

自主可控嵌入式实验报告

实验信息

实验者：王舒贤 22009290060

实验五同做：王乐山 22009290068

实验一 ARM汇编指令示例调试

实验目的

掌握在鲲鹏云服务器上进行环境配置，搭建ARMv8汇编实验环境

充分理解GNU ARM汇编代码运行环境的搭建、配置及编译运行

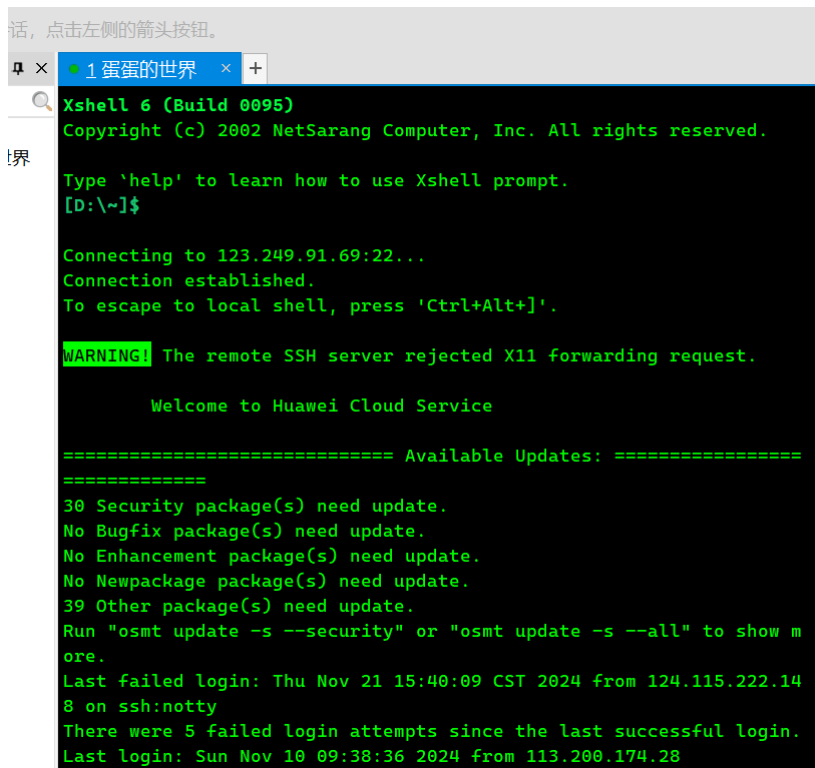
实验内容

通过在华为鲲鹏云服务器上，编译运行3个不同功能的示例程序

- 实现 ARM 平台精简指令集（RISC）编写的 "hello-world" 程序的编译和运行;
- 掌握在ARM平台上使用C语言源码来调用汇编源码的方法;
- 掌握在ARM平台上实现C语言代码中内嵌汇编代码的方法。

实验原理

购买华为鲲鹏云服务器，搭建ARMv8汇编实验环境，用绿化的Xshell+ssh连接远程服务器终端，再用vim编辑器练习编写和编译运行ARM汇编语言代码



```
话，点击左侧的箭头按钮。
X x 1 蛋蛋的世界 x +
Xshell 6 (Build 0095)
Copyright (c) 2002 NetSarang Computer, Inc. All rights reserved.

Type 'help' to learn how to use Xshell prompt.
[D:\~]$

Connecting to 123.249.91.69:22...
Connection established.
To escape to local shell, press 'Ctrl+Alt+J'.

WARNING! The remote SSH server rejected X11 forwarding request.

Welcome to Huawei Cloud Service

===== Available Updates: =====
=====
30 Security package(s) need update.
No Bugfix package(s) need update.
No Enhancement package(s) need update.
No Newpackage package(s) need update.
39 Other package(s) need update.
Run "osmt update -s --security" or "osmt update -s --all" to show more.
Last failed login: Thu Nov 21 15:40:09 CST 2024 from 124.115.222.14
8 on ssh:notty
There were 5 failed login attempts since the last successful login.
Last login: Sun Nov 10 09:38:36 2024 from 113.200.174.28
```

实验步骤

“helloworld”示例程序

步骤1 创建 "hello" 目录

执行以下命令，创建 "hello"目录，存放该程序的所有文件，并进入 "hello" 目录

```
1 | mkdir hello
2 | cd hello
```

步骤2 创建示例程序源码 "hello.s"

```
1 | vim hello.s
```

代码如下：

```
1 | .text
2 | .global tart1
3 | tart1:
4 |     mov x0,#0
5 |     ldr x1,=msg
6 |     mov x2,len
7 |     mov x8,64
8 |     svc #0
9 |     mov x0,123
10 |    mov x8,93
11 |    svc #0
12 | .data
13 | msg:
14 |     .ascii "Hello world!\n"
15 |     len=.-msg
```

解释：

在本例子中，两次使用软中断指令 `svc` 来进行系统调用，系统调用号通过 `x8` 寄存器传递。在第一次使用 `svc` 指令来在屏幕上打印一个字符串 "Hello": `0` 寄存器用于存放标准屏幕输出 `stdout` 描述符 `0`，表明将向屏幕输出一些内容；`x1` 寄存器用于存放待输出的字符串的首地址 `msg`；`x2` 寄存器用于存放待输出字符串的长度 `len`；`x8` 寄存器用于存放系统功能调用号 `64`，即 `64` 号系统功能即系统写功能 `sys_write()`，写的目标在 `x0` 中定义；`svc #0` 表示是一个系统功能调用。

第二次使用 `svc` 指令来退出当前程序：`x0` 寄存器用于存放退出操作码 `123`，不同的退出操作码将对应不同的退出操作；`x8` 寄存器用于存放系统功能调用号 `93`，即 `93` 号系统功能即系统退出功能 `sys_exit()`，退出操作码在 `x0` 中定义；`svc #0` 表示是一个系统功能调用。

在 `.data` 部分，加载 `msg` 和 `len` 实际上使用的是文字池的方法，即将变量地址放在代码段中不会执行到的位置（因为第二次使用 `svc` 指令来退出当前程序之后，是不可能将 `svc #0` 指令之后的内容来当做指令加以执行的），使用时先加载变量的地址，然后通过变量的地址得到变量的值

本代码是 Aarch64 体系结构的汇编代码，需要在 ArmV8 处理器上运行。寄存器 `Xn` 都是 Aarch64 体系结构中的寄存器，`svc` 是 Aarch64 体系结构中的指令

步骤三 编译运行

保存示例源码文件，然后退出 vim 编辑器。在当前目录中依次执行以下命令，进行代码编译运行。观察输出结果。

```
1 as hello.s -o hello.o
2 ld hello.o -o hello
3 ./hello
```

```
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# ls
hello.s
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# vim hello.s
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# as hello.s -o hello.o
hello.s: Assembler messages:
hello.s:14: Error: junk at end of line, first unrecognized character is `h'
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# vim hello.s
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# as hello.s -o hello.o
hello.s: Assembler messages:
hello.s:14: Error: junk at end of line, first unrecognized character is `H'
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# vim hello.s
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]#
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# as hello.s -o hello.o
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# ld hello.o -o hello
ld: warning: cannot find entry symbol _start; defaulting to 00000000004000b0
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# ./hello
Hello World!
```

使用C语言代码调用汇编程序

步骤1 创建目录

执行以下命令，创建called目录存放该程序的所有文件，并进入 "called" 目录。

```
1 mkdir called
2 cd called
```

步骤2 创建 "globalCalling.c" 源代码

执行以下命令，创建示例调用C语言程序源码 "gc.c"。

```
1 vim gc.c
```

代码内容如下：

```
1 /* globalCalling.c */
2 #include <stdio.h>
3 extern void strcpy1(char *d, const char *s);
4 int main()
5 {
6     const char *srcstring="Source string";
7     char dststring[]="Destination string";
8     printf("Original Status: %s %s\n",srcstring,dststring);
9     strcpy1(dststring,srcstring);
10    printf("Modified Status: %s %s\n",srcstring,dststring);
11    return 0;
12 }
```

步骤3 创建gc.s源代码

执行以下代码命令，创建被调用的汇编语言程序源码gc.s

```
1 vim gc.s
```

代码如下：

```
1  /* globalCalled.S */
2  .global strcpy1
3  # Start the function: strcpy1
4  strcpy1:
5  LDRB w2, [x1], #1
6  STR w2, [x0], #1
7  CMP w2, #0 //ascii code "NUL" is the last character of a string;
8  BNE strcpy1
9  RET
```

该汇编代码是针对 Aarch64 架构的。在汇编程序中，用 .global 定义一个全局函数 "strcpy1"，然后该函数就可以在C代码中用 extern 关键字加以声明，然后直接调用

步骤4 编译运行

保存示例源码文件，然后退出vim编辑器。在当前目录中依次执行以下命令，进行代码编译运行。

```
1 gcc gc.c gc.s -o called
2 ./called
```

```
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# mkdir called
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 ~]# cd called
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 called]# vim gc.c
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 called]# vim gc.s
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 called]# gcc gc.c gc.s -o called
gc.s: Assembler messages:
gc.s:5: Warning: unpredictable transfer with writeback -- `str w2, [x2], #1'
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 called]# ./called
Original Status:Source string Destination string
Segmentation fault (core dumped)
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 called]# vim gc.c
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 called]# vim gc.s
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 called]# gcc gc.c gc.s -o called
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 called]# ./called
Original Status:Source string Destination string
Modified Status:Source string Source string
```

通过上述代码运行，可以看出，编写的使用C语言代码调用汇编程序已经在华为鲲鹏云服务器上通过编译和运行，并成功输出结果：

Original Status: Source string Destination string

Modified Status: Source string Source string

使用C语言代码内嵌汇编程序

步骤1 创建目录

执行以下命令，创建builtin目录存放该程序的所有文件，并进入builtin目录。

```
1 mkdir builtin
2 cd builtin
```

步骤2 创建C语言内嵌汇编程序源代码

执行以下命令，创建C语言内嵌汇编程序源码gb.c

```
1 vim gb.c
```

代码如下：

```
1  /* globalBuiltin.c */
2  #include <stdio.h>
3  int main()
4  {
5  int val=0x12345678;
6  __asm__ __volatile__(
7      "mov x3,%1\n"
8      "mov w3,w3, ror #8\n"
9      "bic w3,w3, #0x00ff00ff\n"
10     "mov x4,%1\n"
11     "mov w4,w4, ror #24\n"
12     "bic w4,w4, #0xff00ff00\n"
13     "add w3,w4,w3\n"
14     "mov %0,x3\n"
15     : "=r"(val)
16     : "0"(val)
17     : "w3", "w4", "cc"
18 );
19 printf("out is %x \n",val);
20 return 0;
21 }
```

通过C语言代码内嵌汇编代码，将一个整数类型值，以字节为单位从小尾端转到大尾端或者相反的功能。例如小尾端时 32bit 整数值用16进制表示为 0x12345678，将其以字节为单位转换为大尾端存储后，该值为 0x78563412。

步骤3 编译运行

CTRL + x保存示例源代码，然后退出vim编辑器。在目录中依次执行以下命令，进行代码编译

预处理

```
1 gcc -E gb.c -o gb.i
```

编译

```
1 gcc -S gb.i -o gb.s
```

汇编

```
1 gcc -c gb.s -o gb.o
```

生成可执行文件

```
1 gcc gb.o -o gb
```

```
1 | ./gb
```

```
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 builtin]# vim gb.c
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 builtin]# gcc -E gb.c -o gb.i
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 builtin]# gcc -S gb.i -o gb.s
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 builtin]# gcc -c gb.s -o gb.o
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 builtin]# gcc gb.o -o gb
[root@Server-cb50e404-e351-4a21-ab2a-e19c7fd8fff3 builtin]# ./gb
out is 78563412
```

结果如上

实验er S3C2410基本接口实验

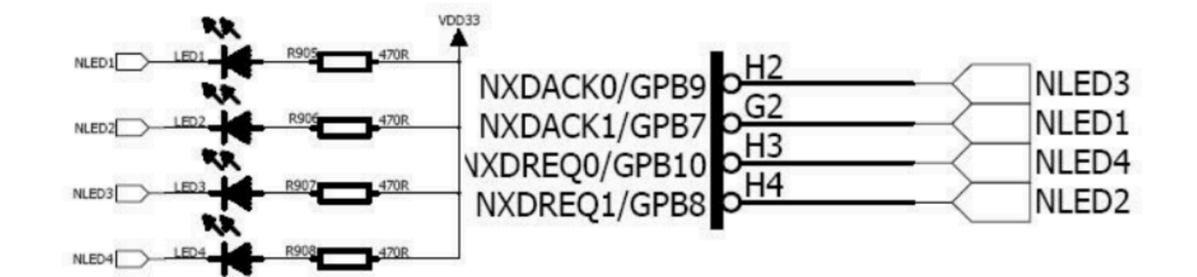
实验目的

- 掌握 S3C2410X 芯片的 I/O 控制寄存器的配置；
- 通过实验掌握 ARM 芯片使用 I/O 口控制 LED 显示；
- 了解 ARM 芯片中复用 I/O 口的使用方法；
- 通过实验掌握键盘控制与设计方法。
- 熟练编写 ARM 核处理器 S3C2410X 中断处理程序。

实验内容

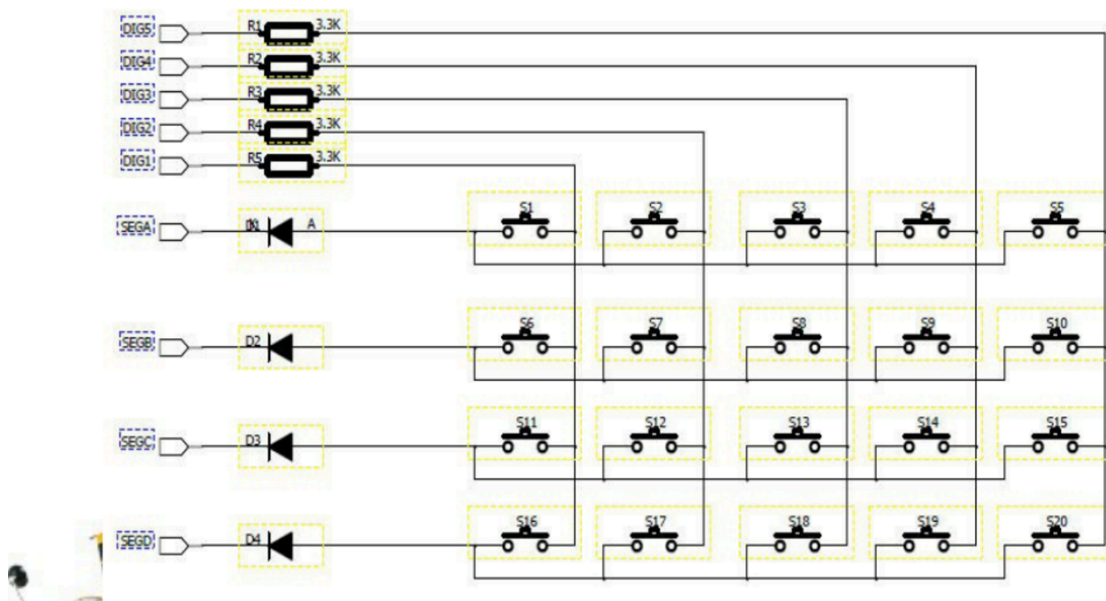
- 编写程序，控制实验平台的发光二极管 LED1、LED2、LED3、LED4，使它们有规律的点亮和熄灭；
- 使用实验板上 5x4 用户键盘，编写程序接收键盘中断；
- 使用键盘控制发光二极管，按照不同模式点亮

实验原理

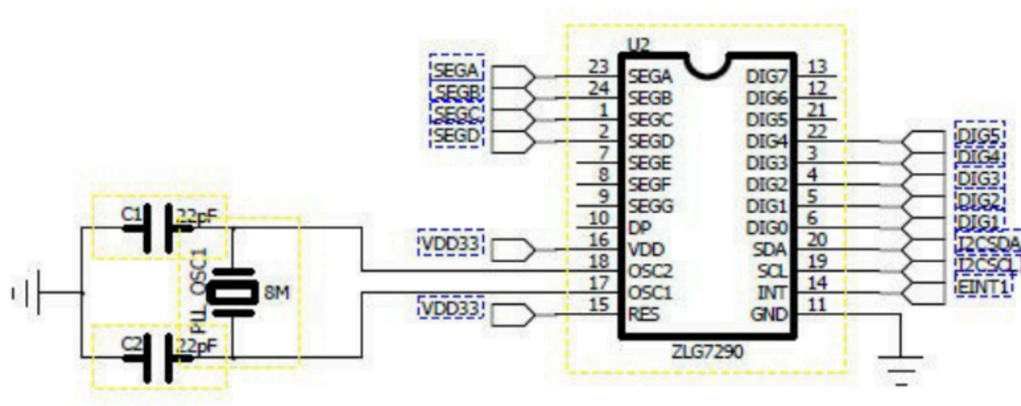


Register	Address	R/W	Description	Reset Value
GPACON	0x56000000	R/W	Configure the pins of port A	0x7FFFFFFF
GPADAT	0x56000004	R/W	The data register for port A	Undefined
Reserved	0x56000008	—	Reserved	Undefined
Reserved	0x5600000C	—	Reserved	Undefined

键盘连接电路



键盘控制电路



实验步骤

1. 验证示例源码

- 拷贝整个实验例程源码目录到本地磁盘自己的工作目录下;
- 使用 μ Vision IDE for ARM 通过 ULINK2 仿真器连接实验板, 打开实验例程目录 02_led_test 子目录下的 led_test.Uv2 例程, 编译链接工程;
- 调试程序, 观察运行结果;
- 打开实验例程目录 12_KeyBoardTest 子目录下的 KeyBoardTest.Uv2 例程, 编译链接工程调试程序, 观察运行结果。

2. 设计实现自己的 I/O 控制程序

- 拷贝示例实验源码工程；
- 设计程序，实现使用键盘控制发光二极管按照不同模式点亮。

实验结果

运行设计的 I/O 控制程序后，摁下实验箱上键盘的“0”、“1”，“2”，“3”按键能分别控制发光二极管按照从左到右依次点亮、从右往左依次熄灭、同时电路然后同时熄灭、先从左到右依次点亮再从右往左依次熄灭再同时点亮然后同时熄灭的不同模式工作。

程序说明

1.在 "KeyBoardTest" 工程中添加 "led_test.c" 源文件

2.在 "keyboard_test.c" 源文件中声明外部函数数"led_on"、"led_off"、"led_on_off"、"led_test"。当调用这几个函数时，发光二极管就会按照不同模式点亮

```
1 extern void led_on(void);
2 extern void led_off(void);
3 extern void led_on_off(void);
4 extern void led_test(void);
```

3.在 "keyboard_test.c" 源文件中修改 "keybord_test.c" 函数，在定义ucChar后增加分支判断，通过判断摁下键盘按键后处理得到的变量 "ucChar" 值决定执行 "led_on"、"led_off"、"led_on_off"、"led_test" 中的哪个。

```
1 if(ucChar=='0') led_on();
2 if(ucChar=='1') led_off();
3 if(ucChar=='2') led_on_off();
4 if(ucChar=='3') led_test();
```

直接用RAM烧录即可

实验三 人机接口实验

实验目的

- 掌握液晶屏的使用及其电路设计方法；
- 掌握 S3C2410X 处理器的 LCD 控制器的使用；
- 通过实验掌握液晶显示文本及图形的方法与程序设计；
- 通过实验掌握触摸屏（TSP）的设计与控制方法。

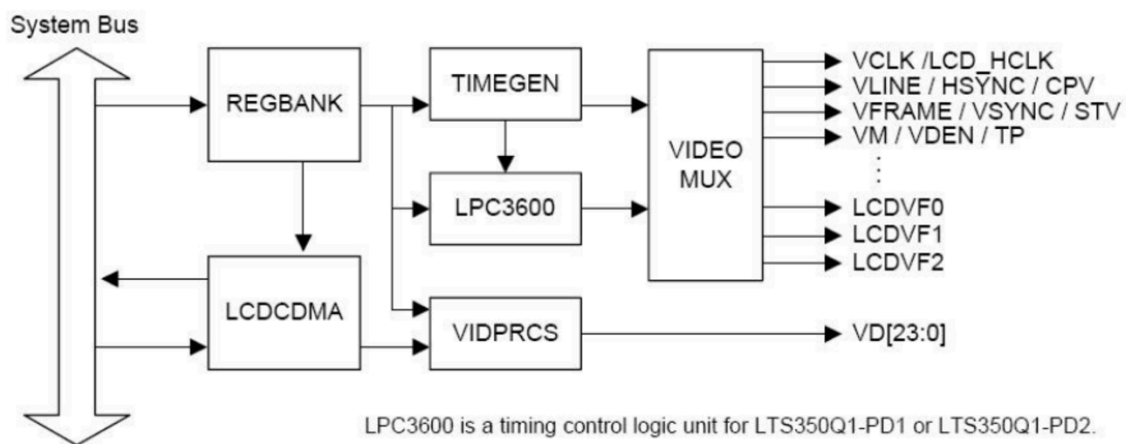
实验内容

- 掌握液晶屏作为人机接口界面的设计方法，并编写程序实现；
- 编程实现触摸屏坐标转换为液晶对应坐标；
- 编程实现由液晶屏和触摸屏构成的可以互动的人机界面，至少实现 3 屏。



- 型号:
- 像素: $320 * 240$
- 模式:

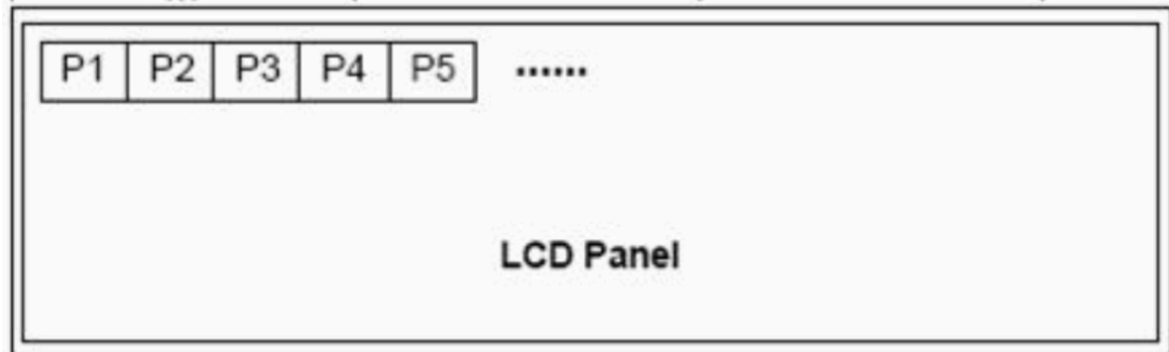
LCD 控制器框图



像素在内存里的表示

(BSWP = 0, HWSWP = 0)

	D[31:16]	D[15:0]
000H	P1	P2
004H	P3	P4
008H	P5	P6
...		



显存的位置

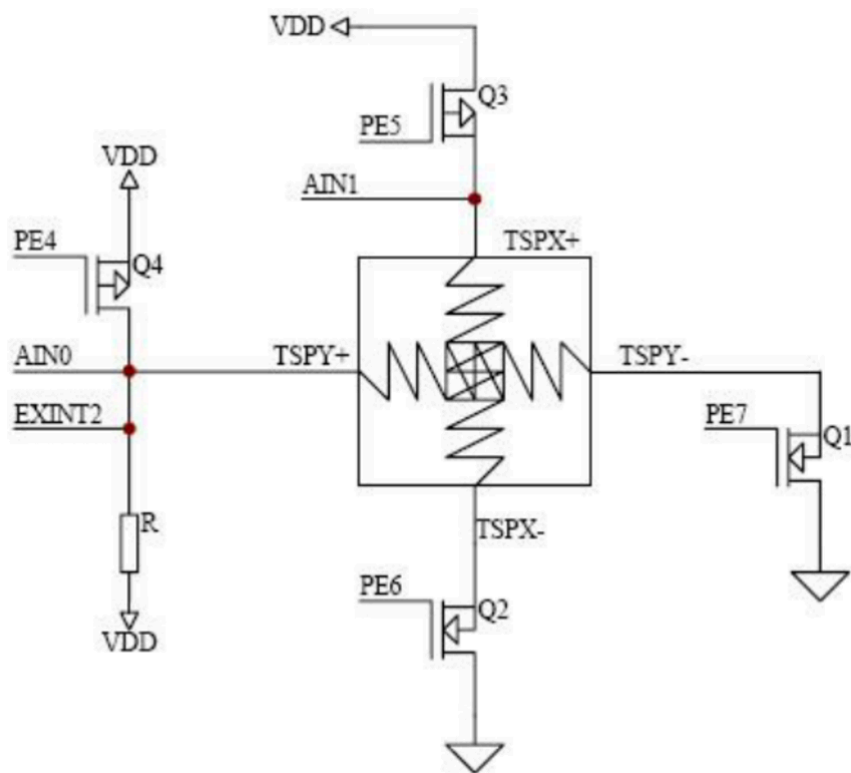
➤ **#define LCDFRAMEBUFFER 0x31000000**

- ❑ 显存位于内存起始地址 (0x3000 0000) 偏移 16M (0x0100 0000) 的地方。

➤ 显存的大小:

- ❑ $320 \times 240 \times 2 = 153600$ 字节

触摸屏的等效电路



实验步骤

1.验证示例源码

- 拷贝整个实验例程源码目录到本地磁盘自己的工作目录下；
- 使用 μ Vision IDE for ARM 通过 ULINK2 仿真器连接实验板，打开实验例程目录 11_LCD_Test
- 子目录下的 LCD_Test.Uv2 例程，编译链接工程；
- 调试程序，观察运行结果；
- 打开实验例程目录 07_TSP_Test 子目录下的TSP_Test.Uv2 例程，编译链接工程；
- 调试程序，观察运行结果。

2.设计实现自己的人机互动界面程序

- 拷贝示例实验源码工程；
- 设计程序，实现由液晶屏和触摸屏构成的可以互动的人机界面，至少实现 3 屏

实验结果

通过触摸屏幕触发要求，这里忘记拍照了，我们假装这里有图

代码说明

1.在 "LCD_Test" 工程中添加 "tsp_test.c" 源文件。同时添加需要的头文件

```
1 #include "2410lib.h"
2 #include "tsp_test.h"
3 #include "lcd.h"
4 #include <string.h>
5 #include <math.h>
6 #include <stdlib.h>
```

2.在 "main.c" 源文件中添加头文件 "sys_init.h" 和 "tsp_test.h" 。并修改 "main" 函数, 调用 "tsp_test" 函数

```
1 #include "sys_init.h"
2 #include "tsp_test.h"
```

```
1 int main()
2 {
3     sys_init();
4     while(1)
5     {
6         tsp_test();
7     }
8 }
```

3.修改 "tsp_test.c" 源文件中的 "__irq tsp_int" 中断处理函数, 在其末尾调用编写的 "my_lcd_test" 函数

```
1 my_lcd_test();
```

4.编写 "my_lcd_test" 函数。点击触摸屏时通过判断变量 "lcd_state" 和 "g_nPosY" 的值决定应该切换到哪个屏。切换时通过调用 "Lcd_Clear" 函数改变屏幕背景色, 调用 "Lcd_DspASCII8X16" 函数在屏幕上显示颜色与背景色不同的提示性英文字符串, 并改变变量 "lcd_state" 的值

```
1 int lcd_state=0;//状态指示
2 void my_lcd_test(void)
3 {
4     if(lcd_state==0)
5     {
6         Lcd_Clear(0x0a);//刷新
7         Lcd_DspASCII8X16(50,100,0x7e0,"win1");//屏幕显示 (x坐标, y坐标, 颜色, 显示
        字符)
8         Lcd_DspASCII8X16(200,100,0x7e0,"win2");
9         Lcd_DspASCII8X16(100,200,0x1f,"win3");
10        delay(100);//延迟1s
11        lcd_state=1;
12    }
13    else if(lcd_state==1)
14    {
15        if(g_nPosY<500)
16        {
17            Lcd_Clear(0x7e0);
18            Lcd_DspASCII8X16(100,100,0x1f,"test_0");
19        }
20        else
21        {
22            Lcd_Clear(0x7e0);
23            Lcd_DspASCII8X16(100,100,0x1f,"test_1");
24        }
25        delay(100);
26        lcd_state=0;
27    }
28    else lcd_state=0;
29 }
```

代码完成以后通过RAM烧录进去就可以运行，触摸屏幕，不同位置触发自己写的不同提示词

总结

通过这次实验，我进一步增强了使用 集成开发环境进行工程开发的能力，并通过自己实现一个 简陋的人机交互接口对嵌入式系统的设计过程有了更深入的了解

实验四 $\mu\text{C}/\text{OS-II}$ 系统原理实验

实验目的

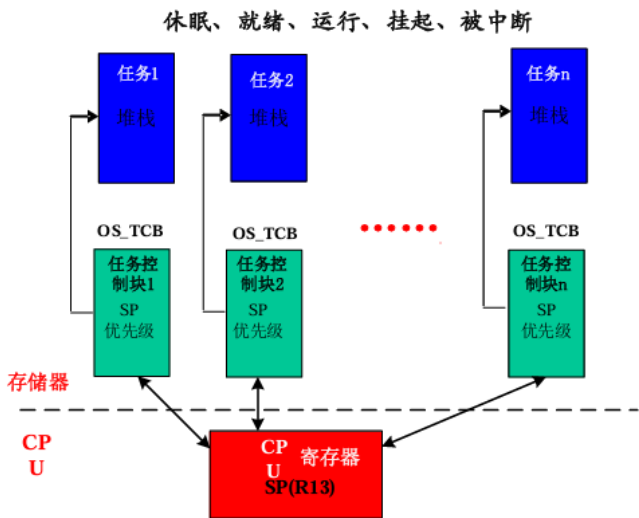
理解任务管理的基本原理，了解任务的各个基本状态及其变迁过程；
掌握 $\mu\text{C}/\text{OS-II}$ 中任务管理的基本方法：创建、启动、挂起、解挂任务
掌握 $\mu\text{C}/\text{OS-II}$ 中任务使用信号量的一般原理

实验内容

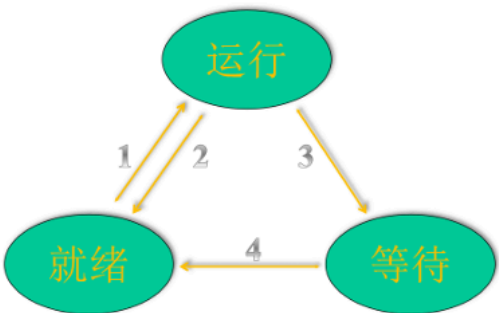
设计多个应用任务，验证任务管理函数；
通过实现“哲学家就餐”问题，验证信号量对任务间互斥访问的支持；
应用信号量实现任务间的同步，设计 7 个任务，并顺序执行

实验原理

1. 任务管理的基本状态



2. 任务的状态切换



实验步骤

1. 验证示例源码

- 拷贝整个实验例程源码目录到本地磁盘自己的工作目录下；
- 使用μVision IDE for ARM 通过ULINK2仿真器连接实验板，打开实验例程目录04-ucOS\2.1_Task_test 子目录下的 ucos2.Uv2 例程，编译链接工程；
- 将程序下载到实验平台的 NorFlash 中，观察串口输出；
- 打开实验例程目录\04-ucOS\2.3_Semaphore_test 子目录下的 ucos2.Uv2 例程，编译链接工程；
- 下载调试，观察结果。

2. 设计实现一个多任务应用程序

- 拷贝示例实验源码工程；
- 设计7个任务，并用信号量实现7个任务顺序执行，将执行结果在串口 上打印出来。

实验结果

通过将源码编译并用Flash下载，在超级终端使用COM4串口查看串口 输出信息，预期的输出结果应如下：

```
1  Emboard Emsbc2410 boot success!  //忘记拷源码了，核心意思不变
2  uCOS-II Running...  //这里的显示可以更改，原本是5个哲学家就餐改7个
3  Now task1 is running!  //哲学家1 eating!
4  Task1 resumed task2!  //哲学家1 finish eating!
5  Now task2 is running!
6  Task2 resumed task3!
7  Now task3 is running!
8  Task3 resumed task4!
9  Now task4 is running!
10 Task4 resumed task5!
11 Now task5 is running!
12 Task5 resumed task6!
13 Now task6 is running!
14 Task6 resumed task7!
15 Now task7 is running!
16 Task7 resumed task1!
17 Now task1 is running!
18 Task1 resumed task2!
```

说明

本实验程序基于给定的示例工程 Semaphore_test，进行自主设计修改，程序的核心代码如下：

1. 定义指向 OS_EVENT 类型的指针，sem1~sem7 控制任务之间的同步关系；初始化每个人所需要的静态堆栈TestTask1Stk——TestTask7Stk，同时声明7个任务函数Task1~Task7，之后生成每个任务所对应的同步信号量，sem1~sem7，并且实验要求是顺序执行，因此使得所有同步信号量的初值只有一个1，即任务一先运行。
2. 创建7个并发任务，并用创建的信号量实现任务之间的同步；其中，任务一首先运行，然后会唤醒任务二，之后的每个任务依次唤醒下一个任务，直到任务七会再次唤醒任务一，整个流程会形成循环顺序执行。

总结

这是一个简单演示实验，基于这次实验，我对操作系统的知识有了更深刻的理解，在实验中以 C/OS II 的内核结构和任务管理为内容，着重掌握了如何利用信号量PV机制实现任务通信。此次实验经历让我为接下来的实验有了充分的准备和坚实的基础

实验五 μ C/OS-II硬件接口实验

实验目的

理解任务管理的基本原理，掌握 μ COS-II中任务管理的基本方法；
掌握 μ COS-II中任务间通信的一般原理和方法；
掌握嵌入式系统中LCD与键盘控制的一般方法。

实验内容

设计多个应用任务，用其中一个任务控制LED灯的状态；
设计多个应用任务，用其中一个任务读取键盘键值，键盘的响应用中断实现；
实现一个简易的计算器，用LCD显示。

实验过程和代码解析

要了解整个代码含义才能更好的理解和完成整个问题，下面是代码详解

Main.c

全局变量和信号量

```
1 OS_EVENT *UART_sem;  
2 OS_EVENT *REFRESH_sem;  
3 OS_EVENT *InterruptSem;//三个同步信号量  
4  
5 unsigned char err;//存储信号量操作的错误码
```

任务栈的定义

```
1 #define STACKSIZE 256//每个任务栈大小  
2 //下面是栈空间  
3 OS_STK Stack1[STACKSIZE];//任务1  
4 OS_STK Stack2[STACKSIZE];//任务2  
5 OS_STK Stack3[STACKSIZE];//任务3  
6 OS_STK StackMain[STACKSIZE];//启动任务
```

LED控制函数

```
1 void led_init()//初始化LED控制引脚为输出模式。  
2 {  
3     rGPBCON |= ((0x1<<14) | (0x1<<16) | (0x1<<18) | (0x1<<20));  
4     rGPBCON &= ~( (0x1<<15) | (0x1<<17) | (0x1<<19) | (0x1<<21));  
5 }  
6  
7 void led_off()//关闭所有LED  
8 {  
9     rGPBDAT |= ((0x1<<7) | (0x1<<8) | (0x1<<9) | (0x1<<10));
```

```

10 }
11
12 void led_on()//打开所有LED
13 {
14     rGPBDAT  &= ~(0x1<<7) | (0x1<<8) | (0x1<<9) | (0x1<<10));
15 }

```

任务函数

```

1 void Task1(void *Id)//处理键盘输入
2 {
3     while(1)
4     {
5         keyboard_fun_swq(); //REFRESH_sem在这释放
6     }
7 }
8
9 void Task2(void *Id)//每隔50个ticks切换一次LED状态（氛围感灯光！）
10 {
11     while(1)
12     {
13         led_on();
14         OSTimeDly(50);
15         led_off();
16         OSTimeDly(50);
17     }
18 }
19
20 void Task3(void *Id)//等待 REFRESH_sem 信号量，然后执行LCD测试函数
21 {
22     while(1)
23     {
24         OSSemPend(REFRESH_sem, 0, &err); //这步作用是为了阻塞函数运行，避免无意义的刷新和渲染
25         color_lcd_test();
26     }
27 }

```

启动任务

```

1 void TaskStart(void *Id)
2 {
3     Init_Timer4(); //初始化定时器
4
5     //创建信号量
6     UART_sem = OSSemCreate(1);
7     REFRESH_sem = OSSemCreate(0); //初始化为0
8
9     //创建三个任务
10    OSTaskCreate(Task1, (void *)&Id1, &Stack1[STACKSIZE - 1], 3);
11    OSTaskCreate(Task2, (void *)&Id2, &Stack2[STACKSIZE - 1], 2);
12    OSTaskCreate(Task3, (void *)&Id3, &Stack3[STACKSIZE - 1], 1);
13
14    OSTaskDel(OS_PRIO_SELF); //删除自身任务
15

```


主函数

```

1 void Main(void)
2 {
3     change_clock_divider(1,1);
4     change_value_MPLL(M_MDIV, M_PDIV, M_SDIV);
5     delay(0);
6
7     port_init();
8     led_init();
9     uart_init(PCLK, 115200, UART0);
10    uart_select(UART0);
11    uart_sendstring("\r\nEmboard Emsbc2410 boot success!\r\n");
12    uart_sendstring("uCOS-II Running...\r\n");
13    keyboard_init();
14    color_lcd_init();
15
16    OSInit();
17    OSTimeSet(0);
18
19    OSTaskCreate(TaskStart, (void *)0, &StackMain[STACKSIZE - 1], 0);
20
21    OSStart();
22
23 }
```

- `Main` 函数初始化系统时钟、端口、LED、UART、键盘和LCD。
- 初始化uC/OS-II操作系统，设置系统时间，创建启动任务，并启动操作系统。

keyboard.c

函数声明

```

1 void keyboard_test(void); //测试键盘功能
2 void __irq OS_BUTTON_ISR(void); //键盘中断服务程序
3 UINT8T key_set(UINT8T ucChar); //将键盘扫描码映射实际字符
```

全局变量

```

1 UINT32T g_nKeyPress = 0; //标志键盘按键事件
2 int x, y, result0;
3 unsigned char op, flag, mark;
4 int fun_step = 0;
5 //存储历史输入值
6 int x_l[NUM] = {0, 0, 0, 0, 0};
7 int y_l[NUM] = {0, 0, 0, 0, 0};
```

键盘初始化

```
1 void keyboard_init(void)
2 {
3     iic_init_keybd();
4     x = 0;
5     y = 0;
6     flag = 0;
7 }
```

键盘测试函数

```
1 void keyboard_test(void)
2 {
3     UINT8T ucChar;
4     if (g_nKeyPress == 1)
5     {
6         g_nKeyPress = 0;
7         iic_read_keybd(0x70, 0x1, &ucChar);
8         if (ucChar != 0)
9         {
10             ucChar = key_set(ucChar);
11             if (ucChar < 10) ucChar += 0x30;
12             else if (ucChar < 16) ucChar += 0x37;
13             if (ucChar < 255)
14             {
15                 uart_sendstring("press key ");
16                 uart_sendstring(&ucChar);
17                 uart_sendstring("\r\n");
18                 // 处理按键输入
19             }
20             if (ucChar == 0xFF)
21             {
22                 uart_sendstring(" press key FUN (exit now)\n\r");
23                 return;
24             }
25         }
26     }
27 }
```

键盘功能函数

```
1 void clear_para(void)//清除计算参数
2 {
3     int i;
4     for (i = 0; i < NUM - 1; i++)
5     {
6         op = '\0';
7         mark = '\0';
8         x_l[i] = 0;
9         y_l[i] = 0;
10        result0 = 0;
11    }
12    mark = '\0';
```

```

13 }
14
15 void restore(void)//保存历史输入值
16 {
17     int i;
18     for (i = NUM - 1; i > 0; i--)
19     {
20         x_l[i] = x_l[i - 1];
21         y_l[i] = y_l[i - 1];
22     }
23 }
24
25 void recover(void)//恢复历史输入值
26 {
27     int i;
28     for (i = 0; i < NUM - 1; i++)
29     {
30         x_l[i] = x_l[i + 1];
31         y_l[i] = y_l[i + 1];
32     }
33 }
34
35 void keyboard_fun_swq(void)//处理键盘输入
36 {
37     UINT8T ucChar;
38     if (g_nKeyPress == 1)
39     {
40         g_nKeyPress = 0;
41         iic_read_keybd(0x70, 0x1, &ucChar);
42         if (ucChar != 0)
43         {
44             ucChar = key_set(ucChar);
45             if (ucChar < 10) ucChar += 0x30;
46             else if (ucChar < 16) ucChar += 0x37;
47             uart_printf("fun_step:%d\n", fun_step);
48             if (ucChar < 255)
49             {
50                 uart_sendstring("press key ");
51                 uart_sendstring(&ucChar);
52                 uart_sendstring("\r\n");
53                 // 处理按键输入
54             }
55             if (ucChar == 'A')
56             {
57                 fun_step = 0;
58                 clear_para();
59                 color_lcd_init();
60             }
61             else if (ucChar == 'B') restore();//保存历史输入值
62             else if (ucChar == 'C') recover();//返回上一步直到历史输入值
63             else if (ucChar == 'D') fun_step = 1;//标志设置1, 返回计算第二步, 得
出结果以后可重新修改第二步值
64             else if (ucChar == 'E');//没设计, 可以设计个除数什么的
65             else if (ucChar == 0xFF)
66             {
67                 uart_sendstring(" press key FUN (exit now)\n\r");

```

```

68         clear_para();
69         return;
70     }
71     OSSemPost(REFRESH_sem);
72 }
73 }
74 }

```

中断服务程序

```

1 void __irq OS_BUTTON_ISR(void)
2 {
3     rINTMSK |= (BIT_EINT1);
4     ClearPending(BIT_EINT1);
5     g_nKeyPress = 1;
6     rINTMSK &= ~(BIT_EINT1);
7 }

```

按键映射函数

```

1  UINT8T key_set(UINT8T ucChar)
2  {
3      switch (ucChar)
4      {
5          case 1:
6          case 2:
7          case 3:
8          case 4:
9          case 5:
10         ucChar -= 1; break;
11         case 9:
12         case 10:
13         case 11:
14         case 12:
15         case 13:
16         ucChar -= 4; break;
17         case 17:
18         case 18:
19         case 19:
20         case 20:
21         case 21:
22         ucChar -= 7; break;
23         case 25: ucChar = 0xF; break;
24         case 26: ucChar = '+'; break;
25         case 27: ucChar = '-'; break;
26         case 28: ucChar = '*'; break;
27         case 29: ucChar = 0xFF; break;
28         default: ucChar = 0;
29     }
30     return ucChar;
31 }

```

lcdapp.c

这边主要只看几个部分

LCD初始化函数

```
1 void color_lcd_init(void)//初始化LCD，设置显示模式，清屏，并显示一些初始文本和图形
2 {
3     Lcd_port_init();
4     Lcd_Init_16Bit_320240();
5     Lcd_Clear(0xab);
6     Lcd_PowerEnable(0, 1);
7     Lcd_EnvidOnOff(1);
8     Lcd_DspAscII6x8(10, 10, 0x7e0, "22009290068");
9     Lcd_DspAscII6x8(10, 20, 0x7e0, "22009290060");
10    Lcd_DspHz24(200, 10, 0x7e0, "王乐山 ");
11    Lcd_DspHz24(100, 10, 0x7e0, "王舒贤");
12    Lcd_Draw_Line(0, 40, 320, 40, 0x1f, 5);
13    Lcd_DspAscII6x8(20, 50, 0x7E0, "FUNCTION");
14    Lcd_DspAscII6x8(20, 70, 0xf800, "A CLEAR");
15    Lcd_DspAscII6x8(20, 100, 0xf800, "B RESTORE");
16    Lcd_DspAscII6x8(20, 130, 0xf800, "C RECOVER");
17    Lcd_DspAscII6x8(20, 160, 0xf800, "D RECOUNT");
18    Lcd_DspAscII6x8(20, 190, 0xf800, "F OUTPUT");
19 }
```

这个相当于封面，可以自己随便改，不用和计算机计算界面一样

LCD测试函数

字符串转换函数

```
1 unsigned char *exchr(UINT32T num)
2 {
3     if (mark == '-')
4     {
5         str[pos] = mark;
6         pos++;
7         while (num != 0)
8         {
9             str[pos] = num % 10 + '0';
10            num /= 10;
11            pos++;
12        }
13        str[pos] = '\0';
14        for (i = 1; i <= pos / 2; i++)
15        {
16            temp = str[i];
17            str[i] = str[pos - i];
18            str[pos - i] = temp;
19        }
20        pos = 0;
21    }
22    else if (mark == '+')
23    {
24        while (num != 0)
```

```

25     {
26         str[pos] = num % 10 + '0';
27         num /= 10;
28         pos++;
29     }
30     for (i = 0; i < pos / 2; i++)
31     {
32         temp = str[i];
33         str[i] = str[pos - i - 1];
34         str[pos - 1 - i] = temp;
35     }
36     pos = 0;
37 }
38 return str;
39 }
40
41 unsigned char *exch(UINT32T num)
42 {
43     while (num != 0)
44     {
45         str[pos] = num % 10 + '0';
46         num /= 10;
47         pos++;
48     }
49     str[pos] = '\0';
50     for (i = 0; i < pos / 2; i++)
51     {
52         temp = str[i];
53         str[i] = str[pos - 1 - i];
54         str[pos - 1 - i] = temp;
55     }
56     pos = 0;
57     return str;
58 }

```

计算机界面函数

```

1 void color_lcd_test(void)
2 {
3     Lcd_Clear(0x00);
4     Lcd_DspAscII6x8(10, 10, 0x7e0, "22009290068");
5     Lcd_DspAscII6x8(10, 20, 0x7e0, "22009290060");
6     Lcd_DspHz24(200, 10, 0x7e0, "王乐山 ");
7     Lcd_DspHz24(100, 10, 0x7e0, "王舒贤");
8     Lcd_Draw_Line(0, 40, 320, 40, 0x1f, 5);
9     Lcd_DspAscII6x8(20, 50, 0x7E0, "FUNCTION");
10    Lcd_DspAscII6x8(20, 70, 0xf800, "A CLEAR");
11    Lcd_DspAscII6x8(20, 100, 0xf800, "B RESTORE");
12    Lcd_DspAscII6x8(20, 130, 0xf800, "C RECOVER");
13    Lcd_DspAscII6x8(20, 160, 0xf800, "D RECOUNT");
14    Lcd_DspAscII6x8(20, 190, 0xf800, "F OUTPUT");
15    Lcd_DspAscII8x16(100, 80, 0x7e0, exch(x_l[0]));
16    op1[0] = op;
17    op1[1] = '\0';
18    Lcd_DspAscII8x16(100, 100, 0x7e0, op1);

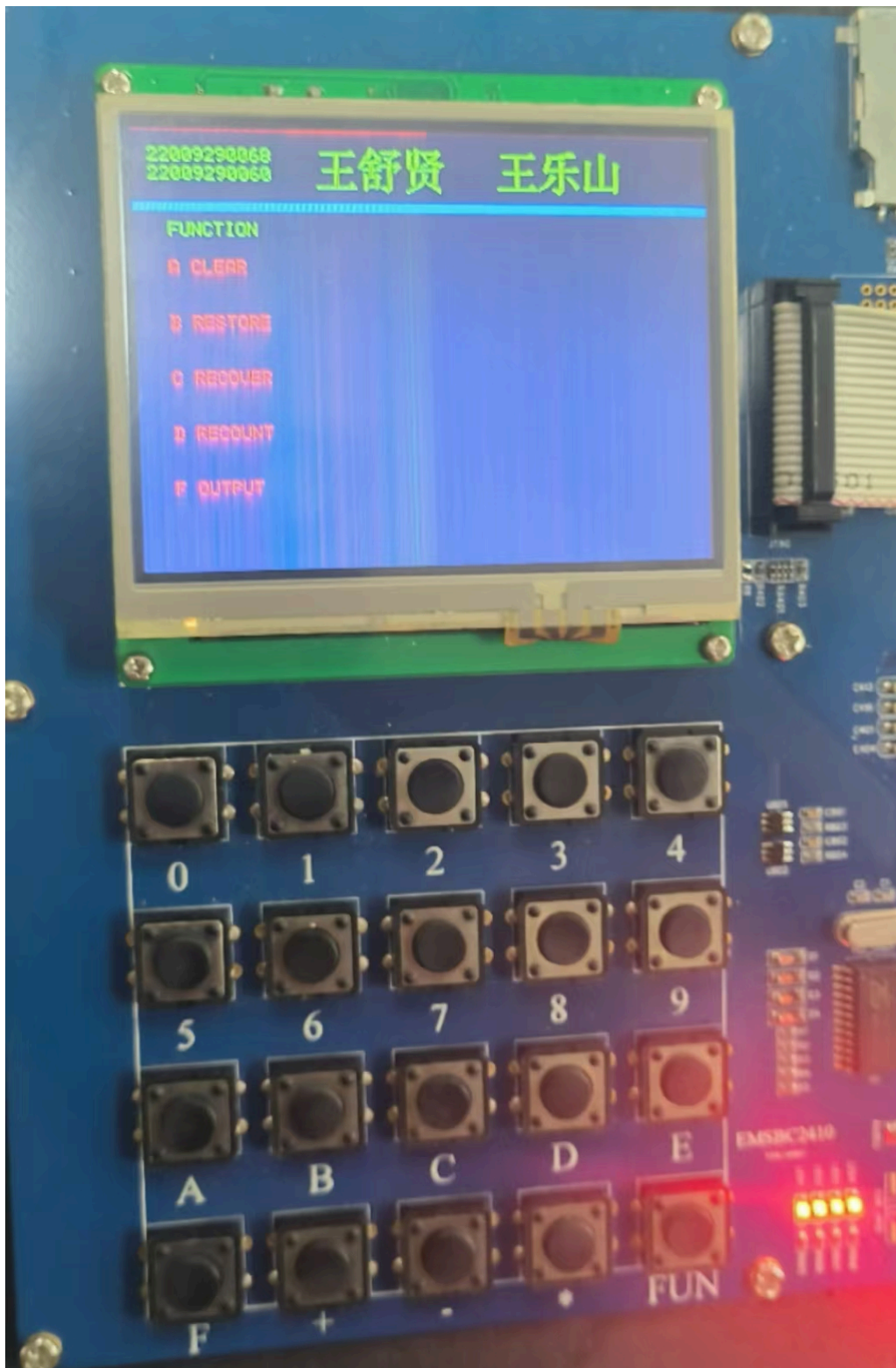
```

```

19     Lcd_DspAscii8X16(100, 120, 0x7e0, exch(y_1[0]));
20     Lcd_DspAscii8X16(100, 140, 0x7e0, "=");
21     OSTimeDly(10);
22     if (mark == '-')
23     {
24         Lcd_DspAscii8X16(90, 160, 0x7e0, "-");
25     }
26     Lcd_DspAscii8X16(100, 160, 0x7e0, exch(result0));
27 }

```

最终结果



总结

通过这次实验，我综合运用课堂上所学的知识 and 前几次实验的经验，实现了一个相对简单的计算器，锻炼了自己编写多任务嵌入式程序的能力，可谓收获颇丰