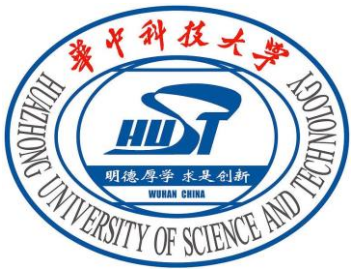


华中科技大学
实验报告
电子器件与电路（二）
嵌入式部分 5
按键动作检测

学院	工程科学学院	
班级	工程科学学院（生医）1701 班	
小组成员	汪能志	U201713082
	周政宏	U201714460
指导老师	钟国辉	

2020 年 9 月 18 日

目录

1. 实验要求

1. 编写程序检测按键，并将检测结果通过数码管反映，同时应该考虑反映按键触发的次数。
2. 考虑实现单击、双击或长按、短按等的检测。

2. 实验原理与流程 方案一

2.1. 按键动作判断

通过定时器触发中断，为了和数码管驱动的显示内容切换错开，这里选择在中断计数模 4 为 2 时，进行按键扫描。因此，每 20ms 会进行一次按键扫描。

由于存在 16 个按键，这里使用 uint16 变量来存储扫描得到的按键值。按键 AN0-AN9 分别存储在第 0-9 位，如果按键被按下，则对应位为 1，否则为 0。如果扫描显示所有按键都没有被按下，则最高位为 1。

通过连续两次的扫描结果，来判断按下或松开按键的动作。如果扫描结果从 0 变成 1，则标记出现一次按下动作；扫描结果从 1 变成 0，则标记出现一次松开动作。

以下以单一按键的按下动作检测为例，来说明按键动作检测与消抖的原理。

由于完全松开时的采样数据为 0，而完全按下时的采样数据为 1，因此在按下按键时，一定会出现至少一次 0 到 1 的转换，因此一定可以检测到按键被按下的动作。

由于扫描周期约 20ms，超出了按键抖动的典型时间（5-10ms），因此从 0 到 1 的转换有以下三种可能情况：

1. 0（松开），0（松开），标记动作，1（按下），1（按下）
2. 0（松开），0（抖动），标记动作，1（按下），1（按下）
3. 0（松开），0（松开），标记动作，1（抖动），1（按下）

这三种情况下，都之后判断为出现一次按下动作。

因此，对于一次按下按键的动作及其附带的抖动，这一方法可以唯一确定一次按键被按下的动作。但是由于采样次数不够多，这一方法不能有效地区分按下和按键被干扰出现抖动。

2.2. 按键组合动作判断

按键组合操作检测中有以下几个需要格外注意的问题：

1. 双击的第一次点击不能被判定为单击；
2. 达到长按阈值时就需要判定长按，而非松开时判定长按；
3. 短按+迅速长按应判定为双击，而非单击+长按；
4. 长按+迅速短按应判定为长按+短按，而非双击。

因此需要引入两个计时器，分别对按下时间（计时器 T1）和松开时间（计时器 T2）进行计时，在第一次按下时，只标记未知事件。按下后一定时间就判定长按；松开后一定时间后判定单击；短时间内第二次按下时，判定双击。

程序的流程图如下（完成动作判定后，需要清零相应变量）：

图 1 方案一流程图

3. 实验原理与流程 方案二

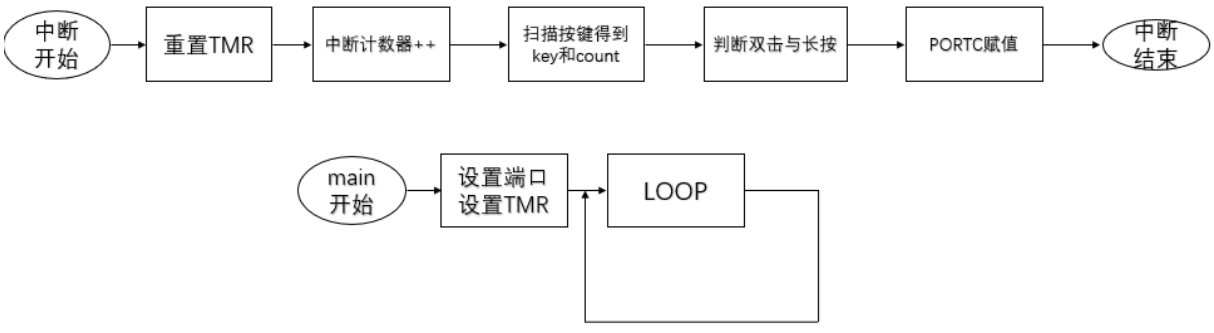


图 2 方案二流程图

如上图所示为主函数的流程，一直在 loop 内循环，当运行语句达到 5000 个周期(即 5ms)后，会进入中断程序，在中断程序的基本流程图中，可以大致分为扫描、判断动作、显示三个模块：

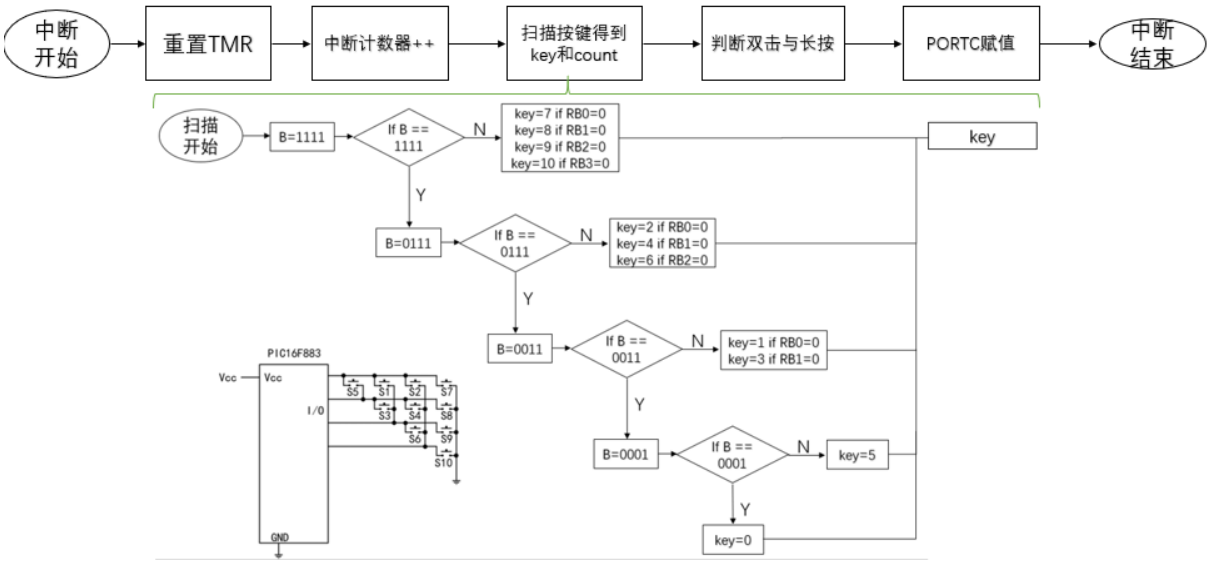
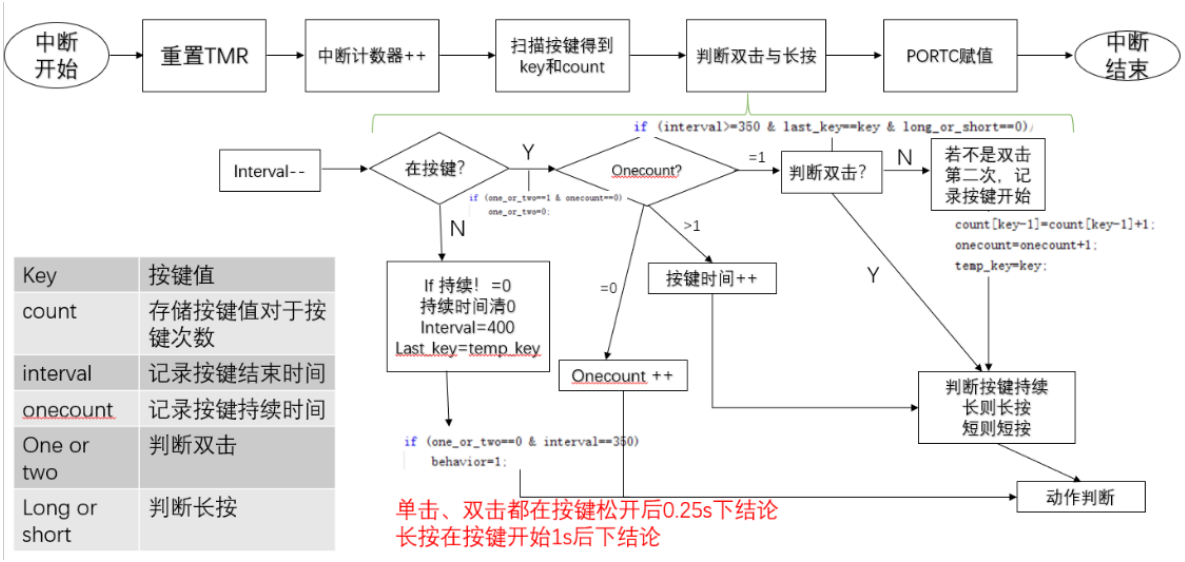


图 3 方案二的中断程序的扫描模块

这个扫描模块与方案一的扫描方案基本一致，故不再赘述，也是对 B 赋四次不同的值然后依次判断按键值与次数。



Key	按键值
count	存储按键值对于按键次数
interval	记录按键结束时间
onecount	记录按键持续时间
One or two	判断双击
Long or short	判断长按

图 4 方案二的按键动作判断流程图

判断动作模块中，我主要使用了上一次按键松开以及按键的持续时间来辅助判断动作，如上图所示，值得注意的是，单击、双击都在按键松开后 0.25s 下结论，长按在按键开始 1s 后下结论。整个的基本思路就是，首先判断是不是在按键，是的话判断上一次扫描是不是也在按，不是的话意味着这是新的一次按下，则进一步判断双击与否；是的话则按键持续时间加一，进一步判断是否长按；如果没有按键则对其进行一个收尾工作，比如记录按键松开时间，判断单击等工作。详见的已经在流程图上列出来了。

4. 实验结果

我们有两种方案分别可以达到所有的实验要求。

4.1. 方案一

第一位显示键值，其中按键 10 显示为 0；
第二位显示按键结果，1 为单击，2 为双击，3 为长按；
第三、四位显示计数值。

4.1.1. 显示按键值与次数

分别短按，双击和长按按键 1，显示结果如下。

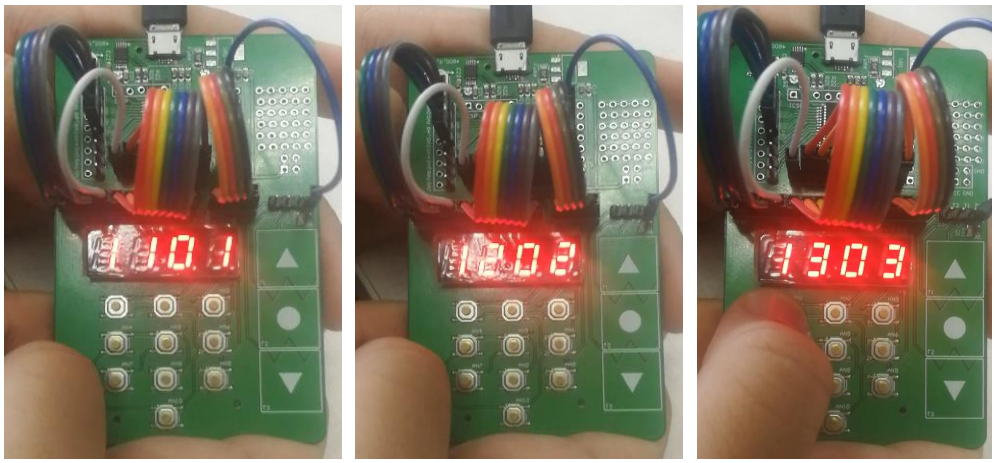


图 5 方案一测试。从左到右分别位短按，双击，长按

4.1.2. 复杂组合动作判断

我们对双击的定义是小于 0.6s 内按两次，对长按的定义是按超过 0.6s。我们对所有按键依次进行短按(单击)、双击、长按及其各种排列组合，均完美符合预期结果。下面展示详细结果：

短按：计数 1 次，在松手后完成动作判断；
长按：计数 1 次，在松手前完成动作判断；
双击：计数 1 次，在第二次点击时完成动作判断；
双击，第二下长按：判断为双击；
长按后迅速短按：判断为长按+短按；
在不同键值双击两次：计数两次，均为短按；
快速点击三次：判断为一次双击，一次单击；
快速点击四次：判断为两次双击。

4.2. 方案二

4.2.1. 显示按键值与次数

图 6 方案二测试

方案二的结果：如上图所示，左边两位表示按键的值，右边两位表示计数，第一张图表示按 2 按了 1 次，第二张图表示按 8 按了 1 次，第三张图表示按 8 按了 2 次。

4.2.2. 扫描的时间间隔

```
Target halted. Stopwatch cycle count = 5000 (5 ms)
Target halted. Stopwatch cycle count = 5000 (5 ms)
Target halted. Stopwatch cycle count = 5000 (5 ms)
Target halted. Stopwatch cycle count = 5000 (5 ms)
Target halted. Stopwatch cycle count = 5000 (5 ms)
Target halted. Stopwatch cycle count = 5000 (5 ms)
Target halted. Stopwatch cycle count = 5000 (5 ms)
Target halted. Stopwatch cycle count = 5000 (5 ms)
```

图 7 方案二扫描周期调试

4. 2. 3. 判断按键双击与长按

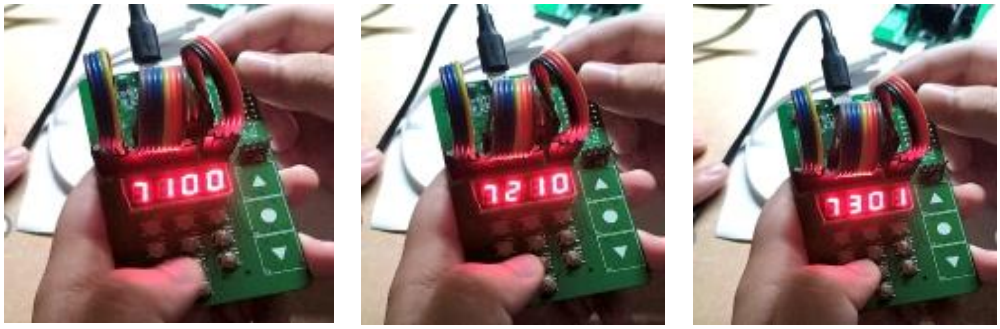


图 8 方案二测试。从左到右以此位短按，双击和长按

方案二的结果：如上图所示，有四个数码管，从左到右分别表示按键值、按键次数、是否为双击、是否为长按。第一张图表示第一次按键 7 为短按(单击)；第二张图表示第二次按键 7 为双击；第三张图表示第三次按键 7 为长按。由于 word 放不了视频，以上图片均是从视频截取下来的。可以实现实验拓展要求。

此外对于双击单击长按等组合，也符合之前方案一所提到的所有要求。

5. 遇到问题

5.1. 方案一

5.1.1. 双击的第一次被判定为单击

松开后一定时间后判定单击；
短时间内第二次按下时，判定双击。

5.1.2. 长按在松开后才会做出判断

按下后一定时间就判定长按。

5.1.3. 做出判定后，同一按键在短时间内无响应

做出判定后立刻清零变量，准备下一次输入。

5.2. 方案二

5.2.1. PORTB 的值改动无效，使得始终判断为一个按键值

经过检查发现没有对 PORTB 进行初始化，在加入了 ANSELB = 0x00;LATB = 0x00;TRISB = 0x00; 之后就能够正常运行了。

5.2.2. 调试时无法按键，不能排除所有情况

只能暂时按照没有按键的情况进行调试，然后对出现的异常情况比如乱码进行代码分析，列出可能的原因，一次次修正调试。

5.2.3. 按键次数统计过快，按一次却记了几次数

我们扫描时间间隔设置为 5ms，扫描过快，而按一次的时间却肯定不止 5ms，因此出现了按一次却多次计数的情况。

一开始我们的解决方案是设置了一个变量，如果两次按键时间间隔过短，那么后面的按键就无效，使用 temp 变量控制，每一次按键后，temp 初始化为 50，然后每次进入中断扫描按键，temp 自减 1(已经为 0 则不减)，只有在 temp 为 0 的情况下，按键才有效，如果 temp 不为 0，则按键无效。这样就要求两次按键的时间间隔为 50*5ms=250ms。

后来在听取了老师的建议后，我们决定采用设置按键持续时间的方式，如果连续几次扫描都判断按键有效，则当成一次按键，并将其计入按键的持续时间，这样设置还有一个好处是可以用来判断长按与短按。

5.2.4. 双击与长按的动作判别

即对双击以及长按的判定，刚开始我们采用类似于不应期的方式解决过快计数的问题，后来在老师的建议下改成设置按键持续时间的方式，但是光有按键持续时间是不能够判断双击的，因此我又设置了一个变量用来存储上一次按键结束后的时间间隔。有了这两个变量以及当前按键值 key，我们可以判断按键双击与长按，由于情况复杂，需要考虑许多情况，所以我调试了很久，最后还是完成了实验。

5.2.5. 长按自动终止

检查发现是记录长按时间的变量设置为 char 类型，最大只有 255，改成 int 类型后恢复正常。

5.2.6. 两个不同键快速按会引发双击

记录上一次的有效键值 key，然后和这一次的比较，如果相同则有可能触发双击。

5.2.7. 先暂定认为是单击，再确定是长按还是双击

改成单击、双击都在按键松开后 0.25s 下结论；长按在按键开始 1s 后下结论。

5.2.8. 没有去抖

必须需要连续两次都进入中断并且判断为有键值则认为才是有效键值。

5.2.9. 双击检测使用的间隔判断是测的两次按键开始，而不是上次结束到现在

将间隔记录改成上次按键的结束到现在(这次按键的开始)，这个需要通过“双击第一下 0.8s 按”检测，计数 1 次，动作判断 2(双击)才是正确的。

6. 实验总结

6.1. 周政宏

按键实验需要考虑的情况非常之多，而且调试时不能用按键也给实验带来了不小的难度，整体实验难度比起前几个实验还是大了不少，一开始我们的思考方向也有问题，后来经过讨论，发现了不少的问题，是在最后，明明只差一个小小的功能就可以了，但是却改了半天，此次实验基本是是对各种“if else”条件的训练，因此一定需要流程图梳理，而且想要微调功能，也是牵一发而动全身，并且不能够轻易的引入变量，变量的数量上升，则程序复杂程度呈几何级数上升，因此一定要明白每个变量的含义，什么条件下应该赋什么值，在什么时候代表什么意义，如果将其改变，那么什么时候将其变回来等等。

总的来说，此次实验让我在更加熟悉嵌入式系统与 C 语言的同时，也让我对流程图的重要性有了更深的认识，收获颇丰。

6.2. 汪能志

按键实验是这一些列的嵌入式实验中第一次涉及到外部信号输入的情况。相对于之前的实验，这次实验中会遇到的问题也更多。首先就是为了节约引脚，而采用了四根输入引脚和地线，五根引脚两两连接来驱动 10 个按键。这导致了需要按照特定的顺序去扫描按键，而且存在误判等情况。按键组合动作的判断也较为复杂，特别是需要考虑复杂的输入情况。一开始我先完成了一个版本，但是只考虑到了用户在知道一些设置（如长按阈值）是，简单输入的情况，出现了较多的问题。而且由于流程图大体已定，对其进行修改难度较大。在课上简短汇报后，老师也提出了很多问题，针对这些问题我又重新绘制了流程图，才较好地完成了实验任务。

通过这次实验，我对于软件工程中流程图和预先明确需求的重要性有了更深刻的认识，对于一些输入信号检测的方法也有了自己的了解。

附录：方案一代码

```
#define TMR0_rst TMR0H = 0xEC, TMR0L = 0x82, PIR0bits.TMR0IF = 0x00
//#define TMR0_rst TMR0H = 0xDF, TMR0L = 0x70, PIR0bits.TMR0IF = 0x00

const unsigned char digital_decode[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66,
0xB6, 0xBE, 0xE0, 0xFE, 0xF6};
const unsigned char LED_select_signal[4] = {0xEE, 0xDD, 0xBB, 0x77};
// Display LEDs Select decode
unsigned char repeat_num;      //
unsigned char frame_repeat_num; // max number of a frame repeat
unsigned char interrupt_count = 0;

unsigned char display_signal[4] = {0xFF, 0xFF, 0xFF, 0xFF};

unsigned char display_cache[128] = {
   28, 0xFC, 238, 0xFC, // LOAD
   29, 0xFC, 238, 0xFC, // LOAD
   29, 0xFD, 238, 0xFC, // LOAD
   29, 0xFD, 239, 0xFC, // LOAD
   29, 0xFD, 239, 0xFD, // LOAD
   182, 158, 182, 0,    // SES

   0xDA, 0xFC, 0xDA, 0xFC,
   0xDA, 0xFC, 0xDA, 0xFC,
   0xFC, 0xF7, 0x60, 0xE0,
   0xFC, 0xF7, 0x60, 0xE0,
   0x02, 0x02, 0x02, 0x02,
   0x00, 0x00, 0x00, 0x00};
unsigned char dis_cache_size = 52; // byte saved in cache
unsigned char sum_frame_num = 13;  // number of frame, in the display
signal cache
//unsigned char display_ctrl = 0b00000001;
unsigned char display_ctrl;
/*
   bit 0
      1   shift 4
      0   shift 1
   bit 1
      1   loop
      0   clear
   bit 2
      0   frame
      1   real time
```

```
 */

#define display_clear_4 display_ctrl = 3, repeat_num = 0
#define display_clear_1 display_ctrl = 2, repeat_num = 0
#define display_loop_4 display_ctrl = 1, repeat_num = 0, frame_num = 0,
frame_start = 0, frame_cache_num = (dis_cache_size >> 2)
#define display_loop_1 display_ctrl = 0, repeat_num = 0, frame_num = 0,
frame_start = 0, frame_cache_num = dis_cache_size
#define display_real_time display_ctrl = 4
unsigned char frame_cache_num;
unsigned char frame_num;   // index of displaying frame
unsigned char frame_start; // start index in display signal cache
// used when loading data from display signal cache to register

unsigned char frame_refresh_enable = 0;
unsigned char display_write_in = 0x00;

inline void frame_switch(void)
{
    if (display_ctrl != 4)
    {
        if (repeat_num == frame_repeat_num)
        {
            repeat_num = 0;

            if (display_ctrl & 0x02)
            {

                if (display_ctrl & 0x01) // shift 4 byte once
                {
                    display_signal[0] = display_cache[0];
                    display_signal[1] = display_cache[1];
                    display_signal[2] = display_cache[2];
                    display_signal[3] = display_cache[3];
                    for (unsigned char j = 0; j < dis_cache_size; j++)
                    {
                        display_cache[j] = display_cache[j + 4];
                    }
                    display_cache[dis_cache_size - 1] = 0x00;
                    display_cache[dis_cache_size - 2] = 0x00;
                    display_cache[dis_cache_size - 3] = 0x00;
                    display_cache[dis_cache_size - 4] = 0x00;
                    dis_cache_size -= 4;
                }
```

```
            else // shift 1 byte once
            {
                display_signal[0] = display_cache[0];
                display_signal[1] = display_cache[1];
                display_signal[2] = display_cache[2];
                display_signal[3] = display_cache[3];

                for (unsigned char j = 0; j < dis_cache_size; j++)
                {
                    display_cache[j] = display_cache[j + 1];
                }
                display_cache[dis_cache_size - 1] = 0x00;

                dis_cache_size--;
            }
        }
        else
        {
            if (frame_num == sum_frame_num)
            {
                frame_num = 0;
                frame_start = 0;
            }
            if (display_ctrl & 0x01)
            {
                display_signal[0] = display_cache[frame_start];
                display_signal[1] = display_cache[frame_start + 1];
                display_signal[2] = display_cache[frame_start + 2];
                display_signal[3] = display_cache[frame_start + 3];
                frame_start += 4;
            }
            else
            {
                frame_start = frame_num;
                for (unsigned char j = 0; j < 4; j++)
                {

                    if (frame_num + j > frame_cache_num - 1)
                    {
                        display_signal[j] = display_cache[frame_start + j
- frame_cache_num];
                    }
                    else
                    {
```

```
                        display_signal[j] = display_cache[frame_start +
j];
                    }
                }
            }
        }
        frame_num++;
    }
  }
}
unsigned char key = 0x00;
unsigned int last_state, current_state, key_press, key_loose;

unsigned char press_count[10] = {0x00};
unsigned char is_press;
unsigned char is_loose;
unsigned char is_double;
unsigned char key_buf;
unsigned char period_press;
unsigned char period_loose;
unsigned char key_action;
//
// 0   nothing
// 1   short
// 2   double
// 3   long

void key_scan(void)
{
   PORTB = 0x0f;
   if (PORTB != 0x0f)
   {
      if (PORTBbits.RB0 == 0)
      {
         key = 7;
      }
      else if (PORTBbits.RB1 == 0)
      {
         key = 8;
      }
      else if (PORTBbits.RB2 == 0)
      {
         key = 9;
      }
```

```
    else if (PORTBbits.RB3 == 0)
    {
        key = 0;
    }
}
else
{
    PORTB = 0x07; //0000 0111
    if (PORTB != 0x07)
    {
        if (PORTBbits.RB0 == 0)
        {
            key = 2;
        }
        else if (PORTBbits.RB1 == 0)
        {
            key = 4;
        }
        else if (PORTBbits.RB2 == 0)
        {
            key = 6;
        }
    }
    else
    {
        PORTB = 0x03; //0000 0011
        if (PORTB != 0x03)
        {
            if (PORTBbits.RB0 == 0)
            {
                key = 1;
            }
            else
            {
                key = 3;
            }
        }
        else
        {
            PORTB = 0x01;     //0000 0001
            if (PORTB != 0x01) //0000 0001
            {
                key = 5;
            }
```

```
            else
            {
                key = 15;
            }
        }
    }
}

void key_action_scan(void)
{
    current_state = 0x00;
    key_scan();
    if (is_press)
    {
        period_press++;
    }
    if (is_loose)
    {
        period_loose++;
    }

    current_state = (1 << key);
    key_press |= (~last_state) & current_state;
    key_loose |= last_state & (~current_state);
    last_state = current_state;
}

void __interrupt() isr(void)
{
    // reset TMR0
    TMR0_rst;
    interrupt_count++;
    // clear PORTC
    PORTC = 0x00;
    /************display part************/
    PORTA = LED_select_signal[interrupt_count & 0x03];
    PORTC = display_signal[interrupt_count & 0x03];

    if (!(interrupt_count & 0x03))
    {
        repeat_num++;
        frame_switch();
    }
```

```
    if ((interrupt_count & 0x03) == 2)
    {
        key_action_scan();
    }
}

void display_cache_push_back()
{
    //    if (display_ctrl & 0x02)
    // can only push display data in clear mode

    display_cache[dis_cache_size] = display_write_in;
    dis_cache_size++;
    display_write_in = 0x00;
}

void display_cache_clear(void)
{
    for (unsigned short j = 0; j < 128; j++)
    {
        display_cache[j] = 0x00;
    }
    dis_cache_size = 0;
}

void port_init(void)
{
    // init PORTC
    ANSELA = 0x00;
    LATA = 0x00;
    TRISA = 0x00;
    ANSELB = 0x00;
    LATB = 0x00;
    TRISB = 0x00;
    ANSELC = 0x00;
    LATC = 0x00;
    TRISC = 0x00;
}

void int_tmr_init(void)
{
    // init interrupt
    INTCONbits.GIE = 1;
```

```
    // global interrupt    enable
    INTCONbits.PEIE = 0;
    // peripheral interrupt disable
    INTCONbits.INTEDG = 1;
    // interrupt          rising edge
    PIE0bits.TMR0IE = 1;
    // Timer0 interrupt    enable

    // init TMR0
    T0CON0 = 0xD0;
    T0CON1 = 0x40;
    TMR0_rst;
}

void start_disp(void)
{
    frame_repeat_num = 15;
    display_clear_4;
    while (dis_cache_size)
    {
    };
    display_cache_clear();
}

void key_init(void)
{
    key = 0;
    last_state = 0;
    current_state = 0;
    key_press = 0;
    key_loose = 0;
    is_press = 0;
    is_loose = 0;
    is_double = 0;
    period_press = 0;
    period_loose = 0;
}

void setup(void)
{
    port_init();
    int_tmr_init();
    start_disp();
    key_init();
```

```
    display_real_time;
}

void loop(void)
{

    if (key_press << 1)
    {
        key_press = 0;
        is_press = 1;
        is_loose = 0;
        period_press = 0;

        if ((period_loose > 30) || (key_buf != key) || key_action != 0)
        {
            // single click or quick click another key
            period_loose = 0;
            key_action = 0;
            key_buf = key;
            display_signal[0] = digital_decode[key_buf];
            press_count[key_buf]++;
                }
        else
        {
            // double click
            if (key_action == 0)
            {
                key_action = 2;
            }
        }
    }
    if (key_loose << 1)
    {
        key_loose = 0;
        is_press = 0;
        is_loose = 1;
        period_loose = 0;
    }

    if (key_action == 0 && period_loose > 30)
    {
        key_action = 1;
        period_loose = 0;
    }
```

```
    if (period_press > 30)
    {
        if (key_action == 0)
        {
            key_action = 3;
            period_loose = 0;
        }
    }

    if (press_count[key_buf] == 100)
    {
        press_count[key_buf] = 0x00;
    }
    display_signal[2] = digital_decode[press_count[key_buf] / 10];
    display_signal[3] = digital_decode[press_count[key_buf] % 10];
    if (key_action != 0)
    {
        display_signal[1] = digital_decode[key_action];
    }
    else
    {
        display_signal[1] = 0x00;
    }
}

void main(void)
{
    setup();
    while (1)
    {
        loop();
    }
    return;
}
```

附录：方案二代码

```
const unsigned char digital_decode[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66,
0xB6, 0xBE, 0xE0, 0xFE, 0xF6};
const unsigned int LED_select_signal[4] = {0xEE, 0xDD, 0xBB, 0x77};
// Display LEDs Select decode

unsigned char interrupt_count = 0;

unsigned char display_signal[4] = {0x00, 0x00, 0x00, 0x00};
unsigned char count[10] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00}; //1 2 3 4 5 6 7 8 9 10
unsigned char long_or_short = 0x00;
unsigned char one_or_two = 0x00;
unsigned char behavior = 0x00;
unsigned char key = 0x00;
unsigned char last_key = 0x00;
unsigned char temp_key = 0x00;
unsigned int interval = 0;
unsigned int onecount = 0;

void scan(void)
{
    PORTB = 0x0f;
    if (PORTB != 0x0f)
    {
        if (PORTBbits.RB0 == 0)
            key = 7;
        else if (PORTBbits.RB1 == 0)
            key = 8;
        else if (PORTBbits.RB2 == 0)
            key = 9;
        else if (PORTBbits.RB3 == 0)
            key = 10;
    }
    else
    {
        PORTB = 0x07; //0000 0111
        if (PORTB != 0x07)
        {
            if (PORTBbits.RB0 == 0)
                key = 2;
            else if (PORTBbits.RB1 == 0)
                key = 4;
```

```
            else if (PORTBbits.RB2 == 0)
                key = 6;
        }
        else
        {
            PORTB = 0x03; //0000 0011
            if (PORTB != 0x03)
            {
                if (PORTBbits.RB0 == 0)
                    key = 1;
                else
                    key = 3;
            }
            else
            {
                PORTB = 0x01;     //0000 0001
                if (PORTB != 0x01) //0000 0001
                    key = 5;
                else
                    key = 0;
            }
        }
    }

    if (interval != 0)
        interval = interval - 1;

    if (key != 0)
    {
        if (one_or_two == 1 & onecount == 0)
            one_or_two = 0;
        if (onecount > 1) //key is continued
        {
            onecount = onecount + 1;
            if (onecount > 200)
                long_or_short = 1;
            else
                long_or_short = 0;
        }
        else if (onecount == 1) //key is started
        {
            if (interval >= 350 & last_key == key & long_or_short == 0)
//key is too closed
                one_or_two = 1;
```

```
            if (one_or_two != 1) //key is not too closed
            {
                count[key - 1] = count[key - 1] + 1;
                onecount = onecount + 1;
                temp_key = key;
            }

            if (onecount > 200)
                long_or_short = 1;
            else
                long_or_short = 0;
        }
        else if (onecount == 0) //prevent shake
        {
            onecount = onecount + 1;
        }
    }
    else
    {
        if (onecount > 0) //key is ended. or clean the shake
        {
            onecount = 0;
            interval = 400;
            last_key = temp_key;
        }

        if (one_or_two == 0 & interval == 350)
            behavior = 1;
    }

    if (one_or_two == 1)
        behavior = 2;
    else if (long_or_short == 1)
        behavior = 3;
    else if (behavior == 1)
        behavior = 1;
    else
        behavior = 0;
}
void __interrupt() isr(void)
{
    // reset TMR0
    TMR0_rst;
```

```
    interrupt_count++;
    // clear PORTC
    PORTC = 0x00;
    PORTA = LED_select_signal[interrupt_count & 0x03];

    scan();
    if (key == 0)
    {
        display_signal[3] = digital_decode[behavior];
        PORTC = display_signal[interrupt_count & 0x03];
    }
    else if (key == 10)
    {
        display_signal[0] = digital_decode[1];
        display_signal[1] = digital_decode[0];
        display_signal[2] = digital_decode[count[key - 1] % 10];
        display_signal[3] = digital_decode[behavior];
        PORTC = display_signal[interrupt_count & 0x03];
    }
    else
    {
        display_signal[0] = digital_decode[0];
        display_signal[1] = digital_decode[key];
        display_signal[2] = digital_decode[count[key - 1] % 10];
        display_signal[3] = digital_decode[behavior];
        PORTC = display_signal[interrupt_count & 0x03];
    }
    PORTC = display_signal[interrupt_count & 0x03];
}

void port_init(void)
{
    // init PORTC
    ANSELA = 0x00;
    LATA = 0x00;
    TRISA = 0x00;
    ANSELB = 0x00;
    LATB = 0x00;
    TRISB = 0x00;
    ANSELC = 0x00;
    LATC = 0x00;
    TRISC = 0x00;
}
```

```
void int_tmr_init(void)
{
    // init interrupt
    INTCONbits.GIE = 1;
    // global interrupt    enable
    INTCONbits.PEIE = 0;
    // peripheral interrupt disable
    INTCONbits.INTEDG = 1;
    // interrupt          rising edge
    PIE0bits.TMR0IE = 1;
    // Timer0 interrupt    enable

    // init TMR0
    T0CON0 = 0xD0;
    T0CON1 = 0x40;
    TMR0_rst;
}

void setup(void)
{
    port_init();
    int_tmr_init();
}

void loop(void)
{
}

void main(void)
{
    setup();
    while (1)
    {
        loop();
    }
    return;
}
```