簡易ターミナル(LCD&Keypad)(PIC18F2620)

概要

PICを使った装置と、それを操作する人間とのインターフェイスには、いろいろな物が利用されています。 ◎入力(操作者→装置)

- トグルスイッチ、プッシュスイッチ、サムロータリスイッチ、ロータリエンコーダなど

◎出力(操作者←装置)

- LED□7セグ□LCD□アナログメータなど

また、パソコン経由(RS232C+ターミナルソフト)で操作する場合もあります。 ◎入力(操作者→装置)

- キーボード

◎出力(操作者←装置)

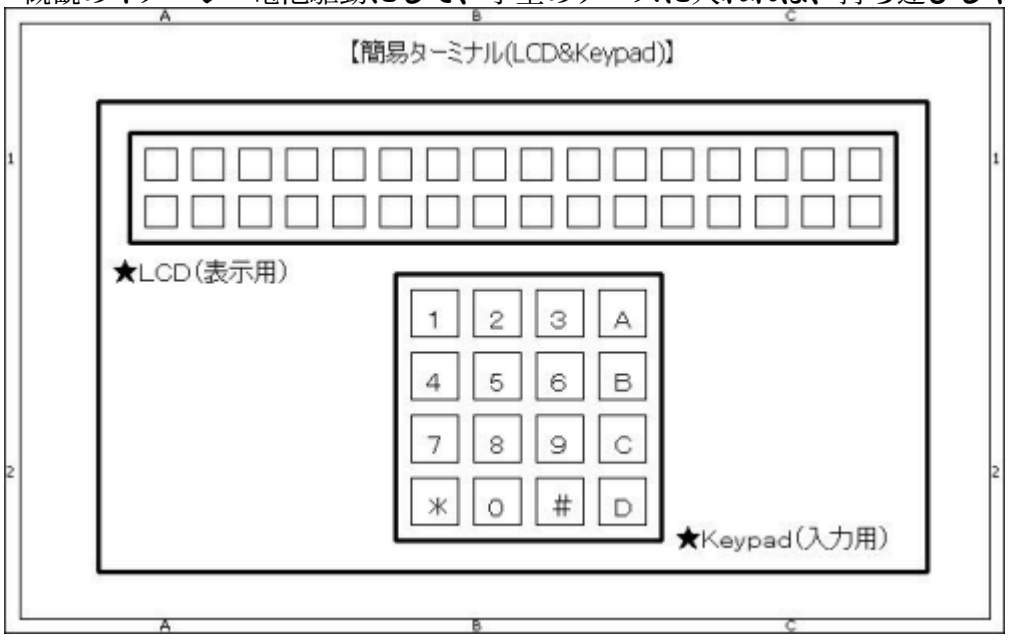
- モニター画面

各々、一長一短はありますが、今回は、汎用的で、小型で、PICとの接続が容易で、直感的に操作できるように LCDとKeypadを組み合わせた「簡易ターミナル」を製作しました。

<仕様>

- 簡易ターミナルと装置側とはUSART(Tx□Rx)で直結可能とします。(レベル変換用の部品は不要です)
- 通信速度は、9600bpsとします。
- 入力(操作者→装置)には、4×4のKeypad(16キー)を使用します。
- 出力(操作者←装置)には、16文字×2行のLCDを使用します。
- 装置側のプログラムから簡易ターミナルにアクセスしやすいように、アクセスライブラリを提供します。

<概観のイメージ> 電池駆動にして、小型のケースに入れれば、持ち運びしやすくなります。



動作原理

プログラムを、次の4つの箇所に分けて説明します。

- 簡易ターミナルの通常モードの処理について
- 簡易ターミナルのテストモードの処理について
- 装置側から簡易ターミナルにアクセスするライブラリについて
- 簡易ターミナルを評価するための装置側のサンプルプログラムについて(PIC12F683を使用)

<処理の流れ(通常モード)>※SWをOFFの状態で起動します。 ■ 入力(操作者→装置)(Keypad→装置)

1. Keypadのキーがクリックされると、キーコードを装置側に返します。
2. 1.に戻ります。

■ 出力(操作者←装置)(LCD←装置)

1. 装置側からのデータを受信します。(割り込み処理)
2. データのSTX(0x02)~ETX(0x03)までを、一つのコマンドとして受け付けます。
   コマンドフォーマット→[(STX)(コマンド)、(引数1→2...)(ETX)]
3. "A"コマンド→行と列を指定してテキストを表示します。
4. "B"コマンド→カーソル位置からテキストを表示します。
5. "C"コマンド→LCDのコマンド(_LCD_CLEAR→_LCD_CURSOR_OFF→_LCD_TURN_ON等)処理を行います。
6. 1.に戻ります。

<処理の流れ(テストモード)>※SWをONの状態で起動します。 テストモードでは、簡易ターミナルのTxとRxは直結されるため、簡易ターミナルからの送信(Tx)データは、 そのまま簡易ターミナルの受信(Rx)データとなります。 ■ 入力(操作者→装置)(Keypad→簡易ターミナル(見かけ上は装置側となる))

1. "＊"キーがクリックされると、装置側にLCDクリアコマンドを送信します。
2. "＃"キーがクリックされると、装置側にLCD表示データ(カーソル位置)を送信します。
3. "A"キーがクリックされると、装置側にLCD表示オンコマンドを送信します。
4. "B"キーがクリックされると、装置側にLCD表示オフコマンドを送信します。
5. "C"キーがクリックされると、装置側にLCDカーソル移動(1行目)コマンドを送信します。
6. "D"キーがクリックされると、装置側にLCDカーソル移動(2行目)コマンドを送信します。
7. 上記以外のキーがクリックされると、装置側にLCD表示データ(カーソル位置)を送信します。

■ 出力(操作者←装置)(LCD←簡易ターミナル(見かけ上は装置側となる)) 通常モードの処理と同じです。

<アクセスライブラリ> ■ 簡易ターミナル→装置側

- クリックされたKeypadのキーコードを返します。
  charterminal_get_char();

■ 簡易ターミナル←装置側

- LCDの行と列を指定して、文字を表示します。
  voidterminal_put_char(short row, short col, char dat);
- LCDの行と列を指定して、文字列を表示します。
  voidterminal_put_str(short row, short col, char *dat);
- LCDのカーソル位置に、文字を表示します。
  voidterminal_put_char_cp(char dat);

- LCDのカーソル位置に、文字列を表示します。
  voidterminal_put_str_cp(char *dat);
- LCDをコマンド制御します。
  voidterminal_cmd(char cmd);

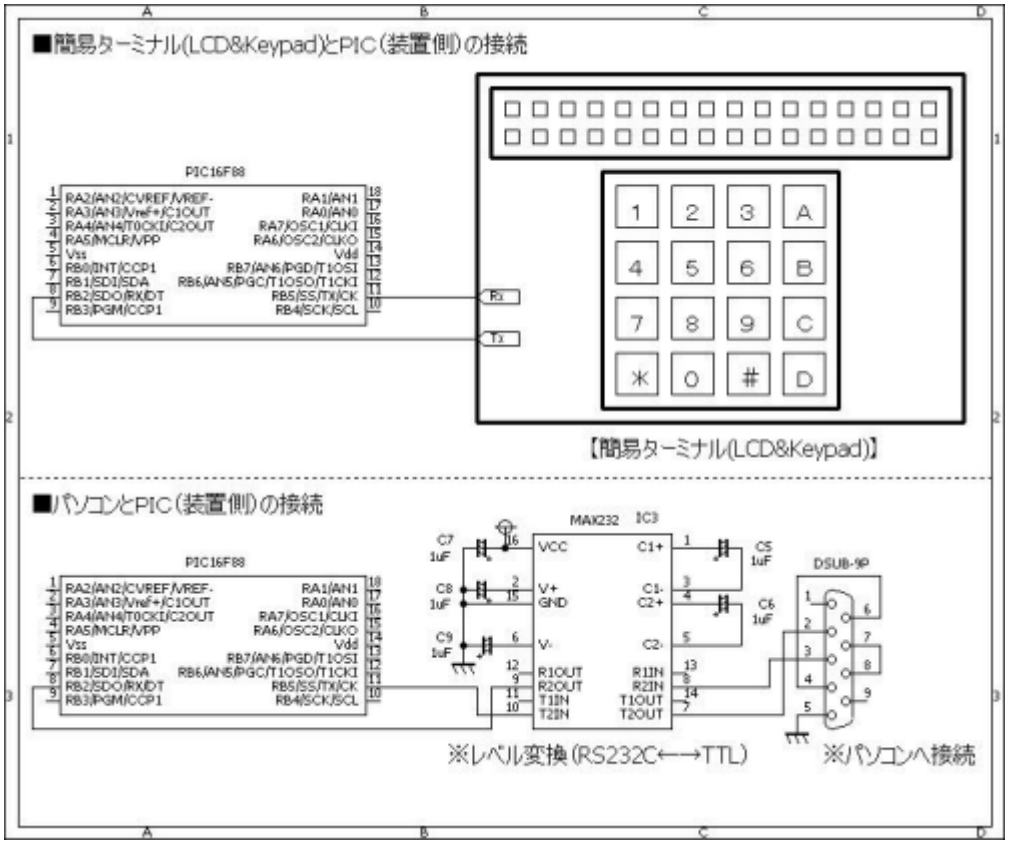
PIC(装置側)のプログラムで用意する関数 PICによっては、USARTモジュールが内蔵されているもの(PIC16F88など)もあれば、内蔵されていないもの(PIC12F683など)もあります。そのため、USARTの送受信関数には、いろいろなバリエーションが考えられます。

- USARTモジュールを使用する標準関数(UART1_Read関数、UART1_Write関数など)
- USARTモジュールを使用しない標準関数(Soft_UART_Read関数、Soft_UART_Write関数など)
- 独自に作成した関数
  つまり、ライブラリ内では特定できない、次の二つの関数を実装して頂く必要があります。
- USARTから1文字受信するための関数
  charterminal_stub_get_char();
- USARTへ1文字送信するための関数
  voidterminal_stub_put_char(char c);

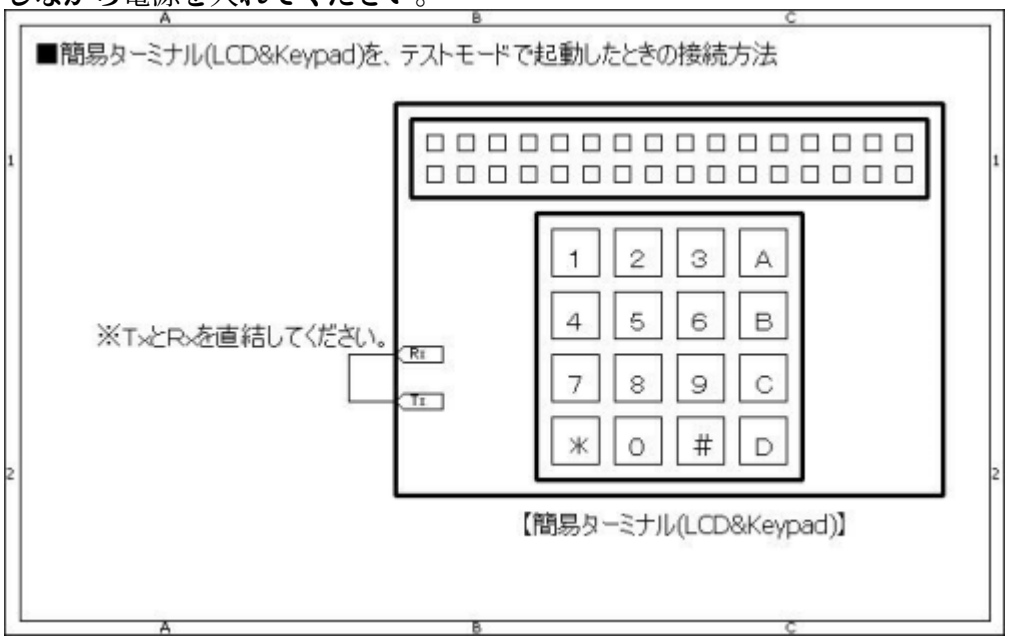
<処理の流れ(評価用のプログラム:PIC12F683)>

1. アクセスライブラリ用のスタブ関数を実装します。
   charterminal_stub_get_char();
2. アクセスライブラリ用のスタブ関数を実装します。
   voidterminal_stub_put_char(char c);
3. USARTを初期化します。
4. 簡易ターミナルより、キーコードを受信します。
   charterminal_get_char();
5. キーコードが“A”であれば、A/D変換(CH1)した結果を、簡易ターミナルに送信します。
   voidterminal_put_str(short row, short col, char *dat);
6. キーコードが“B“であれば、A/D変換(CH2)した結果を、簡易ターミナルに送信します。
   voidterminal_put_str(short row, short col, char *dat);
7. キーコードが“C”であれば、LED1を反転させます。
8. キーコードが“D“であれば、LED2を反転させます。
9. キーコードが“*”であれば、LCDクリアコマンドを、簡易ターミナルに送信します。
   voidterminal_cmd(char cmd);
10. 上記以外のキーコードであれば、無視します。
11. 4.に戻ります。

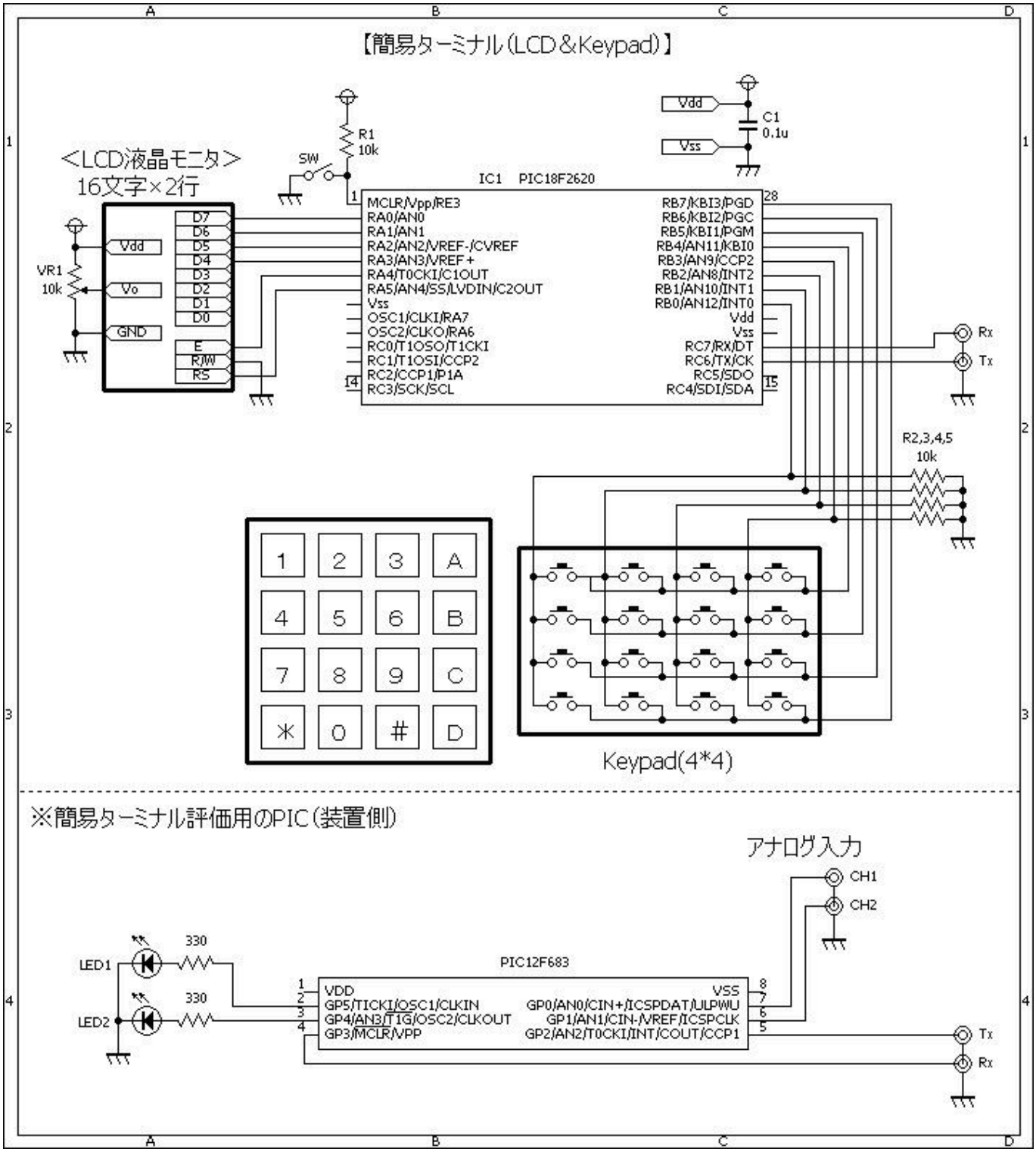
<簡易ターミナルとPIC(装置側)との接続> 簡易ターミナルとPIC(装置側)は直結可能なので、パソコンとPIC(装置側)の接続時に必要となる、レベル変換用の部品は不要になります。



<テストモード時の接続> 簡易ターミナルのTxとRxを直結してください。起動時はスイッチ(SW)を押下しながら電源を入れてください。



回路図



ソースコード

◎簡易ターミナルの本体側

terminal_v1.c

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

```
      ＜簡易ターミナル□LCD&Keypad□□
*/
//*****************************************************************************
*
//■■■インクルード■■■
#include    "lib_terminal.h"
//*****************************************************************************
*
//■■■関数＆データ宣言■■■
extern    void    main();
extern    void    init_lcd();
extern    void    init_keypad();
extern    char    get_keypad();
extern    void    init_usart();
extern    char    recv_data[], recv_flg;
extern    void    command_proc(char * cmd);
extern    void    test_mode();
//*****************************************************************************
*
//■■■マクロ定義■■■
//keypad
char    keypadPort          at   PORTB;
//LCD
sbit    LCD_RS              at   RA5_bit;
sbit    LCD_EN              at   RA4_bit;
sbit    LCD_D7              at   RA0_bit;
sbit    LCD_D6              at   RA1_bit;
sbit    LCD_D5              at   RA2_bit;
sbit    LCD_D4              at   RA3_bit;
sbit    LCD_RS_Direction    at   TRISA5_bit;
sbit    LCD_EN_Direction    at   TRISA4_bit;
sbit    LCD_D7_Direction    at   TRISA0_bit;
sbit    LCD_D6_Direction    at   TRISA1_bit;
sbit    LCD_D5_Direction    at   TRISA2_bit;
sbit    LCD_D4_Direction    at   TRISA3_bit;
//USART
sbit    TX_Direction        at   TRISC.B6;
sbit    RX_Direction        at   TRISC.B7;
//SW
sbit    SW                  at   RE3_bit;
//other
#define     INPUT_MODE          1
#define     OUTPUT_MODE         0
//*****************************************************************************
*
//■■■メイン関数■■■
void    main()
{
        char    kd;
        //クロックを8MHzに設定します。
        OSCCON.IRCF2 = 1;
        OSCCON.IRCF1 = 1;
```

```
        OSCCON.IRCF0 = 1;
        //□□□変換は使用しません。
        ADCON1.PCFG3 = 1;
        ADCON1.PCFG2 = 1;
        ADCON1.PCFG1 = 1;
        ADCON1.PCFG0 = 1;
        //
        init_lcd();
        init_keypad();
        init_usart();
        Lcd_Out(1, 1, "terminal v1.00");
        Delay_ms(1000);
        Lcd_Cmd(_LCD_CLEAR);
        //割り込みを許可します。
        INTCON.PEIE = 1;
        INTCON.GIE = 1;
        //
        if (SW == 0) {
                test_mode();
        }
        while (1) {
                //データの受信と表示
                if (recv_flg == 1) {
                        command_proc(recv_data);
                        recv_flg = 0;
                }
                //キーの取得と送信
                if ((kd = get_keypad()) != 0) {
                        UART1_Write(kd);
                }
        }
}
//*********************************************************************
*
//■■■□□□初期化関数■■■
void    init_lcd()
{
        Lcd_Init();
        Lcd_Cmd(_LCD_CLEAR);
        Lcd_Cmd(_LCD_CURSOR_OFF);
}
//*********************************************************************
*
//■■■キーパッド初期化関数■■■
void    init_keypad()
{
        Keypad_Init();
}
//*********************************************************************
*
//■■■キー取得関数■■■
```

```
char    kp_tbl[17] =
{0,'1','2','3','A','4','5','6','B','7','8','9','C','*','0','#','D',};
char    get_keypad()
{
        char        kd;
        //
        kd = Keypad_Key_Click();
        if (kd != 0) {
                Delay_ms(100);
        }
        return (kp_tbl[kd]);
}
//*********************************************************************
*
//■■■□□□□□初期化関数■■■
void    init_usart()
{
        TX_Direction = OUTPUT_MODE;         //USART->TX
        RX_Direction = INPUT_MODE;          //USART->RX
        UART1_Init(9600);
        PIE1.RCIE = 1;
        PIR1.RCIF = 0;
}
//*********************************************************************
*
//■■■割り込み関数■■■
char    recv_data[32], recv_flg = 0;
char    recv_temp[32], recv_pnt = 0;
void    interrupt()
{
        char        rd;
        //
        if (PIR1.RCIF == 1) {
                PIR1.RCIF = 0;
                //
                rd = UART1_Read();
                if (rd == STX) {
                        recv_pnt = 0;
                        return;
                }
                if (rd == ETX) {
                        strncpy(recv_data, recv_temp, recv_pnt);
                        recv_data[recv_pnt] = 0x00;
                        recv_flg = 1;
                        recv_pnt = 0;
                        return;
                }
                recv_temp[recv_pnt] = rd;
                recv_pnt++;
        }
}
```

```
//*********************************************************************
*
//■■■コマンド処理関数■■■
void    command_proc(char * cmd)
{
        short    row = 0, col = 0;
        //
        switch (cmd[0]) {
        case 'A':  //行と列を指定してテキストを表示する。
                row = atoi(&cmd[2]);
                col = atoi(&cmd[5]);
                Lcd_Out(row, col, &cmd[8]);
                return;
        case 'B':  //カーソル位置からテキストを表示する。
                Lcd_Out_Cp(&cmd[2]);
                return;
        case 'C':  //コマンド処理を行います。
                Lcd_Cmd(cmd[2]);
                return;
        }
}
//*********************************************************************
*
//■■■テスト関数■■■
void    test_mode()
{
        char    kd;
        //
        while (1) {
                //データの受信と表示
                if (recv_flg == 1) {
                        command_proc(recv_data);
                        recv_flg = 0;
                }
                //キーの取得と送信
                if ((kd = get_keypad()) != 0) {
                        switch (kd) {
                        case '*':
                                terminal_cmd(_LCD_CLEAR);
                                break;
                        case '#':
                                terminal_put_str_cp("terminal test!");
                                break;
                        case 'A':
                                terminal_cmd(_LCD_TURN_ON);
                                break;
                        case 'B':
                                terminal_cmd(_LCD_TURN_OFF);
                                break;
                        case 'C':
                                terminal_cmd(_LCD_FIRST_ROW);
```

```
                                break;
                        case 'D':
                                terminal_cmd(_LCD_SECOND_ROW);
                                break;
                        default:
                                terminal_put_char_cp(kd);
                                break;
                        }
                }
        }
}
//*********************************************************************
*
char    terminal_stub_get_char()
{
        if (UART1_Data_Ready() == 1) {
                return (UART1_Read());
        }
        return (0);
}
void    terminal_stub_put_char(char c)
{
        UART1_Write(c);
}
//*********************************************************************
*
```

◎アクセスライブラリ

lib_terminal.c

```
//*********************************************************************
*
//■■■インクルード■■■
#include    "lib_terminal.h"
//*********************************************************************
*
char    tmp[4];
//*********************************************************************
*
void    terminal_stub_put_str(char *s)
{
        while (*s != 0x00) {
                terminal_stub_put_char(*s);
                s++;
         }
}
//*********************************************************************
*
//■■■ターミナル文字受信関数■■■
```

```
char    terminal_get_char()
{
        return (terminal_stub_get_char());
}
//*********************************************************************
*
//■■■ターミナル文字送信（行列指定）関数■■■
void    terminal_put_char(short row, short col, char c)
{
        terminal_stub_put_char(STX);
        terminal_stub_put_str("A,");
        ByteToStr(row, tmp);
        terminal_stub_put_str(&tmp[1]);
        terminal_stub_put_char(',');
        ByteToStr(col, tmp);
        terminal_stub_put_str(&tmp[1]);
        terminal_stub_put_char(',');
        terminal_stub_put_char(c);
        terminal_stub_put_char(ETX);
}
//*********************************************************************
*
//■■■ターミナル文字列送信（行列指定）関数■■■
void    terminal_put_str(short row, short col, char *s)
{
        terminal_stub_put_char(STX);
        terminal_stub_put_str("A,");
        ByteToStr(row, tmp);
        terminal_stub_put_str(&tmp[1]);
        terminal_stub_put_char(',');
        ByteToStr(col, tmp);
        terminal_stub_put_str(&tmp[1]);
        terminal_stub_put_char(',');
        terminal_stub_put_str(s);
        terminal_stub_put_char(ETX);
}
//*********************************************************************
*
//■■■ターミナル文字送信関数■■■
void    terminal_put_char_cp(char c)
{
        terminal_stub_put_char(STX);
        terminal_stub_put_str("B,");
        terminal_stub_put_char(c);
        terminal_stub_put_char(ETX);
}
//*********************************************************************
*
//■■■ターミナル文字列送信関数■■■
void    terminal_put_str_cp(char *s)
{
        terminal_stub_put_char(STX);
```

```
        terminal_stub_put_str("B,");
        terminal_stub_put_str(s);
        terminal_stub_put_char(ETX);
}
//*****************************************************************************
*
//■■■ターミナルコマンド送信関数■■■
void    terminal_cmd(char cmd)
{
        terminal_stub_put_char(STX);
        terminal_stub_put_str("C,");
        terminal_stub_put_char(cmd);
        terminal_stub_put_char(ETX);
}
//*****************************************************************************
*
```

◎簡易**ターミナル**評価用のPIC(装置側)※PIC12F683使用

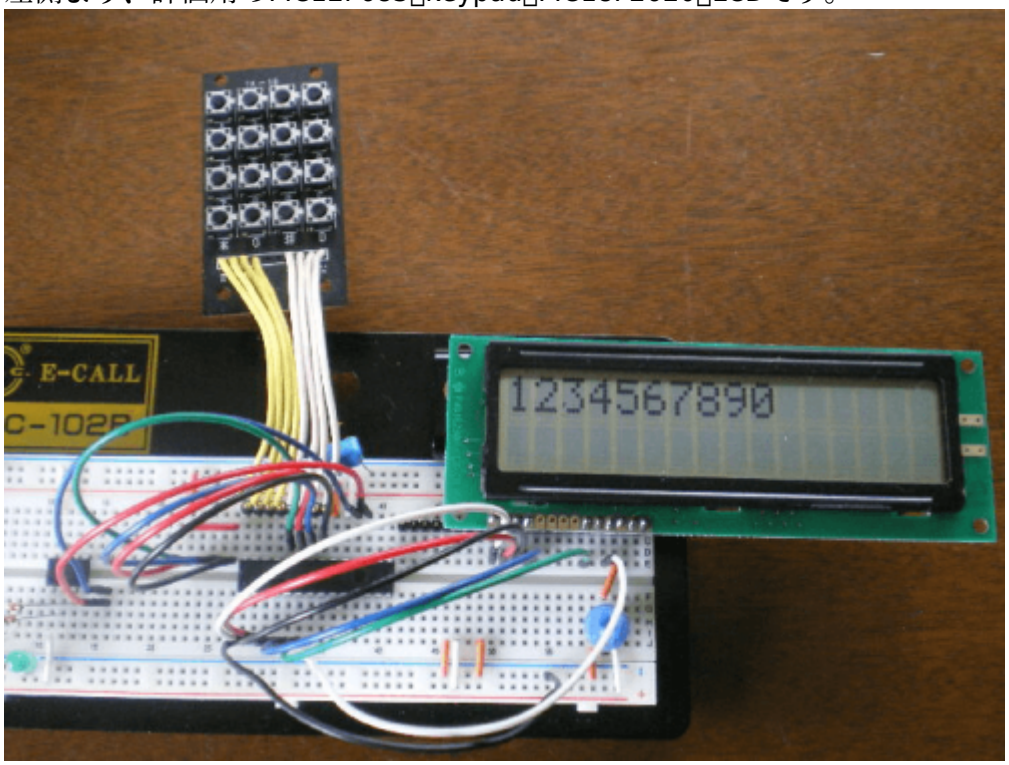
terminal_test.c

```
//*****************************************************************************
*
//■■■インクルード■■■
#include    "lib_terminal.h"
//*****************************************************************************
*
//■■■マクロ定義■■■
#define     LED1    GPIO.F5
#define     LED2    GPIO.F4
//*****************************************************************************
*
//■■■メイン関数■■■
void    main()
{
    char    rd, buf[6];
    double  ad;
    //
    OSCCON = 0b01110000;  //クロックは8Mhz
    CMCON0 = 0b00000111;  //コンパレータは使用しない。
    ANSEL  = 0b00000011;  // AN0□AN1を使用する。
    TRISIO = 0b00001011;
    //
    LED1 = 0;
    LED2 = 0;
    //
    Soft_UART_Init(&GPIO, 3, 2, 9600, 0);
    //
    while (1) {
        rd = terminal_get_char();
```

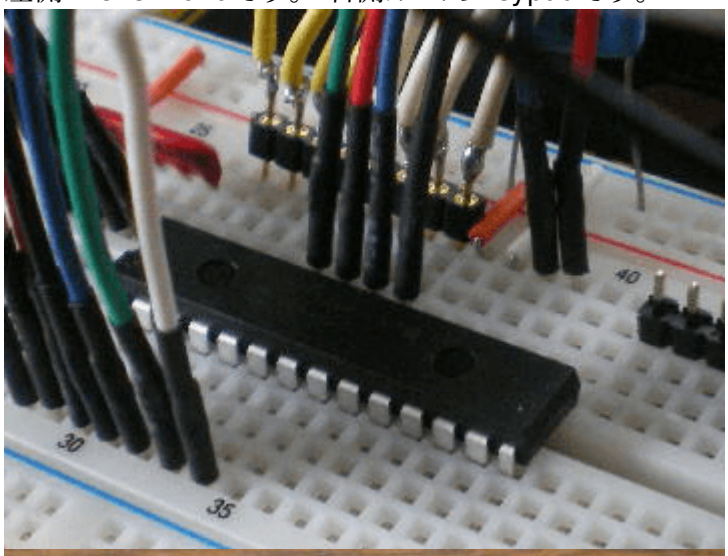
```
        switch (rd) {
        case 'A':
            ad = Adc_Read(0);
            ad = ad * 4.8828125;
            WordToStr(ad, buf);
            terminal_put_str(1, 1, buf);
            terminal_put_str(1, 6, "mV");
            break;
        case 'B':
            ad = Adc_Read(1);
            ad = ad * 4.8828125;
            WordToStr(ad, buf);
            terminal_put_str(2, 1, buf);
            terminal_put_str(2, 6, "mV");
            break;
        case 'C':
            LED1 = ~LED1;
            break;
        case 'D':
            LED2 = ~LED2;
            break;
        case '*':
            terminal_cmd(_LCD_CLEAR);
            break;
        }
    }
}
//*********************************************************************
*
char    terminal_stub_get_char()
{
    char    rd, error;
    //
    while (1) {
        rd = Soft_UART_Read(&error);
        if (error == 0)
            return(rd);
    }
}
void    terminal_stub_put_char(char c)
{
    Soft_UART_Write(c);
}
//*********************************************************************
*
```

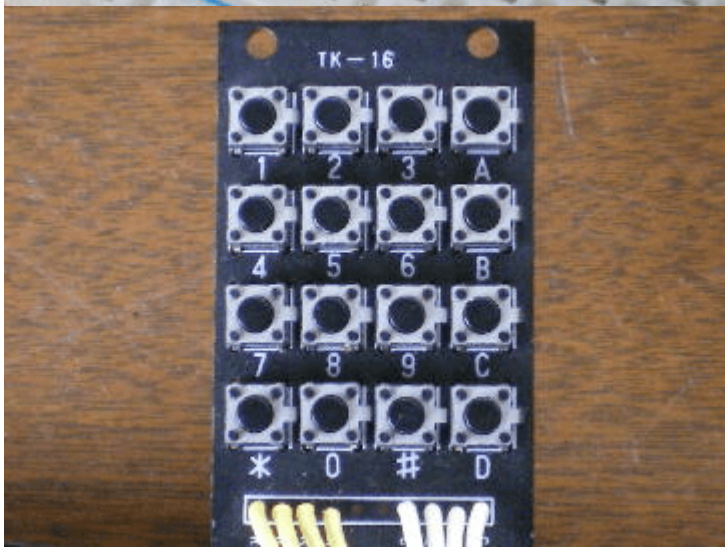
動作確認

左側より、評価用のPIC12F683、keypad、PIC18F2620、LCDです。

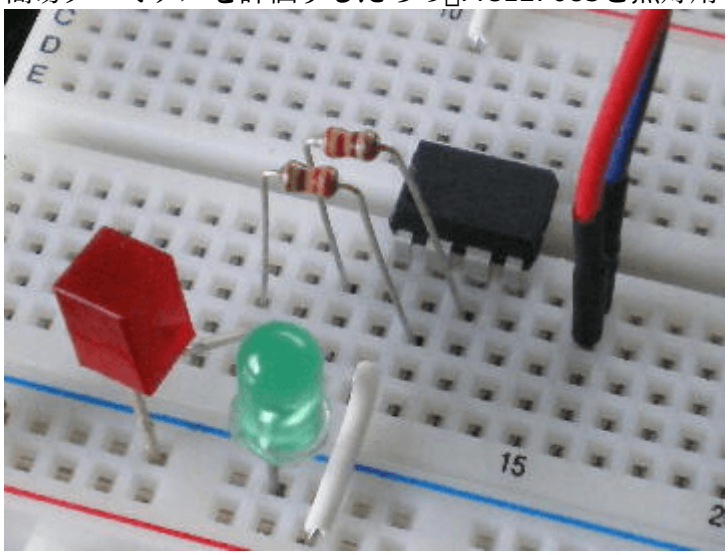


左側:PIC18F2620です。 右側:4×4のKeypadです。

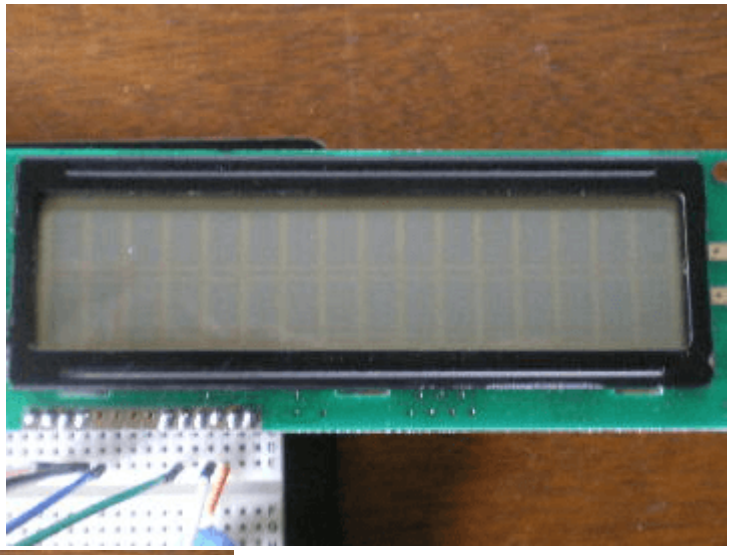




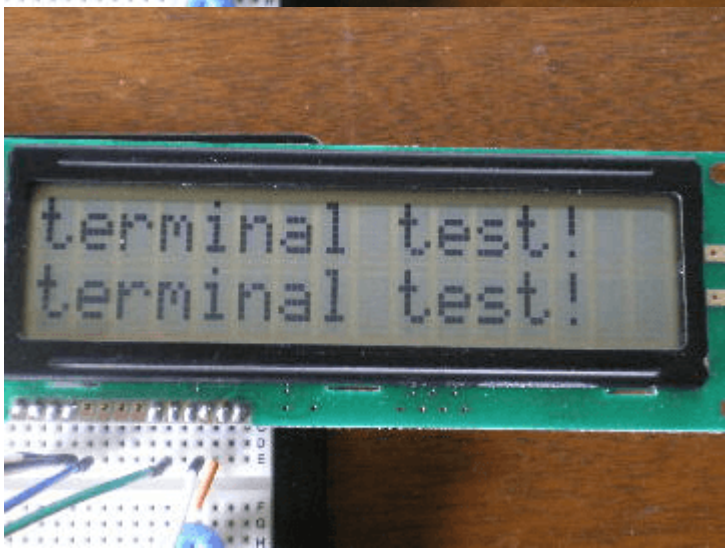
簡易ターミナルを評価するための□PIC12F683と点灯用のLEDです。



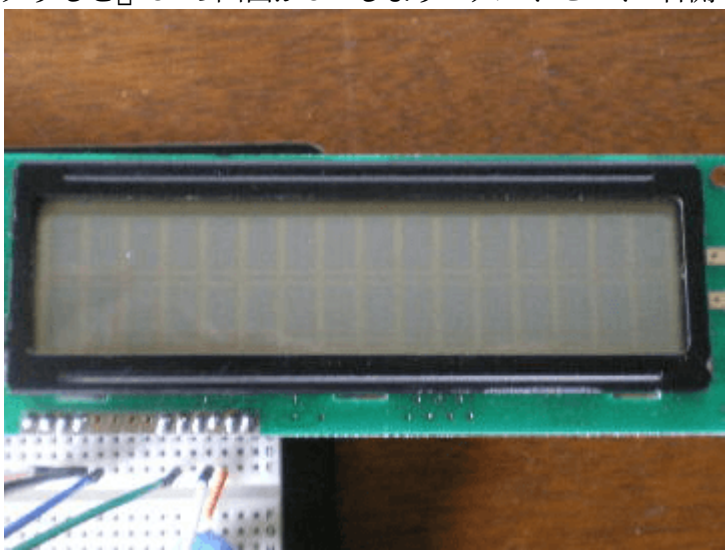
テストモード<左側:起動直後は、LCDの画面はクリアされます> テストモード<右側:1~0のキーを順次クリックすると、上段に表示されます>



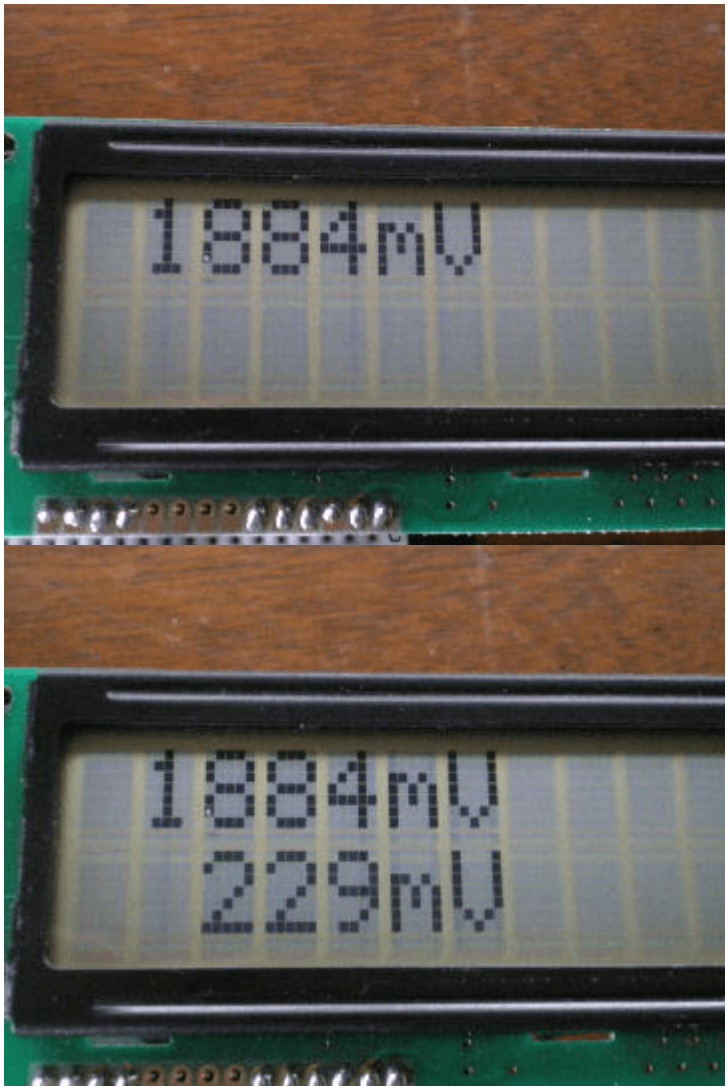
テストモード<左側: “ # “ をクリックすると、文字列“terminal test!”が上段に表示されます> テストモード<右側:“D“と “ # “ を順次クリックすると、カーソルが2行目に移動し、文字列が表示されます>



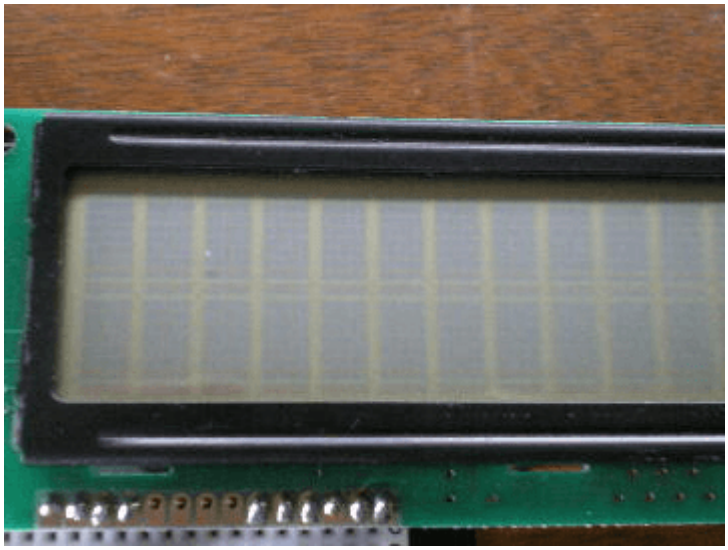
テストモード<左側:“A”をクリックすると□LCDの画面がOFFします> テストモード<右側:“B”をクリックすると□LCDの画面がONします>



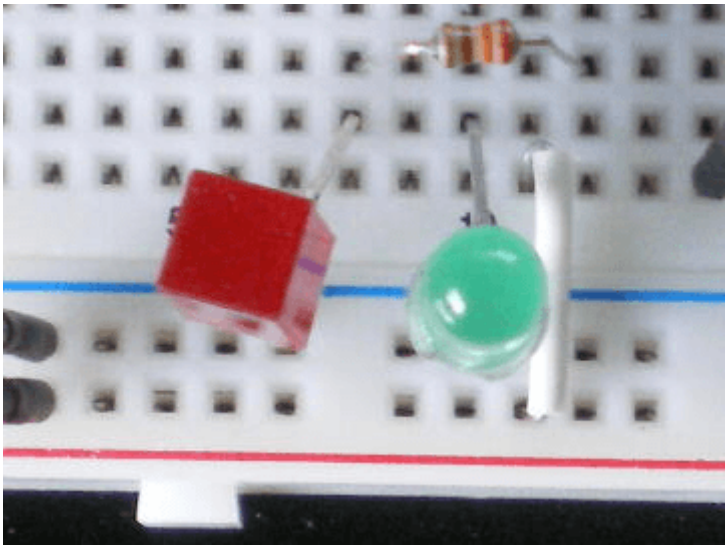
通常モード<左側:“A“をクリックすると、装置側のCH1のアナログデータがLCDの上段に表示されます>
通常モード<左側:“B”をクリックすると、装置側のCH2のアナログデータがLCDの下段に表示されます>



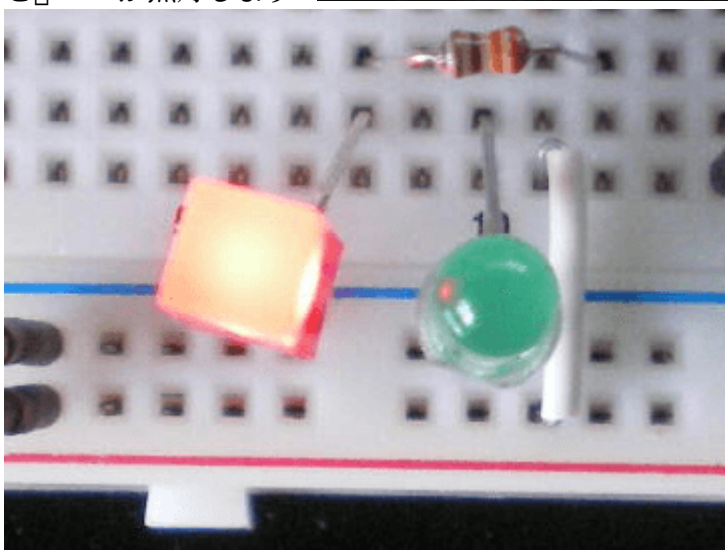
通常モード<左側: “ * “ をクリックするとLCDの画面がクリアされます>



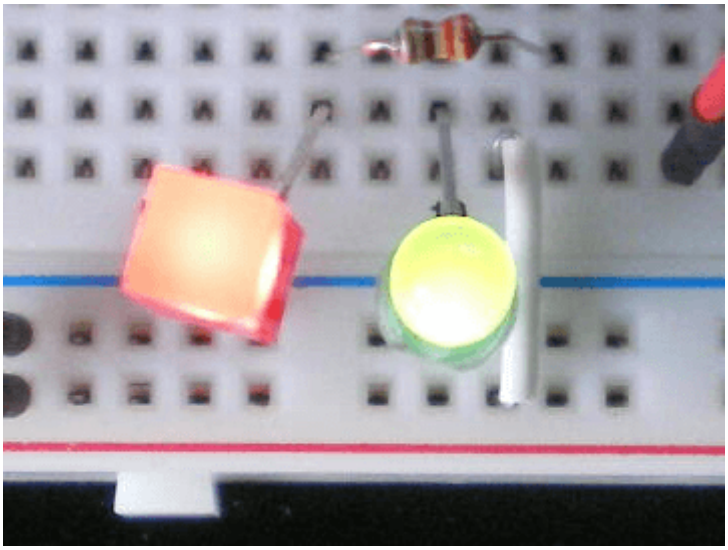
通常モード<左側:起動直後は、LED1とLED2は消灯しています> 通常モード<右側:“C”をクリックする



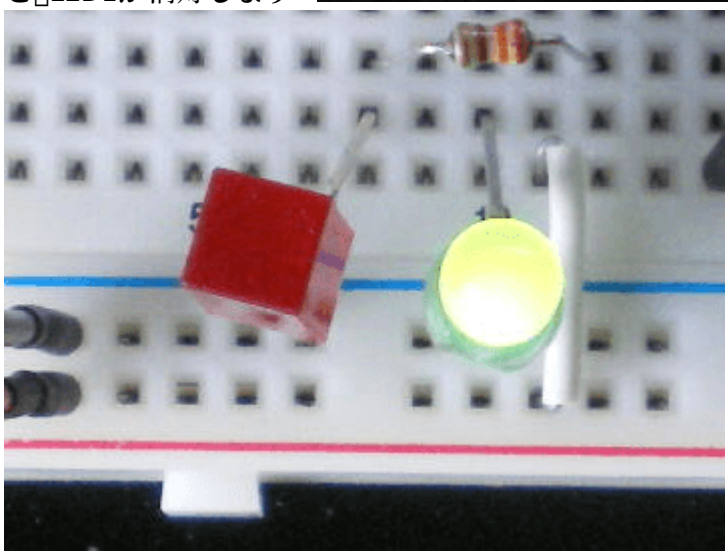
と□LED1が点灯します>



通常モード<左側:“D“をクリックすると□LED2が点灯します> 通常モード<右側:“C”をクリックする



と□LED1が消灯します>



如何ですか? 簡易ターミナルを使用することにより□PICを使った自作装置の開発(デバッグ作業など)や運用時の操作性が 大幅に改善されるのではないでしょうか? 例えば、自作の電源装置への応用を考えると、次のように改善されます。

- 簡易ターミナル未使用→出力電圧をボリュームやロータリーエンコードエンコーダで設定します。
- 簡易ターミナル使用→出力電圧をKeypadでダイレクト設定(例えば□“13.8V“)できます。