ARM 实验（Cortex-M4）

实验一　时钟选择与 GPIO 实验

一. 实验目的

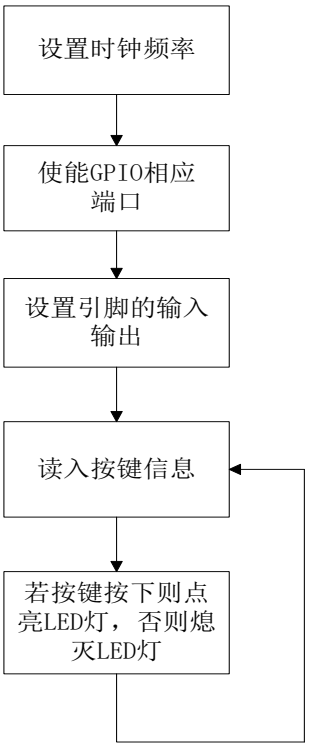
熟悉 ARM 的集成开发环境 KEIL uVision5，能够自行建立一个实验用工程项目。
理解 CPU 的时钟信号，了解不同时钟对电源消耗的不同。
掌握 GPIO 的工作原理，能够结合 GPIO 的输入与输出功能进行实验。

二. 实验要求

1. 实验 1.1 程序 exp1.c，实现功能如下：
   当按下 USR_SW1 按键时，PF0 快闪；当松开 USR_SW1 时，PF0 慢闪。共调用了三个函数，分别为
   S800_GPIO_Init　　使能 F 端口，使 PF0 管脚为输出
   PF0_Flash　　根据传入的按键参数，决定 PF0 快闪或慢闪
   Delay　　根据传入的参数，决定长延时或短延时
   请修改程序：
   分别使用内部 16M 的 HSI 时钟 ，外部 25M 的 HSO 时钟，以及 PLL 时钟进行 GPIO-PF0 的闪烁。
   参阅实验 1.1 的程序。并观察电流表显示的电流变化。
   用示波器观察 PF0 的频率在不同的时钟频率下是否有变化。
   进入 DEBUG 模式，设置断点，观察设置时钟后时钟返回值的大小是否符合设定。
2. 修改实验 1.1 的程序，实现如下功能：
   当按下 USR_SW1 键时，点亮 LED_M0，放开时，熄灭 LED_M0
   当按下 USR_SW2 键时，点亮 LED_M1，放开时，熄灭 LED_M1
3. 修改实验 1.1 的程序，实现如下功能：
   当第一次短按 USR_SW1 键时，闪烁 LED_M0，
   当第二次短按 USR_SW1 键时，熄灭 LED_M0
   当第三次短按 USR_SW1 键时，闪烁 LED_M1
   当第四次短按 USR_SW1 键时，熄灭 LED_M1
   如此循环往复

三. 实验框图

实验要求 2：



四. 实验结果

应能现场演示实验的效果。

五. 讨论

1. 人为修改内部时钟或外部时钟，如将内部时钟改为 8M，或将外部时钟改为 30M，会有什么结果？
2. 能否将 PLL 时钟调整到外部时钟的频率以下？如将 25M 外部时钟用 PLL 后调整为 20M？
3. 将 PLL 后的时钟调整为最大值 120M，LED 闪烁会有什么变化？为什么？
4. GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_PIN_0)此函数中，每个函数项的意义。第三个函数项为 GPIO_PIN_0，如果改为 1 或改为 2，或其他值，分别有什么现象？
5. 结合硬件说明 GPIOPinConfigure 行的作用。如果此行注释，在 WATCH 窗口中观察 key_value 值会有什么变化。
6. 为什么在慢闪时，按住 USR_SW1 时，没有马上进入快闪模式。

六、程序中涉及相关固件库函数说明，

页码为 SW-TM4C-DRL-UG-2.1.4.178.pdf 中页码。

函数名	页码	功能说明
SysCtlPeripheralEnable	505	外设使用前必须先使能对应的外设。不使能直接使用会报总线错误。
SysCtlPeripheralReady	508	等待外设使能后硬件准备完成
SysCtlClockFreqSet	487	设置系统时钟值
IntMasterEnable	354	使能 CPU 总中断
GPIOIntRegister	261	为 GPIO 端口注册中断处理程序。
GPIOPinTypeGPIOOutput	273	配置某个管脚为 GPIO 输出

GPIOPinTypeGPIOInput	272	配置某个管脚为 GPIO 输入
GPIOPadConfigSet	265	配置某个管脚的 PAD 功能
GPIOIntTypeSet	262	设置某个管脚的中断触发类型，例上升，下降或双边
GPIOIntEnable	260	使能某个管脚的中断，需要结合中断触发类型以及 GPIO 端口注册中断使用
GPIOIntStatus	262	获取指定 GPIO 端口的中断状态
GPIOIntClear	259	清除指定的 GPIO 中断源
GPIOPinWrite	284	对某个管脚写入指定值，高或低
GPIOPinRead	266	读取指定引脚的当前值

实验二 I2C GPIO 扩展及 SYSTICK 中断实验

一. 实验目的

了解 I2C 总线标准及在 TM4C1294 芯片的调用方法
掌握用 I2C 总线扩展 GPIO 芯片 PCA9557 及 TCA6424 的方法
能够通过扩展 GPIO 来输出点亮 LED 及动态数码管
熟悉 SYSTICK 中断调用方式，掌握利用软定时器模拟多任务切换的方法

二. 实验要求

1. 例程 2-1.c 为两个芯片的初始化，PCA9557 点亮所有 LED；TCA6424 控制数码管显示 0 在第一位；阅读并理解。

2. 进行 LED 的跑马灯实验，当 LED 在某位点亮时，同时在数码管的某位显示对应的 LED 管号。如 LED 跑马灯时，从左到右依次点亮 LED1~LED8，此时在数码管上依次显示 1~8。

3. 接程序 2，当按键 USR_SW1 按下时，停止跑马灯，但 LED 及数码管显示维持不变；当按键松开后，继续跑马灯。

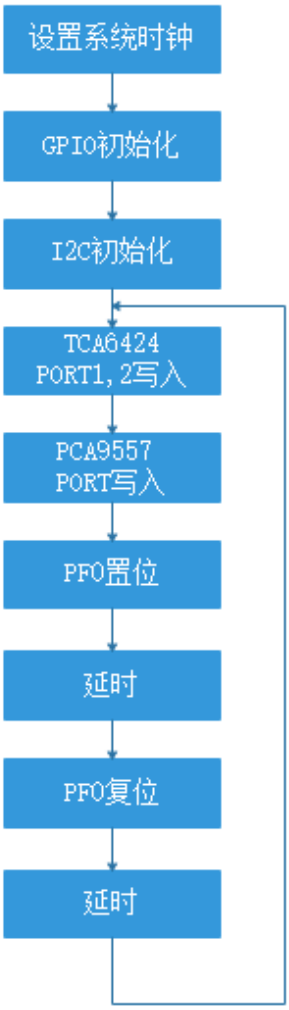
4. 根据中断例程，在中断程序中实现实验 2，3。

5. 选做。PF0 闪烁作为任务 1，LED 跑马灯及数码管循环显示作为任务 2，在 SYSTICK 的时间调度下，两个任务优先级相同，按时间片在运行。现在增加任务 3，按下 USR_SW1 按键时，PN0 常亮，松开按键熄灭。任务 3 的优先级高于任务 1，2，即任务 3 在执行时（按下 USR_SW1），任务 1，2 均不执行。

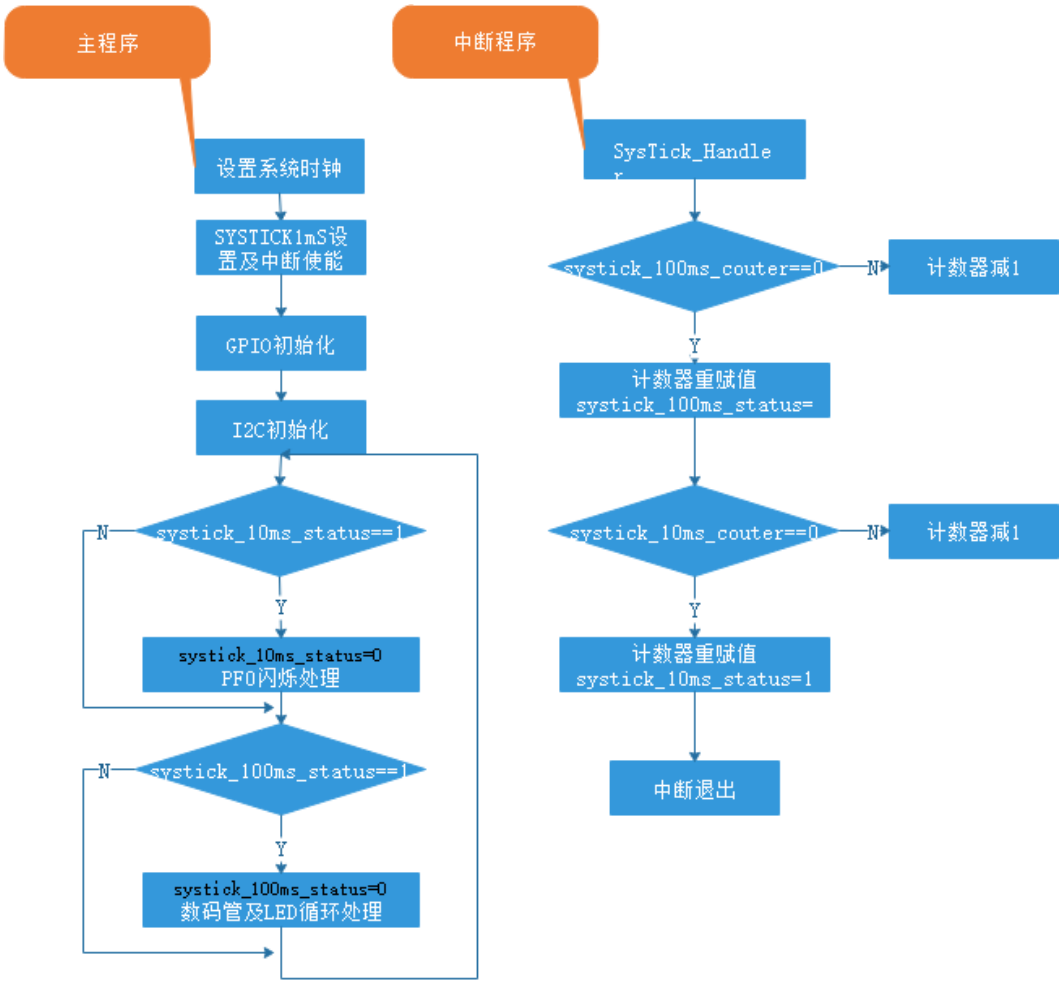
注：当有 I2C 器件时，系统时钟不宜超过 20M。

三.实验框图

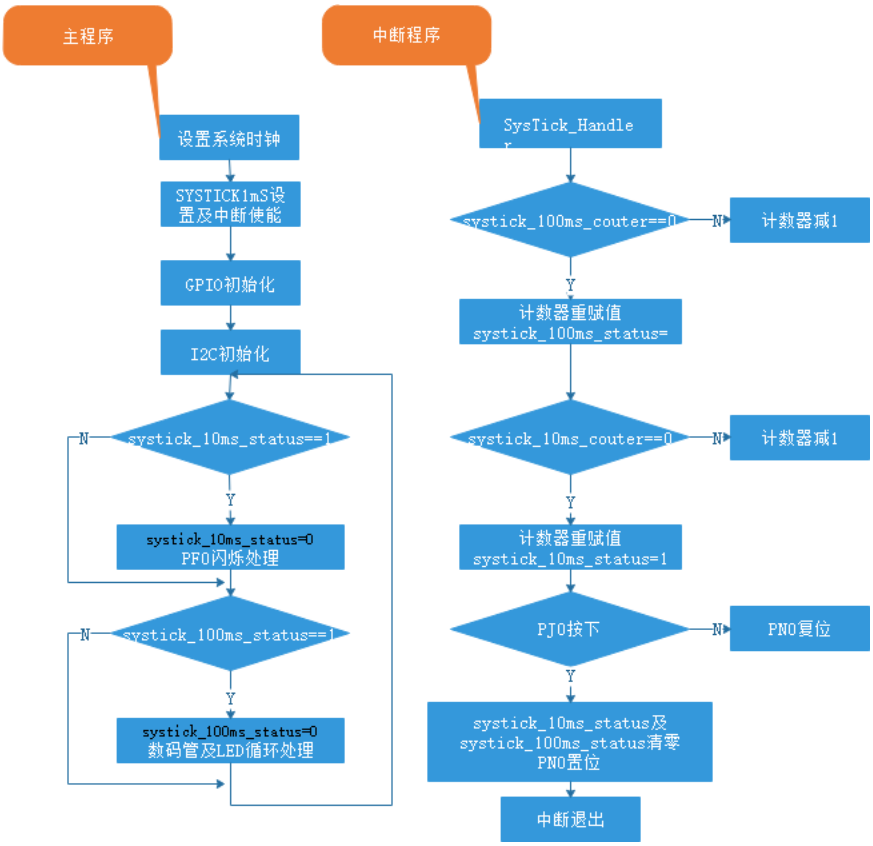
实验 1：


设置系统时钟
GPIO初始化
I2C初始化
TCA6424
PORT1,2写入
PCA9557
PORT写入
PFO置位
延时
PFO复位
延时

实验 4：


主程序
设置系统时钟
SYSTICK1mS设置及中断使能
GPIO初始化
I2C初始化
systick_10ms_status==1
N
Y
systick_10ms_status=0
PF0闪烁处理
systick_100ms_status==1
N
Y
systick_100ms_status=0
数码管及LED循环处理
中断程序
SysTick_Handler
systick_100ms_couter==0
N
计数器减1
Y
计数器重赋值
systick_100ms_status=
systick_10ms_couter==0
N
计数器减1
Y
计数器重赋值
systick_10ms_status=1
中断退出

实验 5



四.实验结果

五. 讨论

1. 如果跑马灯要求为 2 位跑马，例：当显示为 1 时，跑马灯点亮 LED8，LED1，当显示为 2 时，跑马灯点亮 LED1，LED2，如此循环，如何实现？
2. 在 3 基础上，数码管显示也改为 2 位显示。例
   第一步　显示 1，2
   　　跑马灯显示 LED1,2
   第二步　显示 2，3
   　　跑马灯显示 LED2，3
   。。。
   第 8 步　显示 8，1
   　　跑马灯显示 LED8，1
   第 9 步　回到第一步
3. 用 USR_SW1 控制跑马灯的频率，
   按第 1 下，间隔为 1S
   按第 2 下，间隔为 2S
   按第 3 下，间隔为 0.2S
   按第 4 下，回到上电初始状态，间隔 0.5S
   以 4 为模，循环往复
4. 请编程在数码管上实现时钟功能，在数码管上最左端显示分钟+秒数，其中分钟及秒数均为 2 位数字。

如 12:00，共 5 位。

每隔一秒，自动加 1，当秒数到 60 时，自动分钟加 1，秒数回到 00，分钟及秒数显示范围 00~59。

当按下 USR_SW1 时，秒数自动加 1

当按下 USR_SW2 时，分钟自动加 1

当按下以上一个或两个按键不松开时，对应的显示跳变数每隔 200mS 自动加 1。即如下按下 USR_SW1 1S，则显示跳变秒数加 5

六、程序中涉及相关固件库函数说明，

页码为 SW-TM4C-DRL-UG-2.1.4.178.pdf 中页码。

函数名	页码	功能说明
GPIOPinConfigure	266	配置 GPIO 引脚的备用功能，即第二功能
GPIOPinTypeI2CSCL	275	配置引脚作为 I2C 设备的 SCL 引脚
GPIOPinTypeI2C	274	配置引脚作为 I2C 设备的 SDA 引脚
I2CMasterInitExpClk	331	初始化 I2CMaster 时钟
I2CMasterEnable	329	使能 I2C Master 功能
I2CMasterBusy	327	指示 I2C 主机是否繁忙
I2CMasterSlaveAddrSet	336	设置 I2C 主机放置在总线上的地址
I2CMasterDataPut	329	从 I2C 主机发送一个字节
I2CMasterControl	327	控制 I2C 主机的状态变更
I2CMasterErr	330	获取 I2C 主机的错误状态。
I2CMasterDataGet	328	接收已发送到 I2C 主机的字节

实验三 UART 串行通讯口及中断优先级实验

二. 实验目的

了解 UART 串行通讯的工作原理
掌握在 PC 端通过串口调试工具与实验板通过 UART 通讯的方法
掌握 UART 的堵塞式与非堵塞式通讯方法

二. 实验要求

1. 例程为 UART0 的初始化，实验板在初始化完成后向主机发送“HELLO,WORLD!”字符串。阅读 3-1.c 并理解。

2. 在实验 1 的基础上，通过 PC 端发送字符串，实验板收到后并原样返回。称为 UART ECHO。阅读 3-2.c 并理解。

3. 将实验 2 改写为非堵塞式方式，即中断方式进行发送与接收。当进行数据接收时，点亮 PN1。阅读 3-3.c 并理解。

4. 请编程实现一个虚拟 AT 指令集：
当 PC 端发来 AT+CLASS 后，实验板回以 CLASS######，其中#####为你的班级号
当 PC 端发来 AT+STUDENTCODE 后，实验板回以 CODE######，其中#####为你的学号

5. 选做。将 4 改为大小写均能适应。

6. 选做。请编程实现以下三个命令：
底板运行后自动实现 1S 计时。

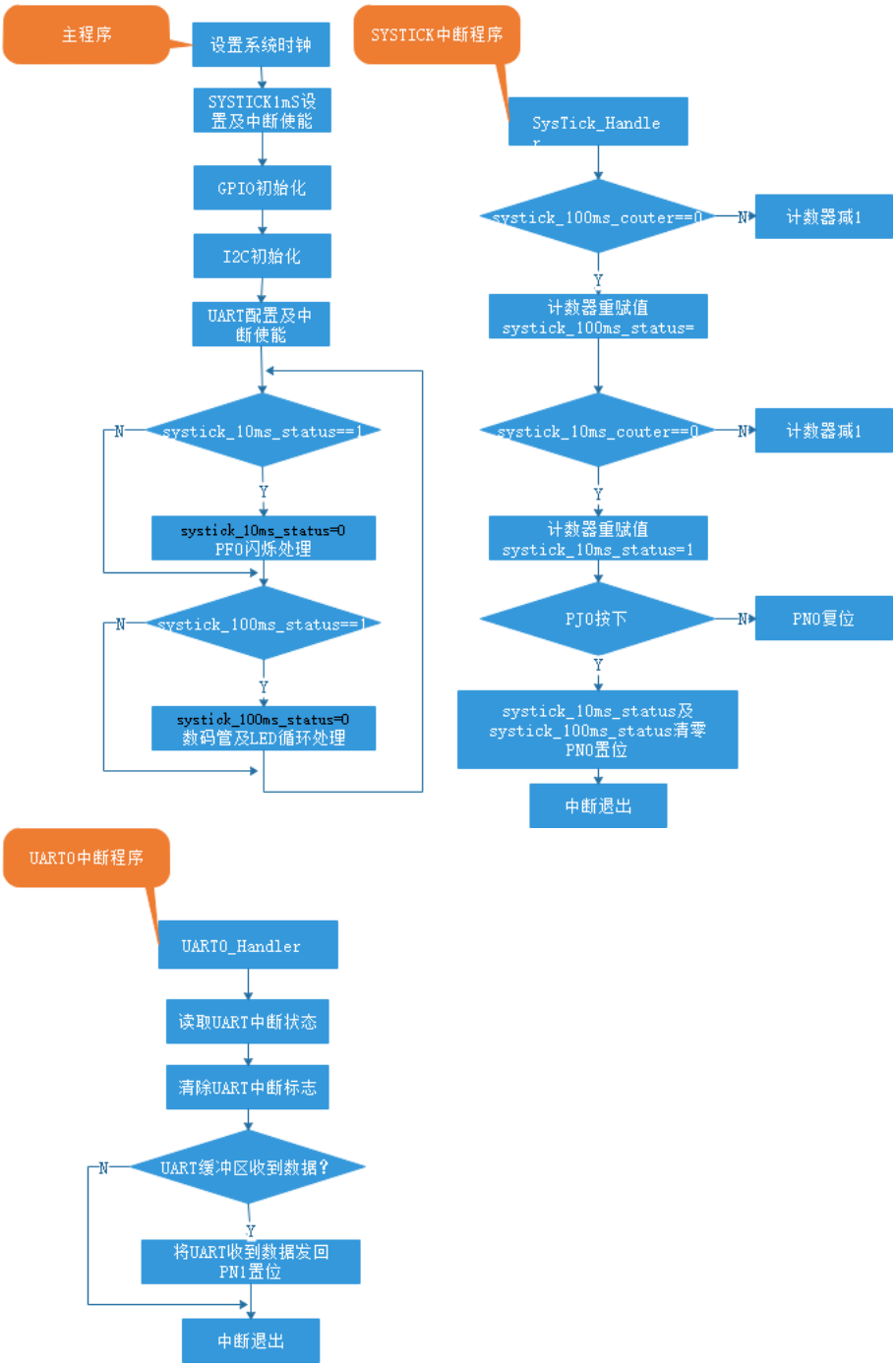
a) PC 端发来绝对对时命令，如 SET12:56:03 或 12-56-03，自动将当前时间同步到 12:56:03，并回之以当前时间
b) PC 端发来相对对时命令，如 INC00:00:12，自动将当前时间加 12 秒，并回之以当前时间
c) PC 端发来查询命令，GETTIME，自动回之以当前时间
d) 当前时间格式统一为 TIME12:56:03，其中 TIME 为字符，后续为时间值

7. 优先级调整实验。基于实验 3，调整 UART0 的优先级，使之高于 SYSTICK 的优先级，并处于抢占式优先，这样当按下 USR_SW2 时，UART0 不退出，导致 SYSTICK 中断不能进入，整个系统停滞。显示不再跳变。请阅读 3-4.c 并理解。

改写程序，将 SYSTICK 的抢占式优先级高于 UART0，从而达到即使 USR_SW2 按下时，SYSTICK 中断仍然能进入。

三.实验框图

实验 3



四.实验结果

五. 讨论

1. 实验 3-2，if (UARTCharsAvail(UART0_BASE))此行程序的作用。如果没有此行，会导致什么问题？
2. 实验 3-3，void UART0_Handler(void)为什么没有在主函数声明？
3. 为什么 3-3 的中断中需要读取中断标志并清除，而 SYSTICK 不需要
4. 请根据上位机的命令，如“MAY+01”，格式为：

其中 MAY 为月份，（JAN,FEB,…DEC）均为三位。
+表示加运算符,-表示减运算符，均为 1 位。
01 表示增加或减少量，均为 2 位。范围 00-11
以上均为 ASCII 码，
MAY+01 应该回之以 JUNE
MAY-06 应该回之以 NOV

5. 请根据上位机的命令，如“14:12+05:06”，格式为：
其中 14:12 为分钟与秒，共 5 位，包括一个“:”。
+表示加运算符,-表示减运算符，均为 1 位。
05:06 为分钟与秒的变化量，共 5 位。包括一个“：”，范围 00:00~23:59
以上均为 ASCII 码，
14:12+05:06 回之以 19:18

六、程序中涉及相关固件库函数说明，

页码为 SW-TM4C-DRL-UG-2.1.4.178.pdf 中页码。

函数名	页码	功能说明
GPIOPinTypeUART	281	配置引脚作为 UART 设备
UARTConfigSetExpClk	568	配置 UART，包括地址，波特率，数据格式等
UARTCharPut	565	从指定端口发送一个字节直到发送完成
UARTCharPutNonBlocking	565	从指定端口发送一个字节
UARTIntStatus	576	获取当前中断状态值
UARTIntClear	574	清除 UART 中断源
UARTCharsAvail	566	确定接收 FIFO 中是否有字符
SysTickPeriodSet	532	设置 SYSTICK 计数器的周期值
SysTickEnable	530	使能 SYSTICK 计数器
SysTickIntEnable	531	SYSTICK 中断允许
UARTIntEnable	575	启用单个 UART 中断源
IntEnable	352	使能某个中断
IntPriorityMaskGet	265	获取优先级屏蔽级别
IntPrioritySet	358	设置某个中断的优先级级别
IntPriorityGroupingGet	356	获取中断控制器的优先级分组

实验四　UART BootLoader 实验

一. 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。

二. 实验要求及操作

1. 实验程序说明
需要提前预装：
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

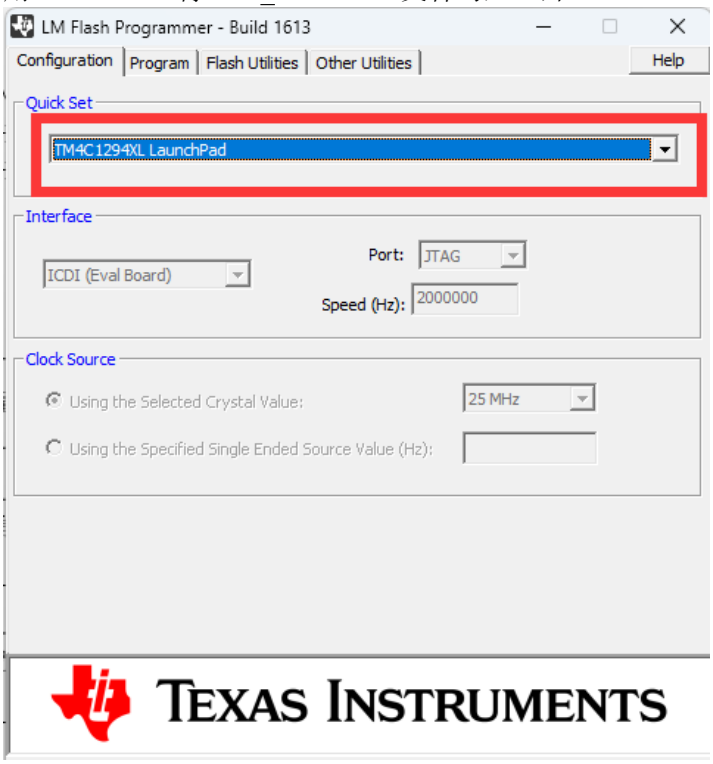
三个项目中，在各自的子目录.\rvmdk 下，分别有如下三个 bin 文件
Boot_serial.bin
Boot_demo1.bin
Boot_demo2.bin
axf 文件、hex 文件与 bin 文件都是可以运行在我们的芯片上的，它们都存储了编译器根据源代码生成的机器码，根据应用场合的不同，它们又有所区别：
axf 文件：包含调试信息。
hex 文件：包含地址信息。
bin 文件：最直接的代码映像。
axf 文件是编译默认生成的文件，不仅包含代码数据，而且还包含着调试信息，在 MDK 里进行 debug 调试用的就是这个文件。
hex 文件是一种使用十六进制符号表示的代码记录， 记录了代码应该存储到 FLASH 的哪个地址，下载器可以根据这些信息辅助下载。

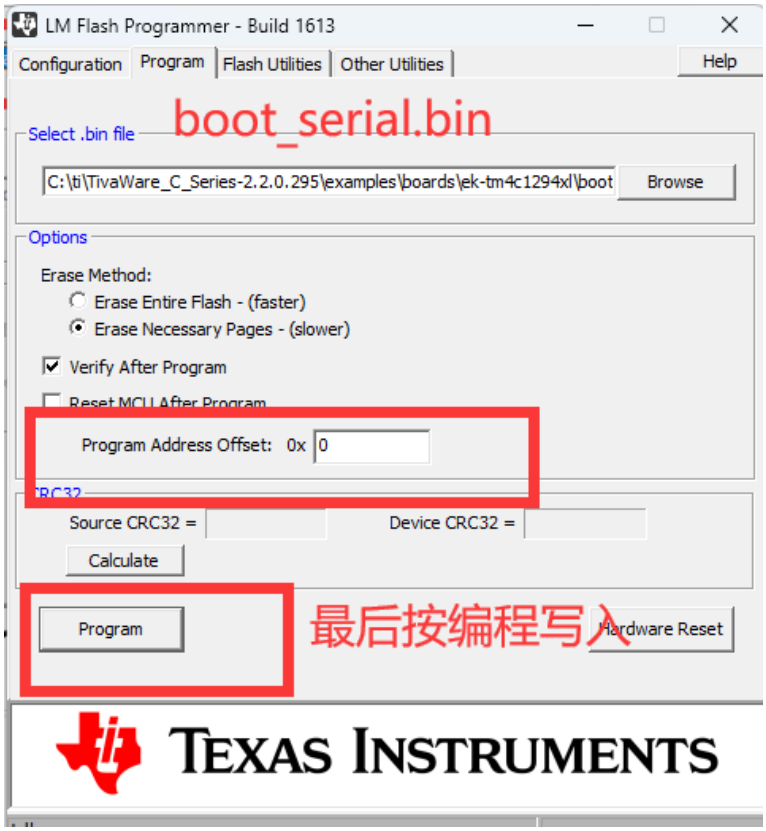
bin 文件就是最小的可以运行的文件了，其包含最直接的代码映像。

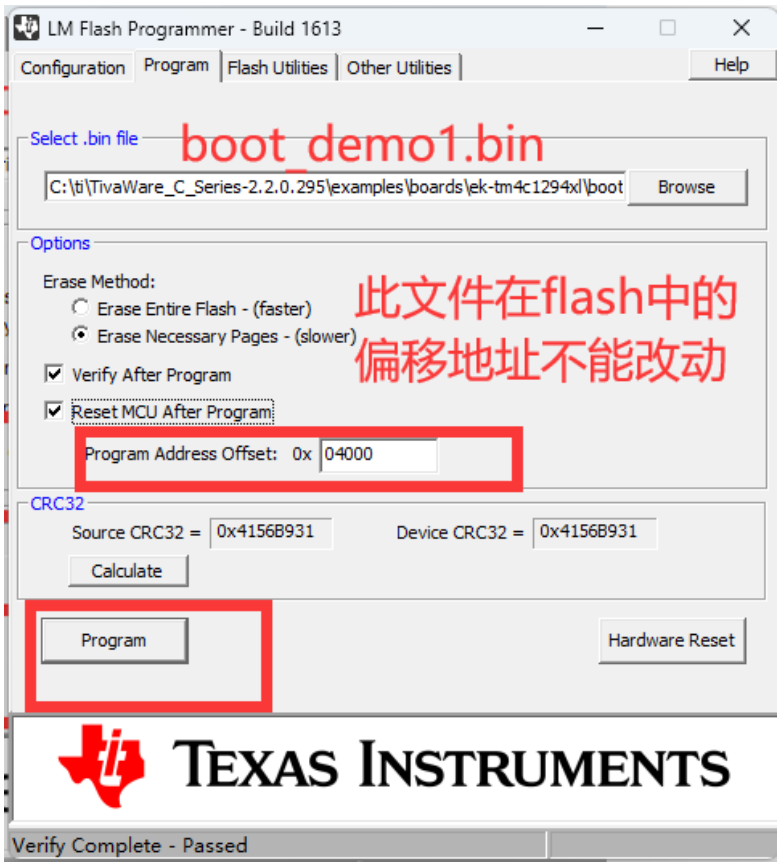
2. 程序写入操作

用 LMFLASH 将 boot_serial.bin 文件写入芯片。

用 LMFLASH 将 boot_demo1.bin 文件写入芯片。




LM Flash Programmer - Build 1613
Configuration
Program
Flash Utilities
Other Utilities
Help
boot_serial.bin
Select .bin file
C:\ti\TivaWare_C_Series-2.2.0.295\examples\boards\ek-tm4c1294xl\boot
Browse
Options
Erase Method:
Erase Entire Flash - (faster)
Erase Necessary Pages - (slower)
Verify After Program
Program Address Offset: 0x 0
Source CRC32 =
Device CRC32 =
Calculate
Program
最后按编程写入
Hardware Reset
TEXAS INSTRUMENTS


LM Flash Programmer - Build 1613
Configuration
Program
Flash Utilities
Other Utilities
Help
boot_demo1.bin
Select .bin file
C:\ti\TivaWare_C_Series-2.2.0.295\examples\boards\ek-tm4c1294xl\boot
Browse
Options
Erase Method:
Erase Entire Flash - (faster)
Erase Necessary Pages - (slower)
此文件在flash中的
偏移地址不能改动
Verify After Program
Reset MCU After Program
Program Address Offset: 0x 04000
CRC32
Source CRC32 = 0x4156B931
Device CRC32 = 0x4156B931
Calculate
Program
Hardware Reset
TEXAS INSTRUMENTS
Verify Complete - Passed

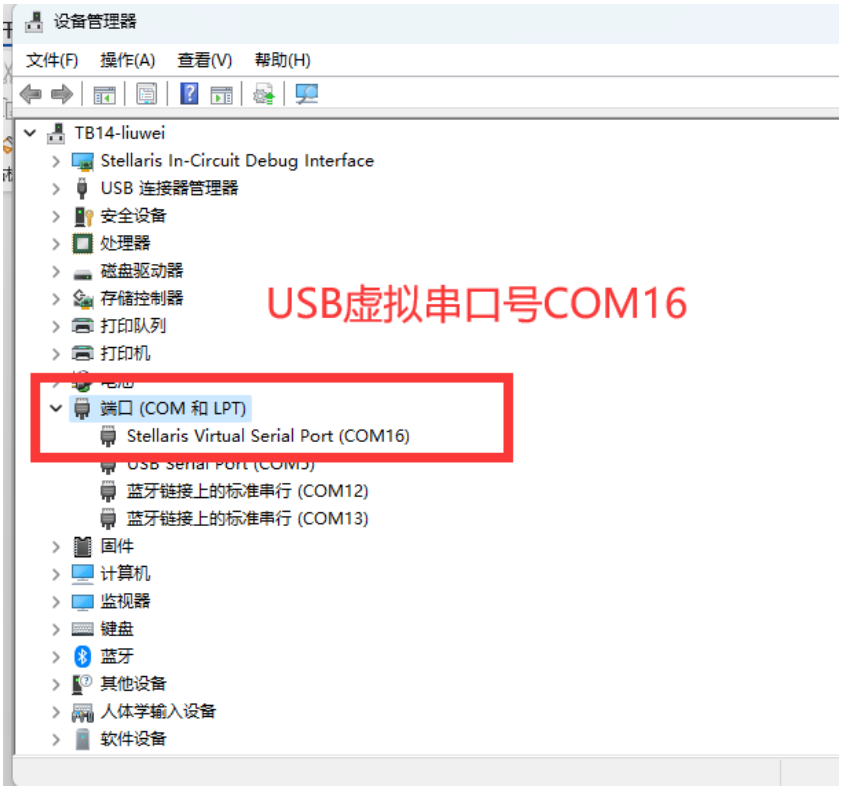
操作完成后，应该能看到 D1 开始闪烁。

这样将两个程序，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。

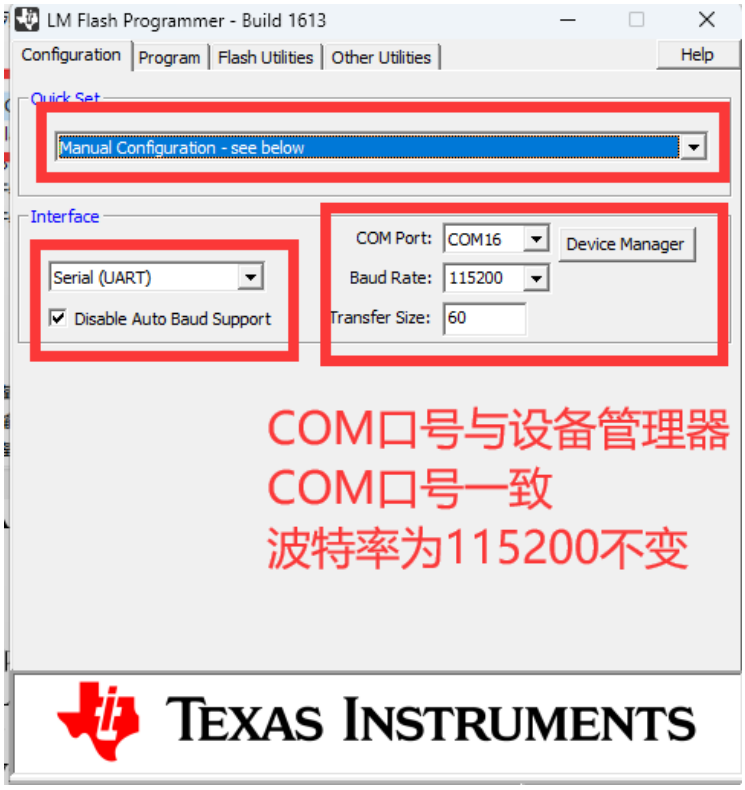
三. BootLoader-UART 实验

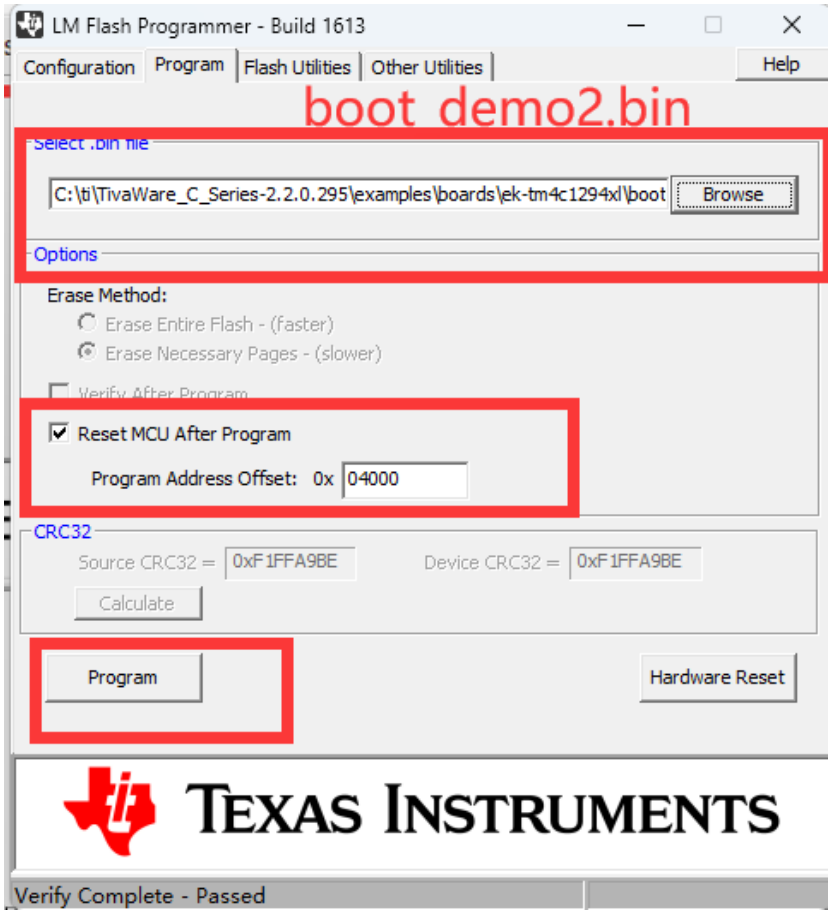
将 boot_demo2.bin 通过 UART，利用 BootLoader 写入到 FLASH 的 0x4000 地址

先检查设备管理器确定 USB 虚拟串口号并记录



修改 LMFLASH 的配置为手动，选择 UART 通信。


LM Flash Programmer - Build 1613
Configuration
Program
Flash Utilities
Other Utilities
Help
Quick Set
Manual Configuration - see below
Interface
Serial (UART)
Disable Auto Baud Support
COM Port: COM16
Device Manager
Baud Rate: 115200
Transfer Size: 60
COM口号与设备管理器
COM口号一致
波特率为115200不变
TEXAS INSTRUMENTS


LM Flash Programmer - Build 1613
Configuration
Program
Flash Utilities
Other Utilities
Help
boot_demo2.bin
Select .bin file
C:\ti\TivaWare_C_Series-2.2.0.295\examples\boards\ek-tm4c1294xl\boot
Browse
Options
Erase Method:
Erase Entire Flash - (faster)
Erase Necessary Pages - (slower)
Verify After Program
Reset MCU After Program
Program Address Offset: 0x 04000
CRC32
Source CRC32 = 0xF1FFA9BE
Device CRC32 = 0xF1FFA9BE
Calculate
Program
Hardware Reset
TEXAS INSTRUMENTS
Verify Complete - Passed

上图中，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。

五. 讨论

请参考 boot_demo2 项目，将你自己的项目修改为适合 boot_serial 调用的应用 APP。注意如下 两点即
1、 偏移地址为 0x4000
2、 需要使用 fromelf 将 axf 文件生成为 bin 文件供调用。

六、BootLoader 及应用 APP 的说明

参考文献：SPMU301E-TivaWare 引导加载程序用户指南

1、什么是 BootLoader 及应用 APP

BootLoader 引导加载程序是为微控制器（MCU）提供自编程能力的小程序。这种自编程功能通常通过 MCU 上的各种通信接口实现远程应用程序更新。通常，BootLoader 引导加载程序是在设备通电时执行的第一个固件。当引导加载程序运行时，它首先检查是否存在有效的应用程序固件。默认情况下，如果找到有效的应用程序，引导加载程序将执行该应用程序。如果找不到有效的应用程序，则引导加载程序会扫描每个支持的通信接口，寻找下载新应用程序固件的命令。

当应用程序正在运行时，也可以更新应用程序固件。该应用程序可以被设计为监听指定通信接口上的命令或 pin 事件。在接收到有效的命令或 pin 事件后，应用程序可以调用引导加载程序来重新启动引导加载过程。

有两种不同类型的引导加载程序：基于 ROM 的引导加载和基于闪存的引导加载。应用程序可以根据需要调用其中任一个。

因为闪存引导加载程序位于 0x0000.0000，所以应用程序必须位于闪存的另一个区域。通常，应用程序驻留在具有不同地址的不同闪存扇区中。有关给定设备的闪存扇区的详细信息，请参阅设备数据表。与检查地址 0x0000.0000 处是否存在有效应用程序的 ROM 引导加载程序相反，闪存引导加载程序检查指定给闪存引导加载器的不同地址处是否存在应用程序固件。闪存引导加载程序检查指定地址的前两个位置的内存是否具有不等于 0xFFFF 的值。FFFF。检查的结果确定闪存引导加载程序是执行用户应用程序还是加载新的应用程序固件。闪存引导加载程序提供了引导加载设备的最佳灵活性。

TivaWare 库提供了使用各种通信端口对设备进行编程的闪存引导加载程序的示例。

2、闪存引导加载程序

闪存引导加载程序是一小段代码，可以在闪存开始时编程，作为应用程序加载程序和 TM4C 微控制器上运行的应用程序的更新机制。引导加载程序可以构建为使用 UART、SSI、I2C、以太网、USB 或 can 端口来更新微控制器上的应用程序固件。因为提供了完整的源代码，所以引导加载程序可以完全自定义。闪存引导加载程序库的源位于 TivaWare boot_loader.h 文件夹中。

通过编辑 bl_config.h 文件，可以修改所提供的 flash 引导加载程序示例项目以使用不同的协议。bl_config.h 提供了各种 define 语句，用于配置如何执行 flash 引导加载程序。可以使用这些语句进行自定义，包括调整应用程序固件的起始地址和引导加载程序的通信接口。应用程序起始地址必须是 1024 字节的倍数（使其与页面边界对齐）。通常，在这个位置也需要一个矢量表。不要定义与闪存引导加载程序占用的区域

重叠的应用程序启动地址。闪存引导加载程序从地址 0x0000.0000 开始。

当在 bl_config.h 中定义应用程序起始地址时，应用程序固件必须使用 Code Composer Studio 链接器命令文件中的匹配地址构建。这是通过修改项目的.cmd 文件中 APP_BASE 的值来完成的。APP_BASE 的新值必须与 bl_config.h 中定义的应用程序起始地址匹配。此外，还必须调整应用程序内存长度。这是通过编辑系统内存映射内 FLASH（RX）的长度参数来完成的。新的长度是总闪存大小减去指定的起始地址。例如，如果新的起始地址位于位置 0x0000.4000，并且闪存大小为 0x0010.0000，则将设置的新长度参数为（0x0010.0000-0x0000.4000）=0x000F。C000。bl_config.h 中定义的应用程序起始地址指示闪存引导加载程序在哪里对应用程序映像进行编程，而链接器命令文件中的应用程序开始地址指示链接器在哪里分配代码。

在引导加载程序将控制权转移到应用程序之前，它将嵌套矢量中断控制器（NVIC）矢量表起始地址编程为 bl_config.h 文件中为 VTABLE_START_address 定义的值。这通常与应用程序的起始地址相同。应用程序的链接器命令文件应将中断向量节.intvecs 绑定到此地址。如果使用动态中断矢量，则在 RAM 存储器中定义一个矢量表，并将其与 1024 字节的边界对齐。这可以在 TivaWare boot_demo 示例项目中看到，其中该部分在链接器命令文件中标记为.vtable。对 TivaWare 函数 IntRegister（）的第一个调用将应用程序的矢量表从基于闪存的表.intvcs 复制到 RAM 表.vtable，然后修改要注册的中断源的矢量。

根据外围设备的不同，使用了三种不同的更新协议。对于 UART、SSI、I2C 和 CAN，使用自定义协议与下载实用程序通信，以传输固件映像并将其编程为闪存。然而，对于以太网或 USB DFU，使用不同的协议。在以太网上，使用标准引导协议（BOOTP）。对于 USB DFU，更新是通过标准的设备固件升级（DFU）类执行的。

3、启动代码

启动代码包含配置矢量表、初始化内存、将引导加载程序从闪存复制到 SRAM 以及从 SRAM 执行所需的最小代码集。由于一些特定于工具链的结构用于指示代码、数据和 bss 段在内存中的位置，因此每个受支持的工具链都有自己的独立文件来实现启动代码。启动代码包含在以下文件中：

bl_startup_rmdk.S　（Keil RV-MDK）
bl_startup_ccs.s　（德州仪器代码编辑器工作室）

每个工具链的启动代码附带有链接器脚本，用于将矢量表、代码段、数据段初始化器和数据段放置在内存中的适当位置。脚本位于以下文件中：
bl_link.sct　（Keil RV-MDK）
bl_link_ccs.cmd　（TI Code Composer Studio）

引导加载程序的代码及其相应的链接器脚本使用完全存在于 SRAM 中。这意味着代码和只读数据的加载地址与执行地址不同。这个内存映射允许引导加载程序更新自己，因为它实际上只是从 SRAM 运行的。SRAM 的第一部分用作引导加载程序的复制空间，而其余部分保留用于引导加载程序堆栈和读/写数据。一旦引导加载程序调用应用程序，所有 SRAM 都可以由应用程序使用。

Cortex-M4 微处理器的矢量表包含四个必需的条目：初始堆栈指针、重置处理程序地址、NMI 处理程序地址和硬故障处理程序地址。重置后，处理器加载初始堆栈指针，然后开始执行重置处理程序。由于 NMI 或硬故障随时可能发生，因此需要初始堆栈指针；由于处理器自动将八个项目推送到堆栈上，因此堆栈需要接受这些中断。

Vectors 数组包含引导加载程序的矢量表，该矢量表的大小根据添加的自动波特功能或 USB DFU 支持而变化。这些选项需要额外的中断处理程序来扩展矢量表以填充相关条目。由于引导加载程序是从 SRAM 而不是从闪存执行的，因此使用特定于工具链的构造来向链接器提供此阵列位于 0x2000.0000 的提示。

IntDefaultHandler 函数包含默认的故障处理程序。这是一个简单的无限循环，如果发生任何意外故障，可以有效地停止应用程序。因此，应用程序状态被保留以供调试器检查。如果需要，定制的引导加载程序可以通过向 Vectors 数组添加适当的处理程序来提供自己的处理程序。

重置后，启动代码将引导加载程序从闪存复制到 SRAM，分支到 SRAM 中引导加载程序的副本，并通过调用 CheckForceUpdate（）检查是否应执行应用程序更新。如果不需要更新，则调用应用程序。否则，调用的函数

将基于引导加载程序的操作模式。对于 UART、SSI 和 I2C，通过调用 ConfigureDevice（）初始化微控制器，然后引导加载调用串行控制循环 Updater（）。

应用程序更新的检查（在 CheckForceUpdate（）中）包括检查应用程序区域的开始，以及检查 GPIO 引脚的状态。如果第一个位置是有效的堆栈指针（也就是说，它位于 SRAM 中，值为 0x2xxx.xxxx），而第二个位置是无效的重置处理程序地址（也就是，它位于闪存中，值为 0x000x.xxxx，其中值为奇数），则假定应用程序是有效的。如果这些测试中的任何一个失败，那么应用程序将被认为无效，并强制进行更新。可以使用 bl_config.h 头文件中的 ENABLE_UPDATE_check 启用 GPIO 引脚检查，在这种情况下，可以通过更改 GPIO 引脚的状态（例如，使用按钮）强制进行更新。如果应用程序有效且 GPIO 引脚未请求更新，则调用该应用程序。否则，将通过进入引导加载程序的主循环来启动更新。

此外，应用程序可以调用引导加载程序，以便执行应用程序导向的更新。在这种情况下，引导加载程序假设应用程序已经配置了将用于更新的外围设备。这允许引导加载程序按原样使用外围设备来执行更新。引导加载程序还假设内核的中断也已启用，这意味着该应用程序在调用引导加载程序之前不应调用 IntMasterDisable（）。一旦应用程序调用了引导加载程序，引导加载程序就会将自身复制到 SRAM，分支到引导加载程序的 SRAM 副本，并通过调用 Updater（）（用于 UART、SSI 和 I2C）。矢量表的 SVCall 条目包含应用程序定向更新入口点的位置。

4、 序列更新

当通过串行端口（UART、SSI 或 I2C）执行更新时，ConfigureDevice（）用于配置所选串行端口，使其可以用于更新固件。然后，Updater（）处于一个无休止的循环中，接受命令并在请求时更新固件。来自这个主例程的所有传输都使用数据包处理程序函数（SendPacket（）、ReceivePacket（。更新完成后，可以通过向引导加载程序发出重置命令来重置引导加载程序。

当更新应用程序的请求通过并且定义了 FLASH_CODE_PROTECTION 时，引导加载程序首先擦除整个应用程序区域，然后接受新应用程序的二进制文件。这防止了闪存的部分擦除在新二进制文件被下载到微控制器之前暴露出任何代码。引导加载程序本身保留在适当的位置，这样它就不会引导部分擦除的程序。一旦所有的应用程序闪存区域都被成功擦除，引导加载程序就会继续下载新的二进制文件。如果未定义 FLASH_CODE_PROTECTION，则引导加载程序只会擦除足够的空间，以适应正在下载的新应用程序。

如果引导加载程序本身需要更新，则引导加载程序必须在闪存中进行替换。通过执行到地址 0x0000.0000 的下载来识别引导加载程序的更新。引导加载程序再次根据 FLASH_CODE_PROTECTION 的设置进行不同的操作。当定义 FLASH_CODE_PROTECTION 并且下载地址指示引导加载程序更新时，引导加载程序通过在擦除和替换自身之前擦除整个应用程序区域来保护微控制器上已经存在的任何应用程序代码。如果进程在任何时候中断，则旧的引导加载程序仍然存在于闪存中，并且不会引导部分应用程序，或者应用程序代码已经被擦除。如果没有定义 FLASH_CODE_PROTECTION，则引导加载程序只会擦除足够的空间来容纳自己的代码，并保持应用程序不变。

数据包处理

引导加载程序使用定义良好的数据包来确保与更新程序的可靠通信。数据包总是被通信设备确认或不被确认。数据包使用相同的格式来接收和发送数据包。这包括用于确认分组的成功或不成功接收的方法。虽然串行端口上的实际信令不同，但数据包格式与传输数据的方法无关。

引导加载程序使用 SendPacket（）函数将数据包发送到另一个设备。该功能封装了向另一设备发送有效分组所需的所有步骤，包括等待来自另一设备的确认或不确认。必须执行以下步骤才能成功发送数据包：

发送将发送到设备的数据包的大小。大小总是数据的大小+2。

发送数据缓冲区的校验和，以帮助确保命令的正确传输。校验和算法在提供的 checksum（）函数中实现，它只是数据字节的总和。

发送实际的数据字节。

等待来自设备的单字节确认，确认其正确接收到数据或检测到传输中的错误。

接收的数据包使用与发送的数据包相同的格式。引导加载程序使用 ReceivePacket（）函数来接收或等待来自其他设备的数据包。此功能不负责向其他设备确认或不确认数据包。这允许在发送回响应之前检查数据包的内容。必须执行以下步骤才能成功接收数据包：

等待从设备返回非零数据。这是重要的，因为设备可以在发送的和接收的数据包之间发送零字节。接收到的第一个非零字节将是正在接收的数据包的大小。

读取下一个字节，该字节将是数据包的校验和。

从设备中读取数据字节。将有数据包大小-在数据阶段发送 2 字节的数据。例如，如果数据包大小为 3，则只有 1 个字节的数据要被接收。

计算数据字节的校验和，并确保它是否与数据包中接收到的校验和匹配。

向设备发送确认或不确认，以指示分组的成功或不成功接收。

确认接收到数据包所需的步骤在 AckPacket()函数中实现。只要引导加载程序成功接收并验证了数据包，就会发送确认字节。

只要检测到发送的数据包有错误，通常是由于校验和错误或数据包中的数据格式不正确，就会发送未确认字节。这允许发送方重新发送先前的数据包。

传输层

引导加载程序支持通过 Tiva 微控制器上提供的 I2C、SSI 和 UART 端口进行更新。SSI 端口的优点是支持更高、更灵活的数据速率，但它也需要与微控制器的更多连接。UART 的缺点是具有稍低的并且可能不太灵活的速率。然而，UART 需要更少的引脚，并且可以使用任何标准 UART 连接轻松实现。I2C 接口还提供了一个标准接口，仅使用两根导线，并且可以以与 UART 和 SSI 接口相当的速度运行。

I2C 传输

I2C 处理函数是 I2CSend（）、I2CReceive（）和 I2CFlush（）函数。使用 I2C 端口所需的连接是以下引脚：I2CSCL 和 I2CSDA。与引导加载程序通信的设备必须作为 I2C 主机进行操作，并提供 I2CSCL 信号。I2CSDA 引脚是开漏极，可以由主 I2C 设备或从 I2C 设备驱动。

SSI传输

SSI处理函数为SSISend（）、SSIReceive（）和SSIFlush（）。使用SSI端口所需的连接是以下四个引脚：SSITx、SSIRx、SSIClk和SSIFss。与引导加载程序通信的设备负责驱动SSIRx、SSIClk和SSIFss引脚，而Tiva微控制器驱动SSITx引脚。SSI通信使用的格式为摩托罗拉格式，SPH设置为1，SPO设置为1（有关此格式的更多信息，请参阅Tiva系列数据表）。SSI接口具有硬件要求，该硬件要求将SSI时钟的最大速率限制为至多为运行引导加载程序的微控制器的频率的1/12。

UART传输

UART处理函数为UARTSend（）、UARTReceive（）和UARTFlush（）。使用UART端口所需的连接是以下两个引脚：U0Tx和U0Rx。与引导加载程序通信的设备负责驱动Tiva微控制器上的U0Rx引脚，而Tiva微处理器驱动U0Tx引脚。

虽然波特率是灵活的，但UART串行格式固定为8个数据位，无奇偶校验和一个停止位。如果启用了自动波特率功能，则用于通信的波特率可以由引导加载程序自动检测，也可以固定为与引导加载程序通信的设备支持的波特率。波特率的唯一要求是波特率不应超过运行引导加载程序的微控制器频率的1/32。这是任何Tiva微控制器上UART最大波特率的硬件要求。

当使用固定波特率时，必须指定连接到微控制器的晶体的频率。否则，引导加载程序将无法将UART配置为以请求的波特率运行。

引导加载程序提供了一种自动检测用于与之通信的波特率的方法。这种自动波特率检测在UARTAutoBaud（）函数中实现。自动波特率功能尝试与更新程序应用程序同步，并指示它是成功检测到波特率，还是未能正确检测到波特率。如果第一次调用失败，引导加载程序可以多次调用UARTAutoBaud（）来尝试重试同步。在提供的示例引导加载程序中，当启用自动波特功能时，引导加载程序将永远等待来自主机的有效同步模式。为了利用自动波特率功能，UART端口必须检测到足够的边缘以确定波特率的比率。这应该通过传输0x55的两个数据包来处理，以便为自动波特率功能生成足够数量的边缘来确定波特率。

ARM 实验（Cortex-M4）

实验五　图像传输及文字识别实验

一. 实验目的

熟悉 OMRON 工业相机 STC-SCE132POE 的使用方法，能够自动触发拍摄一张图像，通过相机传送到 S800 板，以 S800 板为中介层，实现图像信息与 PC 机的上传以及在 S800 板上实现图像包含信息的自动识别。掌握大容量数据的有线传送与简单的图像文字识别方法的实现。

二. 实验要求

1. 实验程序 5.1，实现功能如下：
   相机位置固定情况下，放入一张 A5 纸，纸上预先打印 8 个数字及字母，能够自动识别纸张的放入，并启动一帧抓取拍摄。
   抓拍完成后，通过传输线与 S800 板进行一帧数据的传输。暂存在 S800 的内存中。要求传输带 CRC 校验，上传后在 PC 端进行 CRC 校验。确保传输过程中的数据完整性。
   分别采用 115200bps 与 38400bps，计算两者传输的时间。
2. 实验程序 5.2，实现功能如下：
   在 1 的基础上，图片已经暂存在 S800 内存中。并且向 PC 端传送完成
   S800 板与 PC 机通过有线 USB 连接，以虚拟串口形式向 PC 机传送此帧数据。完成后，在 PC 机上调用图形识别程序，识别此文字数字。
   在 S800 板上对内存中图片进行文字识别，结果与 PC 机结果进行对比。
   此部分通讯协议采用自由定义，需要实现上传下载，能够完成两端不特定长度文字数据的对比。
3. 实验程序 5.3，扩展部分。实现功能如下：

在 S800 板的 8 位动态数码管上，接收 PC 机传来的不定长数据，并显示在数码管上，请以 10 帧/S 的速率检测此数码管，并将显示的数据拍摄并识别出来。

三、实验框图（略）

四、实验结果

应能现场演示实验的效果。

五. 讨论

1. CRC16 位校验码能保证几位传输错误的识别。
2. 图像识别算法中如何去除光照度与杂质的阀值。
3. 调整 S800 板的数码管的动态刷新率，从 30HZ 一直到 10HZ，会出现肉眼可见的闪

烁感，如何在此种情况下保证图片拍摄与识别的完整性。