基于 MSP430 的电子琴
硬件课程设计

项目名称：基于 MSP430 的电子琴

团队成员：电信提高 0901 班 熊绪胜(U200913668)
电信提高 0901 班 熊倪 (U200913783)
电信提高 0901 班 项明 (U200913845)

指导教师: 汪小燕老师

2012 年 7 月 4 日

课题名称：基于MSP430的电子琴

【摘要】 本次课程设计基于MSP430设计一个电子琴装置，该装置能产生不同的音调，对应电子琴的各个琴键的声音。通过按键能够控制音调的各个属性，例如音阶、音量、节拍等，以模拟电子琴的参数设置，同时该装置具有播放一整段乐曲的功能。

电子琴的每个乐音对应不同频率的声音信号，我们的思路是利用 MSP430 的 DAC 转换电路将预先设定的乐谱信息转化成对应频率的音频信号输出到扬声器，通过矩阵键盘的中断控制 DAC 的各个参数，从而调节每个音调的属性，以模拟出设置音阶、音量、节拍的功能。

通过编程手段可以达到连续 DAC 转换输出连续不同的音调，这样就能播放一整段的乐曲。DAC 模块输出的信号是很稳定、纯净的但音频信号，能保证乐音的音质清晰，但由于 DAC 转换的信号比较微弱，所以通过一个功率放大器，可以调节声音大小，完成音量调节的设计。

【关键词】: MSP430 电子琴 界面

[Abstract] The course design is based on the MSP430 design a keyboard device, the device can produce a different tone, corresponding to the sound of keyboard keys. The button can control the pitch of each property, such as scale, volume, tempo and other parameters to simulate the keyboard settings, the device has a function to play a whole section of music.

The keyboard music corresponding to the signals of different frequencies of sound, our idea is that the DAC conversion circuit using the MSP430 will be pre-set music information into the corresponding frequency of the audio signal output to the speaker through the matrix of the keyboard interrupt control various parameters of the DAC , thereby regulating the properties of each pitch to simulate the function of a set of scales, volume, tempo.

Can be achieved by programming means consecutive DAC conversion output for different tone, so you can play a whole song. The DAC module output signal is very stable, pure audio signal to ensure the sound quality of the music is clear, DAC conversion signal is relatively weak, so a power amplifier, you can adjust the loudness of sound, complete the design volume adjustment.

目录

1.概述

本次课程设计旨在设计一个基于 MSP430 系列微处理器的简易电子琴装置。MSP430 系列 MCU 是 Texas Instruments（TI）公司生产的 16bit RISC 超低功耗混合信号处理器，能为各种低功耗和便携式应用提供最终解决方案。作为混合信号和数字技术的领导者，TI 创新生产的 MSP430 使系统设计人员能够在保持独一无二的低功率的同时同步连接至模拟信号、传感器和数字组件，其典型应用包括实用计量、便携式仪表、智能传感和消费类电子产品。

MSP430 可提供 200 多种超低功耗微处理器器件。每个器件都具有灵活的时钟系统，启用了多达 7 种低功率模式（LPM），可提高优化性能。如果配以低于 1us 的即时唤醒时间以及各种中断源，MSP430 可确保您的应用仅使用手动执行任务时所需的相应时钟和外设。

其主要的超低功耗度量标准：

- 各种低功耗工作模式
  超低功耗工作模式：最低 120μA/MHz@2.2V
  待机模式，具有自我唤醒功能、RAM 保持模式（LPM3）：最低 0.7μA@2.2V
  待机模式，具有自我唤醒功能（LPM4）：最低低于 100nA@2.2V
  停机模式，具有 RAM 保持模式（LPM3.5）：最低低于 100nA@2.2V
- 低功率模式下低于 1us 的即时唤醒时间
- 始终接通的零功耗掉电复位

MSP430 采用冯诺依曼架构，通过通用存储器地址总线（MAB）和存储器数据总线（MDB）将 16 位 RISC CPU、多种外设和灵活的时钟系统进行完美结合。MSP430 通过将先进的 CPU 与模块化内存映像模数外设相结合，为当今和未来的混合信号应用提供了解决方案。MSP430 平台内包括五代超低功耗、高度集成的微处理器产品，涵盖了 200 多款器件。每一代产品都提供各种级别的模拟集成、数字外设和通信协议，以帮助开发者查找用于各种应用的合适的微处理器。

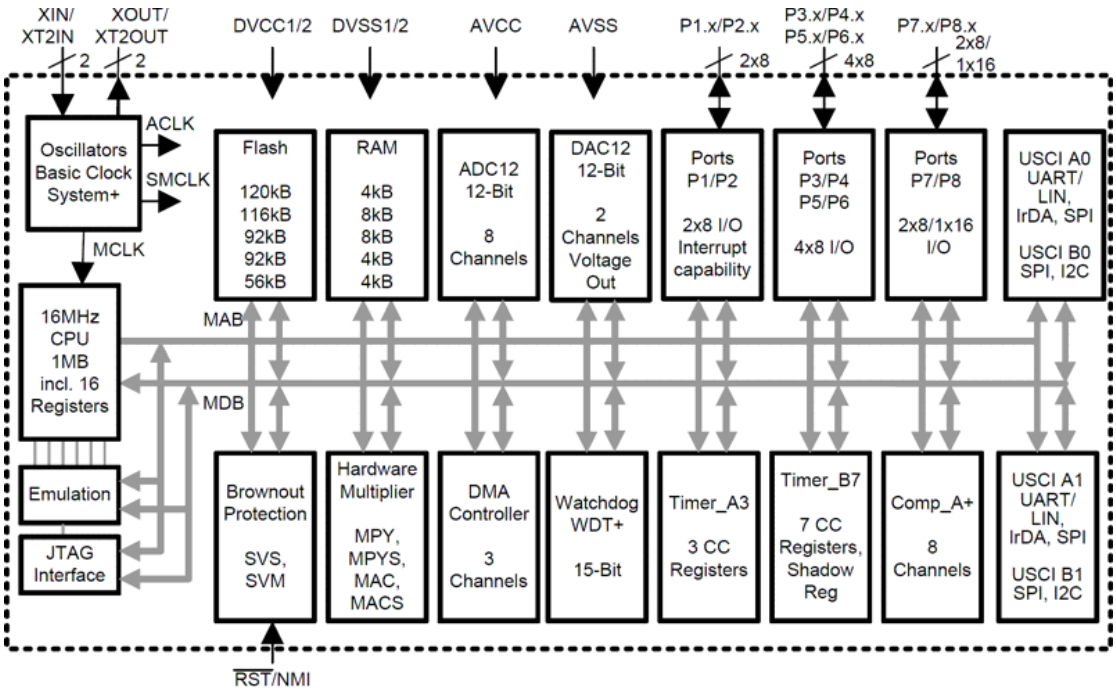
本次硬件课程设计使用 MSP430 的 MSP430F2619 开发板，并使用 CCS v5.1 进行 MSP430 上的编程和调试。

MSP430F2619 简介：

- 低工作电压范围：1.8V—3.6V
- 超低功耗
  Active 模式：365μA@1MHz,2.2V
  待机模式（VLO）：0.5μA
  掉电模式（RAM 保持）：0.1μA
- 从待机模式唤醒时间小于 1us
- 16bit RISC 架构，62.5ns 指令周期
- 3 通道内部 DMA
- 12bit A/D 转换器，包含内部参照、采样保持、自扫描特性
- 双通道异步 D/A 转换器
- 片上比较器
- 4 个通用串行通讯接口（USCI）
  USCI_A0 和 USCI_A1：增强型 UART，支持波特率自动检测，红外通信编解码器，同步 SPI
  USCI_B0 和 USCI_B1：I2C，同步 SPI

- 可编程检测的供电监视器
- 掉电检测器
- 引导程序（BSL）
- 串行板上编程，无需外部编程电源，保险丝实现代码保护
- 120KB+256B Flash, 4KB RAM

MSPF2619 功能方框图：



Note: Memory sizes, available peripherals, and ports may vary depending on the device.

MSP-FET430UIF 简介

MSP-FET430UIF 是一个功能强大用于快速开发 MSP430 MCU 应用的 flash 仿真工具。它包含 USB 调试接口，通过连接板上的 JTAG 接口或保存了 Spy Bi-Wire（2-JTAG）协议的引脚以对 MSP430 进行编程和调试。仅需几次点击便能在数秒内对 Flash 进行擦除和编程，由于 MSP430 flash 是超低功耗的，无需外部电源供电。MSP-FET430UIF 支持所有 MSP430 Flash 设备。USB 调试接口连接基于 Flash 的 MSP430 MCU 和 PC 以实时，在线地进行编程和调试。

其技术特性如下：

- 软件可配置的供电电压：1.8V—3.6V@100mA
- 支持 JTAG 保险丝保护代码
- 支持所有有 JTAG 接头的 MSP430 开发板
- 支持 JTAG 和 Spy-Bi-Wire（2-Wire JTAG）协议

Code Composer Studio（CCS） v5.1 简介

Code Composer Studio（CCStudio）适用于德州仪器（TI）嵌入式处理器系列的集成开发环境（IDE），包含一整套用于开发和调试嵌入式应用的工具。它包含适用于每个 TI 器件系列的编译器、源码编辑器、项目构建环境、调试器、描述器、仿真器、实时操作系统以及多种其他功能。直观的 IDE 提供了单个用户界面，可帮助您完成应用开发流程的每个步骤。借助于精密的高效工具，用户能够利用熟悉的工具和界面快速上手并将功能添加至他们的应用。

Code Composer Studio 以 Eclipse 开源软件框架为基础。Eclipse 软件框架最初作为创建开发工具的开发框架而被开发。Eclipse 为构建软件开发环境提供了出色的软件框架，并且逐渐成为备受众多嵌入式软件供应商青睐的标准框架。CCStudio 将 Eclipse 软件框架的优点和 TI 先进的嵌入式调试功能相结合，为嵌入式开发人员提供了一个引人注目、功能丰富的开发环境。为帮助更多新手开发者学习，CCS 提供了多项许可选项：

- 评估：免费的有限许可可用于评估 TI 工具和器件
- 节点锁定：许可颁发至特定计算机
- 浮动：许可在多个计算机之间共享
- 代码大小限制：MSP430 具有免费 16KB 代码大小有限许可
- 捆绑包/开发套件：免费许可可与 EVM 和开发板（具有板载仿真）以及 XDS100 类仿真器一同使用
- 大学：联系 TI 大学计划相关人员以了解详细信息

2. 设计目标

硬件课程设计的目的在于使学生通过硬件课程设计教学环节较系统地完成电子系统设计从选题、方案论证、电路设计、电路实现、装配调试、系统测试、总结报告等基本过程，加深对模拟电路、数字逻辑电路、通信电子电路、微机原理等相关课程理论知识的分析理解。引导学生把原理分析与工程设计实现相结合，掌握实际电子与通信系统设计的基本方法和一般规则，提高综合应用所学理论知识的能力，培养学生的创新思维和实践能力，为后续专业课程的学习打下坚固的基础。

本项目要求设计并实现一个信号产生与分析处理装置，该装置能够产生预期特殊测试信号，并根据用户输入选择产生指定类型与参数的信号，同时在必要的辅助输出显示设备上显示产生信号的类型与参数。同时可针对输入信号进行分析处理，判读并显示相应信号的特征参数。

2.1 基本功能

1) 具有使用矩阵键盘产生“1,2,3,4,5,6,7”对应音调的电子琴功能；
2) 可通过按键增大或输出波形幅度；
3) 有外界按键输入选择电子琴发音的音调、音阶和幅度；
4) 界面显示输出电子琴发音音调、音阶和幅度等信号参数；
5) 特定音乐的自动播放。

2.2 拓展功能

1) 拓展输出音阶组数；
2) 通过按键选择节拍输出；

3. 团队组成与任务分工

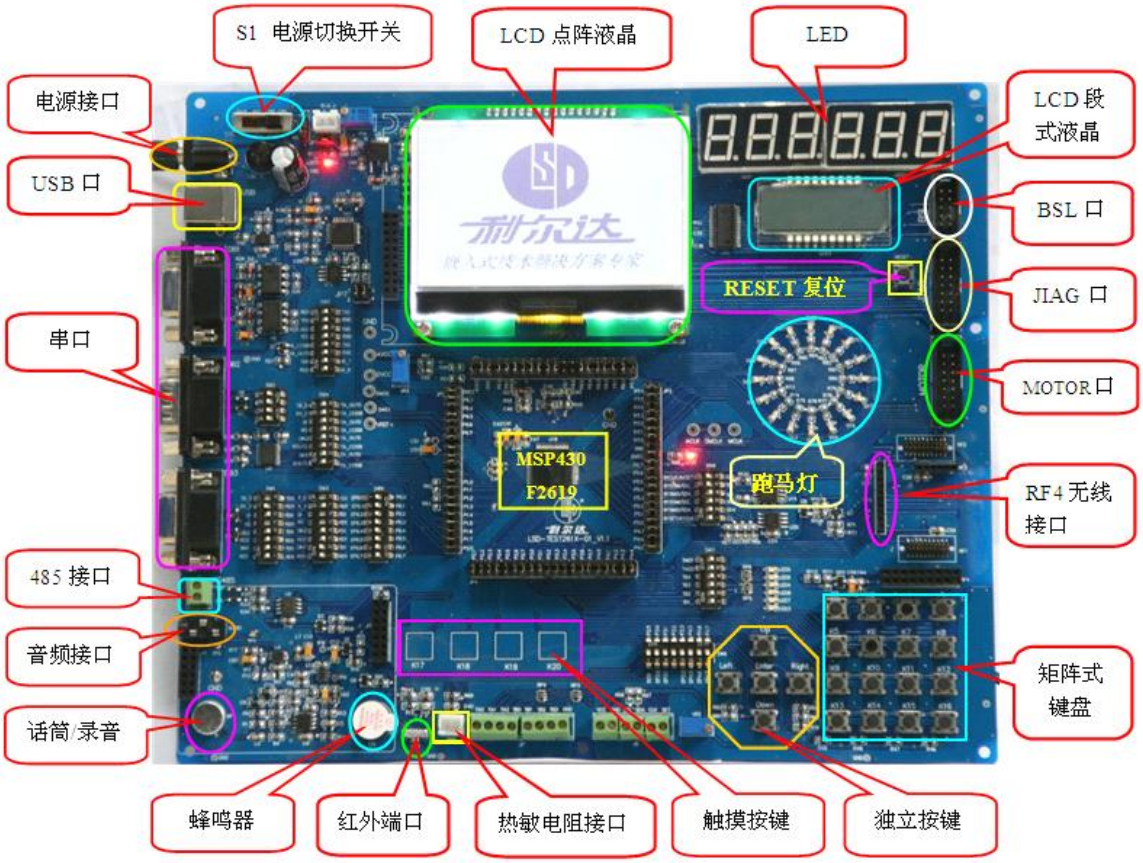
熊倪： 1. 音乐播放功能设计与实现；
2. 播放功能与界面连接；
熊绪胜：1. 人机交互接口的设计与实现；
2. 播放功能与界面连接；
项明： 1. TI 文档撰写；
2. 乐谱编撰

4. 总体设计方案

4.1 系统工作平台

我们经过多次的研究和调查，决定使用实验室已经有的LSD-TEST430F261X实验板。

LSD-TEST430F261x实验板以MSP430F2619为核心，扩展出了许多实用功能。对于MSP430F2619芯片而言，其具有超低功耗，相应时间小，FLASH存储器达120KB+256B,RAM达4KB的优点，便于学习与开发。


S1 电源切换开关
LCD 点阵液晶
LED
电源接口
USB 口
LCD 段式液晶
BSL 口
RESET 复位
JIAG 口
串口
MOTOR 口
MSP430
F2619
跑马灯
RF4 无线接口
485 接口
音频接口
矩阵式键盘
话筒/录音
蜂鸣器
红外端口
热敏电阻接口
触摸按键
独立按键

4.2 总体设计方案

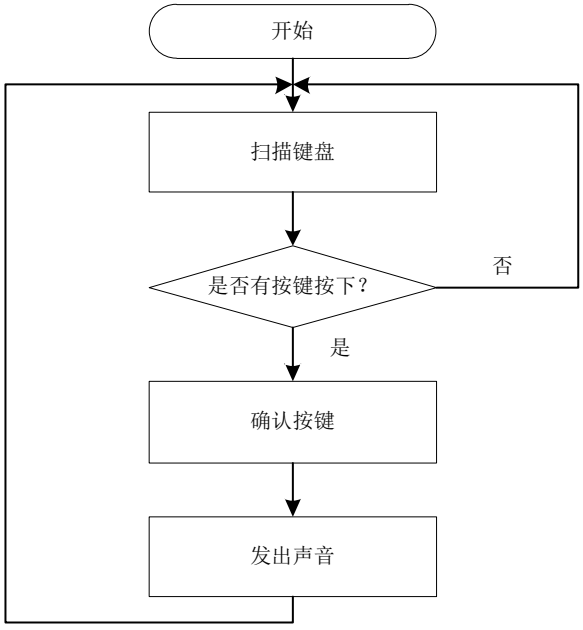
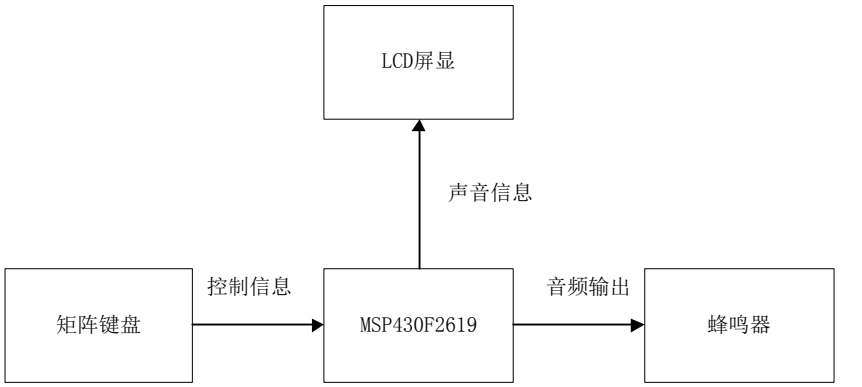

开始
扫描键盘
是否有按键按下？
否
是
确认按键
发出声音

图 1. 软件设计框图



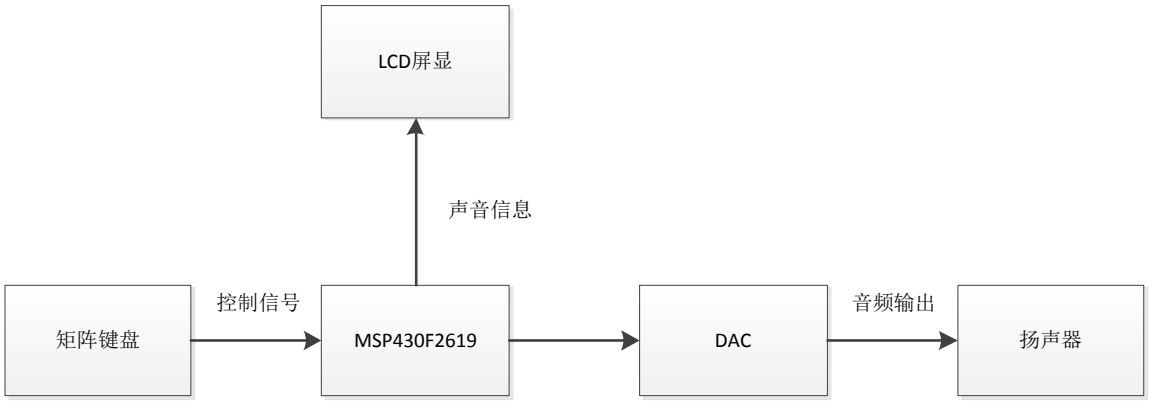


图 2. 硬件设计框图

4.3 项目综述

这款由我们自己设计的基于 MSP430F2619 的电子琴，实现了 C 大调 12 个（包括半音）音符的发声。并且可以通过按键调节音阶，音量，音乐节拍与播放预先存储的音乐（机器猫主题曲）。为了实现发出 Do,Re,Mi,Fa,So,La,Xe7 个基本音符，我们通过 MSP430F2619 输出一路频率可控方波，通过调节频率放出 7 个基本音符。并且可以实现按键调节音调，节拍与音量。除此之外，我们还在 LCD 屏幕编写了人机交互界面，是用户能够更好的使用我们的电子琴。

5. 系统硬件设计与实现

5.1 语音输出模块

5.1.1 综述

为了实现电子琴最基本的发声功能，我们需要从 MSP430F2619 芯片输出一路频率可调的模拟信号，经放大滤波后输出到扬声器中，得到声音信号。此声音信号中，要包含可以调节的频率，幅度和包络信息，分别对应声音的音调，音量和音色。在完成了基本音调发声之后，只需通过键盘控制将多个音组合在一起便可以达到电子琴弹奏的目的。

5.1.2 语音输出硬件设计

由于MSP430F2619 片内集成了数摸转换器(D/A)，所以F2619 可以直接输出模拟量，但是输出的信号相对较微弱，而且还是单极性信号，这样就需要外面连些功率放大、转换这样的电路来放大信号，去驱动相关负载。LSD_TEST430F261x 学习板有如图2所示的信号处理

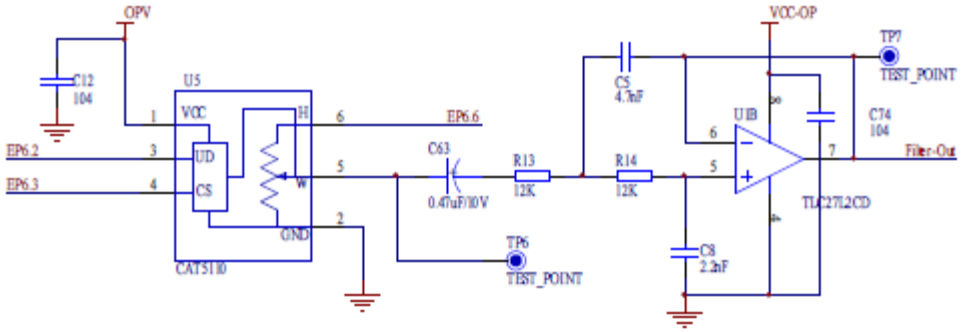


图3.1

电路。为了调节输出信号的大小，也随之使用了一片可编程数字电位器（CAT5110）来调节。

电位器调节后的信号从TLC27L2 的第5 脚输入，经虑波处理后由7 脚输出；由于单片机输出的是单极性信号，而我们的耳机最好是双极性信号驱动，所以使用了一片TI 公司的音频处理专用芯片TPA301。具体如图3.2所示。

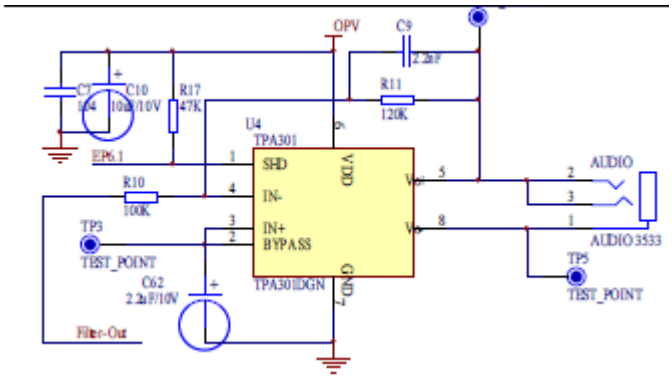
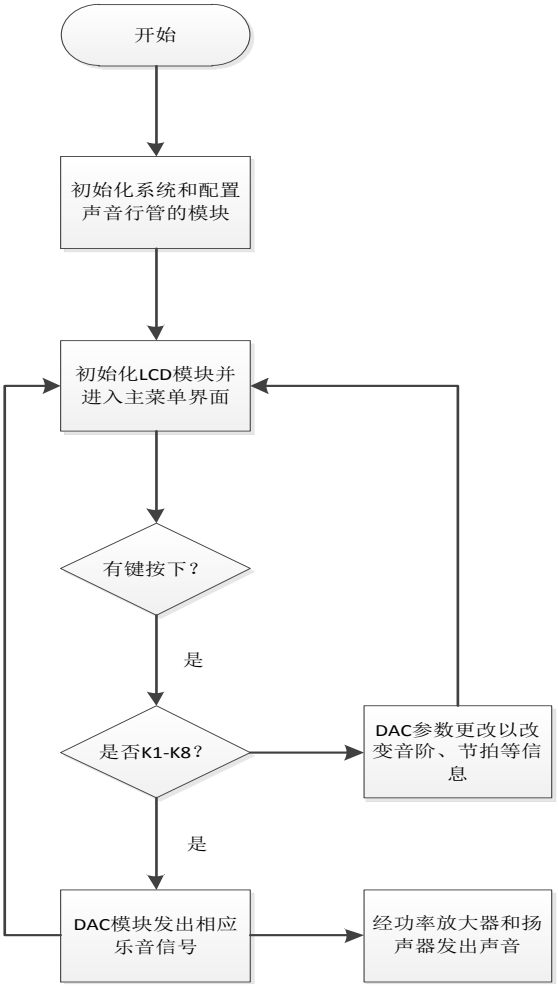


图3.2

5.1.3 语音输出流程



5.1.4 软件模块

- void delay_ms(uint n);延时n/10毫秒

```
void delay_ms(uint n)
{
       uchar i;
       while(n--)
          {
              for(i=0;i<12;i++);
           }
}
```

本函数实现了在MSP430的编程环境下，系统演示n/10毫秒的功能，使得我们可以用一种简单可行的方法实现对波形的频率控制。

- void Play_Sound(int temp[2])

```
void Play_Sound(int temp[2])
    {
          int i;
               for(i=0;i<temp[0];i++)
                {
                  DAC12_0DAT=0xfff;
                  delay_ms(temp[1]);
                  DAC12_0DAT=0x000;
                  delay_ms(temp[1]);
                  }

    }
```

本函数用于播放基本的单音节声音。通过对DAC12_0 data置高电平0xfff与低电平0x000来输出方波信号，延时的时间即是其周期。函数中temp数组的第一个，第二个元素分别对应声音的播放时间与播放频率信息。通过预先定义好的声音信息，调用Play_Sound()函数就能播放出预先定义好的一个个音符。

- void Playsound(int a,int b)

```
void Playsound(int a,int b)
    {
       int i;
           for(i=0;i<a;i++)
            {

              DAC12_0DAT=0xfff;
              delay_ms(b);
              DAC12_0DAT=0x000;
              delay_ms(b);
              }
    }
```

本函数对应与播放音乐的 Play_Music 函数。为了完成播放应用数组存储在 MSP430F2619 中的歌曲，重写撰写的单音节播放函数。将音符的节拍与频率信息分开存储。

- void Play_Music(int a[],int b[]i)

```
void Play_Music(int a[],int b[])
    {

      int i=0,j;
      for (j=0;j<86;j++)
      {
        Playsound(a[i],b[i]);

          delay_ms(5000);
         i++;
      }
    }
```

本函数用于实现播放预先存储在MSP430F2619芯片中的歌曲。将歌曲中每一个音符的节拍与音调信息分别存储在数组a与数组b中。通过Play_Music函数每次播放一个音符，从而得以播放整个音乐。

```
int Do[2]={500,76};
int Doo[2]={1056/2,72};
int Re[2]={1117/2,68};
int Ree[2]={1188/2,64};
int Mi[2]={1245/2,61};
int Fa[2]={1333/2,57};
int Faa[2]={1407/2,54};
int Faaa[2]={703/2,108};
int So[2]={1490/2,51};
int Soo[2]={1583/2,48};
int La[2]={1688/2,45};
int Laa[2]={1767/2,43};
int Xi[2]={1853/2,41};
int* Sound[]={Do,Doo,Re,Ree,Mi,Fa,Faa,So,Soo,La,Laa,Xi};
```

预先定义的12个音符信息。

5.1.5 模块测试结果

首先，我们输出频率为140Hz左右的Do，波形为方波，扬声器发出非常清晰的Do音。进而我们验证了此音阶Do到Xe的所有音符播放，都能得到悦耳清晰的声音。接着我们分别降低八度和升高八度重新试验，都得到了较为清晰的声音。

但是我们发现，在不断升高音阶的同时，声音的音色会逐渐变差，这与噪声的加入有关。在低频，由于频率很低，噪声影响并不明显，但是到了高频，噪音就变得突出，导致声音音色不佳。

5.2 声音调节模块

5.2.1 综述

为了调节我们通过扬声器发出声音的音调，节拍，与音量，我们加入了声音调节模块。使得我们通过控制4*4键盘就能完成对上述物理量的调节。

5.2.2 软件模块

节拍调节

```
for(i=0;i<12;i++)
        { int temp,temp1;
        temp=Sound[i][1];
        Sound[i][1]=temp;
        temp1=Sound[i][0]*2;//temp1=Sound[i][0]/2;//如果是降低
        Sound[i][0]=temp1;
        }
```

此过程用来调节声音播放的节拍，用一个循环更改 12 个音符（包括半音）的播放时间大小。

音阶调节：

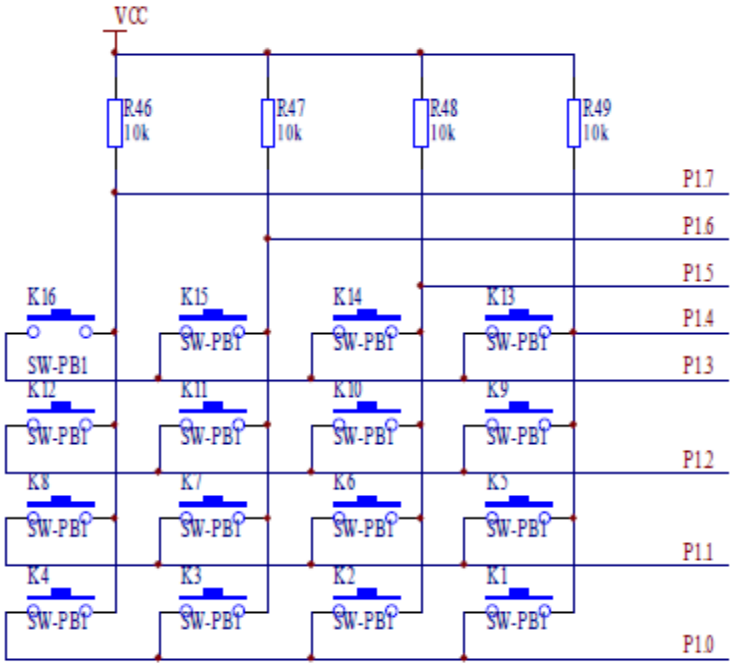
```
  for(i=0;i<12;i++)
        { int temp,temp1;
        temp=Sound[i][1]/2;
        Sound[i][1]=temp;
        temp1=Sound[i][0]*2;
        Sound[i][0]=temp1;
        }
```

此过程用来调节音乐的音阶，通过将每个音符的频率加倍或则减半来达到调节音阶的目的，同时为了不改变节拍，要将播放时间做相反的处理。

5.3 键盘扫描模块

5.3.1 键盘扫描原理

如下图所示，列线P1.4～P1.7 通过上拉电阻接电源，处于输入状态，行线P1.0～P1.3 为输出状态，键盘上没有按键按下时，所有列线输入为高电平。当键盘上某个按键按下时，则对应的行线和列线短接。例如，当K1 键按下时，P1.4 与P1.0 短接，此时P1.4 的输入电平由P1.0 决定。在检测是否有键按下时，先使4 条行线输出低电平，然后读取4 条列线的状态。如果全部为高电平则表示没有任何键被按下；如果有任何一个键被按下，由于列线是弱上拉到VCC，则列线上读到的将是一个非全“1”的值。



上述过程只说明了如何判定是否有键被按下，但不能判定是哪一个键被按下，对于行列扫描式键盘的方法识别按键，首先看输入的行线，假设4 条行线都输出低电平，4 条列线将输入全不为高的电平，这时就确定了有键被按下，这时就将行线的P1.0 置低，其它全置高，读取列线P1.4～P1.7 上的值，如果有键按下，那么就有一个唯一的值对应；反之，如果没有，读到将全为“1”，这时就从P1.1 换到P1.2 置低电平，继续读取列线P1.4～P1.7 上的值，如果有键按下，

那么就有一个唯一的值对应；反之，如果没有读到的将还是全为“1”，继续一下轮的扫描。单片机就这样循环的执行着，只要有键按下，就将有一个唯一的值被对应，根据这个值来判断是哪个键被按下了，执行相应的动作。这就是矩阵式键盘扫描的方法。具体实现代码如下：

```
char KeyScan(void)
{
   char scancode,tempcode;

   P1IFG = 0x00;

   if( (P1IN&0xF0)!=0xF0 )                    //P1.4--P1.7 口不全为“1”则与键按下
   {
      Delays(2);
      if((P1IN&0xF0)!=0xF0)                    //再判断一次
      {
         scancode=0xFE;
         while( (scancode&0x10)!=0 )          //判断 4 次循环扫描结束没？
         {
            P1OUT = scancode;
            if    ((P1IN&0xF0)!=0xF0)          //是该行有键按下
            {
               tempcode = (P1IN&0xF0)|0x0F;
               return((~scancode)|(~tempcode));
            }
            else
            {
               scancode=(scancode<<1)|0x01;
            }
         }
```

5.3.2 键盘实现

在使用键盘时，我们用到了switch/case语句，这样在按下不同的按键时可以根据按下键的不同来执行不同的功能：

```
void Key_Process(uchar KeyCode)
{

        int i;
    switch(KeyCode)
    {
/*发出 do 音*/
    case 0x11:
       ShowBMP(0,0,240,83,PK1);
       delay_ms(10000);
       ShowBMP(0,0,240,83,RK);
         Play_Sound(Sound[0]);
           break;
 /*发出 re 音*/
    case 0x21:
       ShowBMP(0,0,240,83,PK2);
       delay_ms(10000);
       ShowBMP(0,0,240,83,RK);
       Play_Sound(Sound[2]);
          break;
 /*发出 mi 音*/
    case 0x41:
       ShowBMP(0,0,240,83,PK3);
       delay_ms(10000);
       ShowBMP(0,0,240,83,RK);
         Play_Sound(Sound[4]);
          break;
发出 fa 音*/
    case 0x81:
       ShowBMP(0,0,240,83,PK4);
       delay_ms(10000);
       ShowBMP(0,0,240,83,RK);
       Play_Sound(Sound[5]);
          break;
```

```
/*发出 sol 音*/
      case 0x12:
         ShowBMP(0,0,240,83,PK5);
         delay_ms(10000);
         ShowBMP(0,0,240,83,RK);
       Play_Sound(Sound[7]);
            break;
 /*发出 la 音*/
      case 0x22:
         ShowBMP(0,0,240,83,PK6);
         delay_ms(10000);
         ShowBMP(0,0,240,83,RK);
       Play_Sound(Sound[9]);
            break;
 /*发出 si 音*/
      case 0x42:
         ShowBMP(0,0,240,83,PK7);
         delay_ms(10000);
         ShowBMP(0,0,240,83,RK);
         Play_Sound(Sound[11]);
            break;
  /*播放特定音乐*/
      case 0x82:
         ShowBMP(0,0,240,83,PK8);
         Play_Music(a,b);
         ShowBMP(0,0,240,83,RK);
          break;
/*增加音量*/
      case 0x14:
            CAT5110_INC();
            ShowBMP(0,0,240,83,RK);
            vpp++;
            d[0]=(uchar)((vpp/10)%10+48);
            d[1]=(uchar)((vpp)%10+48);
            Display_String(5,100,"音量：  ");
            Display_String(20,100,d);
            Display_String(26,100,"       ");
            Display_String(30,140, "      返回（K16）" );
            break;
```

```
/*减小音量*/
      case 0x24:
            CAT5110_DEC();
            ShowBMP(0,0,240,83,RK);
            vpp--;
            d[0]=(uchar)((vpp/10)%10+48);
            d[1]=(uchar)((vpp)%10+48);
            Display_String(5,100,"音量：  ");
            Display_String(20,100,d);
            Display_String(26,100,"        ");
            Display_String(30,140, "      返回（K16）" );
            break;
/*增大节拍*/
      case 0x44:
        for(i=0;i<12;i++)
          { int temp,temp1;
          temp=Sound[i][1];
          Sound[i][1]=temp;
          temp1=Sound[i][0]*2;
          Sound[i][0]=temp1;
          }
            ShowBMP(0,0,240,83,RK);
            if(jpp==1)
               jpp++;
            jpp=jpp/2;
            j[0]=(uchar)((jpp/10)%10+48);
            j[1]=(uchar)((jpp)%10+48);
            Display_String(5,120,"节拍：1/");
            Display_String(29,120,j);
            Display_String(35,120,"                        ");
            Display_String(30,140, "      返回（K16）" );
            break;

/*减小节拍*/
      case 0x84:
          for(i=0;i<12;i++)
          { int temp,temp1;
          temp=Sound[i][1];
          Sound[i][1]=temp;
```

```
        temp1=Sound[i][0]/2;
        Sound[i][0]=temp1;
        }
          ShowBMP(0,0,240,83,RK);
          jpp=jpp*2;
          j[0]=(uchar)((jpp/10)%10+48);
          j[1]=(uchar)((jpp)%10+48);
          Display_String(5,120,"节拍：1/");
          Display_String(29,120,j);
          Display_String(35,120,"                    ");
          Display_String(30,140, "      返回（K16）" );
          break;

/*升高音调*/
     case 0x18:

        for(i=0;i<12;i++)
        { int temp,temp1;
        temp=Sound[i][1]/2;
        Sound[i][1]=temp;
        temp1=Sound[i][0]*2;
        Sound[i][0]=temp1;
        }
        p++;
        tone_switch(p);


        break;
/*降低音调*/
     case 0x28:
         for(i=0;i<12;i++)
        { int temp,temp1;
        temp=Sound[i][1]*2;
        Sound[i][1]=temp;
        temp1=Sound[i][0]/2;
        Sound[i][0]=temp1;
        }
        p--;
        tone_switch(p);
          break;
```

```
    case 0x48:
    break;
/*进入选择播放模式菜单*/
    case 0x88:
      if(flip)
      {
        ClearRAM();
        Display_String( 20,20,   "电子琴演示实验" );
        Draw_Line( 0,50,240);
        Display_String( 20,70,   "高品质模式 K16" );
        flip=0;
        break;
      }
      else
      {
        ClearRAM();
        Display_String( 22,20,   "高品质模式" );
        Draw_Line( 0,50,240);
        Display_String( 25,70,   "播放歌曲 K8" );
        Display_String( 20,90,   "按 K1~K7 弹奏歌曲" );
        delay_ms(50000);
        delay_ms(50000);
        delay_ms(50000);
        delay_ms(50000);
        delay_ms(50000);
        delay_ms(50000);
        delay_ms(50000);
        ClearRAM();
        ShowBMP(0,0,240,83,RK);
        Display_String(5,100,"音量:10");
        Display_String(5,120,"节拍:");
        Display_String(5,140,"音调: c");
        Display_String(30,140, "      返回（K16）" );
        flip=1;
        break;
      }
    default:
        break;
    }    }
```

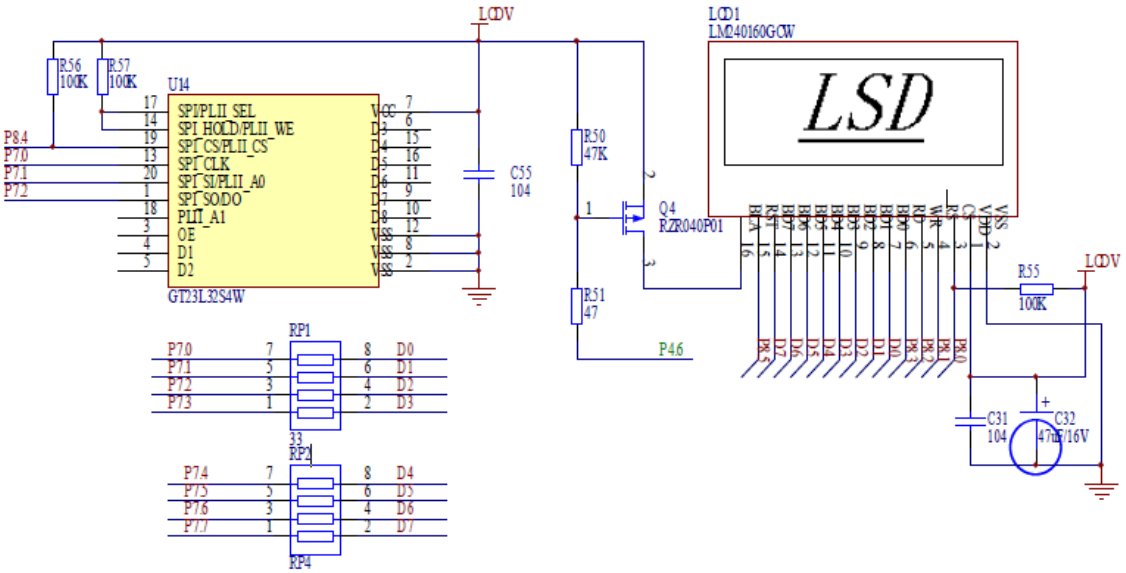
在main函数中我们需要编写如下函数来调用Key_Process函数：

```
while(1)
{
   P1OUT = 0xF0;
   _BIS_SR(GIE+CPUOFF);
   KeyCodeTemp = KeyScan();
   Key_Process(KeyCodeTemp);
}
```

5.4 LCD 显示模块

5.4.1 LCD 显示原理

基于LM240160GCW 的液晶模块与MSP430F2619 的连接原理如图4-12 所示，由于MSP430 系列单片机的总线不对外，所以我们不能以直接的方式而只能采用间接的方式访问外设，如图所示，P7口设为数据总线，用来传输数据或指令，P8 的部分端口用来做控制总线，控制LM240160GCW的相关功能。GT23L32S4W 为大容量字库存储芯片，由P7.0～P7.2、P8.4 通过SPI 总线协议与其通讯，液晶上想要显示的汉字或字符码直接从该芯片中读取。由R50、R51、Q4 组成的是一个LCD 背光控制电路，由MSP430 送出PWM 信号实现。



LM240160GCW 液晶模块显示内容相对较丰富，最大分辨率240X160，其性能介绍如下：
内置ST7529 液晶显示模块系列的指令系统比较简单。需要特别说明与关注的是模块内的显示RAM 的结构。在模块的RAM 中，一个单元的地址由行地址和列地址组合而成，地址定义如下：
行地址：按照像素行定义，以显示屏自上而下顺序排序，取值范围是1~160 行。（ 以240*160 点阵为例）列地址：以水平像素点为基础，每3 个像素点为一组，由一个列地址唯一指定。我

们可以用下面2 个思路来理解列地址：

1、 RAM 单元的数据宽度。在模块RAM 单元的数据位为15 位，每5 位数据为一个像素点的显示数据，一个RAM 单元容纳了3 个像素的数据。
2、 列块表示。以3 个像素点的数据组合成一个列块，每一次的读写RAM 时，必须以3 个数据连续写入/读出为一个最小操作。

这两种理解，前者有利于对如此结构的认同与理解，后者则为读写操作时的提示。列地址（或列块地址取值范围是0-79，（ 以240*160 点阵为例）
行块地址：每4 行为一组，将160 行分为1-40 行块，（ 以240*160 点阵为例），形成行块地址。块地址将应用于局部显示和卷动功能上。一个像素的数据为5 位组成，在数据字节中占据了DB7～DB3，该字节的低3 位DB3～DB0 无用。DB7～DB3 设置数据不同，将产生不同的灰度级显示，最大能产生32 级灰度；数据的DB2～DB0 位无用，一般我们设置为0。

5.4.2 显示字符原理

GT23L32S4W 是一款内含11X12 点阵、15X16 点、24X24 点阵、32X32 点阵的汉字库芯片，支持GB2312 国标汉字（含有国家信标委合法授权）及SCII 字符。排列格式为横置横排。用户通过字符内码，参考例程中的方法计算出该字符点阵在芯片中的地址，可从该地址连续读出字符点阵信息。
SPI 接口引脚描述
串行数据输出（SO）：该信号用来把数据从芯片串行输出，数据在时钟的下降沿移出。
串行数据输入（SI）：该信号用来把数据从串行输入芯片，数据在时钟的上升沿移入。
串行时钟输入（SCLK）：数据在时钟上升沿移入，在下降沿移出。
片选输入（CS#）：所有串行数据传输开始于CE#下降沿，CE#在传输期间必须保持为低电平，在两条指令之间保持为高电平。
线挂起输入（HOLD#）： 该信号用于片选信号有效期间暂停数据传输，在总线挂起期间，串行数据输出信号处于高阻态，芯片不对串行数据输入信号和串行时钟信号进行响应。
当HOLD#信号变为低并且串行时钟信号（SCLK）处于低电平时，进入总线挂起状态。
当HOLD#信号变为高并时串行时钟信号（SCLK）处于低电平时，结束总线挂起状态。
当我们需要用软件来实现LCD显示字符时我们需要首先根据汉字或字符的内码计算出字库中的相对地址，然后通过SPI总线访问，先写地址，再读出数据放入缓存区由外部函数调用显示内容。

5.4.3 LCD 点阵原理

在数字电路中，所有的数据都是以0和1保存的，对LCD控制器进行不同的数据操作，可以得到不同的结果。

5.4.4 LCD 显示实现

由于我们需要用到4*4的键盘、液晶屏和音频，所以我们需要首先将按键端口初始化、液晶端口

初始化和音频端口初始化。下面是初始化的代码：

```
/*************** 4*4 按键端口初始化 ****************************/
P1DIR = 0x0f;   //P1.7--p1.4为输入状态  P1.3--P1.0为输出
P1OUT = 0xf0;
P1IES = 0xf0;   //下降沿中断
P1IE  = 0xf0;   //P1.7--p1.4中断使能
P1IFG = 0x00;   //标志位清0

/***************  液晶接口初始化 ********************************/
P4OUT &= ~( BIT4 + BIT6);
P4DIR |= BIT4 + BIT6;
P7DIR |= 0xFF;
P7OUT =  0xFF;
P8DIR |= BIT0 + BIT1 + BIT2 + BIT3 + BIT4 + BIT5;
P8OUT = 0xFF;

/***************  音频端口初始化 ********************************/
P3DIR |= BIT1;
P3OUT &= ~BIT1;                          // 打开音频电源
P6OUT |= BIT1 + BIT2 + BIT3;
P6DIR |= BIT1 + BIT2 + BIT3;             // 数字电位器端口
```

在main函数中，我们首先编写了首界面，其代码如下：

```
ClearRAM();
Display_String( 22,20,"msp430 电子琴" );
Draw_Line( 0,50,240);
Display_String( 5,70,"小组成员：熊绪胜，熊倪" );
Display_String( 5,90,"指导老师：汪小燕老师" );
Display_String( 0,142,"              下一步(K16)");
```

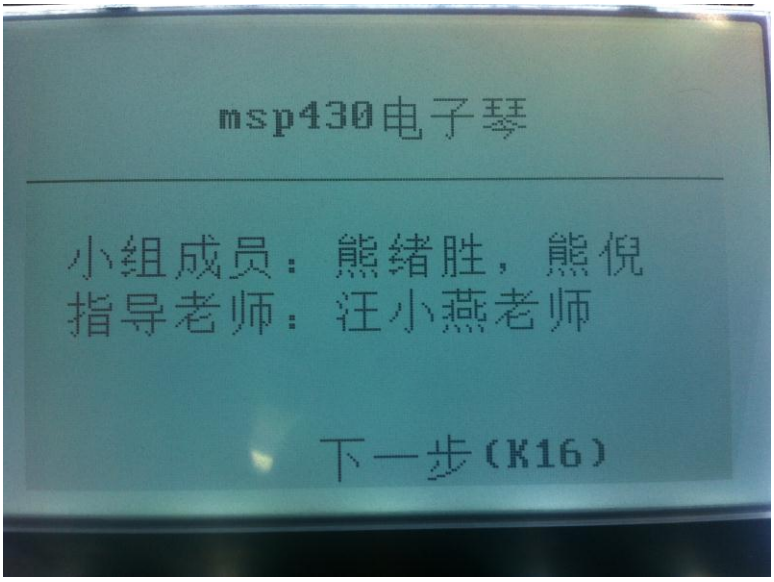
在弹奏音乐的时候我们设计了一个按键界面，当没有键按下时界面保持为8个按键，当有键按下时按下的按键在LCD上对应的按键也模拟按下。这样可以让用户更加真实的感觉到自己在操作电子琴。其中PK_中的_可以表示为1~7中的任意一个数字，PK_在LCD_Graphic.h中被定义为K_按下时所显示的界面图像，而RK则表示没有键按下时显示的界面图像。

```
ShowBMP(0,0,240,83,PK_);
delay_ms(10000);
ShowBMP(0,0,240,83,RK);
```

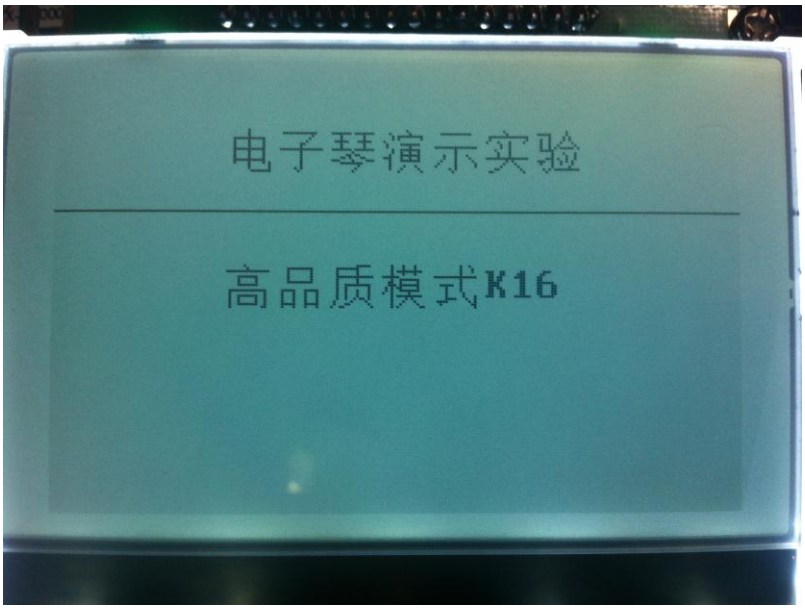
6.系统测试与结果分析

6.1 测试结果

当我们用CCS将代码拷入MSP430F2619开发板中时，我们首先会看到如下的界面显示：

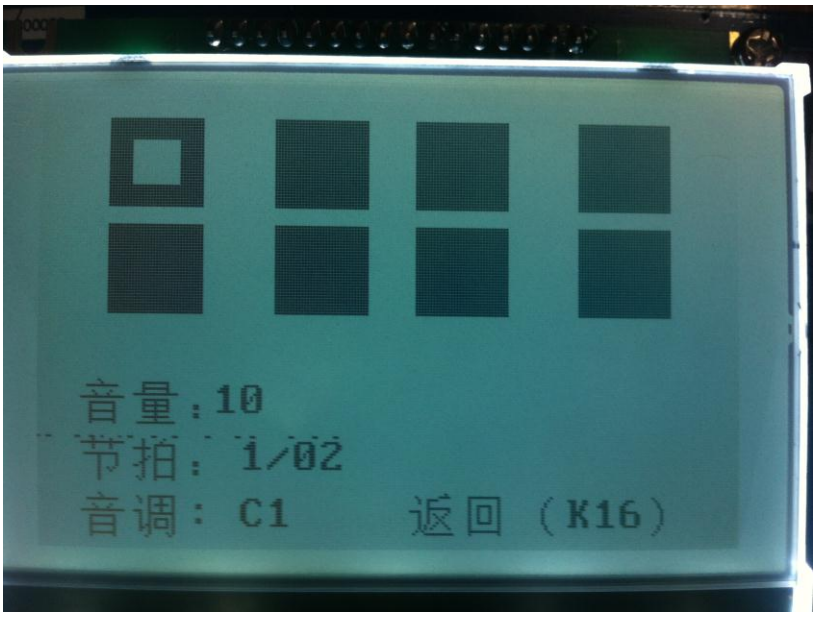


然后我们按下K16，我们就会看到：

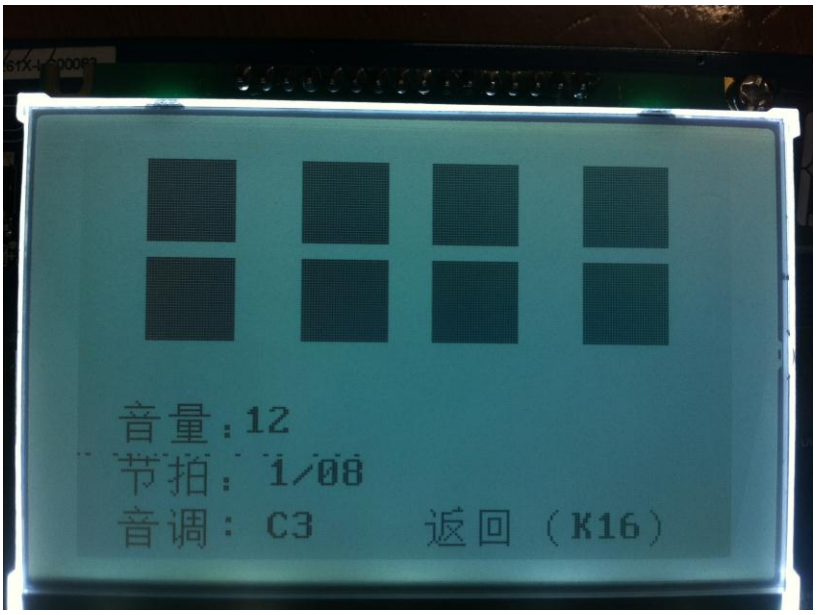


继续按下K16键，出现下面的画面：

由于我们在这没有设定具体的按键进入下一步，所以我们设置了一个延时，等待几秒后就会自动进入下面一个界面：



上图的界面是我们按下K1后，同时发出do音时所出现的界面显示。在这个界面时是音量为10，节拍为1/2拍，音调为C1。这时我们将按下K9（K10）来调节音量，按下K11（K12）来调节节拍，按下K13（K14）来调节音调。在调节后将会出现下图显示：



这时的音量为12，节拍为1/8拍，音调为C3。

6.2 遇到问题

在实际的调试中有时LCD会出现一些乱码，但是大多数时候还是很正常的，出现乱码的可能是由于LCD的电压不足，然后在按键的接触也有点不是太良好，有时会出现按下没反应的情况也有按下过反应的情况。但是这些都只是在很少的时候出现，大多数的时间板子的运作还是比较满意的。

7. 心得体会与项目总结

综上所述，我们的设计完成了既定的所有基本功能，并加入了两个拓展功能。电子琴的各个功能运转良好，没有出现错误。虽然在这过程中，我们遇到过非常大的阻力，比如对矩阵键盘的中断读取，但总算不负我们当初的努力，将各个模块都调试成功。

每一天，我们都到实验室里调试代码，而为了设计出预定的功能，我们自己设计了很多的编程方法，特别是熊绪胜同学为了设计出比较好看的 UI，细致地对比点阵数据，不停的试验，才使得最终我们的按键有了动画效果。而熊倪同学为了代码的成功运行，尝试各种方法，往往一份代码要备份 5-6 份，以免出现错误，丢掉以前的工作。

最终，通过我们的不懈努力，终于制作出这个电子琴装置，这个经历也让我们更加熟悉硬件开发的基本流程和辛苦，也让我们积累下宝贵的经验，我们从这次硬件课设中学到了宝贵的知识技能，这将使我们在以后的学习和工作中收益。

8 致谢

在这次课程设计中，少不了许多人的帮助，在此表示衷心感谢。
特别是汪小燕老师，在设计中给了我们不少激励，鼓励我们向更高的方面要求，每天和我们一起留到实验室关门。对于我们一度的进度缓慢，也给予我们鞭策，希望我们真正扎实地做出一些成绩。同时还要感谢其他组的同学得我们能提供的帮组，通过与他们的交流，我们才能逐渐摸清方向，避免走很多弯路。
最后，感谢电信系和 TI 公司举办这次的竞赛，感谢 TI 公司为我们提供的免费样片，这些都促进了我们更加了解半导体工业的发展潮流，了解我们所学知识的用武之地，感谢所有为此次赛事辛勤付出过的所有人。

9 参考文献

【1】 黄龙松.LSD-TEST430F261X-V1.1 实验指导书.利尔达科技有限公司，2008 年 12 月 10 日.

10 附录

实验主要代码：

```
#include <msp430x26x.h>
#include "General_File.h"


void Delays(uint m )
{
    uint j; uint i;

    for(i=0; i<m; i++)
        for(j=0;j<2000; j++)
            _NOP();
}

void Init_DAC12( void )
{
    DAC12_0CTL = DAC12RES + DAC12IR + DAC12CALON + DAC12AMP_7;        // 输出电压不放大,高速输入输出
```

```
}

void Init_CLK( void )
{
    DCOCTL  = CALDCO_16MHZ;
    BCSCTL1 = CALBC1_16MHZ;
    BCSCTL1 |= DIVA_2;                          // ACLK/4
}

void Init_Port( void )
{
    /***************  音频端口初始化 ********************************/
    P3DIR |= BIT1;
    P3OUT &= ~BIT1;                             // 打开音频电源
    P6OUT |= BIT1 + BIT2 + BIT3;
    P6DIR |= BIT1 + BIT2 + BIT3;                // 数字电位器端口
    /***************  液晶接口初始化 ********************************/
    P4OUT &= ~( BIT4 + BIT6);
    P4DIR |= BIT4 + BIT6;
    P7DIR |= 0xFF;
    P7OUT =  0xFF;
    P8DIR |= BIT0 + BIT1 + BIT2 + BIT3 + BIT4 + BIT5;
    P8OUT = 0xFF;
    /***************  按键端口初始化 ********************************/
    P5OUT &= ~BIT7;                             // 用于按键指示
    P5DIR |= BIT7;
    //P2REN = 0xff;
    /*4*4 按键*/
     P1DIR = 0x0f;   //P1.7--p1.4 为输入状态  P1.3--P1.0 为输出
  P1OUT = 0xf0;
  P1IES = 0xf0;   //下降沿中断
  P1IE  = 0xf0;   //P1.7--p1.4 中断使能
  P1IFG = 0x00;   //标志位清 0
}

void Init_Timer_A( void )
```

```
{
    /************************  语音采样定时  ************************/
    TACTL = TASSEL_2 + MC_1 + TACLR;          //SMCLK = 3M,增计数模式,清空 TAR
    //TACCTL1 = OUTMOD_3;                     //TA1 为输出模式 3,TAR = CCR1 置位,TAR =
CCR0 复位,用于 ADC12/DAC12 的触发信号
    TACCR0 = 2000;
    TACCR1 = 1000;                            //CCR0 = 16M/8K = 2000,CCR1 = CCR0/2,TA1
输出频率为 8K,点空比为 0.5




    /************************  LCD 背光  ****************************/
    TBCTL = TBSSEL_2 + MC_1 + TACLR;          // SMCLK,增计数模式
    TBCCR0 = 8000;                            // PWM 频率设置
    TBCCR3  = 2000;                           // PWM 占空比设置
    TBCCTL3 = OUTMOD_6;                       // LCD 背光控制
}

void CAT5110_INC( void )
{
    //CAT5110_CS_H;
    CAT5110_UD_H;
    CAT5110_CS_L;
    CAT5110_UD_L;
    for( uchar i=0;i<4;i++ )
    {
        CAT5110_UD_H;
        _NOP();
        CAT5110_UD_L;
        Delays( 100 );
    }
    CAT5110_CS_H;
    CAT5110_UD_H;
}
```

```
void CAT5110_DEC( void )
{
    //CAT5110_CS_H;
    CAT5110_UD_L;
    CAT5110_CS_L;
    for( uchar i=0;i<4;i++ )
    {
        CAT5110_UD_H;
        _NOP();
        CAT5110_UD_L;
        Delays( 100 );

    }
    CAT5110_CS_H;
    CAT5110_UD_H;
}

void OFF_CPU( void )
{
    _BIS_SR( CPUOFF );
    _NOP();
}

void Init_MCU( void )
{
    WDTCTL = WDT_ADLY_1000;                          // 定时器模式，ACLK/2,延时2S
    IE1 |= WDTIE;
    Init_CLK();
    Init_ADC12();
    Init_Port();
    Init_Timer_A();
    Init_DAC12();
}

#define     Sampl_Length            0XFFFF
```

```
#pragma vector = ADC12_VECTOR
__interrupt void ADC12_ISR(void)
{
    static ulong Count = 0;

    if( Count++ > Sampl_Length )                //如果内存已满,应该马上退出录音模式
    {
        Count = 0;
        Audio_Exit();
    }
    else
    {
        Sampleresult = (uchar)ADC12MEM0;
        DMA0CTL |= DMAREQ;
    }
}

#pragma vector=WDT_VECTOR
__interrupt void WDT_ISR (void)
{
    IE1 &= ~WDTIE;
    _BIC_SR_IRQ( CPUOFF );
}

#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A1 (void)
{
    static ulong Length = 0;

    switch( TAIV )
    {
    case 2:
        if( Length++ > Sampl_Length )
        {
            Length = 0;
            Audio_Exit();
```

```
        }
        else
        {
            DAC12_0DAT = Flash_Data & 0x00FF;
            DMA0CTL |= DMAREQ;                          // 启动 DMA 传输
        }
        break;
    default:
        break;
    }
}

#include "LCD_Graphic.h"
#include "Sound_Head_File.h"

int
a[]={131,62,187,80,316,85,372,279,105,279,78,250,78,279,158,70,209,74,347,115,316,
93,333,249,77,105,231,125,558,131,62,187,80,316,85,372,279,105,279,78,250,78,279,1
58,70,209,74,463,316,93,250,83,233,70,231,279,500,422,316,93,83,93,211,372,209,77,
263,70,725,422,372,209,77,263,70,745,422,372,666,279,347,105,279,105,279,83};
int
b[]={216,152,152,122,90,122,102,102,90,102,122,114,122,136,180,136,136,114,82,82,9
0,102,114,114,122,180,164,152,136,216,152,152,122,90,122,102,102,90,102,122,114,12
2,136,180,136,136,114,82,90,102,114,114,122,136,164,136,152,90,90,102,114,102,90,1
02,136,122,108,136,102,90,102,136,122,108,136,102,90,102,114,136,82,90,102,90,102,
114};
char KeyCodeTemp;
uint vpp=10;
uchar d[2];
uchar j[2];
uint jpp=4;
uint p=10;
int flip=1;
extern void Delays(uint);
int Do[2]={500,76};
int Doo[2]={1056/2,72};
```

```
int Re[2]={1117/2,68};
int Ree[2]={1188/2,64};
int Mi[2]={1245/2,61};
int Fa[2]={1333/2,57};
int Faa[2]={1407/2,54};
int Faaa[2]={703/2,108};
int So[2]={1490/2,51};
int Soo[2]={1583/2,48};
int La[2]={1688/2,45};
int Laa[2]={1767/2,43};
int Xi[2]={1853/2,41};
int* Sound[]={Do,Doo,Re,Ree,Mi,Fa,Faa,So,Soo,La,Laa,Xi};

void delay_ms(uint n)   //延时n/10毫秒
{
   uchar i;

   while(n--)
   {
     for(i=0;i<12;i++);
}
}
void Play_Sound(int temp[2])
{
  int i;
       for(i=0;i<temp[0];i++)
        {

          DAC12_0DAT=0xfff;
          delay_ms(temp[1]);
          DAC12_0DAT=0x000;
          delay_ms(temp[1]);
          }

}
```

```
void Playsound(int a,int b)
{
   int i;

       for(i=0;i<a;i++)
        {

          DAC12_0DAT=0xfff;
          delay_ms(b);
          DAC12_0DAT=0x000;
          delay_ms(b);
          }
}

void Play_Music(int a[],int b[])
{

  int i=0,j;
  for (j=0;j<86;j++)
  {
    Playsound(a[i],b[i]);

      delay_ms(5000);
     i++;
  }
}

void tone_switch(uint a)
{

  switch(a)
  {
case 4:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"C5");
```

```
        Display_String(30,140, "    返回（K16）" );
        break;
case 5:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"C4");
        Display_String(30,140, "    返回（K16）" );
        break;
case 6:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"C3");
        Display_String(30,140, "    返回（K16）" );
        break;
case 7:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"C2");
        Display_String(30,140, "    返回（K16）" );
        break;
case 8:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"C1");
        Display_String(30,140, "    返回（K16）" );
        break;
case 9:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"C ");
        Display_String(30,140, "    返回（K16）" );
        break;
case 10:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"c ");
```

```
        Display_String(30,140, "    返回（K16）" );
        break;
case 11:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"c1");
        Display_String(30,140, "    返回（K16）" );
        break;
case 12:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"c2");
        Display_String(30,140, "    返回（K16）" );
        break;
case 13:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"c3");
        Display_String(30,140, "    返回（K16）" );
        break;
case 14:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"c4");
        Display_String(30,140, "    返回（K16）" );
        break;
case 15:
        ShowBMP(0,0,240,83,RK);
        Display_String(5,140,"音调:  ");
        Display_String(23,140,"c5");
        Display_String(30,140, "    返回（K16）" );
        break;
  }
}

//键盘
```

```
void Key_Process(uchar KeyCode)
{

      int i;
    switch(KeyCode)
    {
/*发出do音*/
    case 0x11:
      ShowBMP(0,0,240,83,PK1);
      delay_ms(10000);
      ShowBMP(0,0,240,83,RK);
       Play_Sound(Sound[0]);
        break;
 /*发出re音*/
    case 0x21:
      ShowBMP(0,0,240,83,PK2);
      delay_ms(10000);
      ShowBMP(0,0,240,83,RK);
      Play_Sound(Sound[2]);
        break;
 /*发出mi音*/
    case 0x41:
      ShowBMP(0,0,240,83,PK3);
      delay_ms(10000);
      ShowBMP(0,0,240,83,RK);
       Play_Sound(Sound[4]);
        break;
 /*发出fa音*/
    case 0x81:
      ShowBMP(0,0,240,83,PK4);
      delay_ms(10000);
      ShowBMP(0,0,240,83,RK);
      Play_Sound(Sound[5]);
        break;
 /*发出sol音*/
    case 0x12:
```

```
      ShowBMP(0,0,240,83,PK5);
      delay_ms(10000);
      ShowBMP(0,0,240,83,RK);
     Play_Sound(Sound[7]);
        break;
/*发出 la 音*/
   case 0x22:
      ShowBMP(0,0,240,83,PK6);
      delay_ms(10000);
      ShowBMP(0,0,240,83,RK);
     Play_Sound(Sound[9]);
        break;
/*发出 si 音*/
   case 0x42:
      ShowBMP(0,0,240,83,PK7);
      delay_ms(10000);
      ShowBMP(0,0,240,83,RK);
      Play_Sound(Sound[11]);
        break;
 /*播放特定音乐*/
   case 0x82:
      ShowBMP(0,0,240,83,PK8);
      Play_Music(a,b);
      ShowBMP(0,0,240,83,RK);

       break;
/*增加音量*/
   case 0x14:
        CAT5110_INC();
        ShowBMP(0,0,240,83,RK);
        vpp++;
        d[0]=(uchar)((vpp/10)%10+48);
        d[1]=(uchar)((vpp)%10+48);
        Display_String(5,100,"音量：  ");
        Display_String(20,100,d);
        Display_String(26,100,"     ");
```

```
        Display_String(30,140, "    返回（K16）" );
        break;
/*减小音量*/
    case 0x24:
        CAT5110_DEC();
        ShowBMP(0,0,240,83,RK);
        vpp--;
        d[0]=(uchar)((vpp/10)%10+48);
        d[1]=(uchar)((vpp)%10+48);
        Display_String(5,100,"音量：  ");
        Display_String(20,100,d);
        Display_String(26,100,"     ");
        Display_String(30,140, "    返回（K16）" );
        break;
/*增大节拍*/
    case 0x44:
     for(i=0;i<12;i++)
      { int temp,temp1;
      temp=Sound[i][1];
      Sound[i][1]=temp;
      temp1=Sound[i][0]*2;
      Sound[i][0]=temp1;
      }
        ShowBMP(0,0,240,83,RK);
        if(jpp==1)
          jpp++;
        jpp=jpp/2;
        j[0]=(uchar)((jpp/10)%10+48);
        j[1]=(uchar)((jpp)%10+48);
        Display_String(5,120,"节拍：1/");
        Display_String(29,120,j);
        Display_String(35,120,"                ");
        Display_String(30,140, "    返回（K16）" );
        break;

 /*减小节拍*/
```

```
    case 0x84:
      for(i=0;i<12;i++)
      { int temp,temp1;
      temp=Sound[i][1];
      Sound[i][1]=temp;
      temp1=Sound[i][0]/2;
      Sound[i][0]=temp1;
      }
        ShowBMP(0,0,240,83,RK);
        jpp=jpp*2;
        j[0]=(uchar)((jpp/10)%10+48);
        j[1]=(uchar)((jpp)%10+48);
        Display_String(5,120,"节拍：1/");
        Display_String(29,120,j);
        Display_String(35,120,"                ");
        Display_String(30,140, "    返回（K16）" );
        break;

/*升高音调*/
    case 0x18:

      for(i=0;i<12;i++)
      { int temp,temp1;
      temp=Sound[i][1]/2;
      Sound[i][1]=temp;
      temp1=Sound[i][0]*2;
      Sound[i][0]=temp1;
      }
      p++;
      tone_switch(p);


      break;
/*降低音调*/
    case 0x28:
```

```
      for(i=0;i<12;i++)
      { int temp,temp1;
      temp=Sound[i][1]*2;
      Sound[i][1]=temp;
      temp1=Sound[i][0]/2;
      Sound[i][0]=temp1;
      }
      p--;
      tone_switch(p);
        break;

    case 0x48:

      break;
/*进入选择播放模式菜单*/
    case 0x88:
      if(flip)
      {
        ClearRAM();
        Display_String( 20,20,  "电子琴演示实验" );
        Draw_Line( 0,50,240);
        Display_String( 20,70,  "高品质模式K16" );
        flip=0;
        break;
      }
      else
      {
        ClearRAM();
        Display_String( 22,20,  "高品质模式" );
        Draw_Line( 0,50,240);
        Display_String( 25,70,  "播放歌曲K8" );
        Display_String( 20,90,  "按K1~K7弹奏歌曲" );

        delay_ms(50000);
        delay_ms(50000);
        delay_ms(50000);
```

```
        delay_ms(50000);
        delay_ms(50000);
        delay_ms(50000);
        delay_ms(50000);
        ClearRAM();
        ShowBMP(0,0,240,83,RK);
        Display_String(5,100,"音量:10");
        Display_String(5,120,"节拍:");
        Display_String(5,140,"音调: c");
        Display_String(30,140, "    返回（K16）" );
        flip=1;
        break;
      }

    default:
        break;
    }
}


char KeyScan(void)
{
  char scancode,tempcode;

  P1IFG = 0x00;

  if( (P1IN&0xF0)!=0xF0 )              //P1.4--P1.7口不全为“1”则与键按下
  {
    Delays(2);
    if((P1IN&0xF0)!=0xF0)              //再判断一次
    {
      scancode=0xFE;
      while( (scancode&0x10)!=0 )      //判断4次循环扫描结束没?
      {
        P1OUT = scancode;
        if  ((P1IN&0xF0)!=0xF0)        //是该行有键按下
```

```
        {
          tempcode = (P1IN&0xF0)|0x0F;
          return((~scancode)|(~tempcode));
        }
        else
        {
          scancode=(scancode<<1)|0x01;
        }
      }
    }
  }
  return 0;
}

#pragma vector = PORT1_VECTOR
__interrupt void PORT_ISR(void)
{
  P1IFG = 0x00;
  LPM0_EXIT;
}


void main( void )
{
  WDTCTL = WDTPW + WDTHOLD;
  Init_MCU();
  InitLCD();
  _EINT();
  DAC12_0CTL |= DAC12ENC;
  Enabled_TPA301;
  ClearRAM();                                    // 清屏
  Display_String( 22,20, "msp430电子琴" );
  Draw_Line( 0,50,240);
  Display_String( 5,70, "小组成员：熊绪胜，熊倪" );
  Display_String( 5,90, "指导老师：汪小燕老师" );
  Display_String( 0,142, "           下一步(K16)");
```

```
  while(1)
  {
    P1OUT = 0xF0;
    _BIS_SR(GIE+CPUOFF);
    KeyCodeTemp = KeyScan();
    Key_Process(KeyCodeTemp);
  }

}
```