单片机键盘扫描之状态机实现

作者：原野之狼 日期 2008.1.3 23：37
email:hugui_bj@163.com

一、概述

在编写单片机程序的过程中，键盘作为一种人机接口的实现方式，是很常用的。而一般的实现方法大概有：

1、外接键盘扫描芯片（例如 8279，7279 等等），然后由该芯片来完成去抖、键值读取、中断请求等功能。然后单片机响应中断并读取键值，有的时候也可以采用轮训的方式。

2、如果按键数比较少，那么可以直接将按键接到单片机的 IO 口，然后各按键取逻辑或再送到单片机的中断管脚（对于 51 体系），单片机响应中断后再去读取 IO 口的数据。如果单片机的中断向量比较多（例如 AVR 系列的单片机，每个 IO 都可以作为中断），那么也可以直接把各个按键接到各个具有中断功能的 IO 上面。在中断处理程序中往往需要执行这样一个操作序列：延时一定时间来去抖，如果按键有效那么等待按键释放。

这两种方法都有比较明显的缺陷：

第一种方法需要专门的外围芯片，增加成本，且一般不容易检测按键的按下、释放以及长按键等一些事件。

第二种方法同样不容易检测按键的按下、释放以及长按键等一些事件。且采用软件延时的方式，浪费 CPU 资源，很不可取。

二、扫描式方法

鉴于以上两种方法的缺点，我们可以采用扫描式的方法来判断按键事件。
扫描方法即 CPU 在一定的节奏下去扫描按键数据线上的信号，然后分析并确定按键事件。扫描节奏一般为 20MS

三、状态机

在软件工程中，有这样一个概念，即状态机。
状态机是一个抽象的概念，即把一个过程抽象为若干个状态之间的切换，这些状态之间存在着一定的联系。

举个例子：在操作系统中有一个关于进程调度的经典论述。

即把进程的调度过程抽象为运行、就绪、挂起以及睡眠等状态之间的切换。
各个状态之间在满足一定条件下才能进行切换。

在状态机程序的编写中有两点需要注意：
1、过程的抽象。
2、切换的条件以及如何切换。

四、按键过程的状态机分析

众所周知，一个键按下之后的波形是这样的（假定低有效）：
在有键按下后，数据线上的信号出现一段时间的抖动，然后为低，然后当按键释放时，信号抖动一段时间后变高。当然，在数据线为低或者为高的过程中，都有可能出现一些很窄的干扰信号。

1、状态抽象

因此，我们对这个过程抽象为这么几个阶段：

（1）、空闲状态，即数据线信号为高，这里假定为 S1 状态
（2）、确认真的有键按下的状态，这里假定为 S2 状态。
（3）、键按下的状态，这里假定为 S3 状态。
（4）、确认真的有键释放的状态，这里假定为 S4 状态。

2、状态切换

在 S1 状态，如果信号线为高，那么继续保持 S1 状态，如果信号线为低，那么切换到 S2 状态。
在 S2 状态，如果信号线为高，那么切换到 S1 状态，如果信号线为低，那么切换到 S3 状态，此时表示有了键按下的消息事件，把此事件存入消息队列（如果系统不需要此消息，那么为了简单起见，此时可以不存入这个键按下事件）。
在 S3 状态，如果信号线为高，那么切换到 S4 状态，如果信号线为低,那么保持 S3 状态，并对信号为低这一状态进行计数。
在 S4 状态，如果信号线为高，那么切换到 S1 状态，此时表示有了键释放的消息事件，把此消息存入消息队列（如果系统不需要此消息，那么为了简单起见，此时可以不存入这个键按下事件），同时还需要对信号为低这一状态的计数进行判断，如果大于一定的阈值，那么表示之前是一个长按键消息事件，小于此阈值，则表示之前为一个短按键消息事件。如果信号线为低，则切换到 S3 状态。

五、程序实现

```
/************************************************************
    FileName: type.h
```

```
    Author:原野之狼

    Version :V1.0

    Date: 2008.1.2

    Description:

    History:

      <author>      <time>      <version >      <desc>


************************************************************/

#ifndef _TYPE_H_
#define _TYPE_H_

//type define
#define UINT8 unsigned char
#define INT8 char
#define UINT16 unsigned int
#define INT16 signed int
#define UINT32 unsigned long
#define INT32 signed long
#define FLP32 float

#endif

/************************************************************
    FileName: includes.h

    Author:原野之狼

    Version :V1.0

    Date: 2008.1.2

    Description:

    History:

      <author>      <time>      <version >      <desc>
```

```
************************************************************/

#ifndef _INCLUDES_H_
#define _INCLUDES_H_

//system header files
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>

//user header files
#include "type.h"
#include "queue.h"
#include "key.h"


#define SYSTEM_FREQUENCY_HZ 8000000


#endif


/************************************************************
    FileName: queue.h

    Author:原野之狼

    Version :V1.0

    Date: 2008.1.2

    Description:

    History:

      <author>      <time>      <version >      <desc>


************************************************************/
```

```
#ifndef _QUEUE_H_
#define _QUEUE_H_

#include "type.h"

class CQueue
{
private:
    INT8 *pBuf;
    UINT16 BufLen;
    INT8 *pHead;
    INT8 *pTail;
    UINT16 count;
public:
    UINT8 IsQueueEmpty(void);
    void QueueInit(INT8 *pBuffer,UINT16 len);
    UINT8 AddInQueue(INT8 dat);
    UINT8 RequestFrQueue(INT8 *pDat);
};



#endif


/************************************************************
    FileName: queue.cpp

    Author:原野之狼

    Version :V1.0

    Date: 2008.1.2

    Description:

        management of queue

    History:

      <author>      <time>      <version >      <desc>
```

```
************************************************************/


#include "includes.h"


UINT8 CQueue::IsQueueEmpty(void)
{
    return 1;
}


void CQueue::QueueInit(INT8 *pBuffer,UINT16 len)
{
    pBuf=pBuffer;
    pHead=pBuf;
    pTail=pBuf;
    count=0;
    BufLen=len;
}

UINT8 CQueue::AddInQueue(INT8 dat)
{
    if(count<BufLen)
    {
        (*pTail++)=dat;
        count++;

        if(pTail==(pBuf+BufLen))
        {
            pTail=pBuf;
        }
        return 1;
    }
    else
    {
        return 0;
    }
}
UINT8 CQueue::RequestFrQueue(INT8 *pDat)
{
     if(count)
    {
```

```
        *pDat=(*pHead++);
        if(pHead==(pBuf+BufLen))
            pHead=pBuf;
        count--;
        return 1;
    }
    else
    {
        return 0;
    }
}

/************************************************************
    FileName: key.h

    Author:原野之狼

    Version :V1.0

    Date: 2008.1.2

    Description:

    History:

      <author>      <time>      <version >      <desc>


************************************************************/

#ifndef _KEY_H_
#define _KEY_H_

#include "type.h"

#define GET_KEY1_SIGNAL PINE&(0X01<<0X02)
#define GET_KEY2_SIGNAL PINE&(0X01<<0X03)

#define MSG_DOWN 0X01
#define MSG_UP 0X02
#define MSG_SHORT_CLICK 0X03
#define MSG_LONG_CLICK 0X04
```

```
class CSingleKeyManager
{
private:

    #define KEY_STATE_IDLE 0X01
    #define KEY_STATE_IS_DOWN 0X02
    #define KEY_STATE_DOWN 0X03
    #define KEY_STATE_IS_UP 0X04
    UINT8 m_state;

    UINT8 m_DownCount;

    #define MAX_MSG_LENGTH 0X10
    INT8 m_MsgBuf[MAX_MSG_LENGTH];



    CQueue m_MessageQueue;


public:
    CSingleKeyManager();
    void StateChange(UINT8 LineHighLow);
    UINT8 GetMsg(void);
};


#endif


/*************************************************************
    FileName: key.cpp

    Author:原野之狼

    Version :V1.0

    Date: 2008.1.2

    Description:
        key managerment
        1.scan mode
        2.short clicked
        3.long clicked
```

```
        4.more info refer to file:key.jpeg


    History:

      <author>      <time>      <version >      <desc>

************************************************************/


#include "includes.h"

CSingleKeyManager::CSingleKeyManager()
{
    m_state=KEY_STATE_IDLE;
    m_DownCount=0;
    m_MessageQueue.QueueInit(m_MsgBuf,MAX_MSG_LENGTH);
}

void CSingleKeyManager::StateChange(UINT8 LineHighLow)
{
    switch(m_state)
    {
        case KEY_STATE_IDLE:
        {
            if(LineHighLow)
            {
                asm("nop");
//              WriteLog("1H\n");
            }
            else
            {
                m_state=KEY_STATE_IS_DOWN;
//              WriteLog("1L\n");
            }
        }
            break;
        case KEY_STATE_IS_DOWN:
        {
            if(LineHighLow)
            {
                m_state=KEY_STATE_IDLE;
//              WriteLog("2H\n");
```

```
            }
            else
            {
                m_state=KEY_STATE_DOWN;
//              WriteLog("2L\n");
                if(!m_MessageQueue.AddInQueue(MSG_DOWN))
                {
                    asm("nop");//error
//                  WriteLog("KEY QUEUE ERROR\n");
                }
            }
        }
            break;
        case KEY_STATE_DOWN:
        {
            if(LineHighLow)
            {
                m_state=KEY_STATE_IS_UP;
//              WriteLog("3H\n");
            }
            else
            {
                m_DownCount++;
                if(m_DownCount==255)
                    m_DownCount=255;
//              WriteLog("3L\n");
            }
        }
            break;
        case KEY_STATE_IS_UP:
        {
            if(LineHighLow)
            {
//              WriteLog("4H\n");
                m_state=KEY_STATE_IDLE;
                m_MessageQueue.AddInQueue(MSG_UP);
                if(m_DownCount>(2000/SYSTEM_RHYTHM_MS))
                {
//                  WriteLog("LONG\n");
                    m_MessageQueue.AddInQueue(MSG_LONG_CLICK);
                    m_DownCount=0;
                }
                else
                {
```

```
//                WriteLog("SHORT\n");
                  m_MessageQueue.AddInQueue(MSG_SHORT_CLICK);
                  m_DownCount=0;
               }
            }
            else
            {
//             WriteLog("4L\n");
               m_state=KEY_STATE_DOWN;
            }
         }
            break;
         default:
            asm("nop");
      }
}

UINT8 CSingleKeyManager::GetMsg(void)
{
    INT8 msg;

    if(m_MessageQueue.RequestFrQueue(&msg))
    {
       return msg;
    }
    else
    {
       return 0x00;
    }
}
```

六、应用范例

```
/************************************************************
    FileName:main.cpp

    Author:原野之狼

    Version :V1.0

    Date: 2008.1.2

    Description:
```

```
    History:

      <author>      <time>      <version >      <desc>


************************************************************/
#include “includes.h”

CSingleKeyManager key1


//call this function every 20ms
Void TaskKeyScan(void)
{
       key1.StateChange(GET_KEY1_SIGNAL);
}

UINT8 main(void)
{
    while(1)
    {
        switch(key1.GetMsg())
        {
            //add your message handler here
        }
    }
}
```