

**NANCHANG UNIVERSITY**

**单片机原理与接口技术**



**题 目:** 拉幕式类拨号器密码音频播放器

**专 业：** 软件工程

**班 级：** 软件工程1910班

**完成人数：** 1

**成 员：** 王俊洪

**学 号：** 8002119319

**任课教师：** 杨承根、黄永平

**完成时间：** 2021年6月20日

目录

[一、项目简介 1](#_Toc5215)

[1.1 项目功能 1](#_Toc3959)

[1.2 项目目的 1](#_Toc27204)

[1.3 项目基础 1](#_Toc25169)

[1.4 项目扩展 2](#_Toc23227)

[二、模块的开发 3](#_Toc8705)

[2.1 数码管模块的设计 3](#_Toc1043)

[2.1.1数码管模块设计思路 3](#_Toc6622)

[2.1.2数码管模块Proteus电路图 4](#_Toc14698)

[2.1.3 数码管模块Keil代码设计 4](#_Toc14634)

[2.2 音乐模块的设计 5](#_Toc23342)

[2.2.1 音乐模块设计思路 5](#_Toc7178)

[2.2.2 音乐模块Proteus电路图 5](#_Toc1200)

[2.2.3 音乐模块Keil代码设计 6](#_Toc10559)

[2.3 URAT模块的设计 7](#_Toc16255)

[2.3.1 UART模块设计思路 7](#_Toc21161)

[2.3.2 UART模块Proteus电路图 8](#_Toc12894)

[2.3.3 UART模块Keil代码设计 8](#_Toc14040)

[2.4 键盘模块 10](#_Toc12554)

[2.4.1 键盘模块设计思路 10](#_Toc5181)

[2.4.2 键盘模块Proteus电路图 10](#_Toc31547)

[2.4.1 键盘模块Keil代码设计 10](#_Toc17072)

[2.5 combine模块 14](#_Toc11380)

[三、增量模块组合 15](#_Toc20077)

[四、开发过程的挫折 17](#_Toc11176)

[五、心得体会 18](#_Toc26548)

# 一、项目简介

## 1.1 项目功能

本项目需要实现的功能主要有两个，其一就是**按键的拉幕式数码管显示**，就是按键对应的键值或从又到左不断的显示，形成所谓的拉幕式效果；其二就是按下特定的口令再按下确认时，会**播放自己预先设置的音乐节奏**，本项目为了简单起见，就不设计修改口令的功能和音乐多样化的功能，播放的音乐只有一首，那就是在洒水车上经常听到的《兰花草》。

此外，还设计了一个清空拉幕式数码管的按钮，用于**一键清空**数码管，使其显示为默认的一杠。

最后，本项目还实现了按键的双音阶发声，这也是后面播放音乐的前提，发声从1、2、3、4、5、6、7到1^，2^，3^，4^，5^，6^，7^依次类推的十六个音。每按下一个按键都会发出不一样的声音，松开按键后声音消失，模拟手机拨号器按键的声音。对于长按按键，有两种常见的处理方法，第一种就是长按只会触发一次，第二种就是长按一直触发，类似于外部中断的下降沿触发和低电平触发。

本项目采用的两者的结合，即对于长按，本项目会只显示一个数字在数码显示管上，也即是对于拉幕式数码显示管，长按采取一次触发，并且是松开按键时将按下的键值显示到数码管；但是对于音阶发声，长按会一直发生，知道松开按键发声结束。

## 1.2 项目目的

实现本项目的目的主要在于总结这学期单片机课程的学习，在理解基础知识的基础上进行自己的一些开发设计，本次开发大多数都是自己以前学过的知识，但是之前缺少对模块化的认识，导致好多项目都只是一个文件，这样原来项目的代码不能直接进行复用，还得稍作修改才可以使用它。这从另一方面增大了开发的难度。

因此，本项目的设计一个很重要的目的就是体会模块化的好处和学会将自己的工程进行**模块化**，以实现抽象出一些具体的方法，在其他工程中引入后可以直接使用。

## 1.3 项目基础

本项目的基础就是平时的实验积累，以及老师发的资料，主要使用的知识包括：

* 定时器
  + 定时器0用于控制扬声器发出不同频率的声音，即发出双音阶提示音。
  + 定时器1用于UART串口的通信，和虚拟中断结合完成信息的显示，相当于日志信息。
* 行列式键盘

行列式键盘排成4行3列，通过连接在P1口进行使用，通过判断按钮按下的行数和列数从而来获取按键按下的值执行不同的操作。

* 串口中断

我们都知道，C51单片机具有5个中断源，分别为两个定时器中断、两个外部中断和一个串口中断。在这里，我们使用串口中断和Virtual Terminal进行通信，以显示printf输出的信息，提示用户用户进行的操作。

* 8个并排的7-SEG

通过并排的8个7段管来显示用户按下的键值，再通过自己的移位算法实现拉幕式显示数字的效果

以上的知识，都是平时实验做过的内容，由于之前我们老师不强调模块化和封装，因此其代码复用性降低，我们不能直接使用，需要更改其中具体的代码或者接口方可使用。

## 1.4 扩展方向

本项目还可以接着扩展更多的功能，其中我认为主要有两个可扩展的方向。

其一就是，按下口令播放音乐时，音乐的播放和数码管的显示不能同时进行，这是因为音乐播放时是一个循环，其未结束时，不会调用数码管的显示方法，因此播放音乐时数码管会熄灭。解决这个问题，这里只提供一种思路，由于精力有限，因此不去实现这个功能了。

其思路如下：使用两块单片机，通过串口或者I2C进行通信，在播放音乐时，将数码管显示的参数发送给另一个C51单片机进行显示，这样播放音乐和数码显示就可以同时进行。

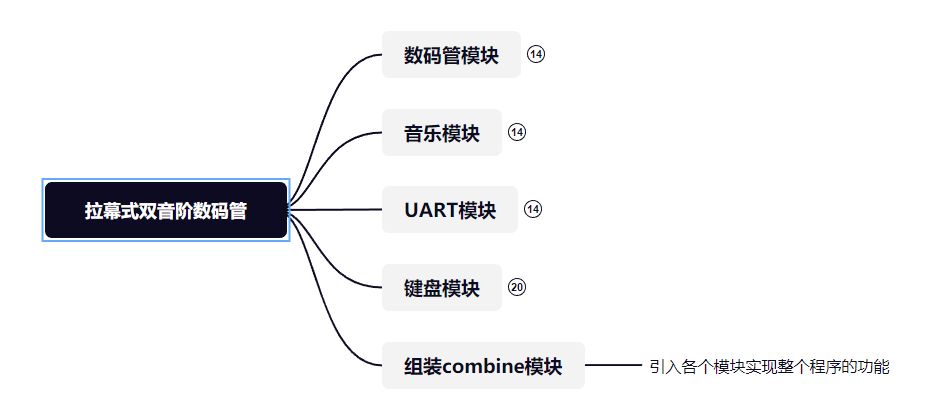
第二个问题就是还可以设计修改口令的功能，否则的话，这个口令就只能在代码中修改，这是很不符合常理的。由于本项目主要是学习单片机基础功能，因此这个修改口令的功能这里也不做实现，只提供一种思路。

其思路如下：多设置一个按键，用于修改口令，按下按键时调用修改口令的方法，其中包括原口令的验证以及新口令的存储。

# 二、模块的开发

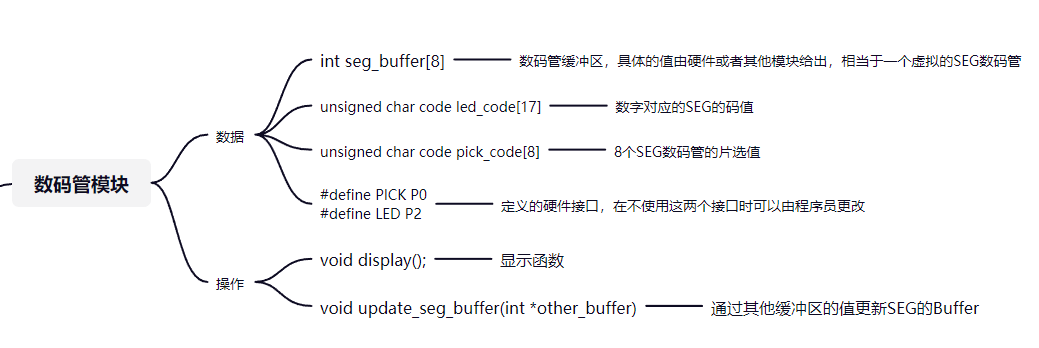
**设计的原则就是模块化，将不同的功能抽象称为具体的模块，增加代码的复用性和移植性。**

分的模块如下：总共五个模块，每个模块可以单独拿出来使用。

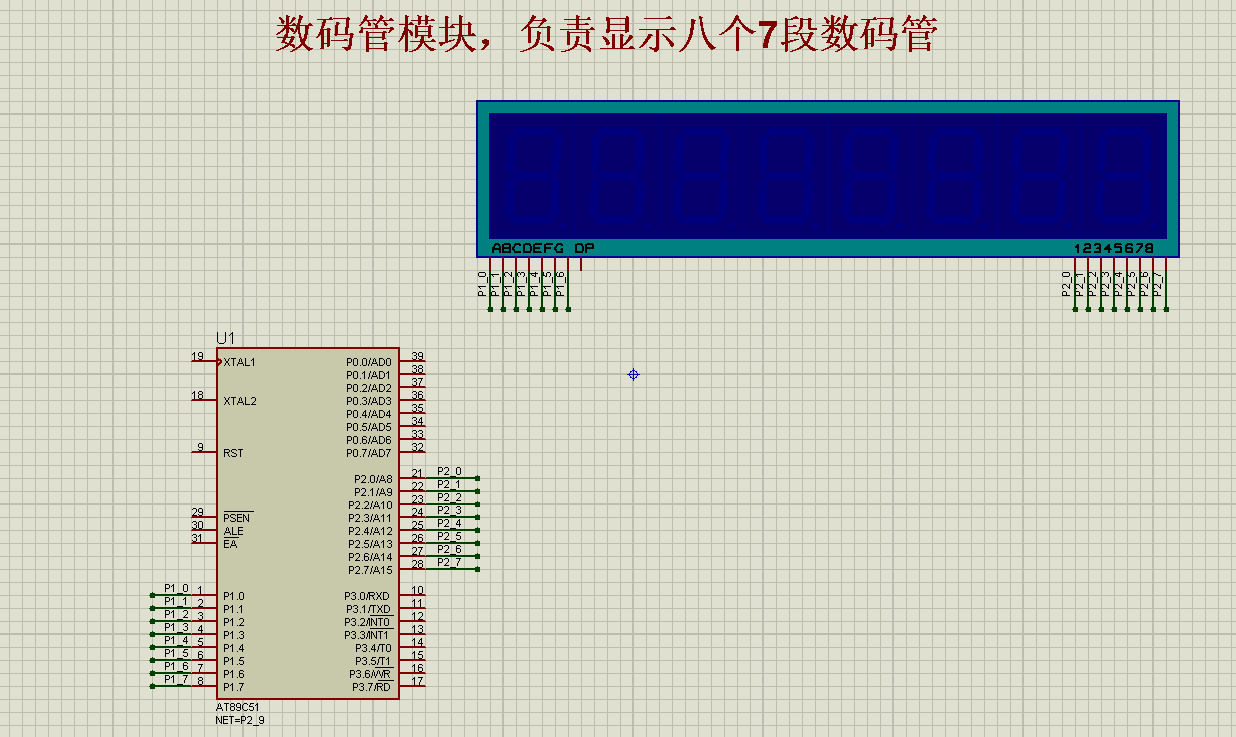


## 2.1 数码管模块的设计

### 2.1.1数码管模块设计思路



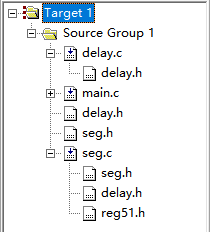
### 2.1.2数码管模块Proteus电路图



在该模块中，使用P1口控制7-SEG的7个发光二极管，使用P2口控制其八个七段管的片选，当然这是一个完整的模块，其硬件的约束只需要定义在代码中一部分即可。

### 2.1.3 数码管模块Keil代码设计

该模块的项目结构如下：



该模块引用了延时接口，定义了自己的模块，seg.h中提供模块的接口，seg.c中实现具体的函数。

在该模块中，为了避免硬件的约束，也就是换一个硬件接口仍可以使用，因此在头文件中使用宏定义：

#define PICK P0

#define LED P2

这样，**在开发过程中需要更换接口，只需要在此处更改对应的硬件接口值即可，而不用更改具体函数中的每一个接口名称。**

同时，该模块提供了一个SEG的缓冲区，该缓冲区位于其他模块或者硬件之间，是数码管模块与其他模块打交道的过渡层。该缓冲区名为：*int seg\_buffer[8];*。

另外，该模块的对应操作有：

void display();

void update\_seg\_buffer(int \*key\_buffer);

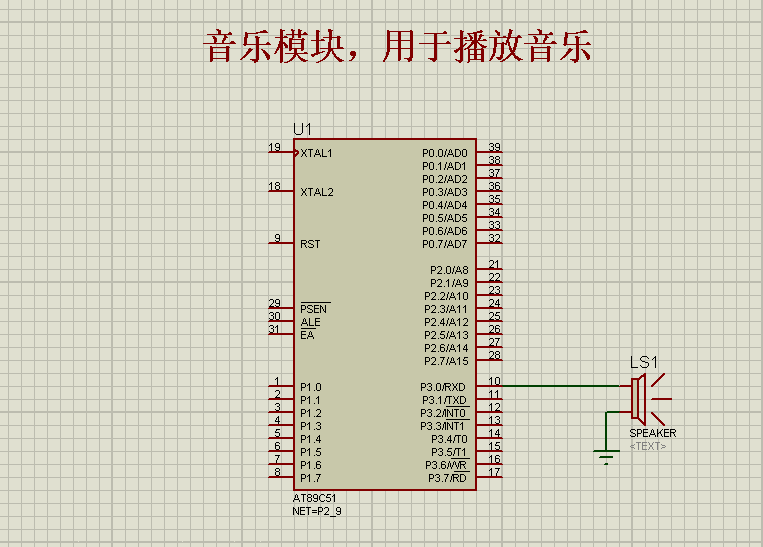
其中，*display()*函数就是将缓冲区的内容显示出来，其中至于缓冲区内容从何而来，与这个函数无关。而与*update\_seg\_buffer(int \*key\_buffer)*有密切关系，这个函数就是将其他缓冲区的内容更新到SEG的缓冲区，然后调用*display()*函数才会正确显示。

## 2.2 音乐模块的设计

### 2.2.1 音乐模块设计思路



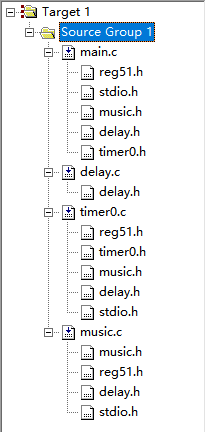
### 2.2.2 音乐模块Proteus电路图



可以看到，将一个扬声器接在P3.0口上就完成了音乐模块的电路图，其只需要一个硬件接口。这个接口在模块中会抽象出来，方便程序员根据自己的需要更改接口，而不是每次都使用P3.0接口。

### 2.2.3 音乐模块Keil代码设计

该项目的结构图如下：



在音乐模块中，包含了许多重要的信息：

// 音阶频率表 高八位

unsigned char code FREQH[28]={

0xF2,0xF3,0xF5,0xF5,0xF6,0xF7,0xF8, //C3

0xF9,0xF9,0xFA,0xFA,0xFB,0xFB,0xFC, //中央C4

0xFC,0xFC,0xFD,0xFD,0xFD,0xFD,0xFE, //C5

0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFF, //C6

};

// 音阶频率表 低八位

unsigned char code FREQL[28]={

0x42,0xC1,0x17,0xB6,0xD0,0xD1,0xB6, //C3 --> -1

0x21,0xE1,0x8C,0xD8,0x68,0xE9,0x5B, //中央C4 --> +6

0x8F,0xEE,0x44,0x6B,0xB4,0xF4,0x2D, //C5 --> +13

0x47,0x77,0xA2,0xB6,0xDA,0xFA,0x16, //C6 --> +20

};

unsigned char code MUSIC[16][16]={

{8,12,2,16,2,16,2,16,2},

{4,16,6,15,2},

{8,14,3,15,1,14,2,13,2},

{2,12,8},

{8,19,2,19,2,19,2,19,2},

{4,19,6,18,2},

{8,16,2,18,2,18,2,17,2},

{2,16,8},

{8,16,2,19,2,19,2,18,2},

{4,16,6,15,2},

{8,14,3,15,1,14,2,13,2},

{4,12,4,9,4},

{8,9,2,14,2,14,2,13,2},

{4,12,6,16,2},

{8,15,3,14,1,13,2,11,2},

{2,12,8}

};

这是播放音乐以及按键发声，必备的信息，也就是音调的频率对应的定时器的初值，以及乐谱信息。

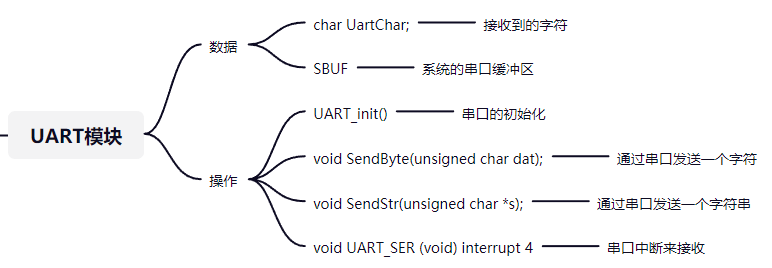
在该模块中还定义了一个变量名为Speaker，该变量是扬声器的一个抽象，具体的值由程序员根据电路图指定，其定义在MUSIC.h头文件中

sbit Speaker=P3^0;

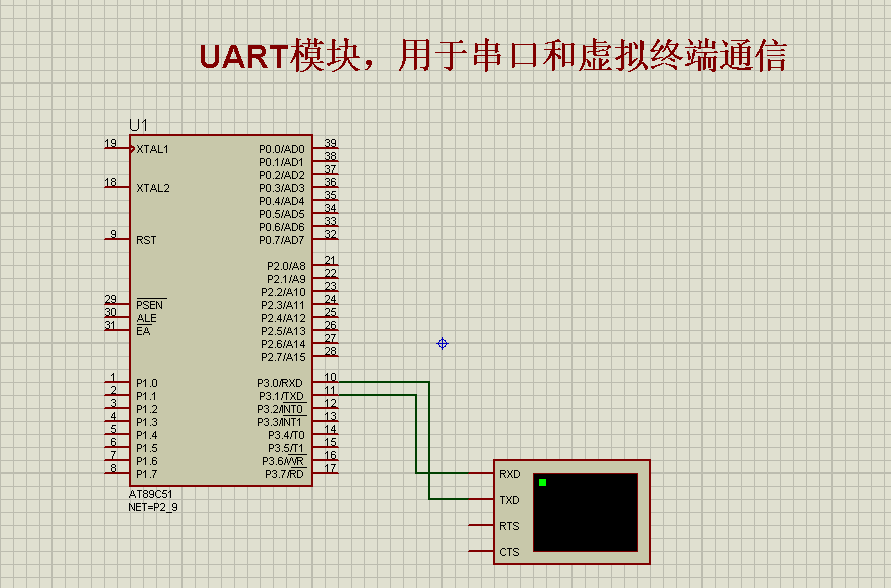
这个值抽象的意义在于当更改了硬件电路的时候，只需要更改这一个值即可，而不用更改函数代码中的对应的值。

## 2.3 URAT模块的设计

### 2.3.1 UART模块设计思路



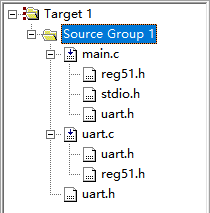
### 2.3.2 UART模块Proteus电路图



该图使用串口主要是为了使用虚拟终端，其和串口进行通信即可，虚拟终端的发送和接收引脚与单片机的接收和发送引脚对应相接即可。

### 2.3.3 UART模块Keil代码设计

UART模块的项目结构如下：



UART主要是用于串口通信的模块，其模块中的数据主要是SBUF缓冲区，51芯片内部的，以及用户自定义的*char UartChar;*用于接收一个收到的字符。

其主要操作有：

* 初始化UART，使其能够发送和接收串口数据。

void UART\_init()

{

// 下面代码设置定时器 1

TMOD = 0x20; // 0010 0000 定时器 1工作于方式 2（8位自动重装方式）

TH1 = 0xFB; // 波特率：6250 /12MHZ

TL1 = 0xFB; // 波特率：6250 /12MHZ

TR1 = 1;

// 下面代码设置定串口

// AUXR = 0x00; // 很关键，使用定时器 1作为波特率发生器，S1ST2=0

SCON = 0x50; // 0101 0000 SM0.SM1=01(最普遍的 8位通信）,REN=1（允许接受）

TI=1; // 很关键，使用 printf函数时必须有此命令

}

* 发送一个字符

void SendByte(unsigned char dat)

{

SBUF = dat;

while(!TI);

TI = 0;

}

* 发送一个字符串

void SendStr(unsigned char \*s)

{

//检测是否字符串末尾, '\0'表示字符串结束标志，

while(\*s != '\0')

{

SendByte(\*s);

s++;

}

}

* 接收数据

void UART\_SER (void) interrupt 4 //串行中断服务程序

{

if(RI) //判断是接收中断产生

{

RI=0; //标志位清零

UartChar = SBUF; //读入缓冲区的值

}

if(TI) //如果是发送标志位，清零

{

TI=0; //发送为查询方式

}

}

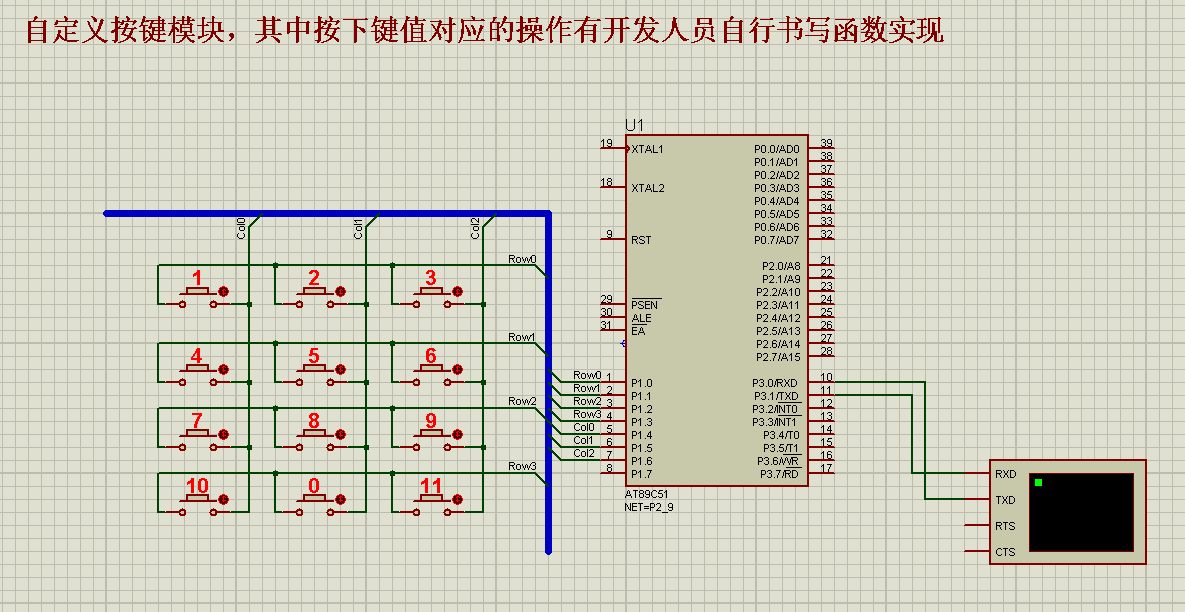
由于UART的使用其硬件接口是固定的，因此这里没有抽象与硬件相关的变量。

## 2.4 键盘模块

### 2.4.1 键盘模块设计思路



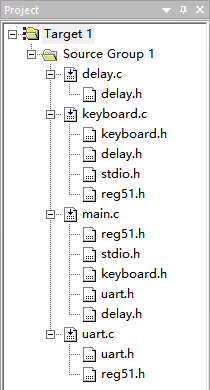
### 2.4.2 键盘模块Proteus电路图



该电路的设计就是在P1口接一个四行三列的按键组，其行线接低电位，列线接高电位，并且和上一个写好的模块组合，给出按键模块的测试。

### 2.4.1 键盘模块Keil代码设计

键盘模块的项目结构如下：



键盘模块主要是确定按键的键值，并且给定一个按键缓冲区，至于**按下每个键的作用是什么则交给程序员根据需要去实现**，因此必须具有这方面的接口。

* 按键模块中的数据有：

int keyValue=0; //存放按键的键值

int isClick=0; //存放按键的按下状态 按下为1 释放为0

int password[8]={8,2,2,0,1,0,0,2}; //密码缓冲区

int key\_buffer[8]={16,16,16,16,16,16,16,16}; //键盘缓冲区

* 键盘模块的操作中最重要的就是获取键值：

void keyDown(){

//printf("KeyDown-Start: isClick = %d\n",isClick);

keyValue=-1;

isClick=0;

//先检测行

INPUT = 0x0f;

if(INPUT!=0x0f){

//有按键按下

delay(1); //消抖

if(INPUT!=0x0f){

isClick = 1;

if(INPUT==0x0e){

//第一行

//测试列

INPUT=0xf0;

if(INPUT!=0xf0){

switch(P1){

case 0xe0: keyValue=1;break;

case 0xd0: keyValue=2;break;

case 0xb0: keyValue=3;break;

}

}

}else if(INPUT==0x0d){

//第二行

//测试列

INPUT=0xf0;

if(INPUT!=0xf0){

switch(INPUT){

case 0xe0: keyValue=4;break;

case 0xd0: keyValue=5;break;

case 0xb0: keyValue=6;break;

}

}

}else if(INPUT==0x0b){

//第三行

//测试列

INPUT=0xf0;

if(INPUT!=0xf0){

switch(INPUT){

case 0xe0: keyValue=7;break;

case 0xd0: keyValue=8;break;

case 0xb0: keyValue=9;break;

}

}

}else if(INPUT==0x07){

//第四行

//测试列

INPUT=0xf0;

if(INPUT!=0xf0){

switch(INPUT){

case 0xe0: keyValue=10;break;

case 0xd0: keyValue=0;break;

case 0xb0: keyValue=11;break;

}

}

}

}

}

}

* 其次还应该提供程序员对于键值处理的接口：

//函数指针，由用户实现这个具体的功能，多态设计

void resolve(void (\*KeyEvent)(int key)){

if(isClick==1){

//按键按下，执行用户函数

KeyEvent(keyValue);

}

}

**这个函数也十分重要，这是由程序员在引入封装好的模块后执行根据自己的需求编写用户函数的，其中keyEvent就是一个函数执行，其具体的实现得交给程序员。体现的是一种多态的设计思想，也就是用户覆写实现同种代码执行不同的功能。**

* 接下来就是操作缓冲区的函数

void update\_key\_buffer(){

int i;

for(i=7;i>=1;i--){

key\_buffer[i]=key\_buffer[i-1];

}

key\_buffer[0] = keyValue;

}

void clear\_key\_buffer(){

int i;

for(i=0;i<8;i++)

key\_buffer[i]=16;

}

这两个操作分别为更新缓冲区和清空缓冲区。

## 2.5 combine模块

这个模块主要是组装前面的模块，这里不赘述，但是这个模块中实现了程序员自定义函数：

void myKeyEvent(int key){

//执行每个按键都做的事情,即播放按键声音

TR0=1;

High = FREQH[keyValue+6];

Low = FREQL[keyValue+6];

while(INPUT!=0xf0)//???????????

{

display(); //????SEG?????????

}

TR0=0;

if(keyValue==10){

printf("Clear...\n");

//清空缓存

clear\_key\_buffer();

update\_seg\_buffer(key\_buffer);

}

if(keyValue==11){

printf("Checking...\n");

if(submit()){

clear\_key\_buffer(); //清空缓存

update\_seg\_buffer(key\_buffer);

PICK=0x00;

playSong(); //密码输出正确后播放音乐

}else{

clear\_key\_buffer(); //清空缓存

update\_seg\_buffer(key\_buffer);

}

}

if(keyValue!=10 && keyValue!=11){

printf("You Press is: %d\n",keyValue);

update\_key\_buffer(); //将刚按下的键存储于键盘缓存

update\_seg\_buffer(key\_buffer); //用键盘缓存去更新SEG缓存

}

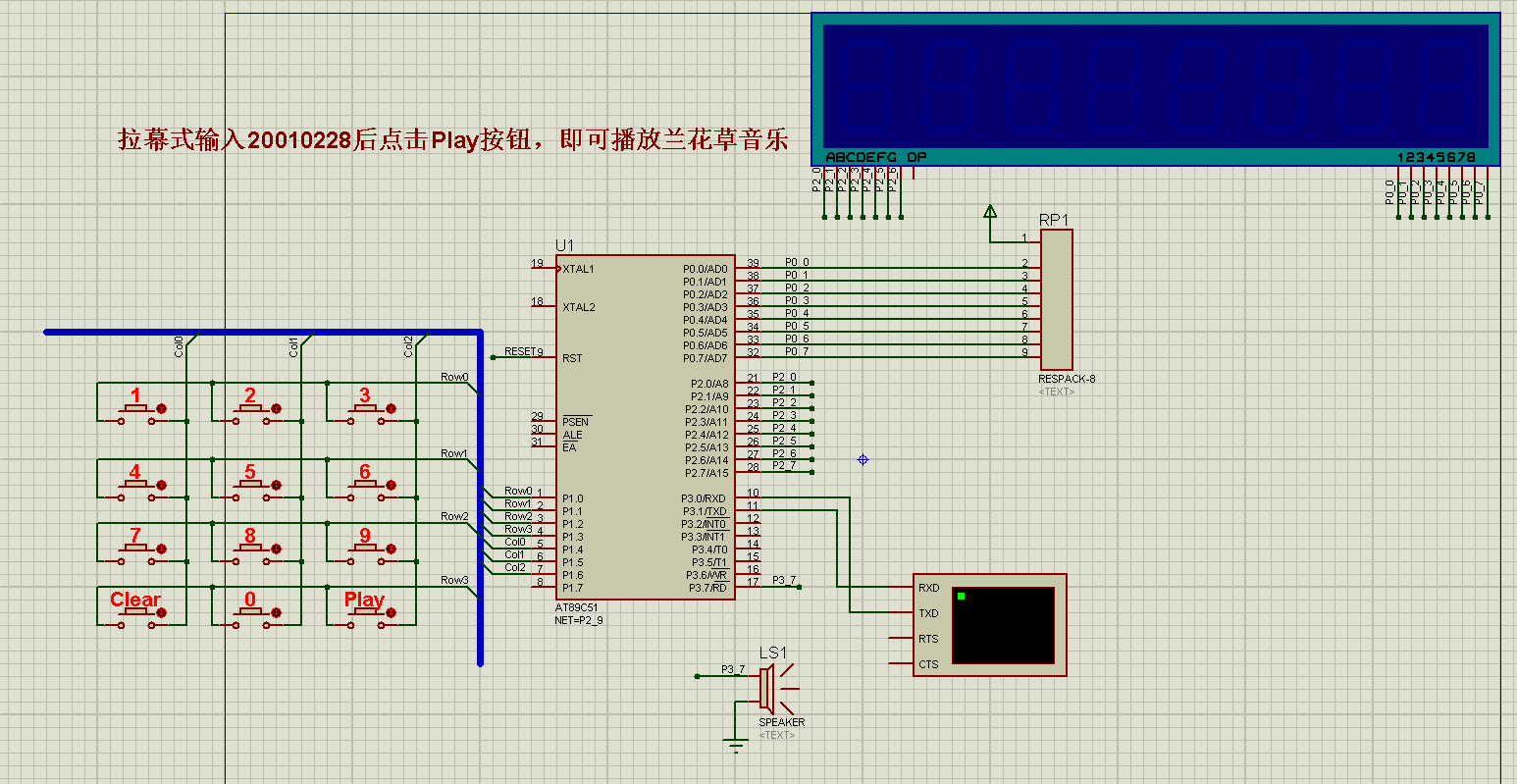
}

具体按键实现的内容都放到这个用户自定义函数中实现，用户必须覆写这个函数，否则就会报错，就类似于C++的虚函数必须被覆写才可以编译通过类似。

# 三、增量模块组合

在上述独立模块开发完成的基础上，只需要将所有的模块组合起来构成自己的项目即可，其中由于已经模块化，其中只需要更改与硬件有关的部分即可。

整个项目组合后的Proteus图如下：



可以清晰的看到，其就是上述各个模块之间的组合，就形成了这个项目，这就是所谓的**增量开发**，**就是在模块化的基础上，在自己的项目中引入已经模块化的模块就可以形成新的项目内容，而不用做大量修改**。

因为上面已经详细介绍了各个模块，因此下面简述组装模块的硬件相关部分的修改。

数码管模块的修改，在模块化时为了避免接上拉电阻，因此使用的是P2口和P1口。在组合时，因为键盘模块比较复杂，因此还是采用P1口，而更改数码管的硬件接口，这样做只需更改头文件中部分代码即可：

#ifndef \_SEG\_H

#define \_SEG\_H

#define PICK P2

#define LED P0

extern unsigned char code led\_code[17];

extern int seg\_buffer[8];

extern unsigned char code pick\_code[];

void display();

void update\_seg\_buffer(int \*key\_buffer);

#endif

可以看到，模块化后就只需要更新SEG.h头文件里的橙色部分代码，而不用大量更改其他的代码。

第二个就是扬声器的接口发生了改变，因此也只需要更改如下MUSIC.h中部分代码即可：

#ifndef \_MUSIC\_H

#define \_MUSIC\_H

#include <reg51.h>

extern unsigned char High,Low;

sbit Speaker = P3^7;

// 音阶频率表 高八位

extern unsigned char code FREQH[28];

// 音阶频率表 低八位

extern unsigned char code FREQL[28];

extern unsigned char code MUSIC[16][16];

void playSong();

#endif

这就是模块化的好处，就是需要什么模块就直接从原来的已经模块化好了的项目中取出来经过微量修改即可使用，而不是自己再写一份类似的代码。**模块化的核心就是分离出硬件层和软件层（硬件无关层），是软件开发中常使用的技巧和技术。**

# 四、开发过程的挫折

每一个程序员在开发过程中都会遇到意想不到的问题，我们俗称这类问题为BUG。下面我将简单介绍我的开发过程中遇到的困难。

* 模块化不清晰

模块化的抽象层次不够，无法将其作为一个单独的模块使用，还存在较强的硬件依赖，无法简单的复制到其他项目中使用，期间要重新书写大量代码，只能在固定接口上做事情。解决这个问题，就应该分析好模块间的关系，和模块的功能，做到数据与操作，软件与硬件有明显的界限，而不是全部混在一起，在软件中使用硬件接口，在硬件接口的初始化中出现软件内容的东西。

* 虚拟中断的使用

虚拟中断的使用有一个很大的易错点，那就是波特率的问题，要是虚拟终端不存在乱码，就应该将其波特率和软件中的波特率对应上，因此这是一个十分隐蔽的错误，不易发现。

由于，C51的内部时钟频率默认是12MHz，因此我将修改硬件初始化部分的代码，使其波特率对应于12MHz的单片机内部时钟频率。其主要是UART的初始化阶段做修改，修改如下：

void UART\_init()

{

// 下面代码设置定时器 1

TMOD = 0x20; // 0010 0000 定时器 1工作于方式 2（8位自动重装方式）

TH1 = 0xFB; // 波特率：6250 /12MHZ

TL1 = 0xFB; // 波特率：6250 /12MHZ

TR1 = 1;

// 下面代码设置定串口

// AUXR = 0x00; // 很关键，使用定时器 1作为波特率发生器，S1ST2=0

SCON = 0x50; // 0101 0000 SM0.SM1=01(最普遍的 8位通信）,REN=1（允许接受）

TI=1; // 很关键，使用 printf函数时必须有此命令

}

可以看到，我将时钟频率选择为12MHz，波特率为6250bit/s，此时在使用虚拟终端的时候即使忘记设置波特率也没关系，因为单片机默认就是12MHz，而虚拟终端的波特率默认也是6250bit/s，如下图：



# 五、心得体会

本次大作业，基本完成了任务，实现了模块化，层次化的方法设计工程。明白了模块化的好处，首先，模块化是软件工程必备的素养，和面向对象的抽象、封装分不开，模块化的好处就是可以在工程中直接引入已经模块化的项目文件，包括头文件提供的接口和源文件的具体实现，而不须做大量修改即可正常使用，几乎独立于硬件，即使依赖硬件也只是小部分，我们应将其抽象为便于程序员修改的部分。

在本次开发过程中，最初采用的是以往的先画图，然后再图上开发，其中出现的问题就是：不仅开发进度慢，而且繁琐，不利于修改和后期调试，更不说移植性问题和兼容性问题。

前期非模块化的开发存在的问题诸多，下面简单介绍几个：

* 开发过程中遇到问题，不好定位

不能确定具体是哪出了问题，因为所有的代码过于集中，不能确定前一部分代码正确的情况下才书写的后面部分代码，因此可能是之前遗留的未被发现的问题，也可能是新书写的代码带来的问题。

* 不方便调试和debug

即使找到的问题出在哪，也不好修改，因为无法确定修改过后会不会引起其他的问题，如果修改过后出现了新的问题那么就会导致这个问题的无限循环，所谓代码一小时，Debug一整天，就是因为采用了不恰当的开发方式。

于是后面的开发中，使用了模块化，增量开发的方法，在自己抽象的模块上一步又一步书写自己的工程，最终形成完整的项目。

由于后期采用模块化、增量开发，也体会到了其中的好处：

* 便于找到错误的地方

如果程序发生了错误，由于采用的是模块化的思路，其模块肯定在使用之前就经过测试了的，因此通过测试的模块一般不会出现问题，那么问题就出在正在书写的模块代码中，只需要着力测试这部分代码即可。

* 代码复用性提高

在需要使用模块的工程中只需要复制对应模块到工程即可，这从一定程度上提高了代码的可移植性和兼容性。

以上就是本次大作业的一些收获，能够学习到单片机这类与硬件密切相关的课程，我觉得十分开心，在今后的学习中，可能不会使用单片机的知识，可能不会和单片机打交道，但是其中学会的思想和方法却会贯穿整个软件工程生涯，那就是：模块化和增量开发。