

# 《嵌入式系统结构与设计》期末 project 设计

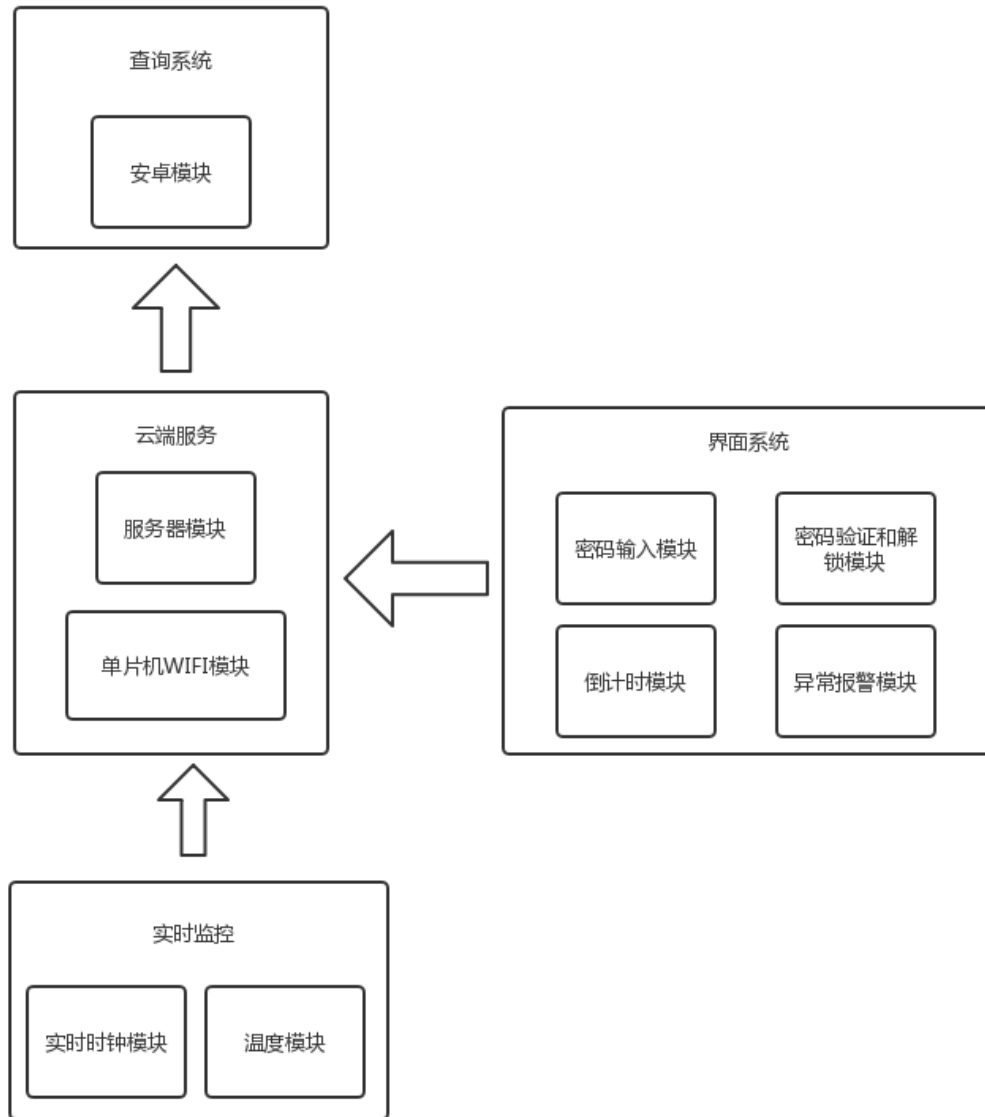
## 目录

第一章：项目概述 .....	2
第二章：相关技术基础 .....	4
第三章：系统设计 .....	4
密码输入模块 .....	4
1. 功能 .....	5
2. 基本思路 .....	5
实时时钟模块 .....	9
温度模块 .....	13
密码验证与解锁模块 .....	15
倒计时模块 .....	18
异常报警模块 .....	20
单片机 WIFI 模块 .....	22
服务器模块 .....	30
安卓模块 .....	34
第四章：总结 .....	40

## 第一章：项目概述

目标：利用本学期学习的的两种设备（可以同时使用/只用一个设备）设计一个可用的智能门禁系统

# 智能门禁系统



已实现功能：

## 1. 对通道进出权限的管理

实现密码开门，持卡人必须输入密码正确才能开门，且多次输入错误密码后会发送警告信息到服务器从而通知用户。

## 2. 防止非法进入功能（扩展）

每次开锁都会有时间记录，用户查询记录可以发现异常信息。

### **3. 实时监控功能（扩展）**

系统管理人员可以实时查看门的状态。用户通过安卓应用程序可以实时向服务器查询门的状态。而单片机每次状态发生改变后也会通知服务器。

### **4. 异常报警功能**

在异常情况下可以实现报警，如门超时未关、多次输入错误密码。服务器记录各种报警记录。

### **5. 记录查询（扩展）**

出入记录查询功能，可查询任何人、任何时间在任何门出入的记录。

报警记录查询，查询所有警告记录。

### **6. 有实时时钟功能，显示时间**

### **7. 有键盘、显示屏**

### **8. 对环境的温度状况进行监测，并且每隔 30s 发送给服务器。**

**9.有网络功能：** 单片机通过 WIFI 连接服务器，手机可以直接控制开关门和读取门的状态

**10.把门的状态和开门记录等数据上传到云服务器、通过 PC 和手机远程控制、开关门和读取门的状态、可在地图查询门的位置**

**扩展：**出入记录查询功能，可查询任何人、任何时间在任何门出入的记录、报警记录查询。

## **第二章：相关技术基础**

**Wifi 模块的使用**

**TCP 服务器的搭建**

温度模块的使用

单片机的使用

安卓应用的设计

## 第三章：系统设计

### 1. 总体设计

单片机运行后，自动连接 WIFI 并连接 TCP 服务器。完成连接 WIFI 后，8 段数码管显示时间，并且每 6 秒在 年月日/时分秒 之间切换一次。系统每隔 50ms 扫描一次键盘，如果有点击，8 段数码管显示输入的数字，最多输入 6 个数字。同时当停止输入一段时间后，单片机自动清空输入的密码，同时继续显示时间。

按下 C 按键后，如果密码正确则将数码管显示为全为 1，同时发送通过 TCP 发送当前密码锁为解锁状态以及创建一条开锁记录。而如果 20s 内没有按 D 键关闭密码锁，P47 即 LED9 开始闪烁警告，并且向服务器发送一条告警信息。如果密码错误则清空输入，当出现三次密码输入错误时单片机向服务器发送一条告警信息。

单片机每隔 30s 向服务器发送一次当前温度，并且每次门锁状态发生改变时单片机均通知服务器。

### 2. 硬件电路设计

#### 密码输入模块

模块简介：

#### 1. 功能

用按键输入 6 位密码，满 6 位无法再输入数字；同时有一个删除键用于删除最后

一位输入的数字，一个清除键用于清除所有输入的数字。

## 2. 基本思路

通过行列式扫描来确定输入的键码，再将输入值发送到 HC595 寄存器中，调用定时器 T0 中断，进行动态扫描显示输入的数字

模块解析：

1. 键盘扫描函数：通过行列式扫描确定按下哪个键，用于后面的数字输入和删除、清除键

```
/*
 行列键扫描程序
 使用xy查找4x4键的方法，只能单键，速度快
*/
      Y      P04      P05      P06      P07
      |      |      |      |
X      |      |      |      |
P00 ---- K00 ---- K01 ---- K02 ---- K03 ----
      |      |      |      |
P01 ---- K04 ---- K05 ---- K06 ---- K07 ----
      |      |      |      |
P02 ---- K08 ---- K09 ---- K10 ---- K11 ----
      |      |      |      |
P03 ---- K12 ---- K13 ---- K14 ---- K15 ----
      |      |      |      |
      +-----+-----+-----+-----+

u8 code T_KeyTable[16] = {0,1,2,0,3,0,0,0,4,0,0,0,0,0,0,0};

void IO_KeyDelay(void)
{
    u8 i;
    i = 60;
    while(--i) ;
}

void IO_KeyScan(void) //50ms call
{
    u8 j;

    j = IO_KeyStatel; //保存上一次状态
    P0 = 0x0f; //Y低，读X
    IO_KeyDelay();
    IO_KeyStatel |= (P0 & 0x0f);
    IO_KeyStatel ^= 0xff; //取反

    if(j == IO_KeyStatel) //连续两次读相等
    {
        j = IO_KeyState;
        IO_KeyState = IO_KeyStatel;
        if(IO_KeyState != 0) //有键按下
        {
            F0 = 0;
            if(j == 0) F0 = 1; //第一次按下
            else if(j == IO_KeyState)
            {
                if(++IO_KeyHoldCnt >= 20) //1秒后重键
                {
                    IO_KeyHoldCnt = 18;
                    F0 = 1;
                }
            }
            if(F0)
            {
                j = T_KeyTable[IO_KeyState >> 4];
                if((j != 0) && (T_KeyTable[IO_KeyState & 0x0f] != 0))
                    KeyCode = (j - 1) * 4 + T_KeyTable[IO_KeyState & 0x0f] + 16; //计算键码，17~32
            }
            else IO_KeyHoldCnt = 0;
        }
        P0 = 0xff;
    }
}
```

2. 显示密码函数：已经输入的位显示所输入的数字，未输入的位不显示

```
/****** 显示密码函数 *****/
void DisplayPassword(void)
{
    u8 i;
    for(i = 0; i < 6; i++)
    {
        if(input[i] < 10)
            LED8[i] = input[i];
        else
            LED8[i] = DIS_BLACK;
    }
    LED8[6] = DIS_BLACK;
    LED8[7] = count;
}
```

3. 数码管扫描显示函数：将数组 LED8 的值发送到 HC595 寄存器中，调用定时器 T0 中断，进行动态扫描显示时间

```
/****** 向HC595发送一个字节函数 *****/
void Send_595(u8 dat)
{
    u8 i;
    for(i=0; i<8; i++)
    {
        dat <<= 1;
        P_HC595_SER = CY;
        P_HC595_SRCLK = 1;
        P_HC595_SRCLK = 0;
    }
}

/****** 显示扫描函数 *****/
void DisplayScan(void)
{
    Send_595(~T_COM[display_index]); //输出位码
    Send_595(t_display[LED8[display_index]]); //输出段码

    P_HC595_RCLK = 1;
    P_HC595_RCLK = 0; //锁存输出数据
    if(++display_index >= 8) display_index = 0; //8位结束回0
}

/****** Timer0 lms中断函数 *****/
void timer0 (void) interrupt TIMER0_VECTOR
{
    DisplayScan(); //lms扫描显示一位
    B_lms = 1; //lms标志
}
```

4. 获取键值：每个 50ms 扫描一次键盘获取键值，若键值在 17~26 之间则分别是输入数字 0~9，若键值为 27 则是删除键，键值为 28 则是清除键，键值 29 为确认键，键值 30 为关门键

```

if(++cnt50ms >= 50)    //50ms扫描一次行列键盘
{
    cnt50ms = 0;
    IO_KeyScan();
}

if(KeyCode > 0 && KeyCode <= 28)    //有键按下
{
    if(KeyCode < 27)    //输入数字0~9
    {
        if(count < 6)
        {
            input[count] = KeyCode - 17;
            count++;
        }
        DisplayPassword();
        KeyCode = 0;
    }

    if(KeyCode == 27)    //输入回退一位
    {
        if(count > 0)
        {
            count--;
            input[count] = 10;
        }
        DisplayPassword();
        KeyCode = 0;
    }

    if(KeyCode == 28)    //清除所有输入
    {
        count = 0;
        for(i = 0; i < 6; i++)
            input[i] = 10;
        DisplayPassword();
        KeyCode = 0;
    }
}

```

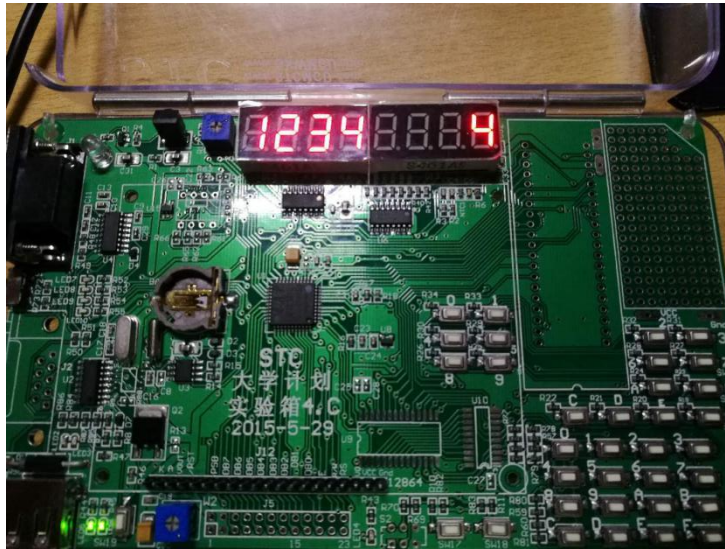
## 模块验证：

### 1. 输入密码并显示：



### 2. 删除一个输入数字：





### 3. 清除输入：



## 实时时钟模块

### 模块简介：

#### 1. 功能

在数码管显示实时时间，每秒切换显示年月日和时分秒，处于时间显示界面时按下任意键能切换到时间显示界面，若处于密码输入界面且还没开锁时，一段时间没有输入会自动清除输入界面并返回时间显示界面。

#### 2. 基本思路



使用 51 单片机内的 pcf8563 芯片来实现实时时钟，通过 I2C 总线向 pcf8563 内的寄存器传送时间初始值可设置时间初值，之后每秒通过 I2C 总线读取 pcf8563 内寄存器的值，并通过数码管扫描显示即可显示实时时间

## 模块解析：

1.I2C 总线：STOP 和 START 状态之间总线空置，数据线（SDA）和时钟线（SCL）保持在高电平。SCL 高电平时 SDA 下降沿，为启动条件（S）；SCL 高电平时 SDA 上升沿为停止条件（P）

```

/*****/
void I2C_Delay(void) //for normal MCS51, delay (2 * dly + 4) T, for STC12Cxxx delay (4 * dly +
{
    u8 dly;
    dly = MAIN_Fosc / 2000000UL; //按2us计算
    while(--dly) ;
}

/*****/
void I2C_Start(void) //start the I2C, SDA High-to-low when SCL is high
{
    SDA = 1;
    I2C_Delay();
    SCL = 1;
    I2C_Delay();
    SDA = 0;
    I2C_Delay();
    SCL = 0;
    I2C_Delay();
}

void I2C_Stop(void) //STOP the I2C, SDA Low-to-high when SCL is high
{
    SDA = 0;
    I2C_Delay();
    SCL = 1;
    I2C_Delay();
    SDA = 1;
    I2C_Delay();
}

void S_ACK(void) //Send ACK (LOW)
{
    SDA = 0;
    I2C_Delay();
    SCL = 1;
    I2C_Delay();
    SCL = 0;
    I2C_Delay();
}

void S_NoACK(void) //Send No ACK (High)
{
    SDA = 1;
    I2C_Delay();
    SCL = 1;
    I2C_Delay();
    SCL = 0;
    I2C_Delay();
}

```

2. 向 pcf8563 寄存器传送和读取值: pcf8563 内存地址 02H~08H 用于时钟计数器（秒~年计数器），将时间初值存入这些寄存器，随后要获取实时时间便从这些寄存器获取即可

```

/*****/
u8 I2C_ReadAbyte(void)    //read A byte from I2C
{
    u8 i,dat;
    i = 8;
    SDA = 1;
    do
    {
        SCL = 1;
        I2C_Delay();
        dat <<= 1;
        if(SDA)    dat++;
        SCL = 0;
        I2C_Delay();
    }
    while(--i);
    return(dat);
}

/*****/
void I2C_WriteAbyte(u8 dat)    //write a byte to I2C
{
    u8 i;
    i = 8;
    do
    {
        if(dat & 0x80)    SDA = 1;
        else                SDA = 0;
        dat <<= 1;
        I2C_Delay();
        SCL = 1;
        I2C_Delay();
        SCL = 0;
        I2C_Delay();
    }
    while(--i);
}

/*****/
void WriteNbyte(u8 addr, u8 *p, u8 number)    /* WordAddress,First Data Address,Byte lenth */
                                              /* F0=0,right, F0=1,error */
{
    I2C_Start();
    I2C_WriteAbyte(SLAW);
    I2C_Check_ACK();
    if(!F0)
    {
        I2C_WriteAbyte(addr);
        I2C_Check_ACK();
        if(!F0)
        {
            do
            {
                I2C_WriteAbyte(*p);    p++;
                I2C_Check_ACK();
                if(F0)    break;
            }
            while(--number);
        }
    }
    I2C_Stop();
}

/*****/
void ReadNbyte(u8 addr, u8 *p, u8 number)    /* WordAddress,First Data Address,Byte lenth */
{
    I2C_Start();
    I2C_WriteAbyte(SLAW);
    I2C_Check_ACK();
    if(!F0)
    {
        I2C_WriteAbyte(addr);
        I2C_Check_ACK();
        if(!F0)
        {
            I2C_Start();
            I2C_WriteAbyte(SLAR);
            I2C_Check_ACK();
            if(!F0)
            {
                do
                {
                    *p = I2C_ReadAbyte();    p++;
                    if(number != 1)    S_ACK();    //send ACK
                }
                while(--number);
                S_NoACK();    //send no ACK
            }
        }
    }
}

```

### 3. 实时时间获取：调用读取和写函数，便能获取 pcf8563 内时间值

```
void ReadRTC(void)
{
    u8 tmp1[4];
    u8 tmp2[2];
    ReadNbyte(2, tmp1, 4);
    ReadNbyte(7, tmp2, 2);
    second = ((tmp1[0] >> 4) & 0x07) * 10 + (tmp1[0] & 0x0f);
    minute = ((tmp1[1] >> 4) & 0x07) * 10 + (tmp1[1] & 0x0f);
    hour = ((tmp1[2] >> 4) & 0x03) * 10 + (tmp1[2] & 0x0f);
    day = ((tmp1[3] >> 4) & 0x0f) * 10 + (tmp1[3] & 0x0f);
    month = ((tmp2[0] >> 4) & 0x0f) * 10 + (tmp2[0] & 0x0f);
    year = ((tmp2[1] >> 4) & 0x0f) * 10 + (tmp2[1] & 0x0f);
}

/***** 读写PCF8563 *****/
void WriteRTC(void)
{
    u8 tmp1[4];
    u8 tmp2[2];

    tmp1[0] = ((second / 10) << 4) + (second % 10);
    tmp1[1] = ((minute / 10) << 4) + (minute % 10);
    tmp1[2] = ((hour / 10) << 4) + (hour % 10);
    tmp1[3] = ((day / 10) << 4) + (day % 10);
    tmp2[0] = ((month / 10) << 4) + (month % 10);
    tmp2[1] = ((year / 10) << 4) + (year % 10);
    WriteNbyte(2, tmp1, 4);
    WriteNbyte(7, tmp2, 2);
}
```

### 模块验证：

#### 1. 实时时间时分秒显示：



#### 2. 实时时间年月日显示：



## 温度模块

### 模块简介：

#### 1. 功能

每隔一段时间将 51 单片机的 NTC 模块获取的温度，发送到服务器上。

#### 2. 基本思路

利用 51 单片机自带的 NTC 模块获取温度，再通过 ADC 模数转换转成离散数字信号以便我们操作

### 模块解析：

#### 1. ADC 配置：设置 ADC 的功率、速度、优先级等参数

```
/****** ADC配置函数 *****/
void ADC_config(void)
{
    ADC_InitTypeDef ADC_InitStructure; //结构定义
    ADC_InitStructure.ADC_Px = ADC_P12 | ADC_P13; //设置要做ADC的IO, ADC_P10 ~ ADC_P17(或操作),ADC_P1_All
    ADC_InitStructure.ADC_Speed = ADC_90T; //ADC速度 ADC_90T,ADC_180T,ADC_360T,ADC_540T
    ADC_InitStructure.ADC_Power = ENABLE; //ADC功率允许/关闭 ENABLE,DISABLE
    ADC_InitStructure.ADC_AdjResult = ADC_RES_H8L2; //ADC结果调整, ADC_RES_H2L8,ADC_RES_H8L2
    ADC_InitStructure.ADC_Polity = PolityLow; //优先级设置 PolityHigh,PolityLow
    ADC_InitStructure.ADC_Interruption = DISABLE; //中断允许 ENABLE,DISABLE
    ADC_Init(&ADC_InitStructure); //初始化
    ADC_PowerControl(ENABLE); //单独的ADC电源操作函数, ENABLE或DISABLE
}
```

2. 计算温度值：使用 12 位的 ADC 值，使用对分查找表格来计算，小数点后一位数是用线性插补来计算的。

```

/***** 计算温度 *****/
// 计算结果: 0对应-40.0度, 400对应0度, 625对应25.0度, 最大1600对应120.0度.
// 为了通用, ADC输入为12bit的ADC值.
/*****/

#define D_SCALE 10 //结果放大倍数, 放大10倍就是保留一位小数
ul6 calculate_temperature(ul6 adc)
{
    ul6 code *p;
    ul6 i;
    u8 j,k,min,max;

    adc = 4096 - adc; //Rt接地
    p = temp_table;
    if(adc < p[0]) return (0xfffe);
    if(adc > p[160]) return (0xffff);

    min = 0; // -40度
    max = 160; // 120度

    for(j=0; j<5; j++) //对分查表
    {
        k = min / 2 + max / 2;
        if(adc <= p[k]) max = k;
        else min = k;
    }
    if(adc == p[min]) i = min * D_SCALE;
    else if(adc == p[max]) i = max * D_SCALE;
    else // min < temp < max
    {
        while(min <= max)
        {
            min++;
            if(adc == p[min]) {i = min * D_SCALE; break;}
            else if(adc < p[min])
            {
                min--;
                i = p[min]; //min
                j = (adc - i) * D_SCALE / (p[min+1] - i);
                i = min;
                i *= D_SCALE;
                i += j;
                break;
            }
        }
    }
    return i;
}

```

3. 计算温度具体值:测温度的 ADC3 进行 4 次 ADC 连续采样, 变成 12 位的 ADC 来计算温度.

```

ul6 get_temperature()
{
    ul6 j;
    j = Get_ADC10bitResult(2);
    j = Get_ADC10bitResult(3); //参数0~7,查询方式做一次ADC, 返回值就是结果, == 1024 为错误
    j += Get_ADC10bitResult(3);
    j += Get_ADC10bitResult(3);
    j += Get_ADC10bitResult(3);

    if(j < 1024*4)
    {
        j = calculate_temperature(j); //计算温度值

        if(j >= 400) F0 = 0, j -= 400; //温度 >= 0度
        else F0 = 1, j = 400 - j; //温度 < 0度
    }

    return j;
}

```

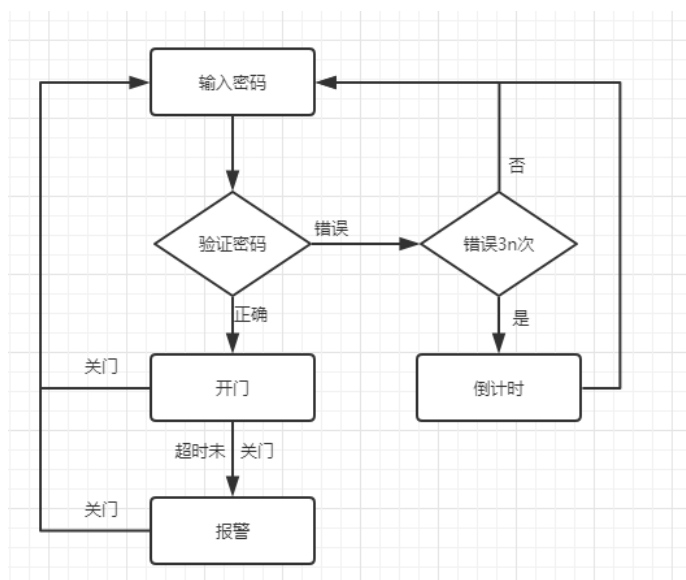
模块验证:

1. 数码管测试显示温度:



## 密码验证与解锁模块

### 模块简介：



### 1. 功能

密码输入动作完成，使用“C”键确认，对输入的密码进行验证，若密码正确，则密码输入位显示：“111111”，解锁成功；若密码错误，则输入位清空（变暗），需要重新输入，错误次数为  $3n$  次（ $n$  为正整数）时，进入倒计时并向服务端报警，倒计时阶段，显示倒数时间且无法执行任何操作，倒数时间为  $3n$  秒。初始化密码为“123456”。

## 2. 基本思路

使用 C 键作为确认键，按下“C”键，则调用密码验证函数 validate()

若密码错误，清空密码，重新输入，输入次数为 3n(n 为整数)时，则进入倒计时并调用 Alarm()报警，倒计时时长也为 3n 秒，在倒计时阶段无法进行任何操作。

模块解析：

使用 C 键作为确认键，按下“C”键，则调用密码验证函数 validate()：

```
else if(KeyCode == 29) //确认密码
{
    validate();
}
```

在 validate() 函数中，我们通过对密码长度和密码内容来判断密码是否正确，

若密码错误，则 inputcount 自增 1

```
do {
    inputcount++;
    if(count < 6) //密码长度不对，密码错误
    {
        inputcount++;
    }
    else
    {
        for(i = 0; i < 6; i++)
        {
            if(input[i] != pass[i])
            {
                inputcount++; //密码错误
                break;
            }
        }
    }
}
```



下图中，t 初始为 inputcount，在验证过程中，若密码错误，则 inputcount 会改变，若 inputcount 不变，则说明密码正确，此时将门的状态 state 置 1，说明门已打开；UnlockSuccess()函数时用于往服务端发送开锁成功的消息。

```
if(t == inputcount)
{
    for(i = 0;i < 6;i++)
        input[i] = 1;          //密码正确，全部显示1
    count = 1;                //尾数显示1
    inputcount = 0;
    time = 0;
    state = 1;
    DisplayPassword(input, count);
    // open_success();
    SendState();
    UnlockSuccess();
    return ;
}
```

若密码错误，清空密码，重新输入，输入次数为 3n(n 为整数)时，则进入倒计时并调用 Alarm()报警，倒计时时长也为 3n 秒，在倒计时阶段无法 进行任何操作

```
else
{
    for(i = 0;i < 6;i++)
        input[i] = 10;
    count = 0;
    DisplayPassword(input, count);
}
if(inputcount == 3)//三次错误
{
    //倒计时
    count_down();
    Alarm("Wrong password for three times\0");//报警
}
```

模块验证：

输入正确的密码，按“C”，打开门，显示“111111”



倒计时模块：

模块简介：

### 1. 功能

用户输入密码错误 3n 次后往服务器报警并进入 3n 秒的倒计时, n 为进入倒计时的次数，即输入次数越多，倒计时越长。，倒计时期间无法进行任何操作，待倒计时结束后可再次输入密码。

### 2. 基本思路

在倒计时内使用循环阻塞进程，delay\_ms 为记时函数，inputcount 逐秒递减，倒计时结束后返回密码输入界面。

模块解析：

在倒计时内使用循环阻塞进程，delay\_ms 为记时函数，inputcount 逐秒递减，倒计时结束后返回密码输入界面。

```
while(1)
{
    delay_ms(1); //延时 1ms

    if(++msecond >= 1000) //1 秒到
    {
        inputcount--; //倒计时秒计数-1
        msecond = 0; //清 1000ms 计数
        input[3] = inputcount / 100;
        input[4] = (inputcount % 100) / 10;
        input[5] = inputcount % 10;
    }
    DisplayPassword(input, count);
    if(inputcount == 0)
    {
        u8 i;
        for(i = 0; i < 6; i++)
            input[i] = 10;
        break;
    }

}
for (i = 0; i < 7; ++i)
{
    input[i] = 10;
}
DisplayPassword(input, count);
count = 0;
```

## 模块验证：

输入三次错误密码：进入倒计时



异常报警模块：

## 模块简介：

### 1. 功能

在门打开的状态下，若超过一定时间未关门，则会向服务器报警并显示，，用户打开门后可按'D'键关闭。

### 2. 基本思路

使用 state 来表示门的状态，0 为关闭，1 为开启，3 为超时未关闭，门开启超过一定的时间上限则将 state 置为 3，此时函数检测到该状态，就会使 P47 这个信号灯闪烁，以提示用户关门，同时会给服务端发送门未关闭消息。

若按下关门按键'D'，则 state 将置 0，关门成功，进入密码输入界面

## 模块解析：

使用 state 来表示门的状态，0 为关闭，1 为开启，3 为超时未关闭，门开启超过一定的时间上限则将 state 置为 3，此时函数检测到该状态，就会使 P47 这个信号灯闪烁，以提示用户关门，同时会给服务端发送门未关闭消息。

```
if(mstime++ >= 1000)
{
    if(state == 1)
    {
        stime++;
        if(stime >= overTime) // 超时未锁门
        {
            state = 3;
            Alarm("Lock is opened for 60s\0");//报警
        }
    }
    else
    {
        stime = 0;
    }
    if(state == 3)
    {
        if( clockCount == 0)
        {
            clockCount = 1;
            P47 = 1;
        }
        else
        {
            clockCount = 0;
            P47 = 0;
        }
    }
    mstime = 0;
}
```

若按下关门按键'D'，则 state 将置 0，关门成功，进入密码输入界面

```
else if(KeyCode == 30) //关门
{
    for ( i = 0; i < 7; ++i)
    {
        input[i] = 10;
```



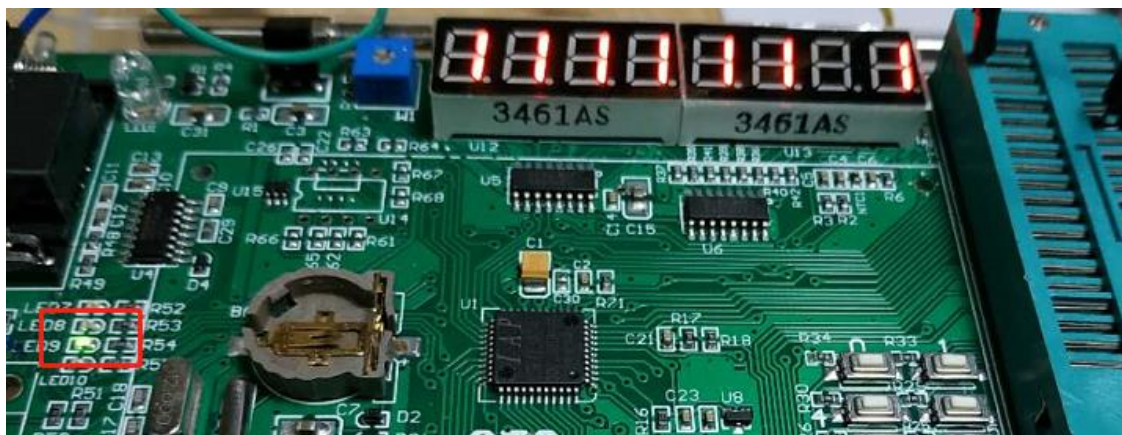
```

}
count = 0;
state = 0;
P47 = 1;
DisplayPassword(input, count);
SendState();
}

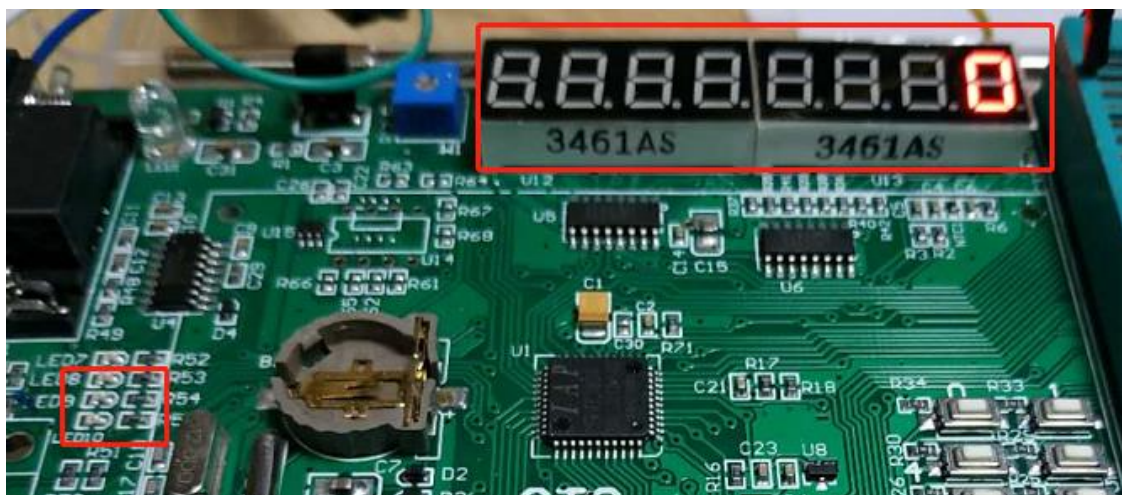
```

## 模块验证：

等待一段时间后（几秒）P47 闪烁，说明超时未关门



此时按下“D”，则关门并回到输入密码界面，P47 熄灭



单片机 WIFI 模块：

## 模块简介：

## 1. 功能

连接指定的 WIFI，WIFI 连接完成后和服务器建立 TCP 连接从而与服务器通讯。

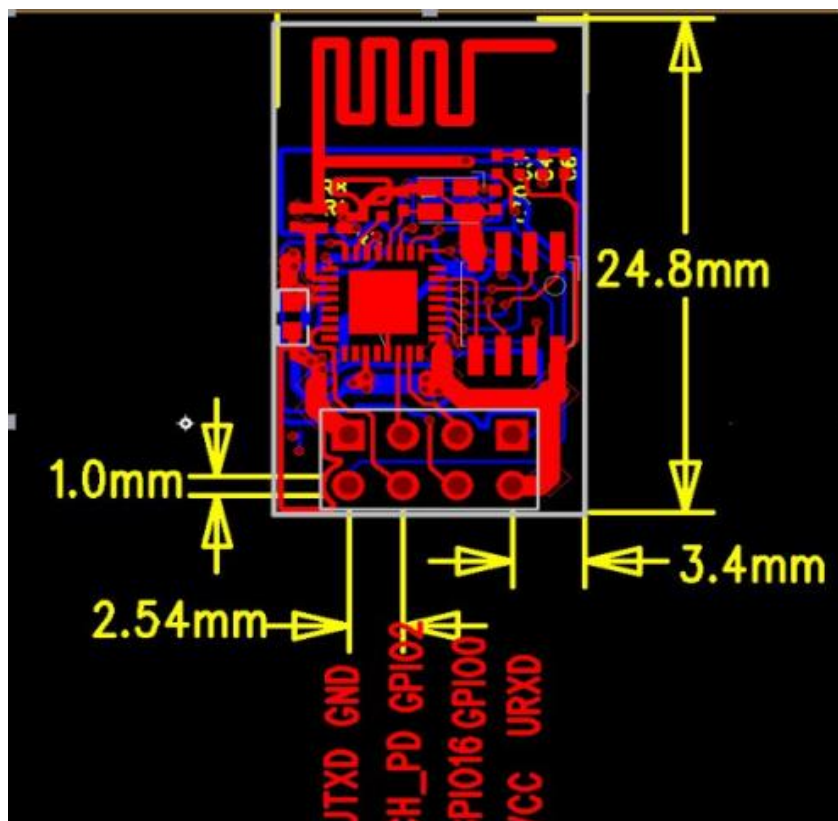
## 2. 基本思路

使用 ESP8266 模块，与单片机 P3.0，P3.1 串口相连。单片机通过串口与模块进行通信，模块通过 TCP 连接与服务器通讯。

模块解析：

连线方式：

WIFI 模块，我们组采用的是 ESP8266 芯片，其中的接线图是这样的：



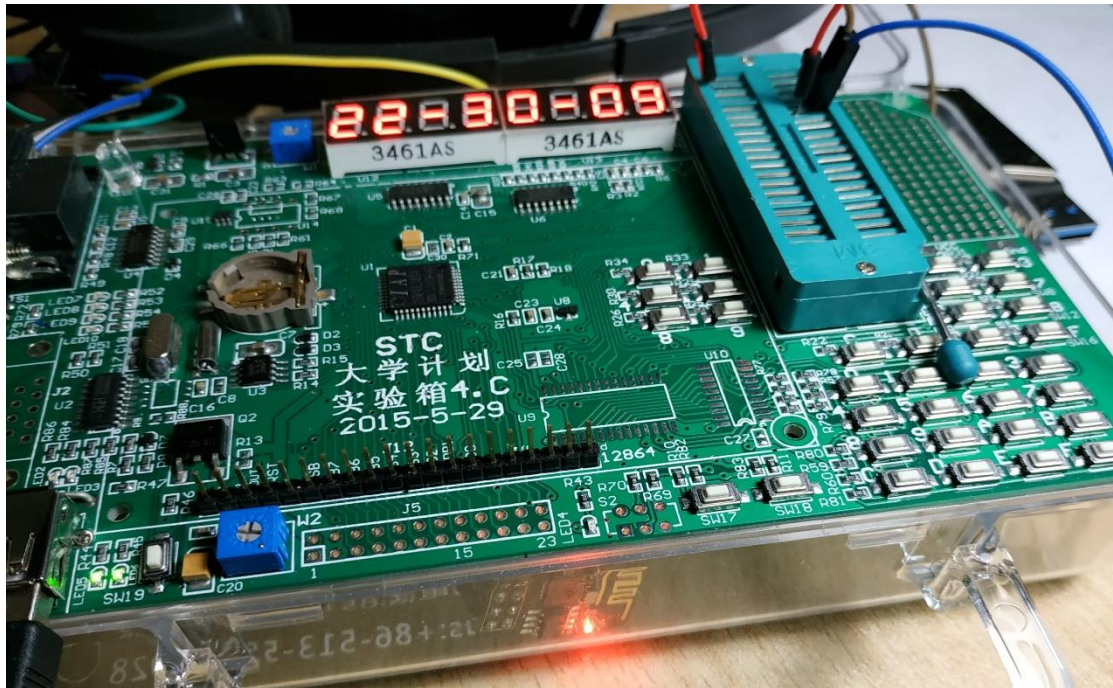
需要使用 GND,VCC,TX,TX,CH\_PD 口。

其中，vcc 负责供电，esp8266 模块要求的电压为 3.3v，而我们的单片机只能提供 5v 的电源，为了防止模块烧坏，于是我们用 usb-ttl 来负责供电。Tx-rx，Rx-tx 负责通信，gnd 接地，ch\_pd 为使能端口，默认接高。其他的口悬空即可。



TX 接入单片机 P1.6\_RxD 端口，RX 接入单片机 P1.7\_TxD 端口。

连线图片：



模块实现：

主要内容实现在 STC\_NET.h 和 STC\_NET.c 中，在 main.c 中添加了少数由 STC\_NET.h 中提供的方法构造的函数。

1. UART1\_config 函数，用于设置使用的串口，以及串口通讯的波特率。

```
void UART1_config() // 选择波特率
{
    /***** 波特率使用定时器 1 *****/
    //115200bps@11.0592MHz
    SCON = 0x50; //8 位数据,可变波特率
    AUXR |= 0x40; //定时器 1 时钟为 Fosc,即 1T
    AUXR &= 0xFE; //串口 1 选择定时器 1 为波特率发生器
    TMOD &= 0x0F; //设定定时器 1 为 16 位自动重装方式
    TL1 = 0xE8; //设定定时初值
    TH1 = 0xFF; //设定定时初值
    ET1 = 0; //禁止定时器 1 中断
    TR1 = 1; //启动定时器 1
    /*****/
    ES = 1; //允许中断
}
```

```

    REN = 1; //允许接收
    P_SW1 &= 0x3f;
    P_SW1 |= 0x80; //UART1 switch to, 0x00: P3.0 P3.1, 0x40: P3.6 P3.7, 0x80: P1.6 P1.7 (必须使用内部时钟)
    B_TX1_Busy = 0;
    RX1_Cnt = 0;
}

```

2. 串口通讯中断处理函数。串口已经设置为全双工，所以要分别处理发送、接收的中断。其中接收数据时，每次收到换行符号代表传输已结束，将缓冲区下一位设置为\0。

```

void UART1_int(void) interrupt 4
{
    if (RI)
    {
        RI = 0;
        RX1_Buffer[RX1_Cnt] = SBUF;
        if (RX1_Buffer[RX1_Cnt] == '\n'){
            RX1_Buffer[RX1_Cnt] = '\0';
            RX1_Cnt = 0;
        }
        else if(++RX1_Cnt >= UART1_BUF_LENGTH){
            RX1_Cnt = 0;
        }
    }

    if (TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}

```

3. 串口 1 发送字符串的函数，在传入的字符串中遇到\0 之前持续发送字符。

```

void PrintString1(u8 *puts) //发送一个字符串
{
    for (; *puts != '\0'; puts++) //遇到停止符 0 结束
    {
        SBUF = *puts;
        B_TX1_Busy = 1;
        while (B_TX1_Busy);
    }
}

```

```
}
```

4. 每次串口接收到传入的字符串后，都需要判断是否为需要的字符串，使用 bit

BuffCMP(u8\* cmp)函数将缓冲区字符串与传入字符串进行比对判断。

```
bit BuffCMP(u8* cmp)
{
    u8 i;
    strncpy(RX1_Buffer_Temp,RX1_Buffer,32);
    for (i = 0; *cmp != '\0' ; i++, cmp++)
    {
        if(*cmp != RX1_Buffer_Temp[i]){
            return 0;
        }
    }
    //清除接收区缓存
    for(i = 0;i<32;i++){
        RX1_Buffer[i]='\0';
    }
    return 1;
}
```

5. 用于 ESP8266 模块启动到连接 WIFI 的三个函数，都是保证一定要完成，否则会阻塞进程直到完成。

```
bit ATConnect()
{
    do{
        PrintString1("AT\r\n");
        delay_ms(500);
    }
    while(!BuffCMP("OK"));
    return 1;
}

bit ATMode()
{
    do{
        PrintString1("AT+CWMODE_CUR=1\r\n");
        delay_ms(500);
    }
    while(!BuffCMP("OK"));
    return 1;
}
```

```
}
```

```
bit ATWifi(u8* username,u8* password)
```

```
{
```

```
    u8 xdata temp[60];
```

```
    sprintf(temp,"AT+CWJAP=\"%s\\\", \"%s\\r\n\",username,password);
```

```
    do{
```

```
        PrintString1(temp);
```

```
        delay_ms(5000);
```

```
    }
```

```
    while(!BuffCMP("OK"));
```

```
    return 1;
```

```
}
```

## 6. 通过传入的 TCP 服务器地址和端口连接 TCP 服务器。

```
bit TCPConnect(u8* address,u8* port)
```

```
{
```

```
    u8 xdata temp[50];
```

```
    sprintf(temp,"AT+CIPSTART=\"TCP\\\", \"%s\\\", %s\r\n",address,port);
```

```
    do{
```

```
        PrintString1(temp);
```

```
        delay_ms(2000);
```

```
    }
```

```
    while(!BuffCMP("OK"));
```

```
    return 1;
```

```
}
```

## 7. TCP 服务器连接后，通过该函数发送字符串到服务器。

```
bit TCPSend(u8* message){
```

```
    int len;
```

```
    u8 xdata temp[24];
```

```
    len = StringLen(message);
```

```
    sprintf(temp,"AT+CIPSENDER=%d\r\n\0",len);
```

```
    PrintString1(temp);
```

```
    do{
```

```
        delay_ms(50);
```

```
    }
```

```
    while(!BuffCMP(">"));
```

```
    PrintString1(message);
```

```
    do{
```

```
        delay_ms(50);
```

```
    }
```

```
    while(!BuffCMP("SEND OK"));
```

```

        return 1;
    }

```

8. 主函数中发送状态码、温度、警告以及解锁成功消息的函数。

规定的消息格式：

单片机 TCP 消息格式：

0. 0000-[门]锁名称] 门]锁连接 FTP

1. 0001-[xx] 门]锁状态

2. 0002-[xx] 温度信息

3. 0003-[xx] 异常告警

4. 0004-[xx] 门]锁开锁成功

void SendState() //状态

```

{
    u8 xdata temp[10];
    sprintf(temp,"0001-%d\0",state);
    TCPSend(temp);
}

```

void SendTemperature(u16 temperature){ //发送温度

```

    u8 xdata temp[10];
    sprintf(temp,"0002-%d\0",temperature);
    TCPSend(temp);
}

```

void Alarm(u8\* msg) //报警

```

{
    //每输入三次错误，就会调用这个函数，往服务端发送报警信号
    u8 xdata temp[50];
    sprintf(temp,"0003-%s\0",msg);
    TCPSend(temp);
}

```

void UnlockSuccess(){ //发送解锁成功消息

```

    u8 xdata temp[10];
    sprintf(temp,"0004-success");
    TCPSend(temp);
}

```

## 模块验证：

WIFI 模块要连接 WIFI 及 TCP 服务器，所以先运行 TCP 服务器。以下截图为服务

器收到的数据。

```
>S E:\OneDrive\OneDrive - mail2.sysu.edu.cn\嵌入式软件开发\InternetLock\EmbeddingServer> go run server.go
Now listening on: http://localhost:3000
Application started. Press CTRL+C to shut down.
2019/01/16 18:56:59 Open TCP Server
2019/01/16 18:56:59 [TCP Address] :3333
2019/01/16 18:56:59 [TCP Listen On]: :3333
```

为了便于调试，将单片机 P3.1\_RxD 与 P1.6\_RxD 接口相连，从而在 PC 与单片机相连的 USB 串口读取 ESP8266 模块发送给单片机的数据。

1. 单片机启动后 main 函数运行下述函数，服务器接收到单片机发送的数据。

```
//WIFI 连接初始化
ATConnect();
ATMode();
ATWifi("test","test123456");
//连接 TCP 服务器
TCPConnect("172.18.32.6","3333");
TCPSend("0000-example"); //该锁的名称

2019/01/16 18:57:14 New Connect
2019/01/16 18:57:15 0000-example
2019/01/16 18:57:15 [Request] 0000-example
2019/01/16 18:57:15 [LockName] example
```

2. 每隔 30ms 发送一次单片机获取的温度。

```
if(++temperatureCount >= 30000){ //30s 更新一次温度
    temperatureCount = 0;
    SendTemperature(get_temperature());
}

2019/01/16 19:01:41 [Request] 0002-231
2019/01/16 19:02:11 0002-230
2019/01/16 19:02:11 [Request] 0002-230
2019/01/16 19:02:41 0002-230
```

3. 解锁成功后向服务器发送门锁状态并通报开锁成功。

```
2019/01/16 20:06:42 0001-1
2019/01/16 20:06:42 [Request] 0001-1
2019/01/16 20:06:42 0004-success
2019/01/16 20:06:42 [Request] 0004-success
```

4. 60s（为了测试方便这里时实际上是 20s）未关门（按下按键 D）报警。

```
2019/01/16 20:07:01 0003-Lock is opened for 60s
2019/01/16 20:07:01 [Request] 0003-Lock is opened for 60s
2019/01/16 20:07:01 [Warning] Lock is opened for 60s
```

## 服务器模块：

### 模块简介：

#### 1. 功能

将单片机数据存储到云服务器。方便安卓手机获取和查询智能门禁系统的信息。

#### 2. 基本思路

由于不是课程重点，所以只简要叙述。运行服务器前需要配置 GO 语言开发环境。

使用 GO 语言 Iris 框架作为 HTTP 服务器框架，提供四种请求格式。

1. GET localhost:3000/api/allUnlockingRecord
2. GET localhost:3000/api/WarningRecord
3. GET localhost:3000/api/LockState
4. Post localhost:3000/api/Unlocking

具体请参阅 **API 文档**。

### 模块解析：

#### 1. 处理各部分请求并按 API 要求返回数据。

```
app.Get("/api/allUnlockingRecord", func(ctx iris.Context) {
    msg := respData{
        Status: "200",
        Msg:     "获取成功",
        Record: records}
    b, err := json.Marshal(msg)
    if err == nil {
        ctx.WriteString(string(b))
    }
})
app.Get("/api/LockState", func(ctx iris.Context) {
    var msg respData
    msg.Status = "200"
    log.Println("[门锁状态]", lockState)
    if lockState == 0 {
        msg.Msg = "门锁状态正常"
```



```

    } else if lockState == 1 {
        msg.Msg = "门锁开启"
    } else {
        msg.Msg = "门锁状态异常，超时未关闭，请检查门锁"
    }
    temp := strings.Split(temperature, "")
    msg.Msg += "\n 当前气温: " + temp[0] + temp[1] + "." + temp[2]
    b, err := json.Marshal(msg)
    if err == nil {
        ctx.WriteString(string(b))
    }
})
app.Get("/api/WarningRecord", func(ctx iris.Context) {
    msg := respData{
        Status: "200",
        Msg:     "获取成功",
        Record: warningRecords}
    b, err := json.Marshal(msg)
    if err == nil {
        ctx.WriteString(string(b))
    }
})
app.Post("/api/Unlocking", func(ctx iris.Context) {
    password := ctx.FormValue("password")
    var msg respData
    msg.Status = "200"
    if password == lockPassword {
        fmt.Println(onlineLock)
        conn, exits := onlineLock[lockName]
        fmt.Println(exits)
        if !exits {
            msg.Msg = "UnlockFailed\n 电子锁不在线"
        } else {
            _, err := (*conn).Write([]byte("password " + password + "\n"))
            if err != nil {
                msg.Msg = "UnlockFailed\n 电子锁状态异常"
            } else {
                msg.Msg = "UnlockOK"
                timestamp := time.Now().Unix()
                tm := time.Unix(timestamp, 0)
                currentTime := tm.Format("2006-01-02 15:04:05")
                fmt.Println("[UnlockTime]", currentTime)
                newRecord := record{Time: currentTime}
                msg.Record = []record{newRecord}
            }
        }
    }
})

```

```

        records = append(records, newRecord)
    }
} else {
    msg.Msg = "UnlockFailed"
}
b, err := json.Marshal(msg)
if err == nil {
    ctx.WriteString(string(b))
}
})

```

2. 运行一个新 GO 程（类似线程）创建一个 TCP 服务器，TCP 服务器对传入数据进行处理。

```

//TCP
go newTCP()
func newTCP() {
    log.Println("Open TCP Server")
    service := ":3333"
    tcpAddr, err := net.ResolveTCPAddr("tcp4", service)
    checkError(err)
    log.Println("[TCP Address]", tcpAddr)
    listener, err := net.ListenTCP("tcp", tcpAddr)
    checkError(err)
    log.Println("[TCP Listen On]:", tcpAddr)
    for {
        conn, err := listener.Accept()
        if err != nil {
            continue
        }
        go handleConn(conn)
    }
}

```

```

/*
单片机 TCP 消息格式：
0. 0000-[门锁名称] 门锁连接 FTP
1. 0001-[xx] 门锁状态
2. 0002-[xx] 温度信息
3. 0003-[xx] 异常告警
4. 0004-[xx] 门锁开锁成功
*/

```

```

func handleConn(conn net.Conn) {
    defer conn.Close() // close connection before exit
    defer log.Println(lockName, "Connect End")
    log.Println("New Connect")
    inputString, err := readInput(conn)
    checkError(err)
    log.Println("[Request]", inputString)
    temp := strings.SplitN(inputString, "-", 2)
    if len(temp) < 2 {
        lockName = "Undefine"
    } else {
        lockName = temp[1]
    }
    log.Println("[LockName]", lockName)
    onlineLock[lockName] = &conn
    defer delete(onlineLock, lockName)
    for {
        inputString, err := readInput(conn)
        if err != nil {
            log.Println(err)
            break
        }
        log.Println("[Request]", inputString)
        temp := strings.SplitN(inputString, "-", 2)
        if len(temp) < 2 {
            continue
        }
        stateCode := temp[0]
        message := temp[1]
        // fmt.Println("[Message]", message)
        switch stateCode {
        case "0001":
            var err error
            lockState, err = strconv.Atoi(message)
            checkError(err)
            break
        case "0002":
            temperature = message
            break
        case "0003":
            timestamp := time.Now().Unix()
            tm := time.Unix(timestamp, 0)
            currentTime := tm.Format("2006-01-02 15:04:05")
            log.Println("[Warning]", message)
        }
    }
}

```

```

        newRecord := warningRecord{Time: currentTime, Content: message}
        warningRecords = append(warningRecords, newRecord)
        lockState = 2
        break
    case "0004":
        timestamp := time.Now().Unix()
        tm := time.Unix(timestamp, 0)
        currentTime := tm.Format("2006-01-02 15:04:05")
        fmt.Println("[UnlockTime]", currentTime)
        newRecord := record{Time: currentTime}
        records = append(records, newRecord)
        break
    }
}
}

func readInput(conn net.Conn) (string, error) {
    request := make([]byte, 256) // set maxium request length to 256B to prevent flood attack
    _, err := conn.Read(request)
    log.Println(string(request))
    return string(bytes.Trim(request, "\x00")), err
}

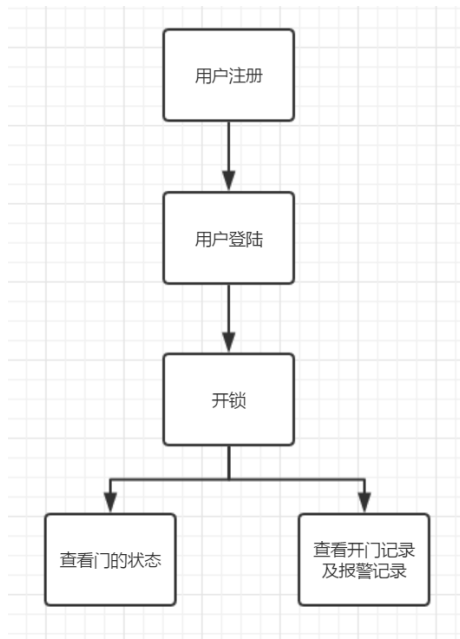
func checkError(err error) {
    if err != nil {
        fmt.Fprintf(os.Stderr, "Fatal error: %s", err.Error())
        os.Exit(1)
    }
}

```

## 模块验证：

模块验证和单片机网络模块以及安卓应用模块验证的并集相等，所以不再赘述。

## 安卓模块：



模块简介：

## 1. 功能

手机端 APP 实现了用户登陆注册，在线开锁，查看门的状态，查看开门记录以及查看报警记录的功能。

## 2. 基本思路

用户的登陆注册是基于本地的数据库进行，开锁，查看门的状态及记录等功能是利用服务器的 API，获取到信息进行的。

模块解析：

### 1. 用户注册登陆：

用户可现在注册界面注册账号以及密码，然后注册成功之后会自动跳转到索引页。在下次使用 app 时，可直接使用注册的账号及密码进行登陆，进入索引页。

### 2. 开锁

用户点击开锁，向服务端发送 POST 请求，发送密码，打开锁。

### 3. 查看门的状态

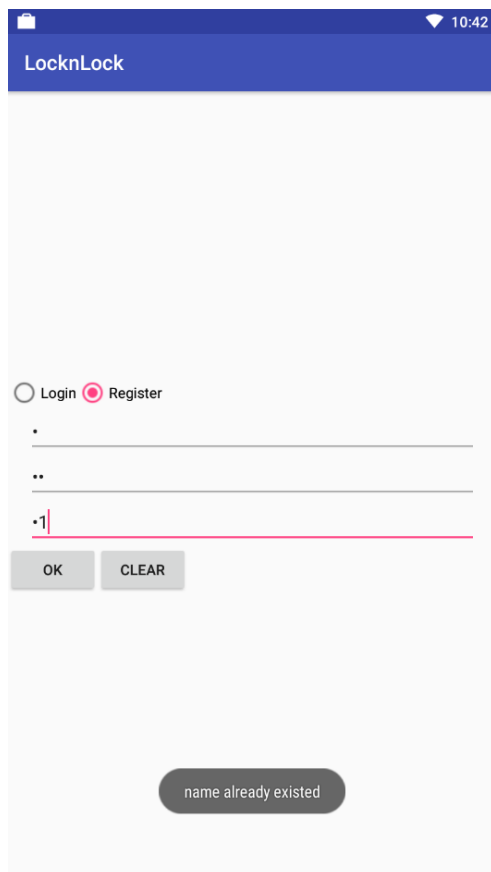
用户点击查看状态，向服务端发送 GET 请求，获取到门的状态信息以及当前的温度，并在手机界面上直接显示出来。

### 4. 查看开门记录及报警记录

用户点击查看记录或查看报警记录，向服务端发送 GET 请求，获取到对应的记录后，在新的界面显示开门时间，开门人，以及报警信息。

## 模块验证：

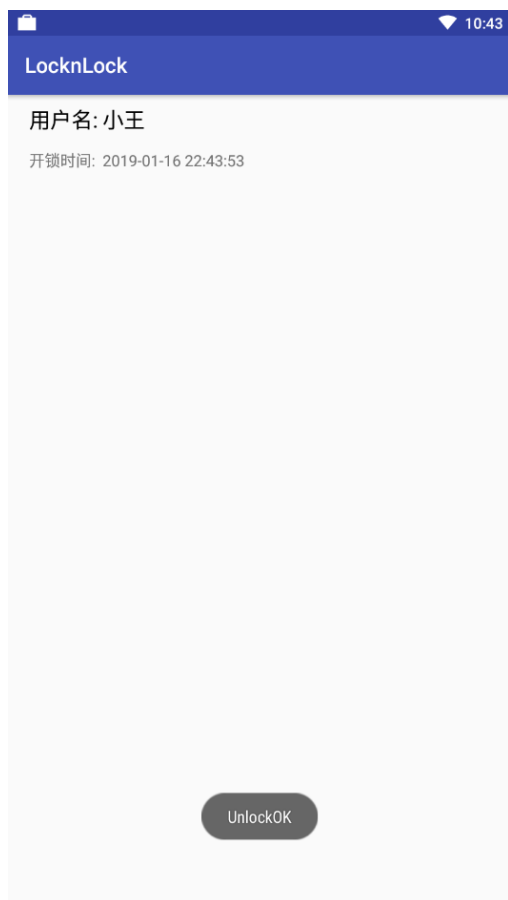
1.



(该用户名已存在无法注册)



(注册成功)



(开锁成功，显示开锁时间以及开锁人)





(查看开门记录)



(查看状态以及温度)



(显示报警记录, 包括时间以及报警信息)

### 3. 软件设计

Stc-isp 进行程序烧写

Android studio

Keil

## 第四章：总结

### 遇到的问题及解决方法

1. 添加新的显示元素时，数码管无法显示在添加的元素后面加上扫描函数
2. 使用循环实现报警计时的时候会阻塞按键结合中断标志实现计时功能
3. 一开始不太清楚 pcf8563 芯片的寄存器对应的功能，导致年月的显示有问题。后来发现在日期之后的寄存器对应的是星期，后面才是年月，修改写入读取地址才解决问题。
4. 生成 target 时遇到文件过大的问题，用注册机破解后仍有该问题，后来太多函数编译了不调用也会占用单片机内存导致这个问题，因此测试时把当前用不到的函数注释掉便解决了。
5. WIFI 模块使用时主要使用了串口通讯，由于对串口通讯不熟练，所以走了很多弯路。通过借阅部分单片机开发书籍，反复翻阅资料最终熟练使用了串口通讯。
6. 安卓端在向服务端请求时，出现没有收到返回信息的情况。原因是接受 json 信息时，接受类的属性类型不符出现错误。

### 实验总结

在本次期末 project 的完成过程中，我们组有了期中项目的基础。在单片机的基础上基本完成了所有的主要功能。单片机的许多功能需要一些额外的拓展模块，我们组的思路是在淘宝上购买模块，比如串口转 WiFi 模块等等。在一起讨论，

分工。在本次实验中，最重要的应该是网络模块了。每一个模块都与之相关。我们组做了 tcp 服务器。因为该系统要建立连接自行传输。所以不能选择 http 的服务器。这个系统很方便的地方在于它支持实时用手机查看和控制门的状态，这就需要写一个安卓应用，我们的 app 可以开锁，显示记录，显示状态，显示报警记录。同时我们约定门锁状态码，0 是关闭，1 是开启，其他值是超时未关闭。在完成温度模块时，keil 总是会编译失败。最后发现是有两个函数没有调用，单片机空间不足导致的。需要使用 keil 注册机破解。等等。在完成实验过程中，遇到了许多意想不到的问题。好在都一一解决了。

### **未解决的问题**

基本完成了所有的主要功能。由于缺少 GPS 模块，所以没有添加获取锁的位置。但是购买模块后，从模块中读取出经纬度并发送给服务器，服务器发送给客户端可以在安卓应用上显示电子锁的位置。

### **我们的建议**

在教学中，希望老师可以适当多添加一些可以实现不同功能的模块相关的实验。比如串口通讯的实验，wifi 模块的使用。这样在完成期末 project 时，可以轻松一些。