综合实验

1.1 实验目的

根据此前实验所学到的知识，综合运用开发板上的各模块，实现一个相对复 杂的嵌入式系统，实现数据传输，（传感器）数据输入，数据展示的功能。

本小组选择挑战使用一个之前实验未使用过的模块 —— USB 模块来完成综合实验。主要内容是实现一个小游戏：接球游戏。

游戏内容：小球从 LCD 屏幕的最上方的随机位置落下。同一时间，屏幕上只会有一个小球下落。玩家需要按下按键控制板子位置，尝试接住小球。成功接球则加分，失败则减分。

1.2 实验原理

开发板与 USB 键盘建立连接，将键盘按键信息通过 IIC 协议写入 24C02 芯片。在程序主循环中，开发板将不断从 24C02 读取数据，并以此操作游戏元素（板子）。当球落到预定结束位置时，判断此时是否接球成功，并对分数进行更新。

其中游戏逻辑部分负责 LCD 显示屏的控制和游戏运行事件监测，而 USB HID 设备运行库负责初始化 USB HOST，并不断轮询。当检测到 USB 键盘的插入后，监听按键按下，并传递到其他组件上。

1.3 代码修改与说明

由于原 USB 鼠标键盘(Host)实验代码没有添加 24CXX 模块，因此需要自行添加。以下是 `main.c` 文件的内容：

12.main.c

```c
#include "delay.h"
#include "ftl.h"
#include "lcd.h"
#include "malloc.h"
#include "nand.h"
#include "pcf8574.h"
#include "sdio_sdcard.h"
#include "sdram.h"
#include "string.h"
#include "sys.h"
#include "text.h"
#include "usart.h"
#include "usbh_usr.h"

USBH_HOST USB_Host;
USB_OTG_CORE_HANDLE USB_OTG_Core_dev;
extern HID_Machine_TypeDef HID_Machine;

void USBH_Msg_Show(u8 msgx) {
  POINT_COLOR = RED;
  if (!msgx)
    LCD_Fill(0, 150, lcddev.width, lcddev.height, WHITE);
}
// HID 重新连接
void USBH_HID_Reconnect(void) {
  // 关闭之前的连接
  USBH_DeInit(&USB_OTG_Core_dev, &USB_Host); // 复位 USB HOST
  USB_OTG_StopHost(&USB_OTG_Core_dev);       // 停止 USBhost
```

```
  if (USB_Host.usr_cb->DeviceDisconnected)   // 存在，才禁止
  {
    USB_Host.usr_cb->DeviceDisconnected(); // 关闭 USB 连接
    USBH_DeInit(&USB_OTG_Core_dev, &USB_Host);
    USB_Host.usr_cb->DeInit();
    USB_Host.class_cb->DeInit(&USB_OTG_Core_dev, &USB_Host.device_prop);
  }
  USB_OTG_DisableGlobalInt(&USB_OTG_Core_dev); // 关闭所有中断
  // 重新复位 USB
  __HAL_RCC_USB_OTG_FS_FORCE_RESET(); // USB OTG FS 复位
  delay_ms(5);
  __HAL_RCC_USB_OTG_FS_RELEASE_RESET(); // 复位结束
  memset(&USB_OTG_Core_dev, 0, sizeof(USB_OTG_CORE_HANDLE));
  memset(&USB_Host, 0, sizeof(USB_Host));
  // 重新连接 USB HID 设备
  USBH_Init(&USB_OTG_Core_dev, USB_OTG_FS_CORE_ID, &USB_Host, &HID_cb,
            &USR_Callbacks);
}

typedef struct {
  u16 x, y, l, w;
  u8 index;
} rect;

typedef struct {
  u16 x;
  u16 y;
  u16 r;
} circle;

const u16 areax = 460;                      // 显示区域 x 440
const u16 areay = 700 - 20;                 // 显示区域 y700
const u16 edge = 3;                         // 纵向下落区域的间隔
const u16 rr = (areax / 4 - edge * 2) / 2; // 根据宽度计算半径
const u16 rect_area_height = 12;            // 矩形区域高
const u16 rect_area_gap = 2;                // 矩形边界与区域的间隔
const u16 string_pos_y = areay + rr + rect_area_height + 1; // 显示信息的纵坐标

// 转换 index 到横坐标起点，index = 0~3
u16 map_index_x(u16 index) { return 5 + areax / 4 * index + edge; }

void initCircle(circle *c, u8 index) {
  c->x = map_index_x(index) + c->r;
  c->y = edge + c->r;
  c->r = rr;
}
void clear_circle(circle *c) {
  LCD_Fill(c->x - c->r, c->y - c->r, c->x + c->r, c->y + c->r, WHITE);
}
void draw(circle *c) { LCD_Draw_Circle(c->x, c->y, c->r); }
void drop(circle *c) {
  clear_circle(c);
  c->y += 6;
  draw(c);
  delay_us(2000);
```

```
}

u8 map_str_index(u8 *str) { // 将键盘信息映射为整数，方便数组索引
  str[2] = 0;               // 取前两位
  if (strcmp(str, "61") == 0) // A
    return 0;
  else if (strcmp(str, "73") == 0) // S
    return 1;
  else if (strcmp(str, "35") == 0) // Num4
    return 2;
  else if (strcmp(str, "36") == 0) // Num5
    return 3;
  else
    return 4;
}

// 显示得分
void show_score(u32 score) {
  u8 score_buf[10];
  sprintf((char *)score_buf, "%u", score);
  LCD_ShowString(250, string_pos_y, 200, 16, 16, score_buf);
}
// 移动矩形
void rect_move(rect *r) {
  LCD_Fill(r->x, r->y, r->x + r->w, r->y + r->l, WHITE);
  r->x = map_index_x(r->index) + rect_area_gap;
  r->y = areay + rr - 1;
  r->l = rect_area_height - 2 * rect_area_gap;
  r->w = rr * 2;
  LCD_DrawRectangle(r->x, r->y, r->x + r->w, r->y + r->l);
}
void draw_line(u16 y) { LCD_DrawLine(0, y, areax, y); } // 画线

int main(void) {
  Stm32_Clock_Init(384, 25, 2, 8); // 设置时钟，192Mhz
  delay_init(192);                 // 初始化延时函数
  uart_init(115200);               // 初始化 USART
  SDRAM_Init();                    // SDRAM 初始化
  LCD_Init();                      // LCD 初始化
  PCF8574_Init();                  // 初始化 PCF8574
  AT24CXX_Init();
  POINT_COLOR = RED;
  // 初始化 USB 主机
  USBH_Init(&USB_OTG_Core_dev, USB_OTG_FS_CORE_ID, &USB_Host, &HID_cb,
            &USR_Callbacks);

  u32 score = 0;  // 得分
  u8 index = 0;   // 圆形在哪个区域
  _Bool flag = 0; // 当前圆形是否结束
  circle now; // 同一时刻只会有一个圆形；注意不能使用 malloc，至少需要 mymalloc
  rect press_rect;
  u8 buf[4]; // 键盘字符

  show_score(score);
  rect_move(&press_rect);
```

```
  LCD_ShowString(200, string_pos_y, 50, 16, 16, "score");
  LCD_ShowString(10, string_pos_y, 40, 16, 16, "key");
  draw_line(string_pos_y - 2);
  while (1) {
    USBH_Process(&USB_OTG_Core_dev, &USB_Host);
    if (bDeviceState == 1) // 连接建立了
    {
      if (USBH_Check_HIDCommDead(&USB_OTG_Core_dev, &HID_Machine))
        // 检测 USB HID 通信，是否还正常？
        USBH_HID_Reconnect();
    } else // 连接未建立的时候，检测
    {
      if (USBH_Check_EnumeDead(&USB_Host))
        // 检测 USB HOST 枚举是否死机了？死机了，则重新初始化
        USBH_HID_Reconnect();
    }
    AT24CXX_Read(0, (u8 *)buf, 4); // 读取 buf 的值
    LCD_ShowString(50, string_pos_y, 40, 16, 16, buf);
    u8 temp = map_str_index(buf);
    if (temp < 4 && press_rect.index != temp) { // 移动矩形
      press_rect.index = temp;
      rect_move(&press_rect);
    }
    if (!flag) { // 没有圆形则生成圆形
      flag = 1;
      index = rand() % 4;
      initCircle(&now, index);
      draw(&now);
    } else
      drop(&now);
    if (now.y >= areay - rect_area_height + rect_area_gap) { // 触线
      clear_circle(&now);
      flag = 0;
      if (map_str_index(buf) == index)
        ++score;
      else if (score > 0)
        --score;
      show_score(score);
    }
  }
}
```

程序删除了前面的 USB 连接状态信息显示代码，避免了显示的信息影响游戏观感，并添加了 24CXX 模块的初始化代码。

程序在全局空间中定义了两个结构：`circle` 和 `rect`，并带有一点“面向对象”的思想，为这两个结构定义了一些函数，作为复杂操作的抽象。

然后程序使用 `const u16` 定义了一大堆常量，这些常量主要用于界面设计，例如总游戏区域的坐标（`areax, areay`），小球下落柱状区域间隔（`edge`），小球半径（`rr`），矩形区域高度（`rect_area_height`）与间隔（`rect_area_gap`），还有信息显示区域的坐标（`string_pos_y`）。这些常量之间相互耦合，使后续对坐标的处理大大简化。

最后，程序定义了一些映射函数，例如键盘代码映射到柱状区域，柱状区域映射到绝对坐标等，我们可以很方便地使用映射函数进行数据的计算处理。

程序对游戏逻辑的处理主要发生在 main 函数主循环中。

主循环每次都会读取 24C02 芯片存储的键盘按键数据，转换为索引值（index），并将其与矩形的当前位置做比较。如果位置发生了偏移，则移动矩形到新的位置。与历史位置进行比较，差分更新，有利于减少矩形绘制的次数，提高游戏的流畅度。

flag 指示当前场上是否存在小球。如果不存在，则重新生成小球，并且在每一帧过后，移动小球到更下方的新位置，实现小球下落的功能。

最后程序判断小球是否下落到了指定位置。如果下落到预计位置，则执行分数更新程序，程序判断板子是否接到了小球，更新分数，并将新的分数显示在 LCD 屏上。

下面是 usbh_usr.c 的部分代码。原先的代码比较复杂，涉及了字符缓存和区域显示功能，但是本次实验用不到这些功能，因此删除了那些代码，然后手动添加写入 24C02 芯片的程序。

12.usbh_usr.c

```c
#include "24cxx.h"
// ... 省略
void USR_KEYBRD_ProcessData(uint8_t data) {
  u8 buf[4];
  sprintf((char *)buf, "%02X", data);
  AT24CXX_Write(0, (u8 *)buf, 4); // 将 buffer 写入 24CXX
}
```

1.4 实验结果

等待程序初始化完成后，小球从屏幕上方以恒定速度落下。按动键盘的 A S Num4 Num5 键，可以控制板子的位置。当小球落到最下方时，若板子接住小球，分数加一。若未接住，分数减一。下面是游戏运行的截图展示。

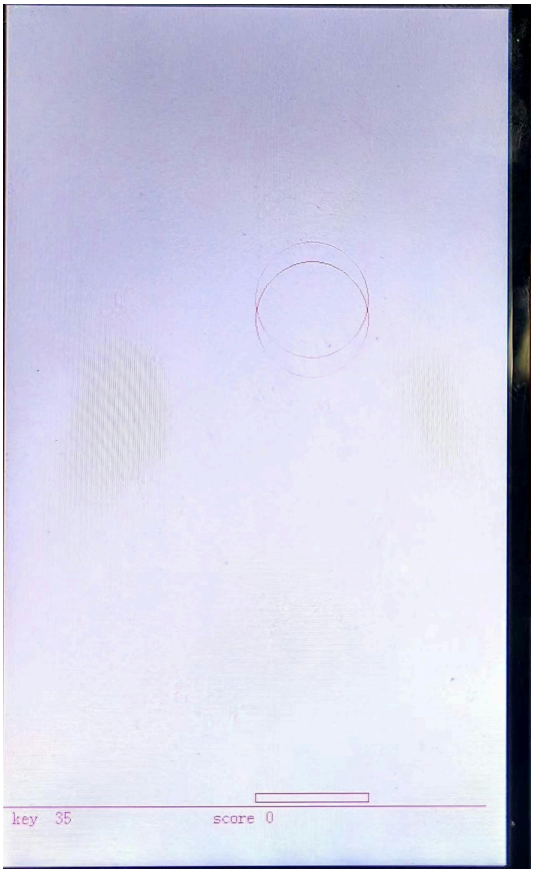

图 **1**　游戏画面

1.5 心得体会

1.5.1 困难

这一次实验花了整整四节课，难度确实还是比较高的。

首先实验的主要部分是 USB 键盘通信，这个在之前的实验中没有做过，并且建立外设通信本身也是一个非常复杂的过程。不过好在原先的代码已经比较成熟，不需要手动改太多东西就能直接投入使用。

其次就是要考虑游戏程序架构。如果将不同元素的动作都做成单独的函数，并且在其中加上延迟效果，则会阻塞其他元素的运动。这一个问题也在之前的实验中多次出现，有了处理经验。因此。本次实验将所有元素都放入主循环，由主循环调函数，统一调度这些元素的运动。

然后一个比较现实的问题就是，USB 按键按下以后通过回调函数（`USR_KEYBRD_ProcessData`）对按键进行处理。这个回调函数类似于 EXTI 外部中断，需要考虑如何把获取到的按键传给主程序，在主循环中进行处理。然后想到了之前做过的 IIC 实验，干脆就使用 24C02 传递按键。

最后也是最大的困难就是如何编写一个游戏的完整逻辑。由于我的 C 语言使用不够熟练。C 语言又是比较底层的语言，没有办法提供太高层的抽象，因此也走了不少弯路。但是我还是尽可能地将不同功能的代码抽象成了函数，结构和全局常量，方便主循环进行后续调用。

1.5.2 遇到的问题

实验中也遇到了不少的问题。

首先是引入其他库。这个操作本应是嵌入式开发的基础，但是之前的实验都不需要自己手动添加，因此并没有实际经验。在上网搜索以后学会了引入库的方法。

1. 将库所在的文件夹复制到项目中的相同结构下。（例如将 `24CXX` 放到 `HARDWARE` 下）
2. 在 Keil 中，右击放入的目录（`HARDWARE`），选择 *Add Existing Files to Group…*，选中库内的 `*.c` 文件。
3. 点开工具栏 `Options for Target`，选择 `C/C++`，在 `Include Paths` 中仿照例子添加库的路径。

然后第一天在改完代码以后，发现每一个文件都能过编译，但是在链接的时候会报错：`Library reports error: __use_no_semihosting was requested, but _ttywrch was referenced`。网上也搜不到解决方法，于是只能慢慢排查。最后发现是调用了 `malloc` 的缘故。然后明白了，嵌入式系统与一般操作系统的系统库不同，在这个项目中应该要调用 `mymalloc` 而不是 `malloc`。

第二天实验中，进行到了插入键盘测试的环节，结果程序一直无法从键盘上读数据。我同样检查了好几遍代码还是没有找到原因，因此只能像上一次那样二分代码排查。最后发现居然是在主循环中使用 `delay_ms` 引起的问题。这个应该涉及到了 USB 协议的底层问题，可能是由于这一个 `delay` 语句把键盘的轮询给阻塞了。这也不是我能解决的，因此我把 `delay` 去掉，改成使用程序本身执行的时间作为延迟。这也最终导致了一个很有意思的现象，就是在最终测试时，键盘如果按的越快，程序没有办法在正常的时间内处理完这些事件，就会导致刷新率变低，小球下落变慢，降低了游戏难度。

最后这个程序其实还有一个 bug 没有解决，就是当分数从 10 以上降低到 10 以下时，分数会从 $+-1$ 的变化幅度变为 $+-10$，并且超出了显示范围。我认为应该是 `sprintf` 函数的具体实现的问题。

1.5.3 可能的改进

我认为在游戏的趣味性和难度方面可以加以改进。

一方面，可以增添更多的游戏元素，例如添加游戏音乐，打击音效，击打特效，背景图等。这些是能够用 STM32F429 开发板实现的，例如音乐播放实验，图像显示实验中都有相应的例子。

另一方面，游戏的难度也能够继续增加。游戏中同一时间只有一个球并不是一个硬性规定。屏幕上完全可以出现多个球。接球的板子只有一个，这意味着两球不能同时落下，必需有依次的顺序关系，这可以用链表/队列实现。