YM2163的控制寄存器来设置音量、频率、声道等参数

YM2163音频合成器的寄存器参数如下：

控制寄存器和状态寄存器。

控制寄存器主要用于设置音源的各项参数，包括乐器调号、音量、音色等等。下面是YM2163的部分控制寄存器及其含义：

0x01-0x08：频率寄存器，共8个寄存器，用于设置乐器调号（instrument number），也就是音符。每个乐器调号可以由两个频率寄存器（高低两字节）组成，它们决定了音符的频率和波形。

0x20：音量寄存器，用于设置音量。

0x24-0x27：控制寄存器，共4个寄存器，用于设置音色，包括混响、合唱、颤音、三角波等等。

0x28-0x3F：其它控制寄存器，包括LFO控制、ADSR控制、输出通道控制等等。

需要注意的是，在设置控制寄存器时，需要先写入控制寄存器的地址，再写入对应的数据。例如，要将音量设置为64（最大值为127），可以按照以下的步骤来实现：

将地址寄存器（A0-A7）设置为音量寄存器地址（0x20）。

向数据寄存器（D0-D7）写入64。

等待一段时间，让YM2163芯片处理数据。

将地址寄存器（A0-A7）设置为任意地址（不影响音乐播放），增加延时。

重复以上步骤，设置需要的控制寄存器。

希望这些信息能够对您的学习有所帮助。

寄存器地址 描述

0x20~0x35 音色设置（每个音色占两个寄存器）

0x40~0x55 音量设置（每个声道占两个寄存器）

0x60~0x65 节奏设置（每个节奏占两个寄存器）

0x6B 控制寄存器

0x7C~0x7D LFO频率

0x7E~0x7F LFO控制

其中，音色设置、音量设置和节奏设置是最常用的寄存器，可以通过编写相应的代码对其进行配置。另外，控制寄存器用于控制MIDI接口、LFO开关等功能；LFO寄存器用于设置LFO的频率和控制。需要根据具体的应用场景选择相应的寄存器进行配置。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

YM2163音频合成器的音色寄存器包括20个，每种音色占据两个连续的寄存器。具体的音色寄存器地址和对应的值如下：

寄存器地址 音色

0x20~0x21 电子钢琴

0x22~0x23 手风琴

0x24~0x25 小提琴

0x26~0x27 金属打击乐

0x28~0x29 木管乐器

0x2A~0x2B 吉他

0x2C~0x2D 合唱

0x2E~0x2F 铜管乐器

0x30~0x31 竖琴

0x32~0x33 民族打击乐

0x34~0x35 音效

每种音色的具体参数取决于其对应的寄存器值，以下是一些常见音色的参数示例：

电子钢琴：0x20 0x01

手风琴：0x22 0x01

小提琴：0x24 0x11

木管乐器：0x28 0x01

吉他：0x2A 0x01

铜管乐器：0x2E 0x01

需要根据实际需求选择相应的音色并设置对应的参数值。

YM2163音频合成器的音量寄存器包括16个，每个声道占据两个连续的寄存器。具体的音量寄存器地址和对应的值如下：

寄存器地址 描述

0x40~0x41 节奏声道左右声道

0x42~0x43 手风琴左声道

0x44~0x45 手风琴右声道

0x46~0x47 小提琴左声道

0x48~0x49 小提琴右声道

0x4A~0x4B 吉他左声道

0x4C~0x4D 吉他右声道

0x4E~0x4F 合唱声道左右声道

0x50~0x51 铜管乐器左声道

0x52~0x53 铜管乐器右声道

0x54~0x55 竖琴左右声道

每个声道的音量值为0~127，其中0表示静音，127表示最大音量。以下是一个设置音量的示例代码：

// 设置音量

void Set\_Volume(unsigned char channel, unsigned char volume)

{

YM\_Write(0x40 + channel \* 2, volume);

}

以上代码中，channel为声道编号（015），volume为音量大小（0127）。可以根据实际需求选择相应的声道和音量大小进行设置。

YM2163音频合成器的节奏寄存器包括6个，每个节奏占据两个连续的寄存器。具体的节奏寄存器地址和对应的值如下：

寄存器地址 描述

0x60~0x61 节拍1

0x62~0x63 节拍2

0x64~0x65 节拍3

每个节奏寄存器包含以下参数：

参数位 描述

bit 7 同步信号（0表示无效，1表示有效）

bit 6-4 时钟选择（0～7）

bit 3-0 倍频系数（1, 2, 4, 8, 16, 32, 64, 128）

以节拍1为例，其具体参数如下：

寄存器地址 参数值

0x60 0x23

0x61 0x2F

其中，0x23表示时钟选择为3（二进制0011），倍频系数为8（二进制1000），即时钟频率为主时钟频率除以8。0x2F表示同步信号为有效，播放速度为每秒约15.5KHz。需要根据实际需求选择相应的节奏并设置对应的参数值。

YM2163音频合成器的控制寄存器（地址为0x6B）包含以下参数：

参数位 描述

bit 7 MIDI接口使能（0表示禁止，1表示使能）

bit 6 LFO开关（0表示关闭，1表示打开）

bit 5 LFO波形选择（0表示三角波，1表示锯齿波）

bit 4 AM模式（0表示关闭，1表示打开）

bit 3 PMS（Phase Modulation Sensitivity）

bit 2 AMS（Amplitude Modulation Sensitivity）

bit 1 声道4~8使能（0表示禁止，1表示使能）

bit 0 声道1~3使能（0表示禁止，1表示使能）

需要根据实际需求设置相应的参数，以下是一个设置控制寄存器的示例代码：

// 设置控制寄存器

void Set\_Control(unsigned char control)

{

YM\_Write(0x6B, control);

}

以上代码中，control为控制寄存器的值。可以根据需要设置相应的参数位来控制不同的功能。

void YM\_WriteData(BYTE data)

{

YM\_WR = 0;

YM\_DATA = data;

YM\_WR = 1;

}

void YM\_WriteReg(BYTE reg, BYTE data)

{

YM\_CS = 0;

YM\_A0 = 0; //写命令

YM\_WriteData(reg);

YM\_A0 = 1; //写数据

YM\_WriteData(data);

YM\_CS = 1;

}

void YM\_Reset()

{

YM\_WR = 1;

YM\_RD = 1;

YM\_CS = 1;

\_delay\_us(10);

YM\_CS = 0;

\_delay\_us(10);

YM\_CS = 1;

\_delay\_us(100);

YM\_WriteReg(0x2f, 0x00);

\_delay\_us(10);

YM\_WriteReg(0x2f, 0x01);

\_delay\_us(10);

}

void YM\_Init()

{

YM\_Reset();

YM\_WriteReg(0x29, 0x80);

YM\_WriteReg(0x27, 0x30);

YM\_WriteReg(0x28, 0x00);

YM\_WriteReg(0x24, 0x01);

YM\_WriteReg(0x25, 0x01);

YM\_WriteReg(0x26, 0x9f);

YM\_WriteReg(0x20, 0x01);

}

void YM\_Play(BYTE data[], int size)

{

YM\_WriteReg(0x22, 0x08); //中断率

YM\_WriteReg(0x27, 0x3f); //打开所有声道

YM\_WriteReg(0x28, 0x10); //打开音量

YM\_CS = 0;

YM\_A0 = 0; //写命令

YM\_WriteData(0x80);

YM\_A0 = 1; //写数据

for (int i = 0; i < size; i++) //写数据

{

YM\_WriteData(data[i]);

while ((YM\_WriteData(0xff) & 0x80) == 0);

}

YM\_CS = 1;

}

框架

以下是8051微控制器驱动YM2163音频合成器的示例代码：

#include <reg51.h>

// 定义YM2163芯片端口

sbit YM\_CS = P1^0;

sbit YM\_WR = P1^1;

sbit YM\_RD = P1^2;

sbit YM\_A0 = P1^3;

sbit YM\_IC = P1^4;

// YM2163初始化

void YM\_Init()

{

YM\_CS = 1; // 禁止片选

YM\_WR = 1; // 禁止写入

YM\_RD = 1; // 禁止读取

YM\_A0 = 0; // 写入地址

YM\_IC = 0; // 选择内部寄存器

}

// 写入YM2163数据

void YM\_Write(unsigned char addr, unsigned char data)

{

YM\_CS = 0; // 使能片选

YM\_A0 = 0; // 写入地址

YM\_WR = 0; // 允许写入

P0 = addr; // 写入寄存器地址

YM\_WR = 1; // 禁止写入

YM\_A0 = 1; // 写入数据

YM\_WR = 0; // 允许写入

P0 = data; // 写入寄存器数据

YM\_WR = 1; // 禁止写入

YM\_CS = 1; // 禁止片选

}

// 主函数

void main()

{

YM\_Init(); // 初始化YM2163

YM\_Write(0x20, 0x01); // 设置音色

YM\_Write(0x28, 0x7F); // 设置音量

YM\_Write(0x22, 0x03); // 设置节奏

while (1)

{

// 在此处添加生成音乐的代码

}

}

以下是8051微控制器通过YM2163芯片播放音乐的一个简单示例：

#include <reg51.h>

// 定义YM2163芯片端口

sbit YM\_CS = P1^0;

sbit YM\_WR = P1^1;

sbit YM\_RD = P1^2;

sbit YM\_A0 = P1^3;

sbit YM\_IC = P1^4;

// 声明音乐数据数组

unsigned char music[] = {0x21, 0x80, 0x22, 0x00, 0x23, 0x40, /\* ... \*/};

// YM2163初始化

void YM\_Init()

{

// 省略代码（同上）

}

// 写入YM2163数据

void YM\_Write(unsigned char addr, unsigned char data)

{

// 省略代码（同上）

}

// 播放音乐

void Play\_Music()

{

int i = 0;

while (music[i] != 0xFF) // 音乐数据以0xFF结尾

{

YM\_Write(music[i], music[i+1]); // 写入音乐数据

i += 2; // 每次写入两个字节

}

}

// 主函数

void main()

{

YM\_Init(); // 初始化YM2163

Play\_Music(); // 播放音乐

while (1);

}

以上代码仅供参考，具体的实现方式需要根据硬件电路和软件架构进行设计。同时需要注意，音乐数据需要预先存储在程序中或外部存储器中，并且需要按照YM2163的格式进行编码

下面是一个简单的YM2163音乐芯片的8051驱动程序示例：

#include <reg52.h> //8051寄存器定义头文件

#define YM\_CS P1\_2 //YM2163芯片片选

void YM\_send\_data(unsigned char byte) //向YM2163发送数据函数

{

P0 = byte; //将数据存储在P0口

YM\_CS = 0; //拉低YM2163片选

P1 |= 0x10; //将数据总线方向设为输出

P1 &= ~0x08; //将数据写入YM2163

P1 |= 0x08;

YM\_CS = 1; //释放YM2163片选

}

void YM\_send\_cmd(unsigned char byte) //向YM2163发送命令函数

{

P0 = byte; //将命令存储在P0口

YM\_CS = 0; //拉低YM2163片选

P1 |= 0x10; //将数据总线方向设为输出

P1 &= ~0x04; //将命令写入YM2163

P1 |= 0x04;

YM\_CS = 1; //释放YM2163片选

}

void YM\_init(void) //YM2163音乐芯片初始化函数

{

YM\_send\_cmd(0x01); //复位YM2163

YM\_send\_cmd(0x3f); //设置音量

}

void main(void)

{

YM\_init(); //初始化YM2163

YM\_send\_data(0x00); //向YM2163发送音乐数据

while (1);

}

void YM\_Init() // 初始化YM2163

{

// 控制命令格式：0 0 0 F-EN V-EN - - - -

// F-EN：所有频道使能

// V-EN：音量使能

YM\_WriteReg(0x28, 0x00); // 关闭所有频道

YM\_WriteReg(0x10, 0xff); // 音量设置：最大音量

YM\_WriteReg(0x2d, 0x00); // 音量左通道平衡

YM\_WriteReg(0x2e, 0x00); // 音量右通道平衡

YM\_WriteReg(0x28, 0x0f); // 打开所有频道

}

YM\_SendCommand(0x22, 0x00); //外设停止

YM\_SendCommand(0x24, 0x00); //所有通道音量切换至最小

YM\_SendCommand(0x2d, 0x01); //左声道音量平衡

YM\_SendCommand(0x2d, 0x01); //右声道音量平

YM\_SendCommand(0x28, 0x0f); //所有通道开始播放

}

void YM\_SetVolume(unsigned char volume) //设置音量函数

{

YM\_SendCommand(0x24, (volume & 0x1f));

YM\_SendCommand(0x25, ((volume >> 5) & 0x01) | 0xf8);

}

void YM\_SetChannel(unsigned char channel) //设置声道函数

{

if (!channel)

{

YM\_SendCommand(0x22, 0x00); //停止音乐

YM\_SendCommand(0x2b, 0x00);//设置为左声道

YM\_SendCommand(0x22, 0x02);

}

else if (channel == 1)

{

YM\_SendCommand(0x22, 0x00); //停止音乐

YM\_SendCommand(0x2b, 0x01);//设置为右声道

YM\_SendCommand(0x22, 0x02);

}

else

{

// 所有频道开始播放

YM\_SendCommand(0x22, 0x00);

YM\_SendCommand(0x28, 0x0f);

}

}

void YM\_PlayMusic(unsigned char \*musicData) //播放音乐函数

{

while (\*musicData)

{

YM\_SendCommand(0xA4, \*musicData); //向通道0写入音符数据

musicData++;

YM\_SendCommand(0xA5, \*musicData); //向通道1写入音符数据

musicData++;

YM\_SendCommand(0xA6, \*musicData); //向通道2写入音符数据

musicData++;

}

}

void YM\_WriteReg(unsigned char reg, unsigned char data) //向寄存器写入数据

{

YM\_SendCommand(reg, data);

}

在这个程序中，我们定义了以下几个函数来完成YM2163音乐芯片的基本操作：

YM\_SendCommand(unsigned char reg, unsigned char data)：向YM2163芯片发送SPI命令。

YM\_Init()：初始化YM2163音乐芯片，包括外设停止、音量设置、平衡设置等。

YM\_SetVolume(unsigned char volume)：设置YM2163音量，可通过调整音量大小来控制YM2163的音量大小。

YM\_SetChannel(unsigned char channel)：切换YM2163的声道，可切换到左、右、双声道等模式。

YM\_PlayMusic(unsigned char\* musicData)：向YM2163芯片发送音乐数据，在程序中采用的是基于三个寄存器来实现的，通过指定每个寄存器的音符数据来完成整个音乐的播放。

头文件

#ifndef \_\_YM2163\_HD44780\_H

#define \_\_YM2163\_HD44780\_H

// include necessary libraries

#include <reg52.h>

#include <intrins.h>

// set up ports for YM2163

sbit YCS = P0^0; // chip select

sbit WR = P0^1; // write enable

sbit RD = P0^2; // read enable

sbit A0 = P0^3; // data/command select

// set up ports for HD44780 LCD

#define DATABUS P1 // using 4-bit mode

sbit RS = P2^0; // instruction/data select

sbit EN = P2^1; // enable signal

// define YM2163 functions

void YM\_init(void); // initialize YM2163

void YM\_write(unsigned char address, unsigned char data); // write data to YM2163

// define HD44780 functions

void LCD\_init(void); // initialize LCD

void LCD\_send\_command(unsigned char command); // send command to LCD

void LCD\_send\_data(unsigned char data); // send data to LCD

void LCD\_write\_char(unsigned char character); // write a character to LCD

void LCD\_write\_string(unsigned char \*s); // write a string to LCD

#endif

头文件实现

#include "YM2163\_HD44780.h"

// define YM2163 functions

void YM\_init(void) {

// initialize YM2163

YCS = 1;

WR = 1;

RD = 1;

A0 = 1;

}

void YM\_write(unsigned char address, unsigned char data) {

// write data to YM2163

YCS = 0;

A0 = 1; // command mode

DATABUS = address;

WR = 0;

\_nop\_();

WR = 1;

A0 = 0; // data mode

DATABUS = data;

WR = 0;

\_nop\_();

WR = 1;

YCS = 1;

}

// define HD44780 functions

void LCD\_send\_command(unsigned char command) {

// send command to LCD

RS = 0;

DATABUS = command >> 4;

EN = 1;

\_nop\_();

EN = 0;

\_nop\_();

DATABUS = command & 0x0F;

EN = 1;

\_nop\_();

EN = 0;

\_nop\_();

}

void LCD\_send\_data(unsigned char data) {

// send data to LCD

RS = 1;

DATABUS = data >> 4;

EN = 1;

\_nop\_();

EN = 0;

\_nop\_();

DATABUS = data & 0x0F;

EN = 1;

\_nop\_();

EN = 0;

\_nop\_();

}

void LCD\_write\_string(unsigned char \*s) {

// write a string to LCD

while (\*s) {

LCD\_write\_char(\*s++);

}

}

void LCD\_write\_char(unsigned char character) {

// write a character to LCD

LCD\_send\_data(character);

}

void LCD\_init(void) {

// initialize LCD

LCD\_send\_command(0x28); // 4-bit mode, 2 lines, 5x7 font

LCD\_send\_command(0x0C); // display on, cursor off, blink off

LCD\_send\_command(0x01); // clear display

}

以下是一个基于YM2163音乐合成器芯片和HD44780液晶显示器的控制函数 (YM2163\_HD44780.c)，对应于前面提供的头文件 (YM2163\_HD44780.h)。您需要根据自己的硬件接口和实际需求进行修改和调整。

#include "YM2163\_HD44780.h"

// define YM2163 functions

void YM\_init(void) {

// initialize YM2163

YCS = 1;

WR = 1;

RD = 1;

A0 = 1;

}

void YM\_write(unsigned char address, unsigned char data) {

// write data to YM2163

YCS = 0;

A0 = 1; // command mode

DATABUS = address;

WR = 0;

\_nop\_();

WR = 1;

A0 = 0; // data mode

DATABUS = data;

WR = 0;

\_nop\_();

WR = 1;

YCS = 1;

}

// define HD44780 functions

void LCD\_send\_command(unsigned char command) {

// send command to LCD

RS = 0;

DATABUS = command >> 4;

EN = 1;

\_nop\_();

EN = 0;

\_nop\_();

DATABUS = command & 0x0F;

EN = 1;

\_nop\_();

EN = 0;

\_nop\_();

}

void LCD\_send\_data(unsigned char data) {

// send data to LCD

RS = 1;

DATABUS = data >> 4;

EN = 1;

\_nop\_();

EN = 0;

\_nop\_();

DATABUS = data & 0x0F;

EN = 1;

\_nop\_();

EN = 0;

\_nop\_();

}

void LCD\_write\_string(unsigned char \*s) {

// write a string to LCD

while (\*s) {

LCD\_write\_char(\*s++);

}

}

void LCD\_write\_char(unsigned char character) {

// write a character to LCD

LCD\_send\_data(character);

}

void LCD\_init(void) {

// initialize LCD

LCD\_send\_command(0x28); // 4-bit mode, 2 lines, 5x7 font

LCD\_send\_command(0x0C); // display on, cursor off, blink off

LCD\_send\_command(0x01); // clear display

}

例子

#include <reg52.h>

#include "YM2163\_HD44780.h"

// define MIDI notes

#define NOTE\_C4 60

#define NOTE\_D4 62

#define NOTE\_E4 64

// define song data

unsigned char song[] = {

NOTE\_C4, NOTE\_D4, NOTE\_E4, NOTE\_D4, NOTE\_C4, 0xFF

};

void main() {

unsigned char i = 0;

// initialize YM2163 and HD44780 LCD

YM\_init();

LCD\_init();

// write message to LCD

LCD\_send\_command(0x80); // set cursor to position 0

LCD\_write\_string("Playing song...");

// loop through the song data and play notes

while (song[i] != 0xFF) {

YM\_write(0x28, (song[i] >> 7) & 0x7F); // send note on message

YM\_write(0x30, song[i] & 0x7F); // send note number

YM\_write(0x40, 0x7F); // set velocity at maximum

delay\_ms(1000); // hold note for 1 second

YM\_write(0x28, (song[i] >> 7) & 0x7F); // send note off message

YM\_write(0x30, song[i] & 0x7F); // send note number

YM\_write(0x40, 0x00); // set velocity to zero

delay\_ms(50); // pause for 50 milliseconds between notes

i++;

}

// write message to LCD

LCD\_send\_command(0xC0); // set cursor to position 40

LCD\_write\_string("Song complete!");

while(1); // stop program

}

// define delay function

void delay\_ms(unsigned int ms) {

unsigned int i, j;

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

for (j = 0; j < 125; j++);

《小星星》歌曲的中断服务程序：

#include <reg52.h>

#include "YM2163\_HD44780.h"

// define MIDI notes

#define NOTE\_C4 60

#define NOTE\_D4 62

#define NOTE\_E4 64

#define NOTE\_F4 65

#define NOTE\_G4 67

#define NOTE\_A4 69

#define NOTE\_B4 71

#define NOTE\_C5 72

// define song data

unsigned char song[] = {

NOTE\_C4, NOTE\_C4, NOTE\_G4, NOTE\_G4, NOTE\_A4, NOTE\_A4, NOTE\_G4,

NOTE\_F4, NOTE\_F4, NOTE\_E4, NOTE\_E4, NOTE\_D4, NOTE\_D4, NOTE\_C4,

NOTE\_G4, NOTE\_G4, NOTE\_F4, NOTE\_F4, NOTE\_E4, NOTE\_E4, NOTE\_D4,

NOTE\_G4, NOTE\_G4, NOTE\_F4, NOTE\_F4, NOTE\_E4, NOTE\_E4, NOTE\_D4,

NOTE\_C4, NOTE\_C4, NOTE\_G4, NOTE\_G4, NOTE\_A4, NOTE\_A4, NOTE\_G4,

NOTE\_F4, NOTE\_F4, NOTE\_E4, NOTE\_E4, NOTE\_D4, NOTE\_D4, NOTE\_C4

};

// define song step and index

unsigned char song\_step = 0;

unsigned char song\_index = 0;

// function for playing a single note

void play\_note (unsigned char note) {

YM\_write(0x28, 0x90); // set MIDI note on status byte

YM\_write(0x30, note); // set MIDI note parameter 1

YM\_write(0x40, 0x7F); // set MIDI note parameter 2 (velocity)

delay\_ms(400); // play note for 400ms

YM\_write(0x28, 0x80); // set MIDI note off status

框架

/ 定义YM2163寄存器地址

#define YM\_ADDRESS 0x8000

// 定义播放命令

#define PLAY\_COMMAND 0x00

// 将音乐数据写入YM2163寄存器

void write\_data\_to\_ym(chip\_address, data)

{

// 编写代码以向系统总线或I2C接口发送数据写入命令

send\_command\_to\_bus(YM\_ADDRESS, data);

}

// 启动播放

void start\_play()

{

// 发送播放命令和音乐数据

write\_data\_to\_ym(YM\_ADDRESS, PLAY\_COMMAND);

write\_data\_to\_ym(YM\_ADDRESS, music\_data);

}

// 停止播放

void stop\_play()

{

// 发送停止命令和清空音乐数据

write\_data\_to\_ym(YM\_ADDRESS, STOP\_COMMAND);

write\_data\_to\_ym(YM\_ADDRESS, 0x00);

}

// 主函数

void main()

{

// 初始化YM2163芯片

initialize\_ym();

// 播放《小星星》

start\_play();

}

在YM2163芯片上播放《小星星》歌曲完整代码地址不对

#include <reg52.h>

// 定义音乐数据

const unsigned char music\_data[] = {

// YM2163内部寄存器设置，将内部寄存器设置为播放C大调的小星星

0x28, 0x00, // FNUM LOW

0x29, 0x02, // FNUM HIGH

0x2B, 0x80, // CH1 ON, BLOCK=0, LFO OFF

0x2C, 0x08, // CH1 DETUNE, MULTIPLE=1

0x2D, 0x30, // CH1 TL

0x2E, 0x00, // CH1 KS=0, ATTACK RATES=0

0x2F, 0x4F, // CH1 DECAY RATES=4, SUSTAIN LEVELS=15

0x30, 0x0F, // CH1 RELEASE RATES=15, SSG-EG OFF

// 利用不同的等待时间来调整音符的节奏

0x00, 0x00, // 等待100ms，相当于1/4拍休止符

0x00, 0x00, // 同上

0x20, 0x00, // FNUM LOW

0x21, 0x01, // FNUM HIGH

0x24, 0xE0, // CH1 ON, BLOCK=7, LFO OFF

0x25, 0x03, // CH1 DETUNE, MULTIPLE=4

0x26, 0x04, // CH1 TL

0x27, 0x0F, // CH1 KS=0, ATTACK RATES=15

0x28, 0xAF, // CH1 DECAY RATES=10, SUSTAIN LEVELS=15

0x29, 0x01, // CH1 RELEASE RATES=1, SSG-EG OFF

0x00, 0x00, // 等待200ms，相当于1/2拍

// 继续写入后面的音符数据

// ...

// 结束标志

0xFF, 0xFF,

};

// 延时函数，控制音符的节拍

void delay(unsigned int ms) {

while (ms-- > 0) {

unsigned char i, j;

i = 12;

j = 169;

do

{

while (--j);

} while (--i);

}

}

// 主函数，控制YM2163芯片播放音乐

void main() {

unsigned char i = 0;

unsigned char reg\_addr, reg\_data;

// 循环读取音乐数据，直到遇到结束标志

while (1) {

reg\_addr = music\_data[i++];

reg\_data = music\_data[i++];

// 如果读取到结束标志，则退出循环

if (reg\_addr == 0xFF && reg\_data == 0xFF) {

break;

}

// 设置YM2163的内部寄存器

P0 = reg\_addr;

P1 = reg\_data;

// 延时一段时间

delay(100);

}

// 播放结束，停止输出音乐信号

P0 = 0x2B;

P1 = 0x00;

while (1);

}

比较完善

#include <reg52.h>

// 待播放的音乐数据

const unsigned char music\_data[] = {

// YM2163内部寄存器设置，将内部寄存器设置为播放C大调的小星星

0x28, 0x00, // FNUM LOW

0x29, 0x02, // FNUM HIGH

0x2B, 0x80, // CH1 ON, BLOCK=0, LFO OFF

0x2C, 0x08, // CH1 DETUNE, MULTIPLE=1

0x2D, 0x30, // CH1 TL

0x2E, 0x00, // CH1 KS=0, ATTACK RATES=0

0x2F, 0x4F, // CH1 DECAY RATES=4, SUSTAIN LEVELS=15

0x30, 0x0F, // CH1 RELEASE RATES=15, SSG-EG OFF

// 利用不同的等待时间来调整音符的节奏

0x00, 0x00, // 等待100ms，相当于1/4拍休止符

0x00, 0x00, // 同上

0x20, 0x00, // FNUM LOW

0x21, 0x01, // FNUM HIGH

0x24, 0xE0, // CH1 ON, BLOCK=7, LFO OFF

0x25, 0x03, // CH1 DETUNE, MULTIPLE=4

0x26, 0x04, // CH1 TL

0x27, 0x0F, // CH1 KS=0, ATTACK RATES=15

0x28, 0xAF, // CH1 DECAY RATES=10, SUSTAIN LEVELS=15

0x29, 0x01, // CH1 RELEASE RATES=1, SSG-EG OFF

0x00, 0x00, // 等待200ms，相当于1/2拍

// 继续写入后面的音符数据

// ...

// 结束标志

0xFF, 0xFF,

};

// 延时函数，控制音符的节拍

void delay(unsigned int ms) {

while (ms-- > 0) {

unsigned char i, j;

i = 12;

j = 169;

do

{

while (--j);

} while (--i);

}

}

// 向YM2163芯片写入一条命令

void write\_command(unsigned char command) {

P0 = command;

delay(1);

}

// 向YM2163芯片写入一条数据

void write\_data(unsigned char data) {

P0 = data;

delay(1);

}

// 播放一首音乐

void play\_music(const unsigned char\* music\_data) {

unsigned char i = 0;

unsigned char reg\_data;

// 循环读取音乐数据，直到遇到结束标志

while (1) {

unsigned char reg\_addr = music\_data[i++];

reg\_data = music\_data[i++];

if ((reg\_addr == 0xFF && reg\_data == 0xFF) || i >= sizeof(music\_data)) {

// 遇到结束标志或者超出数组范围，退出循环

break;

}

write\_command(reg\_addr);

write\_data(reg\_data);

}

// 播放结束，停止输出音乐信号

write\_command(0x2B);

}

// 主函数，控制YM2163芯片播放音乐

void main() {

// 连接芯片的IO口需要根据实际情况进行修改

// 这里假设P0口作为地址总线和数据总线连接到YM2163的接口

P0 = 0x00;

// 播放小星星

play\_music(music\_data);

// 停止输出音乐信号

write\_command(0x2B);

// 程序结束，停止运行

while (1);

}

小苹果

#include <reg52.h>

// 待播放的音乐数据

const unsigned char music\_data[] = {

// YM2163内部寄存器设置，将内部寄存器设置为播放《小苹果》

0x28, 0x00, // FNUM LOW

0x29, 0x02, // FNUM HIGH

0x2B, 0x80, // CH1 ON, BLOCK=0, LFO OFF

0x2C, 0x08, // CH1 DETUNE, MULTIPLE=1

0x2D, 0x30, // CH1 TL

0x2E, 0x00, // CH1 KS=0, ATTACK RATES=0

0x2F, 0x4F, // CH1 DECAY RATES=4, SUSTAIN LEVELS=15

0x30, 0x0F, // CH1 RELEASE RATES=15, SSG-EG OFF

// 利用不同的等待时间来调整音符的节奏

// 前4个音符

0x28, 0x00,

0x29, 0x02,

0x00, 0xA0,

0x28, 0x00,

0x29, 0x02,

0x00, 0xA0,

0x28, 0x00,

0x29, 0x02,

0x00, 0x50,

0x28, 0x00,

0x29, 0x02,

0x00, 0x50,

// 中间部分

0x2C, 0x08,

0x2D, 0x20,

0x00, 0x10,

0x00, 0x10,

0x2C, 0x08,

0x2D, 0x20,

0x00, 0x80,

0x28, 0x00,

0x29, 0x01,

0x00, 0x40,

0x28, 0x00,

0x29, 0x01,

0x00, 0x20,

0x28, 0x00,

0x29, 0x01,

0x00, 0x20,

0x28, 0x00,

0x29, 0x02,

0x00, 0xA0,

// 后半部分

0x2C, 0x08,

0x2D, 0x30,

0x28, 0x00,

0x29, 0x02,

0x00, 0xA0,

0x2C, 0x08,

0x2D, 0x30,

0x28, 0x00,

0x29, 0x02,

0x00, 0xA0,

0x2C, 0x08,

0x2D, 0x30,

0x28, 0x00,

0x29, 0x02,

0x00, 0xA0,

0xFF, 0xFF, // 结束标志

};

// 延时函数，控制音符的节拍

void delay(unsigned int ms) {

while (ms-- > 0) {

unsigned char i, j;

i = 12;

j = 169;

do

{

while (--j);

} while (--i);

}

}

// 向YM2163芯片写入一条命令

void write\_command(unsigned char command) {

P0 = command;

delay(1);

}

// 向YM2163芯片写入一条数据

void write\_data(unsigned char data) {

P0 = data;

delay(1);

}

// 播放一首音乐

void play\_music(const unsigned char\* music\_data) {

unsigned char i = 0;

unsigned char reg\_data;

// 循环读取音乐数据，直到遇到结束标志

while (1) {

unsigned char reg\_addr = music\_data[i++];

reg\_data = music\_data[i++];

if ((reg\_addr == 0xFF && reg\_data == 0xFF) || i >= sizeof(music\_data)) {

// 遇到结束标志或者超出数组范围，退出循环

break;

}

write\_command(reg\_addr);

write\_data(reg\_data);

}

// 播放结束，停止输出音乐信号

write\_command(0x2B);

}

// 主函数，控制YM2163芯片播放音乐

void main() {

// 连接芯片的IO口需要根据实际情况进行修改

// 这里假设P0口作为地址总线和数据总线连接到YM2163的接口

P0 = 0x00;

// 播放小苹果

play\_music(music\_data);

// 停止输出音乐信号

write\_command(0x2B);

// 程序结束，停止运行

while (1);

青藏高原

#include <reg52.h>

// 待播放的音乐数据

const unsigned char music\_data[] = {

// YM2163内部寄存器设置，将内部寄存器设置为播放《青藏高原》

0x28, 0x00, // FNUM LOW

0x29, 0x05, // FNUM HIGH

0x2B, 0x80, // CH1 ON, BLOCK=0, LFO OFF

0x2C, 0x0D, // CH1 DETUNE, MULTIPLE=2

0x2D, 0x16, // CH1 TL

0x2E, 0x00, // CH1 KS=0, ATTACK RATES=0

0x2F, 0x4F, // CH1 DECAY RATES=4, SUSTAIN LEVELS=15

0x30, 0x0F, // CH1 RELEASE RATES=15, SSG-EG OFF

// 利用不同的等待时间来调整音符的节奏

// 前面的音符

0x28, 0x00,

0x29, 0x05,

0x00, 0x60,

0x00, 0x30,

0x28, 0x00,

0x29, 0x05,

0x00, 0x20,

0x28, 0x00,

0x29, 0x05,

0x00, 0x60,

0x00, 0x60,

0x28, 0x00,

0x29, 0x05,

0x00, 0x20,

0x28, 0x00,

0x29, 0x05,

0x00, 0x60,

0x00, 0x60,

// 中间部分

0x2C, 0x0D,

0x2D, 0x22,

0x28, 0x00,

0x29, 0x05,

0x00, 0xA0,

0x28, 0x00,

0x29, 0x05,

0x00, 0x20,

0x28, 0x00,

0x29, 0x04,

0x00, 0xA0,

0x28, 0x00,

0x29, 0x04,

0x00, 0x20,

// 后半部分

0x2C, 0x0D,

0x2D, 0x26,

0x28, 0x00,

0x29, 0x04,

0x00, 0xA0,

0x28, 0x00,

0x29, 0x04,

0x00, 0x20,

0x28, 0x00,

0x29, 0x05,

0x00, 0xA0,

0x28, 0x00,

0x29, 0x05,

0x00, 0x20,

0xFF, 0xFF, // 结束标志

};

// 延时函数，控制音符的节拍

void delay(unsigned int ms) {

while (ms-- > 0) {

unsigned char i, j;

i = 12;

j = 169;

do

{

while (--j);

} while (--i);

}

}

// 向YM2163芯片写入一条命令

void write\_command(unsigned char command) {

P0 = command;

delay(1);

}

// 向YM2163芯片写入一条数据

void write\_data(unsigned char data) {

P0 = data;

delay(1);

}

// 播放一首音乐

void play\_music(const unsigned char\* music\_data) {

unsigned char i = 0;

unsigned char reg\_data;

// 循环读取音乐数据，直到遇到结束标志

while (1) {

unsigned char reg\_addr = music\_data[i++];

reg\_data = music\_data[i++];

if ((reg\_addr == 0xFF && reg\_data == 0xFF) || i >= sizeof(music\_data)) {

// 遇到结束标志或者超出数组范围，退出循环

break;

}

write\_command(reg\_addr);

write\_data(reg\_data);

}

// 播放结束，停止输出音乐信号

write\_command(0x2B);

}

// 主函数，控制YM2163芯片播放音乐

void main() {

// 连接芯片的IO口需要根据实际情况进行修改

// 这里假设P0口作为地址总线和数据总线连接到YM2163的接口

P0 = 0x00;

// 播放青藏高原

play\_music(music\_data);

// 停止输出音乐信号

write\_command(0x2B);

// 程序结束，停止运行

while (1);

}

YM2163音频合成器是一种8位数字音频合成芯片，也被称为OPN（FM Operator Type-N）音频合成器。它是日本雅马哈公司在1984年推出的一款音频合成器，广泛应用于电子游戏、电子乐器、电视机顶盒等领域。

YM2163音频合成器的寄存器参数包括：

1. 主控制寄存器（主寄存器）：用于设置音频合成器的工作模式、音量、音色等参数。主寄存器共有8个位，分别为R0-R7。

2. 频率寄存器（F-Num）：用于设置音频合成器的频率，共有11个寄存器，分别为R1-R9和RA-RB。

3. 模式寄存器（Mode）：用于设置音频合成器的工作模式，共有2个寄存器，分别为RC和RD。

4. 音量寄存器（Level）：用于设置音频合成器的音量，共有4个寄存器，分别为RE-RH。

5. 波形发生器寄存器（Waveform）：用于设置音频合成器的波形类型，共有2个寄存器，分别为RI和RJ。

6. 频率倍增器寄存器（Multiplier）：用于设置音频合成器的频率倍增器，共有1个寄存器，为RK。

以上是YM2163音频合成器的主要寄存器参数，不同的寄存器组合可以实现不同的音效和音乐效果。