

浙江大学 实验报告

专业: 海洋工程与技术

姓名: 王俊

学号: 3170100186

日期: 2022.3.22

课程名称: 嵌入式系统 指导老师: 司玉林 成绩: _____

实验名称: STM32 进阶 - μ C/OS 嵌入式实时操作系统 实验类型: _____ 同组学生姓名: _____

一、实验目的和要求 (必填)

二、实验内容和原理 (必填)

三、主要仪器设备 (必填)

四、操作方法和实验步骤

五、实验数据记录和处理

六、实验结果与分析 (必填)

七、讨论、心得

一、实验目的和要求

1. 阅读《嵌入式实时操作系统 μ COS-II 经典实例》第 3 章和第 4 章, 学习 uC/OS II 实时操作系统与 uC/GUI 图形用户界面用相关基础知识。

2. 阅读《奋斗 STM32 开发板基于 ucos2.86a ucGUI3.90 的 LED 闪烁例程》, 运行并理解 “STM32 奋斗板-LED 闪烁-ucgui ucos” 与 “STM32 奋斗板-ADC-ucgui ucos” 程序, 编写基于 uC/OS II 的应用程序, 同时实现下述功能:

(1) 通过界面上的滑动条控制板子上的 LED1 的亮度 (参考定时器呼吸灯程序)

(2) 实时显示 ADC1 的通道 10 的采样值 0-4096 (参考 ADC 程序)

二、实验内容和原理

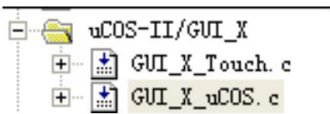
1. ucGUI



uCGUI_DEMO.lib 包含了和 ucgui 各部件资源有关系的库, 在这里包含库文件, 而不是包含源文件, 是为了在开发时, 编译速度更快捷。如果要修改显示屏的分辨率的参数, 及触摸屏原点初始值, 请在光盘里的 uCGUI_LIB 横版显示库或者 uCGUI_LIB 竖版显示库目录下的工程里进行修改。并重新编译为 lib。将这个 lib 文件, 拷贝到应用程序的 uCGUILib 目录下替换以前的旧 lib 文件。uCGUI_LIB 横版显示库目录下的工程是用于奋斗板显示屏的

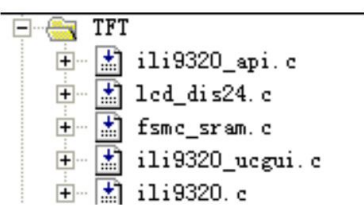
横板 ucgui 例程显示。 uCGUI_LIB 竖版显示库目录下的工程是用于奋斗板显示屏的竖板 ucgui 例程显示。

uCOS-II/GUI_X 组项:



这个组项下的文件和在 ucOSII 内核调度下的触摸操作有关，也和 ucOSII 内核调度下的 ucgui 任务有关。

TFT 组项:



Ili9320_api.c 是 ucGUI 部件所用到的底层函数原型定义
Lcd_dis24.c 包含了奋斗板显示屏模块硬件驱动程序
Fsmc.sram.c 是和 FSMC 显示接口有关的定义
Ili9320_ucgui.c 是和 ucgui 画图函数有关系的接口函数定义
Ili9320.c 是 ucgui 与 fsmc 有关系的接口函数库

2. uCGUI 函数原型

窗体的建立:

```
WM_HWIN GUI_CreateDialogBox(const GUI_WIDGET_CREATE_INFO* paWidget,
                             int NumWidgets, WM_CALLBACK* cb,
                             WM_HWIN hParent,int x0, int y0);
```

Parameter	Meaning
paWidget	Pointer to resource table defining the widgets to be included in the dialog.
NumWidgets	Total number of widgets included in the dialog.
cb	Pointer to an application-specific callback function (dialog procedure).
hParent	Handle of parent window (0 = no parent window).
x0	X-position of the dialog relative to parent window.
y0	Y-position of the dialog relative to parent window.



回调函数

```
void callback(WM_MESSAGE* pMsg);
```

Parameter	Meaning
pMsg	Pointer to a data structure of type WM_MESSAGE.

窗体标题栏字体设置

FRAMEWIN_SetFont()

Before	After
	

Description

Sets the title font.

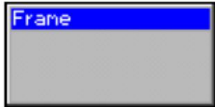

Prototype

```
void FRAMEWIN_SetFont(FRAMEWIN_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
<code>hObj</code>	Handle of frame window.
<code>pFont</code>	Pointer to the font.

窗体背景色设置

FRAMEWIN_SetClientColor()

Before	After
	

Description

Sets the color of the client window area of a specified frame window.

Prototype

```
void FRAMEWIN_SetClientColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
<code>hObj</code>	Handle of frame window.
<code>Color</code>	Color to be set.

获得对话框里某某项目的句柄

WM_GetDialogItem()

Description

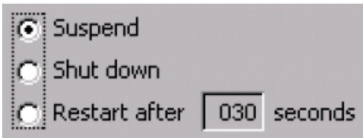
Returns the window handle of a dialog box item (widget).

Prototype

```
WM_HWIN WM_GetDialogItem(WM_HWIN hDialog, int Id);
```

Parameter	Meaning
<code>hDialog</code>	Handle of dialog box.
<code>Id</code>	Window Id of the widget.

TEXT: Text widget



[文本框字体设置](#)

TEXT_SetFont()

Description

Sets the font to be used for a specified text widget.

Prototype

```
void TEXT_SetFont(TEXT_Handle hObj, const GUI_FONT* pFont);
```

Parameter	Meaning
hObj	Handle of text widget.
pFont	Pointer to the font to be used.

[文本框字体颜色设置](#)

TEXT_SetTextColor()

Description

Sets the text color of a specified text widget.

Prototype

```
void TEXT_SetTextColor(TEXT_Handle pObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of text widget.
Color	New text color.

EDIT: Edit widget



[文本编辑框字体设置](#)

EDIT_SetFont()

Description

Sets the edit field font.

Prototype

```
void EDIT_SetFont(EDIT_Handle hObj, const GUI_FONT* pFont);
```

Parameter	Meaning
hObj	Handle of edit field.
pFont	Pointer to the font.

[设置文本编辑框里的文本内容](#)



EDIT_SetText()

Description

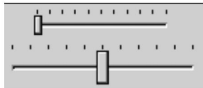
Sets the text to be displayed in the edit field.

Prototype

```
void EDIT_SetText(EDIT_Handle hObj, const char* s)
```

Parameter	Meaning
hObj	Handle of edit field.
s	Text to display.

SLIDER: Slider widget



设置滑动条的滑动取值范围

SLIDER_SetRange()

Description

Sets the range of the slider.

Prototype

```
void SLIDER_SetRange(SLIDER_Handle hObj, int Min, int Max);
```

Parameter	Meaning
hObj	Handle of slider widget.
Min	Minimum value.
Max	Maximum value.

设置滑动条的当前位置

SLIDER_SetValue()

Description

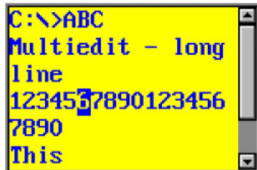
Sets the current value of the slider bar.

Prototype

```
void SLIDER_SetValue(SLIDER_Handle hObj, int v);
```

Parameter	Meaning
hObj	Handle of slider widget.
v	Value to be set.

多行编辑框



在窗体上建立多行编辑框

MULTIEDIT_Create()

(Obsolete, MULTIEDIT_CreateEx should be used instead)

Description

Creates a MULTIEDIT widget of a specified size at a specified location.

Prototype

```
MULTIEDIT_HANDLE MULTIEDIT_Create(int x0, int y0,
                                   int xsize, int ysize,
                                   WM_HWIN hParent, int Id, int Flags,
                                   int ExFlags, const char * pText,
                                   int MaxLen);
```

Parameter	Meaning
x0	Leftmost pixel of the multiedit widget (in parent coordinates).
y0	Topmost pixel of the multiedit widget (in parent coordinates).
xsize	Horizontal size of the multiedit widget (in pixels).
ysize	Vertical size of the multiedit widget (in pixels).
hParent	Parent window of the multiedit widget.
Id	ID of the multiedit widget.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 15: "The Window Manager" for a list of available parameter values).
ExFlags	(see table below)
pText	Text to be used.
MaxLen	Maximum number of bytes for text and prompt.

Permitted values for parameter ExFlags	
MULTIEDIT_CF_AUTOSCROLLBAR_H	Automatic use of a horizontal scrollbar.
MULTIEDIT_CF_AUTOSCROLLBAR_V	Automatic use of a vertical scrollbar.
MULTIEDIT_CF_INSERT	Enables insert mode.
MULTIEDIT_CF_READONLY	Enables read only mode.

设置最大字符数量

MULTIEDIT_SetMaxNumChars()

Description
Sets the maximum number of characters used by text and prompt.

Prototype
void MULTIEDIT_SetMaxNumChars(MULTIEDIT_HANDLE hObj, unsigned MaxNumChars);

Parameter	Meaning
hObj	Handle of multiedit widget.
MaxNumChars	Maximum number of characters.

设置文字对齐方式

MULTIEDIT_SetTextAlign()

Description
Sets the text alignment for the given MULTIEDIT widget.

Prototype
void MULTIEDIT_SetTextAlign(MULTIEDIT_HANDLE hObj, int Align);

Parameter	Meaning
hObj	Handle of multiedit widget.
Align	(see table below)

Permitted values for parameter Align	
GUI_TA_LEFT	Left text align.
GUI_TA_RIGHT	Right text align.

设置文字回绕

MULTIEDIT_SetWrapWord()

Description
Enables the word wrapping mode.

Prototype
void MULTIEDIT_SetWrapWord(MULTIEDIT_HANDLE hObj);

Parameter	Meaning
hObj	Handle of multiedit widget.

在多行文本框里追加内容

MULTIEDIT_AddText()

Description
Adds the given text at the current cursor position.

Prototype
int MULTIEDIT_AddText(MULTIEDIT_HANDLE hObj, const char * s);

Parameter	Meaning
hObj	Handle of multiedit widget.
s	Pointer to a NULL terminated text to be added.

获得当前的内容的长度

MULTIEDIT_GetTextSize()

Description
Returns the buffer size used to store the current text (and prompt).

Prototype
int MULTIEDIT_GetTextSize(MULTIEDIT_HANDLE hObj);

Parameter	Meaning
hObj	Handle of multiedit widget.

设置字体

MULTIEDIT_SetFont()

Description

Sets the font used to display the text and the prompt.

Prototype

```
void MULTIEDIT_SetFont(MULTIEDIT_HANDLE hObj, const GUI_FONT * pFont);
```

Parameter	Meaning
<code>hObj</code>	Handle of multiedit widget.
<code>pFont</code>	Pointer to font to be used.

设置光标位置

MULTIEDIT_SetCursorOffset()

Description

Sets the cursor position to the given character.

Prototype

```
void MULTIEDIT_SetCursorOffset(MULTIEDIT_HANDLE hObj, int Offset);
```

Parameter	Meaning
<code>hObj</code>	Handle of multiedit widget.
<code>Offset</code>	New cursor position.

Additional information

The number of characters used for the prompt has to be added to the parameter Offset. If a prompt is used the value for parameter Offset should not be smaller than the number of characters used for the prompt.

3. uCosII 概念解释

任务：任务通常是一个无限的循环，返回参数必须定义为 `void`。当任务开始执行时，会有一个参数传递给用户任务代码。uCOSII 可以管理 64 个任务，其中系统保留了 8 个任务。开放给用户的有 56 个任务，每个任务的优先级都不同，任务的优先级号越低，任务的优先级越高，在这个版本的 uCOSII 中，任务的优先级号就是任务编号。

任务的状态一定是以下 5 种之一：

睡眠态：就是任务没有交给 uCOSII 调度的，也就是没用经过建立的任务。只保存在存储器空间里。

就绪态：任务一旦建立，任务就进入就绪态。任务可以通过调用 `OSTaskDel()` 返回到睡眠态。

运行态：任何时刻只能有一个任务处于运行态。

等待状态：正在运行的任务可以通过调用以下 2 个函数之一，将自身延迟一段时间。这 2 个函数是 `OSTimeDly()` 或 `OSTimeDlyHMSM()`。这个任务于是进入等待状态，一直到函数中定义的延迟时间到。正在运行的任务可能在等待某一事件的发生，可以通过调用以下函数之一实现：`OSFlagPend()`、`OSSemPend()`、`OSMutexPend()`、`OSMboxPend()` 或 `OSQPend()`。如果某事件并未发生，调用上述函数的任务就进入了等待状态，直到等待的事件发生了。当任务因等待事件被挂起时，下一个优先级最高的就绪任务就得到了 CPU 的使用权。当时间发生了或等待延时超时时，被挂起的任务就进入就绪态。

中断服务态：正在运行的任务是可以被中断的，被中断的任务于是进入了中断服务态，响应中断时，正在执行的任务被挂起，中断服务程序控制了 CPU 的使用权。从中断服务程序返回前，uCOSII 要判定被中断的任务是否是当前就绪任务里优先级最高的，如果不是，就执行优先级最高的那个任务。如果是，就执行被中断的这个任务。

消息邮箱：这是 uCOSII 中的一种通信机制，可以使一个任务或者中断服务程序向另一个任务发送一个指针型的变量，通常该指针指向了一个包含了消息的特定数据结构。在例程中需

要建立邮箱，用到了函数 OSMboxCreate(), 串口 1 中断服务程序用到了向邮箱发送一则消息的函数 OSMboxPost(), 串口接收任务用到了等待邮箱中消息的函数 OSMboxPend()。

在这里特别注意: OSTimeDly()的延时时间是以节拍数来衡量的, OSTimeDlyHMSM()的延时时间则是以具体时间大小来衡量的。调用这两个延时函数都会挂起本任务, CPU 就会执行就绪表中优先级最高的任务。

有一点要注意的是任务调用 OSTimeDly()后, 一旦规定的时间期满或者有其它的任务通过调用 OSTimeDlyResume()取消了延时, 它就会马上进入就绪状态, 而不是执行状态, 也就是说延时结束后本任务不一定会被立刻执行。

4. 程序启动

● 程序流程讲解

经过启动文件的初始化, 首先在 C 代码里, 执行 main (), 在启动 ucOSII 内核前禁止 CPU 的中断-CPU_IntDis(), 防止启动过程中系统崩溃, 然后对 ucOSII 内核进行初始化 OSInit(), 以上两个函数都不用特意修改, 在 STM32 平台上已经被移植好了。

对板子上的一些用到的外设进行初始化设置 BSP_Init(), 这个函数包含了对系统时钟的设置 RCC_Configuration(), 将系统时钟设置为 72MHz, 对 LED 闪烁控制端口进行设置 GPIO_Configuration(), 对中断源进行配置 NVIC_Configuration(), 对触摸控制进行配置 tp_config(), 对显示器接口 FSMC 进行配置, 板子上的外设初始化完毕。

```
/* *****  
 * 名 称: void BSP_Init(void)  
 * 功 能: 奋斗板初始化函数  
 * 入口参数: 无  
 * 出口参数: 无  
 * 说 明:  
 * 调用方法: 无  
 ***** */  
void BSP_Init(void)  
{  
  
    RCC_Configuration();           //系统时钟初始化及端口外设时钟使能  
    NVIC_Configuration();         //中断源配置  
    GPIO_Configuration();         //状态LED1的初始化  
    USART_Config(USART1, 115200); //串口1初始化  
    tp_Config();                 //SPI1 触摸电路初始化  
    FSMC_LCD_Init();             //FSMC TFT接口初始化  
  
}
```

建立主任务, 该任务是为了在内核启动后, 建立另外 2 个用户任务, 并清 0 节拍计数器, 启动 ucOSII 内核。

```
//建立主任务, 优先级最高 建立这个任务另外一个用途是为了以后使用统计任务  
os_err = OSTaskCreate((void *) (void *) App_TaskStart,           //指向任务代码的指针  
                      (void *) 0,                                 //任务开始执行时, 传递给任务的参数的指针  
                      (OS_STK *) &App_TaskStartStk[APP_TASK_START_STK_SIZE - 1], //分配给任务的堆栈的栈顶指针 从顶向下递减  
                      (INT8U) APP_TASK_START_PRIO);               //分配给任务的优先级  
  
OSTimeSet(0);             //ucOSII的节拍计数器清0 节拍计数器是0-4294967295  
OSStart();                //启动ucOSII内核
```


主任务的任务名为 App_TaskStart, 主任务有自己的堆栈, 堆栈尺寸为 APP_TASK_START_STK_SIZE*4 (字节), 然后执行 uclosII 内部函数 OSTimeSet(0), 将节拍计数器清 0, 节拍计数器范围是 0-4294967295, 对于节拍频率 100hz 时, 每隔 497 天就重新计数, 调用内部函数 OSStart(), 启动 uclosII 内核, 此时 uclosII 内核开始运行。对任务表进行监视, 主任务因为已经处于就绪状态, 于是开始执行主任务 App_TaskStart(), uCLOSII 的任务结构规定必须为无返回的结构, 也就是无限循环模式。

```
static void App_TaskStart(void* p_arg)
{
    (void) p_arg;
    //初始化uclosII时钟节拍
    OS_CPU_SysTickInit();

    //使能uclos 的统计任务
    #if (OS_TASK_STAT_EN > 0)

        OSStatInit();          //----统计任务初始化函数
    #endif

    App_TaskCreate();          //建立其他的任务

    while (1)
    {
        OSTimeDlyHMSM(0, 0, 1, 100);
    }
}
```

6 个用户任务各自有自己的堆栈空间。

```
static void App_TaskCreate(void)
{
    Com1_MBOX=OSMboxCreate((void *) 0);          //建立串口1中断的邮箱

    /* 建立用户界面任务 */
    OSTaskCreateExt(AppTaskUserIF,
        (void *)0,
        (OS_STK *) &AppTaskUserIFStk[APP_TASK_USER_IF_STK_SIZE-1],
        APP_TASK_USER_IF_Prio,
        APP_TASK_USER_IF_Prio,
        (OS_STK *) &AppTaskUserIFStk[0],
        APP_TASK_USER_IF_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    /* 建立触摸驱动任务 */
    OSTaskCreateExt(AppTaskKbd,
        (void *)0,
        (OS_STK *) &AppTaskKbdStk[APP_TASK_KBD_STK_SIZE-1],
        APP_TASK_KBD_Prio,
        APP_TASK_KBD_Prio,
        (OS_STK *) &AppTaskKbdStk[0],
        APP_TASK_KBD_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    //串口1接收及发送任务
    OSTaskCreateExt(Task_Com1, (void *)0, (OS_STK *) &Task_Com1Stk[Task_Com1_STK_SIZE-1], Task_Com1_Prio, Task_Com1_Prio,
        (OS_STK *) &Task_Com1Stk[0],
        Task_Com1_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    //LED1 闪烁任务
    OSTaskCreateExt(Task_Led1, (void *)0, (OS_STK *) &Task_Led1Stk[Task_Led1_STK_SIZE-1], Task_Led1_Prio, Task_Led1_Prio,
        (OS_STK *) &Task_Led1Stk[0],
        Task_Led1_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    //LED2 闪烁任务
    OSTaskCreateExt(Task_Led2, (void *)0, (OS_STK *) &Task_Led2Stk[Task_Led2_STK_SIZE-1], Task_Led2_Prio, Task_Led2_Prio,
        (OS_STK *) &Task_Led2Stk[0],
        Task_Led2_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    //LED3 闪烁任务
    OSTaskCreateExt(Task_Led3, (void *)0, (OS_STK *) &Task_Led3Stk[Task_Led3_STK_SIZE-1], Task_Led3_Prio, Task_Led3_Prio,
        (OS_STK *) &Task_Led3Stk[0],
        Task_Led3_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);
}
```

在 app_cfg.h 里面可以看到任务设定的优先级，此处是 ADC+LED 程序的展示：

```
#define APP_TASK_START_PRIO                2
#define APP_TASK_USER_IF_PRIO             13
#define APP_TASK_KBD_PRIO                 12
#define Task_CoM1_PRIO                     4
#define Task_Led1_PRIO                     7
#define Task_Led2_PRIO                     8
#define Task_Led3_PRIO                     9
#define Task_Adc_PRIO                      3
```

● ADC 采样

此处使用了 ADC 过采样，两个通道同时进行采样，并采集了 10 个数据再取平均：

```
void DMAChannel1_IRQHandler(void)
{
    OS_CPU_SR cpu_sr;
    OS_ENTER_CRITICAL(); //保存全局中断标志,关总中断/* Tell uC/OS-II that we are starting an ISR*/
    OSIntNesting++;      //中断嵌套
    OS_EXIT_CRITICAL();  //恢复全局中断标志
    if(DMA_GetITStatus(DMA1_IT_TC1)) //转换完成
    {
        /* 对ADC进行过采样,保证精度*/
        ADC_ConvertedValue1[0]=(ADC_ConvertedValue[40]+ADC_ConvertedValue[42]+ADC_ConvertedValue[44]+ADC_ConvertedValue[46]+ADC_ConvertedValue[48]+ADC_ConvertedValue[50]+ADC_ConvertedValue[52]+ADC_ConvertedValue[54]+ADC_ConvertedValue[56]+ADC_ConvertedValue[58])/10;
        ADC_ConvertedValue1[1]=(ADC_ConvertedValue[41]+ADC_ConvertedValue[43]+ADC_ConvertedValue[45]+ADC_ConvertedValue[47]+ADC_ConvertedValue[49]+ADC_ConvertedValue[51]+ADC_ConvertedValue[53]+ADC_ConvertedValue[55]+ADC_ConvertedValue[57]+ADC_ConvertedValue[59])/10;
        DMA_ClearITPendingBit(DMA1_IT_GL1); //清除中断标志
    }
    OSIntExit(); //在os_core.c文件里定义,如果有更高优先级的任务就绪了,则执行一次任务切换
}
```

ADC 采样在 SysTickHandler 中断执行函数中来决定采样周期，并传送采样信号量。

```
void SysTickHandler(void)
{
    /* 时钟节拍为10ms一次,作为ucos的时钟节拍*/
    OS_CPU_SR cpu_sr;
    OS_ENTER_CRITICAL(); //保存全局中断标志,关总中断/* Tell uC/OS-II that we are starting an ISR*/
    OSIntNesting++;
    OS_EXIT_CRITICAL(); //恢复全局中断标志

    OSTimeTick(); // Call uC/OS-II's OSTimeTick(),在os_core.c文件里定义,主要判断延时的任务是否计时到*/
    ADC_TIMEOUT++;
    if(ADC_TIMEOUT>100){ //1秒采样一次
        ADC_TIMEOUT=0;
        ADC_R=1; //设置界面显示用ADC 采样完毕标志
        OSSemPost(ADC_SEM); //传送ADC采样信号量
    }
    OSIntExit(); //在os_core.c文件里定义,如果有更高优先级的任务就绪了,则执行一次任务切换
}
```

而在 app.c 中的任务就在等待信号量。

```
static void Task_ADC(void *p_arg){ //ADC任务
    INT8U err;

    (void)p_arg;
    while(1){
        OSSemPend(ADC_SEM,0,&err); //等待ADC信号量
        itoa((ADC_ConvertedValue[40]+ADC_ConvertedValue[42])/2, ADC_STR1,10); //ADC1通道10的数值转为字符串
        USART_OUT(USART1, "\r\nADC1 通道10: %s",ADC_STR1); //串口1输出通道10的数值
        USART_OUT(USART1, "\r\nADC1 通道10: %d", (ADC_ConvertedValue[0]-1250)/1650*4200); //串口1输出通道10的数值
        USART_OUT(USART1, "\r\n");
    }
}
```

- 采样LED 闪烁

在感应到 slider 改变了之后读取了 milsec1

```
static void _OnValueChanged(WM_HWIN hDlg, int Id) {

    if ((Id == GUI_ID_SLIDER0)) {                //slider0 的值被改变
        milsec1=SLIDER_GetValue(slider0);        //获得slider0的值
        EDIT_SetValue(edit0,milsec1);            //EDIT0 的值被改变
    }
}

/*****
* 名    称: static void _cbCallback(WM_MESSAGE * pMsg)
* 功    能: 窗体回调函数
* 入口参数: 无
* 出口参数: 无
* 说    明:
* 调用方法: 无
*****/
static void _cbCallback(WM_MESSAGE * pMsg) {
    int NCode, Id;
    WM_HWIN hDlg;
    hDlg = pMsg->hWin;
    switch (pMsg->MsgId) {
        case WM_NOTIFY_PARENT:
            Id = WM_GetId(pMsg->hWinSrc);        /*获得窗体部件的ID*/
            NCode = pMsg->Data.v;                /*动作代码 */
            switch (NCode) {
                case WM_NOTIFICATION_VALUE_CHANGED: /*窗体部件的值被改变 */
                    _OnValueChanged(hDlg, Id);
                    break;
                default:
                    break;
            }
            break;
        default:
            WM_DefaultProc(pMsg);
    }
}

/*****
* 名    称: static void _OnValueChanged(WM_HWIN hDlg, int Id)
* 功    能: 值被改变的动作
* 入口参数: 无
* 出口参数: 无
* 说    明:
* 调用方法: 无
*****/
static void _OnValueChanged(WM_HWIN hDlg, int Id) {

    if ((Id == GUI_ID_SLIDER0)) {                //slider0 的值被改变
        milsec1=SLIDER_GetValue(slider0);        //获得slider0的值
        EDIT_SetValue(edit0,milsec1);            //EDIT0 的值被改变
    }
    else if ((Id == GUI_ID_SLIDER1)) {            //slider1 的值被改变
        milsec2=SLIDER_GetValue(slider1);        //获得slider1的值
        EDIT_SetValue(edit1,milsec2);            //EDIT1 的值被改变
    }
    else if ((Id == GUI_ID_SLIDER2)) {            //slider2 的值被改变
        milsec3=SLIDER_GetValue(slider2);        //获得slider2的值
        EDIT_SetValue(edit2,milsec3);            //EDIT2 的值被改变
    }
}
```

```
volatile unsigned int milsec1,milsec2,milsec3;
```

volatile 的作用是作为指令关键字，确保本条指令不会因编译器的优化而省略；

使用 PWM 模式输出

```
static void Task_Led1(void* p_arg)
{
    TIM_OCInitTypeDef TIM3_OCInitStructure;

    (void) p_arg;
    while (1)
    {
        OSTimeDlyHMSM(0, 0, 0, 100);
        TIM3_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;           //PWM模式2
        TIM3_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //输出禁止
        TIM3_OCInitStructure.TIM_Pulse = milsec1*240;
        TIM3_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
        TIM_OC2Init(TIM3, &TIM3_OCInitStructure);
    }
}
```

三、主要仪器设备

STM32 奋斗开发板、RS232 总线、电脑、ST-Link 口线等。

四、操作方法和实验步骤

1. 硬件接线

- (1) 图略，见实验 PPT
- (2) 注意用 USB 线供电

2. 程序主要文件结构

- (1) app.c 中的 int main(void)函数——程序入口

- ① 功能：main 函数在设置好一些变量和初始化配置之后，建立主任务并启动 ucousii 操作系统内核。
- ② 核心代码：


```

67 int main(void)
68 {
69     CPU_INT08U os_err;
70     /* 禁止所有中断 */
71     CPU_IntDis();
72
73     /* ucOSII 初始化 */
74     OSInit();
75
76     /* 硬件平台初始化 */
77     BSP_Init();
78
79     //建立主任务， 优先级最高 建立这个任务另外一个用途是为了以后使用统计任务
80     os_err = OSTaskCreate((void *) (void *)) App_TaskStart, //指向任务代码的指针
81                         (void *) 0, //任务开始执行时，传递给任务的参数的指针
82                         (OS_STK *) &App_TaskStartStk[APP_TASK_START_STK_SIZE - 1], //分配给任务的堆栈的栈
83                         (INT8U) APP_TASK_START_PRIO); //分配给任务的优先级
84
85     OSTimeSet(0); //ucosII的节拍计数器清0 节拍计数器是0-4294967295
86     OSStart(); //启动ucosII内核
87     return (0);
88 }

```

(2) bsp.c 中的 void BSP_Init(void)函数——硬件初始化配置

- ① 功能：main 函数中调用 BSP_Init 函数初始化硬件，包括定时器、串口、GPIO 复用等等
- ② 核心代码（PWM 时钟配置）

```

62 /*-----*/
63 TIM3CLK=72MHz 预分频系数Prescaler=2 经过分频 定时器时钟为24MHz
64 根据公式 通道输出占空比=TIM3_CCR2/(TIM_Period+1),可以得到TIM_Pulse的计数值
65 捕获/比较寄存器2 TIM3_CCR2= CCR2_Val
66 /*-----*/
67 TIM3_TimeBaseStructure.TIM_Prescaler = 2; //预分频器TIM3_PSC=3
68 TIM3_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //计数器向上计数模式 TIM3_CR1[4]=0
69 TIM3_TimeBaseStructure.TIM_Period = 24000; //自动重载寄存器TIM3_ARR 确定频率为1KHz
70 TIM3_TimeBaseStructure.TIM_ClockDivision = 0x0; //时钟分频因子 TIM3_CR1[9:8]=00
71 TIM3_TimeBaseStructure.TIM_RepetitionCounter = 0x0;
72
73 TIM_TimeBaseInit(TIM3, &TIM3_TimeBaseStructure); //写TIM3各寄存器参数
74
75 TIM3_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2; //PWM模式2 TIM3_CCMR1[14:12]=111 在向上计数时，
76 //一旦TIMx_CNT<TIMx_CCR1时通道1为无效电平，否则为有效电平
77 TIM3_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //输入/捕获2输出允许 OC2信号输出到对应
78 //TIM3_OCInitStructure.TIM_Pulse = CCR2_Val; //确定占空比，这个值决定了有效电平的时间。
79 TIM3_OCInitStructure.TIM_Pulse = 10*LED_interval;
80 TIM3_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low; //输出极性 低电平有效 TIM3_CCER[5]=1;
81
82 TIM_OC2Init(TIM3, &TIM3_OCInitStructure);
83 TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Enable);
84 TIM_Cmd(TIM3, ENABLE); //启动定时器3 TIM3_CR1[0]=1;

```

(3) app.c 中的 void App_TaskStart(void* p_arg)函数

- ① 功能：开始 ucOS 的各项任务（图形界面、ADC 显示、LED 等任务）

(4) Fun.c 中的 void Fun(void)函数——图形界面

- ① 功能：图形界面任务中调用 Fun 函数初始化和刷新 LCD 图形界面
- ② 核心代码

```

104 void Fun(void) {
105     unsigned char edit_cur;
106     GUI_CURSOR_Show();
107
108     //WM_SetCreateFlags(WM_CF_MEMDEV); // Automatically use memory devices on all windows */
109     /* 建立窗体，包含了资源列表，资源数目，并指定回调函数 */
110     hWin = GUI_CreateDialogBox(aDialogCreate, GUI_COUNTOF(aDialogCreate), _cbCallback, 0, 0, 0);
111     hWin1 = GUI_CreateDialogBox(Adcdata, GUI_COUNTOF(Adcdata), 0, 0, 0, 0);

```

```

161 while (1)
162 {
163     if(ADC_R==1){ //1秒间隔采样
164         ADC_R=0;
165         //文本框显示
166         TEXT_SetText(text5,ADC_STR1);
167     }
168     WM_Exec(); //刷新屏幕
169 } //屏幕刷新
170 }

```

(5) Fun.c 中的 `_cbCallback(WM_MESSAGE * pMsg)` 函数和 `void _OnValueChanged(WM_HWIN hDlg, int Id)` 函数——窗口操作处理

① 功能：_cbCallback 当窗口部件产生操作时，进行相应的处理（有值改变时调用 _OnValueChanged 函数）。_OnValueChanged 函数则对 LED 亮度调节滑块被滑动时更新 LED 的亮度值和屏幕显示的亮度值。

② 核心代码

```

87 static void _OnValueChanged(WM_HWIN hDlg, int Id) {
88
89     if ((Id == GUI_ID_SLIDER0)) { //slider0 的值被改变
90         msec1=SLIDER_GetValue/slider0); //获得slider0的值
91         LED_interval = msec1;
92         EDIT_SetValue(edit0,msec1); //EDIT0 的值被改变
93     }
94 }

```

3. 例程需要修改的地方

我使用的是 ADC 例程，在其基础上加入 LED 例程、PWM 例程的部分内容。

(1) app.c 文件

- ① ADC 例程自带 LED 任务函数，但其功能应该由 LED 闪烁变为 LED 亮度根据参数调节
- ② 因为使用 PWM 方式调节亮度，在 LED 任务函数中设置 TIM3 输出模式为 PWM2 模式、根据滑块参数 int LED_interval 调节 PWM 占空比（LED_interval 取值 0~100，乘以 240 后取值正是 0~ARR，实现滑块控制 0~100%占空比调节），以及一些其他的定时器设置
- ③ 在主函数中要实现相应软硬件的初始化及全局变量的定义，在任务中设置任务指针、任务堆栈及优先级，而后启动 ucos 内核。
- ④ 在 LED1 的任务中，将滑动条的值写入比较寄存器中从而实现 PWM 波占空比的控制，要注意在函数后必须加上延时函数，否则会将其他函数挂起无法正确运行。


```

CPU_INT08U os_err;
/* 禁止所有中断 */
CPU_IntDis();
/* ucosII 初始化 */
OSInit();
/* 硬件平台初始化 */
BSP_Init();

//默认LED闪烁间隔500ms
milsec1=5000;

//建立主任务，优先级最高 建立这个任务另外一个用途是为了以后使用统计任务
os_err = OSTaskCreate((void *) (void *)) App_TaskStart, //指向任务代码的指针
                    (void *) 0, //任务开始执行时，传递给任务的参数的指针
                    (OS_STK *) &App_TaskStartStk(APP_TASK_START_STK_SIZE - 1), //分配给任务的堆栈的栈顶指针 从顶向下递减
                    (INT08U) APP_TASK_START_PRIO); //分配给任务的优先级

os_err=os_err;
OSTimeSet(0); //ucosII的节拍计数器清0 节拍计数器是0-4294967295
OSStart(); //启动ucosII内核
return (0);

256 static void Task_Led1(void* p_arg)
257 {
258     TIM_OCInitTypeDef TIM3_OCInitStructure;
259     (void) p_arg;
260     while (1)
261     { /* 100ms间隔LED闪烁 */
262         // Led_ON();
263         // OSTimeDlyHMSM(0, 0, 0, LED_interval);
264         // Led_OFF();
265         // OSTimeDlyHMSM(0, 0, 0, LED_interval);
266         OSTimeDlyHMSM(0, 0, 0, 100);
267         TIM3_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2; //PWM模式2
268         TIM3_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //输出禁止
269         TIM3_OCInitStructure.TIM_Pulse = 240*LED_interval; //确定占空比
270         TIM3_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
271         TIM_OC2Init(TIM3, &TIM3_OCInitStructure);
272     }
273 }

```

(2) bsp.c 文件

- ① GPIO、USART 等都在例程中有了其初始化函数，需要改变的是和 PWM 相关的硬件设置
- ② 首先是定时器初始化函数 void time_ini(void)，拷贝自 PWM 实验例程，主要是将 GPIO 复用到 PB5（即 LED1）、时钟预分频到 24kHz、ARR 设置为 24000 以及其他控制位设置等。

```

67 TIM3_TimeBaseStructure.TIM_Prescaler = 2; //预分频器TIM3 PSC=3
68 TIM3_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //计数器向上计数模式 TIM3_CR1[4]=0
69 TIM3_TimeBaseStructure.TIM_Period = 24000; //自动重载寄存器TIM3_ARR 确定频率为1KHz
70 TIM3_TimeBaseStructure.TIM_ClockDivision = 0x0; //时钟分频因子 TIM3_CR1[9:8]=00
71 TIM3_TimeBaseStructure.TIM_RepetitionCounter = 0x0;
72

```

- ③ 其次是要在 void RCC_Configuration(void)函数中增加一条外设时钟配置的语句：
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE); 以使能 GPIO 重映射引脚的时钟。

```

95 void RCC_Configuration(void) {
96
97     SystemInit();
98     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
99                             RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE, ENABLE);
100     RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
101     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
102     RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
103 }
104

```

(3) Fun.c 文件

- ① 参考 LED 例程，添加滑块等元件，将第一个窗口作为 LED 使用，第二个窗口作为 ADC 使用，注意要在 void _OnValueChanged(WM_HWIN hDlg, int Id)函数中获取滑块的改变后的值。
- ② 在 Fun.c 中主要调整的是显示界面，这里只需要调整 LED1 的亮度，因此将原例程中 LED2 及 LED3 的部分删除即可。而后对布局进行一下简单的调整，将滑动条的范围设置为 10-24000。

```

/* 设置EDIT部件采用10进制 范围10-24000 */
EDIT_SetDecMode(edit0, milsec1, 10, 24000, 0, 0);

/* 设置TEXT部件的字体 */
TEXT_SetFont(text0, pFont);
TEXT_SetFont(text3, pFont);
TEXT_SetFont(text4, pFont);

/* 设置TEXT部件的字体颜色 */
TEXT_SetTextColor(text0, GUI_WHITE);
TEXT_SetTextColor(text3, GUI_WHITE);
TEXT_SetTextColor(text4, GUI_WHITE);

/* 设置slider部件的取值范围10-24000 */
SLIDER_SetRange(slider0, 10, 24000);

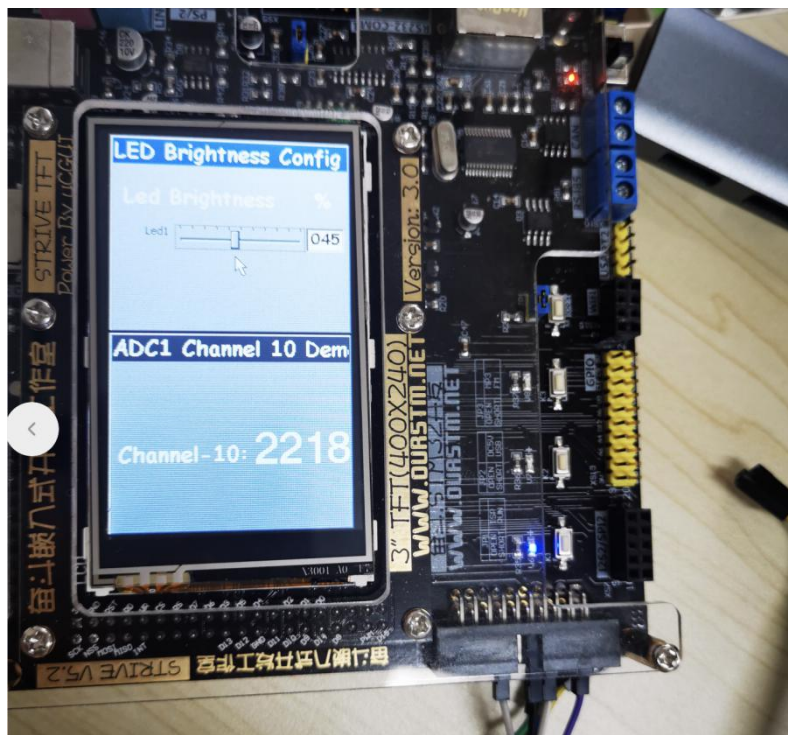
```

五、实验结果与分析

1. ADC 接地 + 亮度 0%



2. ADC 悬空 + 亮度 50%



3. ADC 接 VCC+亮度 100%



六、实验结果与分析

通过本次实验，我们学习了 STM32 的 ucOS 使用，了解了 ucOS 的原理和配置方法，在 ucOS 中实现了之前实验中做过的内容并完成了相应结果的显示。通过 ucOS 的配置，对 STM32 的使用有了更深入的理解，利用 ucOS 实现多任务可以帮助我们实现更多更丰富的功能。在一开始尝试使用 LED 例程进行修改的时候，发现没有办法响应，不知道是哪里出了问题，debug 了很久都没有发现。最后，使用 ADC 程序，在 ADC 程序上进行更改之后就没有怎么出现问题。