任务5设计

按键行为

根据多次测试,按键单次约在100-200ms的范围内,以此作为基础,进行之后的逻辑设计

根据我的逻辑分析仪(80MHz)观察,并没有发现比较明显的抖动,参考老师的数据,可以将抖动时长控制在5ms,

扫描信息

按下的按键	第一次扫描(1111)	第二次扫描(1110)	第三次扫描(1101)	第四次扫描(1011)	第五次扫描(0111)
0	1111	1110	1101	1011	0111
1	1111	1100	1100	1011	0111
2	1111	1010	1101	1010	0111
3	1111	1110	1001	1001	0111
4	1111	0110	1101	1011	0110
5	1111	1110	0101	1011	0101
6	1111	1110	1101	0011	0011
7	1110	1110	1100	1010	0110
8	1101	1100	1101	1001	0101
9	1011	1010	1001	1011	0011
10	0111	0110	0101	0011	0111

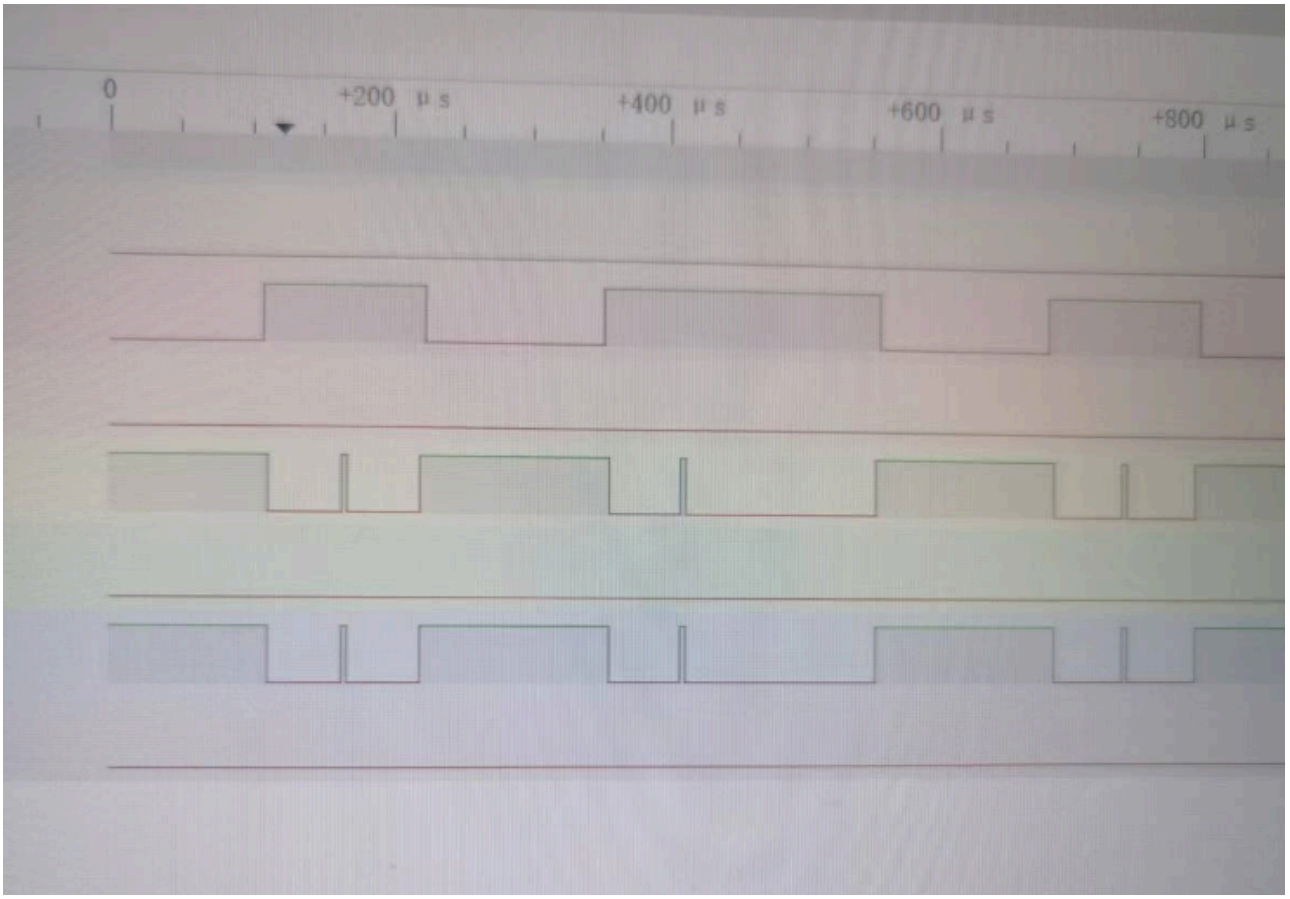
按下的按键	第1次扫描(1110)	第2次扫描(1101)	第3次扫描(1011)	第4次扫描(0111)
10	0110	0101	0011	0111
4	0110	1101	1011	0110
9	1010	1001	1011	0011

按下的按键	第1次扫描(1110)	第2次扫描(1101)	第3次扫描(1011)	第4次扫描(0111)
2	1010	1101	1010	0111
1	1100	1100	1011	0111
8	1100	1101	1001	0101
5	1110	0101	1011	0101
3	1110	1001	1001	0111
7	1110	1100	1010	0110
6	1110	1101	0011	0011
0	1110	1101	1011	0111

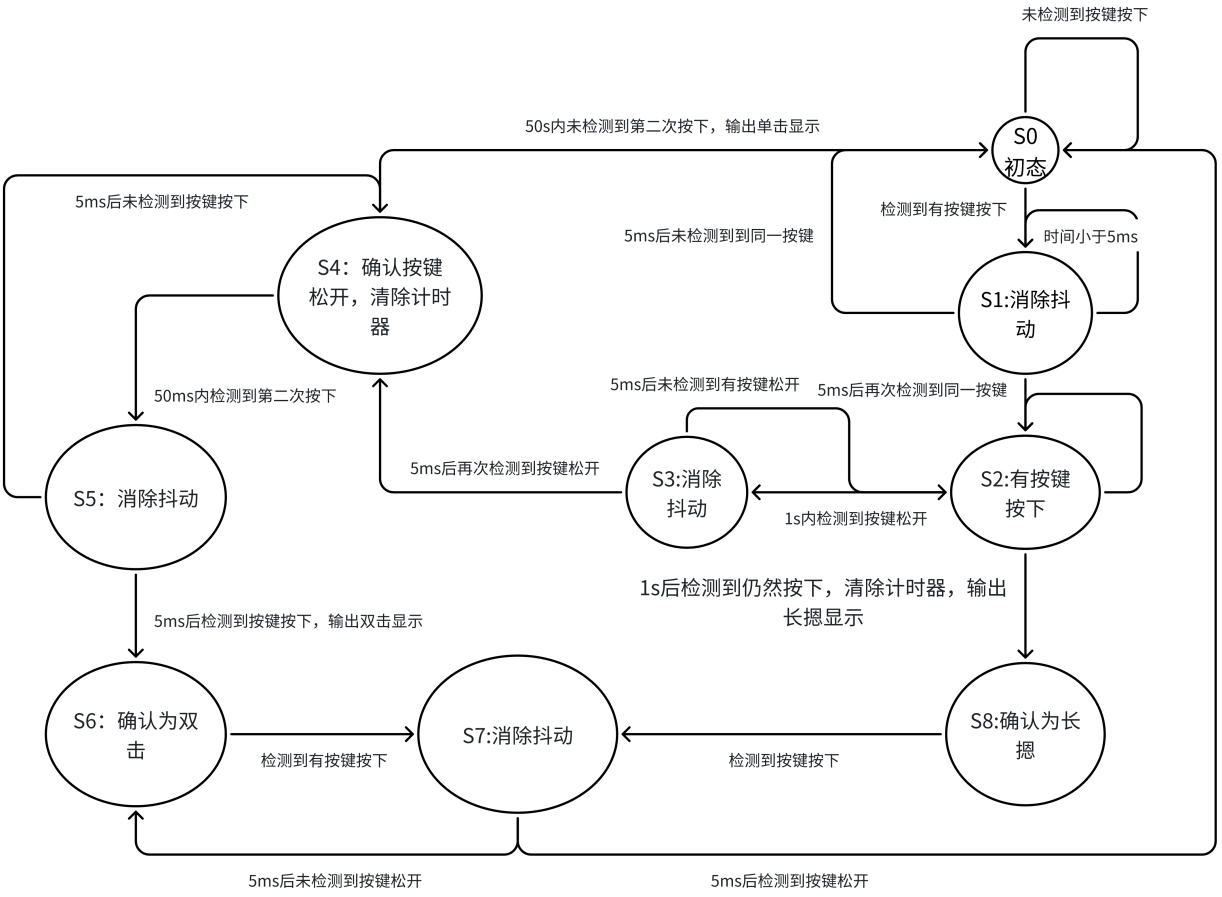
实现方式

前两次分支使用的是查找表,为了加快逻辑判断的速度

利用逻辑分析仪快速调试,快速定位问题



按键类型判断



使用状态机的办法在编程是能做到非常清晰。

代码

```
#include <xc.inc>

/** @brief 配置和初始化段 */
psect   init, class=CODE, delta=2
psect   end_init, class=CODE, delta=2
psect   powerup, class=CODE, delta=2
psect   cinit, class=CODE, delta=2
psect   functab, class=ENTRY, delta=2
psect   idloc, class=IDLOC, delta=2, noexec
psect   eeprom_data, class=EEDATA, delta=2, space=3, noexec
psect   intentry, class=CODE, delta=2
psect   reset_vec, class=CODE, delta=2

/** @brief 全局定义 */
global _main, reset_vec, start_initialization

```

```
/** @brief 配置设置 */
psect config, class=CONFIG, delta=2
    dw    0xDFEC
    dw    0xF7FF
    dw    0xFFBF
    dw    0xEFFE
    dw    0xFFFF

/** @brief 复位向量，跳转到主函数 */
psect reset_vec
reset_vec:
    ljmp    _main

/** @brief 初始化段 */
psect cinit
start_initialization:

/** @brief 公共变量 */
psect CommonVar, class=COMMON, space=1, delta=1
/** @brief 显示数据,display子函数的参数
 *  @details 数据结构从高位到低位分别为：数码管1、数码管2、数码管3、数码管4
 *           每个数码管的数据结构为：8位,从0到15分别对应0到F
 */
display_data:
    ds 4h        ; 分配4个字节的空间，并初始化为0
/**
 * @brief 译码后的数据
 */
display_data_decode: ds 4h//TODO:计划移入bank0
/**
 * @brief 位选数据
 */
digit_select: ds 1h//TODO:计划移入bank0
/**
 * @brief key_data
 */
key_data: ds 1h
/**
 * @brief 公共索引变量
 */
index: ds 1h
```

```
index_1: ds 1h
delay: ds 1h
state: ds 1h
last_button: ds 1h

OPTION_NUM:ds 1h
psect BANK0Var, class=BANK0, space=1, delta=1


/** @brief 中断服务程序向量 */
psect intentry
intentry:
    call display_without_encode
    banksel PIR0
    bcf PIR0, 5
    MOVLW 0x01
    SUBWF state,0
    BTFSS STATUS,2
    goto if_state2
    goto delay_5ms

    if_state2:         ;用于判断是否为state2
    MOVLW 0x02
    SUBWF state,0
    BTFSS STATUS,2
    goto if_state3
    goto delay_50ms

    if_state3:        ;用于判断是否为state3
    MOVLW 0x03
    SUBWF state,0
    BTFSS STATUS,2
    goto if_state4
    goto delay_5ms

    if_state4:
    MOVLW 0x04
    SUBWF state,0
    BTFSS STATUS,2
    goto if_state5
    goto delay_50ms

```

```
100     if_state5:
101     MOVLW 0x05
102     SUBWF state,0
103     BTFSS STATUS,2
104     goto if_state7
105     goto delay_5ms
106
107     if_state7:
108     MOVLW 0x07
109     SUBWF state,0
110     BTFSS STATUS,2
111     goto theend
112     goto delay_5ms
113
114
115     delay_5ms:        ;用于延迟5ms
116     MOVLW 0x01
117     ADDWF delay,1
118     MOVLW 0x03        ;用于判断是否为s3
119     SUBWF state,0
120     BTFSS STATUS,2
121     goto if_is_s5        ;用于判断是否为s5
122     goto delay_50ms
123
124     if_is_s5:
125     MOVLW 0x05
126     SUBWF state,0
127     BTFSS STATUS,2
128     goto theend
129     goto delay_50ms
130
131     delay_50ms:        ;每过1ms index_1++，每过50ms index++
132     MOVLW 00010111B
133     SUBWF index_1,0
134     BTFSS STATUS,2
135     goto not_50ms
136     goto is_50ms
137     not_50ms:
138     MOVLW 0x01
139     ADDWF index_1,1
140     goto theend
141     is_50ms:
```

```
    CLRF index_1
    MOVLW 0x01
    ADDWF index,1
    goto theend


    theend:
    retfie
/**
 * @brief 宏定义数码管显示
 */
#define ZERO_DIS   0x3F
#define ONE_DIS    0x06
#define TWO_DIS    0x5B
#define THREE_DIS  0x4F
#define FOUR_DIS   0x66
#define FIVE_DIS   0x6D
#define SIX_DIS    0x7D
#define SEVEN_DIS  0x07
#define EIGHT_DIS  0x7f
#define NINE_DIS   0x6F
#define A_DIS      0x77
#define B_DIS      0x7C
#define C_DIS      0x39
#define D_DIS      0x5E
#define E_DIS      0x79
#define F_DIS      0x71
#define ZERO_DIS_DP   0xBF
#define ONE_DIS_DP    0x86
#define TWO_DIS_DP    0xDB
#define THREE_DIS_DP  0xCF
#define FOUR_DIS_DP   0xE6
#define FIVE_DIS_DP   0xED
#define SIX_DIS_DP    0xFD
#define SEVEN_DIS_DP  0x87
#define EIGHT_DIS_DP  0xFF
#define NINE_DIS_DP   0xEF
#define A_DIS_DP      0xF7
#define B_DIS_DP      0xFC
#define C_DIS_DP      0xB9
#define D_DIS_DP      0xDE
#define E_DIS_DP      0xF9
```

```
#define F_DIS_DP      0xF1
#define G_DIS         0x7D
#define G_DIS_DP      0xFD
#define H_DIS         0x76
#define H_DIS_DP      0xF6
#define I_DIS         0x06
#define I_DIS_DP      0x86
#define J_DIS         0x1E
#define J_DIS_DP      0x9E
#define K_DIS         0x76
#define K_DIS_DP      0xF6
#define L_DIS         0x38
#define L_DIS_DP      0xB8
#define M_DIS         0x55
#define M_DIS_DP      0xD5
#define N_DIS         0x37
#define N_DIS_DP      0xB7
#define O_DIS         0x3F
#define O_DIS_DP      0xBF
#define P_DIS         0x73
#define P_DIS_DP      0xF3
#define Q_DIS         0x67
#define Q_DIS_DP      0xE7
#define R_DIS         0x33
#define R_DIS_DP      0xB3
#define S_DIS         0x6D
#define S_DIS_DP      0xED
#define T_DIS         0x78
#define T_DIS_DP      0xF8
#define U_DIS         0x3E
#define U_DIS_DP      0xBE
#define V_DIS         0x3E
#define V_DIS_DP      0xBE
#define W_DIS         0x3E
#define W_DIS_DP      0xBE
#define X_DIS         0x76
#define X_DIS_DP      0xF6
#define Y_DIS         0x6E
#define Y_DIS_DP      0xEE
#define Z_DIS         0x5B
#define Z_DIS_DP      0xDB
#define BLANK_DIS     0x00
```

```
226 #define BLANK_DIS_DP  0x80
227 #define DASH_DIS      0x40
228 #define DASH_DIS_DP   0xC0
229 #define UNDERLINE_DIS 0x08
230 #define UNDERLINE_DIS_DP 0x88
231 #define EQUAL_DIS     0x48
232 #define EQUAL_DIS_DP  0xC8
233 #define PLUS_DIS      0x70
234 #define PLUS_DIS_DP   0xF0
235 #define ASTERISK_DIS  0x37
236 #define ASTERISK_DIS_DP 0xB7
237 #define SLASH_DIS     0x5B
238 #define SLASH_DIS_DP  0xDB
239 #define BACKSLASH_DIS 0x6E
240 #define BACKSLASH_DIS_DP 0xEE
241 #define PERCENT_DIS   0x72
242 #define PERCENT_DIS_DP 0xF2
243 #define LESS_DIS      0x71
244 #define LESS_DIS_DP   0xF1
245 #define GREATER_DIS   0x76
246 #define GREATER_DIS_DP 0xF6
247 #define QUESTION_DIS  0x53
248 #define QUESTION_DIS_DP 0xD3
249 #define EXCLAMATION_DIS 0x06
250 #define EXCLAMATION_DIS_DP 0x86
251
252 /** @brief 宏函数
253  *  @param A, B, C, D 数据
254  *  @details 将A, B, C, D分别写入display_data的第1, 2, 3, 4个字节
255  */
256     print0x MACRO param1,param2,param3,param4
257     ; 宏定义开始
258     MOVLW param1
259     MOVWF display_data
260     MOVLW param2
261     MOVWF display_data+1
262     MOVLW param3
263     MOVWF display_data+2
264     MOVLW param4
265     MOVWF display_data+3
266     ; 译码
267     call display_encode
```

```
268     endm
269 
270 /**
271  * @brief 显示不定义画面一帧
272  */
273 printdraw MACRO param1,param2,param3,param4
274     ; 宏定义开始
275     MOVLW param1
276     MOVWF display_data_decode//TODO:计划移入bank0,如果移入,请检查
    这里
277     MOVLW param2
278     MOVWF display_data_decode+1
279     MOVLW param3
280     MOVWF display_data_decode+2
281     MOVLW param4
282     MOVWF display_data_decode+3
283     call display_one_frame_loop
284 endm
285 
286 /** @brief 显示子程序
287  *  @param display_data 4个字节的显示数据
288  *  @details 数据结构从高位到低位分别为：数码管1、数码管2、数码管3、数码
    管4
289  *           每个数码管的数据结构为：4位,从0到15分别对应0到F
290  */
291 
292 psect display, class=CODE, delta=2
293 global display_0,display_without_encode,display_one_frame_loop
294 display_0:
295     // 软件译码
296     call    display_encode
297     // 位选切换 TODO:优化位选切换
298     // 如果是4(0b0111),则切换到1
299 display_without_encode:
300     /**
301      * @brief 显示数据显示位切换程序
302      * @details 位选切换程序,使用BRW查表加goto实现
303      * 如果是4(0b0111 = 7),则goto display_1
304      * 如果是1(0b1110 = 14),则goto display_2
305      * 如果是2(0b1101 = 13),则goto display_3
306      * 如果是3(0b1011 = 11),则goto display_4
307      */
```

```
    //从digit_select加载数据到w
    movf    digit_select, w
    // 取出位选数据
    andlw   0x0F
    // 位选切换
    BRW
    // 位选切换表
    goto    display_1//0
    goto    display_1//1
    goto    display_1//2
    goto    display_1//3
    goto    display_1//4
    goto    display_1//5
    goto    display_1//6
    goto    display_1//7
    goto    display_1//8
    goto    display_1//9
    goto    display_1//10
    goto    display_4//11
    goto    display_1//12
    goto    display_3//13
    goto    display_2//14
    goto    display_1//15
    return
display_1://将位选切换到1
    banksel PORTC
    movlw   0b1110
    movwf   digit_select
    //从display_data_decode中取出数据
    movf    display_data_decode, w
    movwf   PORTC
    movf    digit_select, w
    movwf   PORTA
    return
display_2://将位选切换到2
    banksel PORTC
    movlw   0b1101
    movwf   digit_select
    //从display_data_decode+1中取出数据
    movf    display_data_decode+1, w
    movwf   PORTC
    movf    digit_select, w
```

```
    movwf   PORTA
    return
display_3://将位选切换到3
    banksel PORTC
    movlw   0b1011
    movwf   digit_select
    //从display_data_decode+2中取出数据
    movf    display_data_decode+2, w
    movwf   PORTC
    movf    digit_select, w
    movwf   PORTA
    return
display_4://将位选切换到4
    banksel PORTC
    movlw   0b0111
    movwf   digit_select
    //从display_data_decode+3中取出数据
    movf    display_data_decode+3, w
    movwf   PORTC
    movf    digit_select, w
    movwf   PORTA
    return

display_one_frame_loop:
    //初始化index
    MOVLW 0
    MOVWF index_1
//利用index,循环256次
outer_loop:
    call display_without_encode
    MOVLW 0
    MOVWF index
inner_loop:
    DECFSZ index, 1   ; 将 index 减1，如果结果为零则跳过下一个指令
    goto inner_loop   ; 跳转到 inner_loop 标签处继续内层循环

    // 内层循环结束后继续外层循环
    DECFSZ index_1, 1      ; 将 y 减1，如果结果为零则跳过下一个指令
    goto outer_loop  ; 跳转到 outer_loop 标签处继续外层循环
    return

/**
```

```
392  * @breif 译码子程序
393  * @param display_data 2个字节的显示数据
394  * @details
395  */
396 display_encode:
397     ; 取出第一个字节
398     movf    display_data, 0
399     call    display_encode_h//TODO:计划移入bank0,如果移入,请检查这
    里
400     movwf   display_data_decode
401 
402     ; 取出第二个字节
403     movf    display_data + 1, 0
404     call    display_encode_h//TODO:计划移入bank0,如果移入,请检查这
    里
405     movwf   display_data_decode + 1
406 
407     ; 取出第三个字节
408     movf    display_data + 2, 0
409     call    display_encode_h//TODO:计划移入bank0,如果移入,请检查这
    里
410     movwf   display_data_decode + 2
411 display_encode_4:
412     ; 取出第四个字节
413     movf    display_data + 3, 0
414     call    display_encode_h//TODO:计划移入bank0,如果移入,请检查这
    里
415     movwf   display_data_decode + 3
416 
417     return
418 
419 /**
420  * @brief 译码子程序,5位译码
421  * @param 从display_data中取出数据一个半字节
422  * @details 从display_data中取出数据一个半字节并进行译码，返回数码管显
    示编码
423  */
424 display_encode_h:
425     ; 只保留低5位
426     andlw   0x1F
427     BRW
428     ; 数码管显示编码表
```

```
429     retlw       ZERO_DIS
430     retlw       ONE_DIS
431     retlw       TWO_DIS
432     retlw       THREE_DIS
433     retlw       FOUR_DIS
434     retlw       FIVE_DIS
435     retlw       SIX_DIS
436     retlw       SEVEN_DIS
437     retlw       EIGHT_DIS
438     retlw       NINE_DIS
439     retlw       A_DIS
440     retlw       B_DIS
441     retlw       C_DIS
442     retlw       D_DIS
443     retlw       E_DIS
444     retlw       F_DIS
445     retlw       ZERO_DIS_DP
446     retlw       ONE_DIS_DP
447     retlw       TWO_DIS_DP
448     retlw       THREE_DIS_DP
449     retlw       FOUR_DIS_DP
450     retlw       FIVE_DIS_DP
451     retlw       SIX_DIS_DP
452     retlw       SEVEN_DIS_DP
453     retlw       EIGHT_DIS_DP
454     retlw       NINE_DIS_DP
455     retlw       A_DIS_DP
456     retlw       B_DIS_DP
457     retlw       C_DIS_DP
458     retlw       D_DIS_DP
459     retlw       E_DIS_DP
460     retlw       F_DIS_DP
461     return
462 /**
463  * @brief 键盘相关函数区域
464  */
465 psect keyboard, class=CODE, delta=2
466 global keyboard_scan
467 /**
468  * @brief 键盘扫描函数,扫描一次
469  * @param key_data 键盘数据,在每次扫描后更新
```

```
 * @details 键盘数据结构为：1个字节，表示0-10个按键的状态,为0表示没有按下，为1表示按下1
 *          端口定义：PORTB0-1为键盘端口,采用全扫描的方式
| 按下的按键 | 第1次扫描(1110) | 第2次扫描(1101) | 第3次扫描(1011) | 第4次扫描(0111) |
| ---------- | --------------- | --------------- | --------------- | --------------- |
| 10         | 0110            | 0101            | 0011            | 0111            |
| 4          | 0110            | 1101            | 1011            | 0110            |
| 9          | 1010            | 1001            | 1011            | 0011            |
| 2          | 1010            | 1101            | 1010            | 0111            |
| 1          | 1100            | 1100            | 1011            | 0111            |
| 8          | 1100            | 1101            | 1001            | 0101            |
| 5          | 1110            | 0101            | 1011            | 0101            |
| 3          | 1110            | 1001            | 1001            | 0111            |
| 7          | 1110            | 1100            | 1010            | 0110            |
| 6          | 1110            | 1101            | 0011            | 0011            |
| 0          | 1110            | 1101            | 1011            | 0111            |
 */
keyboard_scan:
    // 1110扫描
    call scan_1110
    BRW ;根据扫描结果跳转
    return//0000
    return//0001
    return//0010
    return//0011
    return//0100
    return//0101
    goto scan_1101_0110//0110
    return//0111
```

```
    return//1000
    return//1001
    goto scan_1101_1010//1010
    return//1011
    goto scan_1101_1100//1100
    return//1101
    goto scan_1101_1110//1110
    return//1111
psect scan_0110_xxxx, class=CODE, delta=2
scan_1101_0110:
    // 1101扫描
    call scan_1101
    BRW ;根据扫描结果跳转
    return//0000
    return//0001
    return//0010
    return//0011
    return//0100
    goto scan_1011_0110_0101//0101
    return//0110
    return//0111
    return//1000
    return//1001
    return//1010
    return//1011
    return//1100
    goto scan_1011_0110_1101//1101
    return//1110
    return//1111
scan_1011_0110_0101:
    // 1011扫描
    call scan_1011
    // 判断w是否为0b0011
    xorlw 0b0011
    // 如果不等于0b0011,则return
    btfss STATUS, 2
    return
    // 0111扫描
    call scan_0111
    // 判断w是否为0b0111
    xorlw 0b0111
    // 如果不等于0b0111,则return
```

```
    btfss STATUS, 2
    return
    // 更新key_data
    movlw 10
    movwf key_data
    return
scan_1011_0110_1101:
    // 1011扫描
    call scan_1011
    // 判断w是否为0b1011
    xorlw 0b1011
    // 如果不等于0b1011,则return
    btfss STATUS, 2
    return
    // 0111扫描
    call scan_0111
    // 判断w是否为0b0110
    xorlw 0b0110
    // 如果不等于0b0110,则return
    btfss STATUS, 2
    return
    // 更新key_data
    movlw 4
    movwf key_data
    return
psect scan_1010_xxxx, class=CODE, delta=2
scan_1101_1010:
    // 1101扫描
    call scan_1101
    BRW ;根据扫描结果跳转
    return//0000
    return//0001
    return//0010
    return//0011
    return//0100
    return//0101
    return//0110
    return//0111
    return//1000
    goto scan_1011_1010_1001//1001
    return//1010
    return//1011
```

```
    return//1100
    goto scan_1011_1010_1101//1101
    return//1110
    return//1111
scan_1011_1010_1001:
    // 1011扫描
    call scan_1011
    // 判断w是否为0b1011
    xorlw 0b1011
    // 如果不等于0b1011,则return
    btfss STATUS, 2
    return
    // 0111扫描
    call scan_0111
    // 判断w是否为0b11
    xorlw 0b11
    // 如果不等于0b11,则return
    btfss STATUS, 2
    return
    // 更新key_data
    movlw 9
    movwf key_data
    return
scan_1011_1010_1101:
    // 1011扫描
    call scan_1011
    // 判断w是否为0b1010
    xorlw 0b1010
    // 如果不等于0b1010,则return
    btfss STATUS, 2
    return
    // 0111扫描
    call scan_0111
    // 判断w是否为0b0111
    xorlw 0b0111
    // 如果不等于0b0111,则return
    btfss STATUS, 2
    return
    // 更新key_data
    movlw 2
    movwf key_data
    return
```

```
624 psect scan_1100_xxxx, class=CODE, delta=2
625 scan_1101_1100:
626     // 1101扫描
627     call scan_1101
628     BRW ;根据扫描结果跳转
629     return//0000
630     return//0001
631     return//0010
632     return//0011
633     return//0100
634     return//0101
635     return//0110
636     return//0111
637     return//1000
638     return//1001
639     return//1010
640     return//1011
641     goto scan_1011_1100_1100//1100
642     goto scan_1011_1100_1101//1101
643     return//1110
644     return//1111
645 scan_1011_1100_1100:
646     // 1011扫描
647     call scan_1011
648     // 判断w是否为0b1011
649     xorlw 0b1011
650     // 如果不等于0b1011,则return
651     btfss STATUS, 2
652     return
653     // 0111扫描
654     call scan_0111
655     // 判断w是否为0b0111
656     xorlw 0b0111
657     // 如果不等于0b0111,则return
658     btfss STATUS, 2
659     return
660     // 更新key_data
661     movlw 1
662     movwf key_data
663     return
664 scan_1011_1100_1101:
665     // 1011扫描
```

```
    call scan_1011
    // 判断w是否为0b1001
    xorlw 0b1001
    // 如果不等于0b1001,则return
    btfss STATUS, 2
    return
    // 0111扫描
    call scan_0111
    // 判断w是否为0b0111
    xorlw 0b0101
    // 如果不等于0b0101,则return
    btfss STATUS, 2
    return
    // 更新key_data
    movlw 8
    movwf key_data
    return
psect scan_1110_xxxx, class=CODE, delta=2
scan_1101_1110:
    // 1101扫描
    call scan_1101
    BRW ;根据扫描结果跳转
    return//0000
    return//0001
    return//0010
    return//0011
    return//0100
    goto scan_1011_1110_0101//0101
    return//0110
    return//0111
    return//1000
    goto scan_1011_1110_1001//1001
    return//1010
    return//1011
    goto scan_1011_1110_1100//1100
    goto scan_1011_1110_1101//1101
    return//1110
    return//1111
scan_1011_1110_0101:
    // 1011扫描
    call scan_1011
    // 判断w是否为0b1011
```

```
    xorlw 0b1011
    // 如果不等于0b1011,则return
    btfss STATUS, 2
    return
    // 0111扫描
    call scan_0111
    // 判断w是否为0b0101
    xorlw 0b0101
    // 如果不等于0b0101,则return
    btfss STATUS, 2
    return
    // 更新key_data
    movlw 5
    movwf key_data
    return
scan_1011_1110_1001:
    // 1011扫描
    call scan_1011
    // 判断w是否为0b1001
    xorlw 0b1001
    // 如果不等于0b1001,则return
    btfss STATUS, 2
    return
    // 0111扫描
    call scan_0111
    // 判断w是否为0b0111
    xorlw 0b0111
    // 如果不等于0b0111,则return
    btfss STATUS, 2
    return
    // 更新key_data
    movlw 3
    movwf key_data
    return
scan_1011_1110_1100:
    // 1011扫描
    call scan_1011
    // 判断w是否为0b1010
    xorlw 0b1010
    // 如果不等于0b1010,则return
    btfss STATUS, 2
    return
```

```
    // 0111扫描
    call scan_0111
    // 判断w是否为0b0110
    xorlw 0b0110
    // 如果不等于0b0110,则return
    btfss STATUS, 2
    return
    // 更新key_data
    movlw 7
    movwf key_data
    return
scan_1011_1110_1101:
    // 1011扫描
    call scan_1011
    // 判断w是否为0b1011
    xorlw 0b1011
    // 如果等于0b1011,则goto
    btfsc STATUS, 2
    goto scan_0
    // 1011扫描
    call scan_1011
    // 判断w是否为0b0011
    xorlw 0b0011
    // 如果不等于0b0011,则return
    btfss STATUS, 2
    return
    // 0111扫描
    call scan_0111
    // 判断w是否为0b0011
    xorlw 0b0011
    // 如果不等于0b0011,则return
    btfss STATUS, 2
    return
    // 更新key_data
    movlw 6
    movwf key_data
    return
scan_0:
    // 0111扫描
    call scan_0111
    // 判断w是否为0b0111
    xorlw 0b0111
```

```
    // 如果不等于0b0111,则return
    btfss STATUS, 2
    return
    // 更新key_data
    movlw 0
    movwf key_data
    return
psect scan_xxxx, class=CODE, delta=2
/**
 * @brief 1110扫描
 */
scan_1110:
    // 将POATB设置为输入
    BANKSEL TRISB
    MOVLW 0b00001110
    MOVWF TRISB
    movwf PORTB
    // 读取PORTB
    MOVF PORTB, 0
    andlw 0x0F
    return
/**
 * @brief 1101扫描
 */
scan_1101:
    // 将POATB设置为输入
    BANKSEL TRISB
    MOVLW 0b00001101
    MOVWF TRISB
    movwf PORTB
    // 读取PORTB
    MOVF PORTB, 0
    andlw 0x0F
    return

/**
 * @brief 1011扫描
 */
scan_1011:
    // 将POATB设置为输入
    BANKSEL TRISB
    MOVLW 0b00001011
```

```
834     MOVWF TRISB
835     movwf PORTB
836     // 读取PORTB
837     MOVF PORTB, 0
838     andlw 0x0F
839     return
840 
841 scan_0111:
842     // 将POATB设置为输入
843     BANKSEL TRISB
844     MOVLW 0b00000111
845     MOVWF TRISB
846     movwf PORTB
847     // 读取PORTB
848     MOVF PORTB, 0
849     andlw 0x0F
850     return
851 
852 /** @brief 主代码段 */
853 psect main, class=CODE, delta=2
854 global _main
855 /** @def RP0
856  *  @brief 寄存器页0
857  */
858 #define RP0 5
859 /** @def RP1
860  *  @brief 寄存器页1
861  */
862 #define RP1 6
863 
864 /**
865  * @brief 主函数
866  *
867  * 该函数初始化微控制器，设置I/O端口，并进入主循环以控制连接到RB0的LED。
868  */
869 _main:
870     BANKSEL PORTA  ;
871     CLRF  PORTA  ;Init PORTA
872     BANKSEL LATA  ;Data Latch
873     CLRF  LATA  ;
874     BANKSEL ANSELA  ;
875     CLRF  ANSELA  ;digital I/O
```

```
876     BANKSEL TRISA  ;
877     MOVLW 00000000B
878     MOVWF TRISA
879
880     BANKSEL PORTC  ;
881     CLRF  PORTC  ;Init PORTC
882     BANKSEL LATC  ;Data Latch
883     CLRF  LATC  ;
884     BANKSEL ANSELC  ;
885     CLRF  ANSELC  ;digital I/O
886     BANKSEL TRISC  ;
887     MOVLW 00000000B
888     MOVWF TRISC
889     /** 初始化PORTB和LATB为0 */
890     BANKSEL PORTB
891     CLRF    PORTB
892     BANKSEL LATB
893     CLRF    LATB
894
895     /** 将ANSELB设置为数字I/O（默认是模拟） */
896     BANKSEL ANSELB
897     CLRF    ANSELB
898
899     /** 打开B组弱上拉 */
900     BANKSEL WPUB
901     MOVLW   0xff
902     MOVWF   WPUB
903
904     MOVLW 0
905     MOVWF index_1
906     MOVWF index
907
908     /** 初始化time 0*/
909     //T0CON0=0b10001000
910     //T0CON1=0b01010110
911     BANKSEL T0CON0
912     MOVLW   0b00000100 // T0CON0配置
913     MOVWF   T0CON0
914     BANKSEL T0CON1
915     MOVLW   0b01010000 // T0CON1配置
916     MOVWF   T0CON1
917     //TMR0H=24=25-1
```

```
    BANKSEL TMR0H
    MOVLW   24
    MOVWF   TMR0H
    //使能TMR0中断
    banksel PIE0
    bsf PIE0, 5
    banksel INTCON
    bsf INTCON, 6
    ; goto draw_0
draw_back:
    //打开定时器
    banksel T0CON0
    bsf T0CON0, 7
    ;进入主循环
    print0x 0x0,0x1,0x2,0x3
    call display_one_frame_loop
    print0x BLANK_DIS, BLANK_DIS, BLANK_DIS, BLANK_DIS
    // 清空key_data
    clrf key_data
    // 打开全局中断
    BANKSEL INTCON
    BSF INTCON, 7
loop:
    //扫描键盘并更新显示数据
    call display_encode
    CALL keyboard_scan
    MOVF key_data,0
    MOVWF last_button
    CLRW
    SUBWF key_data,0
    BTFSS STATUS,2
    goto s1
    goto loop

    s1: ;初态检测到有按键按下,state=0x01
    MOVLW 0x01
    MOVWF state
    MOVLW 0x05
    SUBWF delay,0
    BTFSS STATUS,0;判断延时是否达到5ms
    goto s1        ;没有达到5ms
    CALL keyboard_scan    ;达到5ms
```

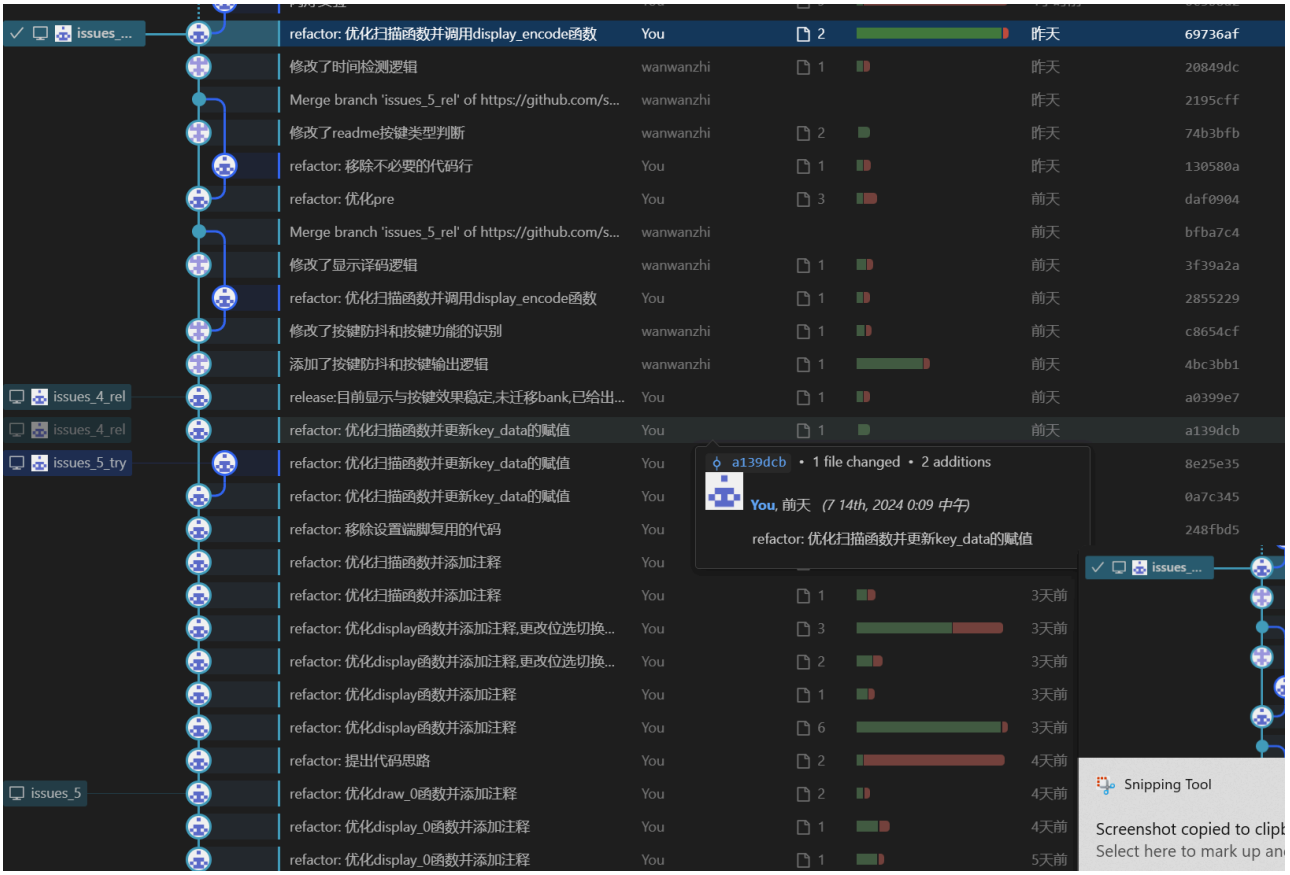
```
960     MOVF key_data,0
961     SUBWF last_button,0  ;检测是否为同一个按键
962     BTFSS STATUS,2
963     goto loop
964     CLRF delay
965     CLRF index        ;用于计算有几个50ms
966     CLRF index_1      ;用于计算现在是多少ms（模五十）
967     goto s2
968
969     s2: ;确认有按键按下，state=0x02
970     MOVLW 0x02
971     MOVWF state
972     MOVLW 00100110B    ;判断是否达到2s
973     SUBWF index,0
974     BTFSS STATUS,0
975     goto if_s3          ;没到2s去判断是否该进入s3
976     goto s8         ;到了则去s8
977
978     if_s3:          ;判断是否为应该进入释放按钮检测
979     CALL keyboard_scan
980     CLRW
981     SUBWF key_data,0    ;判断键盘是否没有按钮按下
982     BTFSS STATUS,2
983     goto s2             ;仍然按下则回到s2
984     goto s3      ;否则去判断是否为一次有效的松开
985
986     s3:          ;用于检测按键是否松开
987     MOVLW 0x03
988     MOVWF state
989     MOVLW 0x05
990     SUBWF delay,0
991     BTFSS STATUS,0;判断延时是否达到5ms
992     goto s3        ;没有达到5ms
993     CALL keyboard_scan    ;达到5ms
994     CLRW
995     SUBWF key_data,0    ;判断键盘是否没有按钮按下
996     BTFSS STATUS,2
997     goto s2
998     CLRF delay        ;用于消抖
999     CLRF index        ;用于计算有几个50ms
1000    CLRF index_1      ;用于计算现在是多少ms（模五十）
1001    goto s4
```

```
1002
1003    s4:          ;按键确实松开
1004    MOVLW 0x04
1005    MOVWF state
1006    MOVLW 00001000B  ;判断是否到达0.4s
1007    SUBWF index,0
1008    BTFSS STATUS,0
1009    goto if_s5       ;没有则去判断是否应该进入s5
1010
1011    ;执行短按显示逻辑
1012    MOVLW 0x01
1013    ADDWF display_data,1
1014    MOVF  last_button,0
1015    MOVWF display_data+3
1016    goto loop          ;回到初态
1017
1018    if_s5:       ;判断是否需要进入s5
1019    CALL keyboard_scan
1020    MOVF key_data,0
1021    SUBWF last_button,0
1022    BTFSS STATUS,2   ;检查与上次按下的是否相同
1023    goto s4
1024    goto s5
1025
1026    s5:                  ;检测是否短时间有第二次摁下
1027    MOVLW 0x05
1028    MOVWF state
1029    MOVLW 0x05
1030    SUBWF delay,0
1031    BTFSS STATUS,0;判断延时是否达到5ms
1032    goto s5       ;没有达到5ms
1033    CALL keyboard_scan      ;达到5ms
1034    MOVF key_data,0
1035    SUBWF last_button,0
1036    BTFSS STATUS,2   ;检查与上次按下的是否相同
1037    goto s4
1038    MOVLW 0x01       ;执行双击按下程序,按键存储在last_button中
1039    MOVWF OPTION_NUM
1040    CLRF delay       ;用于消抖
1041    CLRF index       ;用于计算有几个50ms
1042    CLRF index_1     ;用于计算现在是多少ms（模五十）
1043    ;执行双击逻辑
```

```
1044      MOVLW 0x01
1045      ADDWF display_data+1,1
1046      MOVF  last_button,0
1047      MOVWF display_data+3
1048      CALL display_encode
1049      goto s6
1050
1051      s6:                    ;确认为双击
1052      MOVLW 0x06
1053      MOVWF state
1054      CALL keyboard_scan
1055      CLRW
1056      SUBWF key_data,0    ;判断键盘是否没有按钮按下
1057      BTFSS STATUS,2
1058      goto s6
1059      goto s7
1060
1061      s7:             ;
1062      MOVLW 0x07
1063      MOVWF state
1064      MOVLW 0x05
1065      SUBWF delay,0
1066      BTFSS STATUS,0;判断延时是否达到5ms
1067      goto s7        ;没有达到5ms
1068      CALL keyboard_scan    ;达到5ms
1069      CLRW
1070      SUBWF key_data,0    ;判断键盘是否没有按钮按下
1071      BTFSS STATUS,2
1072      goto s6
1073
1074      goto loop
1075
1076      s8:             ;确认为长摁
1077      ;执行显示逻辑
1078      MOVLW 0x01
1079      ADDWF display_data+2,1
1080      MOVF  last_button,0
1081      MOVWF display_data+3
1082      CALL display_encode
1083
1084      MOVLW 0x08
1085      MOVWF state
```

```
1086        MOVLW 0x02  ;执行程序,按键存储在last_button中
1087        MOVWF OPTION_NUM
1088        CLRF delay        ;用于消抖
1089        CLRF index        ;用于计算有几个50ms
1090        CLRF index_1      ;用于计算现在是多少ms（模五十）
1091        wait:
1092        CALL keyboard_scan
1093        CLRW
1094        SUBWF key_data,0   ;判断键盘是否没有按钮按下
1095        BTFSS STATUS,2
1096        goto wait
1097        goto s7
1098
1099    psect draw_0, class=CODE, delta=2
1100    global draw_0
1101    draw_0:
1102        printdraw 0x39,0b00001001,0b00001001,0b00001111
1103        printdraw 1,1,0,0
1104        printdraw 0,1,1,0
1105        printdraw 0,0,1,1
1106        printdraw 0,0,0,3
1107        printdraw 0,0,0,6
1108        printdraw 0,0,0,12
1109        printdraw 0,0,8,8
1110        printdraw 0,8,8,0
1111        printdraw 8,8,0,0
1112        goto draw_1
1113    psect draw_1, class=CODE, delta=2
1114    global draw_1
1115    draw_1:
1116        printdraw 24,0,0,0
1117        printdraw 48,0,0,0
1118        printdraw 0b00100001,0,0,0
1119        printdraw 0xff,0,0,0
1120        printdraw 0,0,0,0xff
1121        printdraw 0,0,0xff,0
1122        printdraw 0,0xff,0,0
1123        goto draw_back
1124        end
```

实现流程



遇到的问题

btfss和btfsc写反了

查一下发现逻辑不一样

双击显示不是每次增1

问题分析：分析后认为是由于大量数字快速显示导致的，而后根据状态图查找原因，发现我将双击检测的输出放到了防抖部分，由于在防抖部分不停循环，导致输出数据不断递增。

解决方案：将双击检测的输出放到确认双击后，跳转到防抖状态前。