CHAPTER 4

mikroC Libraries

第４章　mikroC　libraries（mikroC ライブラリ）

ｍｉｋｒｏCは、　あなたのプリケーションをより早くかつより容易に開発することを援助する組込み及びライブラリ　ルーチンを提供する。

ADC、CAN、USART、SPI、I2C、1−Wire、LCD、PWM、RS485、Sirial　Ethernet、Toshiba　GLCD、Port　Expander、Sirial　GLCD、数値フォーマット、ビット操作及びその他多数のためのライブラリが、実践的ですぐ使用可能なコード　サンプルとともに含まれている。

BUILT-IN　ROUTINES（組込み型ルーチン）

mikroC コンパイラは役に立つ組込み型ユーティリティ関数を提供する。組込み型関数はインクルードされるべきいかなるヘッダーファイルも必要としない。つまり、あなたはプロジェクトのどの部分でもそれらを使用可能である。

組込み型ルーチンは“インライン”として実行される。つまり、コードが呼出しの場所に生成されるということだ。それ故に、呼出しは入れ子となった呼出しの制限を受けない。例外としては、実際のCルーチンである Vdelay_ms と Delay_Cyc があるだけである。

注) Lo、Hi、Higher、Highest 関数はコンパイラ内でそれ以上は実行できない。もしこれらの関数を使用したいなら、あなたはプロジェクトに built-in.h を含まねばならない。

Lo
Hi
Higher
Highest

Delay_us
Delay_ms
Vdelay_ms
Delay_Cyc
Clock_Khz
Clock_Mhz

Lo

原形	Unsigned short　Lo (long　number)
戻り値	下位８ビット（１バイト）の数、Bit0-7 を返す。
解説	関数は数の下位８バイトを返す。関数は数のビットパターンを翻訳しない。－レジスタで見つけた８ビットをまれに返す。これは“インライン”ルーチンである。コードは呼出しの位置に生成される。ゆえに呼出しは入れ子になった呼出しの制限を受けない。
必要事項	引数はスカラ型でなければならない。(つまり、算術型及びポインタ)
例	d = 0x1AC30F4 tmp=Lo(d) ; /Equals 0xF4

Hi

原形	Unsigned short　Hi (long　number)
戻り値	最下位 8 ビット（1 バイト）の数、Bit0-7 を返す。
解説	関数は数の下位 8 バイトを返す。関数は数のビットパターンを翻訳しない。－レジスタで見つけた 8 ビットをまれに返す。これは“インライン”ルーチンである。コードは呼出しの位置に生成される。ゆえに呼出しは入れ子になった呼出しの制限を受けない。
必要事項	引数はスカラ型でなければならない。（つまり、算術型及びポインタ）
例	d = 0x1AC30F4 tmp=Hi(d) ; /Equals 0xF4

Higher

原形	Unsigned short　Higher (long　number)
戻り値	最上位バイトの数、Bit16-23 の次を返す。
解説	関数は数の最上位バイトの次を返す。関数は数のビットパターンを翻訳しない。－レジスタで見つけた 8 ビットをまれに返す。 これは“インライン”ルーチンである。コードは呼出しの位置に生成される。ゆえに呼出しは入れ子になった呼出しの制限を受けない。
必要事項	引数はスカラ型でなければならない。（つまり、算術型及びポインタ）
例	d = 0x1AC30F4 tmp=Higner(d) ; // Equals 0xAC

Hignest

原形	Unsigned short　Higher (long　number)
戻り値	数の最上位バイト、Bit24-31 を返す。
解説	関数は数の最上位バイトを返す。関数は数のビットパターンを翻訳しない。－レジスタで見つけた 8 ビットをまれに返す。 これは“インライン”ルーチンである。コードは呼出しの位置に生成される。ゆえに呼出しは入れ子になった呼出しの制限を受けない。
必要事項	引数はスカラ型でなければならない。（つまり、算術型及びポインタ）
例	d = 0x1AC30F4 tmp=Hignest(d) ; // Equals 0x01

Delay_us

原形	Void Delay_us（**const**　time_in_us）
戻り値	なし
解説	time_in_us マクロセカンド（定数）の持続期間のソフトウェアディレイを生じる。適用可能な定数の範囲は発振子の周波数に依る。
必要事項	なし
例	Delay_us(10)；/*　10us 停止 */

Delay_ms

原形	Void Delay_ms（**const**　time_in_ms）
戻り値	なし
解説	time_in_ms ミリセカンド（定数）の持続期間のソフトウェアディレイを生じる。適用可能な定数の範囲は発振子の周波数に依る。
必要事項	なし
例	Delay_ms(1000)；/*　1 秒停止 */

Vdelay_ms

原形	Void Vdelay_ms（**const**　time_in_ms）
戻り値	なし
解説	time_in_ms ミリセカンド（変数）の持続期間のソフトウェアディレイを生じる。発生する遅延は delay_ms により生成する遅延ほど正確ではない。
必要事項	なし
例	Pause = 1000; Vdelay_ms(pause)；/*　約 1 秒停止 */

Delay_Cyc

原形	Void Delay_Cyc（**const**　Cycles_div_by_10）
戻り値	なし
解説	MCU クロックを基準とした遅延を発生する。遅延はMCU サイクル×入力パラメータ×10倍持続する。入力パラメータは３から２５５までの範囲であることを要する。 Delay_Cyc は組み込み型ルーチンというよりライブラリ関数であることに注意しよう。つまり、それはこのトピックで利便性のために現れる。
必要事項	なし
例	Delay_Cyc(10)；/*　100MCU サイクル停止 */

Clock_Khz

原形	Unsigned　Clock_Khz（void）
戻り値	直近の整数に丸められた KHz のデバイスクロック
解説	直近の整数に丸められた KHz でデバイスクロックを返す。
必要事項	なし
例	clk　=　Clock_KHz()；

Clock_MHz

原形	Unsigned　Clock_Mhz（void）
戻り値	直近の整数に丸められた MHz のデバイスクロック
解説	直近の整数に丸められた MHz でデバイスクロックを返す。
必要事項	なし
例	clk　=　Clock_MHz()；

LIBRARY　ROUTINES（ライブラリルーチン）

MikroC は、初期化とＰＩＣ　MCUの使用とそのモジュールを簡単にするライブラリ一式を提供する。ライブラリ関数はインクルードするべきいかなるヘッダーファイルも必要としない。あなたはプロジェクト内のどこでもそれらを使用可能である。

現在有効なライブラリは次の通り。

- ADC Library
- CAN Library
- CANSPI Library
- Compact Flash Library
- EEPROM Library
- Ethernet Library
- SPI Ethernet Library
- Flash Memory Library
- Graphic LCD Library
- T6963C Graphic LCD Library
- I²C Library
- Keypad Library
- LCD Library
- LCD Custom Library
- LCD8 Library
- Manchester Code Library
- Multi Media Card Library
- OneWire Library
- PS/2 Library
- PWM Library
- RS-485 Library
- Software I²C Library
- Software SPI Library
- Software UART Library
- Sound Library
- SPI Library
- USART Library
- USB HID Library
- Util Library
- SPI Graphic LCD Library
- Port Expander Library

- SPI LCD Library
- SPI LCD8 Library
- SPI T6963C Graphic LCD Library

Standard ANSI C Libraries

- ANSI C Ctype Library
- ANSI C Math Library
- ANSI C Stdlib Library
- ANSI C String Library

Miscellaneous Libraries

- Conversions Library
- Trigonometry Library
- sprint Library
- Setjmp Library
- Time Library

ADC　Library（アナログーデジタル変換コンバータライブラリ）

　ADC（アナログーデジタル変換コンバータ）モジュールは多くの PIC　MCU モデルに有効である。ライブラリ関数 Adc_Read がモジュールと共に最適な作業をあなたに提供する。

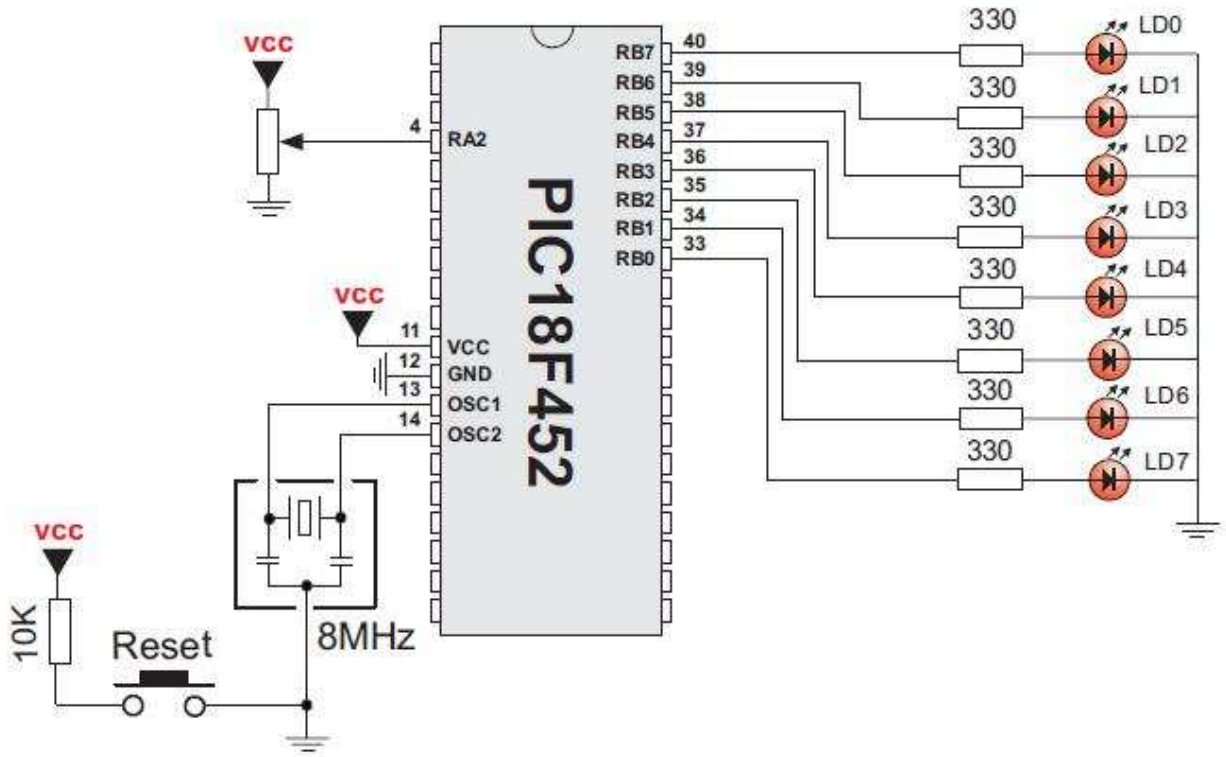
Adc_Read

プロトタイプ	Unsigned Adc_Read(char channel)；
戻り値	指定された ADC チャンネルからの１０ビット符号無し値
記述	RC クロックで動作するために PIC の内部 ADC モジュールを初期化する。 クロックは AD 変換（最小１２TAD）を実行するために必要な時間を決定する。 パラメータ channel1 はアナログ値を取得したいチャンネルを表す。 チャンネルー端子対応図については、適合する PIC　MCU の資料を参照すること。
要求	組込み式 ADC モジュールを持つ PIC　MCU。 あなたは指定のデバイス（PIC16／18 ファミリのデバイスが最も多い）のデータシート資料を調査すべきである。 　関数を使用する前に、必ず端子を入力として設定することで最適な TRISA ビットを構成すること。また、必要な端子をアナログ入力として設定し、且つ Vref（電圧レファレンス値）に設定すること。 現在次の PICmicros は関数をサポートしない。P18F2331、P18F2431,P18F4331 及び P18F4431
例	unsigned tmp； … Tmp＝Adc_Read(1)；　/*　チャンネル１からアナログ値を読み出す　*/

Library Example

```
/* This code snippet reads analog value from channel 2 and displays
   it on PORTD (lower 8 bits) and PORTB (2 most significant bits). */

unsigned temp_res;

void main() {
  ADCON1 = 0x80;    // Configure analog inputs and Vref
  TRISA  = 0xFF;    // PORTA is input
  TRISB  = 0x3F;    // Pins RB7, RB6 are outputs
  TRISD  = 0;       // PORTD is output

  do {
    temp_res = Adc_Read(2);      // Get results of AD conversion
    PORTD = temp_res;            // Send lower 8 bits to PORTD
    PORTB = temp_res >> 2;       // Send 2 most significant bits to RB7, RB6
  } while(1);
}
```

Hardware connection



CAN　Library（CAN ライブラリ）

mikroC は CAN モジュールでの動作のためのライブラリ（ドライバ）を提供する。

CAN はエラー保護や情報伝達、自己診断、及び障害封じ込め機能を有する非常に強固なプロトコルである。
不完全な CAN データ及びリモートフレームは自動的にイーサネット同様に再転送される。

データ転送レートは、40m以下のネットワーク長で 1 Mbit まで、250mケーブルで 250Kbps まで変化する。
また、さらに大きなネットワーク長では、規格により定義される最小ビットレート 200Kbit／s 以下とより低くなる。
使われるケーブルはシールドされたツイストペアで、最大ケーブル長は 1000mである。

CAN は２つのメッセージフォーマットをサポートする。

11 の識別子ビットを有する標準フォーマットと 29 識別子ビットを有する拡張フォーマット

注意 : CAN は現在 P18XXX8　PICmikros でのみサポートされる。マイクロコントローラは CAN バスに接続された CAN トランシーバ（MCP2551 または同等品）に接続されねばならない。

注意 : いくつかの関数を使用するために必要な CAN 定数を必ず確認しよう。145 ページを参照せよ。

Library　Routine

```
CANSetOperationMode
CANGetOperationMode
CANInitialize
CANSetBaudRate
CANSetMask
CANSetFilter
CANRead
CANWrite
```

次のルーチンはコンパイラのみにより内部的な使用を目的とするものである。

```
RegsToCANID
CANIDToRegs
```

CANStOperationMode

原型	`void CANSetOperationMode(char mode, char wait_flag);`
戻り値	なし
解説	リクエステッドモードへ CAN を設定する。即ち、CANSTAT へモードをコピーする。変数 mode は CAN_OP_MODE 定数（CAN 定数を参照せよ。）の一つを必要とする。 変数 wait_flag は 0 または 0xFF のいずれかをとる必要がある。 もし 0xFF に設定すると、これはブロッキング呼出しである。ー関数はリクエステッドモードが設定されるまで“return”しない。(戻らない) もし 0 に設定されると、これは非ブロッキング呼出しである。もし CAN モジュールがリクエステッドモードに切り替えられるかそうでないならば、それは確認されない。 発信者は、モード指定操作を実行する前に、誤り訂正操作モードを確認するため関数 CANGetOperationMode を使用しなければならない。
要求事項	CAN ルーチンは現在 P18XXX8　PICmicros によってのみサポートされる。マイクロコントローラは CAN バスに接続された CAN トランシーバ（MCP2551 または同等品）へ接続されねばならない。
例	`CANSetOperationMode(CAN_MODE_CONFIG, 0xFF);`

CANGetOperationMode

原型	`char CANGetOperationMode(void);`
戻り値	現在の opmode
解説	関数は現在の CAN モジュールのオペレーショナルモードを返す。
要求事項	CAN ルーチンは現在 P18XXX8　PICmicros によってのみサポートされる。マイクロコントローラは CAN バスに接続された CAN トランシーバ（MCP2551 または同等品）へ接続されねばならない。
例	`if (CANGetOperationMode() == CAN_MODE_NORMAL) { ... };`

CANInitialize

原型	`void CANInitialize(char SJW, char BRP, char PHSEG1, char PHSEG2, char PROPSEG, char CAN_CONFIG_FLAGS);`
戻り値	なし

解説	CAN を初期化する。全ての保留となっている送信は中止される。
	全てのメッセージを許可するため全てのマスクレジスタを 0 に設定する。
	コンフィグモードは内部的にこの関数により設定される。
	この関数の実行と同時に、ノーマルモードが設定される。
	
	フィルタレジスタはフラグ値に従い設定される。
	
	`if (CAN_CONFIG_FLAGS & CAN_CONFIG_VALID_XTD_MSG != 0)`
	`// 全フィルタを XTD_MSG へ設定する`
	`else if (config & CONFIG_VALID_STD_MSG != 0)`
	`// 全フィルタを STD_MSG へ設定する`
	`else`
	`// 全フィルタを STD へ半分設定し、XTD_MSG へ休止設定する`
	
	変数：
	
	SJW　データシート（1－4）の 18XXX8 に定義される
	BRP　データシート（1－64）の 18XXX8 に定義される
	PHSEG1　データシート（1－8）の 18XXX8 に定義される
	PHSEG2　データシート（1－8）の 18XXX8 に定義される
	PROPSEG　データシート（1－8）の 18XXX8 に定義される
	CAN_CONFIG_FLAGS は以前に定義された定数（CAN 定数を参照せよ）から構成される。

要求事項	CAN ルーチンは現在 P18XXX8　PICmicros によってのみサポートされる。マイクロコントローラは CAN バスに接続された CAN トランシーバ（MCP2551 または同等品）へ接続されねばならない。

例	`init = CAN_CONFIG_SAMPLE_THRICE &`
	`CAN_CONFIG_PHSEG2_PRG_ON &`
	`CAN_CONFIG_STD_MSG &`
	`CAN_CONFIG_DBL_BUFFER_ON &`
	`CAN_CONFIG_VALID_XTD_MSG &`
	`CAN_CONFIG_LINE_FILTER_OFF;`
	`...`
	`CANInitialize(1, 1, 3, 3, 1, init);  // initialize CAN`

CANSetBaudRate

原型	`void CANSetBaudRate(char SJW, char BRP, char PHSEG1, char PHSEG2, char PROPSEG, char CAN_CONFIG_FLAGS);`
戻り値	なし
解説	CAN ボーレートを初期化する。CAN プロトコルの複雑さのため、あなたは単純に bps 値を無理に決めることはできない。代わりに CAN がコンフィグモードの時、この関数を使用すること。詳細についてはデータシートを参照せよ。 変数： SJW　データシート（1−4）の 18XXX8 に定義される BRP　データシート（1−64）の 18XXX8 に定義される PHSEG1　データシート（1−8）の 18XXX8 に定義される PHSEG2　データシート（1−8）の 18XXX8 に定義される PROPSEG　データシート（1−8）の 18XXX8 に定義される CAN_CONFIG_FLAGS は以前に定義された定数（CAN 定数を参照せよ）から構成される。
要求事項	CAN はコンフィグモードでなければならない。さもなくば、関数は無視される。

例	`init = CAN_CONFIG_SAMPLE_THRICE & CAN_CONFIG_PHSEG2_PRG_ON & CAN_CONFIG_STD_MSG & CAN_CONFIG_DBL_BUFFER_ON & CAN_CONFIG_VALID_XTD_MSG & CAN_CONFIG_LINE_FILTER_OFF;`
	`...`
	`CANSetBaudRate(1, 1, 3, 3, 1, init);`

CANSetMask

原型	`void CANSetMask(char CAN_MASK, long value, char CAN_CONFIG_FLAGS);`
戻り値	なし
解説	関数はメッセージの高度なフィルタリングのためのマスクを設定する。与えられた値はバッファマスクレジスタを最適に合わせたビット値である。 変数： CAN_MASK は以前に定義された定数値（CAN 定数を参照せよ）の一つである。 value はマスクレジスタ値である。 CAN_CONFIG_FLAGS はフィルタへのメッセージ型、CAN_CONFIG_XTD_MSG または CAN_CONFIG_STD_MSG を選択する。
要求事項	CAN はコンフィグモードでなければならない。さもなくば、関数は無視される。

例	// 全マスクビットを１に設定する。つまり、全てフィルタをかけたビットという意味となる。
	`CANSetMask(CAN_MASK_B1, -1, CAN_CONFIG_XTD_MSG);`
	// －１は 0xFFFFFFFF を書き込むための正に手軽な方法である。補数は要領の良いやり方であり、１で満たすことになるだろう。

CANSetFilter

原型	`void CANSetFilter(char CAN_FILTER, long value, char CAN_CONFIG_FLAGS);`
戻り値	なし
解説	関数はメッセージの高度なフィルタリングのためのマスクを設定する。与えられた値はバッファマスクレジスタを最適に合わせたビット値である。 変数： CAN_FILETR は以前に定義された定数値（CAN 定数を参照せよ）の一つである。 value はフィルタレジスタ値である。 CAN_CONFIG_FLAGS はフィルタへのメッセージ型、CAN_CONFIG_XTD_MSG または CAN_CONFIG_STD_MSG を選択する。
要求事項	CAN はコンフィグモードでなければならない。さもなくば、関数は無視される。
例	// フィルタ B1_F1 のｉｄを３に設定する。 `CANSetFilter(CAN_FILTER_B1_F1, 3, CAN_CONFIG_XTD_MSG);`

CANRead

原型	`char CANRead(long *id, char *data, char *datalen, char *CAN_RX_MSG_FLAGS);`
戻り値	受信バッファからのメッセージ、またはメッセージが無ければ0
解説	関数は受信バッファからメッセージを読み出す。もし少なくとも１つの満杯の受信バッファが見つかると、それが抜き出され、且つ戻される。もし見つからなければ関数は0 を返す。 変数： id はメッセージ識別子である。 data は8 バイト長までのバイト配列である。 datalen はデータ長１－8 である。 CAN_RX_MSG_FLAGS は定数から構成される値である。（CAN 定数を参照せよ。）
要求事項	CAN は受信可能なモードでなければならない。
例	`char rcv, rx, len, data[8]; long id;` `rcv = CANRead(id, data, len, 0);`

CANWrite

原型	`char CANWrite(long id, char *data, char datalen, char CAN_TX_MSG_FLAGS);`
戻り値	もしメッセージが順番待ちで無い（バッファ満杯）の場合、0 を返す。
解説	もし少なくとも１つの空の送信バッファが見つかると、関数は送信のためにキューへメッセージを送る。もしバッファが満杯ならば、関数は0 を返す。 変数： id はメッセージ識別子である。11－29 ビットがメッセージ型に依って使われる。（標準または拡張） data は8 バイト長までのバイト配列である。 datalen はデータ長１－8 である。 CAN_TX_MSG_FLAGS は定数から構成される値である。（CAN 定数を参照せよ。）
要求事項	CAN はノーマルモードでなければならない。

例	`char tx, data; long id;`
	`tx = CAN_TX_PRIORITY_0 & CAN_TX_XTD_FRAME;`
	`CANWrite(id, data, 2, tx);`

CAN　Constants

CAN_OP_MODE 定数は CAN オペレーションモードを定義する。
関数 CANSetOperationMode はその引数としてこれらの内１つを要求する。

```
#define CAN_MODE_BITS      0xE0     // アクセスモードビットにそれを使用する     ts
#define CAN_MODE_NORMAL    0
#define CAN_MODE_SLEEP     0x20
#define CAN_MODE_LOOP      0x40
#define CAN_MODE_LISTEN    0x60
#define CAN_MODE_CONFIG    0x80
```

CAN_CONFIG_FLAGS

CAN_CONFIG_FLAGS 定数は CAN モジュールのコンフィグレーションに関わるフラグを定義する。
関数 CAN_Initialize と CAN_SetBaudRate はそれらの引数としてこれらの内の１つ（またはビット単位の組合せ）を要求する。

```
#define CAN_CONFIG_DEFAULT              0xFF    // 11111111

#define CAN_CONFIG_PHSEG2_PRG_BIT       0x01
#define CAN_CONFIG_PHSEG2_PRG_ON        0xFF    // XXXXXXX1
#define CAN_CONFIG_PHSEG2_PRG_OFF       0xFE    // XXXXXXX0

#define CAN_CONFIG_LINE_FILTER_BIT      0x02
#define CAN_CONFIG_LINE_FILTER_ON       0xFF    // XXXXXX1X
#define CAN_CONFIG_LINE_FILTER_OFF      0xFD    // XXXXXX0X

#define CAN_CONFIG_SAMPLE_BIT           0x04
#define CAN_CONFIG_SAMPLE_ONCE          0xFF    // XXXXX1XX
#define CAN_CONFIG_SAMPLE_THRICE        0xFB    // XXXXX0XX

#define CAN_CONFIG_MSG_TYPE_BIT         0x08
#define CAN_CONFIG_STD_MSG              0xFF    // XXXX1XXX
#define CAN_CONFIG_XTD_MSG              0xF7    // XXXX0XXX

// continues..
```

```
// ..continued

#define CAN_CONFIG_DBL_BUFFER_BIT       0x10
#define CAN_CONFIG_DBL_BUFFER_ON        0xFF    // XXX1XXXX
#define CAN_CONFIG_DBL_BUFFER_OFF       0xEF    // XXX0XXXX

#define CAN_CONFIG_MSG_BITS             0x60
#define CAN_CONFIG_ALL_MSG              0xFF    // X11XXXXX
#define CAN_CONFIG_VALID_XTD_MSG        0xDF    // X10XXXXX
#define CAN_CONFIG_VALID_STD_MSG        0xBF    // X01XXXXX
#define CAN_CONFIG_ALL_VALID_MSG        0x9F    // X00XXXXX
```

あなたはこれらの値の他にコンフィグバイトを作るためビット単位の AND（&）を使用可能である。
例：

```
init =  CAN_CONFIG_SAMPLE_THRICE  & CAN_CONFIG_PHSEG2_PRG_ON &
        CAN_CONFIG_STD_MSG        & CAN_CONFIG_DBL_BUFFER_ON &
        CAN_CONFIG_VALID_XTD_MSG  & CAN_CONFIG_LINE_FILTER_OFF;
//...
CANInitialize(1, 1, 3, 3, 1, init);    // initialize CAN
```

CAN_TX_MSG_FLAGS

CAN_TX_MSG_FLAGS は CAN メッセージの送信に関わるフラグである。

```
#define CAN_TX_PRIORITY_BITS      0x03
#define CAN_TX_PRIORITY_0         0xFC  // XXXXXX00
#define CAN_TX_PRIORITY_1         0xFD  // XXXXXX01
#define CAN_TX_PRIORITY_2         0xFE  // XXXXXX10
#define CAN_TX_PRIORITY_3         0xFF  // XXXXXX11

#define CAN_TX_FRAME_BIT          0x08
#define CAN_TX_STD_FRAME          0xFF  // XXXXX1XX
#define CAN_TX_XTD_FRAME          0xF7  // XXXXX0XX

#define CAN_TX_RTR_BIT            0x40
#define CAN_TX_NO_RTR_FRAME       0xFF  // X1XXXXXX
#define CAN_TX_RTR_FRAME          0xBF  // X0XXXXXX
```

あなたは適正なフラグに合わせるためビット単位の AND（&）を使用可能である。
例：

```
// CANSendMessege と共に使われる値を作る
send_config = CAN_TX_PRIORITY_0 && CAN_TX_XTD_FRAME &
              CAN_TX_NO_RTR_FRAME;
//...
CANSendMessage(id, data, 1, send_config);
```

CAN_RX_MSG_FLAGS

CAN_RX_MSG_FLAGS はCAN メッセージの受信に関わるフラグである。もし特別なビットが設定されると、相当の意味するところはTRUE（真）であり、さもなくばFALSE（偽）であろう。

```
#define CAN_RX_FILTER_BITS  0x07  // Use it to access filter bits
#define CAN_RX_FILTER_1     0x00
#define CAN_RX_FILTER_2     0x01
#define CAN_RX_FILTER_3     0x02
#define CAN_RX_FILTER_4     0x03
#define CAN_RX_FILTER_5     0x04
#define CAN_RX_FILTER_6     0x05
#define CAN_RX_OVERFLOW     0x08  // Set if Overflowed; else clear
#define CAN_RX_INVALID_MSG  0x10  // Set if invalid; else clear
#define CAN_RX_XTD_FRAME    0x20  // Set if XTD msg; else clear
#define CAN_RX_RTR_FRAME    0x40  // Set if RTR msg; else clear
#define CAN_RX_DBL_BUFFERED 0x80  // Set if msg was
                                  //   hardware double-buffered
```

あなたは適正なフラグに合わせるためビット単位のAND（&）を使用可能である。

例：

```
if (MsgFlag & CAN_RX_OVERFLOW != 0) {
   ...  //  レシーバオーバーフローが発生した。前のメッセージが失われた。
}
```

CAN_MASK

CAN_MASK 定数はマスクコードを定義する。

関数CANSetMask は引数としてこれらの内の１つを要求する。

```
#define CAN_MASK_B1   0
#define CAN_MASK_B2   1
```

CAN_FILTER

CAN_FILTER 定数はフィルタコードを定義する。

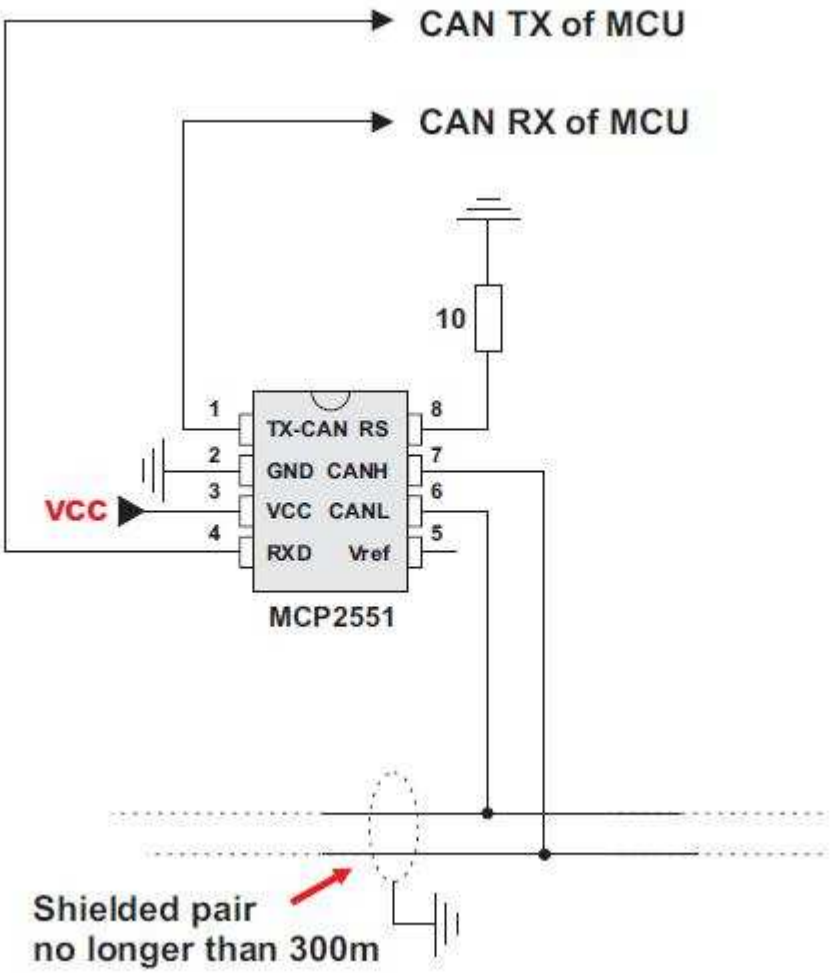
関数CANSetFilter は引数としてこれらの内の１つを要求する。

```
#define CAN_FILTER_B1_F1   0
#define CAN_FILTER_B1_F2   1
#define CAN_FILTER_B2_F1   2
#define CAN_FILTER_B2_F2   3
#define CAN_FILTER_B2_F3   4
#define CAN_FILTER_B2_F4   5
```

```
unsigned short aa, aa1, len, aa2;
unsigned char data[ 8] ;
long id;
unsigned short zr, cont, oldstate;

//........

void main() {
  PORTC = 0;
  TRISC = 0;
  PORTD = 0;
  TRISD = 0;
  aa  = 0;
  aa1 = 0;
  aa2 = 0;

  // Form value to be used with CANSendMessage
  aa1 =  CAN_TX_PRIORITY_0 &
         CAN_TX_XTD_FRAME &
         CAN_TX_NO_RTR_FRAME;

  // Form value to be used with CANInitialize
  aa =   CAN_CONFIG_SAMPLE_THRICE      &
         CAN_CONFIG_PHSEG2_PRG_ON      &
         CAN_CONFIG_STD_MSG            &
         CAN_CONFIG_DBL_BUFFER_ON      &
         CAN_CONFIG_VALID_XTD_MSG      &
         CAN_CONFIG_LINE_FILTER_OFF;

  data[ 0]  = 0;

  // Initialize CAN
  CANInitialize(1,1,3,3,1,aa);

  // Set CAN to CONFIG mode
  CANSetOperationMode(CAN_MODE_CONFIG,0xFF);

  id = -1;
```

```
  // Set all mask1 bits to ones
  CANSetMask(CAN_MASK_B1,ID,CAN_CONFIG_XTD_MSG);

  // Set all mask2 bits to ones
  CANSetMask(CAN_MASK_B2,ID,CAN_CONFIG_XTD_MSG);

  // Set id of filter B1_F1 to 3
  CANSetFilter(CAN_FILTER_B2_F3,3,CAN_CONFIG_XTD_MSG);

  // Set CAN to NORMAL mode
  CANSetOperationMode(CAN_MODE_NORMAL,0xFF);

  PORTD = 0xFF;
  id = 12111;
  CANWrite(id,data,1,aa1);                    // Send message via CAN

  while (1) {
    oldstate = 0;
    zr = CANRead(&id, data , &len, &aa2);
    if ((id == 3) & zr) {
      PORTD = 0xAA;
      PORTC = data[ 0];                       // Output data at PORTC
      data[ 0]++ ;

      // If message contains two data bytes, output second byte at PORTD
      if (len == 2)  PORTD = data[ 1];

      data[ 1] = 0xFF;
      id = 12111;
      CANWrite(id, data, 2,aa1);              // Send incremented data back
    }
  }
} //~!
```

Hardware Connection



CANSPI　Library（CANSPI ライブラリ）

SPI モジュールは多くの PICmicros で有効である。
mikroC は、外付けの CAN モジュール（MCP2515 または MCP2510 のような）を動かすためのライブラリ（ドライバ）を提供する。

mikroC では、CAN ライブラリの各ルーチンは同じ構文を伴うその CANSPI の（組合せの）一方を有する。
コントローラエリアネットワークに関する更なる情報は、CAN ライブラリを調べること。
実際の通信速度は SPI に依存し、“本当の”CAN よりも間違いなく遅いことに注意すること。

注意：いくつかの関数を使用するにあたり、必ず CAN 定数を確認すること。145 ページを見よ。

注意：CANSPI を初期化する前に SPI_Init（ ）が呼び出されねばならない。

Library　Routine

```
CANSPISetOperationMode
CANSPIGetOperationMode
CANSPIInitialize
CANSPISetBaudRate
CANSPISetMask
CANSPISetFilter
CANSPIRead
CANSPIWrite
```

次のルーチンはコンパイラのみにより内部的に利用するためのものである。

```
RegsToCANSPIID
CANSPIIDToRegs
```

CANSPISetOperationMode

原型	`void CANSPISetOperationMode(char mode, char wait_flag);`
戻り値	なし
解説	リクエステッドモードに CAN を設定する。即ち、CANSTAT へモードをコピーする。変数 mode は CAN_OP_MODE 定数（CAN 定数、145 ページを見よ）の一つである必要がある。 変数 wait_flag は 0 または 0xFF のいづれかである必要がある。もし 0xFF に設定されると、これはブロッキング呼出しとなる。ー関数は、リクエステッドモードが設定されるまで“戻らない”。もし 0 に設定されると、これは非ブロッキングモードとなる。それは、CAN モジュールがリクエステッドモードに切り替えられたかそうでないかを確認しない。 モード指定操作を実行する前に、呼出し側は誤ったオペレーションモードを確認するため関数 CANSPISetOperationMode を使用しなければならない。
要求事項	CANSPI 関数は PORTC 上に SPI インターフェースを有する何らかの PIC　MCU によりサポートされる。また MCP2510 または MCP2515 の CS 端子は、RC0 へ接続されねばならない。
例	`CANSPISetOperationMode(CAN_MODE_CONFIG, 0xFF);`

CANSPIGetOperationMode

原型	`char CANSPIGetOperationMode(void);`
戻り値	現在のオペレーションモード
解説	関数は CAN モジュールの現在のオペーレーショナルモードを返す。
要求事項	なし
例	`if (CANSPIGetOperationMode() == CAN_MODE_NORMAL) { ... };`

CANSPIInitialize

原型	`void CANSPIInitialize(char SJW, char BRP, char PHSEG1, char PHSEG2, char PROPSEG, char CAN_CONFIG_FLAGS, char * RstPort, char RstPin, char * CSPort, char CSPin);`
戻り値	なし

解説	CANSPI を初期化する。全ての保留中の送信は停止される。全メッセージを許可するため、全てのマスクレジスタを 0 に設定する。
	
	フィルタレジスタがフラグ値により設定される。
	
	`if (CAN_CONFIG_FLAGS & CAN_CONFIG_VALID_XTD_MSG != 0)`
	`// 全フィルタを XTD_MSG へ設定する`
	`else if (config & CONFIG_VALID_STD_MSG != 0)`
	`// 全フィルタを STD_MSG へ設定する`
	`else`
	`// 全フィルタを STD へ半分設定し、XTD_MSG へ休止設定する`
	
	変数：
	
	SJW　データシート（1−4）の 18XXX8 に定義される
	BRP　データシート（1−64）の 18XXX8 に定義される
	PHSEG1　データシート（1−8）の 18XXX8 に定義される
	PHSEG2　データシート（1−8）の 18XXX8 に定義される
	PROPSEG　データシート（1−8）の 18XXX8 に定義される
	CAN_CONFIG_FLAGS は以前に定義された定数（CAN 定数、145 ページを参照せよ）から構成される。

要求事項	CANSPI を初期化する前に、SPI_Init()が呼び出されねばならない。

例	`init = CAN_CONFIG_SAMPLE_THRICE &`
	`CAN_CONFIG_PHSEG2_PRG_ON &`
	`CAN_CONFIG_STD_MSG &`
	`CAN_CONFIG_DBL_BUFFER_ON &`
	`CAN_CONFIG_VALID_XTD_MSG &`
	`CAN_CONFIG_LINE_FILTER_OFF;`
	`...`
	`// initialize external CAN module`
	`CANSPIInitialize( 1,1,3,3,1,init, &PORTC, 2, &PORTC, 0);`

CANSPISetBaudRate

原型	`void CANSPISetBaudRate(char SJW, char BRP, char PHSEG1, char PHSEG2, char PROPSEG, char CAN_CONFIG_FLAGS);`
戻り値	なし
解説	CANSPI ボーレートを設定する。CAN プロトコルの複雑さのため、あなたは簡単に bps 値を無理やり決めることはできない。代わりに、CANSPI がコンフィグモードにある場合、この関数を使用すること。詳細はデータシートを参照せよ。 変数： SJW　データシート（1−4）の 18XXX8 に定義される BRP　データシート（1−64）の 18XXX8 に定義される PHSEG1　データシート（1−8）の 18XXX8 に定義される PHSEG2　データシート（1−8）の 18XXX8 に定義される PROPSEG　データシート（1−8）の 18XXX8 に定義される CAN_CONFIG_FLAGS は以前に定義された定数（CAN 定数を参照せよ）から構成される。
要求事項	CANSPI はコンフィグモードでなければならない。さもなくば関数は無視される。

例	`init = CAN_CONFIG_SAMPLE_THRICE & CAN_CONFIG_PHSEG2_PRG_ON & CAN_CONFIG_STD_MSG & CAN_CONFIG_DBL_BUFFER_ON & CAN_CONFIG_VALID_XTD_MSG & CAN_CONFIG_LINE_FILTER_OFF;`
	`...`
	`CANSPISetBaudRate(1, 1, 3, 3, 1, init);`

CANSPISetMask

原型	`void CANSPISetMask(char CAN_MASK, long value, char CAN_CONFIG_FLAGS);`
戻り値	なし
解説	関数は高度なメッセージのフィルタリングのためのマスクを設定する。 与えられた値はバッファマスクレジスタを適正な値にするビットである。 変数： CAN_MASK は以前に定義した定数値（CAN 定数を見よ）の一つである。 Value はマスクレジスタ値である。 CAN_CONFIG_FLAGS はフィルタ CAN_CONFIG_STD_MSG または CAN_CONFIG_XTD_MSG いづれかへのメッセージ型を選択する。
要求事項	CANSPI はコンフィグモードでなければならない。さもなくば関数は無視される。

例	`// Set all mask bits to 1, i.e. all filtered bits are relevant:`
	
	`CANSPISetMask(CAN_MASK_B1, -1, CAN_CONFIG_XTD_MSG);`
	
	`/* Note that -1 is just a cheaper way to write 0xFFFFFFFF.`
	`Complement will do the trick and fill it up with ones. */`

CANSPISetFilter

原型	`void CANSPISetFilter(char CAN_FILTER, long value, char CAN_CONFIG_FLAGS);`
戻り値	なし
解説	関数は高度なメッセージのフィルタリングのためのマスクを設定する。 与えられた値はバッファマスクレジスタを適正な値にするビットである。 変数： CAN_Filter は以前に定義した定数値（CAN 定数を見よ）の一つである。 Value はフィルタレジスタ値である。 CAN_CONFIG_FLAGS はフィルタ CAN_CONFIG_STD_MSG または CAN_CONFIG_XTD_MSG いづれかへのメッセージ型を選択する。
要求事項	CANSPI はコンフィグモードでなければならない。さもなくば関数は無視される。
例	`/* Set id of filter B1_F1 to 3: */` `CANSPISetFilter(CAN_FILTER_B1_F1, 3, CAN_CONFIG_XTD_MSG);`

CANSPIRead

原型	`char CANSPIRead(long *id, char *data, char *datalen, char *CAN_RX_MSG_FLAGS);`
戻り値	受信バッファからのメッセージ、またはメッセージが無ければ0
解説	関数は受信バッファからメッセージを読み出す。もし少なくとも１つの満杯の受信バッファが見つかると、それが抜き出され、且つ戻される。もし見つからなければ関数は0 を返す。 変数： id はメッセージ識別子である。 data は8 バイト長までのバイト配列である。 datalen はデータ長１－8 である。 CAN_RX_MSG_FLAGS は定数から構成される値である。（CAN 定数を参照せよ。）
要求事項	CANSPI は受信可能なモードでなければならない。
例	`char rcv, rx, len, data[8]; long id;` `rcv = CANSPIRead(id, data, len, 0);`

CANSPIWrite

原型	`char CANSPIWrite(long id, char *data, char datalen, char CAN_TX_MSG_FLAGS);`
戻り値	もしメッセージが順番待ちで無い（バッファ満杯）の場合、0 を返す。
解説	もし少なくとも１つの空の送信バッファが見つかると、関数は送信のためにキューへメッセージを送る。もしバッファが満杯ならば、関数は0 を返す。 変数： id はメッセージ識別子である。11－29 ビットがメッセージ型に依って使われる。（標準または拡張） data は8 バイト長までのバイト配列である。 datalen はデータ長１－8 である。 CAN_TX_MSG_FLAGS は定数から構成される値である。（CAN 定数を参照せよ。）
要求事項	CAN はノーマルモードでなければならない。

例	`char tx, data; long id;`
	`tx = CAN_TX_PRIORITY_0 & CAN_TX_XTD_FRAME;`
	`CANSPIWrite(id, data, 2, tx);`

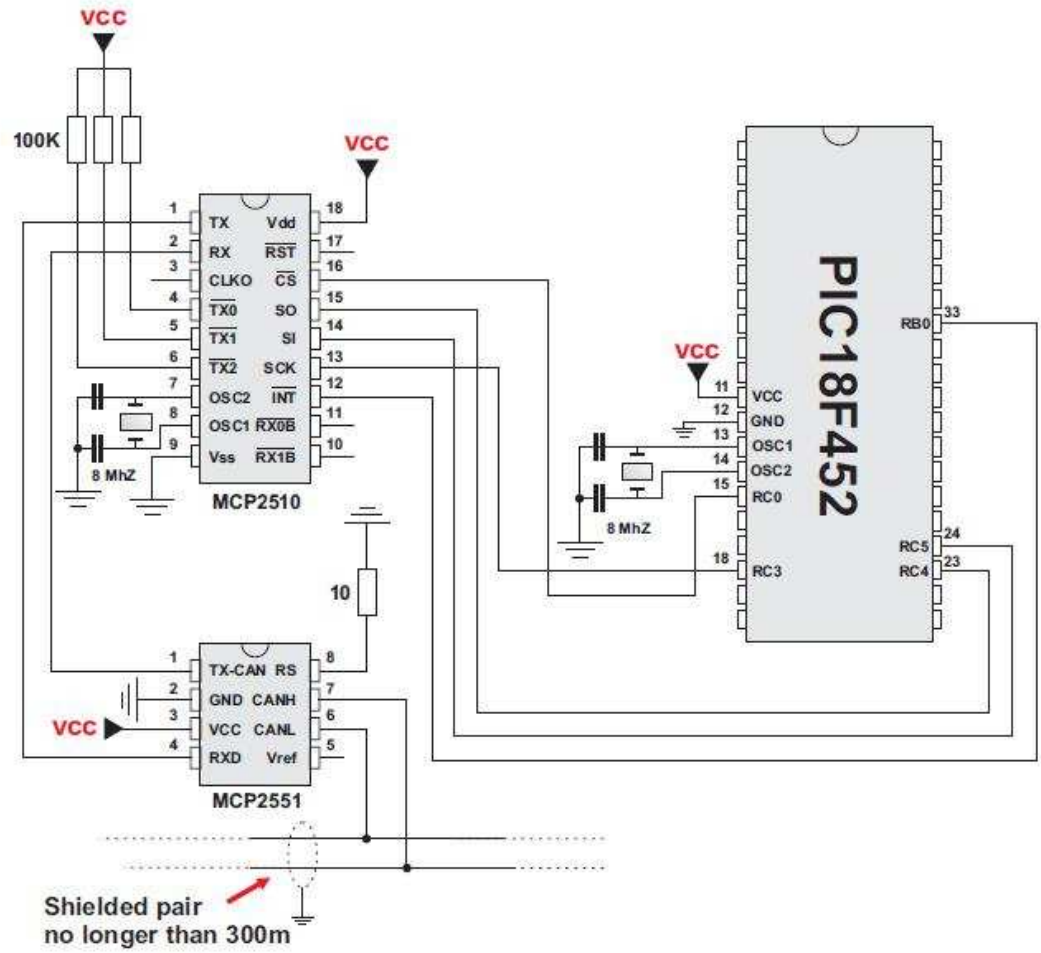
Library Examples

このコードはCANSPIプロトコルの簡単なデモである。
それは２つのPIC間で簡単なデータを交換し、データがバウンス（返送）毎に１増やされる。
データは目視確認のためPORTC（下位バイト）とPORTD（上位バイト）へ出力される。

```
char data[8],aa, aa1, len, aa2;
long id;
char zr;
const char _TRUE  = 0xFF;
const char _FALSE = 0x00;

void main(){
  TRISB = 0;
  Spi_Init();     // Initialize SPI module
  TRISC.F2 = 0;   // Clear (TRISC,2)
  PORTC.F2 = 0;   // Clear (PORTC,2)
  PORTC.F0 = 1;   // Set   (PORTC,0)
  TRISC.F0 = 0;   // Clear (TRISC,0)
  PORTD = 0;
  TRISD = 0;
  aa    = 0;
  aa1   = 0;
  aa2   = 0;


  // Form value to be used with CANSPIInitialize
  aa = CAN_CONFIG_SAMPLE_THRICE  &
       CAN_CONFIG_PHSEG2_PRG_ON  &
       CAN_CONFIG_STD_MSG        &
       CAN_CONFIG_DBL_BUFFER_ON  &
       CAN_CONFIG_VALID_XTD_MSG;

  PORTC.F2 = 1;   // Set (PORTC,2)

  // Form value to be used with CANSPISendMessage
  aa1 = CAN_TX_PRIORITY_0 &
        CAN_TX_XTD_FRAME &
        CAN_TX_NO_RTR_FRAME;

  PORTC.F0 = 1;   // Set (PORTC,0)


// continues ..
```

```
// .. continued

  Spi_Init(); // Initialize SPI

  // Initialize external CAN module
  CANSPIInitialize( 1,1,3,3,1,aa, &PORTC, 2, &PORTC, 0);

  // Set CANSPI to CONFIG mode
  CANSPISetOperationMode(CAN_MODE_CONFIG,_TRUE);
  ID = -1;

  // Set all mask1 bits to ones
  CANSPISetMask(CAN_MASK_B1,id,CAN_CONFIG_XTD_MSG);

  // Set all mask2 bits to ones
  CANSPISetMask(CAN_MASK_B2,id,CAN_CONFIG_XTD_MSG);

  // Set id of filter B1_F1 to 12111
  CANSPISetFilter(CAN_FILTER_B2_F4,12111,CAN_CONFIG_XTD_MSG);

  // Set CANSPI to NORMAL mode
  CANSPISetOperationMode(CAN_MODE_NORMAL,_TRUE);

  while (1) {
    zr = CANSPIRead(&id , &Data , &len, &aa2);     // Receive data, if any
    if (id == 12111 & zr )  {
      PORTB = data[0]++ ;                          // Output data on PORTB
      id = 3;
      Delay_ms(500);

      // Send incremented data back
      CANSPIWrite(id,&data,1,aa1);

      // If message contains 2 data bytes, output second byte at PORTD
      if (len == 2) PORTD = data[1];
    }
  }
}//~!
```

Hardware Connection



Compact　Flash　Library（コンパクトフラッシュライブラリ）

　コンパクトフラッシュライブラリは、コンパクトフラッシュカード（以下表記を略：CF とする）のデータをアクセスするためのルーチンを提供する。
CF カードは通常デジタルカメラに見出されるメモリ素子として広く使用されている。
大容量（８MB～２GB、またそれ以上）且つ標準で数ｕｓというすばらしいアクセスタイムが、マイクロコントローラの応用にあたりＣＦを非常に魅力的にしている。

ＣＦカードでは、データはセクタ、つまり通常５１２ｂｙｔｅ（少々古いモデルでは２５６ｂｙｔｅを有する）で構成される１セクタ、に分割される。
リード／ライト操作は直接実行されないが、５１２ｂｙｔｅバッファを経由して首尾よくアクセス可能である。
次のルーチンはＦＡＴ１６とＦＡＴ３２ファイルシステムでＣＦを使用することを可能にする。
ファイル操作用のルーチンはＦＡＴ１６ファイルシステムでのみ使用可能であることに注意すること。

重要事項！　書き込み操作の前に、あなたのＰＣのカードまたはデジタルカムを読み込み不可にするまで、決してブートまたはＦＡＴセクタに上書きしないよう注意すること。
Ｗｉｎｈｅｘのようなドライブマッピングツールが大きな援助となりうる。

Library　Routines

```
Cf_Init
Cf_Detect
Cf_Total_Size
Cf_Enable
Cf_Disable
Cf_Read_Init
Cf_Read_Byte
Cf_Write_Init
Cf_Write_Byte

Cf_Fat_Init
Cf_Fat_Assign
Cf_Fat_Reset
Cf_Fat_Read
Cf_Fat_Rewrite
Cf_Fat_Append
Cf_Fat_Delete
Cf_Fat_Write
Cf_Fat_Set_File_Date
Cf_Fat_Get_File_Date
Cf_Fat_Get_File_Size
```

　関数 Cf_Det_Reg_Adr はコンパイラの内部的な目的のためだけにある。

Cf_Init

原形	Void Cf_Int(char *ctrlport,char *dataport);
解説	CF カードとの通信のため適切にポートを初期化する。２つの異なったポート、ctrlport と dataport を指定する。
例	Cf_Init(&PORTB,&PORTD)

Cf_Detect

原形	char Cf_Detect(void);
戻り値	もし CF カードが有れば１を、そうでなければ０を返す。
解説	ctrlport に CF カードが有るか確認する。
例	// CF カードが挿入されるまで待つ do nop ; while (Cf_Detect() = = 0);

Cf_Total_Size

原形	unsigned long Cf_Total_Size(void);
戻り値	k byte のカードサイズ
解説	kbyte でコンパクトフラッシュカードのサイズを返す
必要事項	ポートを初期化する必要あり。Cf_Init を見よ。
例	Size = Cf_Total_Size( );

Cf_Enable

原形	void Cf_Enable(void);
解説	デバイスを有効にする。 もしあなたが Cf_Disable によってデバイスを無効にした場合のみ、ルーチンを呼び出す必要がある。複数デバイスで動作する場合、これらの２つのルーチンの組み合わせが開放／占有することを可能にする。この章の終わりの例を確認すること。
必要事項	ポートを初期化しなければならない。Cf_Init を見よ。
例	Cf_Enable( );

Cf_Disable

原形	void Cf_Disable(void);
解説	ルーチンはデバイスを無効にし、他のデバイスに対しデータ線を開放する。デバイスを再度有効にするには、Cf_enable を呼び出す。複数デバイスで動作する場合、これらの２つのルーチンの組み合わせがデータ線の開放／占有することを可能にする。この章の終わりの例を確認すること。
必要事項	ポートを初期化しなければならない。Cf_Init を見よ。
例	Cf_Disable( );

Cf_Read_Init

原形	void Cf_Read_Init (long address, char sectcnt);
解説	読み込むために CF カードを初期化する。パラメータ Address は、データが読み出されるセクタアドレスを指定し、sectcnt は読み出し操作に備えたセクタの番号である。
必要事項	ポートを初期化しなければならない。Cf_Init を見よ。
例	Cf_Read_Init (590、1);

Cf_Read_Byte

原形	char Cf_Read_Byte (void);
戻り値	CF から byte で返す
解説	CF から 1 byte 読み出す
必要事項	CF は読み出し操作のため初期化する必要あり。Cf_Read_Init を見よ。
例	PORTC = Cf_Read_Byte (); //１バイト読んで PORTC にそれを表示する。

Cf_Write_Init

原形	void Cf_Wite_Init (long address, char seccnt);
解説	書き込むため CF カードを初期化する。パラメータ address はデータが書き込まれるセクタアドレスを指定し、seccnt は書き込み操作のために準備されたセクタの全番号である。
必要事項	ポートが初期化されねばならない。Cf_Init を見よ。
例	Cf_Write_Init (590,1);

Cf_Write_Byte

原形	void Cf_Write_Byte (char data);
解説	CF へ1 byte（データ）を書く。全５１２byte がバッファへ転送される。
必要事項	CF は書き込み操作のため初期化する必要あり。Cf_Write_Init を見よ。
例	Cf_Write_Byte (100);

Cf_Fat_Init

原形	Void Cf_Fat_Init (unsigned short *control_port,unsigned short wr,rd,a2,a1,a0,ry,ce,cd, unsigned short *data_port);
戻り値	もし初期化が成功すれば0を、もしブートセクタが見つからねば1を、またカードが検出されなければ255を返す。
解説	CF カードでの FAT 操作のため適切にポートを初期化する。２つの異なったポート、ctrl と dataport を指定する。wr,rd,a2,a1,a0,ry,ce 及び cd はコントロールポートの端子番号である。
必要事項	なし。
例	Cf_Fat_Init (PORTD,6,5,2,1,0,7,3,4,PORTC);

Cf_Fat_Assign

原形	unsigned short　Cf_Fat_Assign(char *filename,char create_file);
戻り値	“1”はファイルが現存するということである。(一方、ファイルが現存しない場合、新しいファイルが作られる)、さもなくば”0”、もしファイルが現存しない場合、新しいファイルが作られる。
解説	FAT 操作のためファイルを割り当てる。もしファイルが無い場合、関数は与えられたファイル名で新しいファイルを生成する。filename パラメータはファイルの名前である。(ファイル名はフォーマット 8.3 により大文字でなければならない。) create_file は新しいファイルを生成するためのパラメータである。もし create_file が（ ）と異なる場合、新しいファイルが生成される。(もし与えられたファイル名のファイルが無い場合)
必要事項	CF の FAT 操作のためポートが初期化されねばならない。Cf_Fat_Init をみよ。
例	Cf_Fat_Assign ('MIKROELE.TXT',1);

Cf_Fat_Reset

原形	void Cf_Fat_Reset(unsigned long *size);
戻り値	byte でのファイルの大きさ。サイズは入力変数のアドレスに保存される。
解説	読み出しのため割り当てられたファイルをオープンする。
必要事項	CF の FAT 操作のためにポートが初期化されねばならない。Cf_Fat_Init をみよ。 ファイルは割当てられねばならない。Cf_Fat_Assign をみよ。
例	Cf_Fat_Reset (size);

Cf_Fat_Read

原形	void Cf_Fat_Read (unsigned short *bdata);
解説	ファイルからデータを読む。bdata はバッファから読まれたデータである。
必要事項	CF の FAT 操作のためポートが初期化されねばならない。Cf_Fat_Init を見よ。 ファイルは割り当てられねばならない。Cf_Fat_Assign を見よ。 ファイルは読み出すために開かねばならない。Cf_Fat_Reset を見よ。
例	Cf_Fat_Read (character);

Cf_Fat_Rewrite

原形	void Cf_Fat_ Rewrite ( );
戻り値	なし
解説	割り当てられたファイルを再書き込みする。
必要事項	CF の FAT 操作のためポートが初期化されねばならない。Cf_Fat_Init を見よ。 ファイルは割り当てられねばならない。Cf_Fat_Assign を見よ。
例	Cf_Fat_ Fat_ Rewrite;

Cf_Fat_Append

原形	void Cf_Fat_ Append ( );
戻り値	なし
解説	書き込みのためにファイルを開く。このプロシージャ（手続き）はファイル内の最後のバイトから書き込みを継続する。
必要事項	CF の FAT 操作のためポートが初期化されねばならない。Cf_Fat_Init を見よ。 ファイルは割り当てられねばならない。Cf_Fat_Assign を見よ。
例	Cf_Fat_ Append;

Cf_Fat_Delete

原形	void Cf_Fat_Delete( );
解説	CF からファイルを削除する。
必要事項	CF の FAT 操作のためポートが初期化されねばならない。Cf_Fat_Init を見よ。 ファイルは割り当てられねばならない。Cf_Fat_Assign を見よ。。
例	Cf_Fat_ Delete;

Cf_Fat_Write

原形	void Cf_Fat_ Write (char *fdata、unsigned　data_len);
戻り値	なし
解説	CF へデータを書き込む。パラメータ *fdata* は、CF へ書き込まれるデータである。*data_len* は CF へ書き込まれるバイト数である。
必要事項	CF の FAT 操作のためポートは初期化されねばならない。Cf_Fat_Init を見よ。 ファイルは割り当てられねばならない。Cf_Fat_Assign を見よ。 ファイルは書き込みのためにオープンしなければならない。Cf_Fat_Rewrite または Cf_Fat_Append を見よ。
例	Cf_Fat_ Write(file_contents,42);　//割り当てられたファイルへデータを書く

Cf_Fat_Set_File_Data

原形	void Cf_Fat_ Set_File_Data (unsigned int year, unsigned short month, unsigned short day, unsigned short hours, unsigned short mins, unsigned short seconds);
戻り値	なし
解説	ファイルの時間属性を設定する。あなたはファイルに年、月、日、時、分、秒を設定できる。
必要事項	CF の FAT 操作のためポートは初期化されねばならない。Cf_Fat_Init を見よ。 ファイルは割り当てられねばならない。Cf_Fat_Assign を見よ。 ファイルは書き込みのためにオープンしなければならない。Cf_Fat_Rewrite または Cf_Fat_Append を見よ。
例	Cf_Fat_ Write(file_contents,42);

Cf_Fat_Get_File_Date

原形	void CF_Fat_Get_File_Date (unsigned int *year, unsignd short *month, unsigned short *day, unsigned short *hours , unsigned short *mins);
解説	ファイルの時間属性を読み出す。あなたは年、月、日、時、分を読み出すことができる。
必要事項	ポートは CF の FAT 操作のため初期化しなければならない。Cf_Fat_Init を見よ。 ファイルは割り当てられねばならない。Cf_Fat_Assign を見よ。。
例	Cf_Fat_Get_File_Date(year, month, day, hours,mins);

Cf_Fat_Get_File_Size

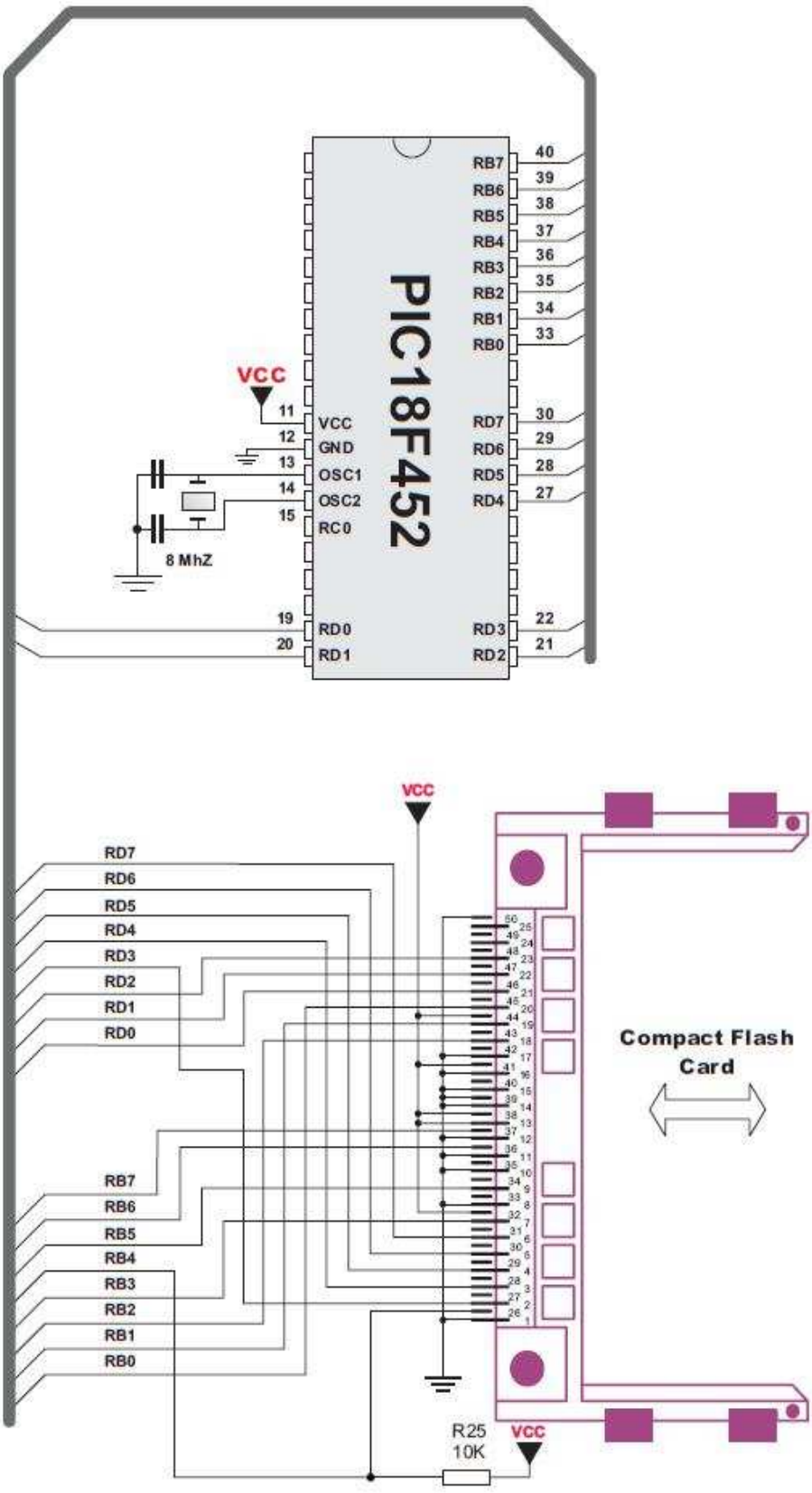
原形	Unsigned long Cf_Fat_Get_File_Size ( );
戻り値	Byte　でのファイルサイズ
解説	この関数はバイトでサイズを返す。
必要事項	CF の FAT 操作のためポートは初期化されねばならない。Cf_Fat_Init を見よ。 ファイルは割り当てられねばならない。Cf_Fat_Assign を見よ。
例	Cf_Fat_Get_File_Size ;

Library Examples

次の例はセクタno.590に512byteを書くものであり、それからデータを読み出して、視覚的に確認するためPORTCへ出力する。

```
unsigned i;

void main() {
  TRISC = 0;                    // PORTC is output
  Cf_Init(&PORTB, &PORTD);       // Initialize ports

  do nop;
  while (!Cf_Detect());         // Wait until CF card is inserted

  Delay_ms(500);
  Cf_Write_Init(590, 1);        // Initialize write at sector address 590

  // Write 512 bytes to sector (590)
  for (i = 0; i < 512; i++) Cf_Write_Byte(i + 11);

  PORTC = 0xFF;
  Delay_ms(1000);
  Cf_Read_Init(590, 1);         // Initialize read at sector address 590

  // Read 512 bytes from sector (590)
  for (i = 0; i < 512; i++) {
    PORTC = Cf_Read_Byte();     // Read byte and display on PORTC
    Delay_ms(1000);
  }
}
```

ハードウェア接続図


PIC18F452
VCC
GND
OSC1
OSC2
RC0
8 MhZ
RB7
RB6
RB5
RB4
RB3
RB2
RB1
RB0
RD7
RD6
RD5
RD4
RD3
RD2
RD1
RD0
Compact Flash
Card
R25
10K

Compact Flash FAT Library v2.xx

これは以前のバージョン（ｖ２．１）のコンパクトフラッシュ FAT ライブラリである。
このライブラリは古いＣＦライブラリでプロジェクトを開発したユーザーのために含まれている。

注意）
このバージョンのコンパクトフラッシュＦＡＴライブラリは悪く言われる。
もはやこのバージョンのライブラリによる開発は無いからだろう。
あなたのプロジェクトでは新しいバージョンのコンパクトフラッシュライブラリを使用してください。

重要！）
ファイル　アクセス　ルーチンはファイルを書くことができる。
ファイル名は正確に８文字長で、且つ大文字で書かれねばならない。
ユーザーは各ファイルに対し異なった名前を確保しなければならない。なぜなら、CF ルーチンは一致の可能性のあるものには検査しないことになっているからである。

重要！）
書き込み操作の前に、ＰＣのカードやデジタルカムを読めなくなるのでブートまたはＦＡＴセクタに上書きしないよう注意すること。Ｗｉｎｈｅｘのようなドライブマッピングツールが大きな手助けになる。

Library Routines

```
Cf_Find_File
Cf_File_Write_Init
Cf_File_Write_Byte
Cf_Read_Sector
Cf_Write_Sector
Cf_Set_File_Date
Cf_File_Write_Complete
```

Cf_Find_File

原形	void Cf_Find_File (unsignedshort　find_first, char *file_name);
解説	ルーチンは CF カード上のファイルを捜す。パラメータ find_first は非ゼロまたはゼロを取りうる。もし非ゼロならば、物理的な書き込み順でカード上の最初のファイルを捜す。一方現在の位置から次のファイルまで、物理的な命令を繰り返し、ルーチンは“前方移動”する。もしファイルが見つかれば、ルーチンはファイル名の文字列名前と拡張子を書き込む。もしファイルが無ければ文字列はゼロで満たされる。
必要事項	ポートが初期化されねばならない。
例	Cf_Find_File (1, file); If (file[0]) { . . . // もし最初のファイルが見つかれば、それを取り扱う

Cf_File_Write_Init

原形	void Cf_File_Write_Init (void)
解説	ファイル書き込み操作（FAT16 のみ）のため、CF カードを初期化する。
必要事項	ポートが初期化されねばならない。Cf_Init を見よ。
例	Cf_File_Write_Init ();

Cf_File_Write_Byte

原形	void Cf_File_Write_Byte(char　data)
解説	ファイルに１バイト（データ）を追加する。あなたはパラメータとして、例えばゼロに対しては４８といった ASCII 値を提供できる。
必要事項	CF はファイル書き込み操作のため初期化されねばならない。Cf_File_Write_Init を見よ。
例	For(i=0 ; i < 50000 ; i++)　Cf_File_Write_Byte(48);

Cf_Read_Sector

原形	void Cf_ Read_Sector (**int**　sector_number, **unsigned short**　*buffer);
解説	バッファに１セクタ（sector_number）を読み込む。Cf_Init を見よ。
必要事項	CF はファイル書き込み操作のため初期化されねばならない。Cf_File_Write_Init を見よ。
例	Cf_Read_Sector (22, data)

Cf_Write_Sector

原形	void Cf_ Write_Sector (**int** sector_number, **unsigned short** *buffer);
解説	sector_number の CF セクタにバッファから value を書き込む。
必要事項	CF はファイル書き込み操作のため初期化されねばならない。Cf_Init を見よ。
例	Cf_Write_Sector (22,data)

Cf_Set_File_Date

原形	void Cf_Set_File_Date (int year, char month, day, hours, min, sec);
解説	システムのタイムスタンプをファイルへ書き込む。ファイルをファイナライズする前にこのルーチンを使用すること。さもないと、ファイルはランダムなタイムスタンプを追加するだろう。
必要事項	CF はファイル書き込み操作のため初期化されねばならない。Cf_File_Write_Init を見よ。
例	Cf_ Set_File_Date (2005,4,1,18,7,0)

Cf_File_Write_Complete

原形	void Cf_File_Write_Complete (**char** filename[8], **char** *extension);
解説	ファイルへ書き込むためにファイナライズをする。全てのデータがファイルに書き込まれたら、ファイルを閉じ読み込み可能にするためにこの関数を使用する。パラメータ filename は大文字の８文字長でなければならない。
必要事項	CF はファイル書き込み操作のため初期化されねばならない。Cf_File_Write_Init を見よ。
例	Cf_ File_Write_Complete (“MY_FILE1”,”txt”)

EEPROM　Library（EEPROM ライブラリ）

EEPROM データメモリは多くのＰＩＣｍｉｃｒｏｓに有効である。
mikroC はＥＥＰＲＯＭで最適に作業するためのライブラリを含む。

Library　Routines

```
Eeprom_Read
Eeprom_Write
```

Eeprom Read

原形	**Unsigned　short**　Eeprom_Read(**unsigned short**　address);
戻り値	指定されたアドレスからバイトを戻す。
解説	指定されたアドレスからデータを読み込む。パラメータ address は整数型であり、EEPROM ２５６バイト以上を持つ MCU をサポートすることを意味する。
必要事項	EEPROM モジュールを必要とする。Eeprom_Read と Eeprom_write のルーチンを首尾よく使用するためには最小 20ms の遅延時間を確保すること。 さもないと、PIC が誤った値を書き込むだろうし、Eeprom_Read は未定義の結果を戻すことになる。
例	Char take; . . . take = Eeprom Read(0x3F);

Eeprom Write

原形	void Eeprom_Write(**unsigned int** address, **unsigned short** data);
解説	指定されたアドレスへデータを書き込む。パラメータ address は整数型であり、EEPROM２５６バイト以上を持つMCU をサポートすることを意味する。 Eeprom_Wite ルーチン実行の間は、全割り込みが禁止となることに注意する。（INTCON の GIE ビットがクリアされるだろう。）終了時、ルーチンはこのビットを前の状態に戻すだろう。
必要事項	EEPROM モジュールを必要とする。 Eeprom_ write と Eeprom_ Read のルーチンを首尾よく使用するには最小20ms の遅延時間を確保すること。 さもないと、PIC が誤った値を書き込むだろうし、Eeprom_Read は未定義の結果を戻すことになる。
例	Eeprom_Write(0x32);

Library Example

```
unsigned short i = 0, j = 0;

void main() {
  PORTB = 0;
  TRISB = 0;

  j = 4;
  for (i = 0; i < 20u; i++)
    Eeprom_Write(i, j++);

  for (i = 0; i < 20u; i++) {
    PORTB = Eeprom_Read(i);
    Delay_ms(500);
  }
} //~!
```

Ethernet　Library（イーサネットライブラリ）

　このライブラリは基礎をなすハードウェア（RTL8019AS）の操作を簡略化するために設計されている。
しかしながら、イーサネットとイーサネットを基本としたプロトコル（ＡＲＰ、ＩＰ、ＴＣＰ／ＩＰ、ＵＤＰ／ＩＰ、ＩＣＭＰ／ＩＰ）に関するのあるレベルの知識がユーザーから期待されている。
イーサネットは高速かつ多目的なプロトコルであるが、それは単純なものではない。
一旦、あなたがそれに慣れ親しんだら、とはいえ、ＲＳ２３２／４８５またはＣＡＮでなし得る以上に、より広範な対象ユーザーに対し、あなたの得意なＰＩＣを利用可能にするだろう。

Library Routines

```
Eth_Init
Eth_Set_Ip_Address
Eth_Inport
Eth_Scan_For_Event
Eth_Get_Ip_Hdr_Len
Eth_Load_Ip_Packet
Eth_Get_Hdr_Chksum
Eth_Get_Source_Ip_Address
Eth_Get_Dest_Ip_Address
Eth_Arp_Response
Eth_Get_Icmp_Info
Eth_Ping_Response
Eth_Get_Udp_Source_Port
Eth_Get_Udp_Dest_Port
Eth_Get_Udp_Port
Eth_Set_Udp_Port
Eth_Send_Udp
Eth_Load_Tcp_Header
Eth_Get_Tcp_Hdr_Offset
Eth_Get_Tcp_Flags
Eth_Set_Tcp_Data
Eth_Tcp_Response
```

Eth_Init

原形	void　Eth_Init (**char** *addrP, **char** *dataP, **char** *ctrlP, **char** *pinreset, **char** pinIOW **char** *pinIOR);
解説	イーサネットカードとライブラリの初期化を実行する。これは次の内容を含む。 －コントロール及びデータポートの設定をする －イーサネットカードの初期化（ネットワークインターフェースカードまたはNIC とも呼ばれる） －NIC のハードウェア（MAC）アドレスの検索とローカルメモリ －NIC を LISTEN モードにする。 パラメータ addrP は、アドレッシング線を操作するアドレスポートを指すポインタである。パラメータ dataP は、データポートを指すポインタである。パラメータ ctrlP はコントロールポートである。パラメータ pinReset はイーサネットカードチップ（コントロールポート上の）に対するリセット／有効端子である。パラメータ pinIOW は I/O 書き込み要求コントロール端子である。パラメータ pinIOR は I/O 読み込み要求コントロール端子である。
必要事項	全ライブラリに対し指定されるのと同様。(このページの先頭を見てください。)
例	Eth_Init (&PORTB, &PORTD, &PORTE, 2,1,0);

Eth_Set_Ip_Address

原形	Eth_Set_Ip_Address (**char** ip1, **char** ip2, **char** ip3, **char** ip4);
解説	接続し初期化したイーサネットネットワークカードの IP アドレスの設定をする。引数は IP ｖ４フォーマットの IP アドレス番号である。（例　127.0.0.1）
必要事項	この関数は NIC 初期化の後、直ちに呼ばれねばならない。(Eth_Init を見よ) あなたは自身の IP アドレスをコード内でいつでもどこでも変更可能である。
例	//　IP アドレス 192.168.20.25 を設定する Eth_Set_Ip_Address (192u, 168u, 20u, 25u);

Eth_Set_Inport

原形	**unsigned short** Eth_Inport (**unsigned short** address);
戻り値	指定されたアドレスからの１バイト。
解説	イーサネットカードチップの指定アドレスから１ｂｙｔｅを取り出す。
必要事項	カード（ＮＩＣ）は正しく初期化されねばならない。Eth_Init をみよ。
例	udp_length |= Eth_Inport(NIC_DATA);

Eth_Scan_For_Event

原形	**unsigned** Eth_Scan_For_Event (**unsigned short** *next_ptr);
戻り値	受信したイーサネットパケットの型。２つの型が区別される。ARP（MAC−IP アドレスデータ要求）と IP（インターネットプロトコル）
解説	発信者の MAC（ハードウェア）アドレスと受信したパケットの型を取り込む。関数の引数は RTL8019 のバッファ内の次のデータパケットを指す（内部）ポインタであり、エンドユーザーにとっては特別重要ということはない。
必要事項	カード（ＮＩＣ）は正しく初期化されねばならない。Eth_Init をみよ。また関数は正しいシーケンス、言い換えれば、正しいカードの初期化及び IP アドレス／UDP ポートの初期化直後に呼び出されねばならない。
例	

```
Eth_Init(&PORTB, &PORTD, &PORTE, 2, 1, 0);
Eth_Set_Ip_Address(192u, 168u, 20u, 25u);
Eth_Set_Udp_Port(10001);
do { // Main block of every Ethernet example
  event_type = Eth_Scan_For_Event(&next_ptr);
  if (event_type) {
    switch (event_type) {case ARP: Arp_Event(); break;
                         case IP : Ip_Event();}
    Eth_Outport(CR, 0x22);
    Eth_Outport(BNDRY, next_ptr);
  }
} while (1);
```

Eth_Get_Ip_Hdr_Len

原形	**unsigned short** Eth_Get_Ip_Hdr_Len (**void**);
戻り値	受信した IP パケットのヘッダー長
解説	受信した IP パケットのヘッダー長を返す。IP プロトコル（TCP、UDP、ICMP）を基本にした他のデータが解析される前に、サブプロトコルデータが、受信した IP パケットから正しくロードされねばならない。
必要事項	カード（ＮＩＣ）は正しく初期化されねばならない。Eth_Init をみよ。関数は正しいシーケンス、言い換えれば、受信したパケットが IP パケットであると判定した直後に、呼ばれねばならない。
例	opt_len = Eth_Get_Ip_Hdr_Len( ) - 20

Eth_Load_Ip_Packet

原形	**void** Eth_Load_Ip_Packet (**void**);
解説	PIC のイーサネット変数に様々な IP パケットデータをロードする。
必要事項	カード（ＮＩＣ）は正しく初期化されねばならない。Eth_Init をみよ。正しい呼び出しシーケンスが遵守されねばならない。(提供されたイーサネットの例の中の Ip_Event 関数をみよ)
例	Eth_Load_Ip_Packet ();

Eth_Get_Hdr_Chksum

原形	**void**　Eth_ Get_Hdr_Chksum (**void**);
解説	受信した IP パケットのヘッダーチェックサムをロードして返す。
必要事項	カード（ＮＩＣ）は正しく初期化されねばならない。Eth_Init をみよ。正しい呼び出しシーケンスが遵守されねばならない。(提供されたイーサネットの例の中の Ip_Event 関数をみよ)
例	Eth_Load_Ip_ Chksum ( );

Eth_Get_Souce_Ip_Address

原形	**void**　Eth_Get_Souce_Ip_Address (**void**);
解説	受信した IP パケットの送信者の IP アドレスをロードして返す。
必要事項	カード（ＮＩＣ）は正しく初期化されねばならない。Eth_Init をみよ。正しい呼び出しシーケンスが遵守されねばならない。(提供されたイーサネットの例の中の Ip_Event 関数をみよ)
例	Eth_Get_Souce_Ip_Address ( );

Eth_Get_Dest_Ip_Address

原形	**void**　Eth_Get_ Dest _Ip_Address (**void**);
解説	パケットが指定される受信 IP パケットの IP アドレスをロードする。
必要事項	カード（ＮＩＣ）は正しく初期化されねばならない。Eth_Init をみよ。正しい呼び出しシーケンスが遵守されねばならない。(提供されたイーサネットの例の中の Ip_Event 関数をみよ)
例	Eth_Get_Dest_Ip_Address ( );

Eth_Apr_Responce

原形	**void**　Eth_Apr_Responce (**void**);
解説	自動化された ARP 応答。一旦 NIC で APR イベントが検出されれば、ユーザーはこの関数を容易に呼び出せるはずである。
必要事項	全ライブラリに対し指定されるのと同様。
例	Eth_Apr_Responce ();

Eth_Get_Icmp_Info

原形	**void**　Eth_Get_Icmp_Info (**void**);
解説	ICMP プロトコル情報（受信した ICMP パケットのヘッダーからの）をロードし、PIC のイーサネット変数へそれを保存する。
必要事項	カード（ＮＩＣ）は正しく初期化されねばならない。Eth_Init をみよ。この関数は、Eth_Ping_Response の前に、正しいシーケンス（順序）で呼び出されねばならない。
例	Eth_Get_Icmp_Info ();

Eth_ Ping_Response

原形	**void**　Eth_ Ping_Response (**void**);
解説	自動化された ICMP（Ping）応答。ユーザーは、ICMP／IP イベントに応答する場合、この関数を呼び出すべきである。
必要事項	全ライブラリに対し指定されるのと同様。
例	Eth_ Ping_Response ();

Eth_Get_Udp_Source_Port

原形	**unsigned** Eth_Get_Udp_Source_Port (**void**);
戻り値	受信したUDPパケットのソースポート（ソケット）を返す。
解説	この関数は受信したUDPパケットのソースポート（ソケット）を返す。有効なIPパケットの受信が検出され、且つその型がUDPであると決定された後、UDPパケットヘッダが解釈されねばならない。UDPソースポートはUDPヘッダの最初のデータである。
必要事項	この関数は正しいシーケンス、言い換えれば、IPパケットヘッダの解釈の直後に（UDPパケットヘッダー検索のちょうど始りで）で呼ばれねばならない。
例	udp_source_port = Eth_Get_Udp_Source_Port();

Eth_Get_Udp_Dest_Port

原形	**unsigned** Eth_Get_Udp_ Dest _Port (**void**);
戻り値	受信したUDPパケットのデスティネーションポートを返す。
解説	この関数は受信したUDPパケットのデスティネーションポートを返す。UDPパケットヘッダに含まれる第二の情報が、パケットが目標とされたデスティネーションポート（ソケット）である。
必要事項	この関数は正しいシーケンス、言い換えれば、Eth_Get_Udp_ Dest _Port 関数を呼び出した直後に呼ばれねばならない。
例	udp_ dest _port = Eth_Get_Udp_ Dest _Port();

Eth_Get_Udp_Port

原形	**unsigned short** Eth_Get_Udp_ Port (**void**);
戻り値	PIC のイーサネットカードへ設定されるた UDP ポート（ソケット）番号を返す。
解説	この関数は PIC のイーサネットカードへ設定された UDP ポート（ソケット）番号を返す。UDP ポートがセッション（Eth_Set_Udp_Port）の始めに設定された後、その番号が、受信した UDP パケットが、われわれが使用しているポートをターゲットにされているかどうか、その後で検査される。
必要事項	ネットワークカードは正しく初期化されねばならない。（Eth_Init を見よ。）そして UDP ポートが正しく設定する。(Eth_Set_Udp_Port を見よ。) このライブラリは現在、1 度に 1 UDP ポート（ソケット）のみでの作業をサポートしている。

例	`if (udp_dest_port == Eth_Get_Udp_Port()) {`
	`  ... // Respond to action`
	`}`

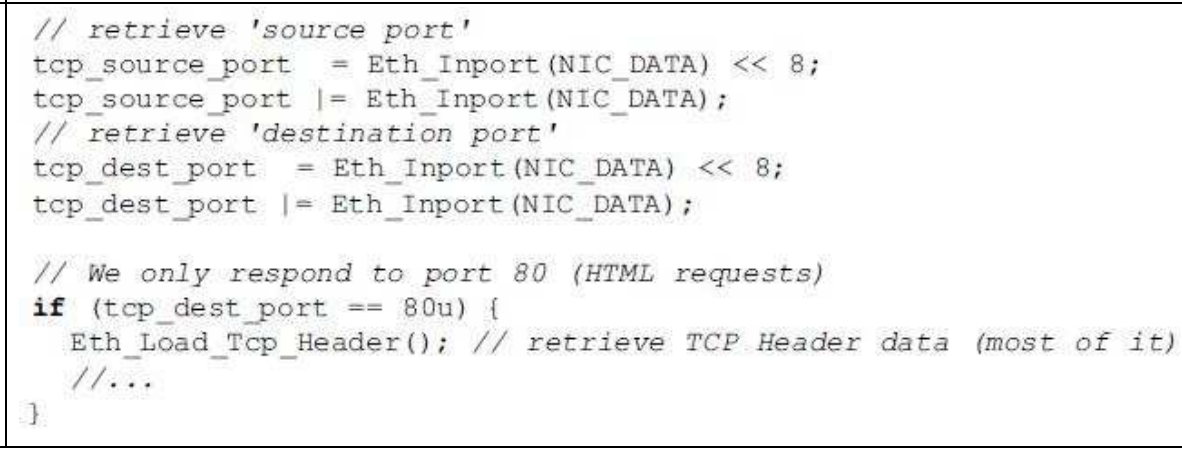
Eth_Set_Udp_Port

原形	**void** Eth_Set_Udp_ Port (**unsigned** udp_port);
解説	ユーザーリクエストを扱うであろうデフォルト UDP ポートを設定する。ユーザーは、UDP パケットを受信することでどのポートがこのパケットを送ったか、また、扱うのか拒否するのかを決定可能である。
必要事項	全ライブラリに対し指定されるのと同様。
例	Eth_Set_Udp_ Port(10001)

Eth_Send_Udp

原形	**void**　Eth_Send_Udp (**char** *msg);
解説	用意しておいた 16byte（文字）までの UDP メッセージ（msg）を送信する。 ICMP と TCP とは違い、UDP パケットは一般的にクライアント要求への応答として生成されない。UDP はメッセージの配達に対する保証を与えてはいないし、一旦ネットワーク上に送出されると、送信者は UDP メッセージ上で状態を保持することは無い。これは、誰かの要求に対し応答する代わりに、UDP パケットが単純に送信される、という理由による。
必要事項	全てのライブラリに対する指定のし方と同様。送信されるべきメッセージはＮＵＬＬで終わるストリングとして初期化されねばならない。端に"0"を含んだメッセージ長は 16 文字を越えてはならない。
例	Eth_Send_Udp (udp_tx_message);

Eth_Load_Tcp_Header

原形	**void**　Eth_Load_Tcp_Header (**void**);
解説	ＰＩＣのイーサネット変数にさまざまなＴＣＰヘッダーデータをロードする。
必　要事項	この関数は正しいシーケンス、即ち、ＴＣＰメッセージのソースとデスティネーションポート（ソケット）を検索した直後に呼ばれねばならない。
例	

```
// retrieve 'source port'
tcp_source_port  = Eth_Inport(NIC_DATA) << 8;
tcp_source_port |= Eth_Inport(NIC_DATA);
// retrieve 'destination port'
tcp_dest_port  = Eth_Inport(NIC_DATA) << 8;
tcp_dest_port |= Eth_Inport(NIC_DATA);

// We only respond to port 80 (HTML requests)
if (tcp_dest_port == 80u) {
  Eth_Load_Tcp_Header(); // retrieve TCP Header data (most of it)
  //...
}
```

Eth_Get_Tcp_Hdr_Offset

原形	**unsigned short** Eth_Get_Tcp_Hdr_Offset (**void**);
戻り値	byte で TCP パケットヘッダ長（又はオフセット値）を返す。
解説	この関数は byte で TCP パケットヘッダ長（又はオフセット値）を返す。 有効な TCP パケットを受信するまで、正しく応答するため、そのヘッダは解析されるはずである。（例えば、他の要求への応答、メッセージ内の様々なパケットの結合など） その中に含まれた情報を抜き出すことを可能にするため、ヘッダー長を知ることは重要である。
必要事項	この関数のために使われる専用の変数を初期化した後、この関数は Eth_Load_Tcp_Header の後に呼び出されねばならない。

例	// オフセット（TCP ヘッダー長）を計算する
	tcp_options = Eth_Get_Tcp_Hdr_Offset( ) - 20

Eth_Get_Tcp_Flags

原形	**unsigned short** Eth_Get_Tcp_Flags (**void**);
戻り値	受信した TCP パケットのヘッダからフラグデータを返す。
解説	この関数は受信した TCP パケットのヘッダからフラグデータを返す。 TCP フラグはさまざまな情報を示す。例えば、SYN（同期要求）、ACK（受信肯定応答）や同様の情報。例えば、正しい HTTP 通信が確立されたということがこれらのフラグの情報である。
必要事項	この関数のために使われる専用の変数を初期化した後、この関数は Eth_Load_Tcp_Header の後に呼び出されねばならない。
例	Flags = Eth_Get_Tcp_Flags ( );

Eth_Set_Tcp_Data

原形	**void** Eth_Set_Tcp_Data (**const unsigned short** *data);
解説	HTTP の要求に対し、送信するためのデータを準備する。 エラーを発生するであろう FTP のような他の TCP を基本としたプロトコルを送信するので、このライブラリは HTTP の要求のみを取り扱うことが可能である。 あなたのＨＴＴＰ要求を容易に扱うために、ＴＣＰ／ＩＰは８ビット MCU で設計されていなかったことを念頭に置き注意すること。
必要事項	全ライブラリに対し指定されるのと同様。

例	`// Let's prepare a simple HTML page in our string:`
	`const char httpPage1[] =`
	`"HTTP/1.0 200 OK\nContent-type: text/html\n"`
	`"<html>\n" "<body>\n"`
	`"<h1>Hello world!</h1>\n"`
	`"</body>\n" "</html>";`
	`//...`
	`Eth_Set_Tcp_Data(httpPage1);`
	`//...`

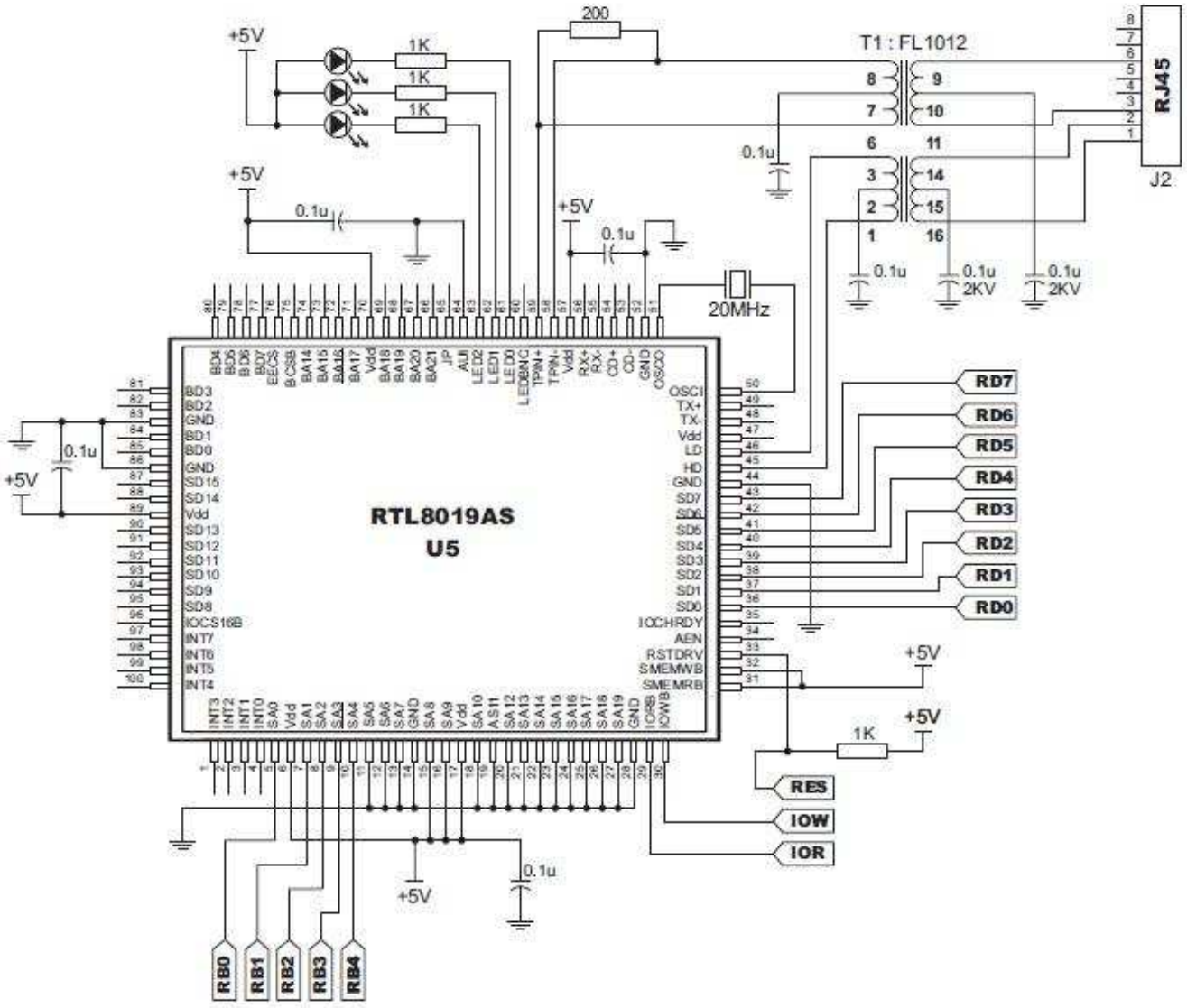
Eth_Tcp_Responce

原形	**void** Eth_ Tcp_Responce (**void**);
解説	TCP／IP イベントに対し、ユーザーの応答を実行する。 ユーザーが、受信した要求（HTTP, HTTPD, FTP など）により、送信されるべきデータを指定する。これは関数 Eth_Set_Tcp_Data により実行される。
必要事項	ハードウェア要求は全てのライブラリに対して指定するのと同様である。この関数を使用するより先に、ユーザーは TCP を通して送信されるべきデータを準備しなければならない。Eth_Set_Tcp_Data を見よ。
例	Eth_Tcp_Responce ();

Library Example

提供された *Examples* フォルダ内のイーサネットの例を調べよ。

HW Connection



SPI　Ethernet　Library（ＳＰＩイーサネットライブラリ）

ENC28J60 は、産業用標準シリアル周辺インターフェース（SPI）を持ったスタンドアローン　イーサネットコントローラである。それは、ＳＰＩを装備したいかなるコントローラにも対応したイーサネット　ネットワーク　インターフェースとして役立つよう設計されている。

ＥＮＣ28J60 はＩＥＥＥ802.3 仕様全てを満たす。それは入ってくるパケットを制限するために、スキームをフィルタリングして多数のパケットを取り込む。それはまた急速なデータ処理能力とＩＰチェックサム演算を援助するハードウェアのために、内部DMAモジュールを提供する。ホストコントローラとの通信は、２つの割り込み端子とＳＰＩを経て、１０Ｍｂ／ｓまでのデータレートで実行される。２つの提供された端子はリンクとネットワーク有効表示 LED のために使われる。

このライブラリは基礎となるハードウェア（ＥＮＣ28Ｊ60）の取扱いを容易にするために設計されている。
それは集積されたＳＰＩを持つ全ての PIC と 4Kbyte 以上の ROM メモリで動作する。
38~40MH z クロックが 8～10MHzSPI クロックを得るために推奨されるが、その一方、PIC は、SPI ハードウェア内の ENC シリコンバグにより、ENC クロック出力によりクロック供給されるべきである。
もしあなたがより低いPICクロックスピードに挑もうとするなら、いくつかの要求を取り下げるか失わねばならない。
このライブラリは、PIC16F877@10MHz、PIC18F452@40MHz　で評価されている。

注意）進んだユーザーのため、SPI イーサネットライブラリで実行される全ての関数の詳細な記述のついた P16 と P18 のフォルダ内にヘッダー（ファイル）（“enc28j60_libprivate.h”）がある。

注意）SPI イーサネットを初期化する前に、SPI_Init()が呼び出されねばならない。

Librasry　Routine

```
SPI_Ethernet_Init
SPI_Ethernet_doPacket
SPI_Ethernet_putByte
SPI_Ethernet_getByte
SPI_Ethernet_UserTCP
SPI_Ethernet_UserUDP
```

SPI_Ethernet_Init

原形	`void SPI_Ethernet_Init(unsigned char *resetPort, unsigned char resetBit, unsigned char *CSportPtr, unsigned char CSbit, unsigned char *mac, unsigned char *ip, unsigned char fullDuplex);`
戻り値	なし。
解説	SPI と ENC コントローラを初期化する。メモリ不足の場合に備え、この関数はリンカーを助けるため２つの部分に分けられている。 resetPort－ポートのリセット端子を指すポインタ resetBit－resetPort のビット番号をリセット CSPprt－ポートの CS ピンを指すポインタ Csbit－CS ポートの CS ビット番号 mac－MAC アドレス６文字分の配列を指すポインタ ip－IP アドレス４文字分の配列を指すポインタ fullduplex－半二重のための SPI_Ethernet_HALFDUPLEX または 全二重のための SPI_Ethernet_FULLDUPLEX
必要事項	SPI イーサネットを初期化する前に、SPI_Init()が呼び出されねばならない。
例	`SPI_Ethernet_Init(&PORTC, 0, &PORTC, 1, myMacAddr, myIpAddr, SPI_Ethernet_FULLDUPLEX);`

SPI Ethernet_doPacket

原形	void SPI_Ethernet_doPacket();
戻り値	なし。
解説	可能ならば、１つの入力パケットを処理する。この関数はパブリックである。
必要事項	使用する度に、この関数が呼び出されるその前に、SPI_ Ethernet_init( ) が呼び出されねばならない。
例	SPI Ethernet doPacket();

P214

SPI_Ethernet_putByte

原形	**void**　SPI_Ethernet_putByte (**unsigned char**　v);
戻り値	なし。
解説	**v**ー保存するための値 現在の EWRPT　ENC ローケーションへ１バイトを保存する。この関数はパブリックである。
必要事項	この関数を呼び出す前に、SPI_ Ethernet_Init が呼び出されねばならない。
例	SPI_Ethernet_putByte (0xa0);

SPI_Ethernet_getByte

原形	**unsigned char**　SPI_Ethernet_ getByte ( );
戻り値	@addr.の値
解説	現在の ERDPT　ENC ローケーションから１バイトを得る。この関数はパブリックである。
必要事項	この関数を呼び出す前に、SPI_ Ethernet_Init が呼び出されねばならない。
例	b = SPI_Ethernet_ getByte ( );

P215

SPI_Ethernet_UserTCP

原形	`unsigned int SPI_Ethernet_UserTCP(unsigned char *remoteHost, unsigned int remotePort, unsigned int localPort, unsigned int reqLength);`
戻り値	HTTP 応答のバイト長、または送信無しなら 0 を返す。
解説	この関数はライブラリにより呼び出される。 ユーザーは、首尾よく SPI_Ethernet_getByte( )へ呼出しすることで HTTP 要求にアクセスする。ユーザーは首尾よく SPI_Ethernet_putByte( )へ呼び出しすることで送信バッファにデータを置く。関数は HTTP 応答のバイト長か、送信するものが無い場合は 0 を返す。もしあなたが HTTP 要求にこたえる必要が無いならば、単一ステートメントとして、戻り値(0)でこの関数を定義すればよい。
必要事項	この関数を呼び出す前に、SPI_ Ethernet_Init が呼び出されねばならない。
例	(空)

SPI_Ethernet_UserUDP

原形	`unsigned int SPI_Ethernet_UserUDP(unsigned char *remoteHost, unsigned int remotePort, unsigned int destPort, unsigned int reqLength);`
戻り値	UDP 応答へのバイト長、または、送信無しならば 0 を返す。
解説	この関数はライブラリにより呼び出される。 ユーザーは、首尾よく SPI_Ethernet_getByte()へ呼出しすることで UDP 要求にアクセスする。ユーザーは首尾よく SPI_Ethernet_putByte( )へ呼び出しすることで、送信バッファにデータを置く。関数は UDP 応答のバイト長か、送信するものが無い場合は 0 を返す。もしあなたが UDP 要求にこたえる必要が無いならば、単一ステートメントとして、戻り(0)でこの関数を定義すればよい。
必要事項	SPI_ Ethernet_Init が、この関数を呼び出す前に、呼び出されねばならない。
例	(空)

Library Example

次の例はSPI イーサネットライブラリの簡単なデモンストレーションである。PIC はIP アドレス１９２．１６８．２０．６０を割付けし、もしローカルエリアネットワークへ接続されれば、PING へ応答するであろう

```
#define SPI_Ethernet_HALFDUPLEX      0
#define SPI_Ethernet_FULLDUPLEX      1

/************************************************************************
 * ROM constant strings
 */
const unsigned char httpHeader[] = "HTTP/1.1 200 OK\nContent-type: " ; // HTTP header
const unsigned char httpMimeTypeHTML[] = "text/html\n\n" ;           // HTML MIME type
const unsigned char httpMimeTypeScript[] = "text/plain\n\n" ;       // TEXT MIME type
unsigned char httpMethod[] = "GET /";
/*
 * web page, splited into 2 parts :
 * when coming short of ROM, fragmented data is handled more efficiently by linker
 *
 * this HTML page calls the boards to get its status, and builds itself with
javascript
 */
const char    *indexPage = "<HTML><HEAD></HEAD><BODY>\
<h1>PIC + ENC28J60 Mini Web Server</h1>\
<a href=/>Reload</a>\
<script src=/s></script>\
<table><tr><td valign=top><table border=1 style=\"font-size:20px ;font-family: termi-
nal ;\">\
<tr><th colspan=2>ADC</th></tr>\
<tr><td>AN2</td><td><script>document.write(AN2)</script></td></tr>\
<tr><td>AN3</td><td><script>document.write(AN3)</script></td></tr>\
</table></td><td><table border=1 style=\"font-size:20px ;font-family: terminal ;\">\
<tr><th colspan=2>PORTB</th></tr>\
<script>\
var str,i;\
str=\"\";\
for(i=0;i<8;i++)\
{ str+=\"<tr><td bgcolor=pink>BUTTON #\"+i+\"</td>\";\
if(PORTB&(1<<i)){ str+=\"<td bgcolor=red>ON\";}\
else { str+=\"<td bgcolor=#cccccc>OFF\";}\
str+=\"</td></tr>\";}\
document.write(str) ;\
```

```
</script>\
" ;

const char     *indexPage2 =  "</table></td><td>\
<table border=1 style=\"font-size:20px ;font-family: terminal ;\">\
<tr><th colspan=3>PORTD</th></tr>\
<script>\
var str,i;\
str=\"\";\
for(i=0;i<8;i++)\
{str+=\"<tr><td bgcolor=yellow>LED #\"+i+\"</td>\";\
if(PORTD&(1<<i)){str+=\"<td bgcolor=red>ON\";}\
else {str+=\"<td bgcolor=#cccccc>OFF\";}\
str+=\"</td><td><a href=/t\"+i+\">Toggle</a></td></tr>\";}\
document.write(str) ;\
</script>\
</table></td></tr></table>\
This is HTTP request #<script>document.write(REQ)</script></BODY></HTML>\
" ;
//                                      str+=\"</td><td><a
href=/t\"+i+\">Toggle</a></td></tr>\";}\

/************************************
 * RAM variables
 */
unsigned char    myMacAddr[ 6]  = { 0x00, 0x14, 0xA5, 0x76, 0x19, 0x3f}  ;//my MAC address
unsigned char    myIpAddr[ 4]  = { 192, 168, 20, 60}  ;                    // my IP address
unsigned char    getRequest[ 15]  ;                                      // HTTP request buffer
unsigned char    dyna[ 31]  ;                                    // buffer for dynamic response
unsigned long    httpCounter = 0 ;                              // counter of HTTP requests

/*******************************************
 * functions
 */

/*
 * put the constant string pointed to by s to the ENC transmit buffer
 */
```

```
unsigned int    putConstString(const char *s)
        {
        unsigned int ctr = 0 ;

        while(*s)
                {
                SPI_Ethernet_putByte(*s++) ;
                ctr++ ;
                }
        return(ctr) ;
        }

/*
 * put the string pointed to by s to the ENC transmit buffer
 */
unsigned int    putString(char *s)
        {
        unsigned int ctr = 0 ;

        while(*s)
                {
                SPI_Ethernet_putByte(*s++) ;
                ctr++ ;
                }
        return(ctr) ;
        }

/*
 * this function is called by the library
 * the user accesses to the HTTP request by successive calls to
 * SPI_Ethernet_getByte()
 * the user puts data in the transmit buffer by successive calls to
 * SPI_Ethernet_putByte()
 * the function must return the length in bytes of the HTTP reply, or 0 if nothing
 * to transmit
 *
 * if you don't need to reply to HTTP requests,
 * just define this function with a return(0) as single statement
 *
 */
```

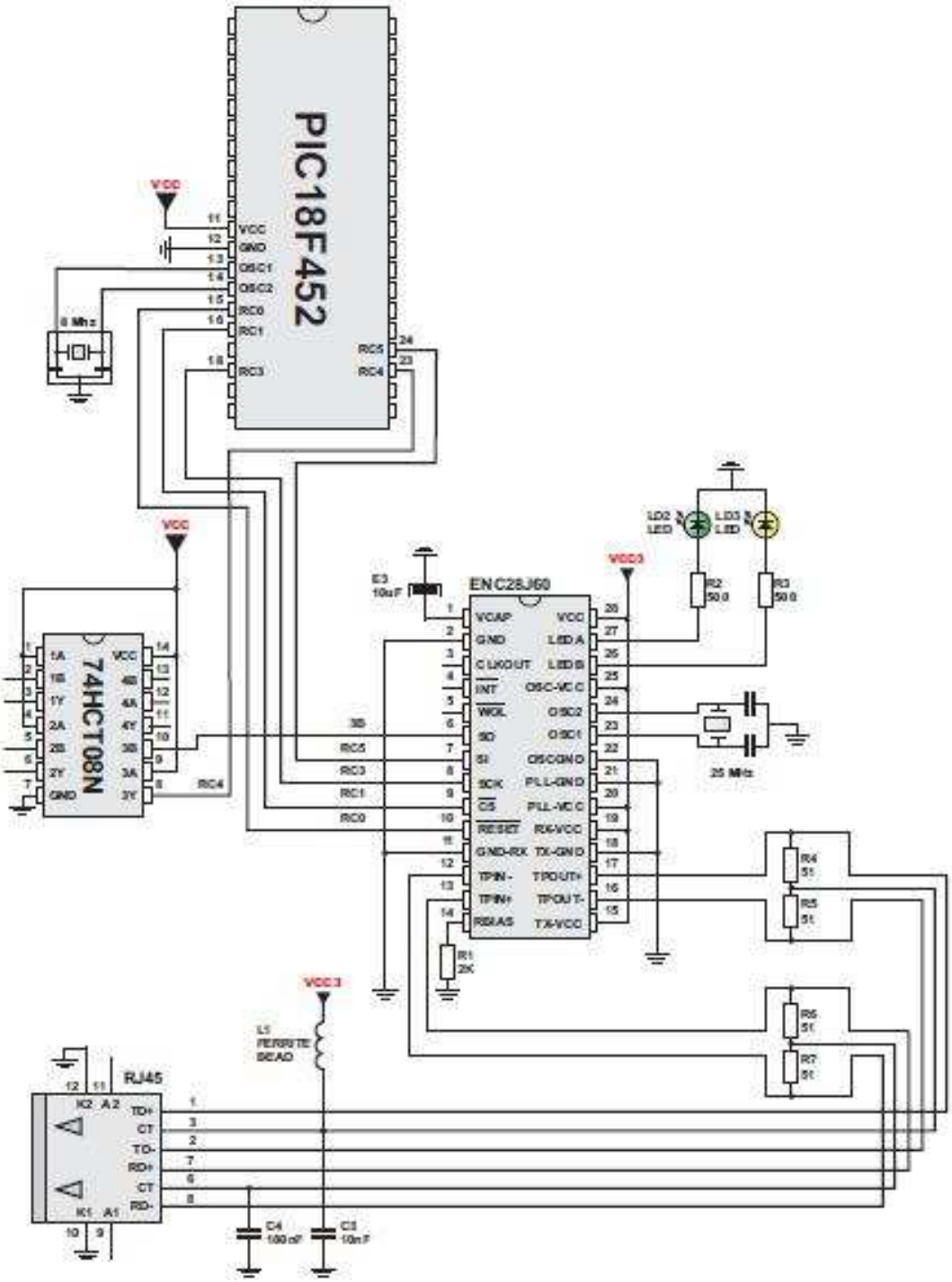
```
unsigned int    SPI_Ethernet_UserTCP(unsigned char *remoteHost, unsigned int
remotePort, unsigned int localPort, unsigned int reqLength)
        {
        unsigned int    len = 0 ;       // my reply length
        unsigned int    i ;             // general purpose integer

        if(localPort != 80)             // I listen only to web request on port 80
                {
                return(0) ;
                }

        // get 10 first bytes only of the request, the rest does not matter here
        for(i = 0 ; i < 10 ; i++)
                {
                getRequest[ i]  = SPI_Ethernet_getByte() ;
                }
        getRequest[ i]  = 0 ;

        if(memcmp(getRequest, httpMethod, 5))  // only GET method is supported here
                {
                return(0) ;
                }

        httpCounter++ ;                 // one more request done

        if(getRequest[ 5]  == 's')
// if request path name starts with s, store dynamic data in transmit buffer
                {
// the text string replied by this request can be interpreted as javascript
// statements by browsers


                len = putConstString(httpHeader) ;            // HTTP header
                len += putConstString(httpMimeTypeScript) ;   // with text MIME type

                // add AN2 value to reply
                intToStr(ADC_Read(2), dyna) ;
                len += putConstString("var AN2=") ;
                len += putString(dyna) ;
                len += putConstString(";");
```

```
                // add AN3 value to reply
                intToStr(ADC_Read(3), dyna) ;
                len += putConstString("var AN3=") ;
                len += putString(dyna) ;
                len += putConstString(";") ;

                // add PORTB value (buttons) to reply
                len += putConstString("var PORTB=") ;
                intToStr(PORTB, dyna) ;
                len += putString(dyna) ;
                len += putConstString(";") ;

                // add PORTD value (LEDs) to reply
                len += putConstString("var PORTD=") ;
                intToStr(PORTD, dyna) ;
                len += putString(dyna) ;
                len += putConstString(";") ;

                // add HTTP requests counter to reply
                intToStr(httpCounter, dyna) ;
                len += putConstString("var REQ=") ;
                len += putString(dyna) ;
                len += putConstString(";") ;
                }
        else if(getRequest[ 5] == 't')
// if request path name starts with t, toggle PORTD (LED) bit number that comes after
                {
                unsigned char   bitMask = 0 ;                   // for bit mask

                if(isdigit(getRequest[ 6]))
// if 0 <= bit number <= 9, bits 8 & 9 does not exist but does not matter
                        {
                        bitMask = getRequest[ 6] - '0' ; // convert ASCII to integer
                        bitMask = 1 << bitMask ;      // create bit mask
                        PORTD ^= bitMask ;       // toggle PORTD with xor operator
                        }
                }
```

```
        if(len == 0)                                        // what do to by default
                {
                len =  putConstString(httpHeader) ;         // HTTP header
                len += putConstString(httpMimeTypeHTML) ;   // with HTML MIME type
                len += putConstString(indexPage) ;          // HTML page first part
                len += putConstString(indexPage2) ;         // HTML page second part
                }

         return(len) ; // return to the library with the number of bytes to transmit
         }
/*
 * this function is called by the library
 * the user accesses to the UDP request by successive calls to SPI_Ethernet_getByte()
 * the user puts data in the transmit buffer by successive calls to
 * SPI_Ethernet_putByte()
 * the function must return the length in bytes of the UDP reply, or 0 if nothing to
 * transmit
 *
 * if you don't need to reply to UDP requests,
 * just define this function with a return(0) as single statement
 *
 */
unsigned int SPI_Ethernet_UserUDP(unsigned char *remoteHost, unsigned int remotePort,
unsigned int destPort, unsigned int reqLength)
        {
        unsigned int    len ;                   // my reply length
        unsigned char   *ptr ;                  // pointer to the dynamic buffer

        // reply is made of the remote host IP address in human readable format
        byteToStr(remoteHost[ 0], dyna) ;               // first IP address byte
        dyna[ 3]  = '.' ;
        byteToStr(remoteHost[ 1], dyna + 4) ;           // second
        dyna[ 7]  = '.' ;
        byteToStr(remoteHost[ 2], dyna + 8) ;           // third
        dyna[ 11]  = '.' ;
        byteToStr(remoteHost[ 3], dyna + 12) ;          // fourth

        dyna[ 15]  = ':' ;                              // add separator

        // then remote host port number
        intToStr(remotePort, dyna + 16) ;
        dyna[ 22]  = '[ ' ;
        intToStr(destPort, dyna + 23) ;
        dyna[ 29]  = '] ' ;
        dyna[ 30]  = 0 ;
```

```
// the total length of the request is the length of the dynamic string plus the text
// of the request
        len = 30 + reqLength ;
        // puts the dynamic string into the transmit buffer
        ptr = dyna ;
        while(*ptr)
                {
                SPI_Ethernet_putByte(*ptr++) ;
                }

// then puts the request string converted into upper char into the transmit buffer
        while(reqLength--)
                {
                SPI_Ethernet_putByte(toupper(ENC28J60_getByte())) ;
                }

        return(len) ;    // back to the library with the length of the UDP reply
        }
/*
 * main entry
 */
void    main()
        {
        ADCON1 = 0x00 ;           // ADC convertors will be used
        PORTA = 0 ;
        TRISA = 0xff ;           // set PORTA as input for ADC
        PORTB = 0 ;
        TRISB = 0xff ;           // set PORTB as input for buttons
        PORTD = 0 ;
        TRISD = 0 ;              // set PORTD as output
        /* starts ENC28J60 with :
         * reset bit on RC0
         * CS bit on RC1
         * my MAC & IP address
         * full duplex
         */
Spi_Init();     // Initialize SPI module

SPI_Ethernet_Init(&PORTC, 0, &PORTC, 1, myMacAddr, myIpAddr, SPI_Ethernet_FULLDUPLEX);
        while(1)                  // do forever
                {
                SPI_Ethernet_doPacket() ;   // process incoming Ethernet packets

                /*
                 * add your stuff here if needed
                 * SPI_Ethernet_doPacket() must be called as often as possible
                 * otherwise packets could be lost
                 */
                }
        }
```

HW Connection



Flash　Memory　Library（フラッシュメモリライブラリ）

　このライブラリはマイクロコントローラ　フラッシュメモリをアクセスするためのルーチンを提供する。
プロトタイプはPIC16,PIC18ファミリとは異なることに注意すること。

注意：P16ファミリ　フラッシュ仕様により、フラッシュライブラリはMCUに依存する。フラッシュメモリ操作をサポートするMCUには３つの種類がある。

１．フラッシュ読み込み操作のみをサポート。MCUのこのグループに対し、Flash_Read関数のみが実行される。
２．読み込みと書き込み操作がサポートされる。（書き込みは消去＋書き込みとして実行される）。MCU のこのグループに対し、リードとライト関数が実行される。
３．読み込み＋書き込み＋消去操作をサポート。MCU のこのグループに対し、読み込み＋書き込み＋消去関数が実行される。更に尚、フラッシュメモリのブロックは、書き込みに優先して消去されねばならない。（書き込み操作は消去＋書き込みとして実行されない）

フラッシュライブラリを使用する前にデータシートを参照すること。

Library Routines

Flash_Read
Flash_write
Flash_Erase

Flash_Read

原形	**unsigned** Flash_Read(**unsigned**　address); //　PIC16用 **unsigned short** Flash_Read(**long** address)); //　PIC18用
戻り値	フラッシュメモリからバイトでデータを返す。
解説	フラッシュメモリ内の指定アドレスからデータを読み込む
例	Flash_Read(0x0D00)

Flash_Write

原形	PIC16 用 **viod** Flash_Write(**unsigned** address, **unsigned int** *data) PIC18 用 **viod** Flash_Write(**long** address, **unsigned short** *fdata)
解説	フラッシュメモリへひとかたまりのデータを書き込む。PIC18 では、正確に６４バイトの大きさのデータが必要である。 データをフラッシュメモリに書き込む前に、この関数は対象となるメモリを消去するということを念頭におくこと。これは、もし書き込みが成功しなかった場合、以前のデータが失われるであろう、ということを意味する。

例	// ６４の連続した記憶場所に連続した値を書き込む。
	`// Write consecutive values in 64 consecutive locations`
	`char toWrite[64];`
	
	// 配列の初期化
	`// initialize array:`
	`for (i = 0; i < 63; i++) toWrite[i] = i;`
	`Flash_Write(0x0D00, toWrite);`

Flash_Erase

原形	**void** Flash_Erase(**unsigned** address)
解説	与えられたアドレスから始まる３２バイトのメモリブロックを消去する。フラッシュメモリが消去＋書き込み操作をサポートしないこれらの MCU に対してのみ実行される。(詳細はデータシートを参照すること)
例	アドレス＄０D00 から始まる３２バイトメモリブロックを消去する。 Flash_Erase(0x0D00)

Library Example

この例はPIC18用のフラッシュメモリへの簡単な書き込みの説明である。それからデータを読み込み且つ、PORTBにそれを出力する。

```
unsigned short i = 0, j = 0;
unsigned long addr;
unsigned short dataRd;
unsigned short dataWr[ 64] =
             {1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,
              1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,
              1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,
              1,2,3,4} ;

void main() {
  PORTB = 0;
  TRISB = 0;
  PORTC = 0;
  TRISC = 0;

  addr = 0x00000A30;           // valid for P18F452
  Flash_Write(addr, dataWr);

  addr = 0x00000A30;
  for (i = 0; i < 64; i++) {
    dataRd = Flash_Read(addr++);
    PORTB = dataRd;
    Delay_ms(500);
  }

} //~!
```

I2C　Library（I2C ライブラリ）

　Ｉ２Ｃフルマスター**MSSP**モジュールは多くのＰＩＣ　ＭＣＵモデルに有効である.
ｍｉｋｒｏＣはマスターＩ²Ｃモードをサポートするライブラリを提供する。

注意 : P18F8722 のように２つの I²C モジュールを持つある PICmicros は、あなたが使用したいモジュールを指定するよう要求する。単純に I²C に番号１または２を付与する。例えば、I2C2_Wr( ) ; のように。また、以前のバージョンのコンパイラとの後方互換性とコード管理をより容易にするため、複数の I2C モジュールをもつ MCU は I2C1 と同等の I²C ライブラリを持たねばならない。（例えば、I²C 操作のため、I2C1_Init( )の代わりに I2C_Init( )を使用可能である）

Library　Routines

I2C_Init
I2C_Start
I2C_Repeated_Start
I2C_Is_Idle
I2C_Rd
I2C_Wr
I2C_Stop

I2C_Init

原形	**void** I2C_Init (**long** clock)
解説	必要とされる clock で I²C を初期化する。(Fosc に関する正しい値についてはデバイスデータシートを参照すること。) I2C ライブラリの他の関数を使用する前に呼び出す必要がある。
必要事項	ライブラリは PORTB または PORTC 上に MSSP モジュールを要求する。
例	I2C_Init(100000)

I2C_Start

原形	**char** I2C_Start (**void**)
戻り値	もしエラー無しならば、関数は0を返す。
解説	もしI²Cバスが開放状態で、START信号を出すならば決定する。
必要事項	この関数を使用する前に、I²Cはコンフィグレーションされねばならない。I2C_Initを見よ。
例	I2C_Start();

I2C_Repeated_Start

原形	**void** I2C_Repeated_Start (**void**)
解説	繰り返しSTART信号を出す。
必要事項	この関数を使用する前に、I²Cはコンフィグレーションされねばならない。I2C_Initを見よ。
例	I2C_Repeated_Start() ;

I2C_Is_Idle

原形	**char** I2C_Is_Idle (**void**)
戻り値	もしI²Cバスが開放状態ならば1を、それ以外は0を返す。
解説	I²Cバスが開放状態かどうかテストする。
必要事項	この関数を使用する前に、I²Cはコンフィグレーションされねばならない。I2C_Initを見よ。
例	if (I2C_Is_Idle ( )) {…}

I2C_Rd

原形	**char** I2C_Rd (**char** ack)
戻り値	スレーブから１バイト返す。
解説	スレーブから１バイト読み込み、もしパラメータ ack が０ならばアクノリッジ (ACK : 承認) 信号を送信しないが、そうでなければ、アクノリッジを送信する。
必要事項	この関数を使用するためには START 信号が送出される必要がある。I2C_Start を見よ。
例	Temp = I2C_Rd (0) ; // データを読み込んだ後、アクノリッジ信号を送信しない。

I2C_Wr

原形	**char** I2C_Wr (**char** data)
戻り値	もしエラーがなければ０を返す。
解説	I²C バス経由でバイトデータ（パラメータデータ）を送信する。
必要事項	この関数を使用するためには START 信号が送出される必要がある。I2C_Start を見よ。
例	I2C_Write (0xA3);

I2C_Stop

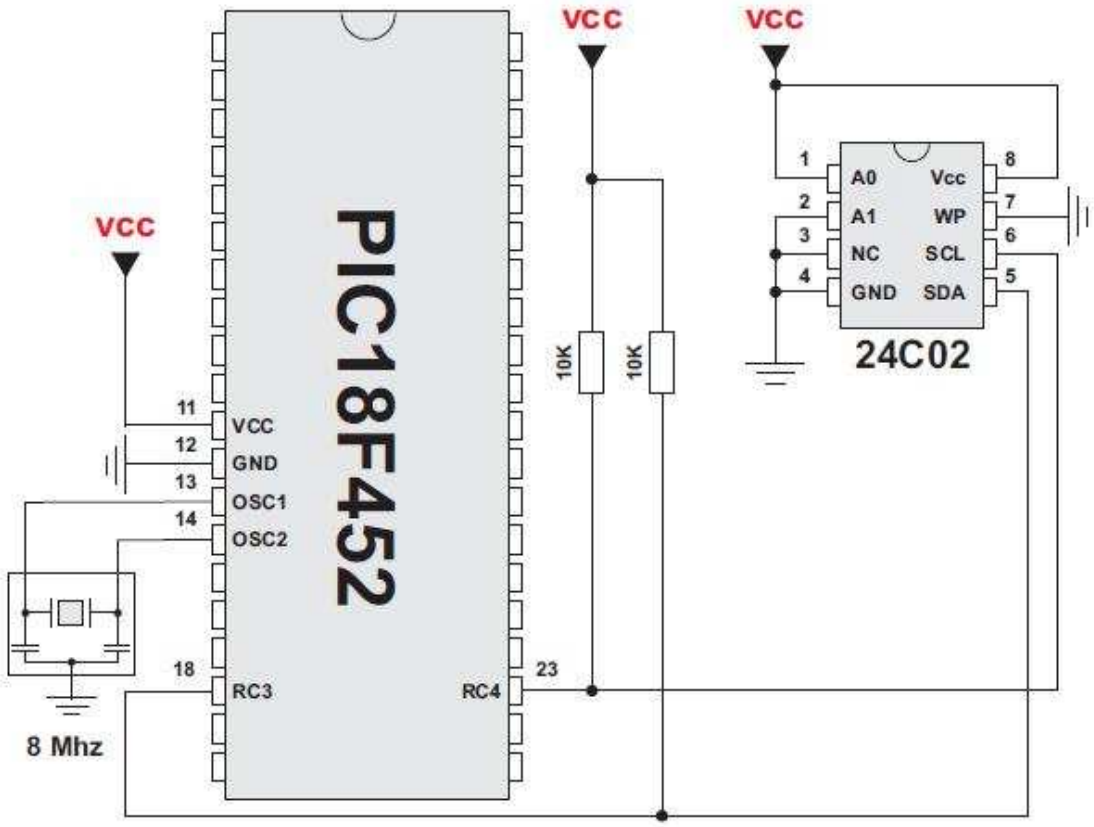
原形	**void** I2C_ Stop (**void**)
解説	STOP 信号を出す。
必要事項	この関数を使用する前に、I²C はコンフィグレーションされねばならない。I2C_Init を見よ。

Library　Example

このコードはI²C ライブラリ関数の使用方法を説明するものである。PIC　MCU はEEPROM　２４ｃ０２に(SCL、SDA 端子）接続される。プログラムはデータを EEPROM（データがアドレス２に書き込まれる）へ送る。それからわれわれはI2C 経由で EEPROM からデータを読み込み、サイクルが成功したかどうかチェックするために、PORTD へ値を送り出す。（２４ｃ０２の PIC へのインターフェース方法は以下の図を見ること。）

```
void main(){
  PORTB = 0;
  TRISB = 0;

  I2C_Init(100000);
  I2C_Start();                  // Issue I2C start signal
  I2C_Wr(0xA2);                 // Send byte via I2C  (command to 24c02)
  I2C_Wr(2);                    // Send byte (address of EEPROM location)
  I2C_Wr(0xF0);                 // Send data (data to be written)
  I2C_Stop();

  Delay_ms(100);

  I2C_Start();                  // Issue I2C start signal
  I2C_Wr(0xA2);                 // Send byte via I2C  (device address + W)
  I2C_Wr(2);                    // Send byte (data address)
  I2C_Repeated_Start();         // Issue I2C signal repeated start
  I2C_Wr(0xA3);                 // Send byte (device address + R)
  PORTB = I2C_Rd(0u);           // Read the data (NO acknowledge)
  I2C_Stop();
}
```

HW Connection



Keypad　Library（キーパッドライブラリ）

ｍｉｋｒｏCは４ｘ４キーパッドを機能するためのライブラリを提供する。ルーチンは４ｘ１、４ｘ２または４ｘ３キーパッドでの使用も可能である。このトピックの終わりの接続概略図を確認すること。

Library　Routines

Keypad_Init
Keypad_Read
Keypad_Released

Keypad_Init

原形	**void**　Keypad_Init (**char**　*port)
解説	キーパッドを機能するためポートを初期化する。キーパッドライブラリの他の関数を使用する前に、この関数が呼び出される必要がある。
例	Keypad_Init (&PORTB)

Keypad_Read

原形	**unsigned**　Keypad_Read (**void**)
戻り値	押されたキーにより 1..16 を、またキーが押されていないならば 0 を返す。
解説	どのキーが押されたのか確認する。押されたキーにより、関数は 1..16 を、キーが押されていないならば 0 を返す。
必要事項	ポートが適切に初期化される必要がある。Keypad_Init を見よ。
例	Kp = Keypad_Read( )

Keypad_Released

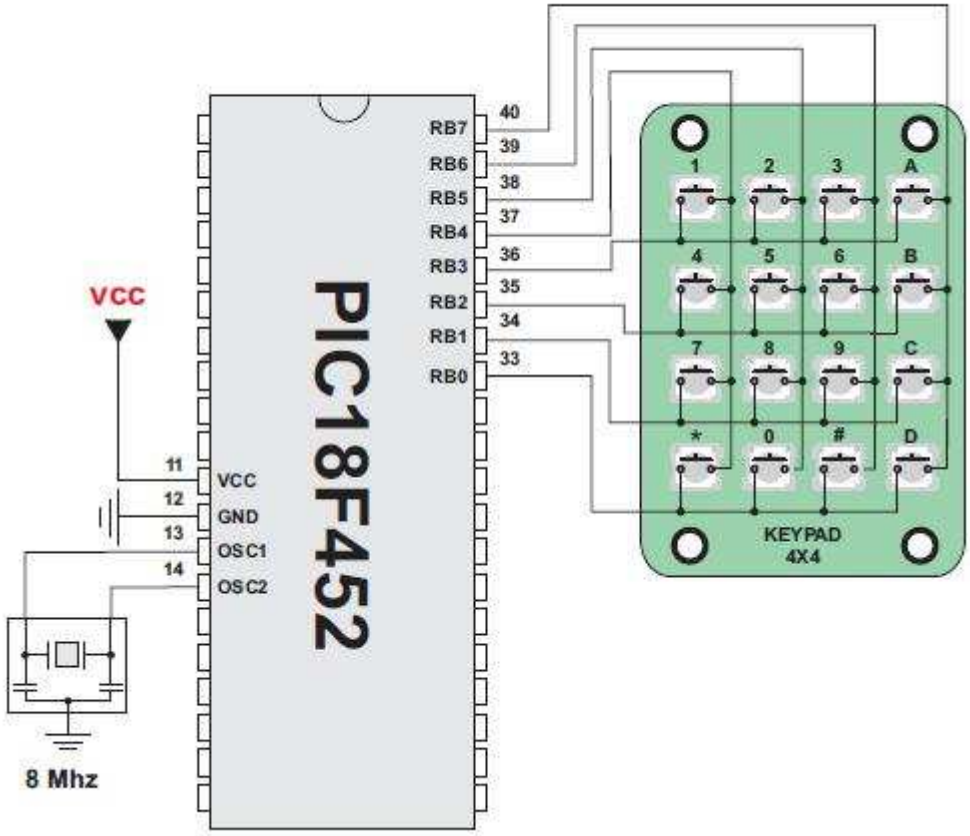
原形	**unsigned** Keypad_ Released (**void**)
戻り値	キーにより 1..16 を返す。
解説	Keypad_ Released 呼出しはブロッキング呼び出しである。いづれかのキーが押され且つ離されるまで、関数は待つ。離されると、関数はキーにより 1 から 16 を返す。
必要事項	ポートが適切に初期化される必要がある。Keypad_Init を見よ。
例	kp = Keypad_ Released ( ) ;

Library Example

次のコードはキーパッドを検査するために使用することが可能である。それは1～4列及び 1～4行のキーパッドをサポートする。キーパッド関数（1..16）により返されるコードは、ACII コード［0..9, A..F］に変換される。
加えて、小さな1バイトカウンタが第二の LCD 列で押されたキーの全番号を表示する。

```
unsigned short kp, cnt;
char txt[ 5];

void main() {
  cnt = 0;
  Keypad_Init(&PORTC);
  Lcd_Init(&PORTB);                 // Initialize LCD on PORTC
  Lcd_Cmd(LCD_CLEAR);               // Clear display
  Lcd_Cmd(LCD_CURSOR_OFF);          // Cursor off

  Lcd_Out(1, 1, "Key  :");
  Lcd_Out(2, 1, "Times:");

  do {
    kp = 0;

    //--- Wait for key to be pressed
    do
      //--- un-comment one of the keypad reading functions
      kp = Keypad_Released();
      //kp = Keypad_Read();
    while (!kp);

    cnt++;

    //--- prepare value for output
    if (kp > 10)
      kp += 54;
    else
      kp += 47;

    //--- print it on LCD
    Lcd_Chr(1, 10, kp);
    WordToStr(cnt, txt);
    Lcd_Out(2, 10, txt);

  } while (1);
} //~!
```

HW Connection



Lcd　Library（4 ビット　インターフェース）（LCD ライブラリ）

　ｍｉｋｒｏCは一般によく使われているＬＣＤ（４ビットインターフェース）との通信のためのライブラリを提供する。ＰＩＣとＬＣＤのHW（ハードウェア）接続を示す図をこの章の最後に示す。

注）次のライブラリ関数のいずれも使用前に、出力としてＬＣＤにポートを割り当てるよう注意すること。

Library　Routines

Lcd_Config
Lcd_Init
Lcd_Out
Lcd_Out_Cp
Lcd_Chr
Lcd_chr_Cp
Lcd_Cmd

Lcd_Config

原形	`void Lcd_Config(char *port, char RS, char EN, char WR, char D7, char D6, char D5, char D4);`
戻り値	なし
解説	あなたが指定する端子設定のポートでLCD を初期化すること。パラメータ RS, EN, D7..D4 は０から７の値の組み合わせである必要がある。（例えば、３，６，０，７，２，１，４）
必要事項	なし
例	Lcd_Config（&PORTD、１，２，０，３，５，４，６）

Lcd_Init

原形	**void** Lcd_Init (**void**)
戻り値	なし
解説	デフォルトの端子設定のポートでLCDを初期化する。（この章の最後の接続図を見よ。） D7 -> PORT.7, D6 -> PORT.6, D5 -> PORT.5, D4 -> PORT.4, E -> PORT.3, RS -> PORT.2.
必要事項	なし
例	Lcd_Init(&PORTB)

Lcd_Out

原形	**void** Lcd_Out (**char** row, **char** col, **char** *text)
戻り値	なし
解説	LCD 上の指定した列と行にテキストを表示する（パラメータ row と col)。ストリング変数とリテラル共に text として引渡し可能である。
必要事項	LCD に割当てたポートが初期化されねばならない。Lcd_config またはLcd_Init をみよ。
例	Lcd_Out(1,3,"Hello!") ; // 第１行の３文字目に“Hello!”を出力する。

Lcd_Out_Cp

原形	**void** Lcd_Out_Cp (**char** *text)
戻り値	なし
解説	LCD 上の現在のカーソル位置に text を表示する。ストリング変数とリテラル共に text として引渡し可能である。
必要事項	LCD 用のポートが初期化されねばならない。Lcd_config またはLcd_Init を見よ。
例	Lcd_Out_Cp(“Here!”) ; // 現在のカーソル位置に“Here！“と出力する。

Lcd_Chr

原形	**void** Lcd_Out_Chr (**char** row, **char** col, **char** character)
戻り値	なし
解説	LCD 上の指定の列と行へ character を出力する（パラメータ列と行)。変数とリテラル共に character として引渡し可能である。
必要事項	LCD に割当てたポートが初期化されねばならない。Lcd_Config と Lcd_Init を見よ。
例	Lcd_Chr(2,3,'i') ; // 第２行の３文字目に“i”を出力する。

Lcd_chr_Cp

原形	**void** Lcd_Chr_Cp (**char** character)
戻り値	なし
解説	LCD 上の現在のカーソル位置に character を表示する。変数とリテラル共に文字列として引渡し可能である。
必要事項	LCD に割当てたポートは初期化されねばならない。Lcd_Config または Lcd_Init を見よ。
例	Lcd_chr_Cp('e') ; // 現在のカーソル位置に‘ｅ’を出力する。

Lcd_Cmd

原形	**void** Lcd_Cmd (**char** command)
戻り値	なし
解説	LCD へ command を送る。あなたは関数に対し、前に定義した定数の１つを引渡し可能である。 有効なコマンドの完全リストが次のページに示される。
必要事項	LCD に割当てたポートは初期化されねばならない。Lcd_Config と Lcd_Init を見よ。
例	Lcd_Cmd(LCD_Clear) ; // ＬＣＤディスプレイをクリアする。

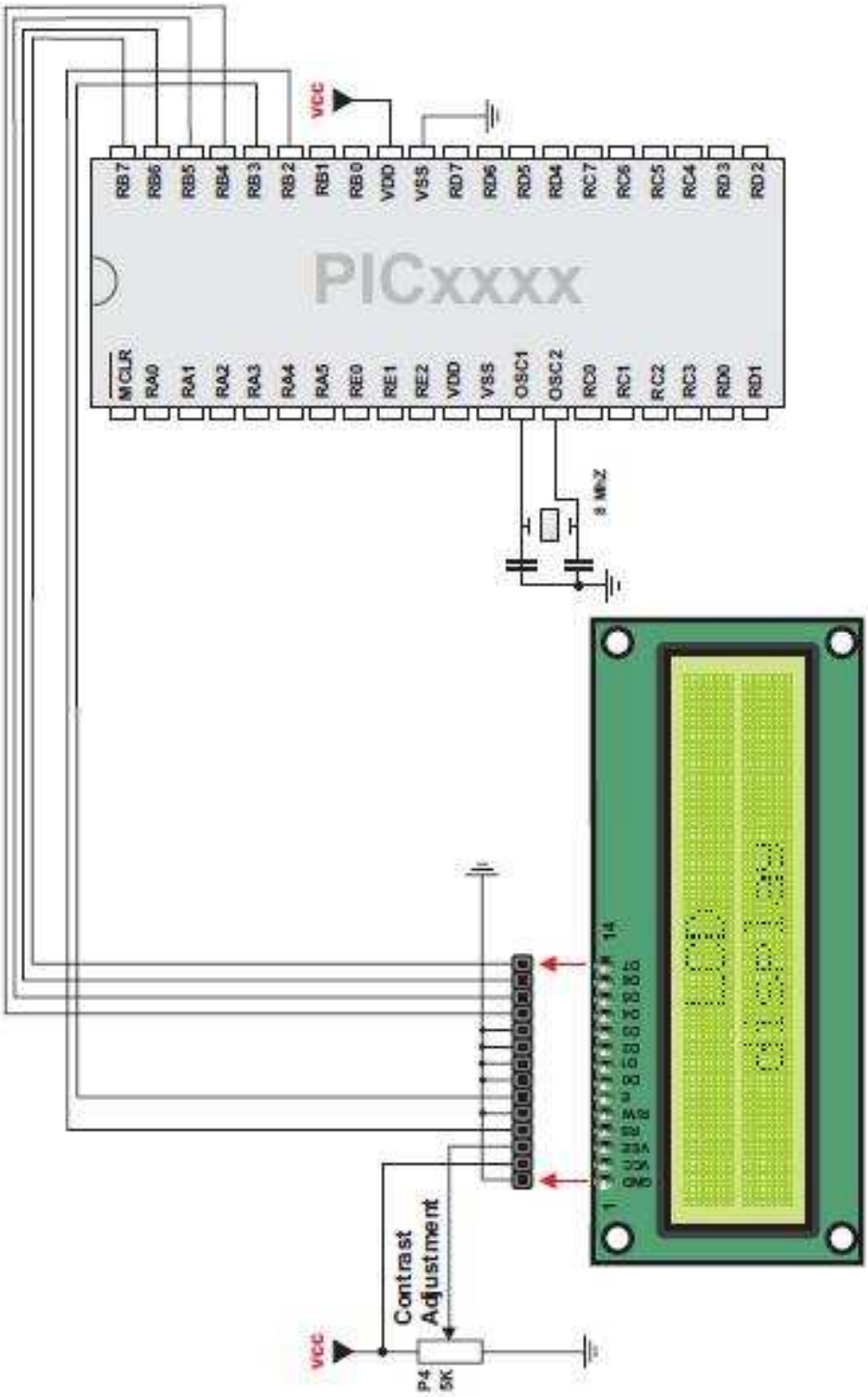
LCD Command

LCD Command	Purpose
LCD_FIRST_ROW	第１列へカーソルを移動
LCD_SECOND_ROW	第２列へカーソルを移動
LCD_THIRD_ROW	第３列へカーソルを移動
LCD_FOURTH_ROW	第４列へカーソルを移動
LCD_CLEAR	表示をクリアする
LCD_RETURN_HOME	ホーム位置へカーソルを戻し、シフトした表示を原点へ戻す。表示データRAMへは影響しない。
LCD_CURSOR_OFF	カーソルをオフする。
LCD_UNDERLINE_ON	アンダーライン　カーソルをオンする。
LCD_BLINK_CURSOR_ON	ブリンク　カーソルをオンする。
LCD_MOVE_CURSOR_LEFT	表示データＲAMを変えずに、カーソルを左へ移動する。
LCD_MOVE_CURSOR_RIGHT	表示データＲAMを変えずに、カーソルを右へ移動する。
LCD_TURN_ON	ＬＣD表示をオンする。
LCD_TURN_OFF	ＬＣD表示をオフする。
LCD_SHIFT_LEFT	表示データＲAMを変えずに表示を左へシフトする。
LCD_SHIFT_RIGHT	表示データＲAMを変えずに表示を右へシフトする

Library Example

```
char *text = "mikroElektronika";

void main() {
  TRISB = 0;                    // PORTB is output
  Lcd_Init(&PORTB);             // Initialize LCD connected to PORTB
  Lcd_Cmd(Lcd_CLEAR);           // Clear display
  Lcd_Cmd(Lcd_CURSOR_OFF);      // Turn cursor off
  Lcd_Out(1, 1, text);          // Print text to LCD, 2nd row, 1st column
} //~!
```

Hardware Connection



P242

Lcd　Custom　Library（LCD カスタムライブラリ）

　ｍｉｋｒｏＣは通常よく使われるＬＣＤ（４ビットインターフェース）との通信のためのライブラリを提供する。ＰＩＣとＬＣＤに特注のHW接続を示す図をこの章の最後に示す。

Library routines

Lcd_Custom_Config
Lcd_Custom_Out
Lcd_Custom_Out_Cp
Lcd_Custom_Chr
Lcd_Custom_Chr_Cp
Lcd_Custom_Cmd

Lcd_Custom_Config

原形	`void Lcd_Custom_Config(char * data_port, char D7, char D6, char D5, char D4, char * ctrl_port, char RS, char WR, char EN);`
戻り値	なし
解説	あなたが指定する設定端子でＬＣＤデータポートとコントロールポートを初期化する。
必要事項	LCD に割当てたポートは初期化されねばならない。Lcd_Config を見よ。
例	Lcd_ Custom_Out(1,3, “Hello!”) ; //　第１行の３文字目に “Hello!” を表示する。

Lcd_Custom_Out

原形	**void** Lcd_ Custom_ Out (**char** row, **char** col, **char** *text)
戻り値	なし
解説	LCD 上の指定された列と行（変数 row と col）に text を出力する。ストリング変数とリテラル共に text として引渡し可能である。
必要事項	LCD に割り当てたポートは初期化されねばならない。Lcd_Config を見よ。
例	Lcd_ Custom_ Out(1,3,"Hello!") ; // 第１行目の３文字目に "Hello!" を出力する。

Lcd_Custom_Out_Cp

原形	**void** Lcd_ Custom_ Out_Cp (**char** *text)
戻り値	なし
解説	LCD 上の現在のカーソル位置に text を出力する。ストリング変数とリテラル共に text として引渡し可能である。
必要事項	LCD に割当てたポートは初期化されねばならない。Lcd_config を見よ。
例	Lcd_ Custom_ Out_Cp("Here!") ; // 現在のカーソル位置に "Here!" を出力する。

Lcd_Custom_Chr

原形	**void** Lcd_ Custom_ Chr (**char** row, **char** col, **char** character)
戻り値	なし
解説	LCD 上の指定された列と行に（変数 row と col）character を出力する。変数とリテラル共に chracter として引渡し可能である。
必要事項	LCD に割当てたポートは初期化されねばならない。Lcd_Config をみよ。
例	Lcd_Custom_Chr(2,3,'i') ; // 第２行の３文字目に‘i’を出力する。

Lcd_Custom_Chr_Cp

原形	**void** Lcd_ Custom_ Chr_Cp (**char** character)
戻り値	なし
解説	LCD 上の現在のカーソル位置に character を出力する。変数とリテラル共に character として引渡し可能である。
必要事項	LCD に割当てたポートは初期化されねばならない。Lcd_config を見よ。
例	Lcd_ Custom_ Out_Cp ('e') ; // 現在のカーソル位置に‘e’を出力する。

Lcd_Custom_Cmd

原形	void Lcd_Custom_Cmd (char command)
戻り値	なし
解説	LCD へコマンドを送る。あなたは関数に対し、以前定義した定数の一つを引渡し可能である。有効なコマンドの完全な表は次のページに示す。
必要事項	LCD に割当てたポートは初期化されねばならない。Lcd_config を見よ。
例	Lcd_Custom_Cmd(Lcd_Clear) ; // LCD ディスプレイをクリアする。

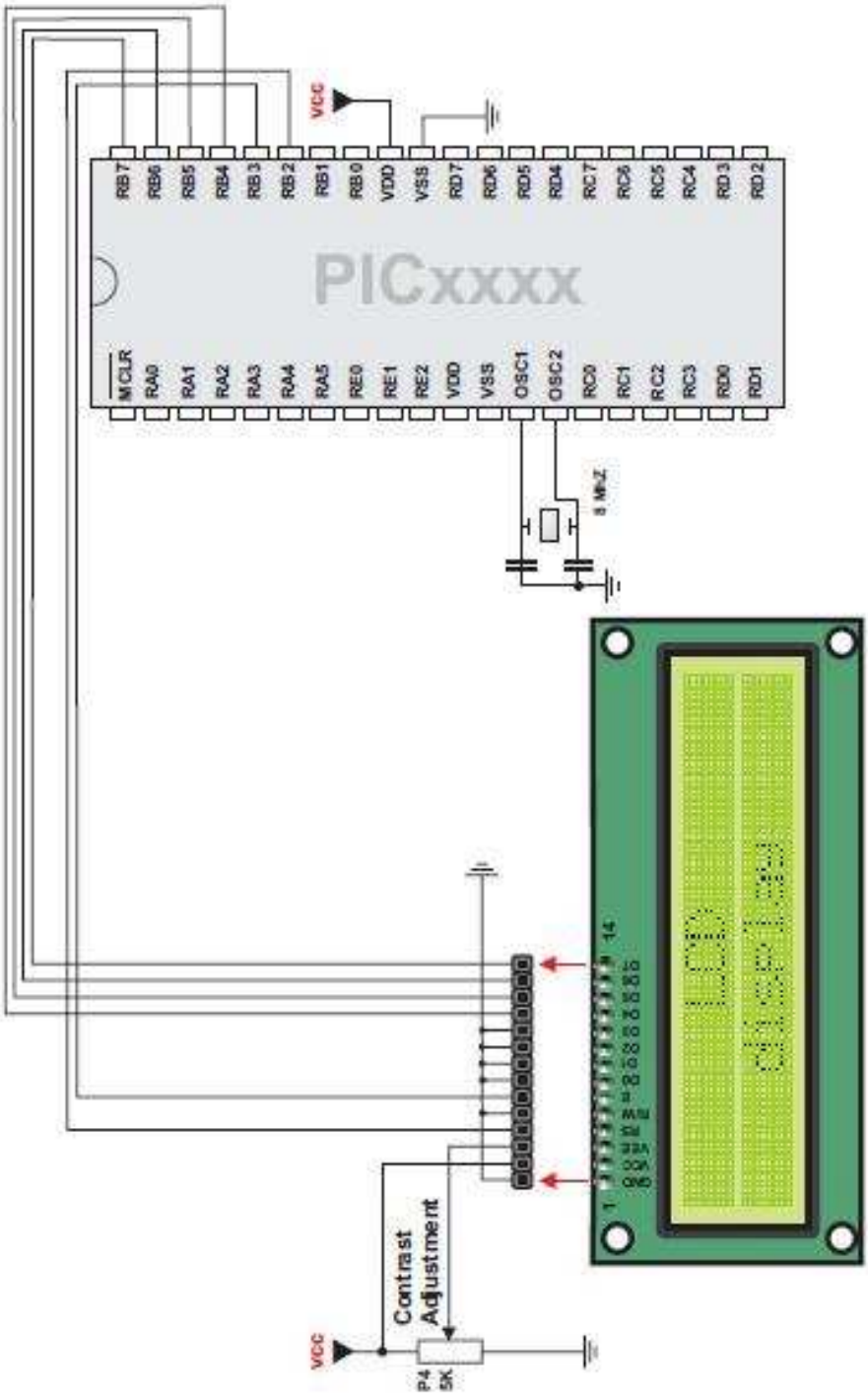
LCD Commands

LCD Command	Purpose
LCD_FIRST_ROW	第１列へカーソルを移動
LCD_SECOND_ROW	第２列へカーソルを移動
LCD_THIRD_ROW	第３列へカーソルを移動
LCD_FOURTH_ROW	第４列へカーソルを移動
LCD_CLEAR	表示をクリアする
LCD_RETURN_HOME	ホーム位置へカーソルを戻し、シフトした表示を原点へ戻す。表示データRAM へは影響しない。
LCD_CURSOR_OFF	カーソルをオフする。
LCD_UNDERLINE_ON	アンダーライン　カーソルをオンする。
LCD_BLINK_CURSOR_ON	ブリンク　カーソルをオンする。
LCD_MOVE_CURSOR_LEFT	表示データＲAMを変えずに、カーソルを左へ移動する。
LCD_MOVE_CURSOR_RIGHT	表示データＲAMを変えずに、カーソルを右へ移動する。
LCD_TURN_ON	ＬCD表示をオンする
LCD_TURN_OFF	ＬCD表示をオフする。
LCD_SHIFT_LEFT	表示データＲAMを変えずに表示を左へシフトする。
LCD_SHIFT_RIGHT	表示データＲAMを変えずに表示を右へシフトする。

Library Example

```
char *text = "mikroElektronika";

void main() {
  TRISB = 0;                        // PORTB is output
  Lcd_Custom_Config(&PORTB,7,6,5,4,&PORTB,3,0,2);// Initialize LCD connected to
PORTB
  Lcd_Custom_Cmd(Lcd_CLEAR);              // Clear display
  Lcd_Custom_Cmd(Lcd_CURSOR_OFF);         // Turn cursor off
  Lcd_Custom_Out(1, 1, text);             // Print text to LCD, 2nd row, 1st col-
umn
} //~!
```

Hardware Connection



Lcd8　Library（８ビット　インターフェース）（LCD8 ライブラリ）

　ｍｉｋｒｏＣは通常よく使われる８ビットインターフェースＬＣＤ（日立 HD44780 コントローラ）との通信のためのライブラリを提供する。ＰＩＣとＬＣＤに特注のHW接続を示す図をこの章の最後に示す。

Library routines

Lcd8_Config
Lcd8_Init
Lcd8_Out
Lcd8_Out_Cp
Lcd8_Chr
Lcd8_Chr_Cp
Lcd8_Cmd

Lcd8_Config

原形	`void Lcd8_Config(char *ctrlport, char *dataport, char RS, char EN, char WR, char D7, char D6, char D5, char D4, char D3, char D2, char D1, char D0);`
戻り値	なし
解説	あなたが指定する端子設定で LCD の制御ポート（ctrlport）とデータポート（dataport）を初期化する。変数 RS,EN 及び WR は０－７の範囲でなければならない。変数 D7..D0 は数値０－７の組み合わせでなければならない。（例えば、3,6,5,0,7,2,1,4）
必要事項	なし
例	Lcd8_Config(&PORTC,&PORTD,0,1,2,6,5,4,3,7,1,2,0)

Lcd8_Init

原形	**void** Lcd8_Init (**char** *ctrlport, **char** *dataport)
戻り値	なし
解説	デフォルト端子設定で LCD の制御ポート（ctrlport）とデータポート（dataport）を初期化する。（この章の終わりの接続図を見よ。） E -> ctrlport.3, RS -> ctrlport.2, R/W -> ctrlport.0, D7 -> dataport.7, D6 -> dataport.6, D5 -> dataport.5, D4 -> dataport.4, D3 -> dataport.3, D2 -> dataport.2, D1 -> dataport.1, D0 -> dataport.0
必要事項	なし
例	Lcd8_Init(&PORTB,&PORTC)

Lcd8_Out

原形	**void** Lcd8_Out (**char** row, **char** col, char *text)
戻り値	なし
解説	LCD 上の指定された列と行に text を出力する（変数行と列）。ストリング変数とリテラル共に text として出力されうる。
必要事項	LCD に割当てられたポートは初期化されねばならない。Lcd8_Config または Lcd_Init を見よ。
例	Lcd8_Out(1,3,"Hellow!") ; // 第一行目の３文字目に "Hello!" を出力する。

Lcd8_Out_Cp

原形	**void** Lcd8_Out _Cp (**char** *text)
戻り値	なし
解説	LCD 上の現在のカーソル位置に text を出力する。ストリング変数とリテラル共に text として通過可能である。
必要事項	LCD に割当てられたポートは初期化されねばならない。Lcd8_Config または Lcd_Init を見よ。
例	Lcd8_Out_Cp ("Hellow!") ; // 現在のカーソル位置に "Hello ! " を出力する。

Lcd8_Chr

原形	**void** Lcd8_ Chr ((**char** row, **char** col, **char** character)
戻り値	なし
解説	LCD 上の指定された列と行に（変数 row と col）character を出力する。変数とリテラル共に character として通過可能である。
必要事項	LCD に割当てられたポートは初期化されねばならない。Lcd8_Config または Lcd_Init を見よ。
例	Lcd8_ Out (2,3,"i") ; // 第２行３文字目に‘i’を出力する。

Lcd8_Chr_Cp

原形	**void** Lcd8_ Cp (**char** character)
戻り値	なし
解説	LCD 上の現在のカーソル位置に character を出力する。変数とリテラル共に character として通過可能である。
必要事項	LCD に割当てられたポートは初期化されねばならない。Lcd8_Config または Lcd_Init を見よ。
例	Lcd8_ Out _ Cp ("e") ; // 現在のカーソル位置に‘e’を出力する。

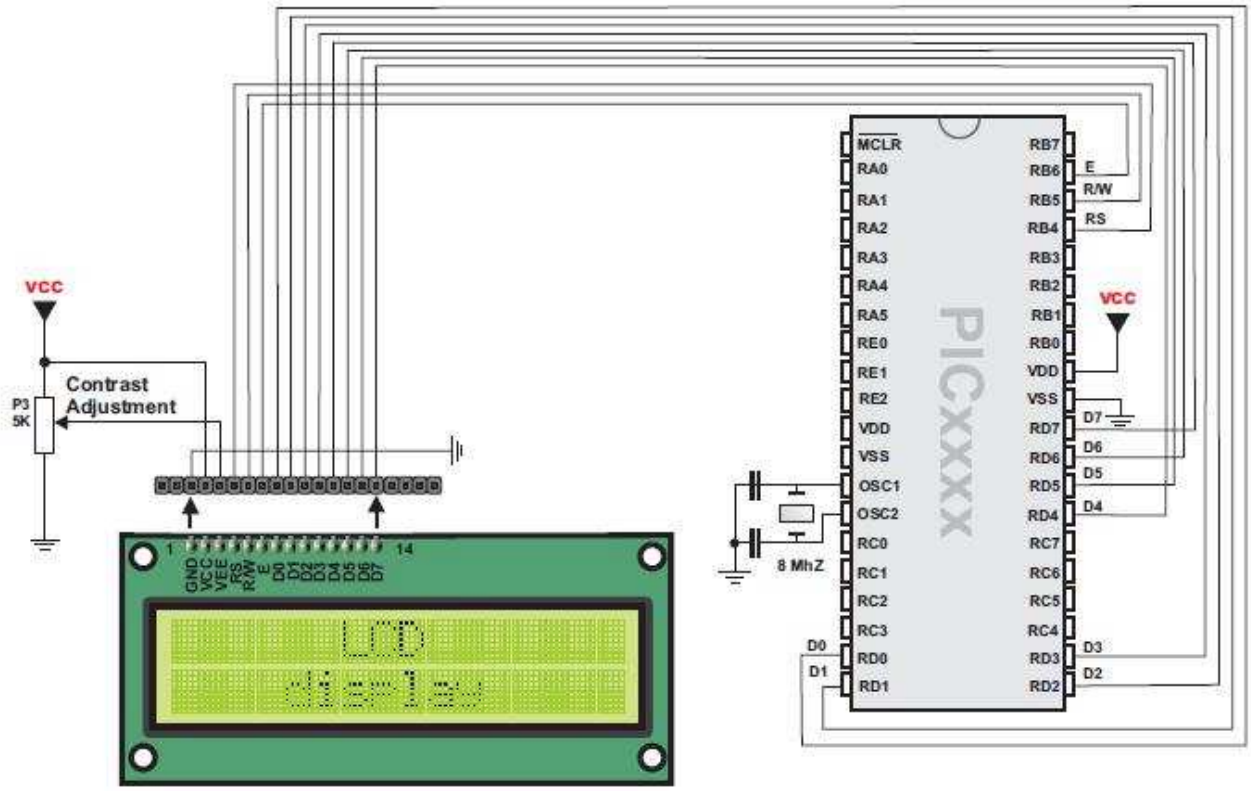
Lcd8_Cmd

原形	**void** Lcd8_ Cmd (**char** command)
戻り値	なし
解説	LCD へ command を送る。あなたは関数に対し以前に定義した定数の一つを通過可能である。有効なコマンドの完全な表は 186 ページにある。
必要事項	LCD に割当てられたポートは初期化されねばならない。Lcd8_Config または Lcd_Init を見よ。
例	Lcd8_ Cmd (Lcd_clear) ; // LCD ディスプレイをクリアする。

Library Example

```
char *text = "mikroElektronika";

void main() {
  TRISB = 0;                    // PORTB is output
  TRISC = 0;                    // PORTC is output
  Lcd8_Init(&PORTB, &PORTC);    // Initialize LCD at PORTB and PORTC
  Lcd8_Cmd(Lcd_CURSOR_OFF);     // Turn off cursor
  Lcd8_Out(1, 1, text);         // Print text on LCD
}
```

HW connection



GLCD　Library（GLCD ライブラリ）

　ｍｉｋｒｏＣはグラフィックＬＣＤに描画したり文字を書いたりするためのライブラリを提供する。これらのルーチンはよく使用される１２８ｘ６４ＧＬＣＤで動作し、且つＰＩＣ１８ファミリでのみ動作する。

Library Routines

Basic routines：

```
Glcd_Init
Glcd_Set_Side
Glcd_Set_Page
Glcd_Set_X
Glcd_Read_Data
Glcd_Write_Data
```

Advanced routines：

```
Glcd_Fill
Glcd_Dot
Glcd_Line
Glcd_V_Line
Glcd_H_Line
Glcd_Rectangle
Glcd_Box
Glcd_Circle
Glcd_Set_Font
Glcd_Write_Char
Glcd_Write_Text
Glcd_Image
```

Glcd_Init

原形	`void Glcd_Init(unsigned char *ctrl_port, char cs1, char cs2, char rs, char rw, char rst, char en, unsigned char *data_port);`
戻り値	なし
解説	あなたが指定する端子設定の data_port の下位バイトを初期化する。変数 cs1,cs2,rs,rw,rst 及び en は全て有効なポートの端子に割り当て可能である。GLCD ライブラリの他のルーチンを使用する前に、この関数が呼び出されねばならない。
必要事項	なし
例	Glcd_Init (PORTB,PORTC,3,5,7,1,2) ;

Glcd_Set_Side

原形	**void** Glcd_Set_Side(**unsigned short** x)
戻り値	なし
解説	GLCD の側面、左または右を選択する。パラメータ x は側面を指定する。０から６３の値は左側を指定し、６４以上は右側を指定する。GLCD の正確な位置を指定するには、関数 Glcd_Set_Side, Glcd_Set_X, Glcd_Set_Page を使うこと。それからあなたは Glcd_Write_Data または Glcd_Read_Data をその場所で使用可能である。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Set_Side (0) ;

Glcd_Set_Page

原形	**void** Glcd_Set_Page (**unsigned short** Page)
戻り値	なし
解説	GLCD のページ、技術的には表示行を選択する。変数 page は０～７をとる。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Set_ Page (5) ;

Glcd_Set_X

原形	**void** Glcd_Set_X (**unsigned short** X_pos)
戻り値	なし
解説	与えられたページ内の GLCD の左端から x ドットへ配置する。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Set_X (25) ;

Glcd_Read_Data

原形	**void** Glcd_ Read_Data (**unsigned short** data)
戻り値	GLCD メモリから１ワード
解説	GLCD メモリの現在位置からデータを読み込む。GLCD 上の正確な位置を指定するため、関数 Glcd_Set_Side 、Glcd_Set_X 、Glcd_Set_Page を使用すること。それからあなたはその位置で Glcd_Write_Data 、Glcd_Read_Data を使用できる。
必要事項	GLCD メモリの現在位置からデータを読み出す。
例	tmp=Glcd_ Read_Data ( ) ;

Glcd_Write_Data

原形	**void** Glcd_ Write _Data (**unsigned short** data)
戻り値	なし。
解説	GLCD メモリ内の現在位置へ data を書き込み、次の位置へ移動する。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_ Write _Data (data) ;

Glcd_Fill

原形	**void** Glcd Fill (**unsigned short** pattern)
戻り値	なし。
解説	バイト形式で GLCD メモリを満たす。GLCD 画面を消去するためには GLCD_Fill(0)を使う。 画面を完全に埋めるには GLCD_Fill($FF)を使う。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd Fill (0) ; // 画面を消去する。

Glcd_Dot

原形	**void** Glcd_Dot (**unsigned short** x, **unsigned short** y, **char** color)
戻り値	なし。
解説	GLCD 上の座標（ｘ、ｙ）へ１ドット描画する。変数 color はドットの状態を決定する。０はドット消去、１は１ドットを置き、２はドットの状態を反転する。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Dot (0 ,0 ,2) ; // 左上端のドットを反転する

Glcd_Line

原形	**void** Glcd_Line (**int** x1, int y1, **int** x2, int y2, **char** color)
戻り値	なし。
解説	GLCD 上の座標（ｘ1、ｙ1）から座標（ｘ2、ｙ2）へ直線を描画する。変数 color はドットの状態を決定する。０は空の直線（ドット消去)、１は実線（ドットを置く)、２は“スマート”な直線（各ドットを反転）を描く。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Line (0,63 ,50,0 ,2) ;

Glcd_V_Line

原形	**void** Glcd_V_Line (**unsigned short** y1, **unsigned short** y2, **unsigned short** x, **char** color)
戻り値	なし。
解説	Glcd_Line と同じく、GLCD 上の座標（ｘ、ｙ1）から座標（ｘ、ｙ2）へ垂直線を描画する。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_V_Line (0,63 ,0,1 ,1) ;

Glcd_H_Line

原形	**void** Glcd_ H _Line (**unsigned short** x1, **unsigned short** x2, **unsigned short** y, **char** color)
戻り値	なし。
解説	Glcd_Line と同じく、GLCD 上の座標（ｘ1、ｙ）から座標（ｘ2、ｙ）へ水平線を描画する。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_ H _Line (0,63 ,0,1 ,1) ;

Glcd_Rectangle

原形	`void Glcd_Rectangle(unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2, char color);`
戻り値	なし。
解説	GLCD 上に四角形を描画する。変数(x1,y1)は左上端を設定し、(x2,y2) は右下端を設定する。変数 color は輪郭を定義する。0 は空の輪郭を描き（ドットを消す)、1 は実線で描き（ドットを置く)、2 は“スマート”な輪郭（各ドットを反転）を描く。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Rectangle (10,0 ,30,35 ,1) ;

Glcd_Box

原形	`void Glcd_Box(unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2, char color);`
戻り値	なし。
解説	GLCD 上に長方形を描画する。変数(x1,y1)は左上端を設定し、(x2,y2) は右下端を設定する。変数 color は塗りつぶしを定義する。0 は白の長方形を描き（ドットを消す)、1 は塗りつぶした長方形を描き（ドットを置く)、2 は反転した長方形（各ドットを反転）を描く。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Rectangle (10,0 ,30,35 ,1) ;

Glcd_Circle

原形	**void** Glcd_Circle (**int** x, **int** y, **int** radius, **char** color)
戻り値	なし。
解説	半径 radius で（x,y）を中心とした円を GLCD 上に描画する。変数 color は円の線を定義する。０は空の線を描き（ドットを消す)、１は実線を描き（ドットを置く)、２は“スマート”な線（各ドットを反転）を描く。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Rectangle (63,31,25 ,2) ;

Glcd_Set_Font

原形	**void** Glcd_Set_Font(**const char** *font, **unsigned short** font_width, **unsigned short** font_height, **unsigned** font_offset);
戻り値	なし。
解説	Glcd_Write_Char、Glcd_Write_Text ルーチンのためのフォントを設定する。変数 font はバイト配列でフォーマットされる必要がある。 変数 font_width と font_hight は文字のドットの幅と高さを指定する。フォント幅は 128 ドットを超えてはならない、またフォント高さは８ドットを超えてはならない。 変数 font_offset は、供与のフォントで始まる ASCII 文字を決定する。ライブラリで提供されたデモ用フォントは３２オフセットを有し、それはそれらが空白で始まることを意味する。 あなたはファイル“Glcd_Fonts.c”で供与されるガイドラインに従い、あなた自身のフォントを作ることができる。このファイルは GLCD のためのデフォルト　フォントを含み、あなたのインストール　フォルダ“Extra　Examples”＞“GLCD”に置かれている。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	//　空白(32)で始まる特注の 5x8 フォントを使う。 Glcd_Set_Font (myfont_5x8,5,8,32) ;

Glcd_Write_Char

原形	`void Glcd_Write_Char(unsigned short character, unsigned short x, unsigned short page, char color);`
戻り値	なし。
解説	page（８つの GLCD 行０～７の内の）のディスプレイの左上端から x ドット離れた位置に character を出力する。変数 color は塗りつぶしを定義する。０は“白”文字を出力し（ドットを消す)、１は実線文字を出力し（ドットを置く)、２は“スマート”な文字（各ドットを反転）を出力する。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Set_Char（‘C’,0,0,1）;

Glcd_Write_Text

原形	`void Glcd_Write_Text(char *text, unsigned short x, unsigned short page, unsigned short color);`
戻り値	なし。
解説	page（８つの GLCD 行０～７の内の）のディスプレイの左上端から x ドット離れた位置に text を出力する。変数 color は塗りつぶしを定義する。０は“白”文字を出力し（ドットを消す)、１は実線文字を出力し（ドットを置く)、２は“スマート”な文字（各ドットを反転）を出力する。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Set_Text（“Hello　world!”,0,0,1）;

Glcd_Image

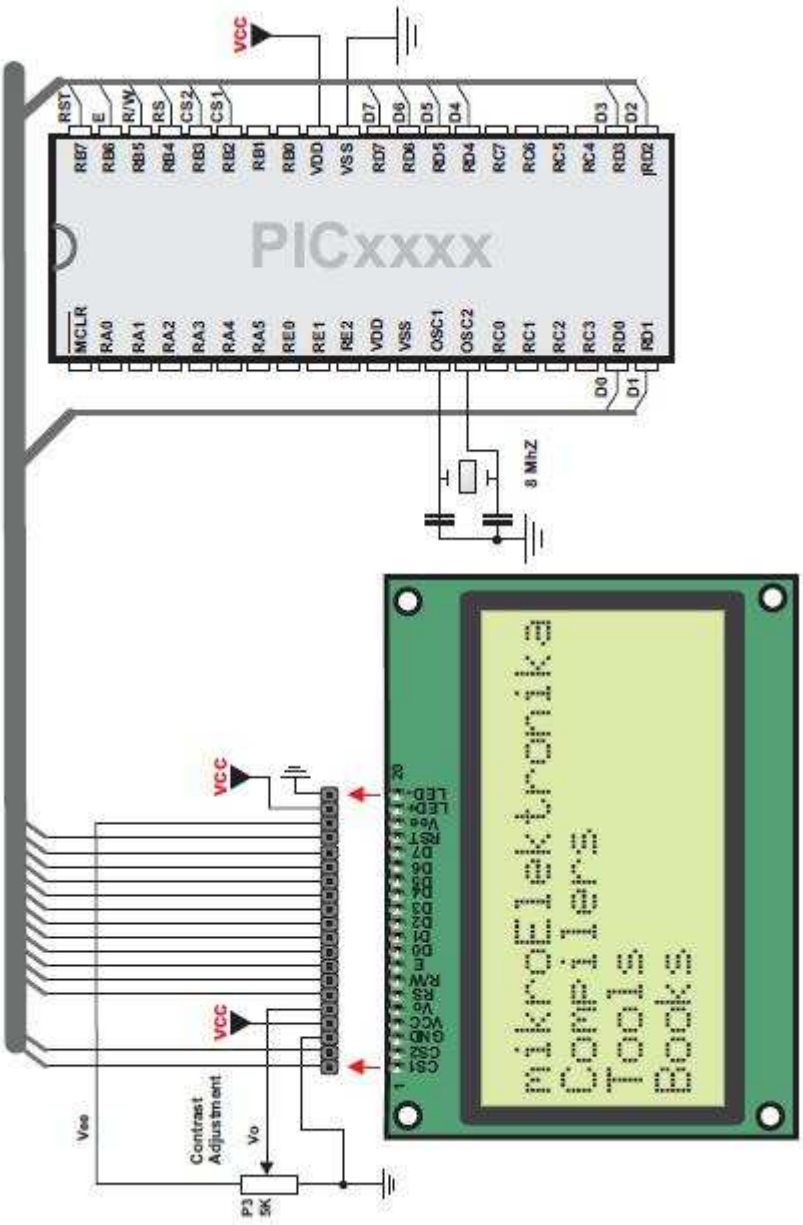
原形	**void** Glcd_Image (**const char** *image)
戻り値	なし。
解説	GLCD へビットマップイメージを表示する。image は整数配列としてフォーマットされねばならない。GLCD 上に最適表示するため定数配列にイメージを変換するMikroCの統合bitmap-to-LCD エディタ（メニューの option Tools＞BMP 2 LCD）を使用する。
必要事項	GLCD は初期化されねばならない。Glcd_Init を参照せよ。
例	Glcd_Image (my_image) ;

Library Example

次の描画用デモはGLCD ライブラリのアドバンスト　ルーチンをテストするものである。

```
unsigned short j, k;

void main() {

  Glcd_Init(PORTB, 2, 0, 3, 5, 7, 1, PORTD);

  // Set font for displaying text
  Glcd_Set_Font(@FontSystem5x8, 5, 8, 32);

  do {
    // Draw circles
    Glcd_Fill(0);  // Clear screen
    Glcd_Write_Text("Circles", 0, 0, 1);
    j = 4;
    while (j < 31) {
      Glcd_Circle(63, 31, j, 2);
      j += 4;
    }
    Delay_ms(4000);

    // Draw boxes
    Glcd_Fill(0);  // Clear screen
    Glcd_Write_Text("Rectangles", 0, 0, 1);
    j = 0;
    while (j < 31) {
      Glcd_Box(j, 0, j + 20, j + 25, 2);
      j += 4;
    }
    Delay_ms(4000);

    // Draw Lines
    Glcd_Fill(0);  // Clear screen
    Glcd_Write_Text("Lines", 0, 0, 1);
    for (j = 0; j < 16; j++) {
      k = j*4 + 3;
      Glcd_Line(0, 0, 127, k, 2);
    }
    for (j = 0; j < 31; j++) {
      k = j*4 + 3;
      Glcd_Line(0, 63, k, 0, 2);
    }
    Delay_ms(4000);
  } while (1);
}//~!
```

Hardware Connection



P263

T6963c Graphic Lcd Library（T6963C GLCD ライブラリ）

MikroC は、東芝 T6963C グラフィック LCD（サイズ変更可能）の描画と書き込みのためのライブラリを提供する。

Library Routines

```
T6963C_Init
T6963C_writeData
T6963C_writeCommand
T6963C_setPtr
T6963C_waitReady
T6963C_fill
T6963C_dot
T6963C_write_char
T6963C_write_text
T6963C_line
T6963C_rectangle
T6963C_box
T6963C_circle
T6963C_image
T6963C_sprite
T6963C_set_cursor
T6963C_clearBit
T6963C_setBit
T6963C_negBit
T6963C_displayGrPanel
T6963C_displayTxtPanel
T6963C_setGrPanel
T6963C_setTxtPanel
T6963C_panelFill
T6963C_grFill
T6963C_txtFill
T6963C_cursor_height
T6963C_graphics
T6963C_text
T6963C_cursor
T6963C_cursor_blink
T6963C_Init_240x128
T6963C_Init_240x64
```

T6963C_Init

原形	`void T6963C_init(unsigned int w, unsigned int h, unsigned int fntW, unsigned int *data, unsigned int *cntrl, unsigned int bitwr, unsigned int bitrd, unsigned int bitcd, unsigned int bitreset);`
戻り値	なし。
解説	グラフィック LCD コントローラを初期化する。すべての T6963C ライブラリルーチンを呼び出す前に、この関数が呼び出されねばならない。 width－ディスプレイにおける水平ピクセル（x）の数 height－ディスプレイにおける垂直ピクセル（y）の数 fntW－フォント幅、テキスト文字のピクセル数はハードウェアにしたがって設定されねばならない。 data－データバスが接続されたポートのアドレス cntrl－コントロールバスが接続されたポートのアドレス wr－*cntrl　port の！WR 線のビット番号 rd－*cntrl　port の！RD 線のビット番号 cd－*cntrl　port の！CD 線のビット番号 rst－*cntrl　port の！RST 線のビット番号 Display　RAM： ライブラリは有効な RAM の大きさを知らない。 ライブラリはパネルに RAM を切り分ける。完全なパネルはテキストパネルに続く 1 つのグラフィックパネルである。プログラマーは自身でどれだけのパネルを持てるかを知るため、そのハードウェアを知らねばならない。
必要事項	なし。
例	`T6963C_init(240, 128, 8, &PORTF, &PORTD, 5, 7, 6, 4) ;` /* *　ディスプレイを 240 ピクセル幅で 128 ピクセル高さに初期化する。 *　文字幅 8 ビット *　PORTF がデータバス *　bit5 が!WR *　bit7 が!RD *　bit6 が!CD *　bit4 が RST */

T6963C_writeData

原形	**void** T6963C_writeData(**unsigned char** data)
戻り値	なし。
解説	T6963C コントローラへ data を書き込むルーチン
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	T6963C_writeData(AddrL)

T6963C_writeCommand

原形	**void** T6963C_writeCommand (**unsigned char** data)
戻り値	なし。
解説	T6963C コントローラへ command を書き込むルーチン
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	T6963C_writeCommand (T6963C_CURSOL_POINTER_SET) ;

T6963C_setPtr

原形	**void** T6963C_setPtr (**unsigned int** addr, **unsigned int** t)
戻り値	なし。
解説	このルーチンはコマンド c のためのメモリポインタ p を設定する。
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	T6963C_writeCommand (T6963C_CURSOL_POINTER_SET) ;

T6963C_waitReady

原形	**void** T6963C_waitReady (**void**)
戻り値	なし。
解説	このルーチンはステータス　バイトを利用し、レディとなるまでループする。
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	T6963C_ waitReady ( ) ;

T6963C_fill

原形	**void** T6963C_fill(**unsigned char** data, **unsigned int** start, **unsigned int** len);
戻り値	なし。
解説	このルーチンは開始アドレスからコントローラメモリへバイト長だけデータで埋める。
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	T6963C_ fill (0x33, 0x00FF, 0x000F) ;

T6963C_dot

原形	**void** T6963C_dot(**int** x, **int** y, **unsigned char** color)
戻り値	なし。
解説	このルーチンはワークパネルに現在のテキストを設定する。それはストリングを x 列、y 行に書き込む。Mode=T6963C_ROM_MODE_［OR｜EXOR｜AND］
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	T6963C_dot (x0, y0, pcolor) ;

T6963C_write_char

原形	**void** T6963C_write_char (**char** c, **int** x, **int** y, **unsigned char** mode)
戻り値	なし。
解説	このルーチンはワークパネルに現在のテキストを設定する。 それは x 列 y 行に char c を書き込む。 mode=T6963C_ROM_MODE_［OR｜EXOR｜AND］
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	T6963C_dot ('A', 22, 23,AND) ;

T6963C_write_text

原形	**void** T6963C_write_text (**unsigned char** *str, **unsigned char** x, **unsigned char** y, **unsigned char** mode)
戻り値	なし。
解説	このルーチンはワークパネルに現在のテキストを設定する。 それは x 列 y 行にストリング str を書き込む。 mode=T6963C_ROM_MODE_［OR｜EXOR｜AND］
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	T6963C_write_text(" GLCD LIBRARY DEMO, WELCOME !", 0, 0, T6963C_ROM_MODE_XOR);

T6963C_line

原形	`void T6963C_line(int px0, int py0, int px1, int py1, unsigned char pcolor);`
戻り値	なし。
解説	このルーチンはワークパネルに現在のグラフィックを設定する。 それは(x0,y0)から(x1,y1)へ一行描画する。 mode=T6963C_［WHITE｜［BLACK］
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	`T6963C_line(0, 0, 239, 127, T6963C_WHITE);`

T6963C_rectangle

原形	`void T6963C_rectangle(int x0, int y0, int x1, int y1, unsigned char pcolor);`
戻り値	なし。
解説	このルーチンはワークパネルに現在のグラフィックを設定する。 それは(x0,y0)から(x1,y1)へ長方形の枠を描画する。 pcolor=T6963C_［WHITE｜［BLACK］
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	`T6963C_rectangle(20, 20, 219, 107, T6963C_WHITE);`

T6963C_box

原形	`void T6963C_box(int x0, int y0, int x1, int y1, unsigned char pcolor);`
戻り値	なし。
解説	このルーチンはワークパネルに現在のグラフィックを設定する。 それは(x0,y0)から(x1,y1)へソリッドボックスを描画する。 pcolor=T6963C_［WHITE｜［BLACK］
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	`T6963C_box(0, 119, 239, 127, T6963C_WHITE);`

T6963C_circle

原形	`void T6963C_circle(int x, int y, long r, unsigned char pcolor);`
戻り値	なし。
解説	このルーチンはワークパネルに現在のグラフィックを設定する。 それは中心(x, y)、半径 r の円を描画する。 pcolor=T6963C_［WHITE｜［BLACK］
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	`T6963C_circle(120, 64, 110, T6963C_WHITE);`

T6963C_image

原形	`void T6963C_image(const char *pic);`
戻り値	なし。
解説	このルーチンはワークパネルに現在のグラフィックを設定する。 それはMCU によりピクチャーポインタでグラフィックエリアを満たす。 MCU はディスプレイの形状を最適化しなければならない。 例：240 x 128 ディスプレイに対し、MCU は(240/8)*128=3840 バイトの配列を有する必要がある。
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	`T6963C_image(mc);`

T6963C_sprite

原形	`void T6963C_sprite(unsigned char px, unsigned char py, const char *pic, unsigned char sx, unsigned char sy);`
戻り値	なし。
解説	このルーチンはワークパネルに現在のグラフィックを設定する。 それは、MCU によって指示された画像である領域(px,py)-(px+sx,py+sy)にグラフィックの長方形を描画する。 sx,sy は画像の大きさでなければならない。 MCU は sx*sy バイトの配列を持たねばならない。
必要事項	ポートが初期化されねばならない。T6963C_init を見よ。
例	`T6963C_sprite(76, 4, einstein, 88, 119); // draw a sprite`

T6963C_set_cursol

原形	`void T6963C_set_cursor(unsigned char x, unsigned char y);`
戻り値	なし。
解説	このルーチンはx列y行へカーソルを設定する。
必要事項	ポートが初期化されねばならない。T6963C_initを見よ。
例	`T6963C_set_cursor(cposx, cposy);`

T6963C_clearBit

原形	`void T6963C_clearBit(char b);`
戻り値	なし。
解説	コントロールビットをクリアする。
必要事項	ポートが初期化されねばならない。T6963C_initを見よ。
例	`T6963C_clearBit(b);`

T6963C_setBit

原形	`void T6963C_setBit(char b);`
戻り値	なし。
解説	コントロールビットをセットする。
必要事項	ポートが初期化されねばならない。T6963C_initを見よ。
例	`T6963C_setBit(b);`

T6963C_negBit

原形	`void T6963C_negBit(char b);`
戻り値	なし。
解説	コントロールビットを無効にする。
必要事項	ポートが初期化されねばならない。T6963C_initを見よ。
例	`T6963C_negBit(b);`

T6963C_displayGrPanel

原形	`void T6963C_displayGrPanel(unsigned int n);`
戻り値	なし。
解説	グラフィックパネル番号 n を表示する。
必要事項	GLCD は初期化されねばならない。T6963C_init を見よ。
例	`T6963C_displayGrPanel(n);`

T6963C_displayTxtPanel

原形	`void T6963C_displayTxtPanel(unsigned int n);`
戻り値	なし。
解説	テキストパネル番号 n を表示する。
必要事項	GLCD は初期化されねばならない。T6963C_init を見よ。
例	`T6963C_displayTxtPanel(n);`

T6963C_setGrPanel

原形	`void T6963C_setGrPanel(unsigned int n);`
戻り値	なし。
解説	パネル番号 n に対するグラフィック開始アドレスを計算する。
必要事項	ポートは初期化されねばならない。T6963C_init を見よ。
例	`T6963C_setGrPanel(n);`

T6963C_setTxtPanel

原形	`void T6963C_setTxtPanel(unsigned int n);`
戻り値	なし。
解説	パネル番号 n に対するテキスト開始アドレスを計算する。
必要事項	ポートは初期化されねばならない。T6963C_init を見よ。
例	`T6963C_setTxtPanel(n);`

T6963C_panelFill

原形	`void T6963C_panelFill(unsigned int v);`
戻り値	なし。
解説	#n のパネルをビットマップ v で満たす。（クリアするには0）
必要事項	ポートは初期化されねばならない。T6963C_init を見よ。
例	`T6963C_panelFill(v);`

T6963C_grFill

原形	`void T6963C_grFill(unsigned int v);`
戻り値	なし。
解説	グラフィック#n のパネルをビットマップ v で満たす。（クリアするには0）
必要事項	ポートは初期化されねばならない。T6963C_init を見よ。
例	T6963C_grFill(v)

T6963C_txtFill

原形	`void T6963C_txtFill(unsigned int v);`
戻り値	なし。
解説	テキスト#n のパネルを **char**　v+32 で満たす。（クリアするには0）
必要事項	ポートは初期化されねばならない。T6963C_init を見よ。
例	`T6963C_txtFill(v);`

T6963C_cursol_height

原形	`void T6963C_cursor_height(unsigned int n);`
戻り値	なし。
解説	カーソルのサイズを設定する。
必要事項	ポートは初期化されねばならない。T6963C_init を見よ。
例	`T6963C_cursor_height(n);`

T6963C_graphics

原形	`void T6963C_graphics(unsigned int n);`
戻り値	なし。
解説	グラフィックスをオン／オフする。
必要事項	GLCD は初期化されねばならない。T6963C_init を見よ。
例	`T6963C_graphics(1);`

T6963C_text

原形	`void T6963C_text(unsigned int n);`
戻り値	なし。
解説	テキストをオン／オフする。
必要事項	GLCD は初期化されねばならない。T6963C_init を見よ。
例	`T6963C_text(1);`

T6963C_cursol

原形	`void T6963C_cursor(unsigned int n);`
戻り値	なし。
解説	カーソルをオン／オフする。
必要事項	ポートは初期化されねばならない。T6963C_init を見よ。
例	`T6963C_cursor(1);`

T6963C_cursol_blink

原形	`void T6963C_cursor_blink(unsigned int n);`
戻り値	なし。
解説	カーソルブリンクをオン／オフする。
必要事項	ポートは初期化されねばならない。T6963C_init を見よ。
例	`T6963C_cursor_blink(0);`

T6963C_Init_240x128

原形	`procedure T6963C_Init_240x128;`
戻り値	なし。
解説	デフォルト設定する GLCD（240x128 ピクセル）を基本とした T6963C を初期化する。
必要事項	なし。
例	`T6963C_Init_240x128;`

T6963C_Init_240x64

原形	`procedure T6963C_Init_240x64;`
戻り値	なし。
解説	デフォルト設定する GLCD（240x64 ピクセル）を基本とした T6963C を初期化する。
必要事項	なし。
例	`T6963C_Init_240x64;`

Library Example

```
#include         "T6963C.h"
/*
 * bitmap pictures stored in ROM
 */
extern const char mc[] ;
extern const char einstein[] ;
/*
 * initial PWM duty cycle for contrast power supply
 */
unsigned char    PWM_duty = 200 ;

void main(void)
        {
        unsigned char    panel ;          // current panel
        unsigned int     i ;            // general purpose register
        unsigned char    curs ;           // cursor visibility
        unsigned int     cposx, cposy ;   // cursor x-y position

        TRISC = 0 ;                        // port C is output only
        PORTC = 0b00000000 ;  // chip enable, reverse on, 8x8 font


//continues...
```

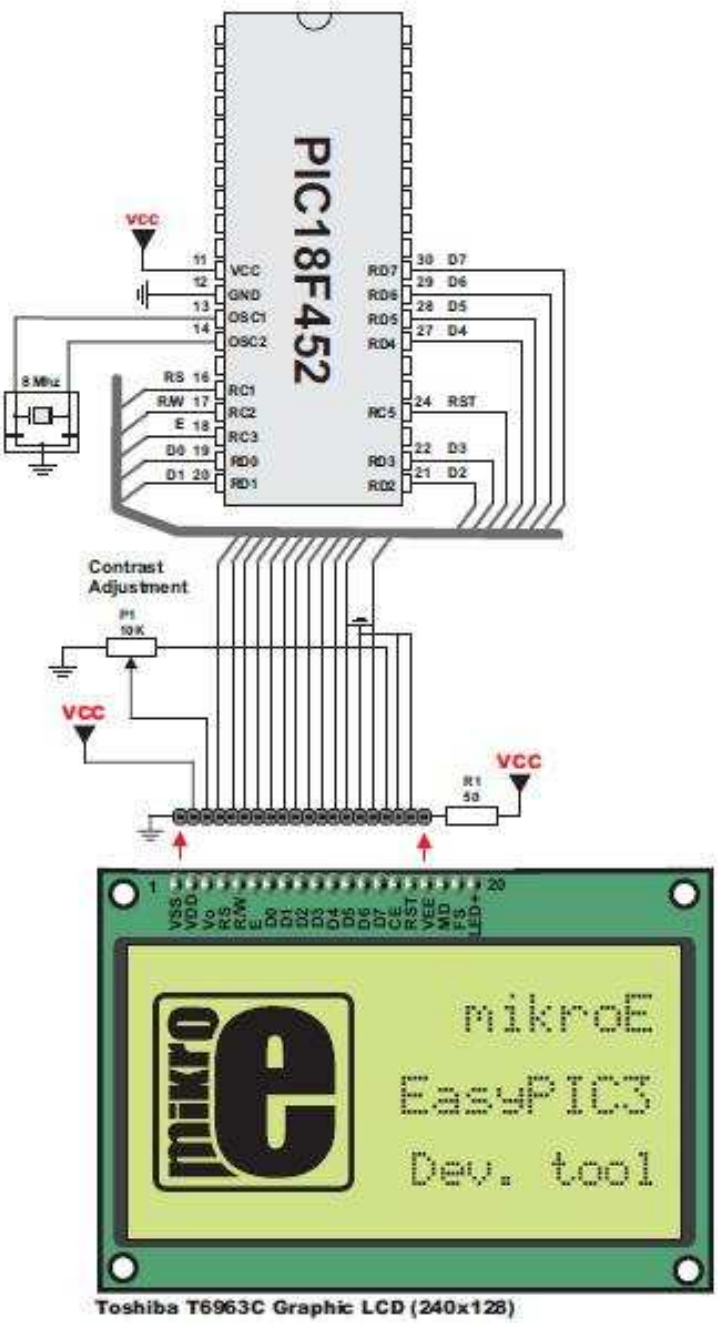
```
         * init display for 240 pixel width and 128 pixel height
         * 8 bits character width
         * data bus on PORTD
         * control bus on PORTC
         * bit 3 is !WR
         * bit 1 is !RD
         * bit 1 is C!D
         * bit 5 is RST
         */
        T6963C_Init_240x128();
        //T6963C_init(240, 128, 8, &PORTD, &PORTC, 3, 2, 1, 5) ;
        /*
         * enable both graphics and text display at the same time
         */
        T6963C_graphics(1) ;
        T6963C_text(1) ;

        panel = 0 ;
        i = 0 ;
        curs = 0 ;
        cposx = cposy = 0 ;

        /*
         * text messages
         */
T6963C_write_text(" GLCD LIBRARY DEMO, WELCOME !", 0, 0,
T6963C_ROM_MODE_XOR) ;
        T6963C_write_text(" EINSTEIN WOULD HAVE LIKED mC", 0, 15,
T6963C_ROM_MODE_XOR) ;

        /*
         * cursor
         */
        T6963C_cursor_height(8) ;        // 8 pixel height
        T6963C_set_cursor(0, 0) ;        // move cursor to top left
        T6963C_cursor(0) ;               // cursor off

        /*
         * draw rectangles
         */
        T6963C_rectangle(0, 0, 239, 127, T6963C_WHITE) ;
        T6963C_rectangle(20, 20, 219, 107, T6963C_WHITE) ;
        T6963C_rectangle(40, 40, 199, 87, T6963C_WHITE) ;
        T6963C_rectangle(60, 60, 179, 67, T6963C_WHITE) ;
```

```
        /*
         * draw a cross
         */
        T6963C_line(0, 0, 239, 127, T6963C_WHITE) ;
        T6963C_line(0, 127, 239, 0, T6963C_WHITE) ;

        /*
         * draw solid boxes
         */
        T6963C_box(0, 0, 239, 8, T6963C_WHITE) ;
        T6963C_box(0, 119, 239, 127, T6963C_WHITE) ;

        /*
         * draw circles
         */
        T6963C_circle(120, 64, 10, T6963C_WHITE) ;
        T6963C_circle(120, 64, 30, T6963C_WHITE) ;
        T6963C_circle(120, 64, 50, T6963C_WHITE) ;
        T6963C_circle(120, 64, 70, T6963C_WHITE) ;
        T6963C_circle(120, 64, 90, T6963C_WHITE) ;
        T6963C_circle(120, 64, 110, T6963C_WHITE) ;
        T6963C_circle(120, 64, 130, T6963C_WHITE) ;

        T6963C_sprite(76, 4, einstein, 88, 119) ;
// draw a sprite

        T6963C_setGrPanel(1) ;      // select other graphic panel

        T6963C_image(mc) ;
// fill the graphic screen with a picture

        for(;;)
                {

                /*
                 * if RB1 is pressed, toggle the display between
graphic panel 0 and graphic 1
                 */
                if(PORTB & 0b00000010)
                        {
                        panel++ ;
                        panel &= 1 ;
                        T6963C_displayGrPanel(panel) ;
                        Delay_ms(300) ;
                        }
                                                }
```

```
/*
  * if RB2 is pressed, display only graphic panel
  */
 else if(PORTB & 0b00000100)
         {
         T6963C_graphics(1) ;
         T6963C_text(0) ;
         Delay_ms(300) ;
         }

 /*
  * if RB3 is pressed, display only text panel
  */
 else if(PORTB & 0b00001000)
         {
         T6963C_graphics(0) ;
         T6963C_text(1) ;
         Delay_ms(300) ;
         }

 /*
  * if RB4 is pressed, display text and graphic
    panels
  */
 else if(PORTB & 0b00010000)
         {
         T6963C_graphics(1) ;
         T6963C_text(1) ;
         Delay_ms(300) ;
         }

 /*
  * if RB5 is pressed, change cursor
  */

 else if(PORTB & 0b00100000)
         {
         curs++ ;
         if(curs == 3) curs = 0 ;
         switch(curs)
```

```
        /*
         * move cursor, even if not visible
         */
        cposx++ ;
        if(cposx == T6963C_txtCols)
                {
                cposx = 0 ;
                cposy++ ;
    if(cposy == T6963C_grHeight / T6963C_CHARACTER_HEIGHT)
                        {
                        cposy = 0 ;
                        }
                }
        T6963C_set_cursor(cposx, cposy) ;

        Delay_ms(100) ;
        }
}
```

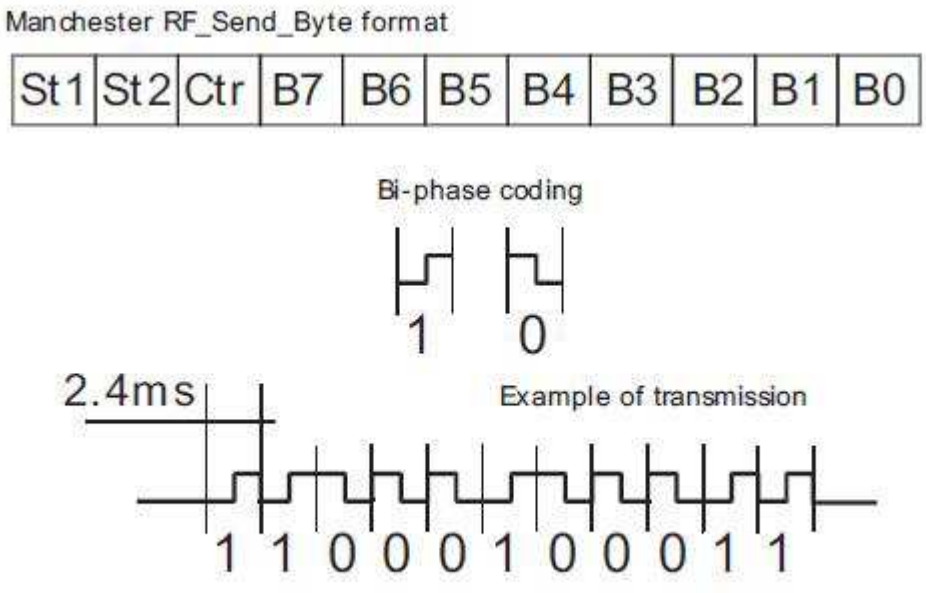
Hardware connection



Toshiba T6963C Graphic LCD (240x128)

Manchester　Code　Library（マンチェスターコードライブラリ）

ｍｉｋｒｏCはマンチェスター符号化信号を取り扱うためのライブラリを提供する。
マンチェスター符号はデータとクロック信号が１つの自己同期データストリームを形成するために組み合わされたコードである。符号化された各ビットはビット期間の中間点で転移（の意味）を内包し、転移の方向がビットが０または１なのかを決定する。後半は真のビット値で、前半は真のビット値の補数である。（以下の図に示すとおり）



注意）マンチェスタ受信ルーチンはブロッキング呼出しである（Man_Receive_Config, Man_Receive_ Init, Man_Receive_）。これはタスクが実行されるまでＰＩＣが待つことを意味する。（例えば、バイトが受信されると同期が成立するなど）受信のためのルーチンは 340～560bps のボーレートスコープに制限される。

Library Routin

Man_Receive_config
Man_Receive_Init
Man_Receive
Man_Send_Config
Man_ Send _Init
Man_ Send

Man_Receive_Config

原形	`void Man_Receive_Config(char *port, char rxpin);`
戻り値	なし
解説	関数は信号を受信するために PIC を準備する。あなたは port と信号を入力するための rxpin(0-7)を指定する必要がある。受信時に複数エラーが発生した場合、あなたは再度同期を取るべく Man_Receive_Init を呼び出さねばならない。
必要事項	なし
例	`Man_Receive_Config(&PORTD, 6);`

Man_Receive_Init

原形	`void Man_Receive_Init(char *port);`
戻り値	なし
解説	関数は信号を受信するために PIC を準備する。あなたは port を指定する必要がある。Rxpin はデフォルトで 6pin である。受信時に複数エラーが発生した場合、あなたは再度同期を取るべく Man_Receive_Init を呼び出さねばならない。
必要事項	なし
例	`Man_Receive_Init(&PORTD);`

Man_Receive

原形	`void Man_Receive(char *error);`
戻り値	信号から１バイトを返す。
解説	関数は信号から１バイト抽出する。もし信号のフォーマットが期待していたものと異なる場合、error フラグは 255 に設定されるであろう。
必要事項	この関数を使用するためには、あなたは最初に受信するための PIC を準備しなければならない。Man_Receive_Config と Man_Receive_Init を参照せよ。
例	`temp = Man_Receive(error);` `if (error) { ... /* error handling */ }`

Man_Send_Config

原形	`void Man_Send_Config(char *port, char txpin);`
戻り値	なし
解説	関数は信号を送信するため PIC を準備する。あなたは port と信号を出力するための txpin(0-7)を指定する必要がある。ボーレートは 500bps 固定である。
必要事項	なし
例	`Man_Send_Config(&PORTD, 0);`

Man_ Send _Init

原形	`Man_Send_Init(&PORTD);`
戻り値	なし
解説	関数は信号を送信するために PIC を準備する。あなたは信号を出力するための port を指定する必要がある。txpin はデフォルトで 0pin である。ボーレートは 500bps 固定である。
必要事項	なし
例	`Man_Send_Init(&PORTD);`

Man_ Send

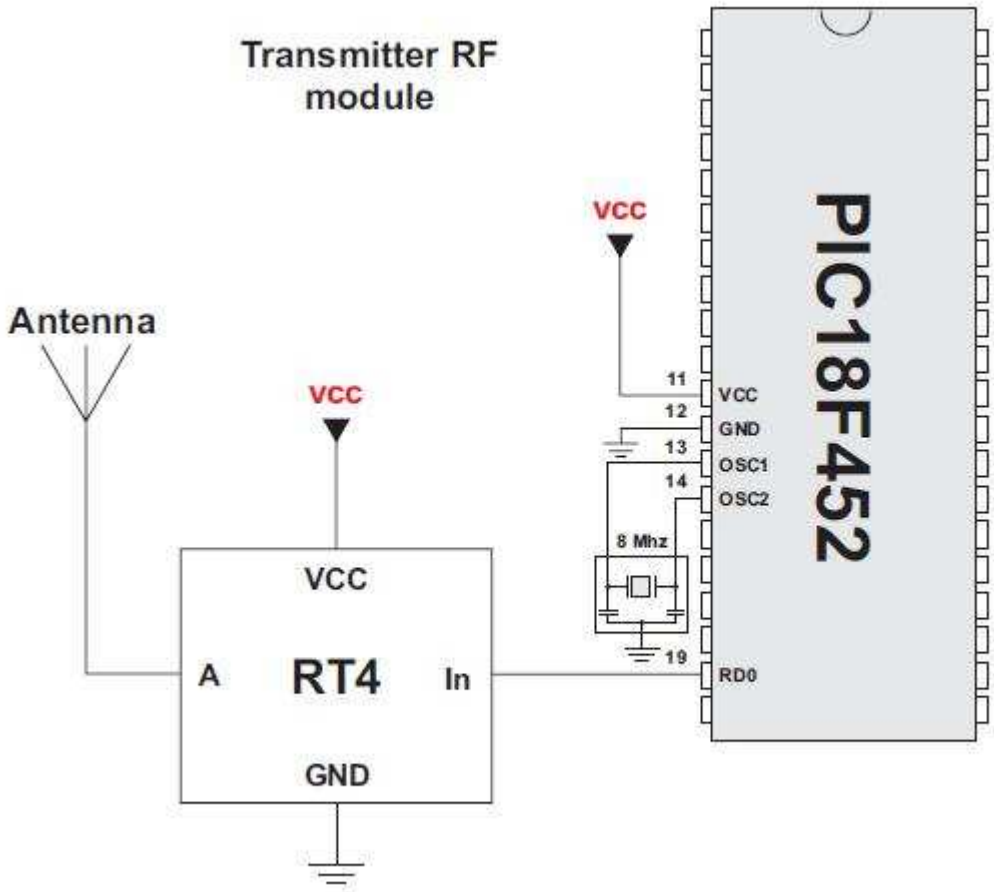
原形	`void Man_Send(unsigned short data);`
戻り値	なし
解説	1バイト（data）を送信する。
必要事項	この関数を使用するためには、あなたは最初に送信するための PIC を準備しなければならない。Man_Send_Config と Man_ Send _Init を参照せよ。

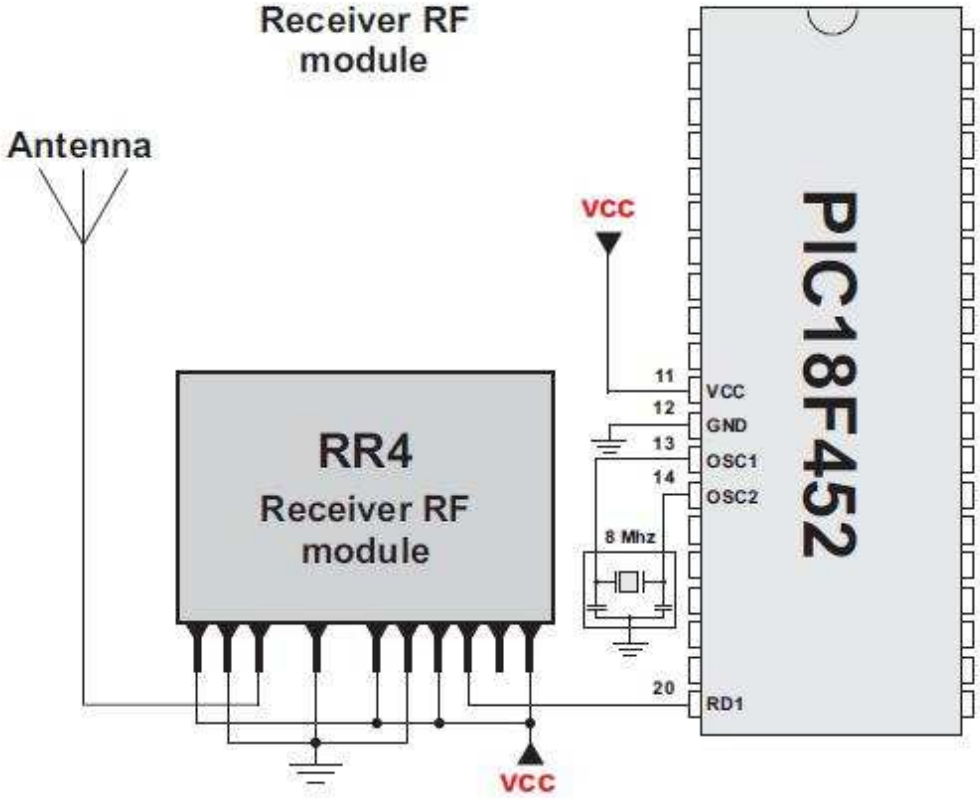
例	`unsigned short msg;`
	`...`
	`Man_Send(msg);`

Library Example

```
unsigned short error, ErrorCount, IdleCount, temp, LetterCount;

void main() {
  ErrorCount = 0;
  TRISC      = 0;                      // Error indicator
  PORTC      = 0;
  Man_Receive_Config(&PORTD, 6);       // Synchronize receiver
  Lcd_Init(&PORTB);                    // Initialize LCD on PORTB

  while (1) {                          // Endless loop
    IdleCount = 0;                     // Reset idle counter
    do {
      temp = Man_Receive(error);       // Attempt byte receive
      if (error)
        ErrorCount++
      else
        PORTC = 0;
      if (ErrorCount > 20) {           // If there are too many errors
        ErrorCount = 0;                //   syncronize the receiver again
        PORTC = 0xAA;                  // Indicate error
        Man_Receive_Init(&PORTD);      // Synchronize receiver
      }
      IdleCount++;
      if (IdleCount > 18) {            // If nothing received after some time
        IdleCount = 0;                 //   try to synchronize again
        Man_Receive_Init(&PORTD);      // Synchronize receiver
        }
    } while (temp != 0x0B);            // End of message marker
      if (error != 255) {              // If no error then write the message
        Lcd_Cmd(LCD_CLEAR);
        LetterCount = 0;
        while (LetterCount < 17) {     // Message is 16 chars long
          LetterCount++;
          temp = Man_Receive(error);
          if (error != 255)
            Lcd_Chr_Cp(temp)
          else {
            ErrorCount++; break;
          }
        }
        temp = Man_Receive(error);
        if (temp != 0x0E)
          ErrorCount++;
      } // end if
    } // end while
}//~!
```

Hardware Connection




Receiver RF
module
Antenna
VCC
PIC18F452
11
VCC
12
GND
13
OSC1
14
OSC2
8 Mhz
RR4
Receiver RF
module
20
RD1
VCC