本科实验指导书

实验名称： 嵌入式系统

开课学院： 计算机学院

国防科技大学教务处制

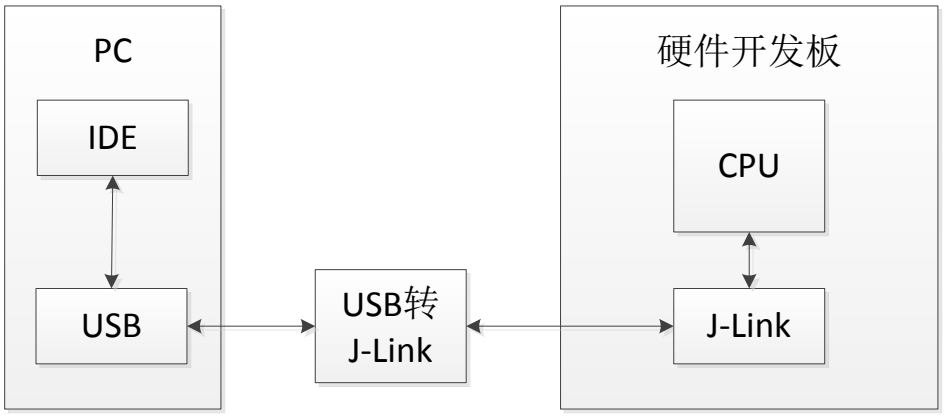
目　录

实验 1 测量 C 语言加法乘法操作执行时间

- 实验目的

  1 能够正确使用 Keil MDK 集成开发环境进行程序开发；
  2 能够正确描述 R0~R15 寄存器的含义；
  3 能够正确使用虚拟逻辑分析仪测量 C 语言加法、乘法（整数、浮点）操作的执行时间；
  4 能够正确使用 Keil MDK 的调试工具；
  5 能够正确下载执行代码。

- 实验框架

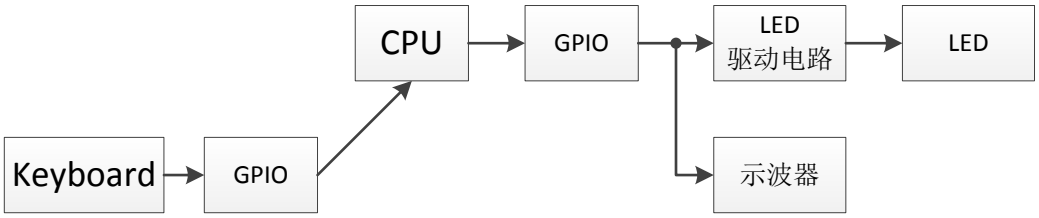


- 实验内容
  - 使用 MDK 创建程序；
  - 构建程序，使用“Simulator”方式调试程序
  - 用虚拟逻辑分析仪测量 C 语言加法、乘法（整数、浮点）的执行时间；
  - 使用硬件开发板（Port103Z）；
  - 构建程序，下载执行代码并运行，检查下载器、开发板是否工作正常。

- 符号说明
  - uC：微控制器 MCU
  - PC：微型计算机
  - IDE：集成开发环境
  - J-Link：仿真器（代码下载、调试）
  - USB：通用串行总线
  - LED：发光二极管
  - IMU：惯性测量组件（加速度计、陀螺仪、磁力计）
  - Sonar：超声波

实验 2 Keyboard+LED（GPIO 接口）

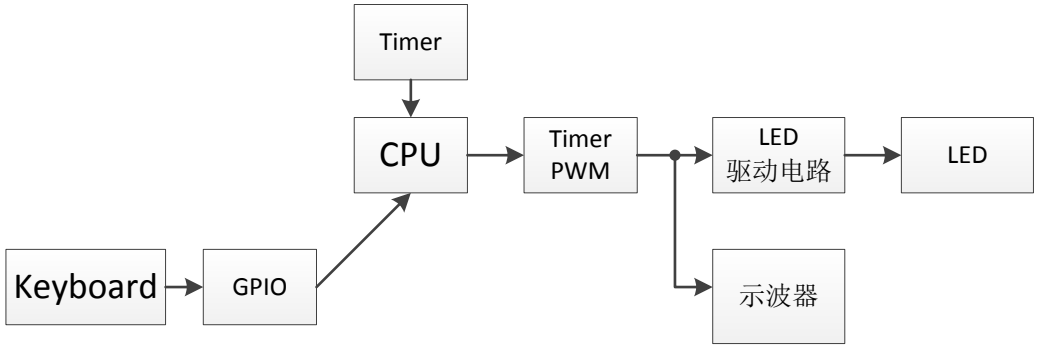
- 实验框架



- 实验内容
  - 通过 GPIO 控制四个 LED 进行四灯闪烁，闪烁时间间隔 2 秒，**软件延时**，用示波器（或虚拟逻辑分析仪）测量延时时间；
  - 由按键通过 GPIO 输入，选择 LED 闪烁的方式；
  - **指令模拟器调试。**

实验 3 Keyboard+LED（Timer+GPIO）

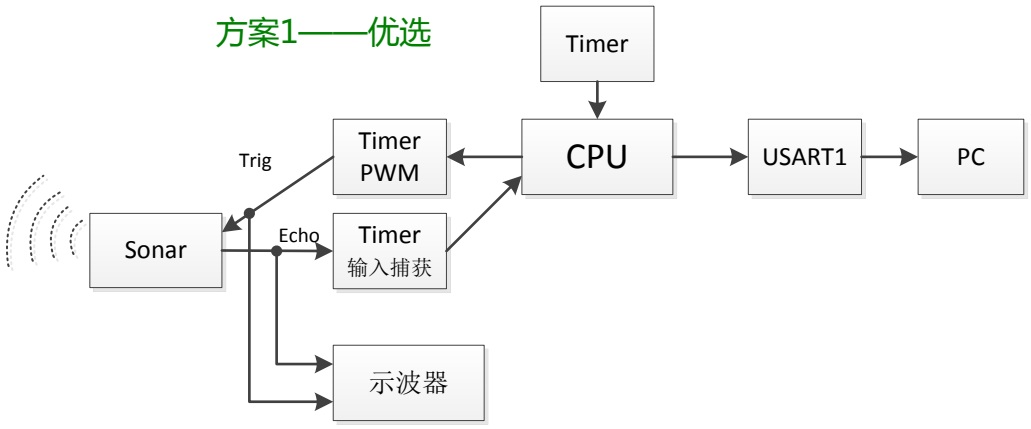
- 实验框架

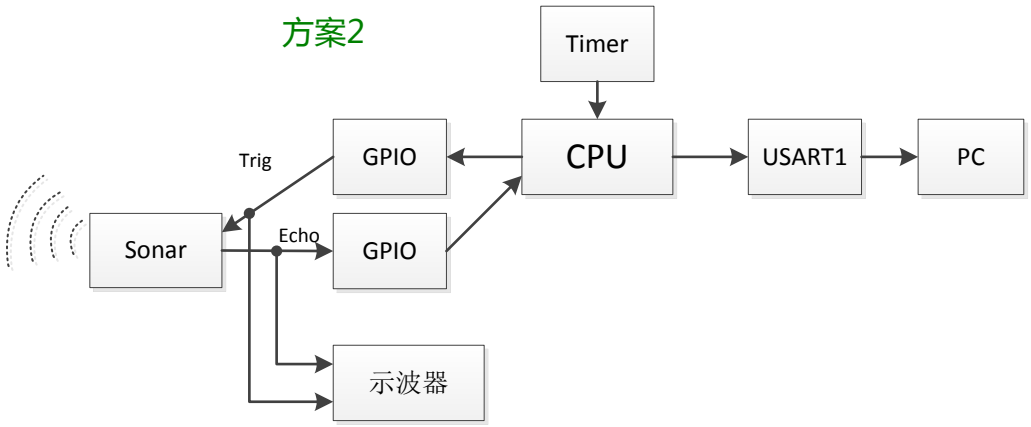


- 实验内容
  - 通过 Timer PWM 控制四个 LED 进行渐亮渐灭（fade in / fade out）实验，渐亮渐灭时间间隔 2 秒，**定时器延时**，用示波器（或虚拟逻辑分析仪）测量延时时间。
  - 由按键通过 GPIO 输入，选择哪一个 LED 进行渐亮渐灭；
  - **在系统调试（ISD，直接硬件调试）。**

实验 4 Sonar 测距（UART+Timer+GPIO）

- 实验框架

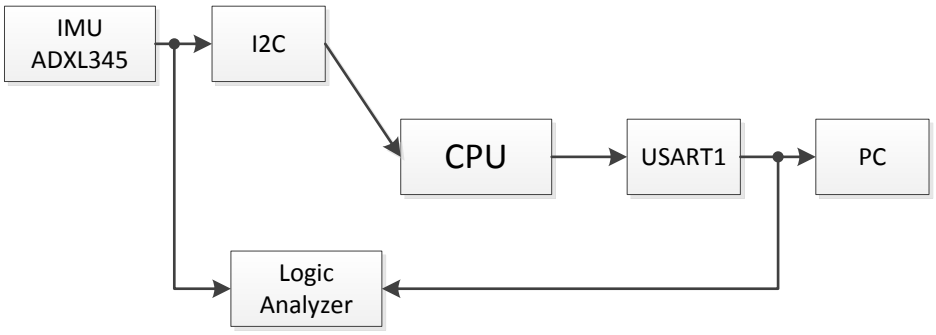




- 实验内容
  - 方案 1：
    - 定时（采样周期为 100ms）通过 Timer PWM 输出通道产生触发脉冲 Trig，触发声纳发出超声波，采样周期 100ms 由定时器获得；
    - 通过 Timer **输入捕获通道**获取 Sonar 传感器 Echo 信号的宽度（上升沿引发中断，记录捕获计数值，然后下降沿引发中断，再次记录捕获计数值，它们的差值就是脉冲宽度），然后计算距离；
    - 通过 USART1（printf）把距离信息发送到 PC 机显示。
  - 方案 2：
    - 定时（采样周期为 100ms）通过 GPIO 输出通道产生触发脉冲 Trig，触发声纳发出超声波，采样周期 100ms 由定时器获得；
    - 通过 GPIO 输入通道获取 Sonar 传感器 Echo 信号的宽度，然后计算距离；
    - 通过 USART1（printf）把距离信息发送到 PC 机显示。

实验 5 IMU 传感器测量（I2C+UART）

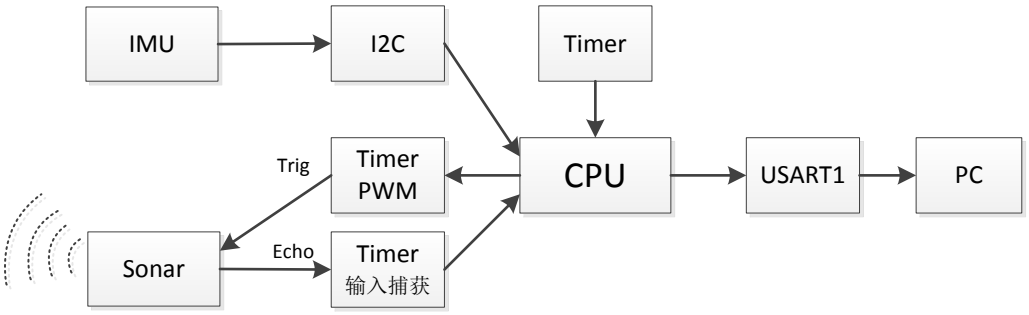
- 实验框架

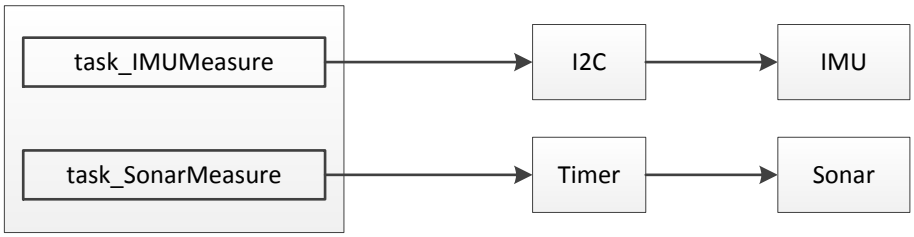


- 实验内容
  - 通过 I2C 总线读取 IMU/ADXL345 传感器的数据，送到 CPU，然后通过 USART1 送到 PC 机显示；
  - 逻辑分析仪解释 I2C、UART 协议；
  - **指令模拟器调试；**
  - **在系统调试（ISD，直接硬件调试）。**

实验 6 基于前后台系统的多传感器测量

- 实验框架





前后台系统

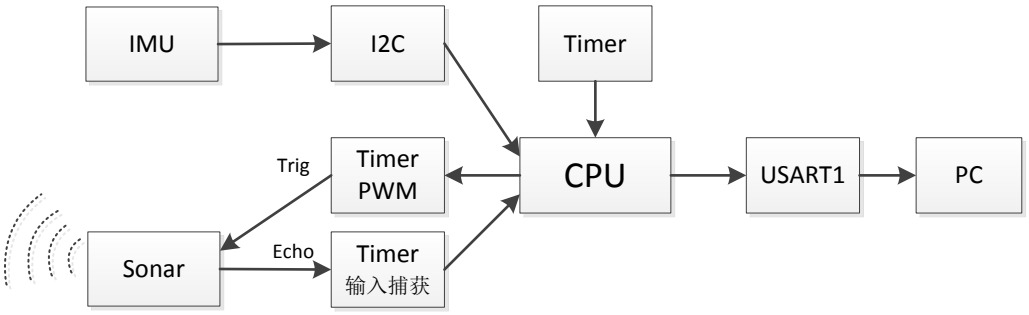
- 实验内容
  - IMU：定时（采样周期为 30ms）通过 I2C 总线读取 IMU 传感器的数据，送到 CPU，然后通过 USART1 送到 PC 机显示。
  - Sonar 方案 1：定时（采样周期为 100ms）通过定时器 PWM 输出通道产生触发脉冲 Trig，触发声纳发出超声波，采样周期 100ms 由定时器获得；通过定时器**输入捕获通道**获取 Sonar 传感器 Echo 信号的宽度（上升沿引发中断，记录捕获计数值，然后下降沿引发中断，再次记录捕获计数值，它们的差值就是脉冲宽度），然后计算距离；通过 USART1（printf）把距离信息发送到 PC 机显示。
  - Sonar 方案 2：定时（采样周期为 100ms）通过 GPIO 输出通道产生触发脉冲 Trig，触发声纳发出超声波，采样周期 100ms 由定时器获得；通过 GPIO 输入通道获取 Sonar 传感器 Echo 信号的宽度，然后计算距离；通过 USART1（printf）把距离信息发送到 PC 机显示。

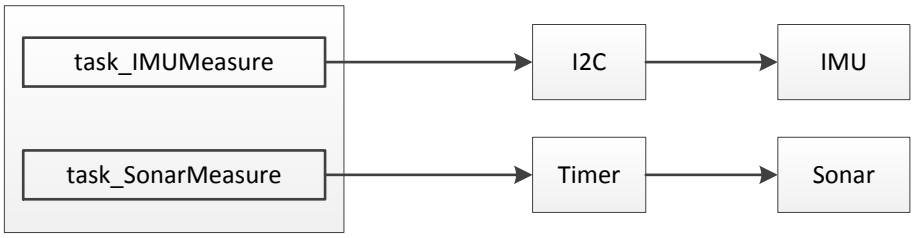
- 实验要求
  - 采用前后台系统编程规范；
  - IMU 测量（采样）周期为 30ms；
  - Sonar 测量（采样）周期为 100ms；
  - 两个任务：任务 1 为 IMU 测量，任务 2 为 Sonar 测量。

- 前后台编程概要
  - 全局变量：记录事件的一组中间变量
  - 前台：处理事件，并通过全局变量记录事件-中断
  - 调度器：主程序，死循环，轮询结构，检查全局变量
  - 后台：一个函数，执行事件对应的功能

实验 7 基于 uC/OS-II 的多传感器测量

- 实验框架





uC/OS-II

- 实验内容
  - uC/OS-II 实时操作系统定制（见实验 A1）。
  - uC/OS-II 实时操作系统移植（见实验 A1）。
  - IMU：定时（采样周期为 30ms）通过 I2C 总线读取 IMU 传感器的数据，送到 CPU，然后通过 USART1 送到 PC 机显示。
  - Sonar 方案 1：定时（采样周期为 100ms）通过定时器 PWM 输出通道产生触发脉冲 Trig，触发声纳发出超声波，采样周期 100ms 由定时器获得；通过定时器**输入捕获通道**获取 Sonar 传感器 Echo 信号的宽度（上升沿引发中断，记录捕获计数值，然后下降沿引发中断，再次记录捕获计数值，它们的差值就是脉冲宽度），然后计算距离；通过 USART1（printf）把距离信息发送到 PC 机显示。
  - Sonar 方案 2：定时（采样周期为 100ms）通过 GPIO 输出通道产生触发脉冲 Trig，触发声纳发出超声波，采样周期 100ms 由定时器获得；通过 GPIO 输入通道获取 Sonar 传感器 Echo 信号的宽度，然后计算距离；通过 USART1（printf）把距离信息发送到 PC 机显示。

- 实验要求
  - 采用实时操作系统 uC/OS-II 编程；

- IMU 测量（采样）周期为 30ms ；
- Sonar 测量（采样）周期为 100ms；
- 两个任务：任务 1 为 IMU 测量，任务 2 为 Sonar 测量。

- uC/OS-II 编程概要
  - main 函数：OS 初始化、创建启动任务、OS 启动（不再返回）
  - 启动任务：创建标志组、创建任务、BSP 初始化、开中断
  - 一般任务：死循环（while(1)），等待事件（OSFlagPend），用户代码
  - 空闲任务：一个死循环，无等待事件
  - ISR：OSIntEnter、OSIntExit、发送事件（OSFlagPost）
  - BSP 初始化：PendSV 中断（优先级最低）、SysTick 中断初始化

实验 A1 μC/OS-II 操作系统的定制和移植

一、实验目的

1、理解 μC/OS-II 实时操作系统的工作原理；
2、对 μC/OS-II 实时操作系统进行定制、移植；
3、理解实时操作系统的任务与优先级等概念；
4、编写基于 μC/OS-II 实时操作系统的中断程序。

二、实验内容

基于 STM32F103 开发板，设计一个基于 μC/OS-II 实时操作系统的双灯闪烁系统（或者跑马灯系统）。所以本实验有两个任务：一个启动任务、一个双灯闪烁（跑马灯）任务。

要求充分体现实时操作系统的编程思想，理解实时操作系统与查询方式、中断方式、前后台系统编程的区别。

CPU 时钟频率设为 72MHz（开发板外接 8MHz，通过锁相环 PLL 倍频）。

三、实验原理

μC/OS-II 实时操作系统的大部分源代码是用 C 语言写成的，但是仍需要用 C 语言和汇编语言完成一些与处理器相关的代码。例如 μC/OS-II 在初始化处理器寄存器时只能通过汇编语言来实现。因为 μC/OS-II 在设计的时候就已经充分考虑了可移植性，所以，μC/OS-II 的移植还是比较容易的。

要使 μC/OS-II 可以正常工作，处理器必须满足如下要求：

（1）处理器的 C 编译器能产生可重入代码。

可重入的代码指的是一段代码（比如：一个函数）可以被多个任务同时调用，而不必担心会破坏数据。也就是说，可重入型函数在任何时候都可以被中断执行，过一段时间以后又可以继续运行，而不会因为在函数中断的时候被其他的任务重新调用，影响函数中的数据。代码的可重入性是保证完成多任务的基础，除了在 C 程序中使用局部变量以外，还要 C 编译器的支持。

（2） 在程序中可以打开或关断中断。

在 μC/OS-II 中，可以通过 OS_ENTER_CRITICAL()或者 OS_EXIT_CRITICAL()宏来控制系统关闭或者打开中断，这需要处理器的支持。在 CORTEX-M3 的处理器上，可以设置相应的寄存器来关闭或者打开系统的所有中断。

（3）处理器支持中断，并能产生定时中断（通常在 10Hz~1000Hz 之间）。

μC/OS-II 是通过处理器产生的定时器的中断来实现多任务之间的调度的。在 CORTEX-M3 的处理器上可以产生定时器中断。

（4）处理器支持能够容纳一定量数据（可能是几千字节）的堆栈。

（5）处理器有将堆栈指针和其他 CPU 寄存器读出和存储到堆栈或内存中的指令。

（一）、移植

移植工作包括以下几个内容：

（1）设置一个常量的值（OS_CPU.H）；

（2）声明十个数据类型（OS_CPU.H）；

（3）声明三个宏（OS_CPU.H）；

（4）用 C 语言编写几个函数（OS_CPU_C.C）；

μC/OS-II 进行任务调度的时候，会把当前任务的 CPU 寄存器存放到此任务的堆栈中，然后，再从另一个任务的堆栈中恢复原来的工作寄存器，继续运行另一个任务。所以，寄存器的入栈和出栈是 μC/OS-II 多任务调度的基础。

在移植过程中，INCLUDES.H 使得用户项目中的每个.C 文件不用分别去考虑它实际上需要那些头文件。使用 INCLUDES.H 的唯一缺点是，它可能会包括一些实际不相关的头文件。这意味着每个文件的编译时间可能会增加。但由于它增强了代码的可移植性，所以我们还是决定使用这一方法。用户可以通过编辑 INCLUDES.H 来增加自己的头文件，但用户的头文件必须添加在头文件列表的最后。

1．设置 OS_CPU.H 中与处理器和编译器相关的代码

```
/*****************************************************
* 与编译器相关的数据类型
*****************************************************/
typedef unsigned char   BOOLEAN;
typedef unsigned char   INT8U;
typedef signed char     INT8S;
typedef unsigned short  INT16U;      // 某些编译器中 int 是 32 位
typedef signed short    INT16S;
typedef unsigned long   INT32U;
typedef signed long     INT32S;
typedef float           FP32;
typedef double          FP64;
typedef unsigned long   OS_STK;      // 堆栈宽度为 32 位，即字对齐方式
typedef unsigned long   OS_CPU_SR; // 定义 CPU 状态寄存器（PSR）
/*****************************************************
* 定义开关中断的方法
*****************************************************/
//#define OS_CRITICAL_METHOD   1    // 使用方式 1 保护临界代码
// 不能使用方法 1，返回时不恰当地开了中断
//#define OS_CRITICAL_METHOD   2    // 使用方式 2 保护临界代码
// 也不能使用方法 2，按照 ARM 规范 ATPCS 要求，进入子程序之前和退出子程序之后的栈指针要相同
#define OS_CRITICAL_METHOD          3 // 使用方式 3 保护临界代码
#define   OS_ENTER_CRITICAL()
                        (cpu_sr = OS_CPU_SR_Save ())  // 关中断
#define   OS_EXIT_CRITICAL()
                        (OS_CPU_SR_Restore (cpu_sr))  // 开中断
/*****************************************************
* 与 ARM 处理器相关的代码
```

```
*********************************************************/
/* 设置堆栈的增长方向 */
#define OS_STK_GROWTH    1        // 定义堆栈生长方向为向下生长
#define OS_TASK_SW    OSIntCtxSw // 宏定义，用于任务切换
```

2．用 C 语言编写操作系统相关的函数（OS_CPU_C.C）

```
/*********************************************************
** Function name:           OSStartHighRdy
** Descriptions:            uC/OS-II 启动使用 OSStartHighRdy 运行第一个任务
** input parameters:        none
** output parameters:       none
** Returned value:          none
*********************************************************/
__asm void OSStartHighRdy (void)
{
    IMPORT   OSRunning
    IMPORT   OSTCBCur
    IMPORT   OSTCBHighRdy
    IMPORT   OSTaskSwHook

    preserve8                       // 8 字节边界对齐
    BL       OSTaskSwHook    // 调用用户的 Hook 函数，空函数
    LDR      R4, =OSRunning  // 将 OSRunning 置 1，声明多任务 OS 开始运行
    MOVS     R5, #1
    STRB     R5, [R4]
    LDR      R4, =OSTCBHighRdy // 伪指令，取得存储 OSTCBHighRdy 的地址
    LDR      R4, [R4]               // 得到最高优先级任务的任务堆栈地址
    LDR      R6, [R4]       // 切换到新任务的堆栈 SP = OSTCBHighRdy->OSTCBStkPtr
    MSR      MSP, R6                // 恢复主栈指针
    POP      {R4-R7}                // 恢复 R4~R7
    POP {R0-R3}                     // 恢复 R8~R11
    MOV      R8, R0
    MOV      R9, R1
    MOV      R10, R2
    MOV      R11, R3

    ADD      SP,SP,#0x10                // 调整栈指针，恢复 PSR PC LR R12
    POP {R0-R3}
    MOV      R12,R0
    MOV      LR,R1
    PUSH     {R2}                   // PC 进栈
    MSR      PSR, R3

    SUBSP,SP,#0x1C                  // 调整栈指针，恢复 R0~R3
```

```
    POP {R0-R3}

    ADD     SP,SP,#0x0C             // 调整栈指针
    POP     {PC}                    // PC 出栈，进入高优先级任务
    ALIGN                           // 字对齐
}

/*********************************************************
** Function name:           OSIntCtxSw
** Descriptions:            调度函数
** input parameters:        none
** output parameters:       none
** Returned value:          none
*********************************************************/
void OSIntCtxSw (void)
{
    //NVIC_SetPendingIRQ(PendSV_IRQn);    // 不能使用，要求 IRQn 非负
    SCB->ICSR = SCB_ICSR_PENDSVSET_Msk;  // 引起 PendSV 中断，引起任务切换
}

/*********************************************************
** Function name:           PendSV_Handler
** Descriptions:            uC/OS-II 任务调度函数
** input parameters:        none
** output parameters:       none
** Returned value:          none
*********************************************************/
__asm void PendSV_Handler(void)
{
    IMPORT  OSTCBCur
    IMPORT  OSTCBHighRdy
    IMPORT  OSPrioCur
    IMPORT  OSPrioHighRdy
    IMPORT  OSTaskSwHook

    preserve8
// 进入异常，处理器自动依次保存{R0-R3,R12,LR,PC,xPSR}，LR = 0xFFFFFFF9
    CPSID   I                       // 关中断
    MOV     R12, LR                 // 关键，保存 LR 到 R12，以备异常返回

    MOV     R0, R8
    MOV     R1, R9
    MOV     R2, R10
    MOV     R3, R11
    PUSH    {R0-R3}                 // 自己编程保存 R8-R11
```

```
    PUSH    {R4-R7}                 // 自己编程保存 R4-R7

    LDR     R4, =OSTCBCur           // 得到当前 TCB 块的地址，传给 R4
    LDR     R5, [R4] // 将 OSTCBCur 中的值传给 R5，注意 OSTCBCur 存的是指针
    MRS     R6, MSP
    STR R6, [R5]        // 将当前任务的 SP 传到 OSTCBCur 存的指针中去

    BL      OSTaskSwHook            // 调用 Hook 函数，此为空函数

    LDR     R4, =OSPrioCur
    LDR     R5, =OSPrioHighRdy
    LDRB    R6, [R5]
    STRB    R6, [R4]                // OSPrioCur = OSPrioHighRdy

    LDR     R4, =OSTCBCur
    LDR     R5, =OSTCBHighRdy
    LDR     R6, [R5]
    STR     R6, [R4]                // OSTCBCur = OSTCBHighRdy

    LDR     R6, [R6]                // 从 R6 中取得要恢复的任务的
                                    // 栈顶指针 SP = OSTCBHighRdy->OSTCBStkPtr
    MSR     MSP, R6                 // 恢复主栈指针
    POP     {R4-R7}                 // 恢复 R4~R7
    POP {R0-R3}                     // 恢复 R8~R11
    MOV     R8, R0
    MOV     R9, R1
    MOV     R10, R2
    MOV     R11, R3

    CPSIE   I                               // 开中断
    BX      R12             // 异常返回，处理器自动依次恢复{R0-R3,R12,LR,PC,xPSR}
    ALIGN                                   // 字对齐
}

/*********************************************************
** Function name:       SysTick_Handler
** Descriptions:        系统节拍中断
** input parameters:    无
** output parameters:   无
** Returned value:      无
*********************************************************/
void SysTick_Handler (void)
{
    OSIntEnter();
    OSTimeTick();                   // 系统节拍处理
    OSIntExit();
```

```
}

//*********************************************************************
//                                  CRITICAL SECTION METHOD 3 FUNCTIONS
// Description: Disable/Enable interrupts by preserving the state of interrupts.
//              Generally speaking you would store the state of the interrupt
//              disable flag in the local variable 'cpu_sr' and then disable
//              interrupts.  'cpu_sr' is allocated in all of uC/OS-II's functions
//              that need to disable interrupts.  You would restore the interrupt
//              disable state by copying back 'cpu_sr' into the CPU's status register.
//
// Prototypes :  OS_CPU_SR  OS_CPU_SR_Save(void);
//               void       OS_CPU_SR_Restore(OS_CPU_SR cpu_sr);
//
// Note(s)    : 1) These functions are used in general like this:
//
//              void Task (void *p_arg)
//              {
// #if OS_CRITICAL_METHOD == 3 /* Allocate storage for CPU status register */
//                  OS_CPU_SR  cpu_sr;
//              #endif
//              OS_ENTER_CRITICAL();       /* cpu_sr = OS_CPU_SaveSR();  */
//                                :
//              OS_EXIT_CRITICAL();        /* OS_CPU_RestoreSR(cpu_sr);  */
//                                :
//               }
//*********************************************************************
__asm OS_CPU_SR OS_CPU_SR_Save(void)
{
    preserve8
    MRS     R0, PRIMASK     //; Set priority interrupt mask to mask all (except faults)
    CPSID   I               // 关中断
    BX  LR
}

__asm void OS_CPU_SR_Restore(OS_CPU_SR cpu_sr)
{
    preserve8
    MSR     PRIMASK, R0     // 恢复原有中断屏蔽情况
    BX  LR
}


/*********************************************************
** Function name:           OSTaskStkInit
** Descriptions:  满递减栈，任务栈初始化代码，本函数调用失败会使系统崩溃
** input parameters:         task:  任务开始执行的地址
```

```
**                          p_arg:  传递给任务的参数
**                          ptos:   任务的栈开始位置
**                          opt:    附加参数,当前版本对于本函数无用,
** output parameters:       none
** Returned value:          新栈位置
*********************************************************/
OS_STK *OSTaskStkInit (void (*task)(void *pd), void *p_arg, OS_STK *ptos, INT16U opt)
{    OS_STK *pstk;
     opt     = opt;                          // 避免编译器警告
     pstk    = ptos;                         // 获取堆栈指针

     // 如果创建任务使用&TaskStk[TASK_STK_SIZE - 1]，则第一个 stk 不用自减
     //*(stk) = (OS_STK)task;                 // xPSR
     // 如果创建任务使用&TaskStk[TASK_STK_SIZE]，则第一个 stk 要自减（满递减栈）
     *--pstk = (INT32U)0x01000000;           //  xPSR
     *--pstk = (INT32U)task;                 //  PC，任务入口
     *--pstk = (INT32U)0;                    //  LR
     *--pstk = (INT32U)0;                    //  R12
     *--pstk = (INT32U)0;                    //  R3
     *--pstk = (INT32U)0;                    //  R2
     *--pstk = (INT32U)0;                    //  R1
     *--pstk = (INT32U)p_arg;                //  R0  任务参数
     *--pstk = (INT32U)0;                    //  R11
     *--pstk = (INT32U)0;                    //  R10
     *--pstk = (INT32U)0;                    //  R9
     *--pstk = (INT32U)0;                    //  R8
     *--pstk = (INT32U)0;                    //  R7
     *--pstk = (INT32U)0;                    //  R6
     *--pstk = (INT32U)0;                    //  R5
     *--pstk = (INT32U)0;                    //  R4
     return (pstk);
}
```

后几个函数为钩子函数，基本不加代码。

```
// uCOSII OS 钩子函数
#if OS_VERSION > 203
void OSInitHookBegin (void)
{}
#endif
#if OS_VERSION > 203
void OSInitHookEnd (void)
{}
#endif
void OSTaskCreateHook (OS_TCB *ptcb)
{
     ptcb = ptcb;                            /*  Prevent compiler warning    */
```

```
}
void OSTaskDelHook (OS_TCB *ptcb)
{
    ptcb = ptcb;                                /*    Prevent compiler warning       */
}
void OSTaskSwHook (void)
{}
void OSTaskStatHook (void)
{}

#if OS_VERSION > 203
void OSTCBInitHook (OS_TCB *ptcb)
{
    ptcb = ptcb;                            /*    Prevent Compiler warning       */
}
#endif

void OSTimeTickHook (void)
{}
#if OS_VERSION >= 251
void OSTaskIdleHook (void)
{}
```

完成了上述工作以后，μC/OS-II 就可以正常运行在 ARM 处理器上了。这样移植工作就完成了。

3．PendSV、Systick 中断初始化（BSP_Init）

```
void BSP_Init(void)
{
    __set_PRIMASK(1);    // 关全局中断，只剩下不可屏蔽中断 NMI 和硬故障异常

    // 选择优先级组 2（2 位抢占优先级，2 位响应优先级）
    NVIC_SetPriorityGrouping(NVIC_PriorityGroup_2);

    // 设置 PendSV 中断优先级，中断已经使能，优先级最低，用于任务切换
    NVIC_SetPriority(PendSV_IRQn, 15);          // 最低优先级 15

    // 设置 SysTick 中断优先级，使能中断，缺省优先级为 15，启动定时，节拍为 10ms
    SysTick_Config(SystemCoreClock / OS_TICKS_PER_SEC);
    NVIC_SetPriority(SysTick_IRQn, 1);          // 改写优先级为 1

    // 用户初始化代码
    GPIO_Config();                              // GPIO 初始化
    EXTI_Config();                              // EXTI 初始化
    ……
```

```
    __set_PRIMASK(0);                                   // 开全局中断
}
```

（二）、定制

然后编写应用程序——基于 μC/OS-II 实时操作系统编制程序，并根据应用程序定制操作系统，主要是文件“os_cfg.h”，如下所示。

```
/* ---------------------- MISCELLANEOUS ----------------------- */
#define OS_APP_HOOKS_EN 0 /* Application-defined hooks are called from the uC/OS-II hooks */
#define OS_ARG_CHK_EN 0 /* Enable (1) or Disable (0) argument checking*/
#define OS_CPU_HOOKS_EN 1 /* uC/OS-II hooks are found in the processor port files */

#define OS_DEBUG_EN 0 /* Enable(1) debug variables*/

#define OS_EVENT_NAME_SIZE 16 /* Determine the size of the name of a Sem, Mutex, Mbox*/

#define OS_LOWEST_PRIO 63 /* Defines the lowest priority that can be assigned ... */
/* ... MUST NEVER be higher than 254! */

#define OS_MAX_EVENTS 10    /* Max. number of event control blocks in your application    */
#define OS_MAX_FLAGS    10    /* Max. number of Event Flag Groups    in your application    */
#define OS_MAX_MEM_PART 5    /* Max. number of memory partitions*/
#define OS_MAX_QS    4    /* Max. number of queue control blocks in your application    */
#define OS_MAX_TASKS    20    /* Max. number of tasks in your application, MUST be >= 2 */

#define OS_SCHED_LOCK_EN    1    /* Include code for OSSchedLock() and OSSchedUnlock() */

#define OS_TICK_STEP_EN 1    /* Enable tick stepping feature for uC/OS-View*/
#define OS_TICKS_PER_SEC 100    /* Set the number of ticks in one second    */

/* ---------------------- TASK STACK SIZE ----------------------- */
#define OS_TASK_TMR_STK_SIZE 128    /* Timer    task stack size*/
#define OS_TASK_STAT_STK_SIZE 128 /* Statistics task stack size (# of OS_STK wide entries)*/
#define OS_TASK_IDLE_STK_SIZE 128    /* Idle task stack size (# of OS_STK wide entries)*/

/* ---------------------- TASK MANAGEMENT ----------------------- */
#define OS_TASK_CHANGE_PRIO_EN    0    /* Include code for OSTaskChangePrio()    */
#define OS_TASK_CREATE_EN 1    /* Include code for OSTaskCreate()*/
#define OS_TASK_CREATE_EXT_EN 0    /* Include code for OSTaskCreateExt() */
#define OS_TASK_DEL_EN 1    /* Include code for OSTaskDel()*/
#define OS_TASK_NAME_SIZE 16    /* Determine the size of a task name    */
#define OS_TASK_PROFILE_EN 1    /* Include variables in OS_TCB for profiling */
#define OS_TASK_QUERY_EN    1    /* Include code for OSTaskQuery() */
#define OS_TASK_STAT_EN 0    /* Enable (1) or Disable(0) the statistics task    */
#define OS_TASK_STAT_STK_CHK_EN 0    /* Check task stacks from statistic task    */
#define OS_TASK_SUSPEND_EN 0    /* Include code for OSTaskSuspend() */
#define OS_TASK_SW_HOOK_EN 1    /* Include code for OSTaskSwHook()*/

/* ------------------------ EVENT FLAGS ------------------------ */
#define OS_FLAG_EN 1    /* Enable (1) or Disable (0) code generation for EVENT FLAGS    */
#define OS_FLAG_ACCEPT_EN 1    /* Include code for OSFlagAccept()*/
#define OS_FLAG_DEL_EN 1    /* Include code for OSFlagDel()*/
#define OS_FLAG_NAME_SIZE 16    /* Determine the size of the name of an event flag group    */
#define OS_FLAGS_NBITS 16    /* Size in #bits of OS_FLAGS data type (8, 16 or 32)*/
#define OS_FLAG_QUERY_EN    1    /* Include code for OSFlagQuery() */
#define OS_FLAG_WAIT_CLR_EN 1    /* Include code for Wait on Clear EVENT FLAGS    */

 /* --------------------- MESSAGE MAILBOXES ---------------------- */
#define OS_MBOX_EN 0    /* Enable (1) or Disable (0) code generation for MAILBOXES    */
#define OS_MBOX_ACCEPT_EN 1    /* Include code for OSMboxAccept()*/
#define OS_MBOX_DEL_EN 1    /* Include code for OSMboxDel()*/
#define OS_MBOX_PEND_ABORT_EN 1    /* Include code for OSMboxPendAbort() */
#define OS_MBOX_POST_EN 1    /* Include code for OSMboxPost()    */
```

```
#define OS_MBOX_POST_OPT_EN 1   /* Include code for OSMboxPostOpt() */
#define OS_MBOX_QUERY_EN    1   /* Include code for OSMboxQuery() */


  /* ---------------------- MEMORY MANAGEMENT --------------------- */
#define OS_MEM_EN    0 /* Enable (1) or Disable (0) code generation for MEMORY MANAGER */
#define OS_MEM_NAME_SIZE 16   /* Determine the size of a memory partition name */
#define OS_MEM_QUERY_EN1   /* Include code for OSMemQuery()    */

  /* ---------------- MUTUAL EXCLUSION SEMAPHORES --------------- */
#define OS_MUTEX_EN    0   /* Enable (1) or Disable (0) code generation for MUTEX    */
#define OS_MUTEX_ACCEPT_EN 1   /* Include code for OSMutexAccept() */
#define OS_MUTEX_DEL_EN 1   /* Include code for OSMutexDel()    */
#define OS_MUTEX_QUERY_EN 1   /* Include code for OSMutexQuery()*/

  /* ---------------------- MESSAGE QUEUES ---------------------- */
#define OS_Q_EN 0   /* Enable (1) or Disable (0) code generation for QUEUES */
#define OS_Q_ACCEPT_EN 1   /* Include code for OSQAccept()*/
#define OS_Q_DEL_EN    1   /* Include code for OSQDel() */
#define OS_Q_FLUSH_EN    1   /* Include code for OSQFlush() */
#define OS_Q_PEND_ABORT_EN 1   /* Include code for OSQPendAbort()*/
#define OS_Q_POST_EN 1   /* Include code for OSQPost()    */
#define OS_Q_POST_FRONT_EN1   /* Include code for OSQPostFront()*/
#define OS_Q_POST_OPT_EN    1   /* Include code for OSQPostOpt()    */
#define OS_Q_QUERY_EN    1   /* Include code for OSQQuery() */

  /* ------------------------ SEMAPHORES ------------------------ */
#define OS_SEM_EN    1   /* Enable (1) or Disable (0) code generation for SEMAPHORES */
#define OS_SEM_ACCEPT_EN    1   /*    Include code for OSSemAccept()    */
#define OS_SEM_DEL_EN    1   /*    Include code for OSSemDel()    */
#define OS_SEM_PEND_ABORT_EN    1   /*    Include code for OSSemPendAbort() */
#define OS_SEM_QUERY_EN 1   /*    Include code for OSSemQuery()*/
#define OS_SEM_SET_EN    1   /*    Include code for OSSemSet()    */

  /* --------------------- TIME MANAGEMENT --------------------- */
#define OS_TIME_DLY_HMSM_EN 0   /* Include code for OSTimeDlyHMSM() */
#define OS_TIME_DLY_RESUME_EN 0   /* Include code for OSTimeDlyResume() */
#define OS_TIME_GET_SET_EN 0   /* Include code for OSTimeGet() and OSTimeSet()    */
#define OS_TIME_TICK_HOOK_EN    1   /* Include code for OSTimeTickHook()    */

  /* --------------------- TIMER MANAGEMENT --------------------- */
#define OS_TMR_EN    0   /* Enable (1) or Disable (0) code generation for TIMERS */
#define OS_TMR_CFG_MAX 16   /* Maximum number of timers    */
#define OS_TMR_CFG_NAME_SIZE 16   /* Determine the size of a timer name */
#define OS_TMR_CFG_WHEEL_SIZE 8   /* Size of timer wheel (#Spokes)    */
#define OS_TMR_CFG_TICKS_PER_SEC 10   /* Rate at which timer management task runs (Hz)
```

然后采用 Keil MDK-ARM 进行仿真并下载到开发板运行。

四、实验条件

1、STM32F103 开发板、串口电缆；

2、PC 机及 Windows 操作系统、Keil MDK-ARM 集成开发环境、仿真器下载调试。

五、实验步骤

1、移植、定制 μC/OS-II 实时操作系统；

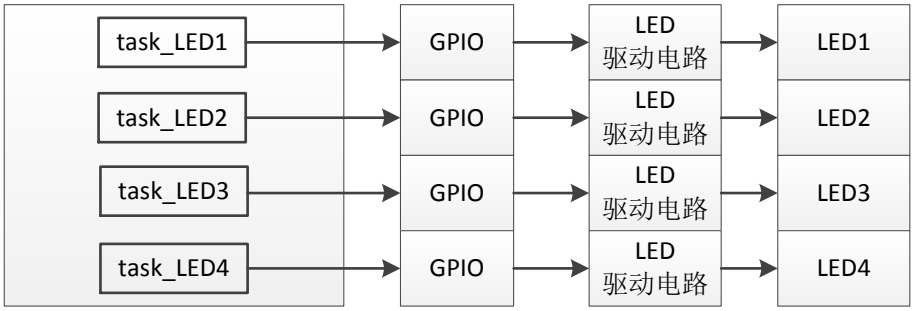
2、编写程序；

3、采用 Keil MDK-ARM 进行编译；

4、准备开发板运行环境；
5、下载到开发板进行检查。

实验 B1 基于 uC/OS-II 的四灯闪烁

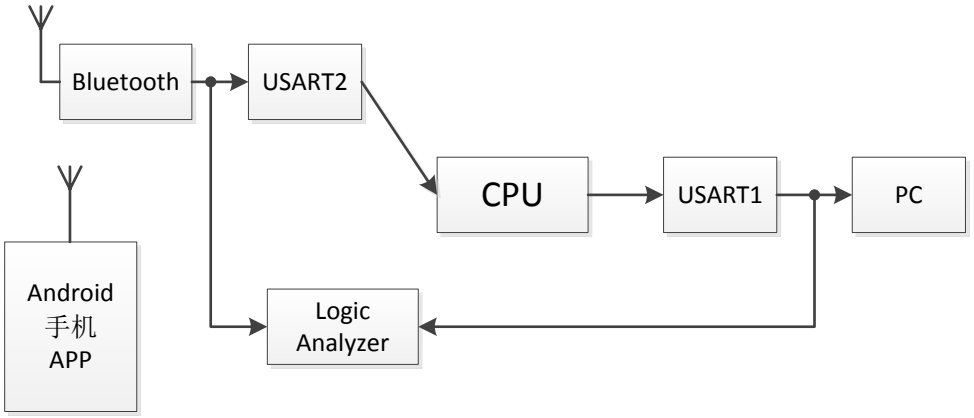
- 实验框架



- 实验内容
  - uC/OS-II 实时操作系统**定制**；
  - uC/OS-II 实时操作系统**移植**；
  - 通过 GPIO 控制 4 个 LED 闪烁，所以需要创建 4 个任务：task_LED1 控制 LED1 按照 2 秒时间间隔闪烁，task_LED2 控制 LED2 按照 3 秒时间间隔闪烁，task_LED3 控制 LED3 按照 5 秒时间间隔闪烁，task_LED4 控制 LED4 按照 7 秒时间间隔闪烁。

实验 B2 Bluetooth 通信（UART 接口）

- 实验框架

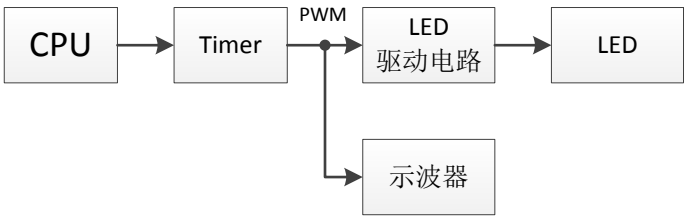


- 实验内容
  - 手机发送命令到蓝牙模块，蓝牙模块通过 USART2 送到 CPU，再通过 USART1 送到 PC 机显示；
  - 逻辑分析仪解释 UART 协议。

- 实验要求
  - 采用前后台系统或 uC/OS-II 编程规范。

实验 B3 LED 渐变闪烁（PWM 输出）

- 实验框架

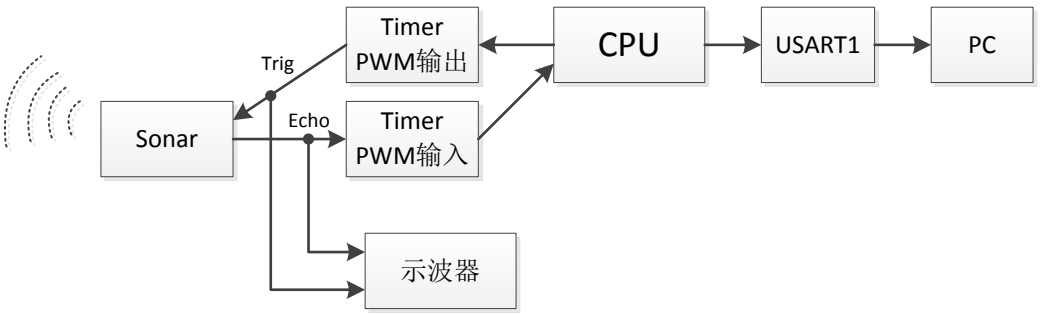


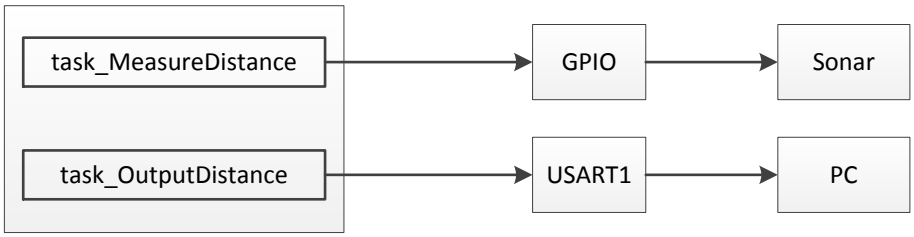
- 实验内容
  - 通过定时器 PWM 输出通道控制 LED 逐渐变亮，然后逐渐变暗（fade in / fade out）。

- 实验要求
  - 采用前后台系统或 uC/OS-II 编程规范。

实验 B4 基于 PWM 输入捕获的 Sonar 测量

- 实验框架





前后台系统 / uC/OS-II

- 实验内容
  - 通过定时器 PWM 输出通道产生触发脉冲 Trig，触发声纳发出超声波；
  - 通过定时器 **PWM 输入捕获通道**获取 Sonar 传感器 Echo 信号的宽度，然后计算距离；
  - 通过 USART（printf）把距离发送到 PC 机显示。

- 实验要求
  - 采用前后台系统或 uC/OS-II 编程规范；
  - 测量（采样）周期为 100ms；
  - 两个任务：任务 1 为超声波测量，任务 2 为串口输出。