东南大学

嵌入式系统设计　实验报告

姓名：殷春锁　　学号：09018117

姓名：王宇　　学号：09018122

姓名：曹思辰　　学号：09118113

时间：2020 年 12 月

一、 实验 1 GPIO 实验

1. 实验目的

- 掌握采用直接寄存器访问模式操作外设；
- 掌握系统时钟主要寄存器功能及其配置流程；
- 掌握 GPIO 外设主要寄存器功能及配置流程；
- 掌握写入 GPIO 引脚状态的方法。

2. 实验内容

- 采用直接访问寄存器模式实现系统时钟和 GPIO 的配置及初始化；
- 编程实现写入 GPIO 引脚上对应的开关量状态，进行不同的操作。

3. 实验设备

- TM4C123G 教学实验箱，Pentium II 以上的 PC 机，J-link 仿真器；
- PC 操作系统 WINXP 或以上，Keil MDK5.26 集成开发环境，J-link 仿真调试驱动程序。

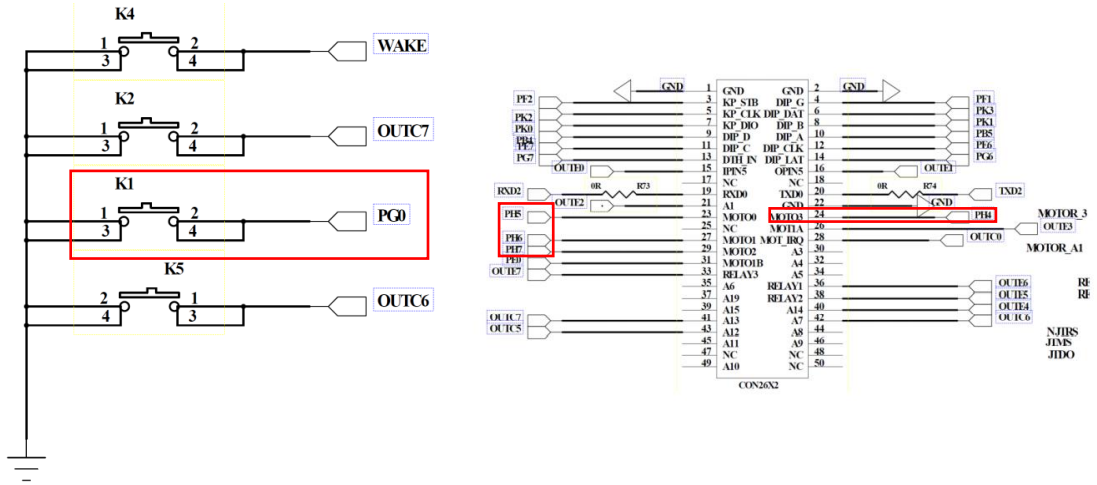
4. 实验要求

- 编译程序后，LED4, LED5, LED6, LED7 亮灭交替；
- 按下按键 K2，所有 LED 灯常亮。

5. 实验原理

本实验涉及到的知识点为 GPIO 端口的配置和读写，主要有以下方面工作：

首先是配置 LED、KEY，根据硬件图找到想要配置的按键和 LED 灯是由什么 IO 口控制的。例如配置 KEY 时，根据硬件示意图可见：K1 是由 PG0 控制的，当 K1 按下，PG0 为低电位。所以在配置 KEY 时，需要配置 GPIO 的 PG0。同样的道理查图找到舵机模块的 LED 相关的 GPIO 口。配置相应的 GPIO 接口即可。配置主要包括片选、选择方向、使能、配置上拉。同时记住 GPIO 读写相关的数据寄存器名，以便后续的读写操作。



接下来是读写 GPIO 口部分。欲实现实验所期望的结果，需要不断查询按键状态并写相应的 LED。首先是读按键 KEY 状态，需读取 PG0 来获取 K1 是否按下的状态。欲使 LED 灯实现流水灯效果，需不断向 LED 灯对应的 GPIO 口轮流写不同的数据即可实现。

6. 实验步骤

- 连线：将实验箱 5V/2A 电源线连接好，并将 J-link 仿真器与试验箱 CPU 板和电脑连接好；

- 新建一个文件夹，在里面建立一个 keil 工程，然后完成相应代码；
- 点击 Build 按钮，编译程序：
- 编译成功后，实验箱上电，点击 Load 按钮下载程序：
- 下载成功，进入测试

7. 实验程序

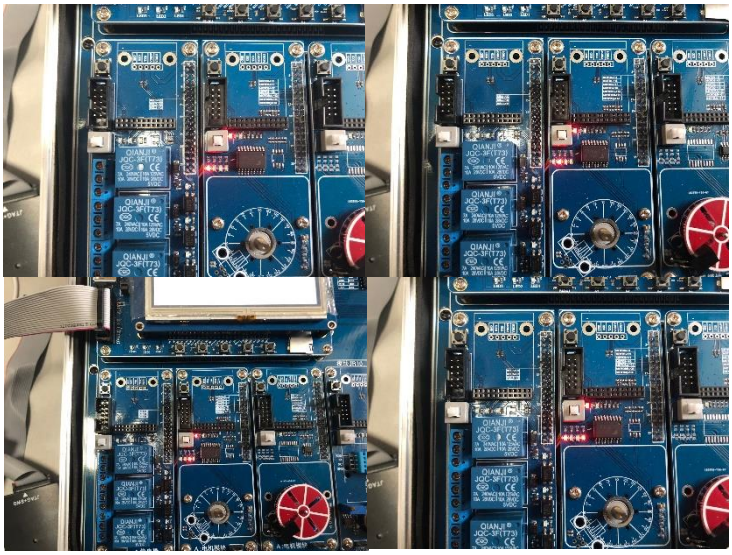
```
#include "lm4f232h5qd.h"
#include <stdbool.h>
#include <stdint.h>
#include <Stdlib.h>
#include <String.h>
#include "driverlib/fpu.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"

#include "driverlib/gpio.h"
#include "inc/hw_memmap.h"
#define SYS_CLOCK_KHZ 80000
//KEY1 地址 = 基地址(GPIO_PORTG_BASE) + 偏移量(GPIO_PIN_0)
//#define KEY1 ROM_GPIOPinRead(GPIO_PORTG_BASE,GPIO_PIN_0)

void sysclock_cfg();//配置时钟
void Key_init();//初始化按键
void LED_init();//初始化LED
void delay();//延时函数

int main(void)
{
    sysclock_cfg();
    Key_init();
    LED_init();
    while(1)
    {
        if(GPIO_PORTG_DATA_R<<7 == 0x00)
        {
            GPIO_PORTH_DATA_R =0xF0;
        }
        else{
            GPIO_PORTH_DATA_R =0x80;
            delay();
            GPIO_PORTH_DATA_R =0x40;
            delay();
            GPIO_PORTH_DATA_R =0x20;
            delay();
```

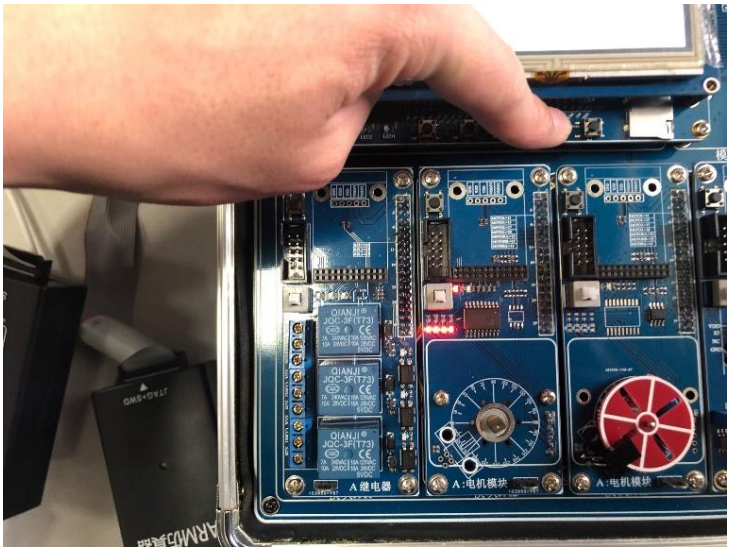
```
            GPIO_PORTH_DATA_R =0x10;
            delay();
        }
    }
}

void sysclock_cfg()
{
    uint32_t ui32Delay, ui32RCC, ui32RCC2;
    ui32RCC = SYSCTL_RCC_R;
    ui32RCC2 = SYSCTL_RCC2_R;
    ui32RCC |= SYSCTL_RCC_BYPASS;
    ui32RCC &= ~(SYSCTL_RCC_USESYSDIV);
    ui32RCC2 |= SYSCTL_RCC2_BYPASS2;
    SYSCTL_RCC_R = ui32RCC;
    SYSCTL_RCC2_R = ui32RCC2;
    SYSCTL_RCC_R &= ~(0x01);
    for(ui32Delay = 524288; ui32Delay > 0; ui32Delay--)
    {
    }
    ui32RCC = SYSCTL_RCC_R;
    ui32RCC2 = SYSCTL_RCC2_R;
    ui32RCC &= ~(0x000007c0 | 0x00000030);
    ui32RCC |= 0x00000440;
    SYSCTL_RCC_R = ui32RCC;
    SYSCTL_RCC2_R &= ~(0x00000070);
    SYSCTL_RCC2_R |= (0x80000000);
    for(ui32Delay = 32768; ui32Delay > 0; ui32Delay--)
    {
    }
    SYSCTL_MISC_R |= 0x00000040;
    SYSCTL_RCC2_R &= ~(0x00002000);
    SYSCTL_RCC_R &= ~(0x00002000);
    ui32RCC = SYSCTL_RCC_R;
    ui32RCC &= ~(0x07800000 | 0x00400000);
    ui32RCC |= 0x07800000;
    ui32RCC2 = SYSCTL_RCC2_R;
    ui32RCC2 &= ~(0x1f800000 | 0x00400000);
    ui32RCC2 |= (0x41000000);
    for(ui32Delay = 32768; ui32Delay > 0; ui32Delay--)
    {
        if(SYSCTL_RIS_R & 0x00000040)
        {
            break;
```

```
        }
    }
    ui32RCC &= ~(0x00000800);
    ui32RCC2 &= ~(0x00000800);
    SYSCTL_RCC_R = ui32RCC;
    SYSCTL_RCC2_R = ui32RCC2;
    for(ui32Delay = 0; ui32Delay < 10000; )
    {
    ui32Delay++;
    }
}

void Key_init(){
    SYSCTL_RCGCGPIO_R |= 0x40;
    //数据方向寄存器，0 为引脚方向输入，1 为引脚方向输出
    //注意：通过 GPIO_PORTG_DIR_R 的写法即可访问 PG0 口的 DIR 寄存器
    GPIO_PORTG_DIR_R |= 0x00;
    //复用功能选择寄存器，复位，引脚不复用,仅用作 GPIO 功能
    GPIO_PORTG_AFSEL_R |= 0x00;
    //数字使能寄存器，启用 PG 管脚数字功能
    GPIO_PORTG_DEN_R |= 0xFF;
    //上拉电阻选择寄存器，PG0 引脚上拉，此引脚置位时为 CPU 内部的弱上拉
    GPIO_PORTG_PUR_R |= 0x01;
    //数据寄存器( 这是实际干活儿的 )，8 位控制 PH0~7 的数值
    GPIO_PORTG_DATA_R =0x10;
}

void LED_init(){
    SYSCTL_RCGCGPIO_R |= 0x80;
    //数据方向寄存器，0 为引脚方向输入，1 为引脚方向输出
    //注意：通过 GPIO_PORTH_DIR_R 的写法即可访问 PH0 口的 DIR 寄存器
    GPIO_PORTH_DIR_R |= 0xF0;
    //复用功能选择寄存器，复位，引脚不复用,仅用作 GPIO 功能
    GPIO_PORTH_AFSEL_R |= 0x00;
    //数字使能寄存器，启用 PH 管脚数字功能
    GPIO_PORTH_DEN_R |= 0xFF;
    //上拉电阻选择寄存器，PH0 引脚上拉，此引脚置位时为 CPU 内部的弱上拉
    GPIO_PORTH_PUR_R |= 0x01;
    //数据寄存器( 这是实际干活儿的 )，8 位控制 PH0~7 的数值
    GPIO_PORTH_DATA_R =0x20;
}

void delay()
{
```

```
    int i =0;
    for(i=0;i<500000;i++);
}
```

8. 实验结果





不按下 K2 时，舵机模块的 4 个 LED 灯从右向左依次熄灭后亮起，并循环往复。即流水灯效果。

按下 K2 时，4 个 LED 灯全亮，松手后恢复成流水灯的效果。

9. 实验心得

本次实验是第一次嵌入式实验，主要做的工作包括学习嵌入式开发的一般步骤流程、搭建实验环境和实验一的内容。实验一本身的内容不是很难，但是整个实验的框架需要我们去理解和掌握。即整体程序主要由**定义+配置+主函数**组成，同时我们需要掌握查看硬件图、查询芯片手册，包括如何上板子进行实验，如何熟悉和使用 Keil 开发 IDE，所以第一次实验的内容方面是非常之足的，在队友们的互帮互助之下，大家都学到了很多东西，能看到最后的结果跑出来还是很开心的。

二、 实验 2 WWDG 实验

1. 实验目的

- 掌握 TM4C123G 的看门狗定时器（WDT）工作原理；
- 掌握看门狗定时器的初始化及配置流程；
- 掌握配置和处理看门狗产生中断信号的方法。

2. 实验内容

- 编程实现看门狗的初始化及配置，观察看门狗溢出的现象；
- 编程实现看门狗的中断信号处理。

3. 实验设备

- TM4C123G 教学实验箱，Pentium II 以上的 PC 机，J-link 仿真器；
- PC 操作系统 WINXP 或以上，Keil MDK5.26 集成开发环境，J-link 仿真调试驱动程序。

4. 实验要求

- 如果 LED4 灭，LED5 闪烁，表示看门狗在正常运行；
- 如果 LED4 亮，LED5 闪烁，表示看门狗因没有喂狗，而产生中断；
- 通过按键 K2 实现喂狗，看门狗在正常运行。

5. 实验原理

当系统由于软件错误或是由于因外部设备故障而无法按预期的方式响应的时候，使用看门狗定时器可以重新获得控制权。

TM4C123 系列微控制器有两个看门狗定时器模块，一个模块使用系统时钟计时(WDT0)，另一个模块使用 PIOSC 计时(WDT1)。这两个模块是相同的，只是 WDT1 在不同的时钟域，因此需要同步器。

(1) 将计数时间写入 WDTLOAD(32 位装载寄存器)之后，看门狗定时器 WDT 便会进入倒计时，当 WDTLOAD 第一次达到 0x0000,0000 的时候产生中断信号，警告 CPU；

(2) 产生第一次中断后，WDTLOAD 自动重新装载，重新计时，并在第二次计时结束后产生系统复位信号。

6. 实验步骤

- 连线：将实验箱 5V/2A 电源线连接好，并将 J-link 仿真器与试验箱 CPU 板和电脑连接好；
- 新建一个文件夹，在里面建立一个 keil 工程，然后完成相应代码；
- 点击 Build 按钮，编译程序：
- 编译成功后，实验箱上电，点击 Load 按钮下载程序：
- 下载成功，进入测试

7. 实验程序

```
#include "lm4f232h5qd.h"
#include <stdbool.h>
#include <stdint.h>
#include <Stdlib.h>
#include <String.h>
#include "driverlib/fpu.h"
```

```
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"

#include "driverlib/gpio.h"
#include "inc/hw_memmap.h"
#include "hw_watchdog.h"
#define SYS_CLOCK_KHZ 80000
#define KEY1 ROM_GPIOPinRead(GPIO_PORTG_BASE,GPIO_PIN_0)

void sysclock_cfg();//时钟配置
void Key_init();//按键初始化
void LED_init();//LED灯初始化
void delay();//延时函数
void watchdog_cfg();//看门狗初始化
void twinkle();////LED5闪烁

int main(void){
    sysclock_cfg();
    Key_init();
    LED_init();
    watchdog_cfg();

    while(1){

        twinkle();
        //按键被按下
        if(!KEY1){
            //喂狗
            WATCHDOG0_ICR_R = 0X01;
            WATCHDOG0_LOCK_R = 0x1ACCE551;
      WATCHDOG0_LOAD_R = 0x0fffffff;
      twinkle();
        }
    }
}

void sysclock_cfg()
{
    uint32_t ui32Delay, ui32RCC, ui32RCC2;
    ui32RCC = SYSCTL_RCC_R;
    ui32RCC2 = SYSCTL_RCC2_R;
    ui32RCC |= SYSCTL_RCC_BYPASS;
    ui32RCC &= ~(SYSCTL_RCC_USESYSDIV);
```

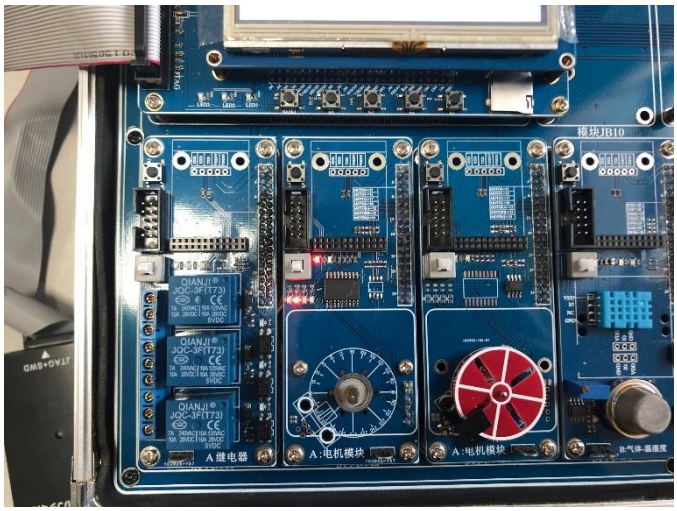
```
    ui32RCC2 |= SYSCTL_RCC2_BYPASS2;
    SYSCTL_RCC_R = ui32RCC;
    SYSCTL_RCC2_R = ui32RCC2;
    SYSCTL_RCC_R &= ~(0x01);
    for(ui32Delay = 524288; ui32Delay > 0; ui32Delay--)
    {
    }
    ui32RCC = SYSCTL_RCC_R;
    ui32RCC2 = SYSCTL_RCC2_R;
    ui32RCC &= ~(0x000007c0 | 0x00000030);
    ui32RCC |= 0x00000440;
    SYSCTL_RCC_R = ui32RCC;
    SYSCTL_RCC2_R &= ~(0x00000070);
    SYSCTL_RCC2_R |= (0x80000000);
    for(ui32Delay = 32768; ui32Delay > 0; ui32Delay--)
    {
    }
    SYSCTL_MISC_R |= 0x00000040;
    SYSCTL_RCC2_R &= ~(0x00002000);
    SYSCTL_RCC_R &= ~(0x00002000);
    ui32RCC = SYSCTL_RCC_R;
    ui32RCC &= ~(0x07800000 | 0x00400000);
    ui32RCC |= 0x07800000;
    ui32RCC2 = SYSCTL_RCC2_R;
    ui32RCC2 &= ~(0x1f800000 | 0x00400000);
    ui32RCC2 |= (0x41000000);
    for(ui32Delay = 32768; ui32Delay > 0; ui32Delay--)
    {
        if(SYSCTL_RIS_R & 0x00000040)
        {
            break;
        }
    }
    ui32RCC &= ~(0x00000800);
    ui32RCC2 &= ~(0x00000800);
    SYSCTL_RCC_R = ui32RCC;
    SYSCTL_RCC2_R = ui32RCC2;
    for(ui32Delay = 0; ui32Delay < 10000; )
    {
    ui32Delay++;
    }
}

void Key_init(){
```

```
    SYSCTL_RCGCGPIO_R |= 0x40;
    GPIO_PORTG_DIR_R |= 0x00;
    GPIO_PORTG_AFSEL_R |= 0x00;
    GPIO_PORTG_DEN_R |= 0xFF;
    GPIO_PORTG_PUR_R |= 0x01;
    GPIO_PORTG_DATA_R =0x00;
}

void LED_init(){
    SYSCTL_RCGCGPIO_R |= 0x80;
    GPIO_PORTH_DIR_R |= 0xF0;
    GPIO_PORTH_AFSEL_R |= 0x00;
    GPIO_PORTH_DEN_R |= 0xFF;
    GPIO_PORTH_PUR_R |= 0x01;
    GPIO_PORTH_DATA_R =0x00;
}

//LED5闪烁
void twinkle(){
      GPIO_PORTH_DATA_R |= 0x40;
    delay();
    GPIO_PORTH_DATA_R &= 0x00;
    delay();
}

void watchdog_cfg(){
    SYSCTL_RCGCWD_R |= SYSCTL_RCGCWD_R0;
    WATCHDOG0_LOCK_R = 0x1ACCE551;
        WATCHDOG0_CTL_R |= 0x03;
        WATCHDOG0_LOAD_R = 0x0fffffff;
    NVIC_EN0_R |= 0x00040000;
    WATCHDOG0_LOCK_R = 0x01;
}

void WATCHDOG(){
    GPIO_PORTH_DATA_R |= 0x80;
    while(1){
        GPIO_PORTH_DATA_R |= 0xc0;
        delay();
        GPIO_PORTH_DATA_R &= 0x80;
        delay();
    if(!KEY1){
            WATCHDOG0_ICR_R = 0X01;
      WATCHDOG0_LOCK_R = 0x1ACCE551;
```

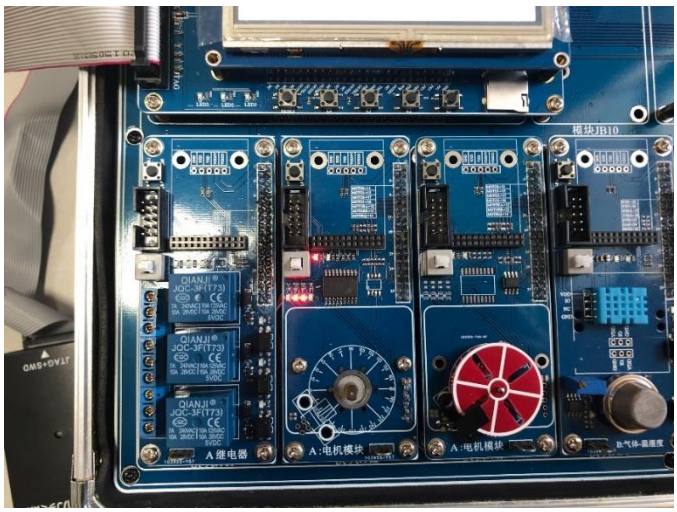
```
        WATCHDOG0_LOAD_R = 0x0fffffff;
        break;
    }
    }
}

void delay()
{
    int i =0;
    for(i=0;i<1000000;i++);
}
```

8. 实验结果

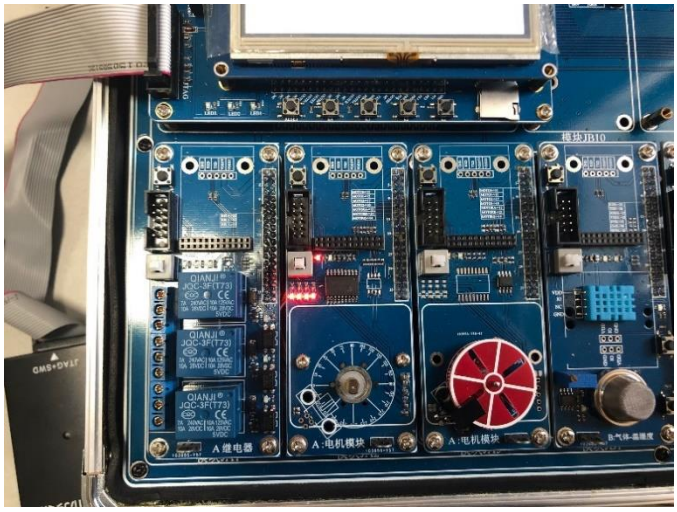
观察到 LED4 灭，LED5 闪烁，表示看门狗正常运行。



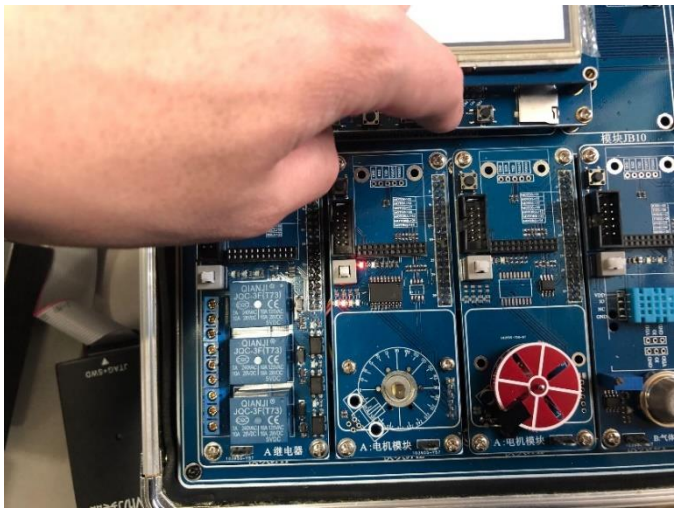
一段时间后 LED4 亮，LED5 闪烁，表示看门狗因没有喂狗，而产生中断。



中断出现一会后，其自动重置，看门狗正常运行：



若在中断中通过按键 K2 实现喂狗，看门狗重新正常运行。



若一直喂狗，则一直正常运行。

9. 实验心得

第二次实验相比第一次实验，难度并没有增加多少。有了第一次的经验，实验二的完成速度相对较快。主要是通过这个实验,我们认识了在嵌入式芯片中非常重要也应用非常广泛的看门狗寄存器，通过模拟看门狗中断和实现喂狗操作的方式来模拟一个简单的看门狗应用场景，我们在此中也进一步学到了看门狗知识。

当然我们在实验中也遇到了一些问题。我们在之前定义了让 LED5 闪烁的函数，但是在中断程序里调用时出现了问题，因为这时还得同时让 LED4 亮，所以在中断里重写了闪烁的代码。还有在中断程序里喂狗时，得先清除中断才行，不然无法立刻变成正常运行。

三、 实验 3 I2C 实验

1. 实验目的

- 了解 I2C 通信原理；
- 掌握 I2C 相应寄存器的初始化及配置流程；
- 掌握 I2C 主机模式下的信号传输及接收方法。

2. 实验内容

- 编程实现 I2C 模块的初始化及配置；
- 编程实现 I2C 主机模式下的信号传输及接收。

3. 实验设备

- TM4C123G 教学实验箱，Pentium II 以上的 PC 机，J-link 仿真器；
- PC 操作系统 WINXP 或以上，Keil MDK5.26 集成开发环境，J-link 仿真调试驱动程序。

4. 实验要求

- 按动 K3 按键，LED1 灯 亮与灭交替；
- 按动 K4 按键，LED2 灯 亮与灭交替；
- 按动 K2 按键，LED3 灯 亮与灭交替。

5. 实验原理

- 通过 I2C 总线实现主机模式下对不同从机地址的信号传输和接收；
- 本试验箱有 4 块 PCF8574 芯片作为 I2C 从机，分别为 UA，UB，UD，UE，其地址分别为 0x20，0x21，0x23，0x27，液晶显示屏周围的 LED 灯和按键都是通过 UD 芯片扩展，通过读取和写入相应的 I2C 信号来实现按键对 LED 灯的操作。

6. 实验步骤

- 连线：将实验箱 5V/2A 电源线连接好，并将 J-link 仿真器与试验箱 CPU 板和电脑连接好；
- 新建一个文件夹，在里面建立一个 keil 工程，然后完成相应代码；
- 点击 Build 按钮，编译程序：
- 编译成功后，实验箱上电，点击 Load 按钮下载程序：
- 下载成功，进入测试

7. 实验程序

```
#include "lm4f232h5qd.h"
#include <stdbool.h>
#include <stdint.h>
#include <Stdlib.h>
#include <String.h>
#include "driverlib/fpu.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#include "driverlib/gpio.h"
```

```
#include "inc/hw_memmap.h"
#include "function/PCF8574.h"
#include "hw_watchdog.h"

#include<stdio.h>

#define SYS_CLOCK_KHZ 80000
#define KEY1 ROM_GPIOPinRead(GPIO_PORTG_BASE,GPIO_PIN_0)
void delay(int time)
{
   while(time --);
}

void setI2C()
{
    SYSCTL_RCGCI2C_R |= 0X01;

    SYSCTL_RCGCGPIO_R |= 0x02;
    GPIO_PORTB_AFSEL_R |= 0xFF;
    GPIO_PORTB_ODR_R |= 0x08;    //peizhi kailou
    GPIO_PORTB_DIR_R |= 0xFF;
    GPIO_PORTB_DEN_R |= 0xFF;
    GPIO_PORTB_PUR_R |= 0xFF;
    GPIO_PORTB_PCTL_R =0X00003300;  //P619

    I2C0_MASTER_MCR_R |=0X10;  //chushihua I2C zhuji
    I2C0_MASTER_MTPR_R |=0X27;  //shizhongpinglv
}
void setKey()
{
    SYSCTL_RCGCGPIO_R |= 0x40;
    GPIO_PORTG_DIR_R |= 0x00;
    GPIO_PORTG_AFSEL_R |= 0x00;
    GPIO_PORTG_DEN_R |= 0xFF;
    GPIO_PORTG_PUR_R |= 0x01;

}

void setLED()
{
    SYSCTL_RCGCGPIO_R |= 0x80;
    GPIO_PORTH_DIR_R |= 0xF0;
    GPIO_PORTH_AFSEL_R |= 0x00;
    GPIO_PORTH_DEN_R |= 0xFF;
```

```
    GPIO_PORTH_PUR_R |= 0x01;
}
void sysclock_cfg()
{
    uint32_t ui32Delay, ui32RCC, ui32RCC2;
    ui32RCC = SYSCTL_RCC_R;
    ui32RCC2 = SYSCTL_RCC2_R;
    ui32RCC |= SYSCTL_RCC_BYPASS;
    ui32RCC &= ~(SYSCTL_RCC_USESYSDIV);
    ui32RCC2 |= SYSCTL_RCC2_BYPASS2;
    SYSCTL_RCC_R = ui32RCC;
    SYSCTL_RCC2_R = ui32RCC2;
    SYSCTL_RCC_R &= ~(0x01);
    for(ui32Delay = 524288; ui32Delay > 0; ui32Delay--)
    {
    }
    ui32RCC = SYSCTL_RCC_R;
    ui32RCC2 = SYSCTL_RCC2_R;
    ui32RCC &= ~(0x000007c0 | 0x00000030);
    ui32RCC |= 0x00000440;
    SYSCTL_RCC_R = ui32RCC;
    SYSCTL_RCC2_R &= ~(0x00000070);
    SYSCTL_RCC2_R |= (0x80000000);
    for(ui32Delay = 32768; ui32Delay > 0; ui32Delay--)
    {
    }
    SYSCTL_MISC_R |= 0x00000040;
    SYSCTL_RCC2_R &= ~(0x00002000);
    SYSCTL_RCC_R &= ~(0x00002000);
    ui32RCC = SYSCTL_RCC_R;
    ui32RCC &= ~(0x07800000 | 0x00400000);
    ui32RCC |= 0x07800000;
    ui32RCC2 = SYSCTL_RCC2_R;
    ui32RCC2 &= ~(0x1f800000 | 0x00400000);
    ui32RCC2 |= (0x41000000);
    for(ui32Delay = 32768; ui32Delay > 0; ui32Delay--)
    {
        if(SYSCTL_RIS_R & 0x00000040)
        {
            break;
        }
    }
    ui32RCC &= ~(0x00000800);
    ui32RCC2 &= ~(0x00000800);
```

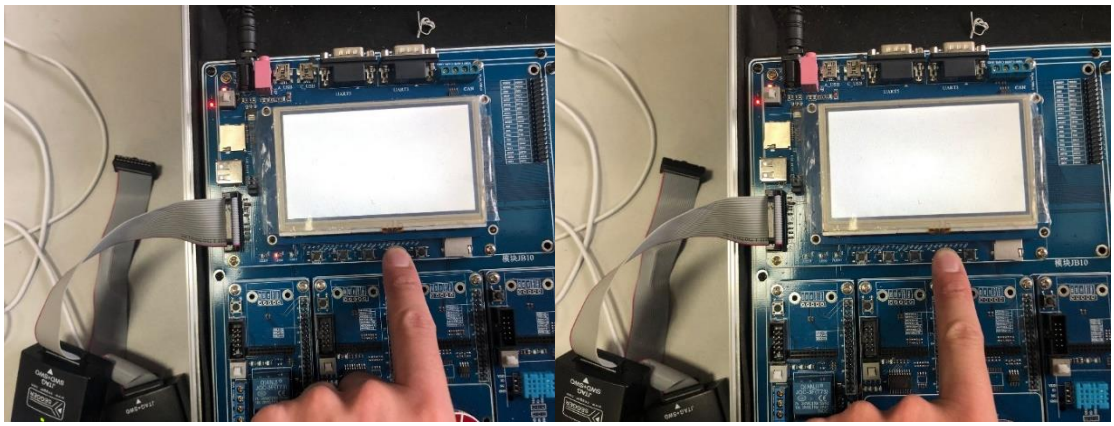
```
    SYSCTL_RCC_R = ui32RCC;
    SYSCTL_RCC2_R = ui32RCC2;
    for(ui32Delay = 0; ui32Delay < 10000; )
    {
        ui32Delay++;
    }
}


void isPress()
{
    while(1)
        {

            if(!KEY1)
            {
            I2C0_MASTER_MSA_R = ((0x23)<<1);
            I2C0_MASTER_MDR_R |= 0xFD;
            I2C0_MASTER_MCS_R = 0x07;
            delay(800000);
            I2C0_MASTER_MSA_R = ((0x23)<<1);
            I2C0_MASTER_MDR_R = 0xFF;
            I2C0_MASTER_MCS_R = 0x07;
            delay(800000);
            }
                        else
                break;
        }
}


int main(){
    sysclock_cfg();
    setKey();
    setLED();
    setI2C();

    while(1)
    {
        isPress();
    }
}
```

8. 实验结果

设置　　I2C0_MASTER_MDR_R |= 0xF5;

I2C0_MASTER_MDR_R = 0xF7;

通过按键，在使灯闪烁的同时，关闭蜂鸣器。



设置 I2C0_MASTER_MDR_R |= 0xFD;

I2C0_MASTER_MDR_R = 0xFF;

通过按键，在使灯闪烁的同时，打开蜂鸣器。

9. 实验心得

本次实验目的在于帮助我们了解 I2C 的。不得不说这次实验相比前两次，难度还是非常大的。

因为这次的实验原理就比较复杂，然后配置 I2C 有许多需要注意的地方，比如左移一位 MSA 地址等等，所以我们在配置 I2C 的过程中就花了很多的时间。配置过程还得得照着芯片手册和硬件图来完成 GPIO 和 I2C 两个的配置。所以在完成实验前得对这些都有一定的了解才行。当然一开始我们还考虑多个灯，但助教老师说考虑一个灯就行，这也算一定的简化吧。虽然有一定的难度，但是我们还是很有收获的，也让我们了解到了嵌入式开发的复杂性。

四、 实验 4 气压与温度传感器实验

1. 实验目的

- 学习 TM4C123G 的 GPIO 使用及其相关的 BMP085 芯片原理；
- 掌握用 GPIO 模拟单总线并与外部芯片进行通讯的方法。
- 使用 I2C 总线模拟读取气压与温度传感器的温度值与气压值。

2. 实验内容

- 用 TM4C123G 的 GPIO 模拟单总线来读取气压与温度传感器的温度和气压；
- 将读取到的温度与气压数值在屏幕上显示出来。

3. 实验设备

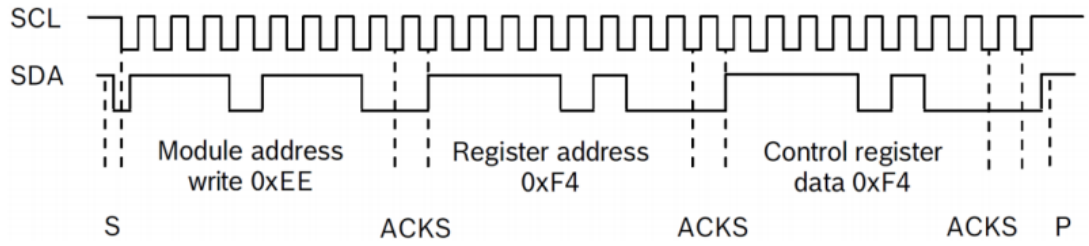
- TM4C123G 教学实验箱，Pentium II 以上的 PC 机，J-link 仿真器；
- BMP085 芯片；
- PC 操作系统 WINXP 或以上，Keil MDK5.26 集成开发环境，J-link 仿真调试驱动程序。

4. 实验要求

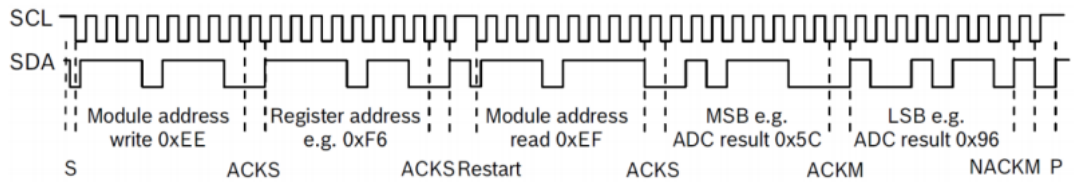
- 运行程序，液晶屏上的显示气压以及温度值；
- 在液晶屏上绘制出温度、气压随时间变化曲线。

5. 实验原理

单片机发送开始信号（I2C_MSA; I2C_MDR）启动温度和压力测量，经过一定的转换时间（4.5ms）后，从 I2C 接口读出结果。下图展示了开启测量流程的时序图。



要获得温度数据，必须先向控制寄存器（0xF4）写 0x2E，然后等待至少 4.5ms，才可以从地址 0xF6 和 0xF7 读取十六位的温度数据。同样，要获得气压数据，必须先向控制寄存器（0xF4）写 0x34，然后等待至少 4.5ms，才可以从地址 0xF6 和 0xF7 读取 16 位的气压数据，若要扩展分辨率，还可继续读取 0xF8（XLSB）扩展 16 位数据到 19 位。获取到的数据还要根据 EEPROM 中的校准数据来进行补偿后才能用，EEPROM 的数据读取的地址从 0xAA~0xBF。下图展示了读取数据流程的时序图。



实验中，采用 bmp085Convert() 读取温度、压力的实际值（ temperature、pressure ），连续读取可绘制曲线，绘制方法为使用 LCD_DrawPoint(x , y) 函数进行“投点”。采用 bmp085_conversion(long temp_data, uint8_t *p) 读取温度、压力各位的值，可将其在 LCD

屏上显示出来。

6. 实验步骤

- 连线：将实验箱 5V/2A 电源线连接好，并将 J-link 仿真器与试验箱 CPU 板和电脑连接好；
- 新建一个文件夹，在里面建立一个 keil 工程，然后完成相应代码；
- 点击 Build 按钮，编译程序：
- 编译成功后，实验箱上电，点击 Load 按钮下载程序：
- 连线：将实验箱 5V/2A 电源线连接好，并将 J-link 仿真器与试验箱 CPU 板和电脑连接好；
- 新建一个文件夹，在里面建立一个 keil 工程，然后完成相应代码；
- 点击 Build 按钮，编译程序：
- 编译成功后，实验箱上电，点击 Load 按钮下载程序：
- 下载成功，进入测试。

7. 实验程序

EX4.c

```
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "grlib/grlib.h"
#include "inc/hw_memmap.h"
#include "driverlib/fpu.h"

#include "hardware/lcd.h"
#include "driverlib/uart.h"
#include "hardware/relay.h"
#include "hardware/PCF8574.h"
#include "hardware/bmp085.h"
#include "hardware/DC_motor.h"
#include "hardware/dht11.h"
#include "hardware/Dot_Matrix.h"
#include "hardware/step_motor.h"
#include "hardware/MMA7455.h"
#include "hardware/tm1638.h"
#include "hardware/key.h"

//enable uart interrupt
#define UART_BUFFERED

void delay()
{
    for(int i=0;i<1000000;i++);
}
```

```
static uint8_t read_val = 0;

u8 play[4]  = {0x7E, 0x02, 0x0D, 0xEF};
u8 prev[4]  = {0x7E, 0x02, 0x02, 0xEF};
u8 next[4]  = {0x7E, 0x02, 0x01, 0xEF};
u8 pause[4] = {0x7E, 0x02, 0x0E, 0xEF};

u8 mode_U[5] =          {0x7E, 0x03, 0x09, 0, 0xEF};
u8 mode_FLASH[5] =  {0x7E, 0x03, 0x09, 4, 0xEF};
u8 mode_TF[5] =         {0x7E, 0x03, 0x09, 1, 0xEF};
int main(void)
{
    int time = 0;
    tContext sContext;
    tRectangle sRect;

    uint32_t i = 0;

    FPUEnable();
    FPULazyStackingEnable();

    // Set the clocking to run directly from the crystal.
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN);

        InitConsole(3);
        //UART2_Int_Config();

        PCF8574_I2C_GPIO_Config();
        KEY_Init();
    lcd_init();                    //初始化LCD驱动

        PCF8574_Single_WriteI2C(PCF8574T_E, 0xe0);  //close DC motor[0][3]
close relay[5:7]

        LCD_Clear(Black);
//      LCD_DrawRectangle(0,5, 479, 271);
//      LCD_DrawRectangle(5,18, 474, 266);
//      LCD_ShowString(150, 6, "--TM4C123G Platform--");  // 实验开发平台
//      LCD_ShowString(100,60, "Step 5:Temperature and Pressure and acclerate
Testing");
```

```
        while(KEY1 == 1)
        {
            BMP085_Test();//加速度计
            //MMA7455_Test();   //压力传感器
            time++;
        }
        Delay(5);

        LCD_ShowString(150, 6, "--TM4C123G Platform--");  // 实验开发平台
        LCD_ShowString(150,60, "Test Complete!");
        while(1)
        {

        }
}
```

bmp085.c

```
#include "bmp085.h"

#define OSS 0 // Oversampling Setting (note: code is not set up to use other
OSS values)

//***BMP085
unsigned char ge,shi,bai,qian,wan,shiwan;
short ac1;
short ac2;
short ac3;
unsigned short ac4;
unsigned short ac5;
unsigned short ac6;
short b1;
short b2;
short mb;
short mc;
short md;

long  temperature;
long  pressure;
float tempdat=0;
unsigned char outdata[6]={0};

void delay5ms(void);
void I2C_Stop();
```

```
//************十六进制转换成字符*****************************
void string_shift(unsigned int dat)
{
  outdata[3] =dat/1000+'0';       // 取千位
  outdata[2] =(dat%1000)/100+'0'; //取百位
  outdata[1] =(dat%100)/10+'0';   // 取十位
  outdata[0] =dat%10+'0';         // 取个位
}

  /*****************************/
void I2C_Gpio_Init(void)
{
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

  ROM_GPIOPinTypeGPIOOutput(SCL_BASE, SCL_PIN);
  ROM_GPIOPinTypeGPIOOutput(SDA_BASE, SDA_PIN);

  PCF8574_I2C_Stop();

  I2C_Stop();
}

void i2c_SDA_input(void)
{
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  ROM_GPIOPinTypeGPIOInput(SDA_BASE, SDA_PIN);
}

void i2c_SDA_output(void)
{
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  ROM_GPIOPinTypeGPIOOutput(SDA_BASE, SDA_PIN);
}

void I2C_delay(void)
{
   u16 i=200; //这里可以优化速度  ，经测试最低到5还能写入
   while(i)
   {
     i--;
   }
}

void delay5ms(void)
```

```
{
   int i=500;
   while(i)
   {
     i--;
   }
}

//I2C起始信号
void I2C_Start()
{
    SDA_H;                    //拉高数据线
    SCL_H;                    //拉高时钟线
    I2C_delay();                 //延时
    SDA_L;                    //产生下降沿
    I2C_delay();                 //延时
    SCL_L;                    //拉低时钟线
    I2C_delay();                 //延时
}

//I2C停止信号
void I2C_Stop()
{
    SCL_L;                    //拉高时钟线
    SDA_L;                    //拉低数据线
    I2C_delay();                 //延时
    SCL_H;                    //拉高时钟线
    I2C_delay();                 //延时
    SDA_H;                    //产生上升沿
    I2C_delay();                 //延时
}

//I2C发送应答信号
void I2C_SendACK(char ack)
{
  if(ack == 1)
    SDA_H;
  else
    SDA_L;                  //写应答信号
  SCL_H;                    //拉高时钟线
  I2C_delay();                 //延时
  SCL_L;                    //拉低时钟线
  I2C_delay();                 //延时
}
```

```
//I2C接收应答信号
char I2C_RecvACK()
{
  char CY;
  i2c_SDA_input();
  I2C_delay();                  //延时
  SCL_H;                       //拉高时钟线
  I2C_delay();                  //延时
  if(SDA_read)
    CY = 1;                    //读应答信号
  else
    CY = 0;
  SCL_L;                       //拉低时钟线
  i2c_SDA_output();
  I2C_delay();                  //延时
  return CY;
}

//向I2C总线发送一个字节数据
void I2C_SendByte(u8 dat)
{
    u8 i;
    for (i=0; i<8; i++)         //8位计数器
    {
        if(dat&0x80)
          SDA_H;                //送数据口
        else
          SDA_L;

        dat <<= 1;              //移出数据的最高位
        SCL_H;                 //拉高时钟线
        I2C_delay();            //延时
        SCL_L;                 //拉低时钟线
        I2C_delay();            //延时
    }
    I2C_RecvACK();
}

//从I2C总线接收一个字节数据
u8 I2C_RecvByte(void)
{
    u8 i;
    u8 dat = 0;
```

```
    i2c_SDA_input();
    for (i=0; i<8; i++)          //8位计数器
    {
        SCL_L;                  //拉低时钟线
        I2C_delay();              //延时
        SCL_H;                  //拉高时钟线
        I2C_delay();              //延时
        dat <<= 1;
        if(SDA_read)
          dat |= 1;             //读数据

    }
    i2c_SDA_output();
    return dat;
}


//向I2C设备写入一个字节数据
void Single_WriteI2C(u8 SlaveAddress, u8 REG_Address,u8 REG_data)
{
    I2C_Start();                  //起始信号
    I2C_SendByte(SlaveAddress);   //发送设备地址+写信号
    I2C_SendByte(REG_Address);    //内部寄存器地址，
    I2C_SendByte(REG_data);       //内部寄存器数据，
    I2C_Stop();                   //发送停止信号
}

//从I2C设备读取一个字节数据
u8 Single_ReadI2C(u8 SlaveAddress, u8 REG_Address)
{
  u8 REG_data;
  I2C_Start();                    //起始信号
  I2C_SendByte(SlaveAddress - 1);    //发送设备地址+写信号
  I2C_SendByte(REG_Address);      //发送存储单元地址，从0开始
  I2C_Start();                    //起始信号
  I2C_SendByte(SlaveAddress);  //发送设备地址+读信号
  REG_data=I2C_RecvByte();        //读出寄存器数据
  I2C_SendACK(1);                 //接收应答信号
  I2C_Stop();                     //停止信号
  return REG_data;
}

short Multiple_read(u8 SlaveAddress, u8 ST_Address)
{
```

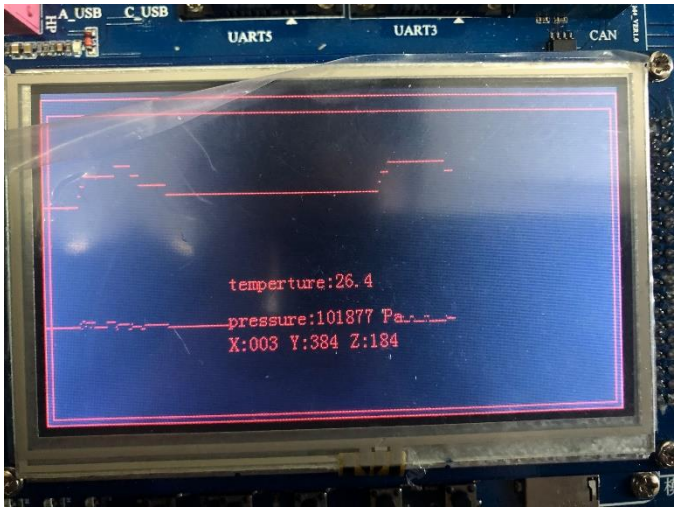
```
  u8 msb, lsb;
  short _data;
    I2C_Start();                          //????
    I2C_SendByte(SlaveAddress);
    I2C_SendByte(ST_Address);
    I2C_Start();
    I2C_SendByte(SlaveAddress+1);

    msb = I2C_RecvByte();
    I2C_SendACK(0);
    lsb = I2C_RecvByte();
  I2C_SendACK(1);

    I2C_Stop();
    //Delay5ms();
    _data = msb << 8;
  _data |= lsb;
  return _data;
}

//初始化MPU6050
void InitBMP085(void)
{
  ac1 = Multiple_read(BMP085_SlaveAddress, 0xAA);
  ac2 = Multiple_read(BMP085_SlaveAddress, 0xAC);
  ac3 = Multiple_read(BMP085_SlaveAddress, 0xAE);
  ac4 = Multiple_read(BMP085_SlaveAddress, 0xB0);
  ac5 = Multiple_read(BMP085_SlaveAddress, 0xB2);
  ac6 = Multiple_read(BMP085_SlaveAddress, 0xB4);
  b1 =  Multiple_read(BMP085_SlaveAddress, 0xB6);
  b2 =  Multiple_read(BMP085_SlaveAddress, 0xB8);
  mb =  Multiple_read(BMP085_SlaveAddress, 0xBA);
  mc =  Multiple_read(BMP085_SlaveAddress, 0xBC);
  md =  Multiple_read(BMP085_SlaveAddress, 0xBE);
}

long bmp085ReadTemp(void)
{

    I2C_Start();
    I2C_SendByte(BMP085_SlaveAddress);
    I2C_SendByte(0xF4);
    I2C_SendByte(0x2E);
    I2C_Stop();
```

```
    I2C_delay();

  return (long) Multiple_read(BMP085_SlaveAddress, 0xF6);
}

long bmp085ReadPressure(void)
{
  long pressure = 0;

    I2C_Start();
    I2C_SendByte(BMP085_SlaveAddress);
    I2C_SendByte(0xF4);
    I2C_SendByte(0x34);
    I2C_Stop();
  I2C_delay();

  pressure = Multiple_read(BMP085_SlaveAddress, 0xF6);
  pressure &= 0x0000FFFF;

  return pressure;
  //return (long) bmp085ReadShort(0xF6);
}

void bmp085Convert()
{
  unsigned int ut = 0;
  unsigned long up = 0;
  long x1, x2, b5, b6, x3, b3, p;
  unsigned long b4, b7;

  ut = bmp085ReadTemp();     // change by environment
  up = bmp085ReadPressure();  // change by environment

  x1 = (((long)ut - (long)ac6)*(long)ac5) >> 15;
  x2 = ((long) mc << 11) / (x1 + md);
  b5 = x1 + x2;
  temperature = ((b5 + 8) >> 4);

  b6 = b5 - 4000;
  // Calculate B3
  x1 = (b2 * (b6 * b6)>>12)>>11;
  x2 = (ac2 * b6)>>11;
  x3 = x1 + x2;
  b3 = (((((long)ac1)*4 + x3)<< OSS) + 2)>>2;
```

```
  // Calculate B4
  x1 = (ac3 * b6)>>13;
  x2 = (b1 * ((b6 * b6)>>12))>>16;
  x3 = ((x1 + x2) + 2)>>2;
  b4 = (ac4 * (unsigned long)(x3 + 32768))>>15;

  b7 = ((unsigned long)(up - b3) * (50000 >> OSS));
  if (b7 < 0x80000000)
    p = (b7<<1)/b4;
  else
    p = (b7/b4)<<1;

  x1 = (p>>8) * (p>>8);
  x1 = (x1 * 3038)>>16;
  x2 = (-7357 * p)>>16;
  pressure = p+((x1 + x2 + 3791)>>4);
}

void bmp085_conversion(long temp_data, uint8_t *p)
{
    *p++ = temp_data/100000+0x30;
    temp_data=temp_data%100000;
    *p++=temp_data/10000+0x30;
    temp_data=temp_data%10000;
    *p++=temp_data/1000+0x30;
    temp_data=temp_data%1000;
    *p++=temp_data/100+0x30;
    temp_data=temp_data%100;
    *p++=temp_data/10+0x30;
    temp_data=temp_data%10;
    *p++=temp_data+0x30;
}

int x=0;
void DATA_Diplay()
{
  u8 tmp[15]={"temperture:00.0"}, pres[19] = {"pressure:000000 Pa"};
  long temp_value;
//  LCD_ShowString(200, 18, bmp_tmp, White, Black);
  temp_value=temperature;

  bmp085_conversion(temp_value, outdata);
```

```
  tmp[11]=outdata[3];
  tmp[12]=outdata[4];
  tmp[14]=outdata[5];

  LCD_ShowString(150, 230, tmp);
  //printf("temperature=%c%c.%c\n",outdata[3],outdata[4],outdata[5]);

  temp_value = pressure;
  bmp085_conversion(temp_value, outdata);

  pres[9]  = outdata[0];
  pres[10] = outdata[1];
  pres[11] = outdata[2];
  pres[12] = outdata[3];
  pres[13] = outdata[4];
  pres[14] = outdata[5];

  LCD_ShowString(150, 250, pres);
  //printf("pressure=%c%c%c%c%c%c
Pa\n",outdata[0],outdata[1],outdata[2],outdata[3],outdata[4],outdata[5]);
  LCD_DrawPoint(x,700-(pressure/200));
  LCD_DrawPoint(x,200-temperature/2);
  x++;
}

void BMP085_Test(void)
{
  InitBMP085();
  bmp085Convert();
  DATA_Diplay();
}
```

8. 实验结果



运行程序，液晶屏上显示出了温压传感器 BMP085 测量得出的气压以及温度值，以及温度、气压随时间分别变化的曲线。用手指触碰探测器模拟外界环境的改变，可以看到曲线有较为明显的波动变化。

9. 实验心得

本次实验是较为综合的实验，涉及到了 I2C 的读写以及温压传感器 BMP085 的配置和使用以及液晶屏的配置以及使用，也用到了很多前几次实验的内容和调试技巧。由于之前几次实验已经建立了较为清晰的实验框架，这次上手实验还是很快的。同样地，本次实验教会我们的不仅仅只有本次实验本身的内容，我们更从中学到了 I2C 读写外部设备的一般方法，学会在实验中举一反三是有必要的，也是本次实验设计的目的。