设计题（五）：STC 单片机 SPI 原理及实现

一、题目：

【设计要求】

1）使用 STC 单片机上的七段数码管，在竖向方向上实现，从上向下“下雨”的效果，填满所有 7 段数码管，然后清空，再重新填充七段数码管。（80 分）

2）在 7 段数码管上实现“贪吃蛇”演示效果（20 分）

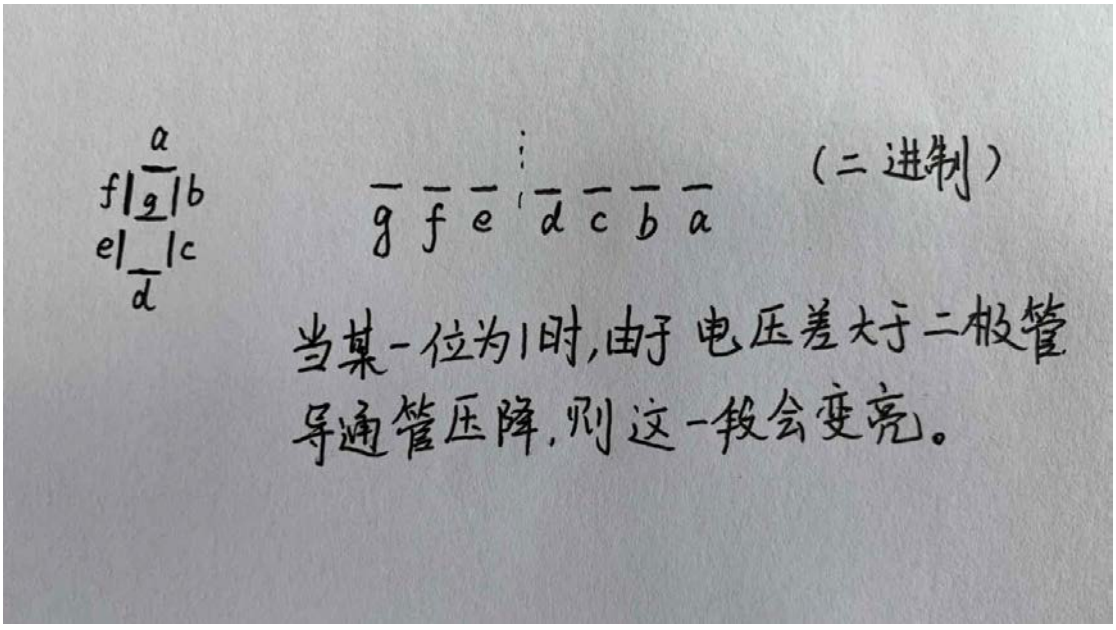
二、设计大体思路

【下雨部分】

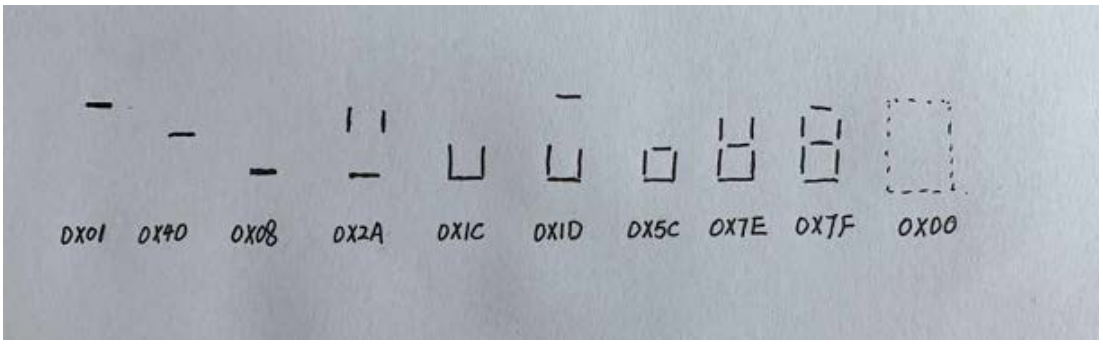
下雨部分的设计实际上就是要搞懂每次从七段数码管上下来的是什么东西？在我设计的下雨试验中是这样的：

```
/*第一种下雨方式*/
unsigned char code t_display[10]={0x01,0x40,0x08,0x2A,
                                  0x1C,0x1D,0x5C,0x7E,
                                  0x7F,0x00};
```

要诀就是要把实际的雨抽象为七段数码管的段码值，用下面这个图来进行理解：



这是我构造的下雨时的几种状态：



一共是 10 种状态，其中有一种状态是 0x00，即什么都不显示，也就是题目中所说的清零状态。

然后调动 seg7scan()函数，这里的 seg7scan()函数需要单独解释一下：

```
void seg7scan(unsigned char index1,unsigned char index2)
{
  SPI_SendByte(~T_COM[index1]);  传送管选码
  SPI_SendByte(t_display[index2]);传送段码
  HC595_RCLK=1;锁存信号
  HC595_RCLK=0;
}
```

每一次调用 seg7scan()函数都会使得其中某一位的七段数码管的某些段点亮，而我们单片机上一共可以使用的数码管共有 8 位，一个 seg7scan()函数是不可能同时控制所有位置的。

那么怎么同时让这八位数码管下雨呢？

```
while(1)
{
  switch(pattern)
  {
    case 0://下雨模式
      if(flag==1)
      {
        flag=0;
        for(i=0;i<8;i++)
         {
           seg7scan(i,m);
         }
      }
      break;
```

那就是利用视觉暂留的原理，让八个数码管在一个循环里，以极快的速度交替显示，实际上是一位显示完再显示下一位，但是在眼睛看起来是同时所有的数码管都被点亮了，也就是同时八位数码管在“下雨”。

这是视频，双击可以打开：

下雨.mp4

【贪吃蛇部分】

“贪吃蛇”部分是这道题的难点所在，在我具体设计的过程中，踩了很多坑，走了不少弯路，也与很多同学交流过，最后也是成功做出来了。

我先简单说一下我设计的贪吃蛇的规则：

1.为了方便演示效果，把撞到墙，也就是四周死亡游戏结束的模式，更改为撞到墙游戏不结束，而是停在原地不动。

（其实撞到墙游戏结束和撞到墙游戏停止的设计思路上是一样的，只是 IF 条件下执行的操作把停止替换为初始化游戏就好了）

2.操作上和大家平时玩的贪吃蛇游戏相同，有左右两个方向的操作按键。

那么，在顶层设计上我是这么想的：

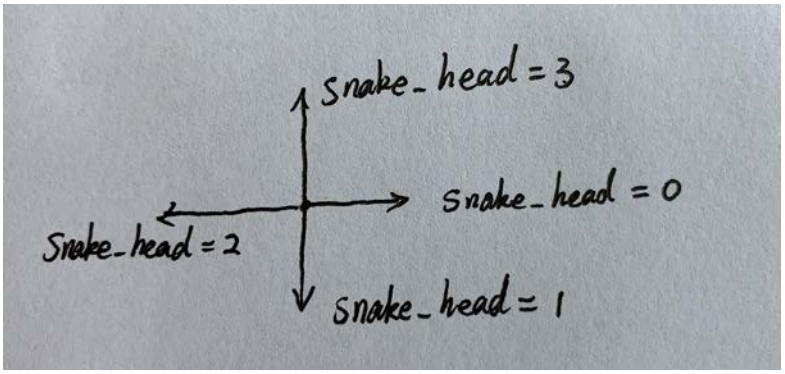
按键	功能
SW17	1.在下雨状态下，按下进入贪吃蛇模式 2.在“贪吃蛇”模式下，按下“贪吃蛇”进行左转
SW18	1.在下雨状态下，按下进入贪吃蛇模式 2.在“贪吃蛇”模式下，按下“贪吃蛇”进行右转
SW19	单片机电源开关，重启时用

那么下面是核心问题了，怎么让“贪吃蛇”按照我们的想法动起来？

关于这个问题，我思考了很久，最后采用了如下方法：

```
unsigned int snake_head; //方向矢量
unsigned int snake_head_tmp;//临时方向矢量
unsigned int way;//中断方向标志
```

采用了两个方向矢量，其中 snake_head 作为当前状态贪吃蛇蛇头的方向，snake_head_tmp 作为在此状态改变之前也就是上一状态贪吃蛇的方向，取值有四种，分别为“上，下，左，右”：



那么下面需要思考的问题是，每次按下 SW17 或者是 SW18 后，snake_head 该怎么变化？

```
void left() interrupt 0//左转部分
{
  if(pattern == 0)//如果处于下雨状态,按下后,完成切换到贪吃蛇模式
  { 进入贪吃蛇模式并进行初始化
    pattern = 1;
    snake_com[0]=0;
    snake_dis[0]=0;
    my_seg7scan(snake_com[0],snake_dis[0]);
    snake_len=1;
    way=0;
    snake_head=0;
    snake_head_tmp = 0;
    food_com=3;
    food_dis=0;
  }
  else//在贪吃蛇模式中,按下左转会进行下面的操作
  {
    way = 1;//中断方向标志
    snake_head_tmp = snake_head;//保存上一状态的方向
    /*改变方向*/
    if(snake_head == 0)
    {
      snake_head = 3;
    }
    else
    {
      snake_head--;
    }
  }
}
```

SW17按下后做的事情

初始方向假设为0，向右，这个可以自定义

中断方向标志赋为1，代表即将进行左转

这个是核心部分，"snake_head--"
由方向示意图可知，方向矢量每次减一都进行了左转

右转操作类似，只是将 snake_head 进行了“加一”操作，在此不再赘述。

下面解决了方向矢量怎么变化的问题，下面就是要解决贪吃蛇怎么“动起来”的问题了。

其实，乍一看觉得情况很多，要把每一种操作所产生的所有的情况（包括蛇的每一节怎么动）都考虑一遍，这显然是不可能的。

但是，如果我们转换一种思维：

从“段码”入手，解决好蛇头的运动状态；

“蛇”身体的位置更新——其实就是后一段经历前一段的位置；

想好了这两个问题，其他问题都将迎刃而解。

也有人会问到，蛇头的运动状态有多少种呢？

一定不能进入这个思维的死循环，要从“段码”入手，先考虑蛇头的位置！

无论蛇怎么走，无论多长，蛇头经历的位置有且仅有七种可能，那就是七段数码管的七段中的任意一段。

我们以蛇头在 a 段，也就是最上面一段进行考虑，分析如下：

在此之前，先看看我们蛇头的表示方法：

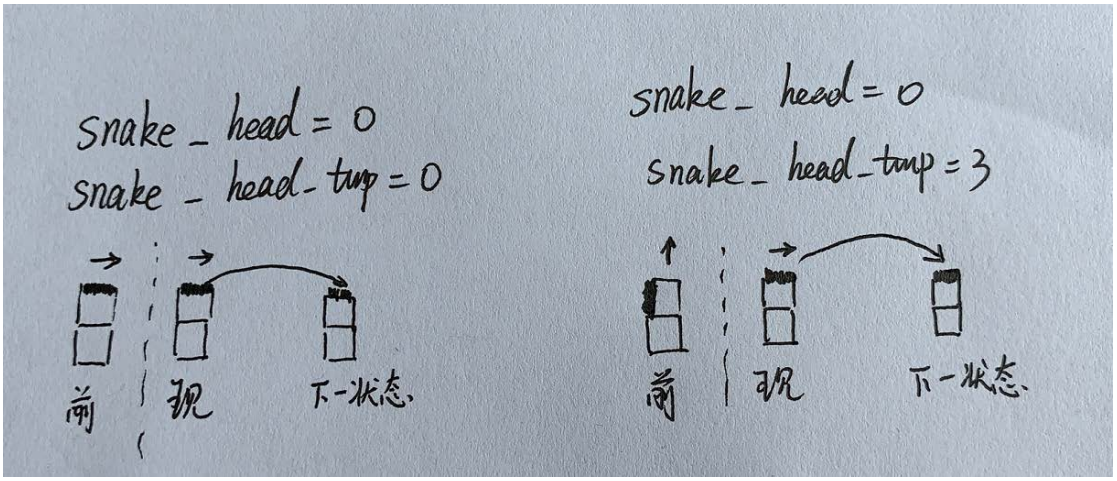
蛇头的“位码”snake_com[0]，表示蛇头目前在哪个数码管。

蛇头的“段码”sanke_dis[0]，表示蛇头目前在数码管的哪一段。

【snake_head＝＝0】

下面我们来看蛇头在数码管 a 段，并且蛇头方向 snake_head＝＝0 的情况：

```
if(snake_head==0 && snake_head_tmp==0)  蛇头的当前方向往右，且前一个方向状态也是往右
{
  if(snake_com[0]<7)  如果没有到最右边的数码管，就往右移动一段
  {
    snake_com[0]++;
  }
}
else if(snake_head==0 && snake_head_tmp==3) 蛇头的当前方向往右，且前一个方向状态是往上
{
  if(snake_com[0]<7)  如果没有到最右边的数码管，就往右移动一段
  {
    snake_com[0]++;
  }
}
```



简单画了个示意图，最左边黑色加粗实线部分代表的是上一状态“蛇头”的位置，中间是当前状态，最右是进行完 IF 中的条件后，蛇头的位置变化。

如果已经到了最右端，即 snake_com[0]==7 时，就什么也不做，原地停止，也不更新蛇身。

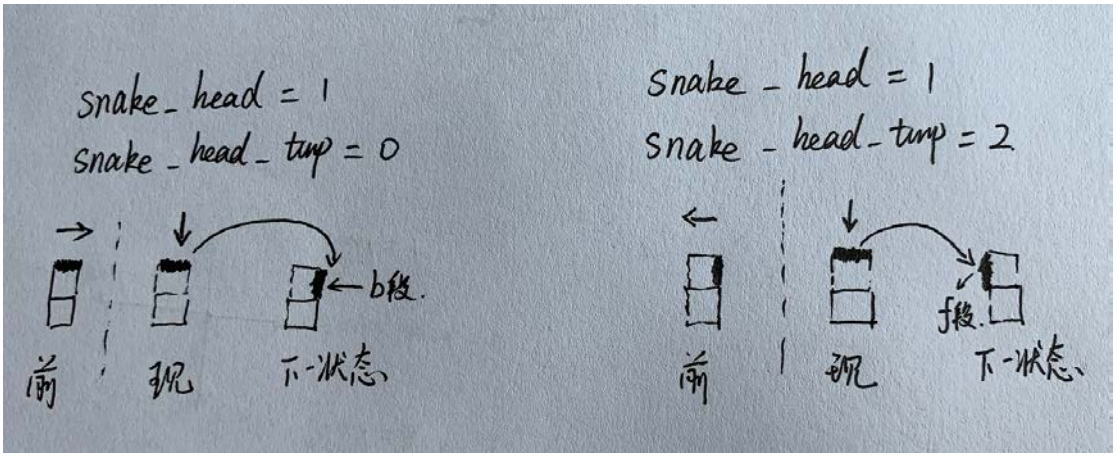
（当然也可以把这个停止改为游戏失败，直接退出游戏，就不用停止了。）

备注：上面两个 IF 条件判断执行的操作是一样的，其实是可以合在一起的，之所以独立出来是为了把过程描述得更清楚，实际编程的过程中建议合在一起。

【snake_head＝＝1】

下面我们来看蛇头在数码管 a 段，并且蛇头方向 snake_head＝＝0 的情况：

```
else if(snake_head==1 && snake_head_tmp == 0) 当前方向向下，且上一状态是向右
{
  snake_dis[0]++;   代表由a段变化为b段
}
else if(snake_head==1 && snake_head_tmp == 2) 当前方向向下，且上一状态是向左
{
  snake_dis[0]=5;   代表由a段变化为f段
}
```



这两种情况就不能合并了，需要单独拿出来考虑。

结果也能看到，不同状态下更新后的蛇头位置是不一样的。

但是，只是“段码”发生了变化，还是在同一根管子里进行显示，所以不需要改变 snake_com[0]的值。

【snake_head＝＝2】

```
else if(snake_head==2)
{
  if(snake_com[0]>0)
  {
    snake_com[0]--;
  }
}
```

如果方向往左
且没有到最左
蛇头就往左移动一位
snake_com[0]--

这样的写法与之前 snake_head 为 0 的时候情况类似，这里采取的是合并的写法，代码更加简结。

这里“位码”发生了变化，是在不同的管子里进行显示，但是“段码”不变，所以不需要改变 snake_dis[0]的值。

【snake_head＝＝3】

这种情况需要考虑吗？

a 段是在管子的最上方，是不可能进行向上移动的操作的，所以这种情况不需要考虑。

当然也可以设计为游戏结束的标志，或者直接什么都不做。

其他的段，与此相类似，也是这样的思路去考虑就好，想清楚了其中一段，其他的段都比较好写。

那么第一个问题解决了，剩下的就是蛇身位置更新的问题了。

这个问题其实更简单，但是需要注意的是，一定要在蛇头位置更新之前，也就是每一个 case 下面最开始的地方，进行位置更新。

代码如下：

```
for(snake_node=snake_len-1;snake_node >= 1;snake_node--)
{
  snake_com[snake_node]=snake_com[snake_node-1];
  snake_dis[snake_node]=snake_dis[snake_node-1];
}
```

从最后一段开始，一直到蛇头后面紧接着的一节
将蛇身前一节的段码赋给后一段
将蛇身前一段的位码赋给后一段

解释一下，snake_len 是蛇的长度，初始值为 1 ，每次吃掉食物都会加 1 。

其实就是“位码”和“段码”的更新，核心代码就是上面的 for 循环里面的两行代码。

至此，关于蛇移动的问题都已经解决了，目前还需要解决的就是食物了。

关于食物有两种方法产生：

一种是自己预先设定食物出现的位置，每次蛇头到达食物的位置后，蛇的长度加 1 ，并且进行更新。

另一种是采用 rand()函数，随机产生食物。

两种方法操作上相类似，这里采用第一种。

```
if(pattern == 1 && snake_len == 1 && snake_com[0]==3 && snake_dis[0]==0)
{
  snake_len++;//蛇的长度加1
  food_com = 5;
  food_dis = 6;
}
```

重新更新食物出现的位置

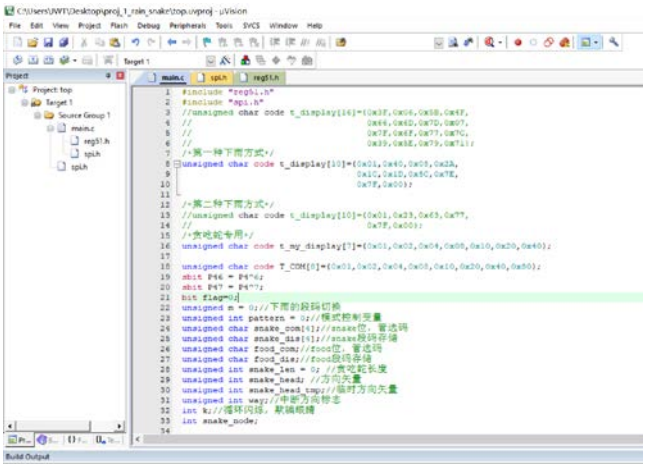
然后还有一个小问题需要注意的是，蛇最长为多长？

其实这个问题就是由设计者自己设计的了，可以是最长为 3 段， 4 段， 8 段等等，改变一下声明时用来表示蛇的数组的大小就好。

本次试验为了方便演示出吃完食物蛇身要变长，并且食物位置要更新等场景，采取了折中的办法：

蛇身初始时只有蛇头，吃完后会变长，变长到四段后，就不给食物了，数码管上不显示食物，但是程序不退出，用户依然可以控制“贪吃蛇”完成各种移动的动作。

三、实验代码



实验代码就不放在实验报告这个位置了，放在文档末的附录中。

四、实验演示视频

下雨.mp4

（双击可以播放）

下雨和贪吃蛇综合演示.mp4

（双击可以播放）

五、总结与反思

晃眼间，才发觉这是自己本学期单片机课程中的最后一个实验了，在深夜调试出最后的结果后，自己长舒了一口气，这个自己费时良久做完的“贪吃蛇”终于大功告成了！

但我也深知，它并不完美，有很多地方还可以改进，一定可以做得更好。

这是这几次作业中难度最大的一个实验，自己踩了很多的坑。比如最初我的“贪吃蛇”无法变长，原因是赋值方向弄反了，应该从蛇尾到蛇头；还有每次吃完食物会显示乱码，之后恢复正常。这个问题困扰了我很久，整个晚上都在想这个问题，但是我也坚信自己的贪吃蛇的显示部分没有问题。我换了一个单片机后，发现乱码还是会出现，但是乱码出现的样子不同了！至此，基本可以断定我遇到了一个非常难解决的问题——时序问题。但是既然知道是时序问题了，那么就解决时序的错乱就好了。经过仔细的检查，我发现，我在更新食物，使得蛇的长度变长的过程是在定时器 0 这个高速的中断中进行的，而在位置的显示却是在主函数里面的 while 下进行的，这两个由于速度上的不匹配，是很容易产生时序上的问题的。于是我把更新食物以及长度变长的代码放在了主函数中，也就是这两个模块在同样的地方运行。果不其然，问题圆满解决了。其实上课时何老师强调过，不要在中断的过程里写复杂的操作。这也是自己在

今后程序设计中需要遵循的一个准则。

做完这个实验，单片机课程也接近尾声了，但是自己在程序设计以及嵌入式开发的道路上才刚刚开始。多实践才是正途，“绝知此事要躬行”，看得再多也不如自己亲自动手写，亲自动手实践学得更快。