键盘模块控制实验

1953729 吴浩泽

## 实验目的

通过实验掌握键盘控制与设计方法。

熟练编写IIC通信处理程序。

## 实验设备

硬件：Embest EduKit-IV平台，ULINK2仿真器套件，PC机。

软件：μVision IDE for ARM集成开发环境，Windows 98/2000/NT/XP。

## 实验内容

使用实验板上5x3用户键盘，编写程序读取键盘控制芯片寄存器值。

通过IIC总线读入键值，并将读到的键值发送到串口。

## 实验原理

**1. 常规键盘电路设计原理**

用户设计行列键盘接口，一般常采用三种方法读取键值。一种是中断式，另两种是扫描法和反转法。

中断式

在键盘按下时产生一个外部中断通知CPU，并由中断处理程序通过不同的地址读取数据线上的状态，判断哪个按键被按下。当然也可用查询来代替中断，本实验采用查询方式实现用户键盘接口。

扫描法

对键盘上的某一行发送低电平，其他为高电平，然后读取列值，若列值中有一位是低，表明该行与低电平对应列的键被按下。否则扫描下一行。

反转法

先将所有行扫描线输出低电平，读列值，若列值有一位是低，表明有键按下；接着所有列扫描线输出低电平，再读行值。根据读到的值组合就可以查表得到键码。

**2. 使用ZLG7290的键盘电路设计原理**

(1) ZLG7290的特点

 IIC串行接口，提供键盘中断信号，方便与处理器接口；

 可驱动8位共阴数码管或64只独立LED和64个按键；

 可控扫描位数，可控任一数码管闪烁；

 提供数据译码和循环，移位，段寻址等控制；

 8个功能键，可检测任一键的连击次数；

 无需外接元件即直接驱动LED，可扩展驱动电流和驱动电压；

 提供工业级器件，多种封装形式PDIP24，SO24。

(2) ZLG7290的引脚说明

采用24引脚封装，引脚图如下所示。其引脚功能分述如下：



ZLG7290引脚图

(3) ZLG7290的寄存器说明

系统寄存器（SystemReg）：地址00H，复位值11110000B。系统寄存器保存ZLG7290的系统状态，并可对系统运行状态进行配置。

KeyAvi（SystemReg.0）：置1时表示有效的按键动作（普通键的单击，连击，和功能键状态变化），/INT 引脚信号有效（变为低电平）；清0 表示无按键动作，/INT 引脚信号无效（变为高阻态）。有效的按键动作消失后或读Key 后KeyAvi 位自动清0。

键值寄存器（Key）：地址01H，复位值00H。Key 表示被压按键的键值。当Key=0 时，表示没有键被压按。

连击次数计数器（RepeatCnt）：地址02H，复位值00H 。RepeatCnt=0 时，表示单击键。RepeatCnt 大于0 时，表示键的连击次数。连击次数计数器用于区别单击或连击，判断连击次数可以检测被按时间。

功能键寄存器（FunctionKey）：地址03H，复位值0FFH。FunctionKey 对应位的值=0 表示对应功能键被压按（FunctionKey.7 ~FunctionKey.0 对应S64 S57）。

命令缓冲区（CmdBuf0~CmdBuf1）：地址07H~08H，复位值00H~00H。用于传输指令。

闪烁控制寄存器（FlashOnOff）：地址0CH，复位值0111B/0111B。高4 位表示闪烁时亮的时间，低4 位表示闪烁时灭的时间，改变其值同时也改变了闪烁频率，也能改变亮和灭的占空比。FlashOnOff的1 个单位相当于150~250ms（亮和灭的时间范围为：1~16，0000B 相当1 个时间单位），所有象素的闪烁频率和占空比相同。

扫描位数寄存器（ScanNum）：地址0DH，复位值7。用于控制最大的扫描显示位数（有效范围为0~7，对应的显示位数为：1~8），减少扫描位数可提高每位显示扫描时间的占空比，以提高LED 亮度。不扫描显示的显示缓存寄存器则保持不变。如ScanNum=3 时，只显示DpRam0~DpRam3 的内容。

显示缓存寄存器（DpRam0~DpRam7）：地址10H~17H，复位值00H~00H。缓存中位置1 表示该像素亮，DpRam7~DpRam0 的显示内容对应Dig7~Dig0 引脚。

(4) ZLG7290的通信接口

ZLG7290 的IIC 接口传输速率可达32kbit/s，容易与处理器接口通信，并提供键盘中断信号，提高主处理器时间效率。ZLG7290 的从地址 slave address 为70H (01110000B)。我们从它的键值寄存器（01H）中读取按键值（ucChar用于保存读到的键值）：

iic\_read(0x70, 0x1, &ucChar);

有效的按键动作都会令系统寄存器（SystemReg）的KeyAvi 位置1，/INT 引脚信号有效（变为低电平）。用户的键盘处理程序可由/INT 引脚低电平中断触发，以提高程序效率；也可以不采样/INT引脚信号节省系统的I/O数，而轮询系统寄存器的KeyAvi位。要注意读键值寄存器会令KeyAvi 位清0，并会令/INT 引脚信号无效。为确保某个有效的按键动作所有参数寄存器的同步性，建议利用IIC 通信的自动增址功能连续读RepeatCnt， FunctionKey 和Key 寄存器，但用户无需太担心寄存器的同步性问题，因为键参数寄存器变化速度较缓慢（典型250ms，最快9ms）。

ZLG7290 内可通过IIC 总线访问的寄存器地址范围为：00H~17H，任一寄存器都可按字节直接读写，也可以通过命令接口间接读写或按位读写，请参考ZLG7290芯片手册。ZLG7290支持自动增址功能（访问一寄存器后寄存器子地址自动加一）和地址翻转功能（访问最后一寄存器后寄存器子地址翻转为00H ）。ZLG7290 的控制和状态查询全部都是通过读/写寄存器实现的，用户只需象读写24C02 内的单元一样即可实现对ZLG7290 的控制，关于IIC 总线访问的细节请参考IIC 总线规范

**3. 键盘硬件电路设计**

(1) 键盘连接电路



5x4 键盘连接电路

(2) 键盘控制电路

键盘控制电路使用芯片ZLG7290控制，如图6-2-3。对应下图中的14引脚KEY\_INT捕捉由键盘按下产生的中断触发信号。



5x4 键盘控制电路

(3) 工作过程

键盘动作由芯片ZLG7290检测，当键盘按下时，ZLG7290系统寄存器（SystemReg）的最低位KeyAvi（SystemReg.0）置1（通过CPU查询方式得到），CPU通过IIC总线读取芯片ZLG7290键值寄存器（01H）中保存的键值。

## 实验步骤

**1. 准备实验环境**

使用ULINK2仿真器连接Embest EduKit-IV实验平台的主板JTAG接口；使用Embest EduKit-IV实验平台附带的交叉串口线，连接实验平台主板上的COM2和PC机的串口（一般PC只有一个串口，如果有多个请自行选择，笔记本没有串口设备的可购买USB转串口适配器扩充）；使用Embest EduKit-IV实验平台附带的电源适配器，连接实验平台主板上的电源接口。最后请将随实验平台一起附带的八段数码管/键盘模块插在Extent A区上。

**2. 串口接收设置**

在PC机上运行windows自带的超级终端串口通信程序，或者使用实验平台附带光盘内设置好了的超级终端，设置超级终端：波特率115200、1位停止位、无校验位、无硬件流控制，或者使用其它串口通信程序。（注：超级终端串口的选择根据用户的PC串口硬件不同，请自行选择，如果PC机只有一个串口，一般是COM1）。

**3. 打开实验例程**

1）拷贝实验平台附带光盘DISK3\_S3C2410\03-Codes\01-MDK\Mini2410-IV文件夹到MDK的安装路径：Keil\ARM\Boards\Embest\（如果本实验之前已经拷贝，可以跳过这一步）。（注：用户也可拷贝工程到任意目录，本实验为了便于教学，故统一实验路径）；

2）运行μVision IDE for ARM软件，点击菜单栏“Project”，选择“Open Project…”，在弹出的对话框选择实验例程目录6.2\_Keyboard\_Test子目录下的Keyboard\_Test.Uv2工程。

3）默认打开的工程在源码编辑窗口会显示实验例程的说明文件readme.txt，详细阅读并理解实验内容。

4）工程提供了两种运行方式：一是下载到SDRAM中调试运行，二是固化到Nor Flash中运行。用户可以在工具栏Select Target下拉框中选择在RAM中调试运行还是固化Flash中运行。如下图所示：



选择运行方式

下面实验将介绍下载到SDRAM中调试运行，所以我们在Select Target下拉框中选择Keyboard\_Test IN RAM。

5）接下来开始编译链接工程，在菜单栏“Projiet”选择“Build target”或者“Rebuild all target files”编译整个工程，用户也可以在工具栏单击“”或者“”进行编译。

6）编译完成后，在输出窗口可以看到编译提示信息，比如“".\SDRAM\Keyboard\_Test.axf" - 0 Error(s), 1 Warning(s).”，如果显示“0 Error(s)”即表示编译成功。

7）拨动实验平台电源开关，给实验平台上电，单击菜单栏Debug->Start/Stop Debug Session项将编译出来的映像文件下载到SDRAM中，或者单击工具栏“”按钮来下载。

8）下载完成后，单击菜单栏Debug->Run项运行程序，或者单击工具栏“”按钮来全速运行程序。用户也可以使用进行单步调试程序。

9）全速运行后，用户可以在超级终端看到程序运行的信息，此时用户可按下模块上的键盘，可在超级终端上显示所对应的键值。

10）用户可以Stop程序运行，使用μVision IDE for ARM的一些调试窗口跟踪查看程序运行的信息。

注：如果在第4）步用户选择在Flash中运行，则编译链接成功后，单击菜单栏Flash->Download项将程序固化到NorFlash中，或者单击工具栏按钮“”固化程序，从实验平台的主板拔出JTAG线，给实验平台重新上电，程序将自动运行。

**4. 观察实验结果**

在执行到第8）步时，可以看到超级终端上输出如下字符。

|  |
| --- |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* 英蓓特EduKit系列嵌入式教学系统平台 \*\*  \*\* Embest EduKit Series Embedded Teaching Platform \*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Keyboard Test Example  8-Segment Digit LED Test Example (Please look at LED) |

此时用户可按下模块上的键盘，可在超级终端上显示所对应的键值。如下：

|  |
| --- |
| press key 0  press key 1  press key 2  press key 3  press key 5  press key 6  press key 7  press key 8  press key 9  press key A  press key B  press key C  press key D  press key E |

## 实验参考程序

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* File name: keyboard\_test.c  \* Author: embest  \* Descript: Keyboard\_Test  \* History:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*------------------------------------------------------------------------------------------\*/  /\* include files \*/  /\*------------------------------------------------------------------------------------------\*/  #include "2410lib.h"  /\*------------------------------------------------------------------------------------------\*/  /\* function declare \*/  /\*------------------------------------------------------------------------------------------\*/  void keyboard\_test(void);  void \_\_irq keyboard\_int(void);  UINT8T key\_set(UINT8T ucChar);  /\*------------------------------------------------------------------------------------------\*/  /\* extern function \*/  /\*------------------------------------------------------------------------------------------\*/  extern void iic\_write\_keybd(UINT32T unSlaveAddr, UINT32T unAddr, UINT8T ucData);  extern void iic\_read\_keybd(UINT32T unSlaveAddr, UINT32T unAddr, UINT8T \*pData);  extern void iic\_init\_keybd(void);  /\*------------------------------------------------------------------------------------------\*/  /\* global variable \*/  /\*------------------------------------------------------------------------------------------\*/  UINT32T g\_nKeyPress;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* name: keyboard\_init  \* func: keyboard initialize  \* para: none  \* ret: none  \* modify:  \* comment:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void keyboard\_init(void)  {  iic\_init\_keybd();  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* name: keyboard\_test  \* func: test keyboard  \* para: none  \* ret: none  \* modify:  \* comment:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void keyboard\_test(void)  {  UINT8T ucChar;  uart\_printf("\n Keyboard Test Example\n");  keyboard\_init();  while(1)  {  // Wait until one key was pressed  while(1)  {  iic\_read\_keybd(0x70, 0x00, &ucChar);  if(ucChar & 0x01)  break;  }  iic\_read\_keybd(0x70, 0x1, &ucChar); // Get data from Key(register of ZLG7290)  if(ucChar != 0)  {  ucChar = key\_set(ucChar); // Key map for EduKitIV  if(ucChar < 10) ucChar += 0x30;  else if(ucChar < 15) ucChar += 0x37;  uart\_printf(" press key %c\n", ucChar);  }  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* name: key\_set  \* func: keyboard setting  \* para: none  \* ret: none  \* modify:  \* comment:  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  UINT8T key\_set(UINT8T ucChar)  {  switch(ucChar)  {  case 1:  case 2:  case 3:  case 4:  case 5:  ucChar-=1; break;  case 9:  case 10:  case 11:  case 12:  case 13:  ucChar-=4; break;  case 17:  case 18:  case 19:  case 20:  case 21:  ucChar-=7; break;  default: ucChar = 0;  }  return ucChar; } |

## 读书笔记

IIC通信是一种同步通信方式，同步通信是采用一个同步时钟线，连到收发双方，使收发双方达到完全同步。则IIC通信硬件连接就有四根线VCC、GND、SDA、SCL，SDA是传送数据的，SCL是控制时序的，IIC它是有规矩的，用到这种通信方式，就得按照它的规矩来做，它才会听话，才能完成通信，也称为IIC通信协议。

IIC通信协议：

IIC通信时，需要用到开始信号、停止信号、等待响应信号、发送字节信号、读取字节信号；

开始信号的条件：当SCL为高电平的时候，SDA线上由高到低的跳变被定义为起始条件；

结束信号的条件：当SCL为高电平的时候，SDA线上由低到高的跳变被定义为停止条件；

在设置这些高低电平时，可以在代码中直接设置，这个也叫软件（代码）模拟IIC通信；

IIC通信的起始信号和结束信号：

当 SCL 线为高电平时，SDA 线由高到低的下降沿，为传输开始标志（S）。直到主设备 发出结束信号（P），否则总线状态一直为忙。结束标志（P）规定为，当 SCL 线为高电平 时，SDA 线由低到高的上升沿

IIC通信的应答信号：

IIC 的数据字节定义为 8-bits 长度，对每次传送的总字节数量没有限制。对每一次传输 必须伴有一个应答（ACK）信号，其时钟由主设备提供，而真正的应答信号由从设备发出， 在时钟为高时，通过拉低并保持 SDA 的值来实现。

如果从设备忙，它可以使 SCL 保持在低电平，这会强制使主设备进入等待状态。当从 设备空闲后，并且释放时钟线，原来的数据传输才会继续。

IIC通信的接收数据信号和发送数据信号：

开始标志（S）发出后，主设备会传送一个 7 位的 Slave 地址，并且后面跟着一个第 8 位，称为 Read/Write 位。R/W 位表示主设备是在接受从设备的数据还是在向其写数据。然 后，主设备释放 SDA 线，等待从设备的应答信号（ACK）。每个字节的传输都要跟随有一 个应答位。应答产生时，从设备将 SDA 线拉低并且在 SCL 为高电平时保持低。数据传输总 是以停止标志（P）结束，然后释放通信线路。然而，主设备也可以产生重复的开始信号去 操作另一台从设备，而不发出结束标志。综上可知，所有的 SDA 信号变化都要在 SCL 时钟 为低电平时进行，除了开始和结束标志。

响应信号、发送字节信号，接收字节信号，我都会在代码中实现出来；

此文章并没有实现任何的功能，只是把IIC底层驱动实现出来，你们用到IIC，完全可以把驱动利用起来，利用IIC实现和OLED屏幕的通信。

IIC点C文件：

#include "iic.h"

//总线引脚定义

sbit SDA = P2^1; /\* 数据线 \*/

sbit SCL = P2^0; /\* 时钟线 \*/

//延迟函数

void IIC\_Delay(unsigned char i)//延时一段时间

{

while(i--)

{

\_nop\_();

\_nop\_();

}

}

//开始信号

void IIC\_Start(void)

{

SDA = 1;

SCL = 1;//当SCL为高电平的时候，SDA线上由高到低的跳变被定义为起始条件

IIC\_Delay(5);

SDA = 0;

IIC\_Delay(5);

SCL = 0;

}

//停止信号

void IIC\_Stop(void)

{

SDA = 0;

SCL = 1;//SCL为高电平的时候，SDA线上由低到高的跳变被定义为停止条件；

IIC\_Delay(5);

SDA = 1;

IIC\_Delay(5);

}

//等待应答信号（作用很大）

bit IIC\_WaitAck(void)

{

bit ackbit;

SCL = 1;

IIC\_Delay(5);

ackbit = SDA;

SCL = 0;

IIC\_Delay(5);

return ackbit;

}

//通过I2C总线发送数据

void IIC\_SendByte(unsigned char byt)

{

unsigned char i;

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

{

SCL = 0;

IIC\_Delay(5);

if(byt & 0x80) SDA = 1;

else SDA = 0;

IIC\_Delay(5);

SCL = 1;

byt <<= 1;

IIC\_Delay(5);

}

SCL = 0;

}

//从I2C总线上接收数据

unsigned char IIC\_RecByte(void)

{

unsigned char i, da;

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

{

SCL = 1;

IIC\_Delay(5);

da <<= 1;

if(SDA) da |= 1;

SCL = 0;

IIC\_Delay(5);

}

return da;

}

头文件

#ifndef \_IIC\_H

#define \_IIC\_H

#include "main.h"

void IIC\_Start(void);

bit IIC\_WaitAck(void);

void IIC\_SendAck(bit ackbit);

void IIC\_SendByte(unsigned char byt);

unsigned char IIC\_RecByte(void);

#endif