华中科技大学电子与信息工程系
2012 年 TI 杯电子设计大赛项目总结报告

项目名称：smart composer

团队成员：韩东珉

刘欣欣

张　曼

指导教师：曾喻江

２０１２年　７　月　３　日

课题名称：smart composer

【摘要】

电子知音是一个根据音乐自动生成五线谱或简谱的嵌入式设备。音乐的灵感总是伴随着人们无意间的口哨或哼唱消失，哪怕只有1分钟，也会足以使你忘记自己曾经创作了足以流芳百世的音乐。电子知音即能及时记录这些音乐，且相较于那些基于PC机的软件，它更小巧，用户的操作更简单。本作品以TI的MSP430作为主控芯片，以FPGA处理音频信号。

【关键词】: 音乐　谱曲　记录　MSP430　FPGA

Abstract

Electronic music reader is an embedded device, which can automatically generate staff notation or musical notatiaon based on the music. The inspiration of the music is always disappeared with people’s inadvertently whistle or hum. Only one minute, they may forget that they had composed immortal music. Electronic music reader can record the music timely. Compared with PC-based software, it’s more compact ,and the user’s operation is more simple. MSP430 serve as the main chip, and FPGA process the audio signals.

Key words: music　compose　record　MSP430　FPGA

目录

1.创意来源

学过提琴的经历让我热爱音乐，每每听到喜欢的曲子和流行音乐便想尝试能够自己演奏，但是流行音乐的五线谱网上是很难找到的，就算少数能找到的乐谱也需要收费获取。所以非常想要一台机器能够将听到的音乐谱成乐谱供自己演奏。

微软曾经推出一款 songsmith 的软件，可以非常智能的帮助人们创作属于自己的音乐，即使我们并不是音乐方面的专家。这款软件业可以帮助音乐家创作音乐，方便老师教学等等。

我们希望完成一个可以移动的微型化专业谱曲设备。拥有它，你可以随时随地，唱一首歌，那么你的音乐将被记录，可以谱出对应的五线谱，同时可以回放你刚才的录音。

2.立项依据

2.1 项目意义

作曲家在即兴创作音乐时一般是在自己弹奏音乐后记下旋律然后写下乐谱谱；绝大多数音乐爱好者是无法自行听出音乐五线谱的，而绝大多数的古典乐谱需要购买乐谱书籍或者在网上购买电子乐谱，多数流行音乐在网上无法找到乐谱或只能找到简谱。这些都预示着，“自动谱曲型”设备，已经成为一种需要。

2.2 现状分析

而现在市场上多数谱曲设备都要求用户进行按键控制，过程机械繁琐，无法实现自动谱曲的功能，不适合初学者和几乎没有音乐基础的多数人。

Microsoft 市场上的谱曲软件几乎都是基于 PC 机的，例如微软的 Song Smith，用户一定要面对一个 PC 机唱出歌曲，软件才能伴奏，虽然它的音乐效果非常好，但是非常不具备便携性，无法实现随时随地享受音乐创作的快乐。

3. 设计目标

3.1 基本要求

- 能够接收音频信号，将音乐对应的乐谱（简谱或五线谱）显示
- 进行谱曲操作时仅考虑音频信号的幅度特征，采用固定的节奏
- 迷你卡拉 OK 功能，实现音乐的有效回放功能

3.2 扩展部分

- 对核心算法进行优化；
- 实现滚动屏幕的功能；
- 由用户进行模式选择；
- 可以自主选择采样率，进行切换；
- 可以在录音模式以外的任何时刻进行音量调节；

4. 团队组成与任务分工

为了最大程度实现并行性，我们按照模块来划分任务。不同模块之间首先需要将相互之间的接口定义好，定义完成以后不同模块就能够相对地独立工作了。我们所划分的四大模块为：频谱分析模块，VGA 显示模块，音频录音存储模块，音频回放模块，launchpad 模块。

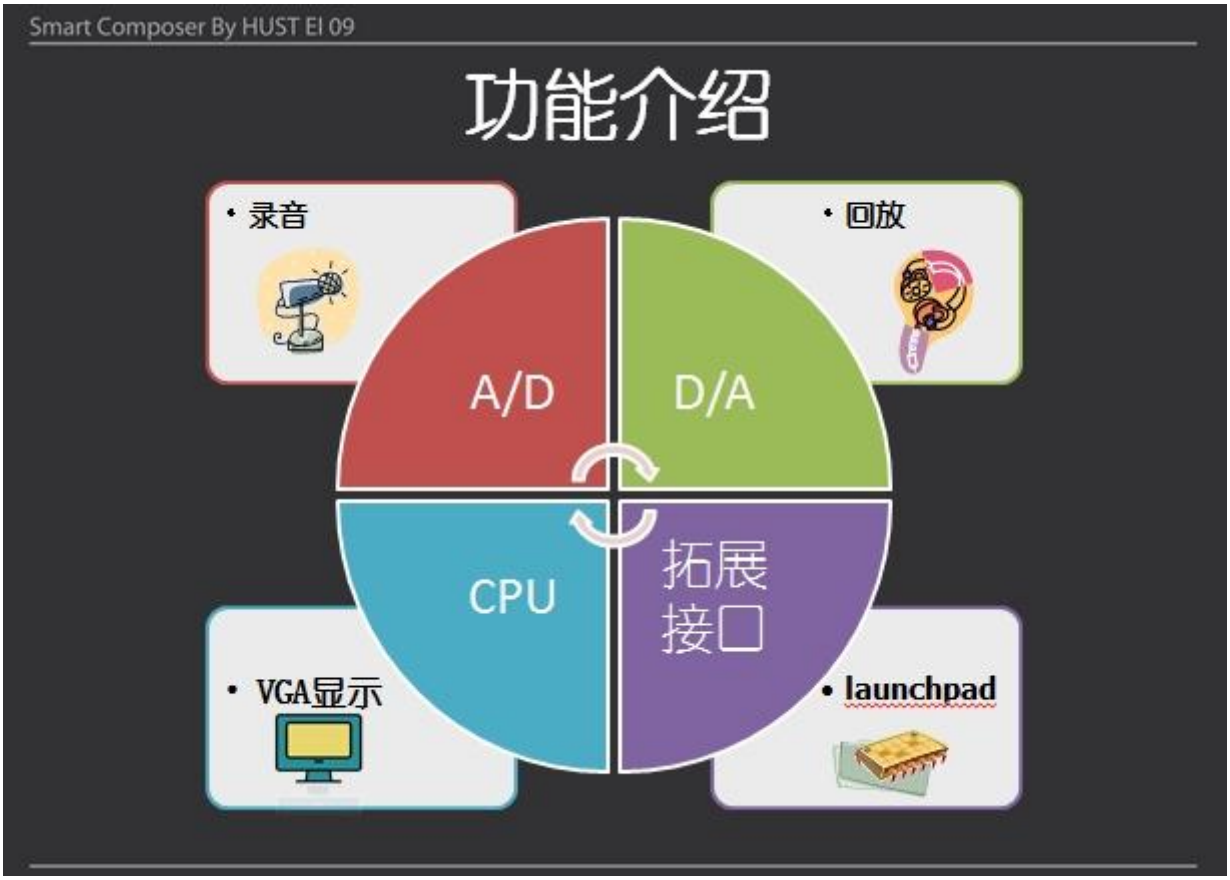
频谱分析模块是我们整个项目的核心，难度系数最高。分成三个子模块：读数据模块，频谱分析模块，频率转换五线谱模块。读数据模块由张曼负责提供思想与程序初稿：频谱分析模块由刘欣欣负责需求分析选择合适的算法，并且进行实现；频率转换五线谱模块由韩东珉负责提供思想与算法。刘欣欣同学负责三个模块程序的编写与调试最终集成为一个模块放在 SOPC 与 NIOS 中实现，实现语言为 C 语言，张曼同学也参与了程序的调试工作。

VGA 显示模块是由韩东珉同学负责，主要是编写 cursor 模块的程序，背景图的制作，以及 quartus 部分的顶层设计。

音频的录音存储模块由张曼同学负责。音频的回放模块由刘欣欣同学负责。

Launchpad 模块是由刘欣欣和韩东珉合作完成。刘欣欣负责熟悉 launchpad 的开发环境，并且对其进行编程控制；韩东珉负责外围电路的焊接以及 launchpad 输出信号与 DE2 的通信工作。

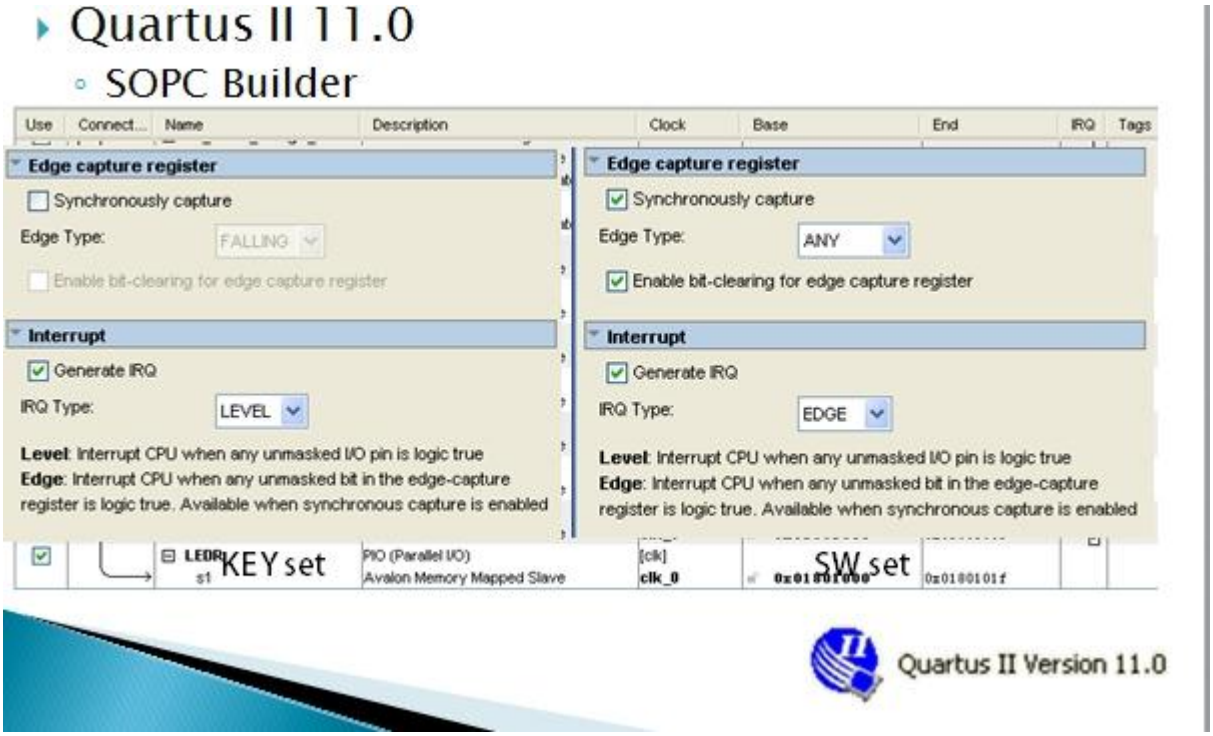
5. 总体设计方案

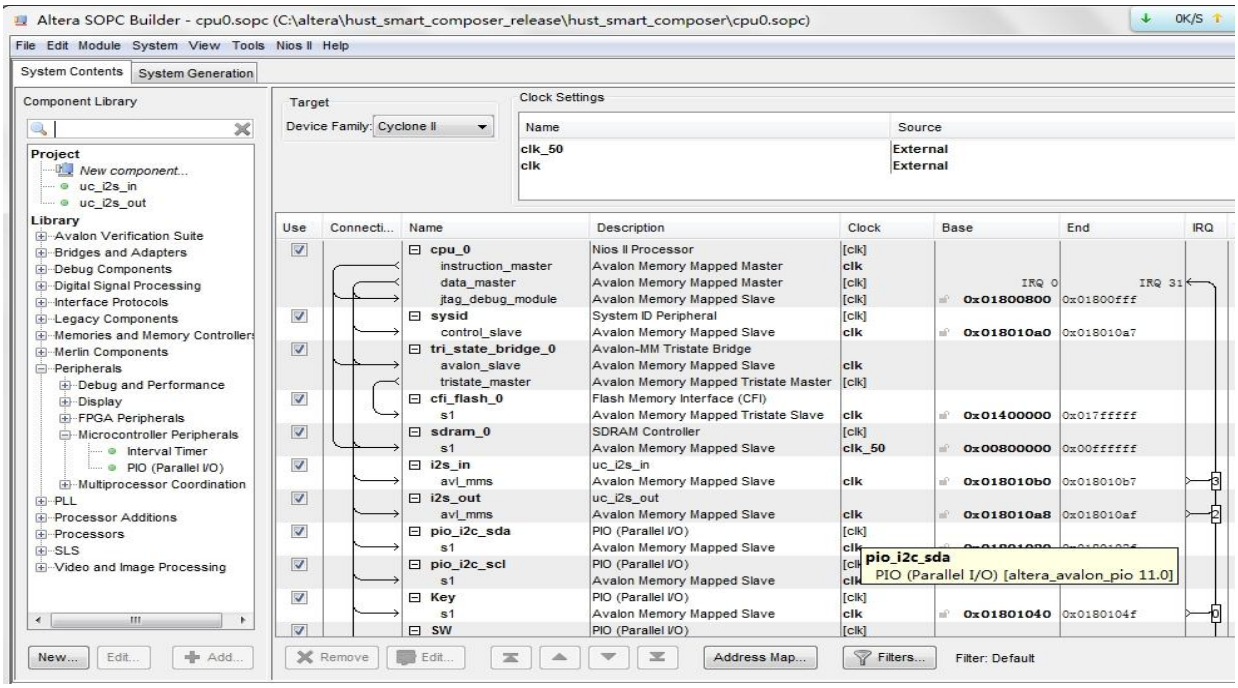


5.1 硬件电路设计

开发平台

- DE2开发平台
- TI公司的launchpad



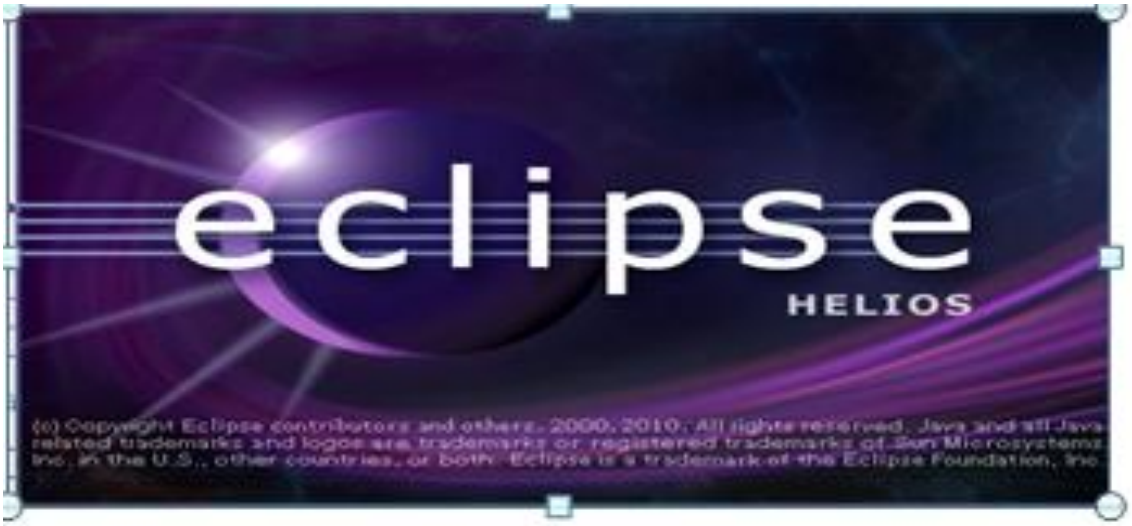


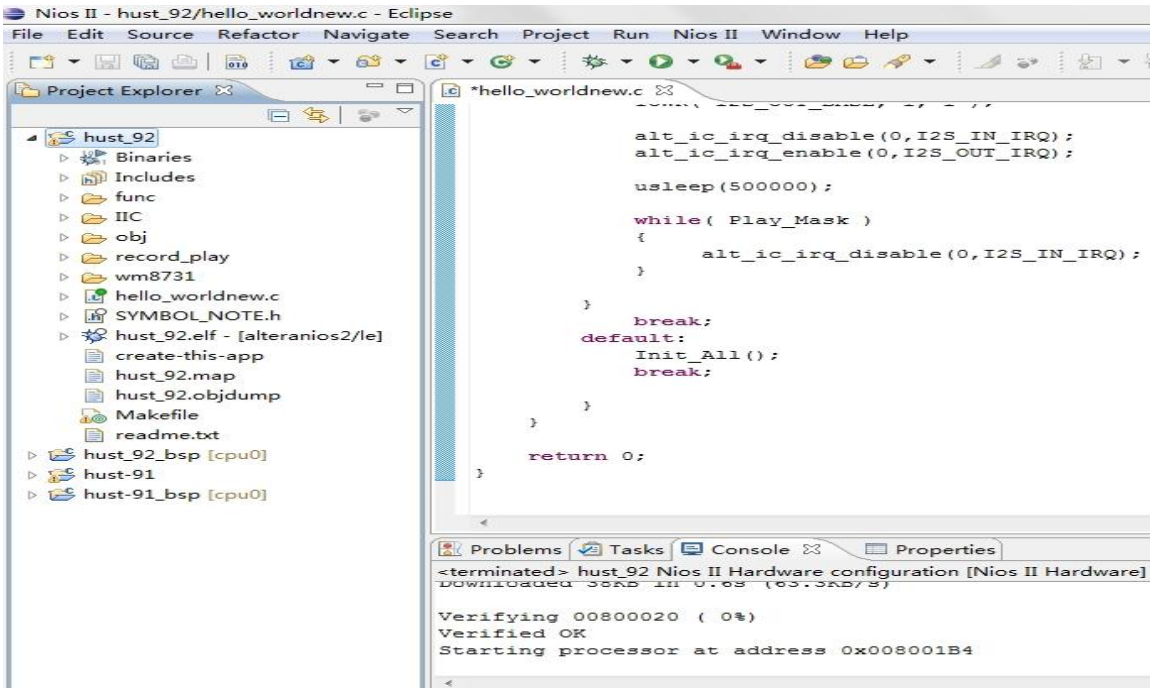
5.2软件整体设计

开发环境1：

Nios II EDS 11.0

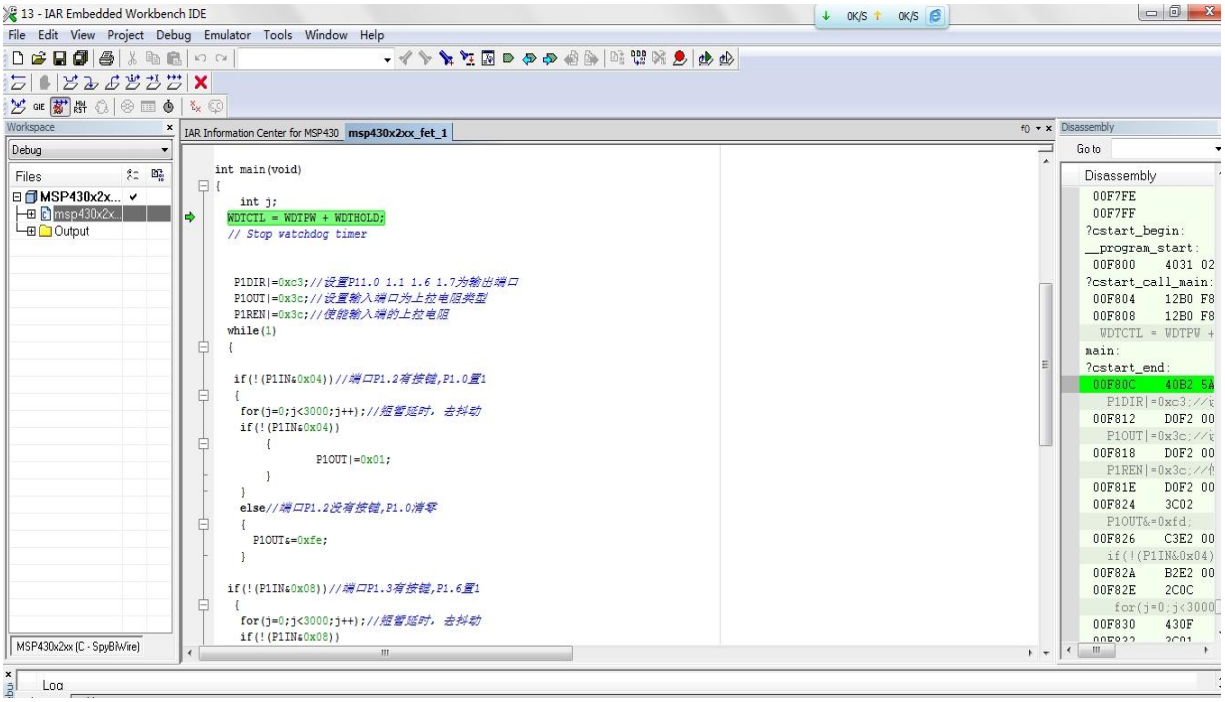
Eclipse C/C++ IDE for Nios II





开发环境2：

IarIdePm



5.3主函数源码

```
#include <io.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/alt_irq.h>
#include "func/Func.h"
#include "wm8731/wm8731.h"
#include "record_play/record_play.h"
#include "SYMBOL_NOTE.h"

extern BYTE Mode;
extern BYTE Side_Mask;
extern BYTE Dire_Mask;
extern BYTE Reco_Mask;
extern BYTE Play_Mask;
UINT16 status;
extern UINT32 Data[Data_Length];
//status = alt_irq_disable_all();
//alt_irq_enable_all( status );
int main()
{

    // IOWR(LEDG_BASE,0,0x8);
    // 初始化
    Init_All();
    /*
     *把输出端口缓冲区空的中断禁掉，因为刚开始的时候要用到direc模式播放
     */
    alt_ic_irq_disable(0,I2S_OUT_IRQ);

    while( !WM8731_Init() );

    // Power up
    while( !WM8731_SetPowerDown(1, 1, 1, 1, 1, 1, 1, 1) );
    // 设置采样率
    while( !WM8731_SetSampling(0, 0, 0x8, 1, 0) );
    // 设置接口
    while( !WM8731_SetInterface(0, 1, 0, 0, 0x0, 0x2) );
    // 激活接口
    while( !WM8731_IfSmpActive() );
```

```
// 设置二级播放参数和音量等
while( !WM8731_SetHeadPhone(1, 1, 120, 120) );

printf("Initialize complete...\n");

while(1)
{
switch(Mode)
{
case 0:
{
        printf("Software_loop Mode...\n");

    // software_loop模式
    while( !WM8731_SetPowerDown(0, 1, 1, 0, 0, 0, 0, 1) );
    while( !WM8731_SetAnalogPath(0, 0, 1, 0, 1, 0, 1) );
    while( !WM8731_SetSampling(0, 0, 0x8, 1, 0) );
    while( !WM8731_SetDigitalPath(1, 0, 0x2, 0) );

    IOWR( I2S_IN_BASE, 1, 1 );


    alt_ic_irq_enable(0,I2S_IN_IRQ);
    alt_ic_irq_disable(0,I2S_OUT_IRQ);

    usleep(500000);

    while( Dire_Mask )
    {
    }

}
    break;
case 1:
{
    printf("Side_tone Mode...\n");

    // PERFORM模式
    while( !WM8731_SetPowerDown(0, 1, 1, 0, 1, 1, 0, 1) );
    while( !WM8731_SetHeadPhone(1, 1, 127, 127) );
```

```
   while( !WM8731_SetAnalogPath(0, 1, 0, 0, 0, 0, 1) );


   // 可以直接说话....
   alt_ic_irq_disable(0,I2S_OUT_IRQ);
  alt_ic_irq_disable(0,I2S_IN_IRQ);
   static UINT32 i=0;
   UINT32 n = 0;
   UINT32 j=0;//中间循环变量
   UINT32 k=0;
   UINT16 PLData[BUFLEN ],PPLData[BUFLEN ],freq;
   BYTE note_valuedata,note_value_tempdata=0,
     symbol_valuedata,note_numdata=0,count=0;

   UINT16 kk=0,jj=0,max_index,cc=BUFLEN/10;//频谱分析定义变量
   UINT32 R[BUFLEN],max,temp;

    while(Side_Mask)
       {
 while(Side_Mask)
{
   n = 0;
  do
 {

   PLData[n]=Data[i]&0xffff;//取一个声道的数据低16位即可


    //IOWR(LEDR_BASE,0,Data[i]&0xe);//调试程序时观察
   i++;
   n++;
 , }
while((n < BUFLEN )&&(Data[i]!=0));
 while(j<BUFLEN)//帧偏移是BUFLEN/2;
{
   PPLData[j]=PLData[j];
   j++;
}
j =0;
```

```
while(k<BUFLEN/2)
{

  PLData[BUFLEN/2+k]=PLData[k];

  PLData[k]=PPLData[k+BUFLEN/2];
  k++;
}
k=0;

for (kk=0;kk<BUFLEN;kk++)//计算一帧数据的自相关

{

    R[kk]=0;//首先将自相关初始化为0


    for(jj=0;jj<BUFLEN-kk;jj++)
    {

       R[kk]+=PLData[jj]*PLData[jj+kk];
    }
 }
      max=R[cc];max_index=cc;

       //从第BUFLEN/10个自相关开始检测自相关的峰值

       while(cc<BUFLEN)
        {
             if(max<R[cc])
              {

                 temp=max;max=R[cc];R[cc]=temp;
                 max_index=cc;
               }
                 cc++;
         }

 cc=BUFLEN/10;//重新初始化cc，非常重要哦


  freq=(2*50*BUFLEN)/max_index;//计算基音频率，单位是Hz

            // IOWR(LEDR_BASE,0,freq);调试程序使用
```

```
    IOWR(LEDR_BASE,0,max_index);//红色LED显示max_index的值

     //自定义端口写数据
     IOWR(NUM_BASE,0,note_numdata);
      IOWR(NOTE_VALUE_BASE,0,note_valuedata);
      note_value_tempdata=note_valuedata;
      IOWR(SYMBOL_VALUE_BASE,0,symbol_valuedata);
       count=0;
   }

           //  usleep(1000); //IOWR(LEDR_BASE,0,note_numdata);
            // IOWR(LEDG_BASE,0,note_valuedata&0xf);
}

          alt_ic_irq_disable(0,I2S_OUT_IRQ);

}
}
       break;
case 2:
   {
       printf("Record Mode...\n");

       // Record模式
       while( !WM8731_SetPowerDown(0, 1, 1, 0, 0, 0, 0, 1) );
       while( !WM8731_SetAnalogPath(0, 0, 1, 0, 1, 0, 1) );
      // while( !WM8731_SetAnalogPath(0, 0, 1, 0, 0, 0, 1) );
       while( !WM8731_SetSampling(0, 0, 0x8, 1, 0) );
       while( !WM8731_SetDigitalPath(1, 0, 0x2, 0) );

       // 使能输入端的中断，很重要哇
       IOWR( I2S_IN_BASE, 1, 1 );


       //说话，但听不到声音，因为在存储数据
       alt_ic_irq_disable(0,I2S_OUT_IRQ);
       alt_ic_irq_enable(0,I2S_IN_IRQ);

       usleep(500000);
```

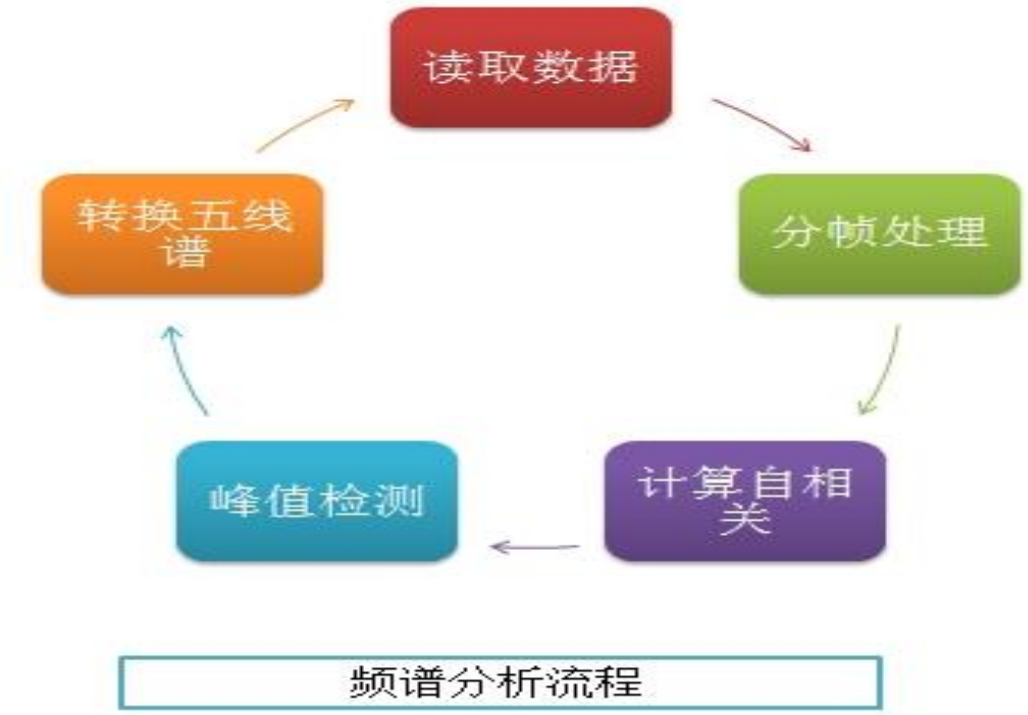
```
        while(Reco_Mask)
        {
        alt_ic_irq_disable(0,I2S_OUT_IRQ);
        }

    }
        break;
    case 3:
    {
        printf("Play Mode...\n");

        // Play模式
        while( !WM8731_SetPowerDown(0, 1, 1, 0, 0, 0, 0, 1) );
        while( !WM8731_SetAnalogPath(0, 0, 1, 0, 1, 0, 1) );
        // 设置DA 8KHz 采样率
        while( !WM8731_SetSampling(0, 0, 0x8, 1, 0) );
        while( !WM8731_SetDigitalPath(1, 0, 0x2, 0) );

        //这句话使能输出端的中断，坑死我 了
        IOWR( I2S_OUT_BASE, 1, 1 );

            alt_ic_irq_disable(0,I2S_IN_IRQ);
            alt_ic_irq_enable(0,I2S_OUT_IRQ);

            usleep(500000);

        while( Play_Mask )
        {
         alt_ic_irq_disable(0,I2S_IN_IRQ);
        }

    }
            break;
    default:
        Init_All();
        break;
    }
    }
    return 0;
}
```

6. 模块设计与分析

6.1 频谱分析模块

6.1.1 综述

频谱分析模块是我们整个项目的核心，难度系数最高。分成三个子模块：读数据模块，频谱分析模块，频率转换五线谱模块。读数据模块由张曼负责提供思想与程序初稿：频谱分析模块由刘欣欣负责需求分析选择合适的算法，并且进行实现；频率转换五线谱模块由韩东珉负责提供思想与算法。刘欣欣同学负责三个模块程序的编写与调试最终集成为一个模块放在SOPC与NIOS中实现，实现语言为C 语言，张曼同学也参与了程序的调试工作。



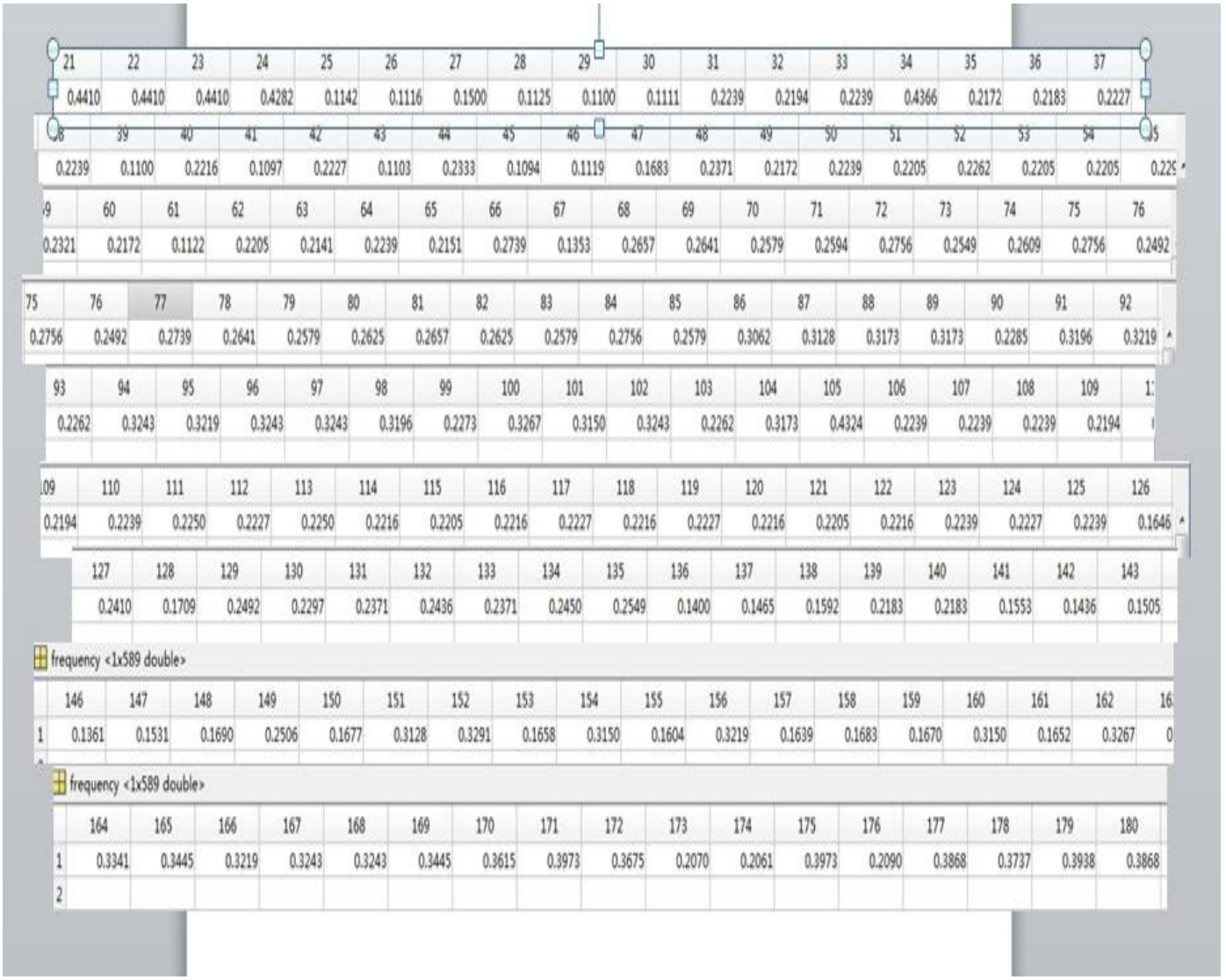
频谱分析流程

6.1.2频谱分析仿真源码

```
[filename,pathname]=uigetfile('music.wav','请选择语音文件:');
[sound,fs]=wavread([pathname,filename]);
lefty =sound(:,1);
%获取声音数据的长度，按字节算
wavesize=length(sound);

%===================基音周期滤波=====================%
%2Khz低通hamming
b =[ 0.0054     0.0074     0.0130     0.0221     0.0341     0.0481...
     0.0628     0.0764     0.0874     0.0947     0.0972     0.0947...
    0.0874     0.0764     0.0628     0.0481     0.0341     0.0221...
     0.0130     0.0074     0.0054];
waveoutpitch = filter(b,1,lefty);

%===================最大幅度=====================%
N=2*fs/100
between=floor(N/2);
FrameNumber =floor(wavesize/between)-1;
Tlefty=waveoutpitch ;
 Axiaobo=zeros(wavesize,1);
% 整个时域上的基音周期轨迹
pitch =zeros(1,FrameNumber);
% 一帧中的自相关函数
R1=zeros(N,1);
for ii=1:FrameNumber
        %
        startPositon=(between)*(ii-1)+1;
    %       startPositon=(between)*(ii-1)+1;
        temp =Tlefty(startPositon:startPositon+N-1);
        %加窗
        temp = temp.*hamming(N);

    %=================计算自相关==================
        R1=zeros(N,1);
        for kk=1:N
              for jj=1:N-kk
                R1(kk)=R1(kk)+temp(jj)*temp(jj+kk);
```

```
                end
            end
        % % % %  计算一帧基音周期
            Rtemp1=R1(100:N);
            R1max=max(Rtemp1);
            j=find(Rtemp1==max(R1max));
            lie=min(j)+100-1;
            %基音周期
             pitch(ii)=lie/44.1;
             %基音频率
             frequency(ii)=1/pitch(ii)
end
```

6.1.3频谱分析仿真部分结果



6.1.4软件模块主要代码分析

```
#include <io.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/alt_irq.h>
#include "func/Func.h"
#include "wm8731/wm8731.h"
#include "record_play/record_play.h"
#include "SYMBOL_NOTE.h"

extern BYTE Mode;
extern BYTE Side_Mask;
extern BYTE Dire_Mask;
extern BYTE Reco_Mask;
extern BYTE Play_Mask;
UINT16 status;
extern UINT32 Data[Data_Length];
//status = alt_irq_disable_all();
//alt_irq_enable_all( status );
int main()
{

    // IOWR(LEDG_BASE,0,0x8);
    // 初始化
    Init_All();
    /*
     *把输出端口缓冲区空的中断禁掉，因为刚开始的时候要用到direc模式播放
     */
    alt_ic_irq_disable(0,I2S_OUT_IRQ);

    while( !WM8731_Init() );

    // Power up
    while( !WM8731_SetPowerDown(1, 1, 1, 1, 1, 1, 1, 1) );
    // 设置采样率
    while( !WM8731_SetSampling(0, 0, 0x8, 1, 0) );
    // 设置接口
    while( !WM8731_SetInterface(0, 1, 0, 0, 0x0, 0x2) );
    // 激活接口
```

```
while( !WM8731_IfSmpActive() );

// 设置二级播放参数和音量等
while( !WM8731_SetHeadPhone(1, 1, 120, 120) );

printf("Initialize complete...\n");

while(1)
{
switch(Mode)
{
case 0:
{
        printf("Software_loop Mode...\n");

    // software_loop模式
    while( !WM8731_SetPowerDown(0, 1, 1, 0, 0, 0, 0, 1) );
    while( !WM8731_SetAnalogPath(0, 0, 1, 0, 1, 0, 1) );
    while( !WM8731_SetSampling(0, 0, 0x8, 1, 0) );
    while( !WM8731_SetDigitalPath(1, 0, 0x2, 0) );

    IOWR( I2S_IN_BASE, 1, 1 );


    alt_ic_irq_enable(0,I2S_IN_IRQ);
    alt_ic_irq_disable(0,I2S_OUT_IRQ);

    usleep(500000);

    while( Dire_Mask )
    {
    }

}
    break;
case 1:
{
    printf("Side_tone Mode...\n");

    // PERFORM模式
    while( !WM8731_SetPowerDown(0, 1, 1, 0, 1, 1, 0, 1) );
```

```
   while( !WM8731_SetHeadPhone(1, 1, 127, 127) );
   while( !WM8731_SetAnalogPath(0, 1, 0, 0, 0, 0, 1) );


   // 可以直接说话....
   alt_ic_irq_disable(0,I2S_OUT_IRQ);
  alt_ic_irq_disable(0,I2S_IN_IRQ);
   static UINT32 i=0;
   UINT32 n = 0;
   UINT32 j=0;//中间循环变量
   UINT32 k=0;
   UINT16 PLData[BUFLEN ],PPLData[BUFLEN ],freq;
   BYTE note_valuedata,note_value_tempdata=0,
     symbol_valuedata,note_numdata=0,count=0;

   UINT16 kk=0,jj=0,max_index,cc=BUFLEN/10;//频谱分析定义变量
   UINT32 R[BUFLEN],max,temp;

    while(Side_Mask)
       {
 while(Side_Mask)
{
   n = 0;
  do
 {


   PLData[n]=Data[i]&0xffff;//取一个声道的数据低16位即可


    //IOWR(LEDR_BASE,0,Data[i]&0xe);//调试程序时观察
   i++;
   n++;
 , }
while((n < BUFLEN )&&(Data[i]!=0));

 while(j<BUFLEN)//帧偏移是BUFLEN/2;

{
   PPLData[j]=PLData[j];
   j++;
}
```

```
j =0;
while(k<BUFLEN/2)
{

  PLData[BUFLEN/2+k]=PLData[k];

  PLData[k]=PPLData[k+BUFLEN/2];
  k++;
}
k=0;
for (kk=0;kk<BUFLEN;kk++)//计算一帧数据的自相关
{
    R[kk]=0;//首先将自相关初始化为0

    for(jj=0;jj<BUFLEN-kk;jj++)
                                          {

    R[kk]+=PLData[jj]*PLData[jj+kk];
                                          }
                                    }
      max=R[cc];max_index=cc;
       //从第BUFLEN/10个自相关开始检测自相关的峰值
       while(cc<BUFLEN)
        {
             if(max<R[cc])
              {

                 temp=max;max=R[cc];R[cc]=temp;
                 max_index=cc;
               }
                 cc++;
         }
 cc=BUFLEN/10;//重新初始化cc，非常重要哦

  freq=(2*50*BUFLEN)/max_index;//计算基音频率，单位是Hz
```

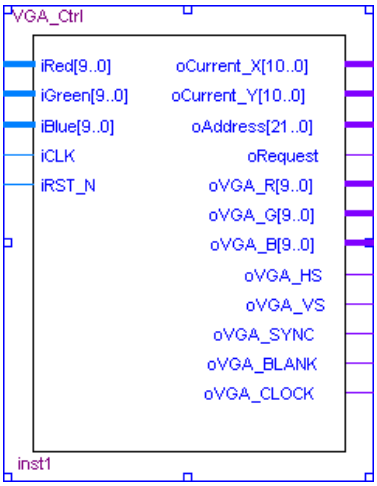
```
// IOWR(LEDR_BASE,0,freq);调试程序使用

IOWR(LEDR_BASE,0,max_index);//红色LED显示max_index的值

//自定义端口写数据
IOWR(NUM_BASE,0,note_numdata);
 IOWR(NOTE_VALUE_BASE,0,note_valuedata);
 note_value_tempdata=note_valuedata;
 IOWR(SYMBOL_VALUE_BASE,0,symbol_valuedata);
```

6.2. VGA显示模块

6.2.1 概述

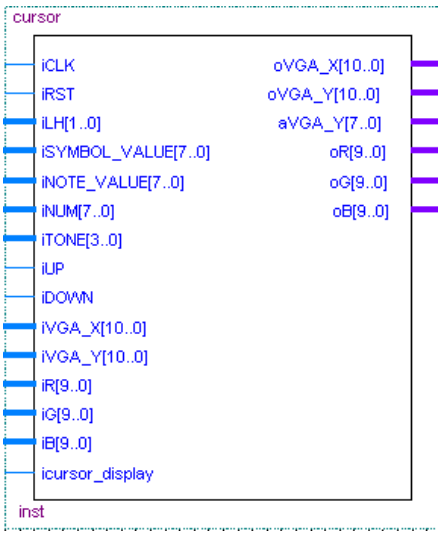
我们的显示部分是用硬件表述的，分为 3 个模块，分别是 VGA_Ctrl，cursor，m4kmem。

VGA_Ctrl：这是一个驱动模块，这个模块有很多现成的可以使用，每一个的写法可能都大不一样，但实现的功能相同。它来控制显示器的 VGA 接口的各种时序，也就是它来决定什么时候分别把最终的 R,G,B 信号送给显示器，它的输入端就是其他模块最终想显示的 R,G,B 信息，而它的输出端中，除了要直接（其实也不是“直接”，只不过 DA 部分底层的硬件做好了）送到 VGA 接口的信号之外，还有两个有用的返回值，就是扫描坐标 oCurrent_X,oCurrent_Y，这个表示当前 VGA 将要扫描的下一个像素的横纵坐标，这个坐标在后面会多次用到（虽然驱动模块只需要学会调用就行了，这一部分还是建议大致了解一下 VGA 的基本原理，网上资料很多在此不赘述）。



m4kmem：是一个 M4K 存储器，它负责存放显示背景，由于网上关于 M4K 的资料非常少,因此关于设置的内容都是我自己猜测的，若有错误请见谅。因为 M4K 的容量很有限，因此我们只能存放一个二值图像的信息，一个地址存放一个 bit，这一个 bit 的值代表了这个二值图像上某个像素的值，“1”代表黑色，即有显示，“0”代表白色，即空白部分。而对应的地址则是背景的横纵坐标生成出的，从最左上角地址为 1，逐行扫描到最右下角地址为最高。比如，想存储一个 640*480 的二值图像到 M4K，则需要 307200 个地址单元，相应的地址为[0..307199]，而我们的图像是 BMP 格式的，这个格式是一个矩阵，存放着每一个像素的信息，这个 BMP 文件是不能用地址方式读取的，因此需要将这个矩阵信息变换成地址与值对应的信息，这个过程叫做存储器的初始化，而地址与值对应的信息就是一个存储器初始化文件，文件后缀是.mif 或者.hex，我们用 matlab 来生成这个.mif 文件，代码在工程里，

是非常简单的几行代码，要注意的是，不要忘记要根据自己使用背景的大小来修改代码里的参数，用 matlab 把图像打开存入工作区后运行代码即可。



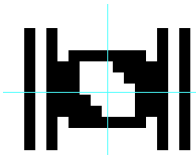
cursor： 这个模块是显示部分需要自己主要编写的模块，它控制了显示内容，也就是哪个坐标该显示什么内容，然后把这个最终要显示的 R,G,B 值发送给 VGA_Ctrl。它的输入端包括和其它模块的接口，例如 iLH(升降号控制符)，iSYMBOL_VALUE(音符节奏、种类)等，还包括从 VGA_Ctrl 返送回来的扫描坐标 iVGA_X，iVGA_Y，这个坐标告诉 cursor，显示器上正准备显示哪个像素，然后由 cursor 通过判断，决定这个像素显示什么，然后向 M4K 发送一组坐标 oVGA_X，oVGA_Y，这组坐标生成一个地址，这个地址告诉 M4K“你现在要给我哪个像素的 bit 值”，再然后，M4K 就把这个 bit 值发送给 cursor，由 cursor 转发给 VGA_Ctrl（至于这里为什么要先发给 cursor 再转发给 VGA_Ctrl，是为了调试方便，在测试模块时可以方便控制这个显示信息，在实际运行中是可以把这个 bit 值经过给 R,G,B 赋值直接给 VGA_Ctrl 的，如果你不知道这句话在说什么没关系，直接忽略，最后就明白了），告诉它“这就是你要显示的内容”。这就是整个显示背景的基本思路，cursor 模块的作用可见一斑，它整个显示部分的主要控制者。

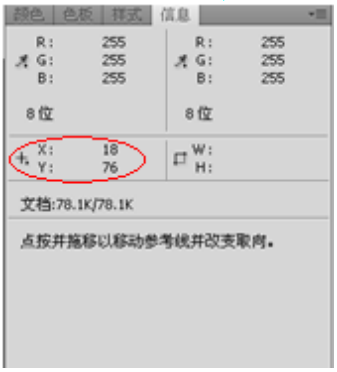
6.2.2 详解

VGA_Ctrl 模块该解释的已经在上面的概述里说清了，这里不多说。关于 cursor 与 m4kmem 的部分还有很多地方需要解释，下面来详细说明一下这两个模块。先说 cursor 后说 m4kmem。

cursor：

在这个模块中，定义了很多常数。例如 dnote_up_x，frest_up_y 等等，这些常数包括了每一种音符的 template 的横纵坐标、第一行五线谱的每一根线和每一个线与线之间间隔的纵坐标、准备放置音符的起始横坐标、每一行五线谱之间的间隔以及每一行乐谱中放置音符的横坐标间隔。这需要用 photoshop（其他能够显示坐标的软件也行）打开背景图片 sheet teste.bmp，按 F8 出现“信息栏”，把光标放在音符的中心，可从信息栏读出该点的坐标。和颜色等信息，想获得精确的坐标信息必须把图片放到足够大。这些常数就是这样一个一个测量出来的。另外，如果需要在图片上进行修改，这个坐标常数也要做相应的变化。





下面来说 cursor 的整体设计思路，代码的核心其实围绕着三个坐标、两个光标、几个输入量和来写。

先来说说三个坐标，分别是 iVGA_X，cur_X，cur_pointer_x。

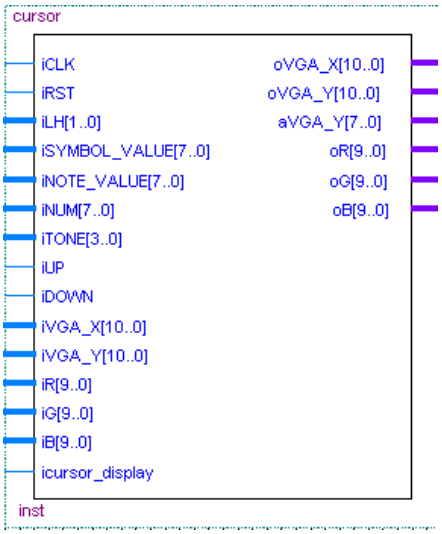
1. iVGA_X 为扫描坐标，就是 VGA_Ctrl 输出的即将要显示的坐标值；

2. cur_X 为光标坐标，表示光标的当前坐标；
3. cur_pointer_x 为 template 坐标，它指向光标需要显示的地方。

```
assign oVGA_X = (arrow1 ? (cur_pointer_x1 + iVGA_X - cur_X1) :
                 arrow  ? (cur_pointer_x + iVGA_X - cur_X) : iVGA_X);
assign oVGA_Y = (arrow1 ? (cur_pointer_y1 + iVGA_Y + rVGA_Y - cur_Y1) :
                 arrow  ? (cur_pointer_y + iVGA_Y + rVGA_Y - cur_Y) : iVGA_Y + rVGA_Y);
```

光标的作用是区别于背景显示，因为在显示的过程中有动态需要添加的显示内容，这个就由光标来实现，它是一块区域在代码中规定了，当扫描坐标落在光标以内的时候，即 arrow=1 时就不显示背景而显示其他内容，在这个工程中，就是显示 template 音符。换句话说，当 arrow=1 时，就把输出的 oVGA_X，oVGA_Y 的坐标由背景坐标（扫描坐标）iVGA_X，iVGA_Y 变成 template 音符的坐标 cur_pointer_x + iVGA_X - cur_X，cur_pointer_y + iVGA_Y + rVGA_Y - cur_Y（rVGA_Y 为滚动屏幕定义的量后面说到）。

几个输入量是 iNUM（输入音符的序号，为 0 时输入 TONE）、iTONE（乐谱的调式）、iSYMBOL_VALUE（音符的节奏，即 template 的种类）、iNOTE_VALUE（音符的音高）、iLH（音符的临时升降号）。我们得清楚这每一个输入量控制的音符显示的哪些特性。



1. iNUM，定义这个接口是为了方便确定音符显示横坐标位置，总共有 5 行乐谱，如果测量所有的五线谱坐标并且定义成常量工作量太大也没有必要。于是我就把每一个音符之间的间隔固定，这样每一行显示的音符数就固定了（28 个），然后把每一行五线谱的纵坐标间隔设置为定值，这样可以通过这是第几个音符来判断它应该显示的横坐标和所在五线谱的行数。另外，当 iNUM=0 时，接受 iTONE 的值，接口的定义值在“SYMBOL_NOTE.h”定义好了。
2. iTONE，调式信息，D 调需要画两个升号，分了两步完成，每一步之间设置一个延时。
3. iSYMBOL_VALUE，音符的 template 种类定义，它决定音符为全音符，二分音符或是四分音符等，这个输入决定的是 template 的坐标，也就是 cur_pointer_x 与 cur_pointer_y。
4. iNOTE_VALUE，音符的音高信息，它与 iNUM 一起决定音符的纵坐标。

```
note_x_num <= iNUM % 28 - 1;/
note_y_num <= iNUM / 28;
cur_X <= note_x_num * 20 + n;
case (iNOTE_VALUE)
    `LOW_C:     begin
                cur_Y <= d_01_line + note_y_num * gap_y;
                end
```

5. iLH,临时升降号控制符,它控制 cur_pointer_x1,cur_pointer_y1,cur_X1,cur_Y1，原理和之前的一样，只是 cur_pointer_x1，cur_pointer_y1 可以由 cur_pointer_x，cur_pointer_y 直接决定。

再来说一下滚屏功能的实现，这个功能比较简单。

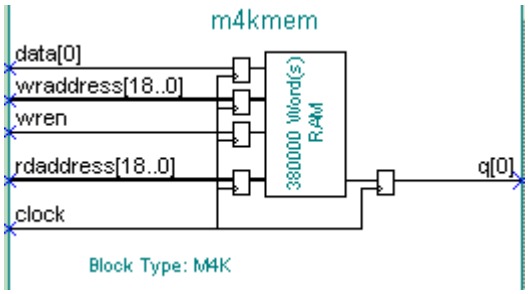
```
always @(posedge iCLK)
begin
    if (counter1 <= 19'd390000)
        counter1 <= counter1 + 1;
    else
        begin
        counter1 <= 19'd0;
        if (!iUP && iDOWN && rVGA_Y < 110)
            rVGA_Y <= rVGA_Y + 1;
        else if (!iDOWN && iUP &&   rVGA_Y > 0)
            rVGA_Y <= rVGA_Y - 1;
        end
end
```

定义一个变量 rVGA_Y，r 取 relative 的意思。产生一个合适延时，按下相应的滚屏开关后，rVGA_Y 以合适的速度增长，增长到 670-480=110 后停止。相反，向上滚屏就是 rVGA_Y 的减计数过程。然后记得在与显示有关的所有纵坐标上加上这个 rVGA_Y 变量即可。

最后总结一下 cursor 整个流程，首先判断 iNUM 的值，为 0 则判断 iTONE 的值，根据 iTONE 分两步画上调式升降号。iNUM 不为 0 时判断 iSYMBOL_VALUE 和 iNOTE_VALUE，根据这三个值来判断 template 坐标与光标坐标的位置并画上音符。

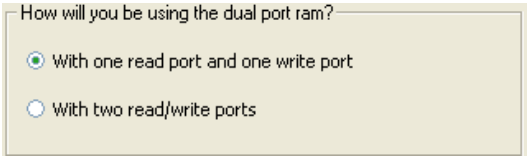
m4kmem：

这个模块是调用 ip 核中的双口 RAM 实现的。之所以用双口，是因为背景的显示，是需要边显示边储存的，因为新显示上去的音符信息只有存进 M4K 才能在光标移走之后依然能显示出来，所以需要有一个口读数据，另一个口写数据。



几个参数的设置：

1. 采用一端输入/输出，其实我们这个工程既

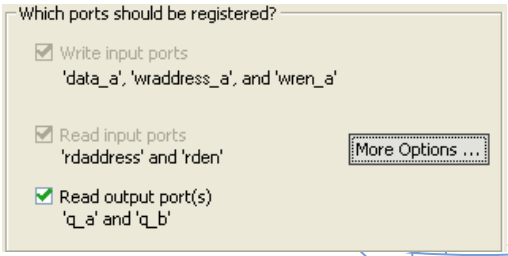


可以双端输入/输出也可以单端，为了方便选择单端；

2. 记忆容量，380000bit；

限于 M4K 的最大容量为 483840bit，并且 NIOS 会占用一部分 M4K，经过一次次编译，最后得到可用的 M4K 大小大概为 400000bit 左右，只能显示 5 行乐谱，因此采用 640*670 作为图片大小，设置 M4K 模块容量为 380000bit。

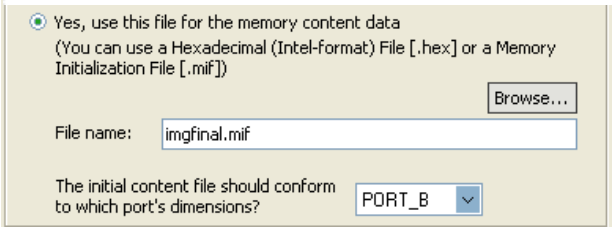
3. 数据位宽，1 位，因为是二值图像，只需输出 0 或 1；
4. 输出端采用锁存器输出；

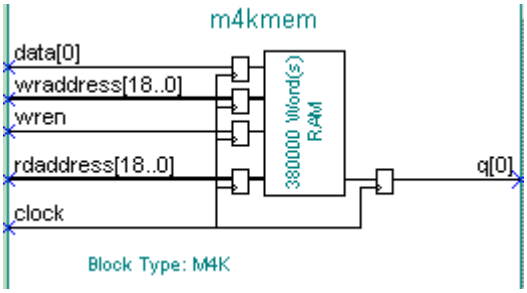


5. 初始化 M4K。

下面我们看 M4K 模块在顶层如何调用，这几行代码很有讲究。

```
assign m4k_addr_b = 640*(VGA_Y + rVGA_Y) + VGA_X;
assign m4k_addr = 640*mVGA_Y + mVGA_X;

m4kmem       u4   (
                  .rdaddress(m4k_addr),
                  .wraddress(m4k_addr_b-2),
                  .clock(OSC_27),
                  .data(mem_out),
                  .wren(mem_out),
                  .q(mem_out)
                  );
```





m4k_addr_b 是扫描坐标生成的地址。m4k_addr 是 cursor 输出的需要读取数据的地址。因此，m4k_addr 接到 M4K 的读取地址端 rdaddress，m4k_addr_b 接到 M4K 的写入地址端 wraddress，至于为什么要减 2 后面说到。先来看输出端的 mem_out，它被直接连到写入使能端 wren 与写入数据端 data 上，这意味着，当 mem_out 为 1 时，写入才被使能，并且将这个“1”写入了 m4k_addr_b-2 也就是即将显示的地址往前推 2 个地址的像素上。这也就意味着，只有当现在扫描坐标上的像素为 1 时，才将这个像素的值存入 M4K 当中变成背景的一部分，那为什么要减 2 呢，因为从当前的 m4k_addr 经过一个锁存器到达 rdaddress 后，输出的 mem_out 又要经过一个锁存器才能到 M4K 的 data 端，那时的 m4k_addr_b 已经是经过两个时钟周期的当前 m4k_addr_b+2 了。因此要把 m4k_addr_b 减 2。

特别提醒：因为显示模块和其它模块是以接口的形式连接的，当中其实还有其它的问题，比如在显示音符的时候，是边显示便储存的，但是如果显示器还没有将当前光标上的音符范围扫描完，光标就已经被下一个音符的信息移动了位置，那么当前的音符的图像是没有被完整存入 M4K 的。因此，为了确保每个音符顺利完成显示与储存，在每个音符到来之间要加上显示器扫描整个屏幕的时间，即 307200 个时钟周期。而在画调式的升降号时，由于要分两步画升降号，所以至少要留两次扫描全屏的时间间隔。

6.2.3 背景图片设置

由于开发平台资源的限制，我们选择了将背景存储在 m4k 中，但是 m4k 容量非常有限，根本不可能存储图片。我们在 matlab 中利用一段代码可以将 bmp 文件转化成 mif 文件，可以直接初始化 m4k.

然后提示一下，那个背景图需要熟练使用 ps,你可能需要精确的知道图片的每一个像素的坐标。

6.2.4 future work

有些工作是我很想做但是没有时间去做的：

1. 限于 M4K 的最大容量问题，我们的图片大小受到了很大的限制，只能显示 5 行乐谱，如果想要显示多一些乐谱，就必须换存储器，可以是 SRAM 也可以是 SDRAM。但要注意的是，我们之所以选择 M4K 是因为它是现成的 IP 核，配置起来非常方便，如果使用 SRAM 或者 SDRAM，可以去查查有没有对应的 IP 核，如果没有就只能找找有没有相应的驱动。否则工作量会有点大。
2. 同样是限于 M4K 的容量问题，我们的乐谱背景是二值的，如果更换存储器，可以尝试多值图像的背景。这样显示的颜色就不限于双色，可以有丰富的颜色显示。由于 SRAM 与 SDRAM 的数据位宽是固定的，不想 M4K 可以随意设置，色彩的宽度与存储器的宽度一定要协调好，这和存储器的驱动很有关系，如何读存储器的数据应该将会是需要解决的一个问题。
3. 在音符的扩充上，做的不够好，因为还存在各种附点音符。可以把它们直接画在背景上变成 template，对音符的种类进行扩充。
4. 这个工程中 template 是作为背景图片显示在屏幕上的，其实完全可以不显示出来，这个实现起来也很简单，我个人想了两个办法：
   a) 直接把这些 template 单独画出来和背景一起变成.mif 文件初始化在存储器中就行了，但自己一定要定义好地址。这样 cursor 在显示光标的内容时就不是输出坐标了二是直接输出存在存储器里的地址进行对存储器调用。这个方法实现起来比较麻烦。
   b) 还是画在背景上，但是不显示出来，就相当于 rVGA_Y 的初值、最小值不为 0 了，为刚好显示出第一行乐谱的合适的纵坐标值。这个方法简单易实现。
5. 乐谱的调式只画在了第一行乐谱的开头，按照乐谱的习惯，应该是每一行乐谱都要标明升降号。因此这也是一个可以改进的地方。
6. 最后有一个大胆的展望：因为被写上去的音符就被存入背景，因此以目前的工程，想撤销或者修改乐谱是无法做到的。如果能在存储器中预留一块区域，用于临时存储每一个音符的信息，并且把这个信息生成一个可读格式储起来，用户可以打开这个文件或者直接对存储器进行修改，便可以达到对乐谱修改的目的。当然，这个实现起来工
   作量是比较大的。

6.3 音频模块

音频模块主要是在参考的语音存储与回放工程的基础上完成。在此向参考工程的原作者以及为我们提供资源的指导老师表示特别感谢。

6.3.1 硬件设计

在 Quartus II 11.0 的 SOPC Builder 中，我们将上述器件的控制器添加进来，在这里注意 WM8731 的 AD 和 DA 对数据进行处理后，向 CPU 传送数据方式有四种工作模式：左对齐，右对齐，IIS，DSP。我们选用其中的 IIS 接口的工作模式，由于

SOPC Builder 中没有 IIS 协议接口，所以我们采用了电工基地老师已经写好的自定义外设 uc_i2s_in 和 uc_i2s_out 的通信模块,它实现了 IIS 的通信协议。配置完成后的截图如下：

注意图中，我们的 i2s_in,i2s_out,还有 SW,KEY 都能产生中断，为后续的软件部分留下中断请求。

在这里生成之后，我们就完成了为软件编写提供各个器件所需的控制方式，亦即驱动文件。在硬件部分，我们还要为该 SOPC 系统配置好时钟(通过添加 Cyclone II 支持的锁相环为处理器，AD，DA，SDRam 等配置所需的时钟)，及其它输入输出接口：我们又为音频芯片与处理器软核之间添加了两个缓冲块。

我们利用了 IIC 的通信的 C 语言实现(IIC.c 和 IIC.h)， WM8731 的寄存器写入函数组(WM8731.c 和 WM8731.h),初始化函数及按键中断处理函数,开关中断函数(Func.c 和 Func.h),WM8731 编码和解码中断(录音和放音)函数(record_play.c 和 record_play.h)。主函数为 hello_worldnew.c.

6.3.2 WM8731 芯片简介：

WM8731 是一款带有集成耳机驱动器的极低功耗、高质量音频解码器，专为便携数字音频应用而设计。

WM8731 的特点：

* 带有集成耳机驱动器的立体声音频编解码器 (50mW on 16W @ 3.3V)
    * 1.42 - 3.6V 数字电源电压
    * 2.7 - 3.6V 模拟电源电压（标准版）
    * 1.8 - 3.6V 模拟电源电压(’L’ 版)
    * 回放模式下功耗< 18mW
    * 100dB 信噪比(‘A’ weighted @ 48kHz)的数模转换器
    * 90dB 信噪比(‘A’ weighted @ 48kHz)的模数转换器
    * 采样率范围：8kHz - 96kHz
    * 主时钟或者从时钟模式
    * USB 时钟模式可以从 USB 时钟直接生成一般 MP3 的所有采样率(incl.

441.kHz)

* 输出音量和静音控制
* 麦克风输入和带有侧音混频器的驻极体偏压
* 可选择的模数转换器(ADC)高通滤波器
* 2 线或 3 线微处理器(MPU)串行控制接口
* 可编程音频数据接口模式
* 28 接脚 SSOP 封装

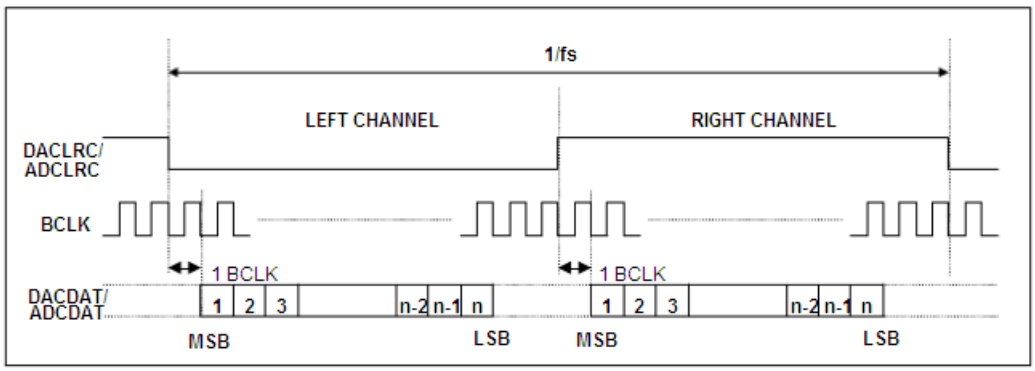


图26 I2S模式

WM8731 有四种音频接口模式：右对齐、左对齐、I2S、DSP。我们的项目中采用的是 I2S 模式。

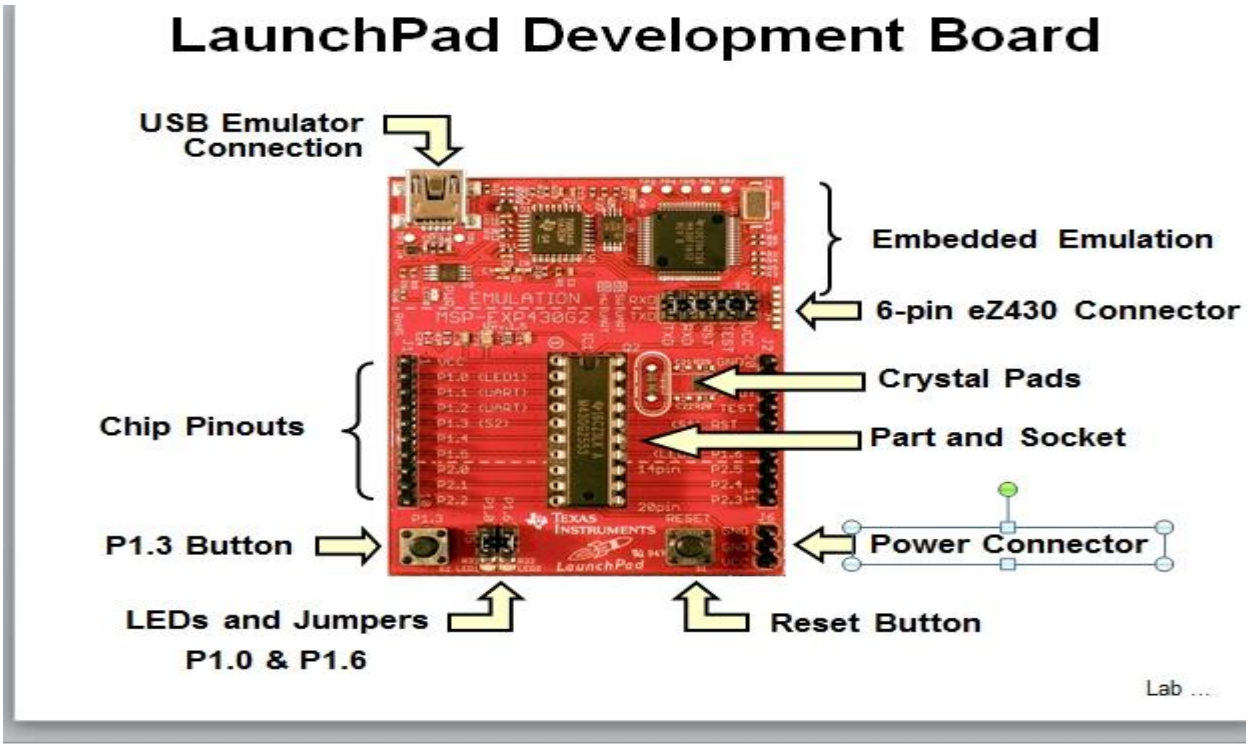
上图表示在 I2S 模式下，采样频率为 fs 时，传输数据的时序图。数据为串行传输，每周期中，每前半周期传输左声道数据，后半周期传输右声道数据。数据的高位 MSB 在时钟第二个有效边缘开始传输。

6.4 Launchpad 模块

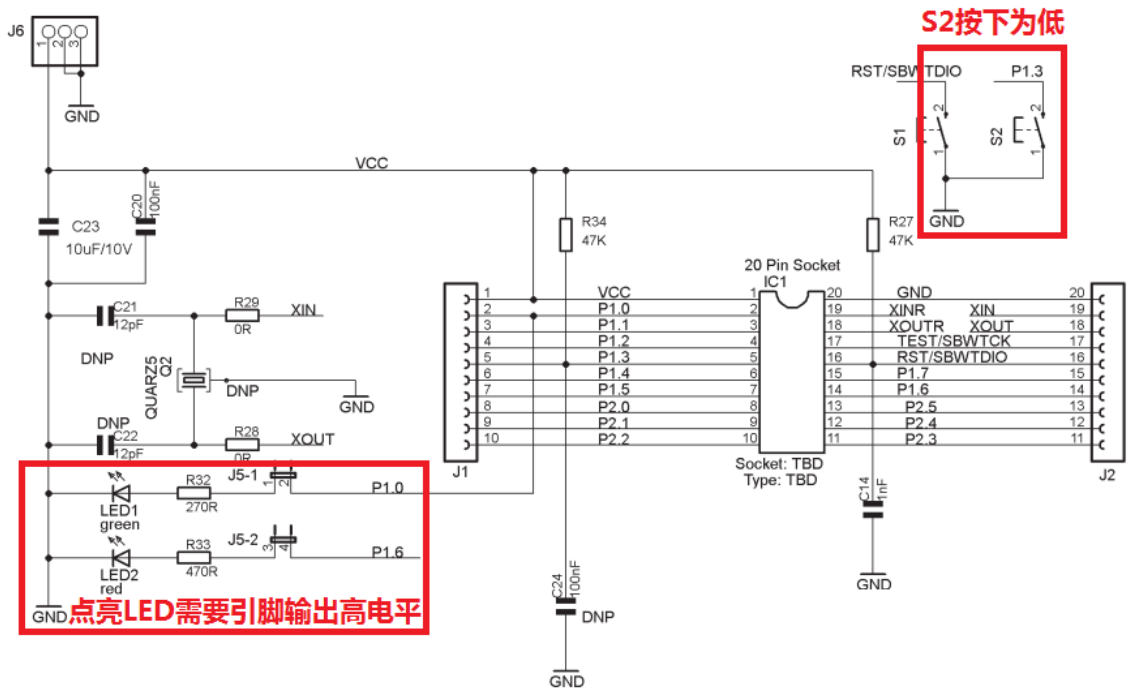
6.4.1 MSP-EXP430G2 LaunchPad概述

名为LaunchPad 的MSP-EXP430G2 低成本试验板是一款适用于TI 最新MSP430G2xx 系列产品的完整开发解决方案。其基于USB 的集成型仿真器可提供为全系列MSP430G2xx 器件开发应用所必需的所有软、
硬件。LaunchPad 具有集成的DIP 目标插座，可支持多达20 个引脚，从而使MSP430 Value Line 器件能
够简便地插入LaunchPad 电路板中。此外，其还可提供板上Flash 仿真工具，以直接连接至PC 轻松进行
编程、调试和评估。LaunchPad 试验板还能够对eZ430-RF2500T 目标板、eZ430-Chronos 手表模块或eZ430-F2012T/F2013T 目标板进行编程。此外，它还提供了从MSP430G2xx 器件到主机PC 或相连目标
板的9600 波特UART 串行连接。

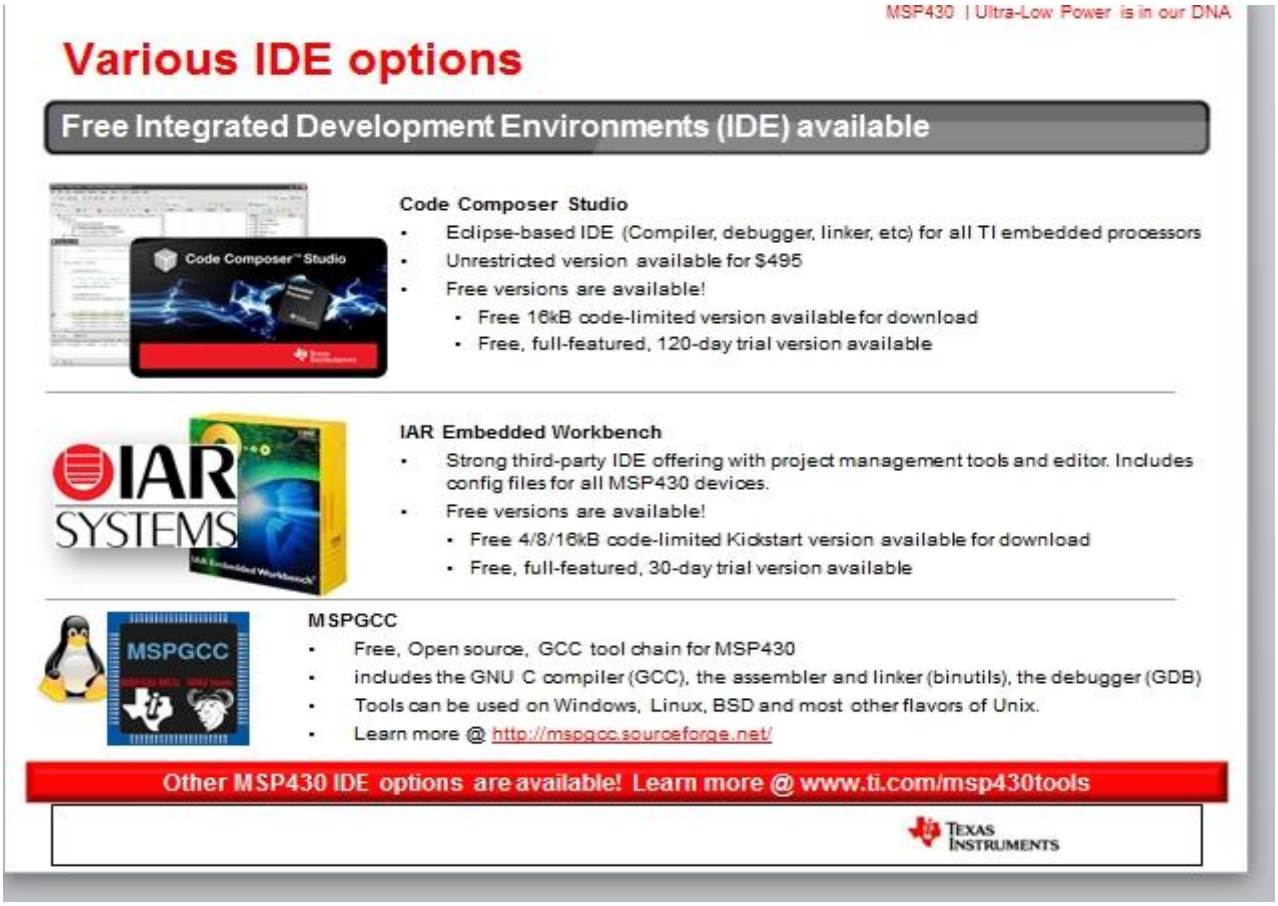
MSP-EXP430G2 采用IAR Embedded Workbench 集成开发环境(IDE) 或Code Composer Studio (CCS)编写、下载和调试应用。调试器是非侵入式的，这使用户能够借助可用的硬件断点和单步操作全速运行应用， 而不耗用任何其他硬件资源。
MSP-EXP430G2 LaunchPad 特性:
• USB 调试与编程接口无需驱动即可安装使用，且具备高达9600 波特的UART 串行通信速度
• 支持所有采用PDIP14 或PDIP20 封装的MSP430G2xx 和MSP430F20xx 器件
• 分别连接至绿光和红光LED 的两个通用数字I/O 引脚可提供视觉反馈
• 两个按钮可实现用户反馈和芯片复位
• 器件引脚可通过插座引出，既可以方便的用于调试，也可用来添加定制的扩展板
• 高质量的20 引脚DIP 插座，可轻松简便地插入目标器件或将其移除
ZHCU010–



6.4.2 MSP-EXP430G2 LaunchPad原理图



6.4.3MSP-EXP430G2 LaunchPad开发环境



6.4.4MSP-EXP430G2 LaunchPad分析

主要是利用launchpad实现了I/O功能，熟悉了launchpad 的软硬件开发环境。
需要查询一些相关数据手册，了解引脚功能特性，根据需求进行编程。
接收外围开关按键电路的输入信号，处理后进行输出和 DE2 的扩展槽进行通信，从而完成控制 DE2 进而控制整个设备的功能。
MSP430上电复位后，IO默认为输入状态，看门狗为开启状态。一般来说，诸如AT89S52那些，复位后默认看门狗是禁止的，而 MSP430 则默认是开启看门狗的，与 C8051F 类似。对于我们开发调试过程来说，前期都是禁止看门狗居多，所以，一般新建 430 工程，都会有下面语句：

```
WDTCTL = WDTPW + WDTHOLD;
```

WDTCTL是看门狗的控制寄存器，长度为16位，其高8位是看门狗口令，固定为0x5A，对 WDTCTL 进行写操作时，必须使用该口令 WDTPW，写错口令会导致系统复位。
WDTHOLD 是对看门口控制寄存器的第八位写 1，禁止看门狗时钟，停止计数。系统复位默认值为 0，即开启看门狗计数。
MSP430G2231 的 IO 是不支持位寻址的，如果之前有 51 使用经验，刚上手 430 可能觉得很别扭。因为对于 51，用 sbit LED = P1^0;定义之后，可以非常方便使用赋值语句 LED = 0;或者 LED = 1;对 P1.0 口进行电平控制。而对 430 的特定管脚控制的时候，却要通过一些与或非等运算来实现，如 P1OUT |= 0x01;或者 P1OUT &= 0xfe;。对于引脚特定位置位或清零，不采用 P1OUT = 0x01;或者 P1OUT = 0x00;，道理是显而易见的，因为如果采用后者直接赋值，那么整个端口的电平状态，可能都会改变。而通常情况，我们都只是想对特定位操作，其它引脚电平不应受到影响。
当然，这仅仅是对习惯 51 的人提个醒，要习惯运用这种表示方法，因为像 AVR、ARM 那些，很多都是不支持位寻址的，都是通过上述与或非等运算来处理的。
当然，对于类似的操作，我们可以通过定义宏的方式来处理，如：

```
#define LEDON()P1OUT |= 0x01
#define LEDOFF()P1OUT &= 0xfe
```

这样，我们使用的时候，就直接 LEDON();或者 LEDOFF();来实现 LED 的亮灭控制，进行代码移植的时候，也会方便很多。
MSP430 的低功耗，很大一部分是得益于它灵活的时钟配置，之所以在这一章节没有介绍它的时钟，而采用了默认设置（DCO1MHz），是因为它太灵活了，所以显得复杂。先点点灯，学会 IO 操作，激发学习兴趣。

6. IO操作主要涉及以下几个寄存器：
7. P1REN：上下拉电阻使能寄存器，用于配置管脚输入时的上下拉电阻，1使能，此时与P1OUT配合选择上拉或者下拉，1为上拉，0位下拉。
8. P1SEL：功能选择寄存器，1为引脚配置为使用外围模块功能。
9. P1IE：中断使能寄存器，1使能引脚中断。
10. P1IES：中断触发沿选择寄存器，1为下降沿触发，0为上升沿触发。
11. P1IFG：中断标志寄存器，发生中断事件时相应位置1。
12. P1DIR：方向寄存器，1为输出，0为输入，复位默认为输入状态。
13. P1OUT：输出寄存器，1时引脚输出高电平，0时输出低电平。

14. P1IN：输入寄存器，用于读取IO电平状态。

6.4.5 MSP-EXP430G2 LaunchPad控制源码

```
#include "msp430.h"

int main(void)
{
    int j;
  WDTCTL = WDTPW + WDTHOLD;
  // Stop watchdog timer


   P1DIR|=0xc3;//设置 P11.0 1.1 1.6 1.7 为输出端口
   P1OUT|=0x3c;//设置输入端口为上拉电阻类型
   P1REN|=0x3c;//使能输入端的上拉电阻
  while(1)
  {

   if(!(P1IN&0x04))//端口 P1.2 有按键,P1.0 置 1
   {
    for(j=0;j<3000;j++);//短暂延时，去抖动
    if(!(P1IN&0x04))
        {
            P1OUT|=0x01;
        }
    }
    else//端口 P1.2 没有按键,P1.0 清零
    {
      P1OUT&=0xfe;
    }

  if(!(P1IN&0x08))//端口 P1.3 有按键,P1.6 置 1
   {
    for(j=0;j<3000;j++);//短暂延时，去抖动
    if(!(P1IN&0x08))
        {
            P1OUT|=0x40;
        }
    }
  else//端口 P1.3 没有按键,P1.6 清零
  {
```

```
      P1OUT&=0xbf;
  }
   if(!(P1IN&0x20))//端口 P1.5 有按键,P1.7 置 1
   {
    for(j=0;j<3000;j++);//短暂延时，去抖动
    if(!(P1IN&0x20))
        {
            P1OUT|=0x80;
        }
    }
   else//端口 P1.5 没有按键,P1.7 清零
   {
       P1OUT&=0x7f;
   }
   if(!(P1IN&0x10))//端口 P1.4 有按键,P1.1 置 1
   {
    for(j=0;j<3000;j++);//短暂延时，去抖动
    if(!(P1IN&0x10))
        {
            P1OUT|=0x02;
        }
    }
   else//端口 P1.4 没有按键,P1.1 清零
   {
       P1OUT&=0xfd;
   }

  }
   return 0;
}
```

7 系统调试与成果展示

7.1 调试电路的方法和技巧

7.1.1 硬件调试

硬件调试主要目的是测试硬件是好的，我们需要编写测试模块对硬件进行测试，主要利用开关作为输入，可以利用 LED 作为输出，但是需要提醒的是软硬件部分的 LED 以及开关是不能冲突的。

我们硬件模块的关键部分是 cursor,在调试过程中也编写了测试硬件的模块。

7.1.2 软件调试

根据系统需求，利用 SOPC 可以多次重构系统。系统有了对应的 BSP 后，可以利用一些参考代码，完成我们的软硬件设计。System.h 文件的熟练应用对与软硬件协同非常重要，还有就是要熟悉各种编译错误，应该需要看到错误提示就基本可以将问题定位。

为了保证程序的逻辑正确性，我们首先在 VC 上将软件子模块进行编程，输入数据后观察输出的结果，测试软件的逻辑正确性。

程序的逻辑虽然正确，但是并不是随意放一个位置它都可以正常执行。必须保证不影响硬件的正常工作。举个例子：我们最初的设想是利用播放模式进行频谱分析显示五线谱，最后发现当初的设想完全不现实。为什么呢？因为 D / A 需要实时处理，否则播放出的音频信号几乎就是噪声。而频谱分析室一个计算量很大的程序，会占用 C P U 很多时间，所以这些完全是无法兼容的。

随后，我选择了一个没有 a d， d a 的模式这样频谱分析就不会影响音频模块的正常工作。

调试代码的过程中一个不起眼的错误，虽然编译通过。但实际程序在那里中断了，因为担心整个软件的代码是否可以顺利执行，这是就必须要调试了。我的调试方法是在程序的适当位置写指令通过判断 LED 是否发光，判断指令是否执行。此方法很有效，我一天都没搞定的问题当我想起这个方法时，很快将错误定位。其实语法错误不可怕，可怕的是这种逻辑错误。

利用中断的方式也是帮助设计人员理解代码，优化设计的很好方法。

关于对硬件的编程其实是一项非常繁琐的工作。你的程序也许逻辑上没有任何问题，但是在硬件上就不一定执行，利用 LED 非常直观的调试，很多时候是没隔几条指令就需要插入调试指令。当退出调试模式之后，这些都可以注释掉。这个点亮 led 的调试方法是我们自己想到的，希望可以提醒读者。

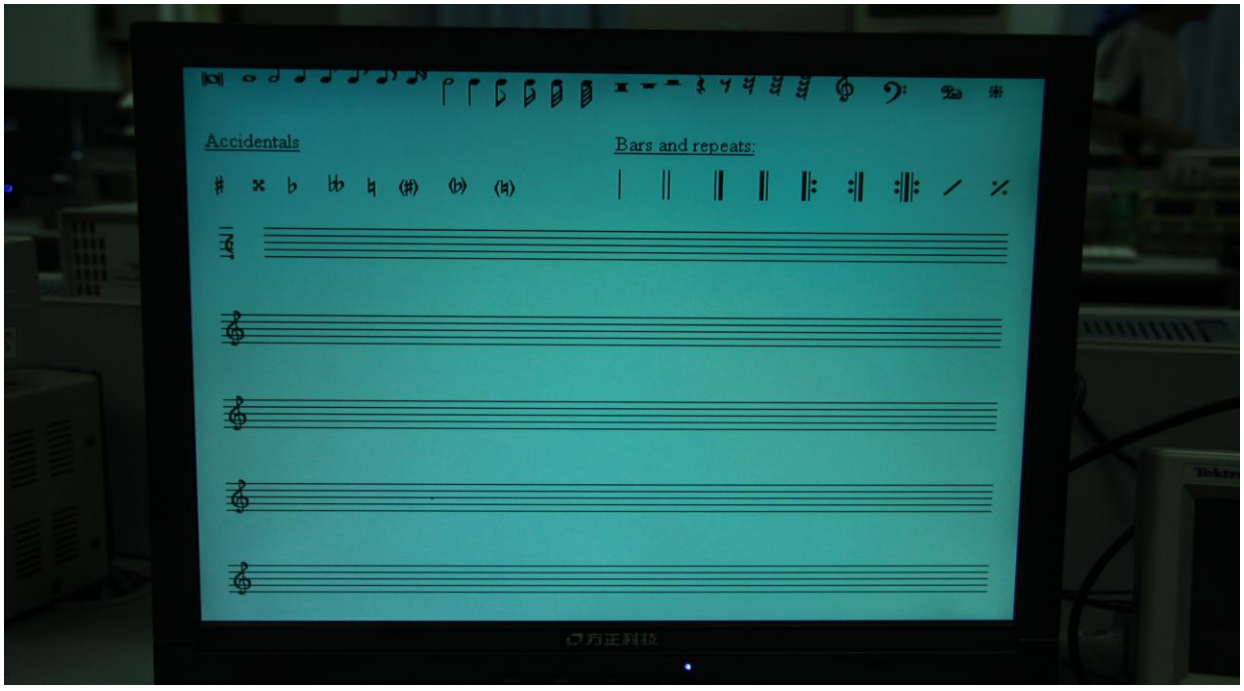
7.1.3 软硬件级联调试

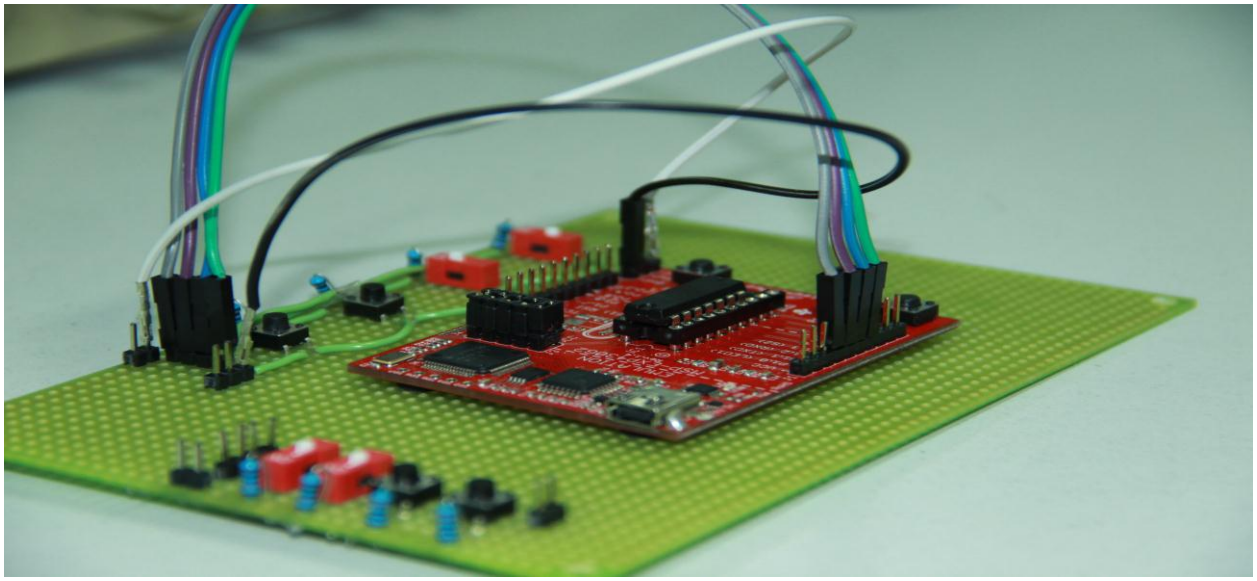
这个过程其实是一个彼此兼容妥协的过程，我们需要更改顶层的设置，需要对软件进行处理等等。因为我们需要一个共同的标准，出现错误的时候首先进行故障定位。这个时候软硬件的测试模块就非常有帮助，如果软硬件都是好的，那就是接口以及大家的定义一定要规范。

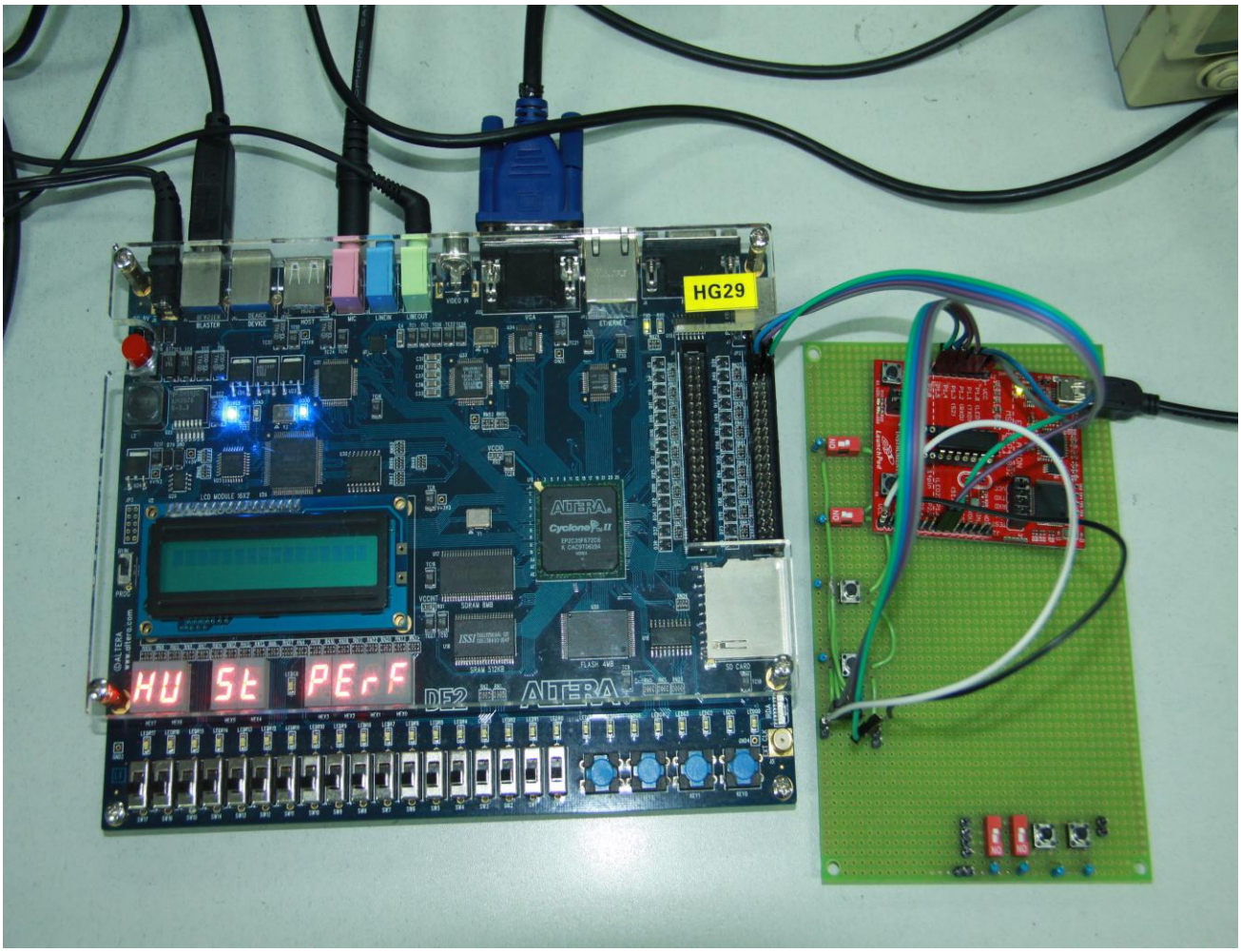
准确定义接口是我们模块化工作的前提，接口都是根据模块的输入输出信号确定的，更加重要的是有很多时序的问题要注意。尤其是和硬件相关，一定要非常注意时序。

7.2 成果展示

能够接收音频信号，将音乐对应的乐谱（简谱或五线谱）显示
进行谱曲操作时仅考虑音频信号的幅度特征，采用固定的节奏
迷你卡拉 OK 功能，实现音乐的有效回放功能
对核心算法进行优化；
实现滚动屏幕的功能；
由用户进行模式选择；
可以自主选择采样率，进行切换；
可以在录音模式以外的任何时刻进行音量调节；







8 项目总结

从方案开始提出的时候，我们就知道，这不会是一项简单的工作。从最初到现在，困难接踵而至。不熟悉的软硬件开发环境，编译时遇到的各种匪夷所思的错误，调试时面对未知错误一次又一次不断试验的茫然……这些都伴随我们走过一个又一个不眠的夜晚。从开始遇到问题的不知所措，到现在镇定的解决迎面而来的问题，我们也像我们的作品一样不断成长，也许它并不完美，但是我们有信心让它一点一点越变越好。

为了实现我们再本科前三年就可以自主设计并且开发一个非常新颖有意义的作品的梦想，我们选择了参加 TI 杯。在这个追逐小小梦想的过程中，我们的经历了多次的方案设计，初期不断地论证其可行性。在开发阶段，我们也是经历了各种艰辛，方案在实行的过程中还是会遇到种种问题，初期主要是我们对于软硬件资源与环境并不熟悉，学习两种新的开发工具使用两个开发平台，对于没有任何基础的我们还是非常有挑战的。在开始进行自选并且最终选择了我们希望的产品，我们就知道，我们需要为此付出的非常多。在这个课程设计的过程中学会的开发平台以及软

件使用已经不是什么新鲜事了，重要的是我们从此拥有了更快的学习能力，可以更好地应对未来的挑战

这个硬件课程设计我们花了很多时间也很有收获。我希望把我自己的工作和收获写下来作为资料留给需要帮助的同学看，让这些同学能够更快地收获到一些东西。

9. 致谢

感谢曾喻江老师给予的启发和帮助，老师会提供一些 reference design 以及一些非常有价值的指导意见。曾喻江老师的教学方式新颖，非常贴近实际，在他的课堂上可以感受到国外一流大学的教学特点，从中更是可以锻炼自己的能力。

感谢和我一起一起奋斗的队友，因为你们我在未知的道路上更有信心了。

感谢给我们提供音频模块设计的原作者，帮助我们完善了作品。

感谢硬件课程设计组以及 TI 公司给我们提供的平台和帮助。

感谢 cornel 的一个参考项目，其中的 VGA 部分对我们很有帮助。

10. 参考资料

曾喻江老师的教学网站 http://ei.hust.edu.cn/teacher/zengyj/

电子与信息工程系 本科教学网
http://ei.hust.edu.cn/benke/ShowSubChannelList.aspx?ChannelId=18e0d4b8-fc3b-4b10-9de2-37d1213c0794

Cornell 大学相关课程 final Project
http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2011/jl2782/musical_critics/index.html

Nios II Software Developer’s Handbook
http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf

经典的语音信号处理资料，主要是频谱分析的算法

11. 关于我们

“电子知音”网页网址
http://ei.hust.edu.cn/teacher/zengyj/2012/EI1300292/Dianzizhiyin/index.html
如果你有问题需要和我们讨论，欢迎联系我们！