- **实验目的**

1. 练习 C51 定时器/计数器、中断使用;
2. 练习并口输入输出（按键、数码管）原理和应用;
3. 了解动态扫描、按键消抖原理及程序实现;
4. 思考嵌入式系统软件特点和合理结构，尝试设计较复杂逻辑关系的程序。

- **实验条件**

1. Keil 、STC-B 学习板
2. Stp-ISP 软件

- **实验要求**

1. 数码管上显示“888--888”共 8 位数码，其中左边 3 位显示按键 K3 按动次数，右边 3 位显示按键 K1 按动次数。
2. 本实验须应用（至少）“定时器/计数器”、“中断”，不限制使用更多资源。
3. 实验完成过程中，可使用前面实验熟悉和掌握的“模拟/仿真”技术手段。
4. 在基本内容基础上，设计为一个按键按动比赛程序。比赛方式可以自行设计。
5. 模块化编程，提高代码复用程度。

- **实验内容**

游戏分为三个步骤:

1、等待模式:

```
while(1){
  if(flag_500us  ==1) {Func_500us();}
  if(flag_1ms   ==1) {Func_1ms();    }
  if(flag_10ms  ==1) {Func_10ms();   }
  if(flag_100ms==1) {Func_100ms();print_Light(WATER);WATER=(WATER<<1)|WATER; }
  if(flag_1s    ==1) {Func_1s();break;}
}
```

每 100ms 将流水灯左移。1s 后退出等待模式进入游戏模式

2、进入游戏模式前先初始化按键和数码管

```
print_Light(0);//流水灯初始化
key_init();//按键初始化
key_enable(1);//使能key1
key_enable(3);//使能key2
set_Beep(1500,1);//蜂鸣器1500hz，响1s
Beep_ON();//开启蜂鸣器，立刻开始响
count_1s=0;//时间计时
WATER=0x01;//流水灯，流水灯开关在上一个模式已经开启
segment_ON();//开启数码管
print_Light(WATER);//流水灯显示
```

3、进入游戏模式:

```
while(1){
  if(flag_500us  ==1){Func_500us();}
  if(flag_1ms   ==1) {Func_1ms();    }
  if(flag_10ms  ==1) {Func_10ms();   }
  if(flag_100ms==1) {Func_100ms(); }
  if(flag_1s    ==1) {Func_1s();print_Light(WATER);WATER<<=1;}
    //每1s，流水灯左移一位
  number=key_push();//获取按键信息
  if(number){game_rule(number);}
  //按键被按下，对应选手比分+1
  if(count_1s==8) break;//8s，比赛结束
}
```

4、结果显示

由于数码管&&流水灯模块不能进行复杂的行为，如不同频率，所以通过 segment_OFF(); Light_OFF();关闭模块扫描。自己编写一个新的数码管扫描代码。

```
void win_display(){
  P0=0;
  P23=0;
  if(win_pos==8)win_pos=0;
  P2=win_pos;
  if(win_pos>=win_begin && win_pos<=win_end){
    //win_begin和win_end是胜者得分所在的数码管
    //win_begin和win_end在游戏结束时被设置
    win_count++;
    if(win_count<=100){//闪烁显示胜者信息
      P0=Dis[win_pos]==0x7f?0x40:LED_Data[Dis[win_pos]];
    }
    else
      P0=0;
    if(win_count>=200) win_count=0;
  }
  else{
    P0=Dis[win_pos]==0x7f?0x40:LED_Data[Dis[win_pos]];
  }
  win_pos++;
}
```

```
if(player1<player2){
  win_begin=0;
  win_end=2;
}
else if(player1>player2){
  win_begin=5;
  win_end=7;
}
else{
  win_begin=0;
  win_end=7;
}
```

1、**实验重点**

STC15W60S2 提高了 14 个中断请求源，他们分别是外部中断 INT0，定时器 0 中断，外部中断 INT1、定时器 1 中断、串口 1 中断、AD 转化中断、低压检测中断、串口 2 中断、SPI 中断、外部中断 2INT2、外部中断 INT3、定时器 2 中断以及外部中断 4INT4。而本次我使用到的是定时器 0 中断和 INT0 外部中断。

1、首先讨论与定时器 0 有关的寄存器：

TCON：定时器/计数器中断控制寄存器（可位寻址）

SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	name	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF0：T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

TR0：定时器T0的运行控制位。该位由软件置位和清零。当GATE（TMOD.3）=0，TR0=1时就允许T0开始计数，TR0=0时禁止T0计数。当GATE（TMOD.3）=1，TR1=0且INT0输入高电平时，才允许T0计数。

TMOD　　地址：89H　　复位值：00H

不可位寻址

7	6	5	4	3	2	1	0
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
定时器1				定时器0			

TMOD.3/	GATE	TMOD.3控制定时器0,置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。
TMOD.2/	C/$\overline{T}$	TMOD.2控制定时器0用作定时器或计数器，清零则用作定时器(对内部系统时钟进行计数)，置1用作计数器(对引脚T0/P3.4的外部脉冲进行计数)
TMOD.1/TMOD.0	M1、M0	定时器/计数器0模式选择
	0　0	16位自动重装定时器，当溢出时将RL_TH0和RL_TL0存放的值自动重装入TH0和TL0中。
	0　1	16位不可重装载模式，TL0、TH0全用
	1　0	8位自动重装载定时器，当溢出时将TH0存放的值自动重装入TL0
	1　1	不可屏蔽中断的16位自动重装定时器

AUXR格式如下：

AUXR：辅助寄存器

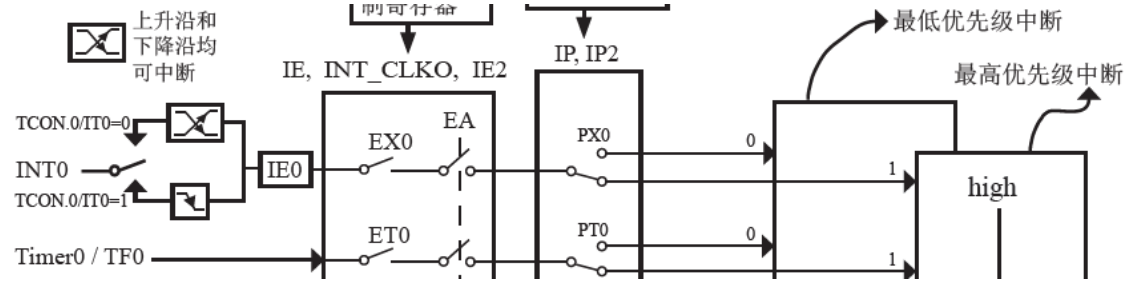
SFR name	Address	bit	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	name	T0x12	T1x12	UART_M0x6	T2R	T2_C/$\overline{T}$	T2x12	EXTRAM	S1ST2

T0x12：定时器0速度控制位

0，定时器0是传统8051速度，12分频；

1，定时器0的速度是传统8051的12倍，不分频

2、定时器 0 中断和 INT0 的产生的电路图:



所以我们可以通过下列函数初始化我们的定时器。

```
void Timer0_Init(void)      //1mS@12.000MHz
{ AUXR &= 0x7f;
  TMOD &= 0xf0;       //使用定时器0，16位重装载模式
  TH0=(65535-500)/256; //高8位赋初值
  TL0=(65535-500)%256; //低8位赋初值
  TR0=1;             //启动定时器1
  ET0=1;                  //开启定时器0中断
}
```

3、模块化编程

本次实验我进行了两套代码的编写，一套是纯粹的为了解决这个需求而编写，另一套（本报告）是为了解决一类需求而编写，下面介绍的是后者，各个模块。

①Beep.h

Beep 模块实现了蜂鸣器的封装，可以设定蜂鸣器的时间和频率。

Beep 蜂鸣器的 API 如下：

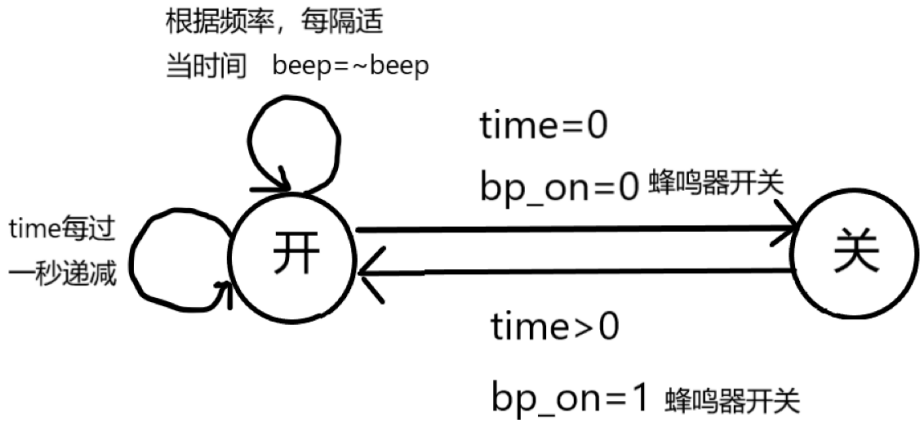
```
Bp_ON()              //打开蜂鸣器
Bp_OFF()             //关闭蜂鸣器
set_Beep(int,char)   //设置蜂鸣器频率和时长
Voice()              //驱动，在500us时调用
```

具体可以通过以下两步完成蜂鸣器的设置：

```
set_Beep(1500,1);
Beep_ON();
```

第一步设置频率和时长，第二步打开蜂鸣器，也可通过 beep_OFF()中断蜂鸣器响

模块实现如下：



接下来主要看一下驱动：

```
void Voice(){// just can be used in Func 500us
  if(bp_ON==0 ) {beep=0;return;}//如果蜂鸣器关闭，则返回
  if(!bp.time) {beep=0;bp_ON=0;return;}
//如果时间到达，则将蜂鸣器置低电平，并关闭蜂鸣器
    if( ++time_count == 2000 ){
      bp.time--;
      time_count=0;
    }//2000个500us是1s，则将时长-1
    if( ++bp_count==bp.voice_judge ){
      bp_count = 0;
      beep =~ beep;
    }//电平反转
}
```

②LED_segment.h

数码管流水灯模块，将驱动扫描和用户传参分离开，可实现

1、分别控制流水灯的开关、数码管的开关。

2、数码管显示的范围

3、数码管和流水灯显示的内容

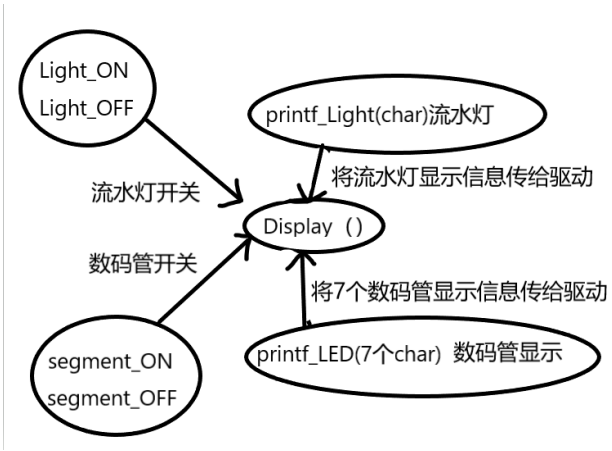
数码管及其流水灯的 API 如下

```
set_pose(char,char)     显示范围
Light_ON                打开流水灯
LIGHT_OFF               关闭流水灯
Segment_ON              打开数码管
Segment_OFF             关闭数码管
print_LED(seven char)   数码管显示
print_Light(one char)   流水灯显示
set_LED_char(pose,char)自定义显示
Display()               驱动，扫描数码管
```

可通过下列例子设置：

```
set_pose(0,7);                      //让0~7位数码管亮
Light_ON();                         //打开流水灯
print_LED(0,0,0,0,0,0,0,0);         //全部显示0
print_Light(WATER);                 //流水灯显示0x01
segment_ON();                       //开启数码管
```

模块实现如下：



主要看一下驱动：

```
void Display(){
  if(led.now_pose <= led.end ) {
    if(SEGMENT_ON){//如果数码管使能，则显示
      P0 = 0;
      sel_led = 0;
      P2 = (P2&0xf0) | (led.now_pose&0x0f);
      P0 = (led.display[led.now_pose]==0x7f)?LED_Char[led.now_pose]:(LED_Data[led.display[led.now_pose]]);
      //该语句时当要显示自定义字符时，该字符为0x7f，然后P0被复制给自定义位选字符。
    }
    ++led.now_pose;
  }
  else if(led.now_pose == led.end+1){
    if( LIGHT_ON ){//如果流水灯使能，则可显示
      P0 = 0;
      sel_led = 1;
      P0 = led.water;
    }
    led.now_pose=led.start;
  }
}
```

第一个 if 语句是扫描数码管，第二 if 语句是扫描流水灯。是否显示由 segment_ON 和 Light_ON 控制。

③ Key.h

按键消抖模块，实现了

1、使能单个按键

2、防丢失处理，即将按键信息放入一个循环队列，防止按键速度过快导致处理跟不上。

3、获取按键的信息，即哪个键被按下。

Key 模块 API；

```
key_init() 初始化按键缓存区
key_enable() 使能某个按键，1、2、3
int key_push() 一旦有按键按下，则返回按下的第几个键
count_key() 驱动，轮询按键的状态，如果按下则将该按键按下的信息存入循环队列中
```

简单两步即可完成按键设置

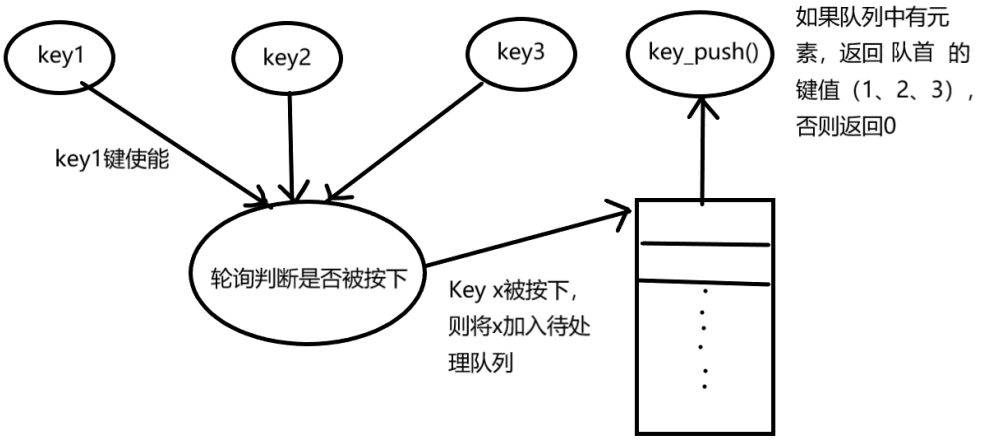
```
key_init();
key_enable(1);
```

设置完毕

接着就可以在函数中调用 `number=key_push();`

当 number 非 0 时则说明键被按下，number=1 说明 key1 被按下，其他同理

模块实现如下：



简单看一下驱动（key1 键为例）：

```
void count_key(){
  if(!key_on[0] && !key_on[1] && !key_on[2]) return;
  //如果没有按键被使能，则直接返回
  if(key_on[1]){
    key1_C1++;
    if(key1==0)
      key1_C2++;
  }//消抖计数
```

```
void judge_push(){
  if((Q.tail+1)%20 == Q.head )
    return;
  //如果队列为空，则直接返回
  if(key1_C1==30){
    if(key1_C2>=20){
      if(k1_status==1)
        Q.queue[Q.tail++] = 1;
        //将按键信息存入队列中
        Q.tail=Q.tail%20;
        //循环队列，所以对尾指针上述处理
        k1_status=0;
      }
    else k1_status=1;
    key1_C1=0;
    key1_C2=0;
  }
```

```
int key_push(){
  if(Q.head!=Q.tail){
    PUSH = Q.queue[Q.head];
    Q.head++;
    Q.head=Q.head%20;
    return PUSH;
  }
  else return 0;
}
```

队列不为空，则返回按键信息

- **按键**

QQ视频20200428211441.mp4

- **实验心得**

本次实验中，我对定时器的使用有了进一步的理解。在这个定时器的测试中，出现的问题主要是 16 位重载时，在变量观察窗口的地方时，并没有观察到定时器被重载我们初始的值，而是继续从 0 开始，而且将定时器换成其他的模式之后，通过仿真仍然无法观察到正确的现象，通过询问老师，才知道了 keil 并不支持对定时器的深层仿真，虽然仿真无法看到正确的现象，但是下载到板子上的现象仍是正确的。

另外，在这个实验中，我尝试使用了状态机，在不同状态下进行代码的编写，这样逻辑比较清晰，而且在进行 Debug 时，能够单个的仿真其中的一个状态，当该状态现象正确后，再继续进行下一个状态的实现。

模块化编程，这是我本次实验收获最大的部分。明白了一个道理，具象很容易，对于一个特定的需求，我们有很多很多种方法达到客户想要的效果。但是如果要把需求抽象化，抽象成一个一个的简单步骤，那是相当困难的。模块化编程大大提高了编程时的逻辑难度，因为要将

复杂的模块抽象成一个个简单的 API，而且不同 API 内相互联系，却又不能相互干扰。在本次编写的蜂鸣器、数码管&&流水灯、按键模块中，数码管&&流水灯模块难度最大，涉及到两类主要硬件，数码管和流水灯，而且数码管和流水灯的亮暗互不影响。接着是按键模块，本来按键信息是希望通过标志位作为参数传递，但是考虑到可靠性，就将按键信息放入一个循环队列，每次从队列中取值，如果队列为空，返回 0 表示没有键被按下。