 | 项目位置 | 说明 |
| Boot\_serial | C:\ti\TivaWare\_C\_Series-2.2.0.295\examples\boards\ek-tm4c1294xl\boot\_serial | Bootloader程序 |
| Boot\_demo1 | C:\ti\TivaWare\_C\_Series-2.2.0.295\examples\boards\ek-tm4c1294xl\boot\_demo1 | 闪烁D1应用程序 |
| Boot\_demo2 | C:\ti\TivaWare\_C\_Series-2.2.0.295\examples\boards\ek-tm4c1294xl\boot\_demo2 | 闪烁D2应用程序 |

三个项目中，在各自的子目录.\rvmdk下，分别有如下三个bin文：Boot\_serial.bin、Boot\_demo1.bin、Boot\_demo2.bin

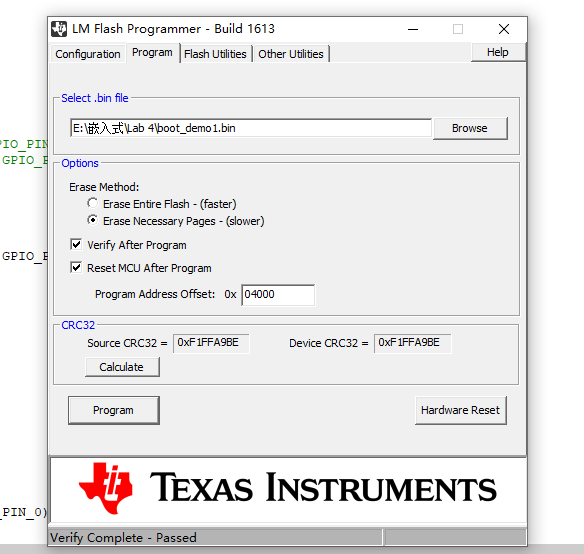
axf文件、[hex](https://so.csdn.net/so/search?q=hex&spm=1001.2101.3001.7020" \t "_blank)文件与bin文件都是可以运行在我们的芯片上的，它们都存储了编译器根据源代码生成的机器码，根据应用场合的不同，它们又有所区别：

axf文件：包含调试信息。axf文件是编译默认生成的文件，不仅包含代码数据，而且还包含着调试信息，在MDK里进行debug调试用的就是这个文件。

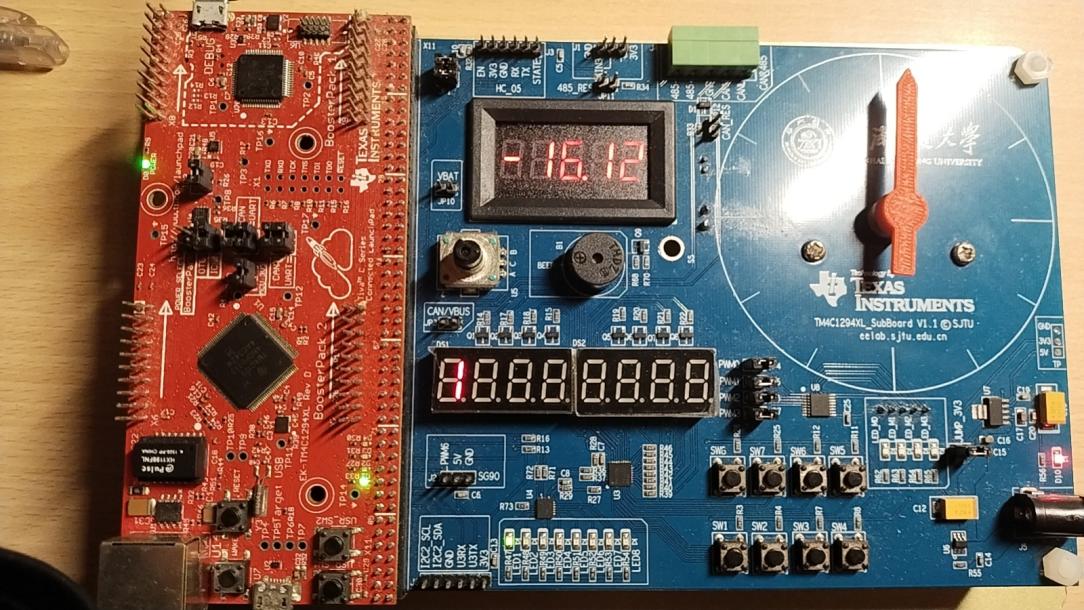
hex文件：包含地址信息。hex 文件是一种使用十六进制符号表示的代码记录， 记录了代码应该存储到FLASH 的哪个地址，下载器可以根据这些信息辅助下载。

bin文件：最直接的代码映像。bin文件就是最小的可以运行的文件了，其包含最直接的代码映像。

# 实验思路与结果

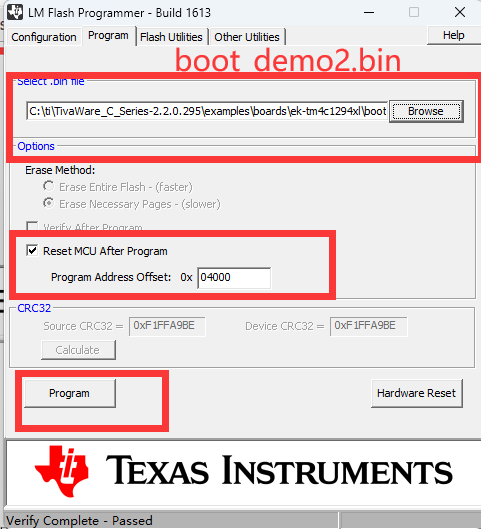
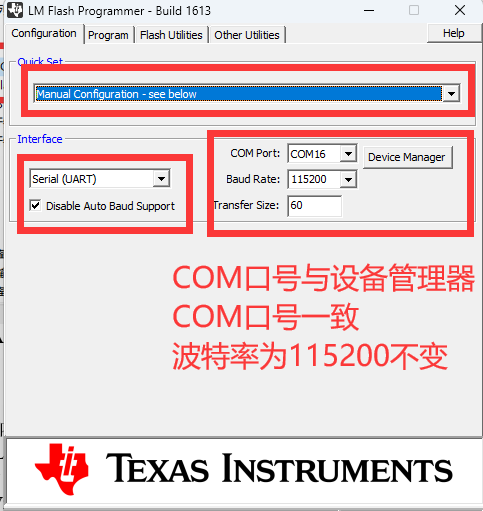


将地址设为0x04000,选中boot\_demo1.bin



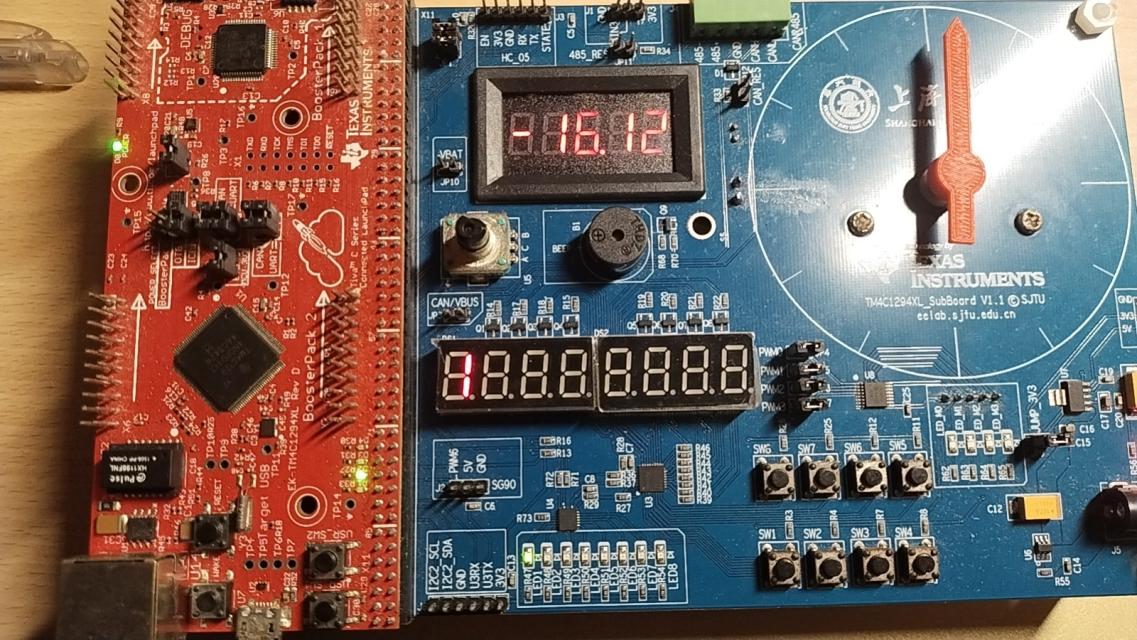
图一、D1闪烁

boot\_serial和boot\_demo1分别烧入0x0和0x4000开始的位置。系统复位后，先进入boot\_serial，因为没有按下SW1，因此bootloader程序自动跳转到预置的0x4000处执行boot\_demo1程序。

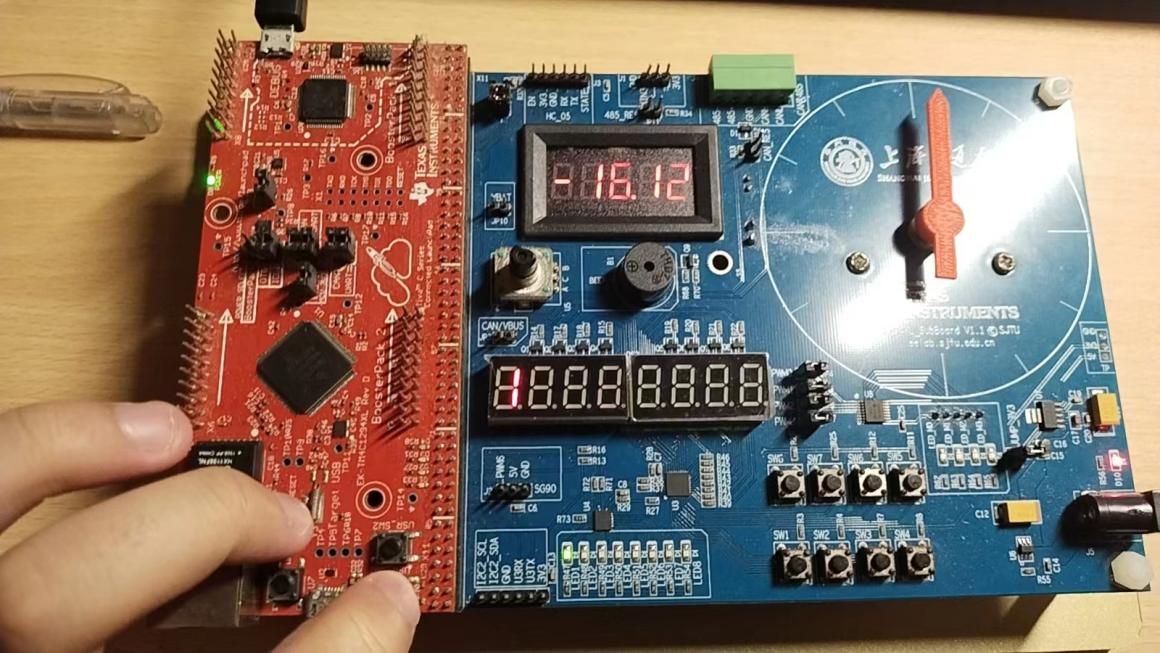


在Program前，按住SW1按键，然后再按Program键。强置bootloader程序进行UART通讯，将boot\_demo2.bin文件通过UART传送到MCU中。然后按一下LaunchPad板上的RESET按键，进行复位。就能看到D2闪烁。

如果在按下RESET时，同时按下SW1键，则进入Bootloader状态，并且不会跳转到boot\_demo2，因此没有LED闪烁。程序始终在bootloader中。此时如果需要跳转到boot\_demo2，则需要再按一次RESET复位即可。当没有SW1按下时，程序自动从bootloader跳转到应用APP中。即boot\_demo2。

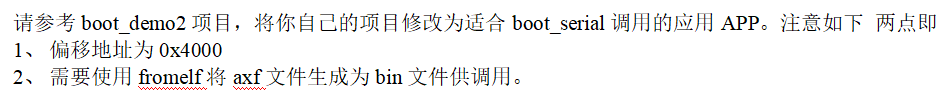


图二、D2闪烁



图三、处于Bootloader状态

# 讨论题



1: 修改偏移地址为0x4000

(1). 打开的Keil项目。

(2). 选择 "Options for Target 'Target 1'"（或你的目标名称）。

(3). 在 "Target" 选项卡中，找到 "IROM1" 的配置，设置起始地址为 0x4000，并相应调整大小。可以设置为：

Start: 0x4000

Size: 0x7C000 (512KB - 16KB)

2: 使用fromelf将axf文件生成为bin文件

(1). 打开Keil的命令行工具（Command Prompt）或者在Windows的命令提示符中，导航到你的项目输出目录。

(2). 使用fromelf工具将 .axf 文件转换为 .bin 文件:

fromelf --bin --output your\_project.bin your\_project.axf

1. 打开Keil，加载项目 MyApp。

2. 进入 Options for Target 'MyApp'，在 Target 选项卡中，设置如下：

- IROM1

- Start: 0x4000

- Size: 0x7C000

3. 编译项目，生成 MyApp.axf 文件。

4. 打开命令提示符，导航到项目输出目录。

5. 将 .axf 文件转换为 .bin 文件：

C:\Keil\_v5\ARM\ARMCC\bin\fromelf --bin --output MyApp.bin MyApp.axf

# 感想与收获

1. Bootloader的原理：

Bootloader（引导加载程序）是一段在系统启动时运行的代码，用于加载操作系统或应用程序。它的主要功能是在设备上电或复位时初始化硬件，加载和执行应用程序或操作系统。

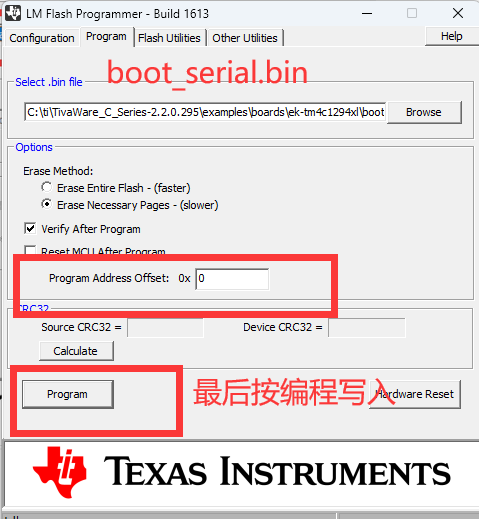
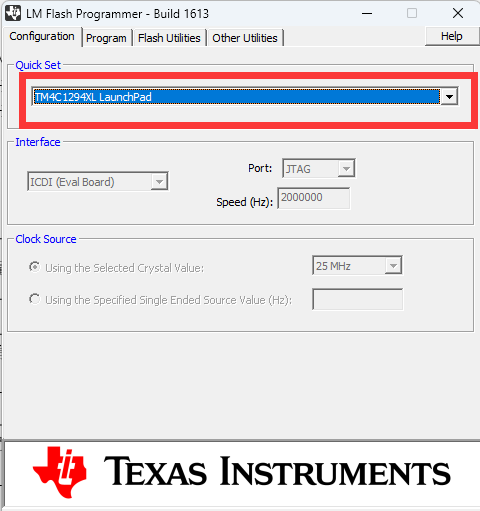
(1).启动硬件初始化：当设备上电或复位时，处理器会从一个预定义的地址开始执行代码。这段代码通常位于只读存储器（如ROM或闪存）中，称为BootLoader的第一阶段（Stage 1）。

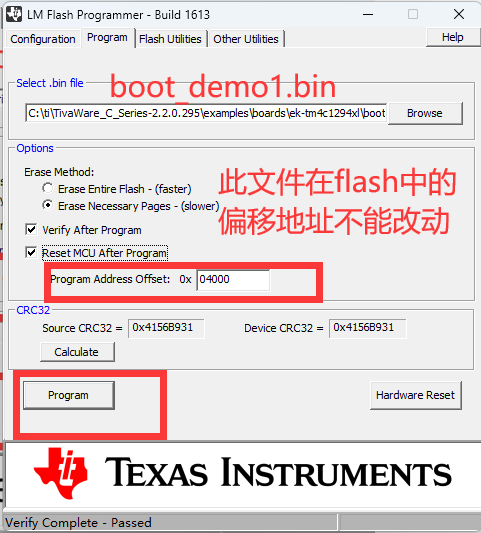
在这一阶段，BootLoader执行最基本的硬件初始化，例如设置时钟、配置堆栈指针、初始化内存控制器等，以便系统能够正常工作。

(2).检查和选择启动源：BootLoader通常会检查一些启动源（如闪存、SD卡、网络等），以确定从哪里加载下一阶段的代码。有些BootLoader还会根据特定条件（如引导引脚状态、配置寄存器等）来选择不同的启动路径。

(3).加载和执行应用程序：BootLoader从预定的存储位置（如闪存中的特定偏移地址）加载应用程序代码到内存中。这一过程可能包括解压缩和校验代码的完整性。一旦应用程序加载到内存中，BootLoader将跳转到应用程序的入口地址，开始执行应用程序的代码。

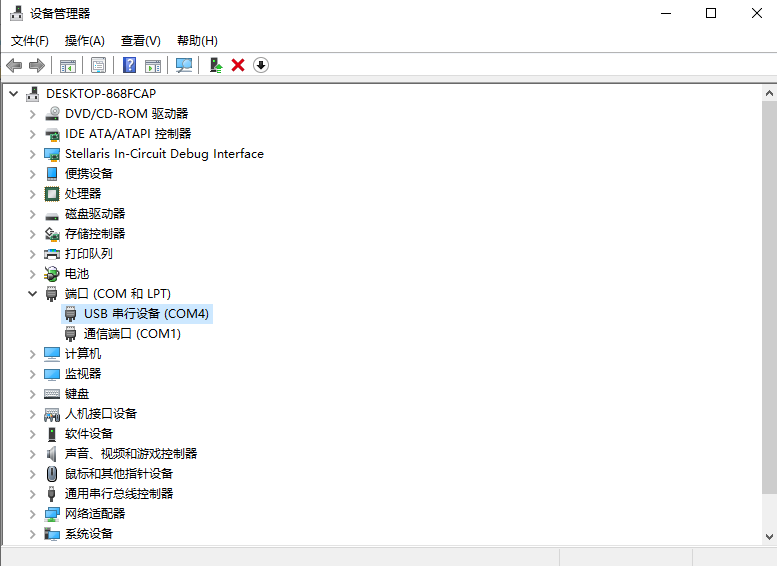
2、学会了程序写入的操作：



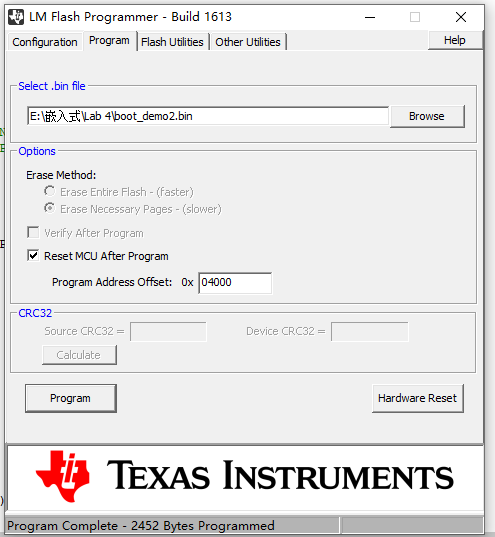
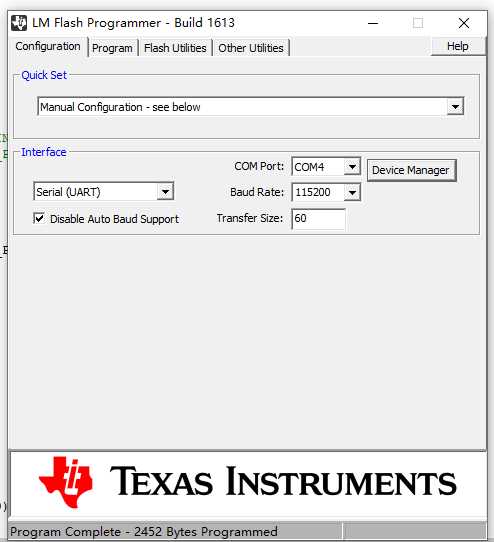


将已有的bin文件录入到板子中并正常运行。

3、将bin文件通过UART，利用BootLoader写入到FLASH的0x4000地址：



在虚拟机中USB串口对应的端口号为COM4



在Program前，按住SW1按键，然后再按Program键。强置bootloader程序进行UART通讯，将boot\_demo2.bin文件通过UART传送到MCU中。然后按一下LaunchPad板上的RESET按键，进行复位。就能看到D2闪烁。

如果在按下RESET时，同时按下SW1键，则进入Bootloader状态，并且不会跳转到boot\_demo2，因此没有LED闪烁。程序始终在bootloader中。此时如果需要跳转到boot\_demo2，则需要再按一次RESET复位即可。当没有SW1按下时，程序自动从bootloader跳转到应用APP中。即boot\_demo2。

从而便可手动载入bin文件。