```
/*  ////////////////////////////////////////////////// /////
 *   PIC16F1827にSC1602互換LCD表示パネルと10キーパッドを接続し
 *   i2cのスレーブとして表示を行う実験用ルーチンです。
 *   sc1602とは4ビットモード接続されます。
 *
 *      i2cの状態管理はマイクロチップ社AN-734を参考にしました。状態1-5
 *    でmsspからの割り込みをハンドリングします。
 *
 *  PIC16F88   V0.1 2011.09.30  V1.0 2011.10.03 をベースに改造
 *      PIC16F1823 V2.1 2011.11.10  2 NiH operation(CLK OUT)
 *      PIC16F1827 V3.0 2012.8.30
 *      PIC16F1827 V4.0 2012.9.25  LCD+KeyPad  Status  1-5
 *              timing tunig for LCD & Key pad connection
 *
 *              MPLAB v8.85 & HI-TECH C V9.83
 *              by nobcha (c)2012
 *
 *      i2c経由LCDデータは必ず２バイト単位でくるとする。MAX6(2,4,6)
 *      １バイト目はRSビット(コマンド0x0、データ0x40)、２バイト目がLCDに
 *      書き込むデータです
 *      ストロベリーリナックスの拡張コマンドはサポートしてません
 *
 *      コマンドで0x11がキースキャンスタート、0x10がキースキャンストップ
 *      キースキャン中はLCDデータを渡してはいけない
 *      i2cのREADコマンドに応答して現在管理しているキーパッドの
 *      情報を返事する。ビット７は長押し（同じキーが600msより長く）
 *      を示し、下位の7ビットはキーパッドのキーコードを示す。
 *      数字は0x30から0x39、米印はアスタリスク0x3A、シャープ＃は
 *      0x3Bを送り返す。
 *      キー入力は50mS間隔のスキャン2回同じコードが続けば有効とする
 *
 *      RA0-3:SC1602は4bit,10キー―列駆動RA0:D,RA1:E,RA2:F,RA3:G
 *      RA5:SW3,KP-C
 *      RA7:動作確認用LED
 *      RB1:SDA1 MSSP
 *      RB2:SW1,KP-A
 *      RB3:SW2,KP-B
 *      RB4:SCL1 MSSP
 *      RB6:LCD EN bit (enable)はRB6につなぎます。
 *      RB7:LCD RS bit (RS)はRB7につなぎます。
 *
 *      SC1602 pin connection via 4bit mode
 *              #1      Vdd=5V
 *              #2      Vss=GND
 *              #3      LCD contrust center of 5k VOL
 *              #4      RS      RB7
 *              #5      R/W     GND
 *              #6      EN      RB6
 *
 *              #11-14  DATA    RA0-3
 *
 *        TMR1 is gate time controller as set (65536-40000)  for5ms
 *
 *        CLK INT 16MHz
 *
 * ////////////////////////////////////////////////////// */
#include        <htc.h>
#include        "lcd.h"

#define _XTAL_FREQ  16000000
#define PIC_CLCK 1600000
#define I2C_ADR 0x7C
#define DEBUG 0
#define MON_LED LATA7

__CONFIG(
    FOSC_INTOSC & WDTE_OFF & PWRTE_ON & MCLRE_OFF & CP_OFF
    & CPD_OFF & BOREN_OFF & CLKOUTEN_ON & IESO_OFF & FCMEN_OFF
        );

__CONFIG(
        WRT_OFF & PLLEN_OFF & STVREN_ON &  LVP_OFF
        );
/* ////////////////////////////////////////////  */
```

```
unsigned char   buffer[20];                     /* 受信バッファ */
unsigned char   stat[20];                       /* デバッグ用ステイタスバッファ*/
char    rcv_count=0 ,cnt=0, kint_on, ssp_int_on, lcd_rs_rcv;

// Function prototyping
void    lcd_dispch( unsigned char);             /* 1バイトを2ヘキサでLCDへ */
unsigned char hextoascii(unsigned char);        /* ニブルをヘキサ変換 */
void    init(void);
void    ssp_init(void);
void    cnt_setup(void);
void    cnt_stop(void);
unsigned char key_col_scan(unsigned char);
void ssp_handler(void);
interrupt i2c_slave(void);

// variable
unsigned char lcdrs, key_in_p, key_in=0, cnt_up=100;
char key_in_cnt=0;
unsigned char   MSG1[] = "i2c lcd_key v4";
unsigned char   MSG2[] = "K=";


void main(void){
char    i, j;
        init();                                 /* ポート初期化                 */
        MON_LED=1;                              // MONITOR LATA7 LED ON
        lcd_init();                             /* LCDの初期化                  */

        ssp_int_on=0;
        MON_LED=0;

        if (DEBUG==1){                          /*      debug mode              */
                lcd_goto(0x40);                 /* 2行目にLCD動作表示           */
                lcd_puts(MSG1);                 /* LCDに初期表示                */
        }

        ssp_init();                             /* SSPの初期化                  */
        SSP1IF = 0;                             /* SSPIF clear                  */
        MON_LED = 1;                            /* 動作モニター用LED点灯        */

        rcv_count=0;                            // i2c receive counting

        while(1){                               /* 繰り返し                     */
                SSP1IE=1;                                       // MSSP  INT ENABLE
                PEIE=1;
                GIE=1;

                if(kint_on&DEBUG){              // Key in occuered?
                        GIE=0;
                        lcd_goto(0x51);         // KEY IN MSG
                        lcd_puts(MSG2);         // K=

                        lcd_dispch(key_in_p);   // 2 HEXA for key code
                        lcd_dispch(key_in_cnt); // Long keying counter
                        kint_on=0;              // Key in int flag reset
                        GIE=1;
                }

                if(SSP1STATbits.P & rcv_count){ /* 受信のストップ状態チェック   */
                        GIE=0;
                        if(DEBUG){                      //      debug mode & received
                                for(i=0;i<(rcv_count+1);i++){
                                        lcd_goto(i*3 );
                                        lcd_putch(0x23);        // #
                                        lcd_dispch(buffer[i]);
//                                      lcd_dispch(stat[i]);
                                }
                                lcd_goto(0x53);         /* 2行目にLCD動作表示   */
                                lcd_putch(cnt++ | 0x30);  // display count
                                if(cnt>0x05){cnt=0;}

                                lcd_putch(0x3C);
                                lcd_dispch(rcv_count);  /* 下位4ビット取り出しASCII */
                                lcd_putch(0x3E);
```

```
                                        main_slave
                                lcd_putch(0x20);

                                for(j=0;j<2;j++){       /* 表示時間を待たせる           */
                                        __delay_ms(100);
                                }
                        }
                        rcv_count=0;
                        MON_LED ^= 1;
                        GIE=1;
                }
        }
}

// Displaying 1 byte via 2 Asciis
void    lcd_dispch( unsigned char ch_code){
                        __delay_us(20);
                lcd_putch(hextoascii((ch_code>>4)&0xF));
                        __delay_us(20);
                lcd_putch(hextoascii(ch_code&0xF));
        }

// Converting 4 bits to hexadicimal asciis
unsigned char hextoascii(unsigned char hex_data){
                hex_data = hex_data | '0';              // ASCII code
                if(hex_data>0x39)hex_data=hex_data+7;   // alphabet
        return hex_data;
        }

// Initializing ports and osc
void    init(void){
        PORTA = 0b00001001;                             //
        PORTB = 0b00000000;                             //
        TRISA = 0b01110000;                             /* RA0-3は出力、RA4-6入力、RA7出力  */
        TRISB = 0b00011111;                             /* RB0ー4は、ｓｗ、i2c用で入力、RB5-7出力設定  */
        OSCCON = 0b011110000;                           /* 内部クロック16MHz            */

        ANSELA = 0x00;                                  /* 全ポートデジタル             */
        ANSELB = 0;
        OPTION_REG = 0x00;                              /* オプション設定なし           */
                                                        // port direction: 1:input
        }

// Initializing MSSP function
void    ssp_init(void){
        SSP1CON1 = 0x36;                                // b5:SSP1EN,b4:CKP,b3-0:SSP_slave_7bit
        SSP1ADD = I2C_ADR;                              // SSP ADDRESS SET
        PEIE = 1;                                       // PEIE enable
        SSP1IE = 1;
        }

// Setting TMR1 up
void cnt_setup(void){
        TMR0 = 0 ;                                      // TMR0 clear
        TMR1L = 0;                                      // Clear Low Byte of TMR1
                                                        // 16MHz 62.5nS*4 250nS  40000 count
        TMR1H = 99;                                     // Set 99*256 + 192
        TMR1L = 192;                                    // Set (99*256+192) =25536=65536-40000
        TMR1IF=0;                                       // TMR1 flag off
        TMR1IE=1;                                       // TMR1 INT ENABLE
        PEIE=1;                                         // PEIE 1
        TMR1ON=1;                                       // TMR1ON 1
        }

// Stop TMR1
void cnt_stop(void){
        TMR1IE=0;                                       // TMR1 INT ENABLE
        PEIE=0;                                         // PEIE 1
        TMR1ON=0;                                       // TMR1ON 0
        cnt=0;
        cnt_up=10;                                      // Key in polling timer counter
}

// Getting Key code by scanning
unsigned char key_col_scan(unsigned char pos){  // pos:0-3
        unsigned char key_pos=0, key_col=1,port_back;   // If no key, key_pos:0
```

```
                                                // If any key, 0x30-0x3B
        key_col=(key_col<<pos);                 // column drive bit
        port_back=PORTA;
        LATA=(PORTA&0xF0)|((key_col^0xF)&0xF);  // SET PORTA
        asm("nop");                             // Wait
        if(RA5==0) key_pos=0x38|pos;    // C line check
        else if(RB3==0) key_pos=0x34|pos;       // B line
        else if(RB2==0) key_pos=0x30|pos;       // A line
        LATA=port_back;
        return key_pos;                         // Return key code 0x30-0x3F
        }

// ssp int handling function
void ssp_handler(void) {
        unsigned char s_state,dummy;
        s_state=SSP1STAT&0b00101101;            // SMP,CKE,-D/A,P,-S,-R/W,UA,-BF

        switch(s_state){
                case 0b00001001:                // State1 address,S,write,buffer full
                        dummy=SSP1BUF;          // if address, read SSPBUF as dummy
                        rcv_count=0;
                        break;
                case 0b00101001:                // State2 data,write,buffer full
                        buffer[rcv_count]=SSP1BUF ;     //
                        if((rcv_count&0x1)==0){         // receiving RS byte and DATA byte as pair
                                lcd_rs_rcv=((buffer[rcv_count]>>6)&0x1);        // Set lcd_rs bit
                        }
                        else if((buffer[rcv_count]==0x10)&(lcd_rs_rcv==0))      cnt_stop();
                        else if((buffer[rcv_count]==0x11)&(lcd_rs_rcv==0))      cnt_setup();
                        else if(DEBUG==0) lcd_write_rs(buffer[rcv_count],lcd_rs_rcv);   // Data byte
                        stat[rcv_count]=SSP1STAT;       // status buffering
                        rcv_count++;                    // next buufer
                        break;
                case 0b00001100:                // State3 address,S, read,buffer empty
                case 0b00001101:                // State3 address,S, read,buffer full
                        SSP1CON1bits.CKP=0;             // CKP clear
                        dummy=SSP1BUF;                  // dummy address read
                        SSP1BUF=key_in_p;               // key in data transmitting
                        SSP1CON1bits.CKP=1;             // CKP releas
                        break;
                case 0b00101100:                // State4 data,S, read,buffer empty
                        SSP1CON1bits.CKP=0;             // CKP clear

                        SSP1CON1bits.CKP=1;             // CKP releas
                        break;
                case 0b00101000:                // State5 data,S, read,buffer empty &full
                case 0b00101101:
                        SSP1CON1bits.CKP=1;             // CKP release

                        break;
                default:
                        lcd_puts("ERR");
                        break;
        }
}

// Interrupu service
interrupt i2c_slave(void){                      // i2c slave receive int func
        unsigned char pos=0;
        GIE=0;                                  // R/W detect? needed  if read send KB_status


        if(TMR1IF==1){
                cnt_up--;
                if(cnt_up==0){
                        cnt_up=10;
                        MON_LED ^= 1;
                        kint_on=1;                              // INT FLAG
                        for(pos=0;pos<=3;pos++){
                                key_in=key_col_scan(pos);       // key scan
                                if(key_in) break;
                        }
                        if(key_in==0){                          // no key
                                key_in_p=0;
                                key_in_cnt=0;                   // key in count reset
```

```
                }
                else {                                  // Any key
                        if(key_in!=(key_in_p&0x7F)) {
                                key_in_p=key_in;        // pre key buffer refresh
                                key_in_cnt=0;           // key in count reset
                        }
                        else if(key_in_cnt<125) key_in_cnt++;   // if same key, cnt up
                }
                if(key_in_cnt>100) key_in_p=(key_in_p | 0x80);  // set long keying flag
        }
        cnt_setup();
    }
    if(SSP1IF==1){
        SSP1IE=0;
        SSP1IF=0;
        ssp_handler();

        SSP1IE=1;
    }

    GIE=1;
}
```