iTask多功能的轻型智能终端——设计文档

白玮琦 易红旭 黄鹏程 季嘉舟 叶鲁毅 陆欣茹

# 引言

* 1. 编写目的

为明确系统需求、安排项目规划与进度、组织软件开发与测试，撰写本文档。本文档供任课老师、助教、设计人员、开发人员参考。

* 1. 项目概况

项目名称：iTask多功能的轻型智能终端

任务来源：应用软件课程设计大作业

开发人员：白玮琦、易红旭、黄鹏程、季嘉舟、叶鲁毅、陆心茹

* 1. 设计特色

本项目利用stm32f103硬件开发板，基于嵌入式平台操作系统uC/OS，软硬结合开发，在操作系统平台上开发实现了时钟软件、日历软件、相机软件、游戏娱乐软件等多方面多功能应用，以满足用户对日常生活场景服务需求，同时也留予一定扩展接口以期后续进一步开发。

uC/OS-III是一个开源的实时操作系统，它可以移植到多种硬件平台，在STM 32中移植uC/OS操作系统作为我们的开发平台,并以此为基础开发多种应用软件，最终实现一个个性化程度高、多功能的智能设备终端。另一方面，我们也希望能够从这次的项目中，完成一个基本的从硬件接口到应用实现的完整过程，具体学习到操作系统的各项功能。

总的来说，嵌入式设备在当今的发展如火如荼，软硬结合开发对于项目完整性有着很高的优势。本项目参考现今已有产品，主要是智能手环等设备，进行功能设计、开发，同时兼顾实用与娱乐两方面综合需求，并在UI设计、界面美化等方面做出了我们的思考与设计，力求功能完备的同时也实现简约的设计思路。

# 项目设计结构

# 整体结构

在整体架构上，我们将本次项目自底向上设计为物理层、数据层、系统底层、内核功能、交互层以及最终的顶层应用层；从底层的stm32硬件芯片到操作系统uC/OS再到提供的应用软件服务。具体框架构成如下图：

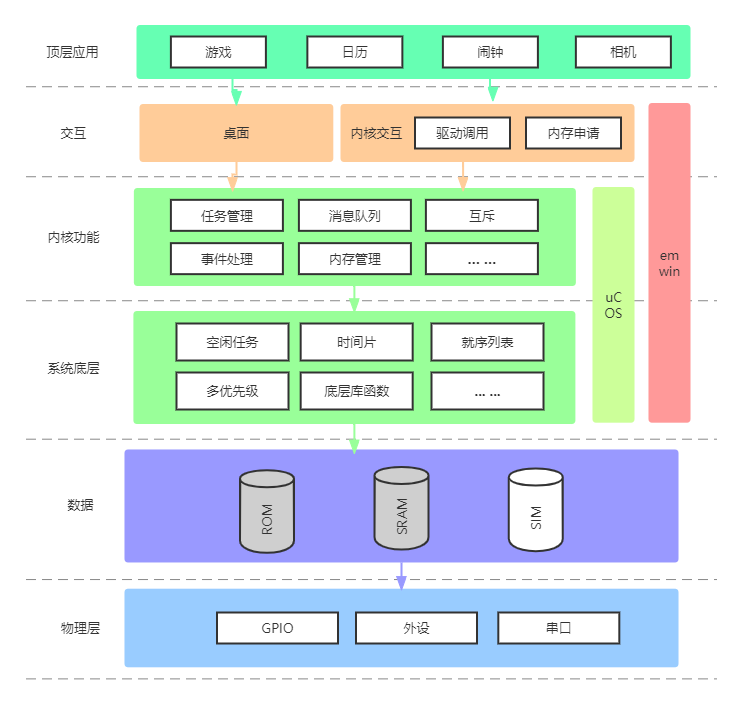


Figure ：项目框架设计

# Uc/OS结构

uC/OS-Ⅲ操作系统主要有EvalBoards、uC-CPU、uC-LIB、uCOS-Ⅲ四个文件夹，总体结构如下：

Software

├─EvalBoard //评估板相关文件

├─uC-CPU //CPU相关文件

│ └─ARM-Cortex-M3

│ ├─GNU

│ ├─IAR

│ └─RealView //对应MDK开发环境

├─uC-LIB //Micrim公司提供的官方库

└─uCOS-III

├─Ports //硬件接口层文件

│ └─ARM-Cortex-M3

│ └─Generic

│ ├─GNU

│ ├─IAR

│ └─RealView

└─Source

# 操作系统的移植和修改

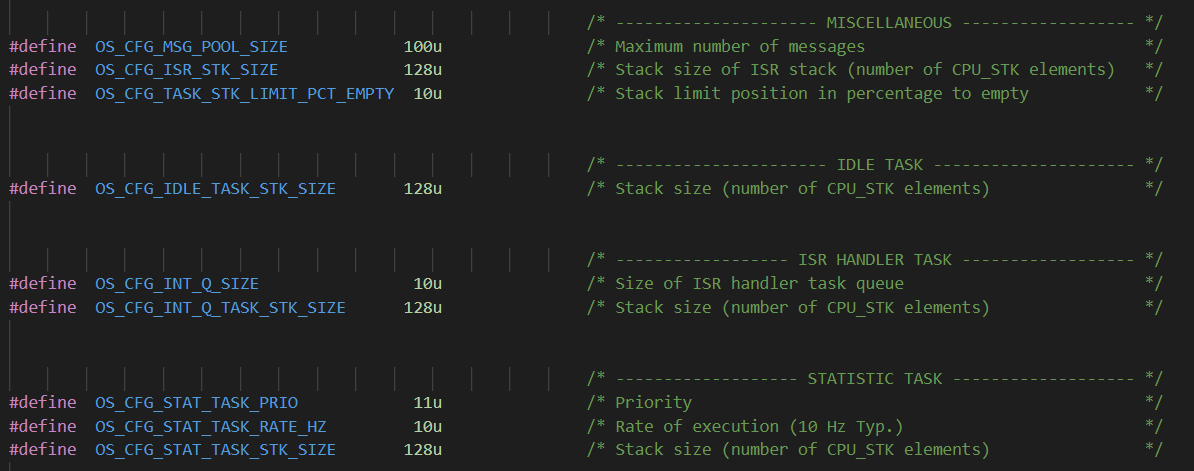
# CPU相关配置

为了使操作系统适合我们的项目，需要对相关的系统文件进行配置，而uC/OS提供了裁剪的功能，可以按需对系统进行裁剪。

定位到系统配置文件os\_cfg.h，原文件默认开启了全部功能，这里我们根据需要关闭了部分功能，并添加了中文注释。

# 

os\_cfg\_app.h使系统应用配置头文件，可以在里面定义任务的优先级、栈大小等基本信息，配置后的部分结果如下：



# 硬件初始化

参考野火开机例程程序，在bsp.c文件种创建BSP\_Init函数，专门用于实现板级外设初始化。

/\*

 \* 函数名：BSP\_Init

 \* 描述  ：时钟初始化、硬件初始化

 \* 输入  ：无

 \* 输出  ：无

 \*/

void BSP\_Init(void)

{

    uint16\_t readCal[8]={0};

    fs=(FATFS \*)comdata;

    /\*CRC和emWin没有关系，只是他们为了库的保护而做的，这样STemWin的库只能用在ST的芯片上面，别的芯片是无法使用的。 \*/

  RCC\_AHBPeriphClockCmd(RCC\_AHBPeriph\_CRC, ENABLE);

    Bsp\_NVIC\_Configuration();  /\* 中断优先级分组配置 \*/

    /\* LED 端口初始化 \*/

    LED\_GPIO\_Config();

    /\* 按键初始化 \*/

    Key1\_GPIO\_Config();

    Key2\_GPIO\_Config();

    /\* 蜂鸣器初始化 \*/

    Beep\_Init();

    /\* 串口1初始化 \*/

    USART1\_Config();

    /\* 触摸屏初始化 \*/

    XPT2046\_Init();

    /\*

     \* 在所有的窗口上自动的使用内存设备，在GUI\_Init后调用不包括

     \* 背景窗口（桌面窗口），之前调用才包括背景窗口

     \*/

    WM\_SetCreateFlags(WM\_CF\_MEMDEV);

    /\* 初始化GUI \*/

    GUI\_Init();

    /\* Register work area for each volume (Always succeeds regardless of disk status) \*/

    result=f\_mount(fs,"0:",1);

    printf("mount function is over!\n");

    //printf("mount\_res:%d\n",result);

    if(result!=FR\_OK)

    {

            printf("Mount Failed!");

            IsCal=0xE0;

            return;

    }

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 读取触摸屏校准参数  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    SPI\_FLASH\_BufferRead(&IsCal,CALADD, 1);

    if(IsCal==0x55)

    {

        SPI\_FLASH\_BufferRead((void \*)readCal,CALADD+2, sizeof(readCal));

        GUI\_TOUCH\_Calibrate(GUI\_COORD\_X,readCal[0], readCal[4], readCal[6], readCal[2]); /\* Calibrate X-axis \*/

        GUI\_TOUCH\_Calibrate(GUI\_COORD\_Y,readCal[1], readCal[5], readCal[7], readCal[3]); /\* Calibrate Y-axis \*/

    }

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    /\* 配置RTC秒中断优先级 \*/

    RTC\_NVIC\_Config();

    rtc\_flag=RTC\_CheckAndConfig(&systmtime);

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

其中的各个xxxx\_Config()用于初始化对应的各个外设，以LED\_GPIO\_Config()为例，用调用库函数的方法进行设置。

 /\*\*

  \* @brief  初始化控制LED的IO

  \* @param  无

  \* @retval 无

  \*/

void LED\_GPIO\_Config(void)

{

        /\*定义一个GPIO\_InitTypeDef类型的结构体\*/

        GPIO\_InitTypeDef GPIO\_InitStructure;

        /\*开启GPIOB和GPIOF的外设时钟\*/

        RCC\_APB2PeriphClockCmd( macLED1\_GPIO\_CLK|macLED2\_GPIO\_CLK|macLED3\_GPIO\_CLK, ENABLE);

        /\*选择要控制的GPIOB引脚\*/

        GPIO\_InitStructure.GPIO\_Pin = macLED1\_GPIO\_PIN;

        /\*设置引脚模式为通用推挽输出\*/

        GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP;

        /\*设置引脚速率为50MHz \*/

        GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

        /\*调用库函数，初始化GPIOB0\*/

        GPIO\_Init(macLED1\_GPIO\_PORT, &GPIO\_InitStructure);

        /\*选择要控制的GPIOF引脚\*/

        GPIO\_InitStructure.GPIO\_Pin = macLED2\_GPIO\_PIN;

        /\*调用库函数，初始化GPIOF7\*/

        GPIO\_Init(macLED2\_GPIO\_PORT, &GPIO\_InitStructure);

        /\*选择要控制的GPIOF引脚\*/

        GPIO\_InitStructure.GPIO\_Pin = macLED3\_GPIO\_PIN;

        /\*调用库函数，初始化GPIOF7\*/

        GPIO\_Init(macLED3\_GPIO\_PORT, &GPIO\_InitStructure);

        /\* 关闭所有led灯 \*/

        GPIO\_SetBits(macLED1\_GPIO\_PORT, macLED1\_GPIO\_PIN);

        /\* 关闭所有led灯 \*/

        GPIO\_SetBits(macLED2\_GPIO\_PORT, macLED2\_GPIO\_PIN);

    /\* 关闭所有led灯 \*/

    GPIO\_SetBits(macLED3\_GPIO\_PORT, macLED3\_GPIO\_PIN);

}

# 任务创建

在uC/OS系统中，每一个任务都是独立的，其运行环境单独保存在各自的栈空间中，在main.c中设置栈空间为128，定义任务主体函数AppTaskStart(),定义任务控制块AppTaskStartTCB。AppTaskStart()的代码如下：

static void  AppTaskStart(void \*p\_arg)

{

    OS\_ERR      err;

   (void)p\_arg;

  /\* 板级初始化 \*/

    BSP\_Init();

    CPU\_Init();

BSP\_Tick\_Init();

#if OS\_CFG\_STAT\_TASK\_EN > 0u

    OSStatTaskCPUUsageInit(&err);

#endif

    OS\_INFO("Creating Application Tasks...\n\r");

    AppTaskCreate();

    OS\_INFO("Creating Application Events...\n\r");

    AppObjCreate();

  /\*Delete task\*/

    while (DEF\_TRUE) {

        GUI\_TOUCH\_Exec();

        OSTimeDlyHMSM(0, 0, 0, 15,

                      OS\_OPT\_TIME\_HMSM\_STRICT,

                      &err);

    }

}

其中CPU\_Init()由操作系统的cpu\_core.c文件提供，BSP\_Init()即为之前我们编写的板级外设初始化函数。

主函数main()将任务主体函数、任务栈和任务控制块相结合，然后启动多任务系统，把控制权交给uC/OS系统，其代码如下：

int main(void)

{

  OS\_ERR err;

  /\* 初始化"uC/OS-III"内核 \*/

  OSInit(&err);

  /\*创建任务\*/

  OSTaskCreate((OS\_TCB     \*)&AppTaskStartTCB,  // 任务控制块指

               (CPU\_CHAR   \*)"App Task Start",  // 任务名称

               (OS\_TASK\_PTR )AppTaskStart,     // 任务代码指针

               (void       \*)0,                 // 传递给任务的参数

               (OS\_PRIO     )APP\_TASK\_START\_PRIO,   // 任务优先级

               (CPU\_STK    \*)&AppTaskStartStk[0],   // 任务堆栈基地址

               (CPU\_STK\_SIZE)APP\_TASK\_START\_STK\_SIZE/10,  // 堆栈剩余警戒线

               (CPU\_STK\_SIZE)APP\_TASK\_START\_STK\_SIZE,     // 堆栈大小

               (OS\_MSG\_QTY  )1u,                           // 可接收的最大消息队列数

               (OS\_TICK     )0u,           // 时间片轮转时间

               (void       \*)0,              // 任务控制块扩展信息

               (OS\_OPT      )(OS\_OPT\_TASK\_STK\_CHK |

                              OS\_OPT\_TASK\_STK\_CLR),       // 任务选项

               (OS\_ERR     \*)&err);                       // 返回值

  /\* 启动多任务系统，控制权交给uC/OS-III \*/

  OSStart(&err);

# 应用软件模块设计

# 4GUI主函数

在main.c中，通过调用AppTaskCreate()创建应用任务，进入任务块AppTaskUserapp，在其中设置while(1)的循环，运行GUI主函数UserAPP()，该函数设计在app.c文件中，该文件也是应用程序的入口文件。

UserAPP()函数如下：

void UserAPP(void)

{

    UserApp\_Flag=0;

    //准备建立3个窗口，以下是使用到的用户定义参数，方便在回调函数中使用

    WinPara.xSizeLCD = LCD\_GetXSize();              //LCD屏幕尺寸

    WinPara.ySizeLCD = LCD\_GetYSize();              //LCD屏幕尺寸

    WinPara.xSizeBM  = ICONVIEW\_Width;              //图标宽度

    WinPara.ySizeBM  = ICONVIEW\_Height;             //图标高度

    WinPara.ySizeBotWin=BOTWIN\_YSpace;              //界面下方窗口高度

    WinPara.xPosWin  = MIDWIN\_xPos;                 //窗口的起始位置

    WinPara.yPosWin  = MIDWIN\_yPos;                 //窗口的起始位置

    WinPara.xSizeWin = WinPara.xSizeLCD;           //窗口尺寸

    WinPara.ySizeWin = WinPara.ySizeLCD-WinPara.ySizeBotWin-WinPara.yPosWin;    //窗口尺寸

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*设置皮肤色\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    PROGBAR\_SetDefaultSkin(PROGBAR\_SKIN\_FLEX);

    FRAMEWIN\_SetDefaultSkin(FRAMEWIN\_SKIN\_FLEX);

    BUTTON\_SetDefaultSkin(BUTTON\_SKIN\_FLEX);

    CHECKBOX\_SetDefaultSkin(CHECKBOX\_SKIN\_FLEX);

    DROPDOWN\_SetDefaultSkin(DROPDOWN\_SKIN\_FLEX);

    SCROLLBAR\_SetDefaultSkin(SCROLLBAR\_SKIN\_FLEX);

    SLIDER\_SetDefaultSkin(SLIDER\_SKIN\_FLEX);

    HEADER\_SetDefaultSkin(HEADER\_SKIN\_FLEX);

    RADIO\_SetDefaultSkin(RADIO\_SKIN\_FLEX);

    MULTIPAGE\_SetDefaultSkin(MULTIPAGE\_SKIN\_FLEX);

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*设置默认字体\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    \_CreateXBF();

    GUI\_UC\_SetEncodeUTF8();

  GUI\_SetDefaultFont(&XBF\_Font);

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    /\* 创建三个窗口 状态栏、控制栏、主窗口\*/

    CreateBotWin();

    GUI\_Delay(10);

    CreateMidWin();

    CreateTopWin();

    GUI\_Delay(50);

    WM\_BringToTop(WM\_GetDialogItem(WM\_HBKWIN, GUI\_ID\_ICONVIEW0));

    WM\_SetCallback(WM\_HBKWIN, \_cbBkWindow);

    WM\_CreateTimer(WM\_HBKWIN, 0, 250, 0);

    LCD\_BK\_EN;

    while(1)

    {

        GUI\_Delay(5);

    }

}

在UserAPP()中创建了三个窗口，包括状态栏、控制栏和主窗口。

其中，状态栏主要用于显示时间、项目名等信息；底部窗口沿用了野火开机例程自带的三个标签；我们主要修改了主窗口进行本项目应用的加载和显示。

主要用到的数据结构有：

typedef struct WIN\_PARA{            //窗口使用到的用户定义参数

    int xSizeLCD, ySizeLCD;         //LCD屏幕尺寸

    int xPosWin,  yPosWin;          //窗口的起始位置

    int xSizeWin, ySizeWin;         //窗口尺寸

    int xSizeBM,  ySizeBM;

    int ySizeBotWin;

    WM\_HWIN hWinMid;                //主显示窗的句柄（中部）

}WIN\_PARA;

/\* 用于桌面ICONVIEW1图标的创建 \*/

static const BITMAP\_ITEM \_aBitmapItem1[] = {

  {&bmclock,      "Clock"       },

  {&bmclendar,    "Calendar"    },

  {&bmcamera,     "Camera"      },

  {&bmcalculator, "Calculator"  },

  {&bmfly,        "FlyWar"      },

  {&bmsnake,      "Snake"       }

};

在WIN\_PARA中设置窗口的大小和位置，各个图标置于BITMAP\_ITEM的数组中，其中图标均为位图显示，首先通过制图软件做好对应图标的bng图案，再用BmpCvtST将图片转换为位图，图片信息unsigned long格式的数组表示，再用emwin的GUI\_BITMAP格式进行封装，封装代码如下：

GUI\_CONST\_STORAGE GUI\_BITMAP bmclendar = {

  48, // xSize

  48, // ySize

  192, // BytesPerLine

  32, // BitsPerPixel

  (unsigned char \*)\_acclendar, // Pointer to picture data

  NULL,  // Pointer to palette

  GUI\_DRAW\_BMP8888

};

在GUI主函数UserAPP中，通过WM\_SetCallback调用了函数\_cbMidWin作为主界面事件的回调函数，其内容如下：

void \_cbMidWin(WM\_MESSAGE \* pMsg)

{

    int NCode, Id;

    switch (pMsg->MsgId)

    {

        case WM\_NOTIFY\_PARENT:

            Id    = WM\_GetId(pMsg->hWinSrc);   /\* Id of widget \*/

            NCode = pMsg->Data.v;             /\* Notification code \*/

            switch (Id)

            {

                case GUI\_ID\_ICONVIEW1:

                    switch (NCode)

                    {

                        /\* ICON控件点击消息 \*/

                        case WM\_NOTIFICATION\_CLICKED:

                            break;

                        /\* ICON控件释放消息 \*/

                        case WM\_NOTIFICATION\_RELEASED:

                            UserApp\_Flag = 1;

                            /\* 打开相应选项 \*/

                            switch(ICONVIEW\_GetSel(pMsg->hWinSrc))

                            {

                                /\* clock  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                                case 0:

                                    Flag\_ICON100 = 1;

                                    FUN\_ICON100Clicked();

                                    break;

                                /\* secedule \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                                case 1:

                                    Flag\_ICON101 = 1;

                                    FUN\_ICON101Clicked();

                                    break;

                                /\* Camera \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                                case 2:

                                    Flag\_ICON102 = 1;

                                    FUN\_ICON102Clicked();

                                    break;

                                /\* Calculator \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                                case 3:

                                    Flag\_ICON103 = 1;

                                    FUN\_ICON103Clicked();

                                    break;

                                /\* Fly War \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                                case 4:

                                    Flag\_ICON104 = 1;

                                    WM\_HideWindow(WinPara.hWinMid);

                                    WM\_HideWindow(MIDWIN);

                                    WM\_HideWindow(hText);

                                    WM\_HideWindow(iText);

                                    WM\_HideWindow(hFrameClose);

                                    WM\_HideWindow(BOTWIN);

                                    FUN\_ICON104Clicked();

                                    break;

                                /\* snake \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                                case 5:

                                    Flag\_ICON105 = 1;

                                    WM\_HideWindow(WinPara.hWinMid);

                                    WM\_HideWindow(MIDWIN);

                                    WM\_HideWindow(hText);

                                    WM\_HideWindow(iText);

                                    WM\_HideWindow(hFrameClose);

                                    WM\_HideWindow(BOTWIN);

                                    FUN\_ICON105Clicked();

                                    break;

                                default:break;

                                }

                             break;

                        }

                    break;

            }

            break;

        /\* 重绘消息\*/

        case WM\_PAINT:

            {

                GUI\_SetBkColor(MIDWIN\_COLOR);

                GUI\_Clear();

            }

        break;

     default:

        WM\_DefaultProc(pMsg);

        break;

}

}

在此函数中，主要获取了图标按钮的ID，判断点击的按钮id进入与其对应的FUN\_ICON10XClicked()函数打开新窗口并完成相应操作。其中，由于游戏需要全屏进行，故在case4和case5中隐藏了所有窗口。

完成这些内容后，就只需要把上层应用的主函数作为FUN\_ICON10XClicked()的回调函数，即可将其安装在当前的系统上。

# 飞机大战

飞机大战是一款简单有趣的小游戏，游戏目标是操作飞机左右移动来躲避随机出现的陨石。

本游戏是emWin模拟器MinGW\_GUI\_V548自带的模拟器应用，主要是为了测试操作系统运行和内存情况进行移植上来的。游戏的图片和项目图标一样使用了GUI\_BITMAP格式，很有复古的感觉，游戏的主要逻辑都可以直接用C代码进行编写，具体逻辑如下图：

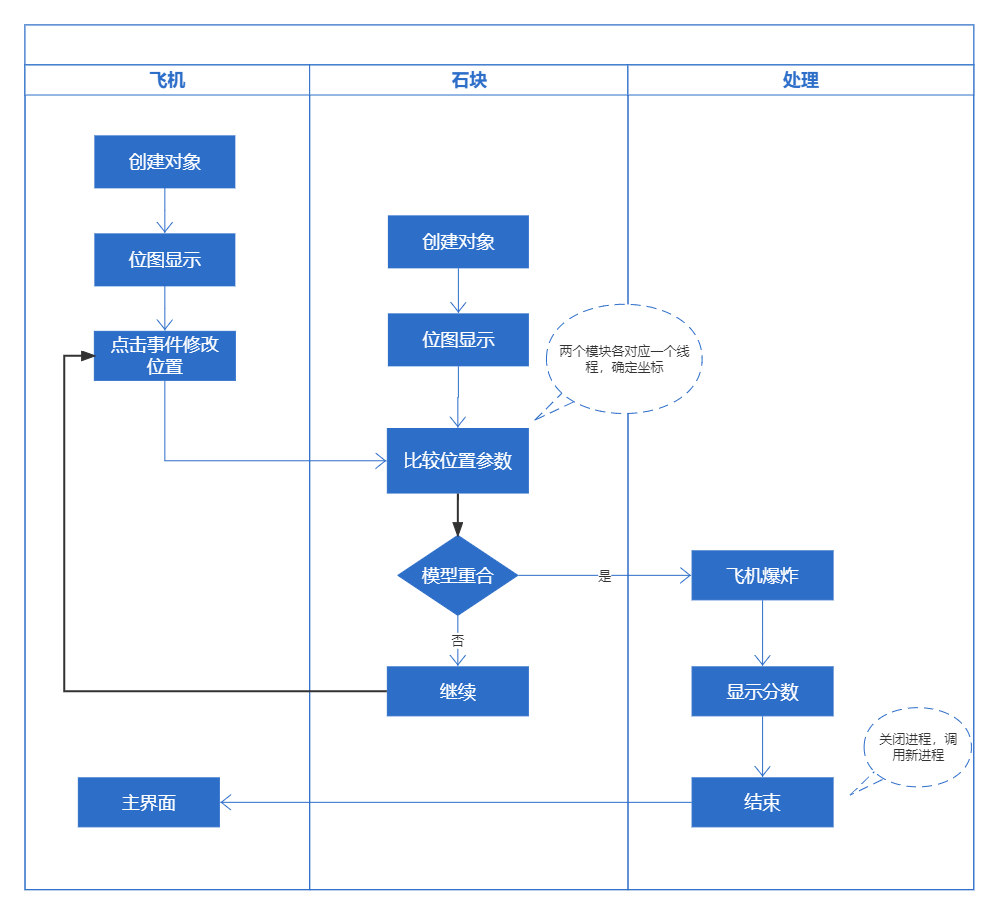


图3 飞机大战逻辑

在模拟器中显示的效果如下图，只需要修改其分辨率与ILI9341显示屏相一致，再将MainTask函数改为对应图标点击后的回调函数，即可移植到本项目中，移植完成后，其核心代码位于app文件夹的AircraftWar.c文件中。

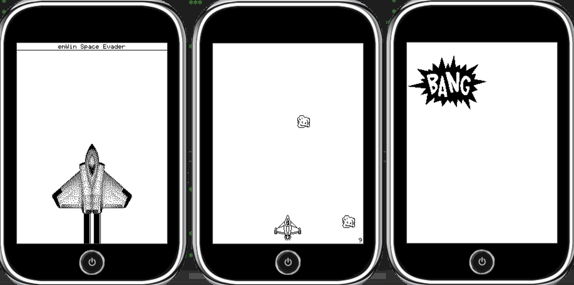


图4 飞机大战模拟效果

# 贪吃蛇

贪吃蛇（也叫做贪食蛇）游戏是一款休闲益智类游戏，有PC和手机等多平台版本。既简单又耐玩。该游戏通过控制蛇头方向吃食物，从而使得蛇变得越来越长。

贪吃蛇作为经典游戏之一，对玩家有着非凡的意义，我们在stm32上结合emwin实现了简易贪吃蛇游戏，增加娱乐项目。其具体流程如下图：

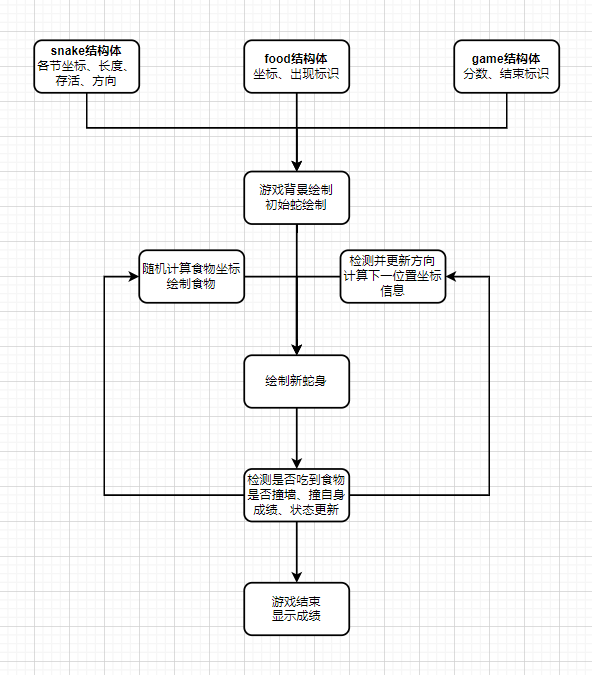


图5 贪吃蛇逻辑

通过预先定义snake蛇身结构体、食物结构体、游戏Game结构体来给出每节蛇身坐标、食物坐标、游戏状态、游戏分数等参数来对游戏数据进行标定；在实际游戏中，通过调用定义的方向判断函数来读取玩家输入目标方向并结合当前行进方向进行目标方向计算，更新蛇身方向；在绘图方面，使用em-Win内置绘图函数接口进行游戏界面绘制、蛇身绘制、食物绘制、游戏参数显示。

整个游戏入口函数通过点击系统主界面贪吃蛇图标进行触发调用。其核心代码位于snake.c中。

游戏仿真演示如下图：

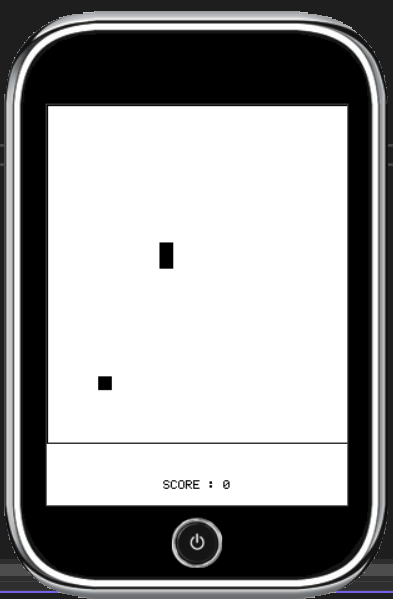


图6 贪吃蛇模拟效果

# 闹钟

轻型智能终端中闹钟功能也是不可或缺的，闹钟功能需要通过输入时间，经过延时，准时的发送提醒消息，以提醒设置的时间到了。其主要显示页面是，输入时间的对话框页面和提醒消息的消息盒子页面。

闹钟功能原理推导：

1. 我们需要一个提醒装置，提醒（时间到）闹钟响了

使用GUI\_MessageBox组件，完成提醒功能。

2、提醒装置需要（延时）时间设置

对MessageBox进行修改添加延时机制。

代码如下：



效果如下：

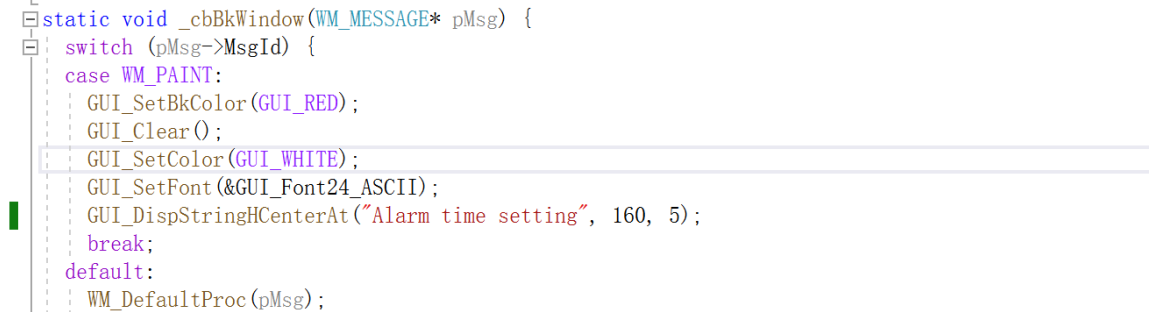
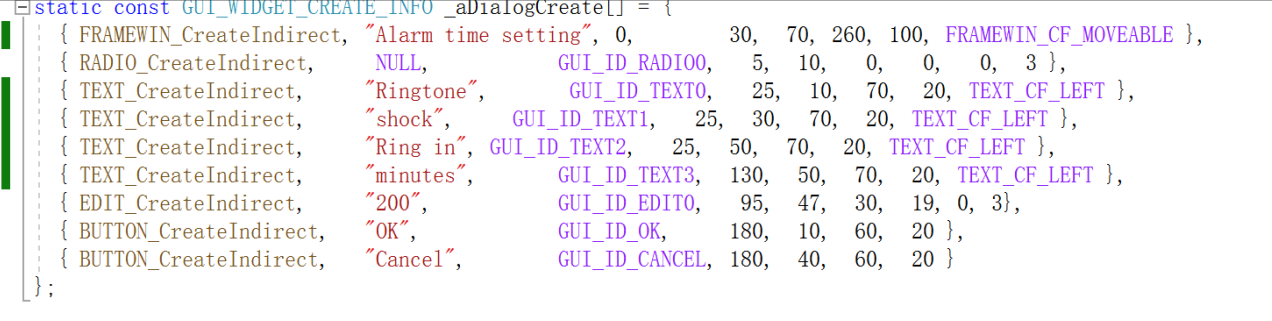


图7 闹钟模拟效果

3、（延时）时间设置的输入

通过对话框传递时间参数

代码如下：



效果如下：



图8 闹钟设置界面

# 日历模块

日历功能是必不可少的功能之一，我们将利用STM32开发板实现日历功能。

再现有包括可穿戴设备在内的嵌入式设备中已有较好实现，我们对其进行复现，并在简单日历实现基础上加入农历、节气、节假日等信息的展示和查阅功能，以满足不同用户群体的更多需求，也进一步在项目中融入我国传统文化元素。

主要包括以下功能：

* 从1900年到2099年的农历查询和2000年到2050年的节气查询
* 支持公农历转换和24节气标注
* 计算甲子年份

在GUIBuilder中效果如下图所示，只需要通过将MainTask函数改为对应图标点击后的回调函数，即可将实现的日历功能移植到我们的项目中。



图9 日历界面

# 便签模块

为满足用户记录事务、标注事务重要性、持续时间等信息事务记录提醒需求，我们设计了便签app。

在功能设计上，对于事项设有增加、删除、修改操作接口；初始实现上我们设计事务项数上限为4，在后续可以实现任意事件的添加及操作。

首先通过GUIBuilder工具搭建初始框架，生成C代码；然后修改代码，写入表格和按键逻辑，实现修改已有事项等功能。逻辑框架如下图所示：

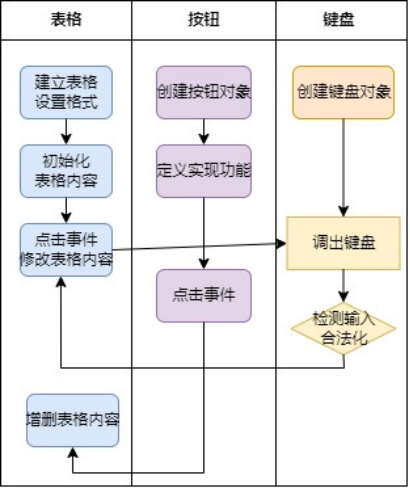


图10 便签逻辑

在GUIBuilder中效果如下图所示，只需要通过将MainTask函数改为对应图标点击后的回调函数，即可将实现的日历功能移植到我们的项目中。

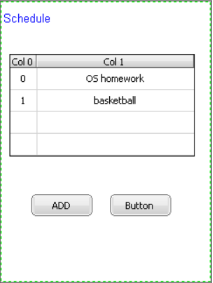
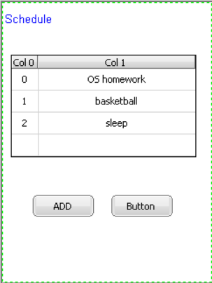
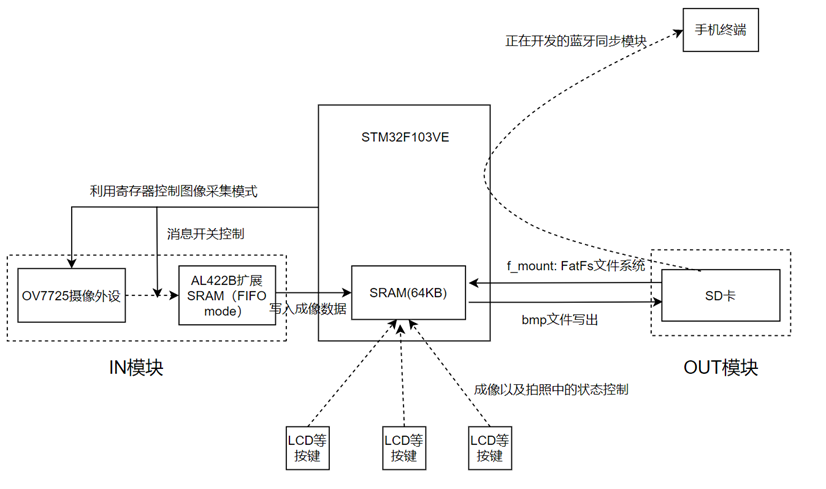


图12 标签界面

# 照相机模块

我们从数据的流动来看，可以把照相机模块的开发分为数据的流入的处理[in]以及数据的流出[out]的处理：

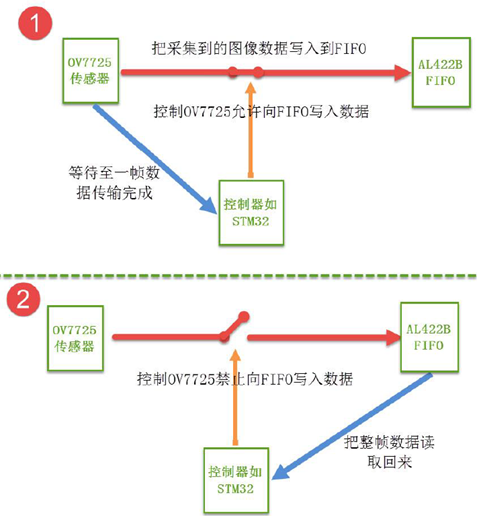
* 在IN模块的开发上，我们使用STM32板上的寄存器控制OV7725的摄像外设的采样模式，比如光照度等参数；利用AL422B作为扩展的SRAM来存储OV7725外设采样到的一帧图片，然后以FIFO的方式读入AL422B的数据
* 在OUT模块上，主要是利用移植的文件系统FatFs来管理SD卡，再利用高层封装的I.O函数按照bmp的格式写出当前LCD屏幕上的图像



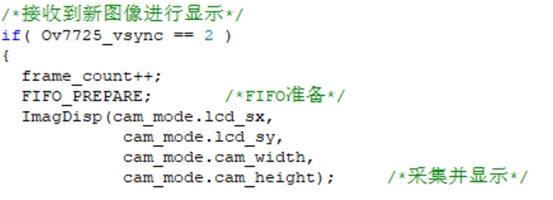
1. **IN模块的开发**

照相机的采集和显示：

* 数据传输与同步：首先底层驱动通过寄存器来控制相应的读偏移以及写偏移；使用下图的方法，通过切断联系，防止OV7725在STM32读取之中写入数据，保证数据的可靠性

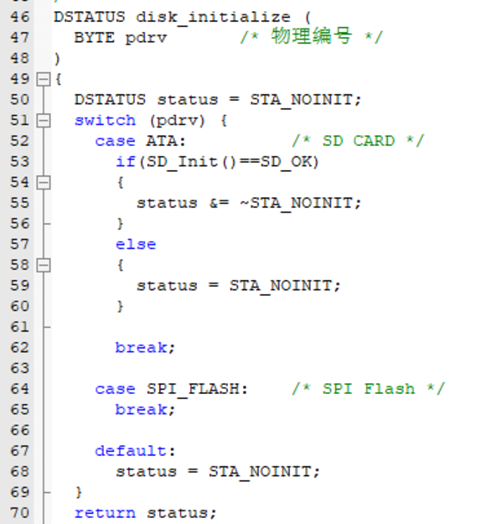
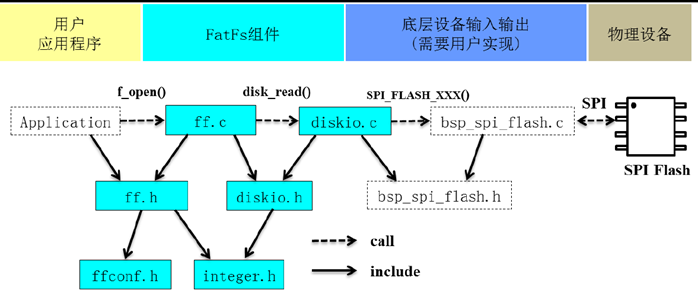


* 采集和显示的逻辑：利用”扩张SRAM”— AL422B（FIFO的读写方式）来对一帧图像进行读写，从而完成数据采集，我们再调用LCD显示的驱动函数，即可完成LCD显示



文件系统的移植

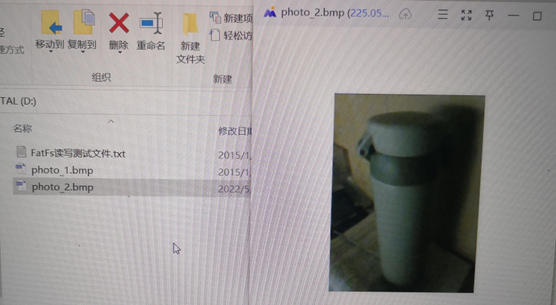
* 与普通的C语言下的I/O的f\_read、fopen不同，由于STM32不能使用这些调用；并自身并没有文件系统，但是管理外加的存储，如果外存本身没有文件系统，那么我们难以管理存储的具体位置，也较难管理不连续存储等问题，故这里我们需要文件系统，在经过文件系统的移植后，我们在f\_mount挂载相应的存储设备后即可调用常规的C语言下的Unix类型的I/O函数，这也为后面的bmp文件的写入打下基础。
* 我们选择FatFs作为SD卡的文件系统，好处在于占用内存小且可以裁剪让我们选择适当的功能，适用于STM32类似的小型嵌入式设备，加载到执行内存上的文件系统对象约占4KB（笔者使用map文件比对过）
* 移植的重点在于编写底层驱动函数（diskio.c 中具有disk\_state, disk\_read需要我们更具外设存储设备的形式进行编写）



1. **OUT模块的开发**

bmp文件写入

* 截取当前的屏幕LCD像素值：锁定当前的屏幕，然后通过读取LCD屏幕的RGB565的驱动调用来得到相应的颜色的像素值，并利用临时变量存储，再利用f\_write调用即可输出
* bmp格式的控制：根据bmp的消息格式，我们按照标准的bmp文件的格式写入相应的比特流即可，注意LCD屏采用的是BGR565，ILI9341\_Read\_Data ()调用



# UI美化设计

在项目设计中，对于我们的应用app界面设计过程，采取了平衡、对比、调和、节奏、变异的思维方式，在保证功能完备的同时也力求简约大气美观，以助美化界面效果增强用户体验。

# 平衡

# 平衡指从视觉反映到心理的一种心理活动，即通过视觉在心理上所达到的一种力的平衡状态。平衡是根据力的重心，将多个形象加以重新配置和调整，从而达到平衡的效果。平衡又可分为绝对平衡和相对平衡。绝对平衡也称为对称、均衡。

# 对称。对称在UI界面设计中表现为以中轴线、中心点为基准，各个元素在大小、形状、排列上具有相对应的关系，形成同形同量的结构。

# 打破对称的平衡。与具有统一构造的对称平衡不同，打破对称的平衡，通过对形象大小、位置的精心配置、重组得到预想的平衡，可以表现出有动感的空间。在UI界面设计中，这种构成比起完全的对称，更富有活力。

# 对比

# 对比是指将不同的质或量形成的强和弱、大和小等相反的东西放置在一起时，产生的区别和差异。由于互相刺激产生了大的显得更大、小的显得更小的视觉效果，起到了使形象更加突出的作用。

# 在实际设计当中，我们用到了背景色与主体色的颜色对比、呈现图标之间的大小对比，以达到突出显示的目的，简洁而重点突出。

# 调和

# 调和即是和谐，指构成画面的各个要素之间的关系，能够适合、安定、和谐一致地配合。和谐强调的是形象的近似性，即两个或者两个以上的元素同时出现时，相互之间有共性。UI界面设计中，设计师需要处理好元素造型特征、色彩、方向的调和。

# 

Figure ：调和设计

# 节奏

# 在音乐中节奏表现为节拍的清晰、高低、强弱、长短，以及优美和谐的曲调，给人愉悦和美感的体验。节奏产生的美感即是韵律。

# 节奏和韵律在UI界面设计中表现为具有一定的秩序性，既有等距离的连续，也有渐变。明暗、形状、高低等的排列构成具有循序渐进的态势。

# 变异

# 变异是对旧秩序的一种突破，在相同性质的形象中，有个别异质性的形象，打破原有的单调格局，立即显示出来。在UI界面设计中，变异的类型有位置的变异、形象的变异、方向的变异、大小的变异、色彩的变异。变异的应用使界面更加活跃，异质造型元素的出现在整体中最具有动感，也称为视觉焦点。

# 总结

# 本次项目实际开发中，我们对项目从底层框架到顶层呈现布局都作了有关思考，其中既有对已有项目的合理参照，也有我们作为开发者对于项目的个人理解与追求。

# 在项目功能设计上，我们既设计了具备实用功能的应用服务，也有对项目娱乐性的改善与提高。设计了日历、闹钟、便签这类具有实际日常生活场景实用功能的软件应用，能够满足用户对于日历查阅、农历节气查询，闹钟提醒，事件添加、提醒的实用需求；也能够满足用户对于娱乐性、娱乐游戏应用的需求，设计开发了飞机大战、贪吃蛇两款简单又不失趣味性的经典游戏，能够给予用户日常生活中的放松项目；此外，我们也利用扩展设备进行了照相机应用的开发，利用开发板的超高扩展性为用户开发了照相机app，能够给予用户更多功能、满足更多元需求。

# 在项目UI美化设计上，我们以简约、实用的理念，融合采取了平衡、对比、调和、节奏、变异的思维方式进行整体设计，做到了保证功能完备的同时也能够给予用户以使用中的美感，改善用户体验。

# 总的来看，在应用设计方面，本次项目给予了我们很多思考，包括对功能、框架的设计，对布局、元素、外观的设计，涵盖项目方方面面，综合性极强，是个挑战，也让我们受益匪浅。