Multimedia　Card　Library　（マルチメディアカードライブラリ）

　ｍｉｋｒｏＣはＳＰＩ通信経由でマルチメディアカード上のデータをアクセスするためのライブラリを提供する。このライブラリはＳｅｃｕｒｅ　Ｄｉｇｉｔａｌ（ＳＤ）フラッシュメモリカードスタンダードもサポートする。

注意）
－ライブラリはＰＩＣ１８ファミリのみで動作する。
－ライブラリ関数はルートディレクトリのみからファイルを作成、読み込みする。
－ライブラリ関数は、ファイルへ書き込みする場合、ＦＡＴ１とＦＡＴ２テーブルにデータを入れる。しかし、ファイルデータはFAT１テーブルからのみ読み込まれる。即ち、ＦＡＴ１テーブルが壊れている場合、リカバー（回復）しない。
－バージョン 5.0.0.3 以降、ライブラリは、セクタ０にマスターブートレコード（MBR）を持つメディアをうまく処理することが可能である。ライブラリはそれから必要な情報を読み込み、最初に有効な基本の論理パーティションへジャンプする。MBR、物理及び論理ドライブ、プライマリ／セカンダリパーティションとパーティションテーブルの更なる情報は、他のリソース、例えばウィキペディアや同種のものを調べること。

注意）
Spi_Init_Advanced（MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLR_LOW, LOW2_HIGH)は Mmc_Init を初期化する前に呼び出されねばならない。

Library　Routines

```
Mmc_Init
Mmc_Read_Sector
Mmc_Write_Sector
Mmc_Read_Cid
Mmc_Read_Csd

Mmc_Fat_Init
Mmc_Fat_Assign
Mmc_Fat_Reset
Mmc_Fat_Rewrite
Mmc_Fat_Append
Mmc_Fat_Read
Mmc_Fat_Write
Mmc_Set_File_Date
Mmc_Fat_Delete
Mmc_Fat_Get_File_Date
Mmc_Fat_Get_File_Size
Mmc_Fat_Get_Swap_File
```

Mmc_Init

原形	`unsigned short Mmc_Init(char *port, char pin);`
戻り値	もし読み込みが成功したら 0 を返し、もしエラーが発生したら 1 を返す。
解説	パラメータ Port と pin により与えられるチップセレクト端子を持つ MMC を初期化する。 通信ポートと端子は個々の MCU から設定するハードウェア SPI により指示される。もし MMC カードが現存し、首尾よく初期化されれば 0 を返し、さもなくば 1 を返す。 このライブラリの他の関数を試用する前に、Mmc_Init は呼び出されねばならない。
必要事項	`Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH);` これらは Mmc_Init を呼び出す前に呼び出されねばならない。
例	`Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH);` `while (Mmc_Init(&PORTC,2)) ;` // MMC が初期化されるまでループする

Mmc_Read_Sector

原形	`unsigned short Mmc_Read_Sector(unsigned long sector, char *data);`
戻り値	もし読み込みが成功したら 0 を返し、もしエラーが発生したら 1 を返す。
解説	関数は MMC カードのセクタアドレス sector から 1 セクタ（512 バイト）を読み込む。 読み込みデータが配列 data に保存される。もし読み込みが首尾よくいったなら 0 を返し、エラーが発生したら 1 を返す。
必要事項	ライブラリは初期化されねばならない。Mmc_Init を見よ。
例	`error = Mmc_Read_Sector(sector, data);`

Mmc_Write_Sector

原形	`unsigned short Mmc_Write_Sector(unsigned long sector,char *data);`
戻り値	もし書込みが成功したら 0 を返し、もし書込みコマンドの送信でエラーがあった場合は 1 を返す。 もし書込み中にエラーが出た場合は 2 を返す。
解説	関数は MMC カードのセクタアドレス sector へ 512 バイトを書き込む。 もし書き込みが首尾よくいったなら 0 を返し、書込みコマンド送信中にエラーが発生したら 1 を返すか、もし書込み中にエラーが出た場合は 2 を返す。
必要事項	ライブラリは初期化されねばならない。Mmc_Init を見よ。
例	`error = Mmc_Write_Sector(sector, data);`

Mmc_Read_Cid

原形	`unsigned short Mmc_Read_Cid(unsigned short *data_for_registers);`
戻り値	もし読み込みが成功したら 0 を返し、もしエラーが発生したら 1 を返す。
解説	関数は data_for_registers に１６バイトの内容を返す。
必要事項	ライブラリは初期化されねばならない。Mmc_Init を見よ。
例	`error = Mmc_Read_Cid(data);`

Mmc_Read_Csd

原形	`unsigned short Mmc_Read_Csd(unsigned short *data_for_registers);`
戻り値	もし読み込みが成功したら 0 を返し、もしエラーが発生したら 1 を返す。
解説	関数は CSD レジスタを読み、data_for_registers に１６バイトの内容を返す。
必要事項	ライブラリは初期化されねばならない。Mmc_Init を見よ。
例	`error = Mmc_Read_Csd(data);`

Mmc_Fat_Init

原形	`unsigned short Mmc_Fat_Init(unsigned short *port, unsigned short pin);`
戻り値	初期化が成功すれば 0 を返し、ブートセクタが見つからない場合は 1 を返し、カードが検出されねば 255 を返す。
解説	FAT ルーチンのために MMC/SD カードを初期化する。通信のための CS 線は変数 port と pin を通して与えられる。 MMC　FAT ライブラリの他の関数を使用する前に、この関数は呼び出されねばならない。
必要事項	`Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH);` これらは Mmc_Fat_Init を呼び出す前に呼び出されねばならない。

例	`Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH);`
	// MMC FAT が RC2 で初期化されるまでループする。
	`while (Mmc_Fat_Init(&PORTC, 2)) ;`

Mmc_Fat_Assign

原形	`void Mmc_Fat_Assign(char *filename);`
戻り値	なし
解説	このルーチンは、われわれが作業するであろうファイル（“割り当て”）を指定する。関数はルートディレィトリ内で filename により指定されたファイルを探す。もしファイルがなければ、ルーチンは開始セクタ、サイズ等を得る事でそれを初期化するだろう。もしファイルが無ければ、空のファイルが供与の名前で作られるだろう。Filename は大文字で 8+3 文字でなければならない。
必要事項	ライブラリは初期化されねばならない。Mmc_Init を見よ。

例	// MMC のルートディレクトリ内にファイル“EXAMPLE1.TXT”を割り当てる
	// もしファイルが発見されねば、ルーチンは 1 を生成するだろう。
	`Mmc_Fat_Assign("EXAMPLE1TXT");`

Mmc_Fat_Reset

原形	`void Mmc_Fat_Reset(unsigned long *size);`
戻り値	なし
解説	ファイルを読み込み可能にするために、関数は割り当てられたファイルのファイルポインタ（ファイルの開始点へそれを移動する）をリセットする。 変数 size は割り当てられたファイルのサイズをバイトで保存する。
必要事項	ライブラリは初期化されねばならない。Mmc_Fat_Init を見よ。
例	`Mmc_Fat_Reset(&filesize);`

Mmc_Fat_Rewrite

原形	`void Mmc_Fat_Rewrite(void);`
戻り値	なし
解説	新しいデータをファイルに書き込むことができるように、関数はファイルポインタをリセットし、割り当てられたファイルをクリアする。
必要事項	ライブラリは初期化されねばならない。Mmc_Fat_Init を見よ。
例	`Mmc_Fat_Rewrite();`

Mmc_Fat_Append

原形	`void Mmc_Fat_Append(void);`
戻り値	なし
解説	データをファイルに追加できるように、関数は割り当てられたファイルの終わりへファイルポインタ移動する。
必要事項	ライブラリは初期化されねばならない。Mmc_Fat_Init を見よ。
例	`Mmc_Fat_Append();`

Mmc_Fat_Read

原形	`void Mmc_Fat_Read(unsigned short *data);`
戻り値	なし
解説	関数はファイルポインタが指す場所のバイトデータを読み込み、変数 data にデータを保存する。ファイルポインタは、Mmc_Fat_Read の呼出し毎に自動的に＋１される。
必要事項	ファイルポインタは初期化されねばならない。Mmc_Fat_Reset を見よ。
例	`Mmc_Fat_Read(&mydata);`

Mmc_Fat_Write

原形	`void Mmc_Fat_Write(char *fdata, unsigned data_len);`
戻り値	なし
解説	関数は data_len バイトのひとかたまりのデータをファイルポインタの位置で現在割り当てられたファイルへ書き込む。
必要事項	ファイルポインタは初期化されねばならない。Mmc_Fat_Append または Mmc_Fat_Rewrite を見よ。
例	`Mmc_Fat_Write(txt, 21);` `Mmc_Fat_Write("Hello\nworld", 1);`

Mmc_Set_File_Date

原形	`void Mmc_Set_File_Date(unsigned year, char month, char day, char hours, char min, char sec);`
戻り値	なし
解説	関数は data_len バイトのひとかたまりのデータをファイルポインタの位置で現在割り当てられたファイルへ書き込む。
必要事項	ファイルポインタは初期化されねばならない。Mmc_Fat_Append または Mmc_Fat_Rewrite を見よ。
例	`// April 1st 2005, 18:07:00` `Mmc_Set_File_Date(2005, 4, 1, 18, 7, 0);`

Mmc_Fat_Delete

原形	`void Mmc_Fat_Delete();`
戻り値	なし
解説	MMC からファイルを削除する。
必要事項	ポートは MMC　FAT 操作のため初期化されねばならない。 Mmc_Fat_Init を参照せよ。ファイルは指定されねばならない。Mmc_Fat_Assign を参照せよ。
例	`Mmc_Fat_Delete;`

Mmc_Fat_Get_File_Date

原形	`void Mmc_fat_Get_File_Date(unsigned int *year, unsigned short *month, unsigned short *day, unsigned short *hours, unsigned short *mins);`
戻り値	なし
解説	ファイルの時間属性を読込む。あなたはファイルの年、月、日、時、分、秒を読出すことができる。
必要事項	ポートは MMC　FAT 操作のため初期化されねばならない。 Mmc_Fat_Init を参照せよ。 ファイルは指定されねばならない。 Mmc_Fat_Assign を参照せよ。
例	`Mmc_Fat_Get_File_Date(year, month, day, hours, mins);`

Mmc_Fat_Get_File_Size

原形	`unsigned long Mmc_fat_Get_File_Size();`
戻り値	なし
解説	ファイルの時間属性を読込む。あなたはファイルの年、月、日、時、分、秒を読出すことができる。
必要事項	ポートは MMC　FAT 操作のため初期化されねばならない。 Mmc_Fat_Init を参照せよ。 ファイルは指定されねばならない。 Mmc_Fat_Assign を参照せよ。
例	`Mmc_Fat_Get_File_Size;`

Mmc_Fat_Get_Swap_File

原形	`unsigned long Mmc_Fat_Get_Swap_File(unsigned long sectors_cnt);`
戻り値	新たに作られたスワップファイル用の開始セクタの No.
解説	この関数は MMC/SD メディアにスワップファイルを生成するために使われる。 それは、ユーザーがスワップファイルを持ちたいと望む連続セクタの数を引数 sectora_cnt として認める。 その実行中に、関数は有効な連続セクタ、引数 sector_cnt により指定されるそれらの数を調査する。 もしメディアにそのような空きがあるならば、MIKROSWP.SYS と名付けられたスワップファイルが生成され、その空きは FAT テーブル内でそれに指定される。 このファイルの属性は、他のファイルとそれを明確にするためシステム、アーカイブ（書庫化）、非表示となる。 もし MIKROSWP.SYS と名付けられたファイルがメディアに存在する場合、関数は新たな MIKROSWP.SYS を生成するのでそれを削除する。 スワップファイルの目的は、Mmc_Read_Sector()や Mmc_Write_Sector()関数を直接使用することにより、FAT システムに潜在的に損傷を与えることなく、可能な限り早く MMC/SD メディアに対し読み書きすることにある。 スワップファイルはユーザーが彼（彼女）が望むような方法で自由にデータを書込み／読込みすることができるメディアの“ウィンドゥ（窓）”として考えられている。 MikroC のライブラリにおけるその主たる目的は、高速なデータ取得のために使用されることにある。時間的に厳しいデータ取得が終了したとき、データは“通常の”ファイルに再書込みされ、最適な方法でフォーマットされる。
必要事項	ポートは MMC　FAT 操作のため初期化されねばならない。 Mmc_Fat_Init を参照せよ。

例	// サイズが少なくとも 1000 セクタはあるであろうスワップファイルを生成しようとする
	// もしそれが成功すれば、USART へ開始セクタの No.を送信する。
	`void M_Create_Swap_File() {`
	`  size = Mmc_Fat_Get_Swap_File(1000);`
	`  if (size) {`
	`  Usart_Write(0xAA);`
	`  Usart_Write(Lo(size));`
	`  Usart_Write(Hi(size));`
	`  Usart_Write(Higher(size));`
	`  Usart_Write(Highest(size));`
	`  Usart_Write(0xAA);`
	`  }`
	`} //~`

Library Example

次のコードはMMC ライブラリルーチンを試験する。第一にわれわれは512 の文字“M”でバッファを満たし、セクタ 56 へそれを書込む。それからわれわれはセクタ 56 で連続文字“E”を繰り返す。最後にわれわれは書込みが成功したかどうか確認するためにセクタ 55,56 を読込む。

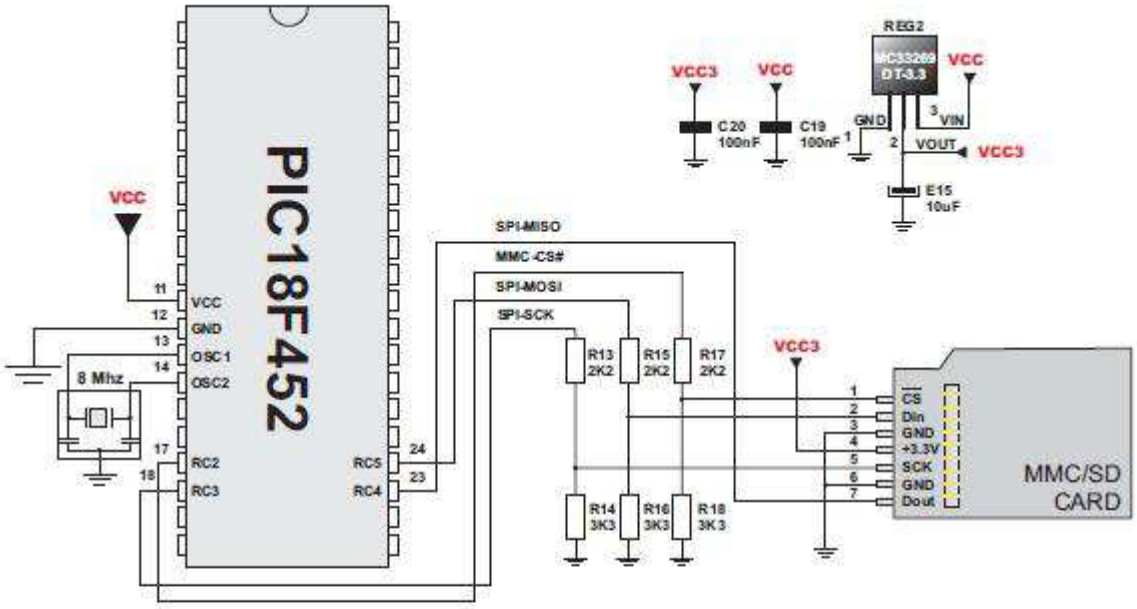
```
unsigned i;
unsigned short tmp;
unsigned short data[ 512] ;

void main() {

  Usart_Init(9600);

Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW,LOW_2_HIGH);
  // Initialize SPI

  // Wait until MMC is initialized
  while (Mmc_Init(&PORTC, 2)) ;

  // Fill the buffer with the 'M' character
  for (i = 0; i <= 511; i++) data[ i] = "M";

  // Write it to MMC card, sector 55
  tmp = Mmc_Write_Sector(55, data);

  // Fill the buffer with the 'E' character
  for (i = 0; i <= 511; i++) data[ i] = "E";

  // Write it to MMC card, sector 56
  tmp = Mmc_Write_Sector(56, data);

  // Read from sector 55
  tmp = Mmc_Read_Sector(55, data);

  // Send 512 bytes from buffer to USART
  if (tmp == 0)
    for (i = 0; i < 512; i++) Usart_Write(data[ i] );

  // Read from sector 56
  tmp = Mmc_Read_Sector(56, data);

  // Send 512 bytes from buffer to USART

  if (tmp == 0)
    for (i = 0; i < 512; i++) Usart_Write(data[ i] );

} //~!
```

Library Example

次のプログラムはMMC FATルーチンの試験をする。それはMMCカードのルートに５つの異なったファイルを生成し、それらを有るデータで満たす。あなたは異なっているべきファイルデータを検査することが可能である。

```
unsigned i;
unsigned short tmp;
unsigned short data[ 512] ;

void main() {

  Usart_Init(9600);

Spi_Init_Advanced(MASTER_OSC_DIV16, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW,LOW_2_HIGH);
  // Initialize SPI

  // Wait until MMC is initialized
  while (Mmc_Init(&PORTC, 2)) ;

  // Fill the buffer with the 'M' character
  for (i = 0; i <= 511; i++) data[i] = "M";

  // Write it to MMC card, sector 55
  tmp = Mmc_Write_Sector(55, data);

  // Fill the buffer with the 'E' character
  for (i = 0; i <= 511; i++) data[i] = "E";

  // Write it to MMC card, sector 56
  tmp = Mmc_Write_Sector(56, data);

  // Read from sector 55
  tmp = Mmc_Read_Sector(55, data);

  // Send 512 bytes from buffer to USART
  if (tmp == 0)
    for (i = 0; i < 512; i++) Usart_Write(data[i]);

  // Read from sector 56
  tmp = Mmc_Read_Sector(56, data);

  // Send 512 bytes from buffer to USART

  if (tmp == 0)
    for (i = 0; i < 512; i++) Usart_Write(data[i]);

} //~!
```

Hardware Connection



OneWire　Library

　ワンワイヤライブラリは、ワンワイヤバス経由での通信のためのルーチンを提供する。例えば、デジタル温度計DS1820である。
これはマスター／スレーブ　プロトコルであり、要求されるすべての結線は１本線である。
それが使用するハードウェア　コンフィグレーションにより、（１つのプルアップ及びオープンコレクタ駆動）
その線からそれらの電力供給を得ることも、スレーブに対し可能である。

このプロトコルのいくつかの基本的な特徴は次の通り。

－１マスターシステム
－低価格
－低転送レート（16Kbpsまで）
－適度な長距離（300mまで）
－少量データ転送パッケージ

各ワンワイヤデバイスはまた独自の64bit登録番号を有する。(8bitデバイスタイプは48bitシリアル番号と8bitCRC)
故に、複数スレーブは同一バス上に共存可能である。

　発振器の周波数Foscは、ダラスデジタル温度計用のルーチンに使用するためには、少なくとも４MHｚでなければならないことに注意すること。

Library　Routines

Ow_Reset
Ow_Read
Ow_Write

Ow_Reset

原形	**char** Ow_Reset（**char** *port, **char** pin）
戻り値	DS1820 が存在するならば 0 を、無いならば 1 を返す
解説	DS1820 に対するリセット信号をワンワイヤに出力する。変数 port と pin は DS1820 の配置を指定する。
必要事項	ダラス DS1820 温度センサのみで動作する
例	Ow_Reset (&PORTA,5)　//　RA5 ピンに接続した DS1820 をリセットする。

Ow_Read

原形	**char** Ow_Read（**char** *port, **char** pin）
戻り値	ワンワイヤバスを通じ外部デバイスから読み込んだデータ
解説	ワンワイヤバス経由で 1 バイトデータを読み込む
必要事項	なし
例	tmp = Ow_Read (&PORTA,5)

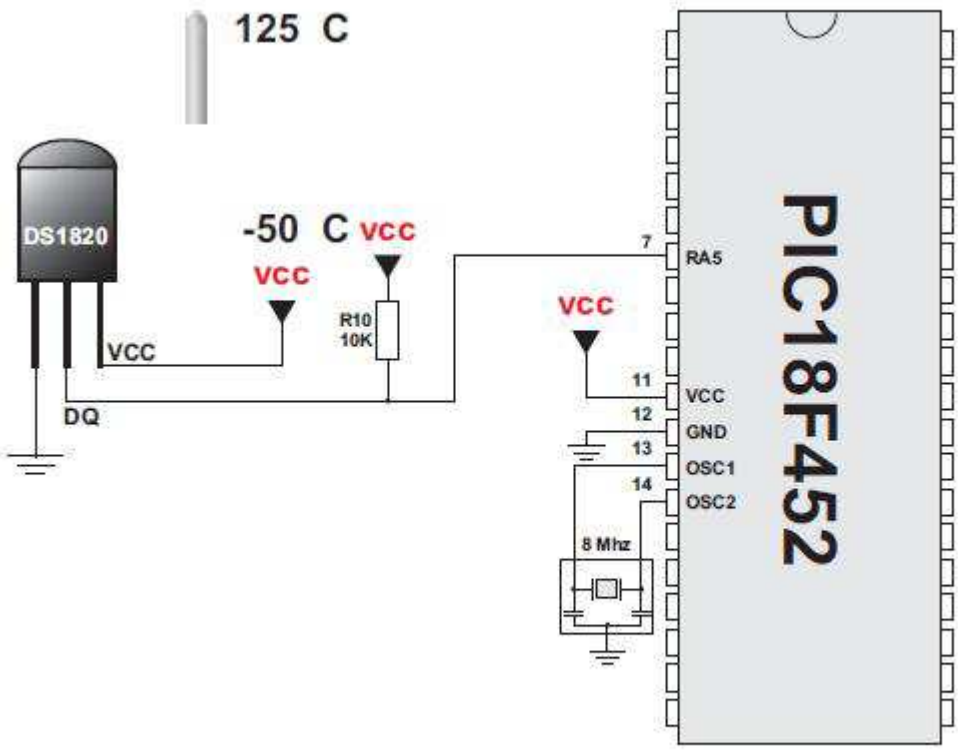
Ow_Write

原形	**char** Ow_Write（**char** *port, **char** pin, **char** par）
戻り値	なし
解説	ワンワイヤバス経由で 1 バイトデータ（引数 par）を書込む
必要事項	なし
例	Ow_Write (&PORTA,5,0xcc)

Library Example

```
unsigned temp;
unsigned short  j;

void Display_Temperature(unsigned int temp) {
  //...
}

void main() {
  ADCON1 = 0xFF;                  // Configure RA5 pin as digital I/O
  PORTA  = 0xFF;
  TRISA  = 0x0F;                  // PORTA is input
  PORTB  = 0;
  TRISB  = 0;                     // PORTB is output

  // Initialize LCD on PORTB and prepare for output

  do {

    OW_Reset(&PORTA,5);           // Onewire reset signal
    OW_Write(&PORTA,5,0xCC);      // Issue command SKIP_ROM
    OW_Write(&PORTA,5,0x44);      // Issue command CONVERT_T
    Delay_us(120);

    OW_Reset(&PORTA,5);
    OW_Write(&PORTA,5,0xCC);      // Issue command SKIP_ROM
    OW_Write(&PORTA,5,0xBE);      // Issue command READ_SCRATCHPAD
    Delay_ms(400);

    j = OW_Read(&PORTA,5);        // Get temperature LSB
    temp = OW_Read(&PORTA,5);     // Get temperature MSB
    temp <<= 8; temp += j;        // Form the result
    Display_Temperature(temp);    // Format and display result on LCD
    Delay_ms(500);

  } while (1);


}//~!
```

Hardware Connection



PS/2 Library

MikroC は、通常の PS/2 キーボードとの通信のためのライブラリを提供する。ライブラリはデータ検索のために割り込みを利用しない。６MH z およびそれ以上の発振クロックを必要とする。

Library Routines

Ps2_Init
Ps2_Config
Ps2_Key_Read

Ps2_Init

原形	void Ps2_Init（**unsigned short** *port）
解説	PS/2 キーボードで動作するためにデフォルトでの端子設定で port を初期化する。ポート端子０はデータ線でポート端子１はクロック線である。 あなたは、Ps/2 ライブラリの他のルーチンを使用する前に、Ps2_Init または Ps2_Config のいずれかを呼び出す必要がある。
必要事項	２つのデータ線とクロック線はプルアップモードにする必要がある。

Ps2_Config

原形	Void Ps2_Config（**char** *port, **char** clock, **char** data）
解説	PS/2 キーボードおよび共通端子設定で使用するため port を初期化する。変数 data、clock はデータ線とクロック線をそれぞれポートの端子に正しく指定する。データとクロックは０－７の範囲でなければならならず、また同じピンを指定することはできない。 あなたは、PS/2 ライブラリの他のルーチンを使用する前に、Ps2_Init または Ps2_Config のいづれかを呼び出す必要がある。
必要事項	２つのデータ線とクロック線はプルアップモードとする必要がある。
例	Ps2_Config (&PORTB,2,3)

Ps2_Key_Read

原形	char Ps2_Key_Read（**char** *value, **char** *special, **char** *pressed）
戻り値	キーボードからのキーの読み込みが成功したら 1 を、そうでなければ 0 を返す。
解説	この関数はキー挿下有無についての情報を調べる。 変数 value は挿下キーの値を保持する。文字、数値、括弧、スペースに対し、適切な ASCII 値が保存されるであろう。ルーチンは Shift、Caps lock の機能を認識し、正しく実行する。 変数 special は特殊なファンクションキー（F1、Enter、Esc など）のためのフラグである。もし押されたキーがこれらの内の 1 つなら、special が 1 にセットされ、そうでなければ 0 になる。 変数 pressed はキーが押されると 1 にセットされ、離すと 0 になる。
必要事項	PS/2 キーボードは初期化される必要がある。Ps2_Init または Ps2_Config を見よ。

例	`// Press Enter to continue:`
	`do {`
	`  if (Ps2_Key_Read(&value, &special, &pressed)) {`
	`  if ((value == 13) && (special == 1)) break;`
	`  }`
	`} while (1);`

Library Example

```
unsigned short keydata, special, down;

void main() {
  CMCON  = 0x07;    // Disable analog comparators (comment this for PIC18)
  INTCON = 0;       // Disable all interrupts
  Ps2_Init(&PORTA); // Init PS/2 Keyboard on PORTA
  Delay_ms(100);    // Wait for keyboard to finish

  do {
    if (Ps2_Key_Read(&keydata, &special, &down)) {
      if (down && (keydata == 16)) { // Backspace
        // ...do something with a backspace...
      }
      else if (down && (keydata == 13)) { // Enter
        Usart_Write(13);
      }
      else if (down && !special && keydata) {
        Usart_Write(keydata);
      }
    }
    Delay_ms(10);     // debounce
  } while (1);
} //~!
```

PWM　Library（ＰWMライブラリ）

　ＣＣＰモジュールは多くのＰＩＣｍｉｃｏｓで有効である。ｍｉｋｒｏCはPWM　HW（ハードウェア）モジュールを使用することで簡略化できるライブラリを提供する。

注意）P18F8520 のように２つまたはそれ以上の CCP モジュールを有するいくつかの 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。簡単に Pwm に１または２の番号を付加すること。例えば、Pwm2_Start（ ）のように。また、以前のコンパイラのバージョンとの後方互換性およびより容易なコード管理のため、複数の PWM モジュールを有する MCU は、PWM1 と同じ PWM ライブラリを有する。（即ち、あなたは CCP1 を初期化するために PWM1_Init()の代わりに PWM__Init()を使用可能である。）

Library　Routine

Pwm_Init
Pwm_Change_Duty
Pwm_Start
Pwm_Stop

Pwm_Init

原形	void　Pwm_Init (long　freq)
戻り値	なし
解説	デューティ比 0 で PWM モジュールを初期化する。変数 freq は必要な PWM 周波数 Hz（Fosc に正しい値を設定するためデバイスデータシートを参照すること）である。 PWM ライブラリから他の関数を使用する前に、Pwm_Init は呼び出されねばならない。
必要事項	あなたはこのライブラリを使用するために CCP モジュールを必要とする。mikroC インストーレーションフォルダ、サブフォルダ “Examples” を、お互いの解決のために確認すること。
例	Pwm_Init(5000) ; //　PWM モジュールを５KH z で初期化する。

Pwm_Change_Duty

原形	void　Pwm_Change_Duty (**char**　duty_ratio)
戻り値	なし
解説	PWM デューティ比を変える。変数 duty_ratio は 0～255 の値を取る。0 は 0%、127 は 50%、255 は 100%のデューティ比である。デューティ比に関するその他の指定値は （パーセント*255）／100 で与えられる。
必要事項	あなたはこのライブラリを使用するため PORTC 上の CCP モジュールを必要とする。
例	Pwm_Change_Duty (192) ; //　デューティ比を７５%に設定する。

Pwm_Start

原形	**void**　Pwm_Start (**void**)
戻り値	なし
解説	PWM 動作開始
必要事項	あなたはこのライブラリを使用するため PORTC 上の CCP モジュールを必要とする。この関数を使用するため、モジュールは初期化されねばならない。Pwm_Init を見よ。
例	Pwm_Start ( )

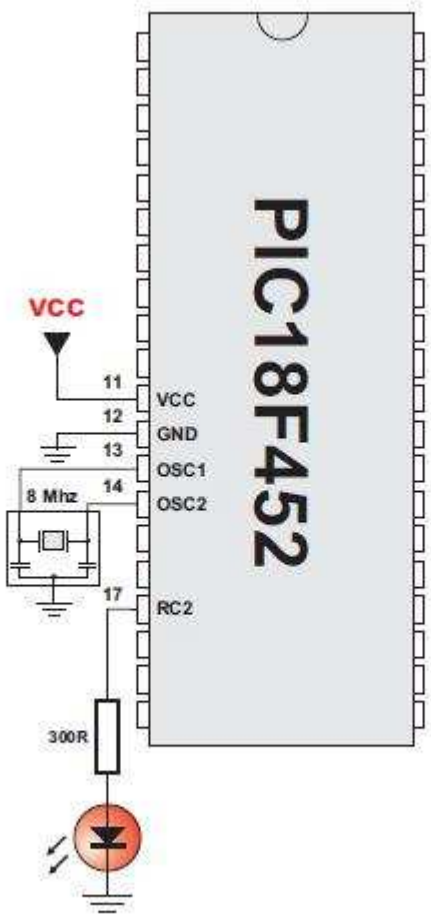
Pwm_Stop

原形	**void**　Pwm_ Stop (**void**)
戻り値	なし
解説	PWM 動作停止
必要事項	あなたはこのライブラリを使用するため PORTC の CCP モジュールを必要とする。この関数を使用するため、モジュールは初期化されねばならない。Pwm_Init を見よ。
例	Pwm_ Stop ( )

Library Example

```
char i = 0, j = 0;

void main() {
  PORTC = 0xFF;              // PORTC is output
  Pwm_Init(5000);            // Initialize PWM module at 5KHz
  Pwm_Start();               // Start PWM

  while (1) {
    // Slow down, allow us to see the change on LED:
    for (i = 0; i < 20; i++) Delay_us(500);
    j++;
    Pwm_Change_Duty(j);      // Change duty ratio
  }
}
```

Hardware Connection



RS485　Library（RS485 ライブラリ）

RS485 は複数デバイスを単線信号ケーブルに接続することを可能にするマルチポイント通信である。
mkroC はマスター／スレーブアーキテクチャを用いた RS-485 システムで最適動作を提供するためのライブラリルーチン一式を提供する。
マスターとスレーブデバイスは、同期バイト、CRC バイト、アドレスバイト、データを含むこれらのパケット毎に情報パケットをやり取りする。
各スレーブはその独自のアドレスを持ち、それにアドレス指定されたパケットのみ受信する。
スレーブは通信を始めることは決してない。１つのデバイスのみが４８５バス経由で一度に送信することを保証する事は、プログラマーの責任である。

RS485 ルーチンは PORTC 上に USART モジュールを要求する。ＵＳＡＲＴの端子は LTC485 または同等の RS-485 インターフェーストランシーバーに接続する必要がある。
トランシーバーの端子（レシーバー出力イネーブルとドライバ出力イネーブル）は PORTC の２ピンに接続すべきである。（この章の終わりの図を確認せよ。）

注意 : アドレス５０は全スレーブに対し共通アドレスである。（アドレス５０を含むパケットが全スレーブで受信されるであろう。）唯一の例外は、パケット内で指定されるべき特殊なアドレスを要求するアドレス 150 と 169 をもつスレーブである。

注意 : Usart_Init()は RS485 を初期化する前に呼び出されねばならない。

Library　Routine

```
RS485Master_Init
RS485Master_Receive
RS485Master_Send
RS485Slave_Init
RS485Slave_Receive
RS485Slave_Send
```

RS485Master_Init

原形	`void Rs485master_Init(unsigned short * port, unsigned short pin);`
戻り値	なし
解説	RS-485 通信でマスタとして PIC MCU を初期化する。
必要事項	USART　HW モジュールは初期化されねばならない。USART_Init を参照せよ。
例	`RS485Master_Init(PORTC, 2);`

RS485Master_Receive

原形	`void RS485Master_Receive(char *data);`
戻り値	なし
解説	スレーブから送られた何らかのメッセージを受信する。メッセージはマルチバイトである。そのためこの関数は受信した各バイト毎に呼出されねばならない。（この章の終わりの例を参照せよ。） メッセージを受信するまで、バッファは次の値で満たされている。 Data[0..2]はメッセージ Data[3]は受信した 1-3 のメッセージバイトの数 Data[4]はメッセージが受信されると 255 をセットする Data[5]はエラーが発生すると 255 をセットする Data[6]はメッセージを送信したスレーブのアドレス 関数は自動的に data[4]と data[5]を全受信メッセージに合わせる。 これらのフラグはプログラムからクリアする必要がある。
必要事項	MCU は、アドレスを割り当てるため RS-485 通信のマスタとして初期化されねばならない。RS485Master_Init を参照せよ。

例	`unsigned short msg[8];`
	`...`
	`RS485Master_Receive(msg);`

RS485Master_Send

原形	`void RS485Master_Send(char *data, char datalen, char address);`
戻り値	なし
解説	MCU は RS-485 経由で address により指定されたスレーブへバッファから data を送信する。 datalen はメッセージ（１≦datalen≦３）のバイト数である。
必要事項	MCU はアドレスを割り当てるため RS-485 通信のマスタとして初期化されねばならない。RS485Master_Init を参照せよ。 １デバイスのみが４８５バス経由で一度にデータを送信することを保証することは、プログラマの責任である。

例	`unsigned short msg[ 8];`
	`...`
	`RS485Master_Send(msg, 3, 0x12);`

RS485Slave_Init

原形	`void Rs485slave_Init(unsigned short * port, unsigned short pin, char address);`
戻り値	なし
解説	RS-485 通信で指定されたアドレスでスレーブとして MCU を初期化する。 スレーブ address は、全スレーブに共通のアドレス 50 を除く 0～255 のいづれかの値をとる。
必要事項	USART　HW モジュールが初期化されねばならない。USAT_Init を参照せよ。、
例	`RS485Slave_Init(PORTC, 2 ,160);` //　アドレス 160 でスレーブとして MCU を初期化する。

RS485Slave_Receive

原形	`void RS485Slave_Receive(char *data);`
戻り値	なし
解説	それにアドレスされたメッセージを受信する。メッセージはマルチバイトであり、故にこの関数は受信されたバイト毎に呼び出されねばならない。（この章の終わりの例を見よ。）メッセージを受信するまで、バッファは次の値で満たされている。 Data[0..2]はメッセージ Data[3]は受信した 1-3 のメッセージバイトの数 Data[4]はメッセージが受信されると 255 をセットする Data[5]はエラーが発生すると 255 をセットする Data[6]はメッセージを送信したスレーブのアドレス 関数は自動的に data[4]と data[5]を全受信メッセージに合わせる。 これらのフラグはプログラムからクリアする必要がある。
必要事項	MCU は、アドレスを割り当てるため RS-485 通信のスレーブとして初期化されねばならない。RS485Slave_Init を参照せよ。

例	`unsigned short msg[ 8];`
	`...`
	`RS485Slave_Send(msg, 2);`

RS485Slave_Send

原形	`void RS485Slave_Send(char *data, char datalen);`
戻り値	なし
解説	RS-485 経由でバッファからマスタへ data を送信する。 datalen はメッセージ（１≦datalen≦３）のバイト数である。
必要事項	MCU はアドレスを割り当てるため RS-485 通信のスレーブとして初期化されねばならない。RS485Slave_Init を参照せよ。 １デバイスのみが４８５バス経由で一度にデータを送信することを保証することは、プログラマの責任である。

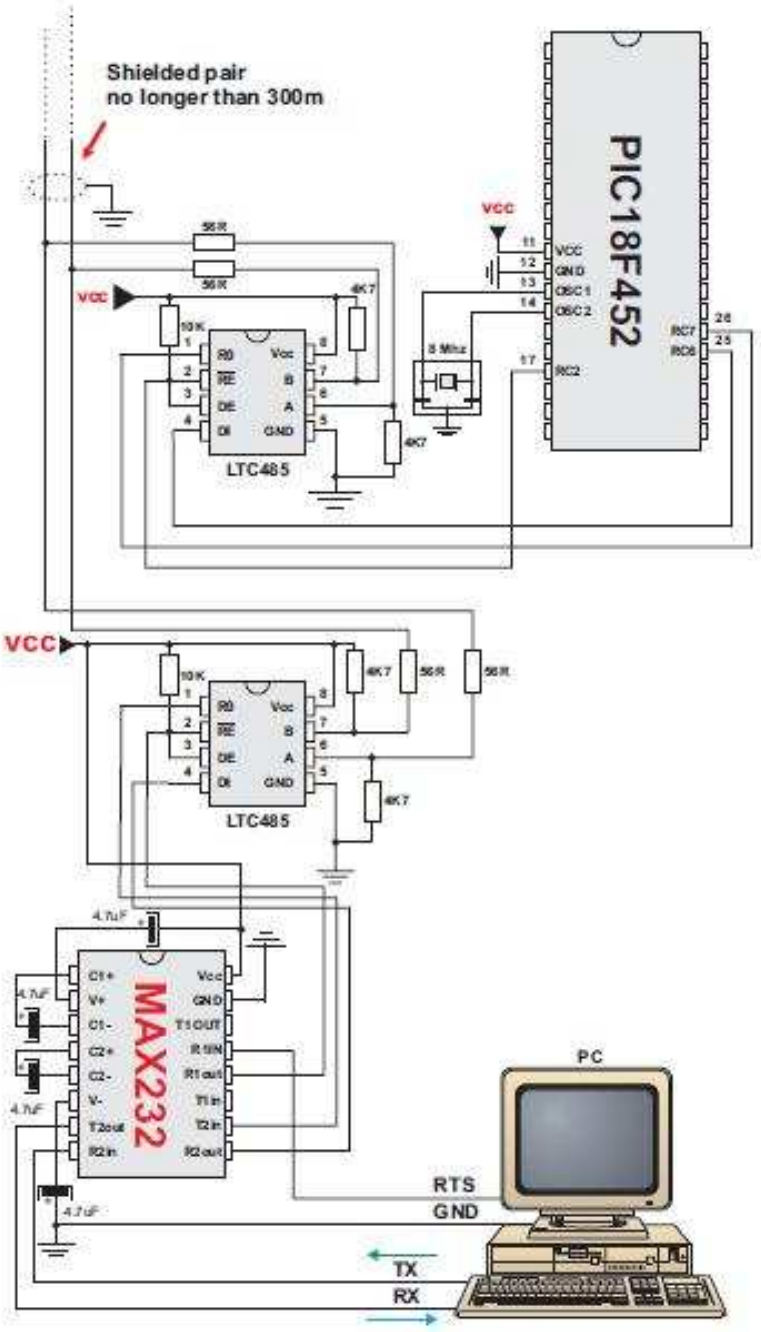
例	`unsigned short msg[ 8];`
	`...`
	`RS485Slave_Send(msg, 2);`

Library Example

この例はRS485通信でスレーブのノードとしてPICで動作することを説明するものである。PICはそれ（われわれの例ではアドレス160）へアドレス指定したパケットのみを受信し、アドレス50で汎用メッセージを受信する。この受信データはPORTBへ送られてからマスターへ返送する。

```
unsigned short dat[ 8] ;    // buffer for receiving/sending messages
char i = 0, j = 0;

void interrupt() {
/* Every byte is received by RS485Slave_Read(dat);
   If message is received without errors,
   data[4] is set to 255 */

 if (RCSTA.OERR) PORTD = 0x81;
 RS485Slave_Read(dat);
} //~

void main() {

  TRISB = 0;
  TRISD = 0;
  Usart_Init(9600);                 // Initialize usart module
  RS485Slave_Init(PORTC ,2 ,160);// Initialize MCU as Slave with address 160
  PIE1.RCIE   = 1;                  // Enable interrupt
  INTCON.PEIE = 1;                  //    on byte received
  PIE2.TXIE   = 0;                  //    via USART (RS485)
  INTCON.GIE  = 1;
  PORTB  = 0;
  PORTD  = 0;
  dat[ 4]  = 0;                     // Ensure that msg received flag is 0
  dat[ 5]  = 0;                     // Ensure that error flag is 0

  do {
    if (dat[ 5]) PORTD = 0xAA;      // If there is error, set PORTD to $AA
    if (dat[ 4]) {                  // If message received:
      dat[ 4]  = 0;                 //     Clear message received flag
      j = dat[ 3];                  //     Number of data bytes received
      for (i = 1; i < j; i++)
        PORTB = dat[ --i];          //     Output received data bytes
      dat[ 0] ++;                   //     Increment received dat[0]
      RS485Slave_Write(dat, 1);     //     Send it back to Master
    }
  } while (1);
} //~!
```

Hardware Connection



Software　I2C　Library（Ｉ２Ｃソフトウェアライブラリ）

　ｍｉｋｒｏＣは、Ｉ２Ｃソフトウェアを実行するルーチンを提供する。
これらのルーチンはハードウェアに依存せず、どのMCUでも使用可能である。
Ｉ２Ｃのソフトウェアは、Ｉ２Ｃ通信におけるマスターとしてMCUを使用するを可能にする。
マルチマスターモードはサポートされていない。

注意）このライブラリは時間依存の動作を実行する。故にＩ２Ｃのソフトを使用する場合、割り込みは禁止にする必要がある。

Library　Routines

Soft_I2C_Config
Soft_I2C_Start
Soft_I2C_Read
Soft_I2C_Write
Soft_I2C_Stop

Soft_I2C_Config

原形	`void Soft_I2C_Config(char *port, const char SDI, const char SDO, const char SCK);`
戻り値	なし
解説	I2C ソフトウェアを構築する。変数 port は、SDA と SCL 端子が割当てられる MCU のポートを指定する。変数 SCL と SDA は 0-7 の範囲でなければならない。また同じ端子を指定することはできない。 Soft_I2C_Config はソフト I2C ライブラリから他の関数を使用する前に呼び出されねばならない。
必要事項	なし
例	`Soft_I2C_Config(PORTB, 1, 2);`

Soft_I2C_Start

原形	**void** Soft_I2C_ Start (**void**)
戻り値	なし
解説	START 信号を出力する。データの送受信に先立って呼び出されねばならない。
必要事項	ソフト I2C はこの関数を使用する前に構築されねばならない。Soft_I2C_Config を見よ。
例	Soft_I2C_ Start ( )

Soft_I2C_Read

原形	**char** Soft_I2C_ Read (**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バイト読込み、変数 ack が０ならば非アクノリッジ信号を送信し、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、START 信号が出される必要がある。Soft_I2C_Start を見よ。
例	tmp = Soft_I2C_ Read (0) ; // データを読込み、非アクノリッジ信号を送信する。

Soft_I2C_Write

原形	**char** Soft_I2C_ Write (**char** data)
戻り値	エラーが無ければ 0 を返す。
解説	I2C バス経由でバイトデータ（変数 data）を送信する。
必要事項	この関数を使用するため、START 信号が出される必要がある。Soft_I2C_Start を見よ。
例	tmp = Soft_I2C_ Write (0xA3)

Soft_I2C_Stop

原形	**void** Soft_I2C_ Stop (**void**)
戻り値	なし
解説	STOP 信号を出力する。
必要事項	この関数を使用するため、START 信号が出される必要がある。Soft_I2C_Start を見よ。
例	Soft_I2C_ Stop ()

Library　Example

例はI2C ライブラリ　ソフトウェアの使用を説明するものである。PIC　MCU はRTC　ＰＣＦ８５８３（リアルタイムクロック）に接続される（SCL,SDA,端子)。プログラムはRTC へデータを送信する。

```
void main() {

  Soft_I2C_Config(&PORTD, 4,3);  // Initialize full master mode

  Soft_I2C_Start();              // Issue start signal
  Soft_I2C_Write(0xA0);          // Address PCF8583
  Soft_I2C_Write(0);             // Start from word at address 0 (config word)
  Soft_I2C_Write(0x80);          // Write 0x80 to config. (pause counter...)
  Soft_I2C_Write(0);             // Write 0 to cents word
  Soft_I2C_Write(0);             // Write 0 to seconds word
  Soft_I2C_Write(0x30);          // Write 0x30 to minutes word
  Soft_I2C_Write(0x11);          // Write 0x11 to hours word
  Soft_I2C_Write(0x30);          // Write 0x24 to year/date word
  Soft_I2C_Write(0x08);          // Write 0x08 to weekday/month
  Soft_I2C_Stop();               // Issue stop signal

  Soft_I2C_Start();              // Issue start signal
  Soft_I2C_Write(0xA0);          // Address PCF8530
  Soft_I2C_Write(0);             // Start from word at address 0
  Soft_I2C_Write(0);             // Write 0 to config word (enable counting)
  Soft_I2C_Stop();               // Issue stop signal

} //~!
```

Software　SPI　Library（SPI ソフトウェアライブラリ）

MikroC は SPI ソフトウェアを実行するライブラリを提供する。
これらのルーチンはハードウェアに依存せず、どの MCU でも使用可能である。
あなたは SPI 経由で他のデバイスと容易に通信可能である。即ち、A／DコンバーターやD／AコンバータやMAX７２１９，ＬＴＣ１２９０など。

ライブラリはマスターモード、Clock=50KHz、インターバルの中心で標本化されたデータ、クロックアイドルステートがＬｏｗ、ローからハイへのエッジ（立ち上がり）で転送されるデータ、にＳＰＩをコンフィグレーション（構築）する。

注意）これら関数は時系列の動作を実行する。、故にライブラリを使用する場合、割り込みは禁止にする必要がある。

Library Routines

Soft_Spi_Config
Soft_Spi_Read
Soft_Spi_Write

Soft_Spi_Config

原形	`void Soft_Spi_Config(char *port, const char SDI, const char SDO, const char SCK);`
戻り値	なし
解説	SPI ソフトウェアを構築し且つ、初期化する。変数 port は SDI,SDO,SCK 端子が割当てられるであろう MCU のポートを指定する。変数 SDI,SDO,SCK は 0-7 の範囲でなければならなず、また同じ端子を指定することはできない。 Soft_Spi_Config はソフト SPI ライブラリから他の関数を使用する前に呼び出されねばならない。
必要事項	なし
例	これは SPI を、クロック 50KH z 、インターバルの中心で抽出されたデータ、クロックアイドルステートロー、ローからハイへのエッジ（立ち上がり）で転送されるデータというにマスターモードに設定するであろう。SDI 端子は RB1、SDO 端子は RB2、SCK 端子は RB3 である。 `Soft_Spi_Config(PORTB, 1, 2, 3);`

Soft_Spi_Read

原形	`char Soft_Spi_Read(char buffer);`
戻り値	受信データを返す。
解説	Buffer の内容を送信することでクロックを出力し、データを受信する。
必要事項	SPI ソフトは初期されねばならない。また、この関数が使用される前に通信を確立せねばならない。Soft_Spi_Config を見よ。
例	`tmp = Soft_Spi_Read(buffer);`

Soft_Spi_Write

原形	`void Soft_Spi_Write(char data);`
戻り値	なし
解説	直ちにデータを送信する。
必要事項	SPI ソフトは初期されねばならない。また、この関数が使用される前に通信を確立せねばならない。Soft_Spi_Config を見よ。
例	`Soft_Spi_Write(1);`

Library　Examples

これはPIC　mcuのマイクロチップMCP4921の12ビットD／Aコンバータを利用を実際に説明するプログラム例である。このデバイスはデジタル入力（0～4095の数）を受け入れて、0～Vrefの範囲の出力電圧に変換する。
この例において、D/AはPORTCに接続され、SPIを通してPICと通信する。マイクロエレクトロニカのDACのレファレンス電圧は5Vである。この例では全てのDACの分解能の範囲が（12bit？4096単位）はカバーされ、それはあなたが範囲の中間から終わりまでを取得するには約7分ボタンを押す必要があるということを意味する。

Software　UART　Library（UART ソフトウェアライブラリ）

MikroC は UART ソフトウェアを実行するライブラリを提供する。
これらのルーチンはハードウェアに依存せず、どの MCU にも使用可能である。
あなたは RS232 プロトコル経由で他のデバイスと容易に通信可能である。ー以下に一覧にした関数を単純に使う。

注意）このライブラリは時間依存の動作を実行する。故に、UART ソフトを使用する場合、割り込みは禁止する必要がある。

Library Routine

Soft_Uart_Init
Soft_Uart_Read
Soft_Uart_Write

Soft_Uart_Init

原形	`void Soft_Uart_Init(unsigned short *port, unsigned short rx, unsigned short tx, unsigned short baud_rate, char inverted);`
戻り値	なし
解説	UART ソフトウェアを初期化する。変数 port は RX と TX 端子が割当てられる MCU のポートを指定する。変数 rx,tx が 0-7 の範囲であること、また同じ端子を指定することはできない。Baud_rate は必要とするボーレートである。ボーレートの最大値は PIC のクロックと動作状態に依存する。変数 inverted は、もし非ゼロ値ならば、出力を論理反転することを意味する。 Soft_Uart_Init はソフト UART ライブラリから他の関数を使用する前に呼び出されねばならない。
必要事項	なし
例	`Soft_Uart_Init(PORTB, 1, 2, 9600, 0);`

Soft_Uart_Read

原形	`unsigned short Soft_Uart_Read(unsigned short *error);`
戻り値	受信した１バイトを返す。
解説	関数は UART ソフトウェアを経由して１バイトを受信する。もし送信が成功すれば、変数 error はゼロとなるだろう。これは非ブロッキング関数呼び出しである。故にあなたは error を手作業で検査しなければならない。（以下の例を確認すること）
必要事項	UART ソフトは初期されねばならない。また、この関数が使用する前に通信を確立せねばならない。Soft_Uart_Init を見よ。

例	// データが受信されるまで保持するループである
	`do`
	`  data = Soft_Uart_Read(&error);`
	`while (error);`
	// 今、われわれはそれで作業可能である
	`if (data) {...}`

Soft_Uart_Write

原形	`void Soft_Uart_Write(char data);`
戻り値	なし
解説	関数は UART 経由で１バイト（data）を送信する。
必要事項	UART ソフトは初期されねばならない。また、この関数を使用する前に通信を確立せねばならない。Soft_Uart_Init を見よ。 送信中、UART ソフトウェアはデータの受信することができないーデータ転送プロトコルはそのような情報の喪失を防止するための方法を設定しなければならない。

例	`char some_byte = 0x0A;`
	`...`
	`Soft_Uart_Write(some_byte);`

Library Examples

この例はUART ソフトウェア経由で簡単なデータの入れ替えを説明する。
PIC MCU がデータを受信すると直ちに同じデータを送り返す。もし PIC が PC に接続されているなら（以下の図をみよ。）、あなたは RS232C 通信で mikroC ターミナルからこの例を試験することが可能である。メニューの **Tools** >**Ternimal** を選択する。

```
unsigned short data = 0, ro = 0;
unsigned short *er;

void main() {
  er = &ro;

  // Init (8 bit, 2400 baud rate, no parity bit, non-inverted logic)
  Soft_Uart_Init(PORTB, 1, 2, 2400, 0);

  do {
    do {
      data = Soft_Uart_Read(er);        // Receive data
    } while (*er);
    Soft_Uart_Write(data);              // Send data via UART
  } while (1);
} //~!
```

Sound　Library（サウンドライブラリ）

MikroC はあなたのアプリケーションにおける音源の信号化に使用可能なサウンドライブラリを提供する。あなたは、指定したポート上に簡単な圧電スピーカ（または他のハードウェア）を必要とする。

Library　Routines

Sound_Init
Sound_Play

Sound_Init

原形	`void Sound_Init(char *port, char pin);`
戻り値	なし
解説	指定したポートと端子に出力するためハードウェアを準備する。変数 pin は 0-7 の範囲になければならない。
必要事項	なし
例	`Sound_Init(PORTB, 2);  // Initialize sound on RB2`

Sound_Play

原形	`void Sound_Play(char period_div_10, unsigned num_of_periods);`
戻り値	なし
解説	指定したポートと端子で音を鳴らす（Sound_Init を見よ）。変数 period_div_10 は１０で割った MCU サイクル内の音の範囲で与えられ、指定した範囲の数（num_of_period）の間、音を生成する。
必要事項	音を聞くには、あなたは設定したポートに圧電スピーカ（又は他のハードウェア）を必要とする。また、あなたは出力のハードウェアを準備するために Sound_Init を呼ばねばならない。
例	１KH z の音を鳴らすには、T=1/f=1ms=1000cycles,@4MHz）となる。これはわれわれに最初の変数 1000/10=100 を与える。このように 150 の期間音を鳴らす。 `Sound_Play(100, 150);`

Library　Examp les

この例は、圧電スピーカに音を出すための音源ライブラリの使用方法に関する簡単なデモ（実演説明）である。コードは、PORTB と PORTA に接続した ADC を有する MCU ならどれでも使用可能である。
この例の音声周波数は ADC から値を読み込み、T（ｆ＝１／T）の元となる結果の下位バイトを使用することで生成される。

```
int adcValue;

void main() {

  PORTB  = 0;                  // Clear PORTB
  TRISB  = 0;                  // PORTB is output
  INTCON = 0;                  // Disable all interrupts
  ADCON1 = 0x82;               // Configure VDD as Vref, and analog channels
  TRISA  = 0xFF;               // PORTA is input
  Sound_Init(PORTB, 2);        // Initialize sound on RB2

  while (1) {                  // Play in loop:
    adcValue = ADC_Read(2);    //   Get lower byte from ADC
    Sound_Play(adcValue, 200); //   Play the sound
  }
}
```

SPI　Library（SPI ライブラリ）

SPI モジュールは多くの PIC　MCU の型に有効である。MikroC は、スレーブモードを初期化するため、またマスターモードで最適な動作をするためのライブラリを提供する。PIC は SPI を経由して他のデバイスと容易に通信可能である。A/D コンバータ、D/A コンバータ、MAX7219、LTC1290 など。あなたは SPI（例えば、PIC16F877）を集積したハードウェアを備えた PIC　MCU を必要とする。

注意）P18F8722 のように２つの SPI モジュールを備えたある 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。Spi に数字１または２を簡単に追加すること。例えば、Spi2_Write( )。また、以前のコンパイラ　バージョンの後方互換性およびより容易なコード管理のために、複数 SPI モジュールを備えた MCU は SPI１と同一の SPI ライブラリを有する。（即ち、あなたが SPI 操作のため SPI1_Init( )の代わりに SPI_Init( )を使用可能である。）

Library　Routines

Spi_Init
Spi_Init_Advanced
Spi_Read
Spi_Write

Spi_Init

原形	**void**　Spi_Init (**void**)
戻り値	なし
解説	デフォルト設定で SPI を構築及び初期化する。SPI_Init_Advanced または SPI_Init は、SPI ライブラリから他の関数を使用する前に呼び出される必要がある。 デフォルト設定とは、マスターモード、クロック fosc/4、クロックアイドルステート Low、ローからハイへのエッジでのデータ転送、インターバルの中心でサンプリングされるデータ、を意味する。 カスタムコンフィグレーションのため、Spi_Init_Avdanced を使用する。
必要事項	あなたは SPI を集積したハードウェアを備えた PIC　MCU を必要とする。
例	Spi_Init ( )

Spi_Init_Advanced

原形	`void Spi_Init_Advanced(char master, char data_sample, char clock_idle, char transmit_edge);`
戻り値	なし

解説	SPI を構築及び初期化する。SPI_Init_Advanced または SPI_Init は、SPI ライブラリから他の関数を使用する前に呼び出される必要がある。
	
	変数 mast_slave は SPI の動作モードを決定する。次の値を取ることができる。
	
	変数 data_sample はデータが抽出される時を決定する。次の値を取ることができる。
	
	`MASTER_OSC_DIV4 // Master clock=Fosc/4`
	`MASTER_OSC_DIV16 // Master clock=Fosc/16`
	`MASTER_OSC_DIV64 // Master clock=Fosc/64`
	`MASTER_TMR2 // Master clock source TMR2`
	`SLAVE_SS_ENABLE // Master Slave select enabled`
	`SLAVE_SS_DIS // Master Slave select disabled`
	
	data_sample はデータがサンプリングされる時を決定する。次の値を取ることができる。
	
	`CLK_IDLE_HIGH // Clock idle HIGH`
	`CLK_IDLE_LOW // Clock idle LOW`
	
	変数 clock_idle はクロックのアイドルステートを決定する。次の値を取ることができる。
	CLOCK_IDLE_HIGH
	CLOCK_IDLE_LOW
	
	変数 transmit_edge は次の値を取ることができる。
	`LOW_2_HIGH // Data transmit on low to high edge`
	`HIGH_2_LOW // Data transmit on high to low edge`

必要事項	あなたは SPI を集積したハードウェアを備えた PIC　MCU を必要とする。
例	これは PIC を、マスターモード、クロック fosc/4、インターバルの中心で抽出されるデータ、クロックアイドルステート Low、ローからハイへのエッジでのデータ転送、に設定する。 `Spi_Init_Advanced(MASTER_OSC_DIV4, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH)`

Spi_Read

原形	`char Spi_Read(char buffer);`
戻り値	受信データを返す。
解説	buffer を送信する事でクロックを提供し、期間の終わりにデータを受信する。
必要事項	SPI は初期化されねばならず、しかも、この関数を使用する前に通信を確立させねばならない。Spi_Init_Avdanced または Spi_Init を見よ。

例	`short take, buffer;`
	`...`
	`take = Spi_Read(buffer);`

Spi_Write

原形	`void Spi_Write(char data);`
戻り値	なし
解説	SSPBUF へバイトデータを書き込み且つ、直接転送を始める。
必要事項	SPI は初期化されねばならず、しかも、この関数を使用する前に通信を確立させねばならない。Spi_Init_Avdanced または Spi_Init を見よ。
例	`Spi_Write(1);`

Library Example

```
//-------------------- Function Declarations
void max7219_init1();
//--------------------------------- F.D. end

char i;

void main() {
  Spi_Init();                    // Standard configuration
  TRISC &= 0xFD;
  max7219_init1();               // Initialize  max7219
  for (i = 1; i <= 8u; i++) {
    PORTC &= 0xFD;               // Select max7219
    Spi_Write(i);                // Send i to max7219 (digit place)
    Spi_Write(8 - i);            // Send i to max7219 (digit)
    PORTC |= 2;                  // Deselect max7219
  }
  TRISB = 0;
  PORTB = i;
} //~!
```

USART　Library（ＵＳＡＲＴライブラリ）

ＵＳＡＲＴハードウェアモジュールは多くのＰＩＣｍｉｃｒｏｓに有効である。ｍｉｋｒｏＣ　ＵＳＡＲＴライブラリは非同期（全二重）モードでの快適な動作を提供する。あなたは、ＲＳ２３２プロトコル経由で他のデバイスと容易に通信可能である。（例えば、ＰＣと。トピックーＲＳ２３２HW接続の最後の図を参照せよ。）

あなたはＵＳＡＲＴを集積したハードウェアを備えた、例えば、ＰＩＣ１６Ｆ８７７－ＰＩＣ　MCUを必要とする。かくして、以下に一覧表にした関数を使用するだけでよい。

注意：ＵＳＡＲＴライブラリ関数は、ＰＯＲＴＢ、ＰＯＲＴＣ、さもなくばＰＯＲＴＧ上のモジュールをサポートし、他のポート上のモジュールで動作しないだろう。他のポート上のモジュールでのＰＩＣｍｉｃｒｏｓの例は、ｍｉｋｒｏＣインストールフォルダ内の“Ｅｘａｍｐｌｅｓ”の中に見つけることが可能である。

Library　Routines

Usart_Init
Usart_Data_Ready
Usart_Read
Usart_Write

注意：P18F8520 のように２つの USART を備えた、ある PICmicros はあなたに使用したいモジュールを指定するよう要求する。関数名に数字の１または２を追加するだけである。例えば、Usart_Write2();
以前のコンパイラバージョンやより容易なコード管理に伴う後方互換のため、複数 USART モジュールを備えた MCU は USART1 と同じ USART ライブラリを有する。（即ち、あなたは Usart 操作のため Usart_Init1 の代わりに Usart_Init を使用可能である。）

Usart_Init

原形	`void Usart_Init(const long baud_rate);`
戻り値	なし
解説	要求されたボーレートで USART モジュールのハードウェアを初期化する。指定した Fosc で許されるボーレートについてはデバイスデータシートを参照せよ。もしあなたがサポートされていないボーレートを指定すると、コンパイラはエラーを報告するだろう。 Usart_Init が USART ライブラリから他の関数を使用する前に呼び出される必要がある。
必要事項	あなたはハードウェア USART を備えた PIC　MCU を必要とする。
例	`Usart_Init(2400);  // Establish communication at 2400 bps`

Usart_Data_Ready

原形	`char Usart_Data_Ready(void);`
戻り値	関数は、データがレディならば1を、データがなければ0を返す。
解説	データが転送のためレディかどうかテストするために関数を使用する。
必要事項	USART　HWモジュールは初期化されねばならず、この関数を使用する前に通信が確立していなければならない。Usart_Initを見よ。

例	`int receive;`
	`...`
	`// If data is ready, read it:`
	`if (Usart_Data_Ready()) receive = Usart_Read;`

Usart_Read

原形	`char Usart_Read(void);`
戻り値	受信したバイトデータを返す。もしバイトデータが受信されなければ0を返す。
解説	関数はUSARTを経由して1バイトデータを受信する。データが最初にレディかどうかテストするために、関数Usart_Data_Readyを使用する。
必要事項	USART　HWモジュールは初期化されねばならず、この関数を使用する前に通信が確立していなければならない。Usart_Initを見よ。

例	`int receive;`
	`...`
	`// If data is ready, read it:`
	`if (Usart_Data_Ready()) receive = Usart_Read;`

Usart_Write

原形	`char Usart_Write(char data);`
戻り値	なし
解説	関数はUSARTを経由して1バイトデータを送信する。
必要事項	USART　HWモジュールは初期化されねばならず、この関数を使用する前に通信が確立していなければならない。Usart_Initを見よ。

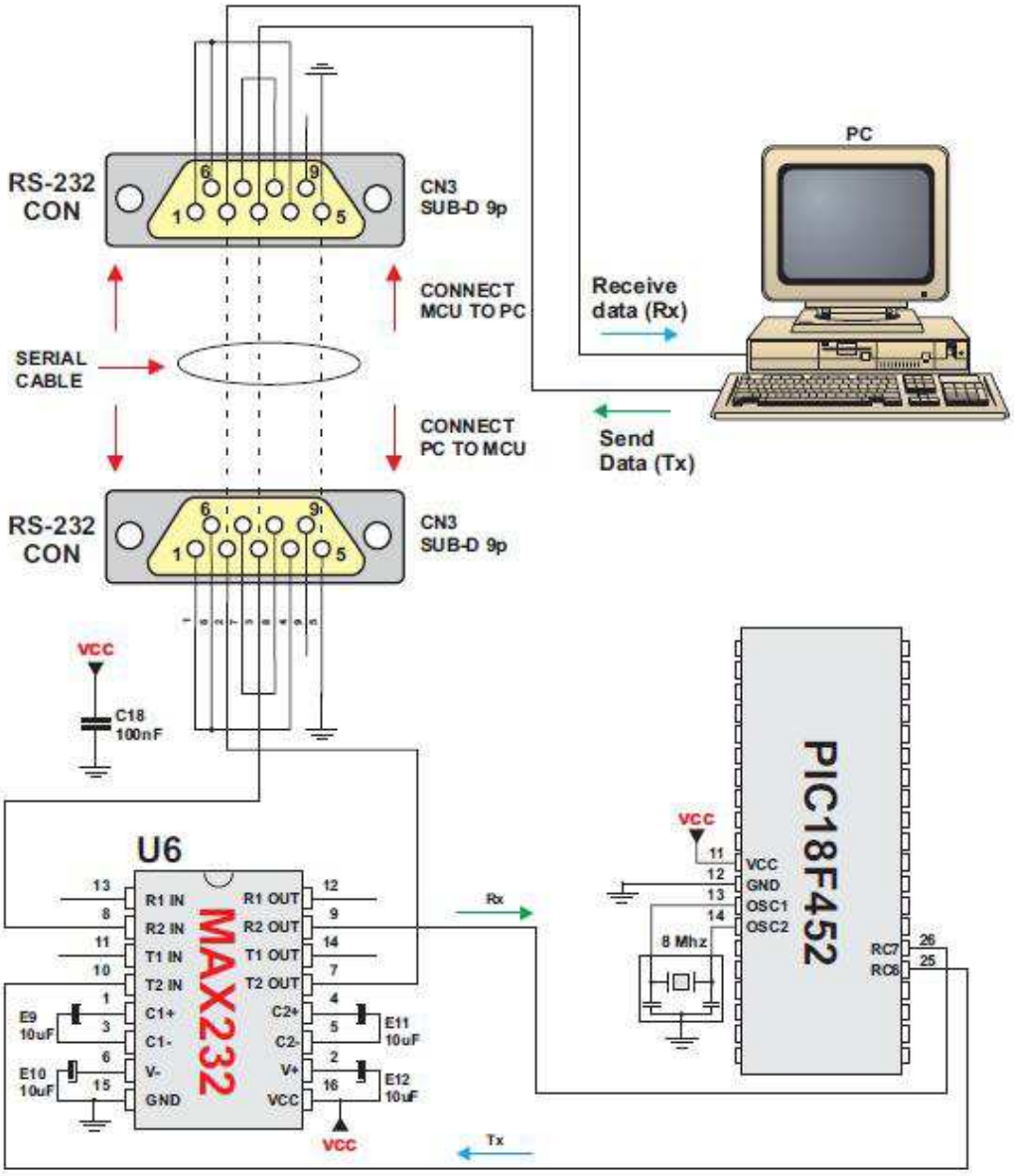
例	`int chunk;`
	`...`
	`Usart_Write(chunk);  /* send data chunk via USART */`

Library　Example

この例はUSART経由で単純なデータの交換を説明する。PIC　MCUがデータを受信すると、直ちに同じデータを返送する。もしPICがPC（以下の図を見よ）に接続されていたら、あなたはRS232通信のメニュー選択 Tool＞Terminal から mikroC ターミナルから例を試験できる。

```
unsigned short i;

void main() {

  // Initialize USART module (8 bit, 2400 baud rate, no parity bit..)
  Usart_Init(2400);

  do {
    if (Usart_Data_Ready()) {     // If data is received
      i = Usart_Read();           // Read the received data
      Usart_Write(i);             // Send data via USART
    }
  } while (1);
} //~!
```

Hardware Connection



USB　HID　Library（USB　HID　ライブラリ）

ユニバーサルシリアルバス（ＵＳＢ）は、コンピュータ、セルフォン、ゲームコンソール、ＰＤＡ（携帯用情報端末）などを含んだ、広範囲で多様なデバイスに接続するためのシリアルバス規格を提供する。

mikroC は、ユニバーサルシリアルバスを経由して、ヒューマンインターフェースデバイスで作業するためのライブラリを含む。
ヒューマンインターフェースデバイス又はＨＩＤは、キーボード、マウス、グラフィックス　タブレット（図形入力装置）や似たようなもののように、人間から入力を受け取り、直接相互に作用するコンピュータデバイスの典型である。

Library　Routines

Hid_Enable
Hid_Read
Hid_Write
Hid_Disable

Hid_Enable

原形	`void Hid_Enable(unsigned *readbuff, unsigned *writebuff);`
戻り値	なし
解説	USB HID 通信を可能にする。変数 readbuff と writebuff は読み込みバッファと書き込みバッファであり、各々、HID 通信のために使用される。 この関数は USB　HID ライブラリの他のルーチンを使用する前に呼び出される必要がある。
必要事項	USART　HW モジュールは初期化されねばならず、この関数を使用する前に通信が確立していなければならない。Usart_Init を見よ。
例	`Hid_Enable(&rd, &wr);`

Hid_Read

原形	`unsigned short Hid_Read(void);`
戻り値	ホストから受信した読込みバッファ内のキャラクタ数を返す。
解説	ホストからのメッセージを受信し、読込みバッファへそれを保存する。関数は読込みバッファ内に受信したキャラクタ数を返す。
必要事項	USB　HID はこの関数を使用する前に有効にする必要がある。Hid_Enable を見よ。
例	`get = Hid_Read();`

Hid_Write

原形	`void Hid_Write(unsigned *writebuff, unsigned short len);`
戻り値	なし
解説	関数は wrbuff からホストへデータを送る。書込みバッファは初期化において使用したものと同じ変数である。変数 len は送信されるはずのデータ長を指定すべきである。
必要事項	USB　HID はこの関数を使用する前に有効にする必要がある。Hid_Enable を見よ。
例	`Hid_Write(&wr, len);`

Hid_Disable

原形	`void Hid_Disable(void);`
戻り値	なし
解説	USB　HID 通信を禁止する。
必要事項	USB　HID はこの関数を使用する前に有効にする必要がある。Hid_Enable を見よ。
例	`Hid_Disable();`

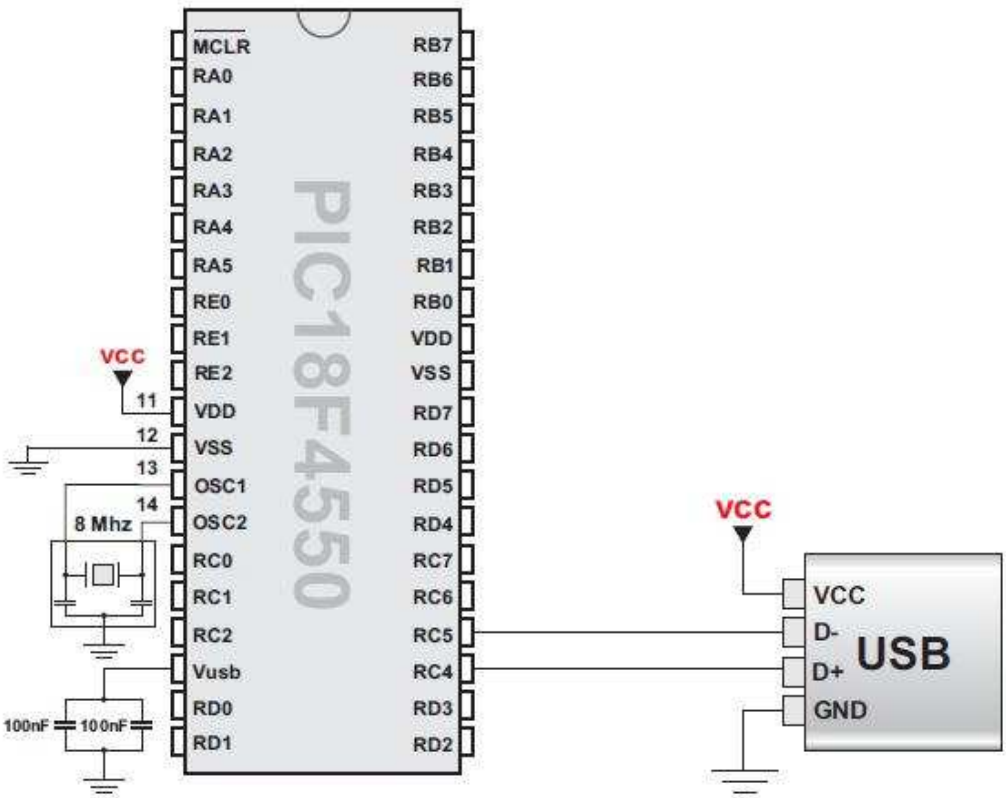
Library Example

次の例はユニバーサルシリアルバスをし経由して PC へ 0～255 の数の連続を継続的に送信する。

```
unsigned short m, k;
unsigned short userRD_buffer[ 64];
unsigned short userWR_buffer[ 64];

void interrupt() {
  asm CALL _Hid_InterruptProc
  asm nop
}//~

void Init_Main() {
  // Disable all interrupts
  // Disable GIE, PEIE, TMR0IE, INT0IE,RBIE
  INTCON = 0;
  INTCON2 = 0xF5;
  INTCON3 = 0xC0;
  // Disable Priority Levels on interrupts
  RCON.IPEN = 0;
  PIE1 = 0; PIE2 = 0; PIR1 = 0; PIR2 = 0;

  // Configure all ports with analog function as digital
  ADCON1 |= 0x0F;

  // Ports Configuration
  TRISA = 0; TRISB = 0; TRISC = 0xFF; TRISD = 0xFF; TRISE = 0x07;
  LATA = 0; LATB = 0; LATC = 0; LATD = 0; LATE = 0;

  // Clear user RAM
  // Banks [00 .. 07] ( 8 x 256 = 2048 Bytes )
  asm {
    LFSR      FSR0, 0x000
    MOVLW     0x08
    CLRF      POSTINC0, 0
    CPFSEQ    FSR0H, 0
    BRA       $ - 2
  }
```

```
  // Timer 0
  T0CON   = 0x07;
  TMR0H   = (65536-156) >> 8;
  TMR0L   = (65536-156) & 0xFF;
  INTCON.T0IE = 1;              // Enable T0IE
  T0CON.TMR0ON = 1;
}//~


/** Main Program Routine **/

void main() {
  Init_Main();
  Hid_Enable(&userRD_buffer, &userWR_buffer);

  do {
    for (k = 0; k < 255; k++) {
      // Prepare send buffer
      userWR_buffer[ 0] = k;

      // Send the number via USB
      Hid_Write(&userWR_buffer, 1);
    }
  } while (1);

  Hid_Disable();
}//~!
```

HW connection



Util　Library（ユーティリティライブラリ）

Util ライブラリはプロジェクトの開発に役立つ種々雑多なルーチンを含む。

Button

原形	`char Button(char *port, char pin, char time, char active_state);`
戻り値	0 又は 255 を返す。
解説	関数はボタンを押す時にカチカチいう接触の影響を除去する。（デバウンシング） 変数 port はボタンの位置を指定する。変数 pin は設計したポートの端子番号であり、０－７の値を取る。変数 time はマイクロセカンドのデバウンス期間である。変数 active_state は 0 又は 1 のどちらかであり、また、それはボタンが論理 0 又は論理 1 で有効かどうか決定する。
必要事項	なし

例	例は、ボタンが接続されている RB0 を読み込む。１から０への変化（ボタンが離れた）で PORTD が反転する。
	
	`do {`
	`  if (Button(&PORTB, 0, 1, 1)) oldstate = 1;`
	`  if (oldstate && Button(&PORTB, 0, 1, 0)) {`
	`  PORTD = ~PORTD;`
	`  oldstate = 0;`
	`  }`
	`} while(1);`

ANSI C Ctype Library

mikroC は文字列を試験し配置するための標準 ANSI C ライブラリ関数一式を提供する。

注意 : 標準関数の全てが含まれてきたわけではない。関数は ANSI C 標準に従い実行されてきたが、ある関数は PIC プログラミングを容易にするため変更されてきた。

Library Routines

```
isalnum
isalpha
iscntrl
isdigit
isgraph
islower
isprint
ispunct
isspace
isupper
isxdigit
toupper
tolower
```

isalnum

原形	`char isalnum(char character);`
戻り値	なし
解説	関数はもし character が英数字(A-Z,a-z,0-9)であれば 1 を、その他はゼロを返す。
必要事項	なし
例	なし

Isalpha

原形	`char isalpha(char character);`
戻り値	なし
解説	関数はもし character がアルファベット(A-Z、a-z)であれば 1 を、その他はゼロを返す。
必要事項	なし
例	なし

iscntrl

原形	`char iscntrl(char character);`
戻り値	なし
解説	関数はもし character が制御文字またはディリート（10 進数 0-31 及び 127）であれば 1 を、その他はゼロを返す。
必要事項	なし
例	なし

isdigit

原形	`char isdigit(char character);`
戻り値	なし
解説	関数はもし character が数字（0-9）であれば 1 を、その他はゼロを返す。
必要事項	なし
例	なし

isgraph

原形	`char isgraph(char character);`
戻り値	なし
解説	関数はもし character がスペース（10 進数 32）を除く出力可能な文字列であれば 1 を、その他はゼロを返す。
必要事項	なし
例	なし

islower

原形	`char islower(char character);`
戻り値	なし
解説	関数はもし character が小文字（a-z）であれば 1 を、その他はゼロを返す。
必要事項	なし
例	なし

isprint

原形	`char isprint(char character);`
戻り値	なし
解説	関数はもし character が出力可能（10 進数 32 から 126）ならば 1 を、その他はゼロを返す。
必要事項	なし
例	なし

ispunct

原形	`char ispunct(char character);`
戻り値	なし
解説	関数はもし character が句読点（10 進数 32-47、58-63、91-96、123-126）ならば 1 を、その他はゼロを返す。
必要事項	なし
例	なし

isspace

原形	`char isspace(char character);`
戻り値	なし
解説	関数はもし character が余白（space、CR、HT、VT、NL、FF）ならば 1 を、その他はゼロを返す。
必要事項	なし
例	なし

isupper

原形	`char isupper(char character);`
戻り値	なし
解説	関数はもしcharacterが大文字（A-Z）ならば1を、その他はゼロを返す。
必要事項	なし
例	なし

isxdigit

原形	`char isxdigit(char character);`
戻り値	なし
解説	関数はもしcharacterが16進数（0-9、A-F、a-f）ならば1を、その他はゼロを返す。
必要事項	なし
例	なし

toupper

原形	`char toupper(int character);`
戻り値	なし
解説	もしcharacterが小文字（a-z）ならば、関数は大文字を返す。そうでなければ、関数は無変化な入力変数を返す。
必要事項	なし
例	なし

tolower

原形	`char tolower(int character);`
戻り値	なし
解説	もしcharacterが大文字（A-Z）ならば、関数は小文字を返す。そうでなければ、関数は無変化な入力変数を返す。
必要事項	なし
例	なし

ANSI C Math Library

mikroC は浮動小数点演算を取り扱う標準 ANSI C ライブラリ関数一式を提供する。

注意）関数は ANSI C 標準に従って実行されてきたが、ある関数は PIC プログラミングを容易にするために変更されてきた。

Library Routines

```
acos
asin
atan
atan2
ceil
cos
cosh
exp
fabs
floor
frexp
ldexp
log
log10
modf
pow
sin
sinh
sqrt
tan
tanh
```

acos

原形	`double acos(double x);`
戻り値	なし
解説	関数は変数 x の arc cos を返す。即ち、コサインが x である値である。入力変数 x は-1～+1 でなければならない（±1 を含む）。戻り値は 0～π のラジアンである（0、π を含む）。
必要事項	なし
例	なし

asin

原形	`double asin(double x);`
戻り値	なし
解説	関数は変数 x の arc　sin を返す。即ち、サインが x である値である。入力変数 x は-1～+1 でなければならない（±1 を含む）。戻り値は－π/2～π/2 のラジアンである（－π/2, π/2 を含む）。
必要事項	なし
例	なし

atan

原形	`double atan(double x);`
戻り値	なし
解説	関数は変数 x の arc　tan を返す。即ち、タンジェントが x である値である。戻り値はラジアンで－π/2～π/2 である。
必要事項	なし
例	なし

atan2

原形	`double atan2(double y, double x);`
戻り値	なし
解説	これは arc　tan 関数の２つの引数である。それは、両引数の極性が結果として象限を決定するために使用われることを除いて、ｙ／ｘのアークタンジェントを計算することと同様であり、また、ｘはゼロを認められている。戻り値は－π/2～π/2 のラジアンである（－π/2, π/2 を含む）。
必要事項	なし
例	なし

ceil

原形	`double ceil(double num);`
戻り値	なし
解説	関数は次の全数を切り上げた変数 num の値を返す。
必要事項	なし
例	なし

cos

原形	`double cos(double x);`
戻り値	なし
解説	関数はラジアンでxのコサインを返す。戻り値は−1～+1である。
必要事項	なし
例	なし

cosh

原形	`double cosh(double x);`
戻り値	なし
解説	関数は（e ^ x+e ^ -x）/2　として数学的に定義された、xのハイパボリック・コサインを返す。もしxの値が大きすぎる場合、(オーバーフローが発生するなら)、関数は機能しない。
必要事項	なし
例	なし

exp

原形	`double exp(double x);`
戻り値	なし
解説	関数はxのべきを立て自然対数を底としたeの値を返す。
必要事項	なし
例	なし

fabs

原形	`double fabs(double num);`
戻り値	なし
解説	関数はnumの絶対値（つまり正の）を返す。
必要事項	なし
例	なし

floor

原形	`double floor(double num);`
戻り値	なし
解説	関数は最も近い整数に切り捨てした変数 num の値を返す。
必要事項	なし
例	なし

frexp

原形	`double frexp(double num, int *n);`
戻り値	なし
解説	関数は標準化した小数と２のべき乗和に浮動小数値 num を分割する。戻り値は標準化した小数であり、整数 exponent はｎで示される対象に保存される。
必要事項	なし
例	なし

ldexp

原形	`double ldexp(double num, int n);`
戻り値	なし
解説	関数は、exp のべき乗に２のべき浮動小数 num をかけた結果を返す。（ｘ＊２＾ｎ）
必要事項	なし
例	なし

log

原形	`double log(double x);`
戻り値	なし
解説	関数は、x の自然対数を返す。（すなわち、loge(x)）
必要事項	なし
例	なし

log10

原形	`double log10(double x);`
戻り値	なし
解説	関数は、x の常用対数を返す。（すなわち、log10(x)）
必要事項	なし
例	なし

modf

原形	`double modf(double num, double *whole);`
戻り値	なし
解説	関数は、num の極性有り分数成分を返し、whole で示された変数に全数成分を配置する。
必要事項	なし
例	なし

pow

原形	`double pow(double x, double y);`
戻り値	なし
解説	関数は、y をべき数に上げた x の値を返す（すなわち、x ^ y)。もし x が負ならば、関数は自動的に y を unsigned long 型に割り当てるであろう。
必要事項	なし
例	なし

sin

原形	`double sin(double x);`
戻り値	なし
解説	関数はラジアンで x のサインを返す。戻り値は−1～+1 である。
必要事項	なし
例	なし

sinh

原形	`double sinh(double x);`
戻り値	なし
解説	関数は（ｅ＾ｘ－ｅ＾-x）/2　として数学的に定義された、x のハイパボリック・サインを返す。もしｘの値が大きすぎる場合、(オーバーフローが発生するなら)、関数は機能しない。
必要事項	なし
例	なし

sqrt

原形	`double sqrt(double num);`
戻り値	なし
解説	関数は num の非負の平方根を返す。
必要事項	なし
例	なし

tan

原形	`double tan(double x);`
戻り値	なし
解説	関数はラジアンで x のタンジェントを返す。戻り値は mikroC の浮動小数演算の許容範囲内に及ぶ。
必要事項	なし
例	なし

tanh

原形	`double tanh(double x);`
戻り値	なし
解説	関数は sinh(ｘ)/cosh(x)として数学的に定義された、x のハイパボリック・タンジェントを返す。
必要事項	なし
例	なし

ANSI C Stdlib Library

mikroC は一般的なユーティリティの標準 ANSI C ライブラリ関数一式を提供する。

注意）標準関数の全てがこれまで含まれてきたわけではない。関数は ANSI C 標準に従い実行されてきたが、ある関数は PIC プログラミングを容易にするために変更されてきた。

Library Routines

```
abs
atof
atoi
atol
div
ldiv
labs
max
min
rand
srand
xtoi
```

abs

原形	`int abs(int num);`
戻り値	なし
解説	関数は num の絶対値（即ち、正数）を返す。
必要事項	なし
例	なし

Atof

原形	`double atof(char *s)`
戻り値	なし
解説	関数は倍精度値に入力文字列 s を変換し、値を返す。 入力文字列 s は、始まりに任意の余白をつけた浮動小数点数定数形式に従うべきである。関数が認識できない（ヌル文字を含む）文字に到達するまで、文字列は一度に 1 文字ずつ処理されるだろう。
必要事項	なし
例	なし

atoi

原形	`int atoi(char *s);`
戻り値	なし
解説	関数は入力文字列 s を整数値に変換し、値を返す。入力文字列 s はもっぱら、始まりに任意の余白と符号をつけた１０進数からのみ成るべきである。関数が認識できない（これはヌル文字を含む）文字にたどり着くまで、文字列は、一度に１文字ずつ処理されるだろう。
必要事項	なし
例	なし

atoll

原形	`long atol(char *s)`
戻り値	なし
解説	関数は入力文字列 s を倍長整数値に変換し、値を返す。入力文字列 s は、もっぱら最初に任意の余白と符号をつけた１０進数からのみ成るべきである。 関数は、認識できない（これはヌル文字を含む）文字に到達するまで、文字は一度に１文字ずつ処理されるであろう。
必要事項	なし
例	なし

div

原形	`div_t div(int numer, int denom);`
戻り値	なし
解説	関数は分母 denom で分子 numer の除算結果を計算する。 関数は商（quot）と余り（rem）で構成された構造体 div_t を返す。
必要事項	なし
例	なし

ldiv

原形	`ldiv_t ldiv(long numer, long denom);`
戻り値	なし
解説	引数と結果の構造体メンバ全てがlong形式を有することを除いて、関数はdivと同等である。 関数は分母denomで分子numerwの除算結果を計算する。 関数は商（quot）と余り（rem）で構成された構造体div_tを返す。
必要事項	なし
例	なし

labs

原形	`long labs(long num);`
戻り値	なし
解説	関数は、倍長整数numの絶対値（即ち、正数）を返す。
必要事項	なし
例	なし

max

原形	`int max(int a, int b);`
戻り値	なし
解説	関数は、２つの整数ａとｂのうち大きい方を返す。
必要事項	なし
例	なし

min

原形	`int min(int a, int b);`
戻り値	なし
解説	関数は、２つの整数ａとｂのうち小さい方を返す。
必要事項	なし
例	なし

rand

原形	`int rand(void);`
戻り値	なし
解説	関数は0～32767 の間の一連の擬似乱数を返す。 関数は、srand()が出発点に戻るために呼ばれない限り、常に同じ一連の数を生成するであろう。
必要事項	なし
例	なし

arand

原形	`void srand(unsigned seed);`
戻り値	なし
解説	関数は、rand() へ続く呼び出しにより戻されるべき新たな一連の擬似乱数の開始点として初期値を使用する。この関数に戻り値はない。
必要事項	なし
例	なし

xtoi

原形	`int xtoi(char *s);`
戻り値	なし
解説	関数は１６進数から成る入力文字列 s を整数値に変換する。 入力変数 s は、もっぱら最初に任意の余白と符号を持った１６進数からのみ成るべきである。関数が認識できない（これはヌル文字を含む）文字に到達するまで、文字列は１度に１文字ずつ処理されるであろう。
必要事項	なし
例	なし

ANSI C String Library

mikroC は、文字列と配列 char を操作するための標準 ANSI C ライブラリ関数一式を提供する。

注意）全ての標準関数が含まれて来たというわけではない。関数は、ANSI C 標準に従い実行されてきたが、ある関数は PIC プログラミングを容易にするために変更されてきた。

Library Routines

```
memcmp
memcpy
memmove
memset
memchr
strcat
strchr
strcmp
strcpy
strlen
strncat
strncpy
strspn
strcspn
strncmp
strpbrk
strrchr
strstr
strtok
```

memcmp

原形	`int *memcmp(void *s1, void *s2, int n);`
戻り値	なし
解説	関数は S1,S2 で指定された対象の最初の n 文字を比較し、もしその対象が等しいならゼロを返し、さもなくば、最初に異っている文字（左から右へ評価）間の違いを返す。従って、もし s1 で指定された対象が s2 で指定された対象よりも大きい場合、結果は、はゼロよりも大きい。また逆も同様。
必要事項	なし
例	なし

memcpy

原形	`void *memcpy(void *s1, void *s2, int n);`
戻り値	なし
解説	関数は、S2 で指定された対象から、S1 で指定された対象へｎ文字をコピーする。対象は重複しないこと。関数は s1 の値を返す。
必要事項	なし
例	なし

memmove

原形	`void *memmove(void *s1, void *s2, int n);`
戻り値	なし
解説	関数は、S2 で指定された対象から、S1 で指定された対象へｎ文字をコピーする。Memcpy()とは異なり、メモリ領域 s1,s2 は重なっても良い。関数は s1 の値を返す。
必要事項	なし
例	なし

memset

原形	`void *memset(void *s, int c, int n);`
戻り値	なし
解説	関数は、文字ｃ（char　に変換された）の値を、ｓで指定された対象の最初のｎ文字へコピーする。関数はｓの値を返す。
必要事項	なし
例	なし

memchr

原形	`void * memchr(void *p, unsigned int n, unsigned int v);`
戻り値	なし
解説	関数は、アドレスｐで始まるメモリ領域の最初のｎバイト内に、バイトｖの最初に出現する位置を示す。 もしｖが見つからなかった場合、関数はメモリアドレスｐまたは$FFFF からこの出現位置のオフセット値を返す。
必要事項	なし
例	なし

strcat

原形	`char *strcat(char *s1, char *s2);`
戻り値	なし
解説	関数は、文字列 s2 に文字列 s1 を付加すし、S1 の終わりでヌル文字を上書きする。 それから、末端のヌル文字が結果に追加される。文字列は重複しない、また s1 は結果を蓄えるため十分な空きがなければならない。関数は結果としての文字列 s1 を返す。
例	なし

strchr

原形	`char *strchr(char *s, char c);`
戻り値	なし
解説	関数は、文字列 s 内で最初に出現する文字 c の場所を示す。 関数は c のポインタか、 c が s 内に無ければヌルポインタを返す。 末端のヌル文字は文字列の一部となるように考慮されている。
例	なし

strcmp

原形	`char strcmp(char *s1, char *s2);`
戻り値	なし
解説	関数は文字列 s 1 と s2 を比較し、もし文字列が等しいならばゼロを返し、さもなくば最初に異なった文字（左から右へ評価）間の相違を返す。 よって結果は、s1 が s2 より大きい場合ゼロ以上を返し、逆も同様である。
例	なし

strcpy

原形	`char *strcpy(char *s1, char *s2);`
戻り値	なし
解説	関数は文字列 s 2 を文字列 s1 へコピーする。もし成功すれば、関数は s1 を返す。文字は重複してはならない。
例	なし

strlen

原形	`unsigned strlen(char *s);`
戻り値	なし
解説	関数は、文字列 s の長さを返す。（末端のヌル文字は文字長として数えない）
必要事項	なし
例	なし

strncat

原形	`char *strncat(char *s1, char *s2, int n);`
戻り値	なし
解説	関数は、文字列 s1 に s2 を n を超えない文字分付加する。s2 の最初の文字が、s1 の終わりのヌル文字に上書きされる。末端のヌル文字は常に結果に追加される。関数は s1 を返す。
例	なし

strncpy

原形	`char *strncpy(char *s1, char *s2, int n);`
戻り値	なし
解説	関数は、文字列 s2 へ s1 を n 文字以上コピーしない。文字列は重複してはならない。もし s2 が文字数 n よりも短い場合、s1 は相違を埋め合わすためにヌル文字を詰め込むであろう。 関数は結果である文字列 s1 を返す。
例	なし

strspn

原形	`int strspn(char *s1, char *s2);`
戻り値	なし
解説	関数は、s2 の文字群を構成する s1 の先頭からの長さを返す。 文字の終わりの末端のヌル文字は比較されない。
例	なし

strcspn

原形	`char strcspn(char * s1, char * s2);`
戻り値	なし
解説	この関数は、s2 で指定される文字が、s1 に含まれない長さを計算する。 関数は文字群の長さを返す。
例	なし

strncmp

原形	`int strncmp(char * s1, char * s2, char len);`
戻り値	なし
解説	関数は、辞書的に文字列 s1 と s2 の最初の n バイトを比較し、それらの関係をしめす値を返す。 値　　意味 < 0　s1 は s2 より小さい = 0　s1 は s2 に等しい > 0　s1 は s2 より大きい 関数によって返される値は、(最初の n バイト内で) 比較される文字列に違いがある、最初のバイトの組の値間の差異により決定される。
例	なし

strpbrk

原形	`char * strpbrk(char * s1, char * s2);`
戻り値	なし
解説	関数は、文字列 s2 の何らかの文字が最初に出現するまで s1 を検索する。 ヌル接尾文字は検索時に含まれない。 関数は s1 内で一致する文字の位置を返す。 もし s1 が s2 を含まない場合、 $FF を返す。
例	なし

strrchr

原形	`char * strrchr(char * ptr, unsigned int chr);`
戻り値	なし
解説	関数は、文字 chr が最後に出現するまで文字列 ptr を検索する。 ptr の末端に付いたヌル文字は検索時に含まれない。 関数は ptr 内で発見した最後の chr の位置を返す。 もし一致する文字が見つからなかった場合、 $FF を返す。
例	なし

strstr

原形	`char * strstr(char * s1, char * s2);`
戻り値	なし
解説	関数は、(末端に付いたヌル文字を除き) 文字列 s 1 内で最初に出現する文字 s2 の場所を示す。 関数は s1 内で s2 が最初に出現した位置を示す数値を返す。もし文字が見つからなければ、関数は $ FF を返す。もし s2 がヌル文字ならば、関数は 0 を返す。
例	なし

strtok

原形	`char * strtok(char * s1, char * s2);`
戻り値	strtok 関数は、トークンの最初の文字を指すポインタを返すか、トークンが無い場合はヌルポインタを返す。
解説	strtok 関数を呼び出す手順は、一連のトークン内で s1 で示される文字列で停止し、S2 によって示される文字列の 1 文字でそれぞれ区切られる。 シーケンスにおける最初の呼び出しは、最初の引数として s1 を有し、それらの最初の引数としてヌルポインタを伴う呼び出しに従う。 S2 で指示される区分文字列は各呼び出し毎に異なってもよい。 シーケンスにおける最初の呼び出しは、s2 で指示される現在の区分文字列内に含まれない最初の文字から s1 で指示される文字列を検索する。 もしそのような文字が見つからない場合、s1 で指示される文字列内にトークンが無く、strtok 関数はヌルポインタを返す。もし文字が見つかれば、それが最初のトークンの始りである。 それから、strtok 関数は現在の区分文字列を含む文字をそこから検索する。 もしそのような文字が見つからない場合、現在のトークンは s1 で指示される文字列の終わりまで (検索を) 広げる。次に続くトークンの検索はヌルポインタを返すであろう。 もしそのような文字が見つかれば、ヌル文字で上書きされ、現在のトークンが末端につけられるであろう。 strtok 関数は、トークンの次の検索が始まる続きの文字を指すポインタを保存する。 最初の引数の値としてヌルポインタを伴い次に続く呼び出しが、保存されたポインタから開始し、上述したように動作する。

例	`char x[10] ;`
	`void main(){`
	
	`  strcpy(x, strtok("mikroEl", "Ek"));`
	`  strcpy(x, strtok(0, "kE"));`
	
	`}`

Conversion　Library

mikroC 変換ライブラリは、数字を文字へ変換するためのルーチンとＢＣＤ／１０進変換のためのルーチンを提供する。

Library　Routines

あなたは次のルーチンの１つに数値を通すことにより数値の文字表示を得られる。

```
ByteToStr
ShortToStr
WordToStr
IntToStr
LongToStr
FloatToStr
```

次の関数はBCD（２進化１０進数）を１０進値に変換すし、逆も同様である。

```
Bcd2Dec
Dec2Bcd
Bcd2Dec16
Dec2Bcd16
```

ByteToStr

原形	`void ByteToStr(unsigned short number, char *output);`
戻り値	なし
解説	関数は小さな符号なし number（0x100 より小さい数値）の文字列出力 output を生成する。出力文字列は３文字幅に合わされる。（もし何かあるなら）左の残りの位置は空白で満たされる。

例	`unsigned short t = 24;`
	`char txt[4];`
	`//...`
	`ByteToStr(t, txt);  // txt is " 24" (one blank here)`

ShortToStr

原形	`void ShortToStr(short number, char *output);`
戻り値	なし
解説	関数は小さな符号付き number（0x100 より小さい数値）の文字列出力 output を生成する。 出力文字列は４文字幅に合わされる。(もし何かあるなら) 左の残りの位置は空白で満たされる。

例	`short t = -24;`
	`char txt[5];`
	`//...`
	`ByteToStr(t, txt);` // txt は“ －２４”である。(空白１個)

WordToStr

原形	`void WordToStr(unsigned number, char *output);`
戻り値	なし
解説	関数は符号無し number（unsigned 型の数値）の文字列出力 output を生成する。 出力文字列は５文字幅に合わされる。(もし何かあるなら) 左の残りの位置は空白で満たされる。

例	`unsigned t = 437;`
	`char txt[6];`
	`//...`
	`WordToStr(t, txt);` // txt は“　　４３７”(空白２個)

IntToStr

原形	`void IntToStr(int number, char *output);`
戻り値	なし
解説	関数は符号有り number（int 型の数値）の文字列出力 output を生成する。 出力文字は６文字幅に合わされる。(もし何かあるなら)左の残りの位置は空白で満たされる。

例	`int j = -4220;`
	`char txt[7];`
	`//...`
	`IntToStr(j, txt);` // txt は“ －４２２０”(空白１個)

LongToStr

原形	`void LongToStr(long number, char *output);`
戻り値	なし
解説	関数は大きな符号付き number（long 型の数値）の文字列出力 output を生成する。 出力文字列は１１文字幅に合わされる。（もし何かあるなら）左の残りの位置は空白で満たされる。

例	`long jj = -3700000;`
	`char txt[12];`
	`//...`
	`LongToStr(jj, txt);` // txt は“　-3700000”である。（空白３個）

FloatToStr

原形	`void FloatToStr(float number, char *output);`
戻り値	なし
解説	関数は浮動小数点数 number の文字列出力 output を生成する。 出力文字列は最初の位置に符号を持つ数（0 と 1 の間の仮数）の標準化されたフォーマットを含む。仮数は6桁、０．ｄｄｄｄｄの形式で表わされる。即ち常に小数点以下が５桁ある。 output 文字列は少なくとも 13 文字長で無ければならない。

例	`float ff = -374.2;`
	`char txt[13];`
	`//...`
	`FloatToStr(ff, txt);` // txt は“-0.37420e3”

Bcd2Dec

原形	`unsigned short Bcd2Dec(unsigned short bcdnum);`
戻り値	１０進値に変換された値を返す。
解説	８ビット BDC 数値 bcdnum を１０進の等価な値に変換する。

例	`unsigned short a;`
	`...`
	`a = Bcd2Dec(0x52);` // 52 に等しい

Dec2Bcd

原形	`unsigned short Dec2Bcd(unsigned short decnum);`
戻り値	BCD 値に変換された値を返す。
解説	８ビット BDC 数値 bcdnum を BCD に変換する。

例	`unsigned short a;`
	`...`
	`a = Dec2Bcd(52);` // 0x52 に等しい

Bcd2Dec16

原形	`unsigned Bcd2Dec16(unsigned bcdnum);`
戻り値	１０進値に変換された値を返す。
解説	１６ビット BDC 数値 bcdnum を１０進の等価な値に変換する。

例	`unsigned a;`
	`...`
	`a = Bcd2Dec16(1234);` // 4660 に等しい

Dec2Bcd16

原形	`unsigned Dec2Bcd(unsigned decnum);`
戻り値	BCD 値に変換された値を返す。
解説	１６ビット１０進数値 decnum を BCD に変換する。

例	`unsigned a;`
	`...`
	`a = Dec2Bcd16(4660);` // 1234 に等しい

Trigonometry　Library（三角法）

　mikroC は基本的な三角関数（三角形と三角関数）というツールを与える。
これらの関数はルックアップテーブルとして実行され、１０００倍して切り上げした整数値として結果を返す。

Library　Routines

```
SinE3
CosE3
```

SinE3

原形	`int SinE3(unsigned angle_deg);`
戻り値	関数は１０００倍（1E3）して最も近い整数に切り上げした入力変数の正弦（sin）を返す。 戻り値の範囲は-1000 から 1000 である。
解説	関数は角度で表現される変数 angle_deg を取り、１０００倍して最も近い整数に切り上げしたその正弦（sin）を返す。関数はルックアップテーブルとして実行される。 取得される最大誤差は± 1 である。
例	`res = SinE3(45);` //　結果は 707 である。

CosE3

原形	`int CosE3(unsigned angle_deg);`
戻り値	関数は１０００倍（1E3）して最も近い整数に切り上げした入力変数の余弦（cos）を返す。 戻り値の範囲は-1000 から 1000 である。
解説	関数は角度で表現される変数 angle_deg を取り、１０００倍して最も近い整数に切り上げしたその余弦（cos）を返す。関数はルックアップテーブルとして実行される。 取得される最大誤差は± 1 である。
例	`res = CosE3(196);` //　結果は-193 である。

Sprint　Library

Sprint 関数のためのライブラリ。
注意）ANSI　C 標準に加えて、mikroC は、より少ない ROM,RAM で済み、且つ PIC のためある状況で便利でありうる、限定バージョンの sprinti、sprintlt を提供する。

Library　Routines

sprintf
sprintl
sprinti

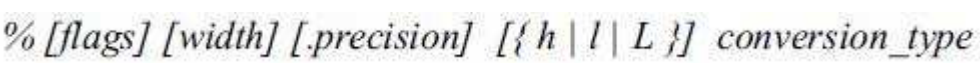
sprintf

解説：関数は連続する文字列や数値を書式指定し、バッファ内に結果として得た文字列を保存する。

注意）書式指定文字列は CONST 領域になければならない。Sprintf は p12 と p16 の PIC　MCU ファミリをサポートしない。

fmtstr 引数は書式指定文字列であり、文字、エスケープシーケンス及び書式仕様を基本とする。
通常の文字やエスケープシーケンスは、それらが中断される順にバッファへコピーされる。
書式指定の仕様は常にパーセント記号（%）で始まり、関数呼び出しに含まれるべき追加の引数を必要とする。

書式指定文字列は左から右へ読み込まれる。
遭遇した最初の書式指定の仕様は fmtstr の後の最初の引数を参照し、書式指定の仕様を用いてそれを変換し出力する。
第二の書式指定の仕様は fmtstr の後の第二の引数をアクセスする、など。
もし書式指定の仕様より多い引数がある場合、超過した引数は無視される。
もし書式指定の仕様に対し引数が不足ならば、結果は予知できない。
書式指定の仕様は次の書式を有する。



書式指定の仕様の各フィールドは特別なフォーマット・オプション（書式）を指定する１文字又は１数字でよい。
Conversion_type フィールドとは、引数が次のテーブルに示すように文字、文字列、数字もしくはポインタのように中断される事を指定する場所である。

Conversion Type	Argument Type	Output Format
d	int	符号付き 10 進数
u	unsigned int	符号無し 10 進数
o	unsigned int	符号無し 8 進数
x	unsigned int	0～F を使う符号無し 16 進化 10 進数
X	double	書式［－］dddd.dddd を使う浮動小数点数
e	double	書式［－］d.dddde［－］dd を使う浮動小数点数
E	double	書式［－］d.ddddE［－］dd を使う浮動小数点数
g	double	指定された値及び精度に対しより最適ならいづれでもよい書式 e またはf を使う浮動小数点数
c	int	int が符号無し char へ変換され、結果の文字が書き込まれる。
s	char *	終端にヌル文字の付いた文字列
p	void *	ポインタ値。X 書式が使用される。
%	none	A　%が書き込まれる。変換される引数は無い。完全な変換の仕様は%%であろう。

フラグ・フィールドは、単一文字が出力を判別し、以下の表に示すように＋／－記号、 空白、小数点及び８進と16進の接頭文字を出力するために使用される場所である。

Flags	Meaning
-	指定されたフィールド巾に出力を左に整える
+	もし出力が符号付の型の場合、＋または－記号の付いた出力値の接頭辞。
space (' ')	もし符号付整数値の場合、空白付きの出力値の接頭辞。そうでなければ空白の接頭辞は無い。
#	o，x及びXフィールド型と共に個々に使われる場合、0、0x、または0xの付いた非ゼロ出力値を前につける。e，E，f，g及びGフィールド型を伴い使用される場合、＃フラグが敢えて10進小数を出力値に含ませようとする。＃は他の場合は全て無視する。
*	＊はフォーマット指示子を無視する。

width フィールドは出力する文字の最小数を指定する非負数である。
もし出力値内の文字数がwidthより小さい場合、空白が最小幅に詰め込むため（－フラグが指定される場合）左または右に追加される。
もしwidthが0を前につけられた場合、ゼロが空白の代わりに詰め込まれる。
widthフィールドはフィールドを決して切り詰めない。
もし出力値の長さが指定の *width* を超えた場合、全文字が出力される。
精度のフィールドは文字数、有効数字の桁数、または小数部分の桁数を出力するための文字数を指定する非負数である。
精度フィールドは、次の表で指定されるように浮動小数点数の場合に、結果として切り捨てまたは出力値のまるめを行うことが可能である。

Flags	Meaning of precision field
d, u, o, x, X	精度フィールドはあなたが出力値に含まれる最小桁数を指定する場所である。もし引数の桁数が精度フィールドで定義される数を超えた場合、桁は切り捨てされない。もし引数の桁数が精度フィールドより少ない場合、出力値はゼロを左に詰めこまれる。
f	The precision field is where you specify the number of digits to the right of the decimal point. The last digit is rounded.
e, E	The precision field is where you specify the number of digits to the right of the decimal point. The last digit is rounded.
g	精度フィールドは、あなたが出力値の有効数字の最大数を指定する場所である。
c, C	精度フィールドは、このフィールド型の影響を受けない。
s	精度フィールドは、あなたが出力値の文字の最大数を指定する場所である。過剰な文字は出力しない。

オプション文字 h と l 又は L は、short または long バージョンの整数タイプ d,I,u,o,x,X を個々に指定するため、*conversion_type* の直ぐ前にあってよい。

あなたは、引数の型が書式指定の仕様のそれと一致することを守らねばならない。
あなたは、適切な型が *sprintf* へパスされることを守るためタイプキャストを使用可能である。

sprintl

原形	`int sprintl (`
	`  char *buffer,  /* storage buffer */`
	`  const char *fmtstr,  /* format string */`
	`  ... );  /* additional arguments */`

戻り値	関数は実際にバッファへ書き込まれた文字数を返す。
解説	sprintf と同様、それが不動小数型の数をサポートしないことを除く。
例	なし

sprinti

原形	`int sprinti (`
	`  char *buffer,  /* storage buffer */`
	`  const char *fmtstr,  /* format string */`
	`  ... );  /* additional arguments */`

戻り値	関数は実際にバッファへ書き込まれた文字数を返す。
解説	sprintl と同様、それがロング整数型の数をサポートしないことを除く。
例	なし

SPI Graphic Lcd Library

mikroC は SPI 経由でグラフィック LCD128 x 64 を操作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ある GLCD128 x 64（サムソン ks0108）を動かす。

重要）SPI ライブラリルーチンを使用する場合、あなたは実際の SPI モジュールか Spi_Glcd_Init 内の SPI1 または SPI2 を指定するよう要求される。

注意）Spi_Init()は SPI GLCD を初期化する前に呼び出さねばならない。

Library Routines

Basic routines

```
Spi_Glcd_Init
Spi_Glcd_Set_Side
Spi_Glcd_Set_Page
Spi_Glcd_Set_X
Spi_Glcd_Read_Data
Spi_Glcd_Write_Data
```

Advanced routines

```
Spi_Glcd_Fill
Spi_Glcd_Dot
Spi_Glcd_Line
Spi_Glcd_V_Line
Spi_Glcd_H_Line
Spi_Glcd_Rectangle
Spi_Glcd_Box
Spi_Glcd_Circle
Spi_Glcd_Set_Font
Spi_Glcd_Write_Char
Spi_Glcd_Write_Text
Spi_Glcd_Image
```

Spi_Glcd_Init

原形	`void SPI_Glcd_Init(char DeviceAddress, unsigned int * rstport, unsigned int rstpin, unsigned int * csport, unsigned int cspin);`
戻り値	
解説	SPI 経由でグラフィック LCD128 x 64 を初期化する。 RstPort と RstPin－spi エキスパンダのリセット端子に接続された端子を設定する。 CSPort と CSPin－spi エキスパンダの CS 端子に接続された端子を設定する。 Device address－spi エキスパンダのアドレス（ Spi エキスパンダ上の A0,A1 及び（VCC または GND に接続された）A2 端子のハードウェア設定 ）
要求事項	注意 : SPI_Init（ ）は SPI GLCD を初期化する前に呼び出されねばならない。この手続きは SPI GLCD ライブラリのたのる一朕を使用する前に呼び出されねばならない。
例	`Spi_Glcd_Init(0,&PORTC, 0, &PORTC, 1);`

Spi_Glcd_Set_Side

原形	`void SPI_Glcd_Set_Side(char x_pos);`
戻り値	
解説	GLCD の左または右側面を選択する。 変数 X は側面を指定する。０～６３の値は左側面を指定し、６４以上の高さは側面を指定する。 GLCD 上の正確な位置を指定するため、関数 Spi_Glcd_Set_Side 、Spi_Glcd_Set_X 、Spi_Glcd_Set_Page を使う。 それから、あなたはその位置で Spi_Glcd_Write_Data、Spi_Glcd_Read_Data を使用できる。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Select_Side(0);` `Spi_Glcd_Select_Side(10);`

Spi_Glcd_Set_Page

原形	`void Spi_Glcd_Set_Page(char page);`
戻り値	なし
解説	GLCD のページを選択する。技術的にはディスプレイ上の１行、変数 page は０～７である。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Set_Page(5);`

Spi_Glcd_Set_X

原形	`void SPI_Glcd_Set_X(char 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 ドットの位置を示す。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Set_X(25);`

Spi_Glcd_Set_Data

原形	`char Spi_Glcd_Read_Data();`
戻り値	なし
解説	GLCD メモリの現在の位置からデータを読み出す。 GLCD の正確な位置を指定するため、関数 Spi_Glcd_Set_Side、Spi_Glcd_Set_X または Spi_Glcd_Set_Page を使用する。それから、あなたはその場所で Spi_Glcd_Write_Data または Spi_Glcd_Read_Data を使用できる。
要求事項	GLCD メモリの現在の位置からのデータを読み出す。Spi_Glcd_Init を見よ。
例	`tmp = Spi_Glcd_Read_Data;`

Spi_Glcd_Write_Data

原形	`void Spi_Glcd_Write_Data(char data);`
戻り値	なし
解説	GLCD メモリの現在の位置 date を書き込み、次の位置へ移動する。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Write_Data(data)`

Spi_Glcd_Fill

原形	`void Spi_Glcd_Fill(char pattern);`
戻り値	なし
解説	バイトパターンで GLCD メモリを満たす。GLCD 画面を消去するには、Spi_Glcd_Fill（0）を用いる。画面を完全に満たすには Spi_Glcd_Fill（$FF）を用いる。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Fill(0);  // Clear screen`

Spi_Glcd_Dot

原形	`void Spi_Glcd_Dot(char x_pos, char y_pos, char color);`
戻り値	なし
解説	ＧＬＣＤ上の直交座標（ｘ，ｙ）に１ドット描画する。変数 color はドットの状態を決定する。０はドットをクリアし、１ではドットを置き、２ではドットの状態を反転する。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Dot(0, 0, 2);` // 左上隅のドットを反転する

Spi_Glcd_Line

原形	`void SPI_Glcd_Line(int x_start, int y_start, int x_end, int y_end, char color);`
戻り値	なし
解説	ＧＬＣＤ上に（x1,y1）から（x2,y2）へ線分を描画する。変数 color はドットの状態を決定する。０は空の線（ドットを消去）を描画し、１は実線（ドッドを置く）を描画し、２は“スマート”な線を（各ドットを反転）描画する。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Line(0, 63, 50, 0, 2);`

Spi_Glcd_V_Line

原形	`void Spi_Glcd_V_Line(char y_start, char y_end, char x_pos, char color);`
戻り値	なし
解説	ＧＬＣＤ上に（x,y1）から（x,y2）へ垂線を描画する。変数 color はドットの状態を決定する。０は空の線（ドットを消去）を描画し、１は実線（ドッドを置く）を描画し、２は“スマート”な線を（各ドットを反転）描画する。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_V_Line(0, 63, 0, 1);`

Spi_Glcd_H_Line

原形	`void Spi_Glcd_H_Line(char x_start, char x_end, char y_pos, char color);`
戻り値	なし
解説	ＧＬＣＤ上に（x1,y）から（x2,y）へ水平線を描画する。変数 color はドットの状態を決定する。０は空の線（ドットを消去）を描画し、１は実線（ドッドを置く）を描画し、２は“スマート”な線を（各ドットを反転）描画する。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_H_Line(0, 127, 0, 1);`

Spi_Glcd_Rctangle

原形	`void Spi_Glcd_Rectangle(char x_upper_left, char y_upper_left, char x_bottom_right, char y_bottom_right, 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 は縁取りを定義する。0 は空の縁取り（ドットを消去）を描画し、１は実線（ドッドを置く）を描画し、２は“スマート”な線を（各ドットを反転）描画する。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Rectangle(10, 0, 30, 35, 1);`

Spi_Glcd_Box

原形	`void Spi_Glcd_Box(char x_upper_left, char y_upper_left, char x_bottom_right, char y_bottom_right, 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 は白の箱（ドットを消去）を描画し、１は塗りつぶしの箱を（ドッドを置く）を描画し、２は“スマート”な箱を（各ドットを反転）描画する。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Box(10, 0, 30, 35, 1);`

Spi_Glcd_Circle

原形	`void Spi_Glcd_Circle(int x_center, int y_center, int radius, char color);`
戻り値	なし
解説	ＧＬＣＤ上に（x,y）を中心とした半径 radius の円を描画する。変数 color は円を定義する。0 は空の線（ドットを消去）を描画し、１は実線を（ドッドを置く）を描画し、２は“スマート”な線を（各ドットを反転）描画する。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Circle(63, 31, 25, 1);`

Spi_Glcd_Set_Font

原形	`void SPI_Glcd_Set_Font(const char * activeFont, char aFontWidth, char aFontHeight, unsigned int aFontOffs);`
解説	Spi_Glcd_Write_Char と Spi_Glcd_Write_Text のテキスト表示ルーチンを設定する。フォントはバイト配列としてフォーマットされる必要がある。変数 font_address はフォントのアドレスを指定する。あなたは@演算子の付いたフォント名を通すことができる。変数 font_width と font_height はドットの文字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ず、フォント高さは８ドットを超えてはならない。変数 font_offsaet は供給されたフォントの開始からの ASCII 文字を決定する。 ライブラリと共に提供されたデモフォントは、それらが空白で始まることを意味する３２オフセットを有する。 もしフォントが指定されなければ、Spi_Glcd_Write_Char と Spi_Glcd_Write_Text はライブラリで供給されるデフォルトの５ｘ８フォントを使用するだろう。あなたは“GLCD_Fonts.dpas”ファイル内に与えられたガイドラインにしたがってあなた自身のフォントを作ることが可能である。このファイルは GLCD 用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“に置かれる。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	// 空白（３２）で始まる“myfont”の独自の 5x7 フォントを使う `Spi_Glcd_Set_Font(@myfont, 5, 7, 32);`

Spi_Glcd_Write_Char

原形	`void SPI_Glcd_Write_Char(char chr1, char x_pos, char page_num, char color);`
解説	ディスプレイに左の端から x ドット離れた、ページ（８つの GLCD 行 0～7 の内の１つ）の文字を出力する。変数 color は“塗りつぶし”を定義する。0 は“白”の字（ドットを消去）を描画し、１は実線の字を（ドッドを置く）を描画し、２は“スマート”な字を（各ドットを反転）描画する。 フォントを指定するため、Spi_Glcd_Set_Font ルーチンを使用すること。さもなくばデフォルトフォント５ｘ８（ライブラリに含まれる）が使用されるだろう。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。表示用のフォントを指定するため Spi_Glcd_Set_Font を使用すること。もしフォントが指定されねば、ライブラリで供給されるデフォルトの５ｘ８フォントが使用されるだろう。
例	`Spi_Glcd_Write_Char('C', 0, 0, 1);`

Spi_Glcd_Write_Text

原形	`void SPI_Glcd_Write_Text(char text[], char x_pos, char page_num, char color);`
解説	ディスプレイの左の端から x ドット離れた、ページ（８つの GLCD 行 0～7 の内の１つ）に text を出力する。変数 color は“塗りつぶし”を定義する。0 は“白”の字（ドットを消去）を描画し、１は実線の字を（ドッドを置く）を描画し、２は“スマート”な字を（各ドットを反転）描画する。 フォントを指定するため、Spi_Glcd_Set_Font ルーチンを使用すること。さもなくばデフォルトフォント５ｘ７（ライブラリに含まれる）が使用されるだろう。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。表示用のフォントを指定するため Spi_Glcd_Set_Font を使用すること。もしフォントが指定されねば、ライブラリで供給されるデフォルトの５ｘ８フォントが使用されるだろう。
例	`Spi_Glcd_Write_Text('Hello world!', 0, 0, 1);`

Spi_Glcd_Image

原形	`void Spi_Glcd_Image(const char * image);`
解説	GLCD にビットマップイメージを表示する。 変数 image は 1024 バイトの配列としてフォーマットされるべきである。 GLCD に表示するための最適な定数配列へイメージを変換するため、MikroPascal の統合ビットマップ－LCD 変換エディタ（メニュー option Tool＞Graphic LCD Editor）を使うこと。
要求事項	GLCD は初期化されねばならない。Spi_Glcd_Init を見よ。
例	`Spi_Glcd_Image(my_image); :`

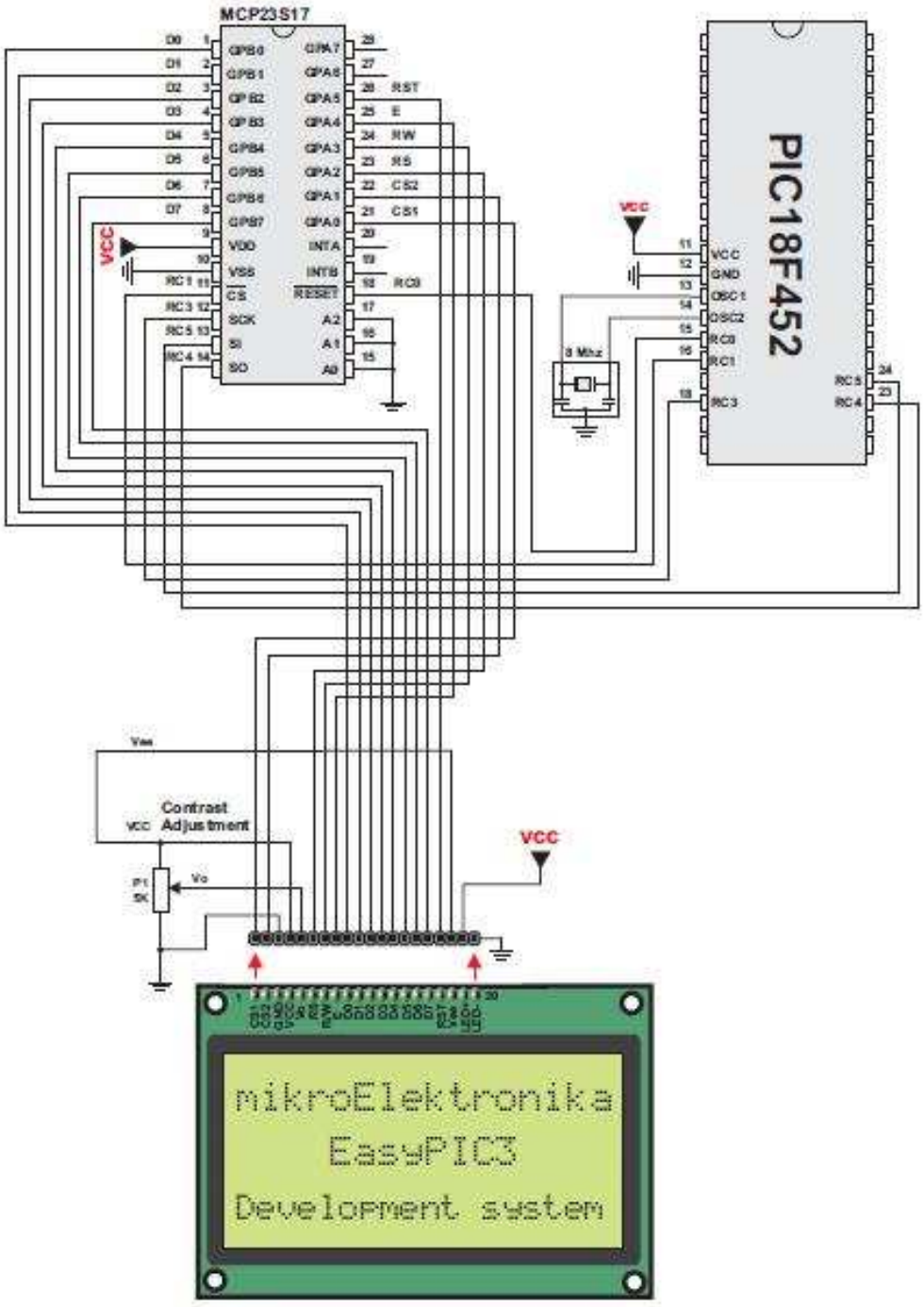
Library　Example

この例は、シリアルーパラレル変換器MCP２３S１７を使用するため、SPIモジュール経由でGLCD　KS0108と通信をする方法を説明している。

```
extern const unsigned short
  truck_bmp[];

char ii;
unsigned int jj;
char *someText;

void delay2S() {
  Delay_ms(2000);
}

void main() {
  ADCON1 |= 0x0F;

  Spi_Init();     // Initialize SPI module

  Spi_Glcd_Init(0,&PORTC, 0, &PORTC, 1);
  Spi_Glcd_Fill(0xAA);
  delay2S();
  while(1) {
      Spi_Glcd_Fill(0x00);
      Spi_Glcd_Image( truck_bmp );
      delay2S();

      for(jj = 1; jj <= 40; jj++)
        Spi_Glcd_Dot(jj,jj,1);
      delay2S();

      Spi_Glcd_Fill(0x00);
      Spi_Glcd_Line(120, 1, 5,60, 1);
      delay2S();
      Spi_Glcd_Line(12, 42, 5,60, 1);
      delay2S();

      Spi_Glcd_Rectangle(12, 20, 93,57, 1);
      delay2S();

      //continues..
```

```
        //continues..

        Spi_Glcd_Line(120, 12, 12,60, 1);
        delay2S();

        Spi_Glcd_H_Line(5, 40, 6, 1);
        delay2S();
        Spi_Glcd_Line(0, 12, 120, 60, 1);
        Spi_Glcd_V_Line(7, 63, 127, 1);
        delay2S();

        for(ii = 1; ii <= 10; ii++)
          Spi_Glcd_Circle(63, 32, 3*ii, 1);

        delay2S();
        Spi_Glcd_Box(12, 20, 70, 57, 2);
        delay2S();

        Spi_Glcd_Fill(0x00);

        Spi_Glcd_Set_Font(&System3x6, 3, 6, 32);
        someText = "SMALL FONT: 3X6";
        Spi_Glcd_Write_Text(someText, 20, 5, 1);

        Spi_Glcd_Set_Font(&FontSystem5x8, 5, 8, 32);
        someText = "Large Font 5x8";
        Spi_Glcd_Write_Text(someText, 3, 4, 1);
        delay2S();
    }
}
```

Hardware Connection



Port Expander Library

SPI エキスパンダ・ライブラリはマイクロチップ社の SPI ポート・エキスパンダ MCP23S17 の動作を容易にする。このチップは以下に示した回路に従って PIC に接続する。

注意）PIC は SPI モジュールのハードウェアを有する必要がある。
注意）SPI_Init()はポートエキスパンダを初期化する前に呼び出されねばならない。

Library Routines

```
Expander_Init
PortExpanderSelect
PortExpanderUnSelect
Expander_Read_Byte
Expander_Write_Byte
Expander_Set_Mode
Expander_Read_Array
Expander_Write_Array
Expander_Read_PortA
Expander_Read_PortB
Expander_Read_ArrayPortA
Expander_Read_ArrayPortB
Expander_Write_PortA
Expander_Write_PortB
Expander_Set_DirectionPortA
Expander_Set_DirectionPortB
Expander_Set_PullUpsPortA
Expander_Set_PullUpsPortB
```

Expander_Init

原形	`void Expander_Init(char ModuleAddress, unsigned int * rstport, unsigned int rstpin, unsigned int * csport, unsigned int cspin);`
戻り値	なし
解説	エキスパンダとの SPI 通信を確立し、エキスパンダを初期化する。Pstport と Rstpin は SPI エキスパンダのリセット端子に接続された端子を設定する。RstPort と RstPin－SPI エキスパンダの CS 端子に接続された端子を設定する。Moduleaddress－SPI エキスパンダのアドレスである。(SPI エキスパンダの（VCC 又は GND に接続された）A0,A1,A2 端子のハードウェア設定)
必要事項	この手続きはポートエキスパンダライブラリの他のルーチンを使用する前に呼び出される必要がある。ポートエキスパンダを初期化する前に、SPI_Init()が呼び出されねばならない。
例	`Expander_Init(0, &PORTC, 0, &PORTC, 1);`

PortExpanderSlect

原形	`void PortExpanderSelect;`
戻り値	なし
解説	現在のポートエキスパンダを選択する。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`PortExpanderSelect;`

PortExpanderUnSlect

原形	`void PortExpanderUnSelect;`
戻り値	なし
解説	現在のポートエキスパンダを非選択とする。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`PortExpanderUnSelect;`

Expander_Read_Byte

原形	`char Expander_Read_Byte(char ModuleAddress, char RegAddress);`
戻り値	ポートエキスパンダからの読込みバイト値
解説	関数はModuleAddress 上のポートエキスパンダと RegAddress 上のポートからバイト値を読込む。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Read_Byte(0,1);`

Expander_Write_Byte

原形	`void Expander_Write_Byte(char ModuleAddress,char RegAddress, char Data);`
戻り値	なし
解説	このルーチンはModuleAddress 上のポートエキスパンダと RegAddress 上のポートへデータを書込む。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Write_Byte(0,1,$FF);`

Expander_Set_Mode

原形	`void Expander_Set_Mode(char ModuleAddress, char Mode);`
戻り値	なし
解説	ModuleAddress 上のポートエキスパンダ mode を設定する。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Set_Mode(1,0);`

Expander_Read_ArrayPortA

原形	`void Expander_Read_ArrayPortA(char ModuleAddress, char NoBytes, char DestArray[]);`
戻り値	なし
解説	このルーチンは ModuleAddress 上のポートエキスパンダ及びポートＡからバイト配列（DestArray）を読込む。NoBytes は読み出しバイト数を示す。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Read_PortA(0,1,data);`

Expander_Read_Array

原形	`void Expander_Read_Array(char ModuleAddress, char StartAddress, char NoBytes, char *DestArray);`
戻り値	なし
解説	このルーチンは ModuleAddress 及び StartAddress 上のポートエキスパンダからバイト配列（DestArray）を読込む。NoByte は読出しバイト数を示す。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Read_Array(1,1,5,data);`

Expander_Write_Array

原形	`void Expander_Write_Array(char ModuleAddress, char StartAddress, char NoBytes, char *SourceArray);`
戻り値	なし
解説	このルーチンは ModuleAddress 及び StartAddress 上のポートエキスパンダへバイト配列（SouceArray）を書込む。NoBytes は書込みバイト数を示す。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Write_Array(1,1,5,data);`

Expander_ Read _ PortA

原形	`char Expander_Read_PortA(char Address);`
戻り値	読出しバイト値
解説	このルーチンは Address 及び PortA 上のポートエキスパンダからバイト配列を読込む。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Read_PortA(1);`

Expander_Read_ Array PortB

原形	`void Expander_Read_ArrayPortB(char ModuleAddress, char NoBytes, char DestArray[]);`
戻り値	なし
解説	このルーチンはModuleAddress 及びポートB 上のポートエキスパンダからバイト配列(DestArray)を読込む。NoBytes は読出しバイト数を示す。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Read_PortB(0,8,data);`

Expander_Write_ PortA

原形	`void Expander_Write_PortA(char ModuleAddress, char Data);`
戻り値	なし
解説	このルーチンはModuleAddress 及びポートA 上のポートエキスパンダへバイト値を書込む。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_write_PortA(3,$FF);`

Expander_ Write _PortB

原形	`void Expander_Write_PortB(char ModuleAddress, char Data);`
戻り値	なし
解説	このルーチンはModuleAddress 及びポートB 上のポートエキスパンダへバイト値を書込む。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_write_PortB(2,$FF);`

Expander_Set_DirectionPortA

原形	`void Expander_Set_DirectionPortA(char ModuleAddress, char Data);`
戻り値	なし
解説	ポートAの端子を入力又は出力としてポートエキスパンダを設定する。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Set_DirectionPortA(0,$FF);`

Expander_Set_DirectionPortB

原形	`void Expander_Set_DirectionPortB(char ModuleAddress, char Data);`
戻り値	なし
解説	ポート B の端子を入力又は出力としてポートエキスパンダを設定する。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Set_DirectionPortB(0,$FF);`

Expander_Set_PullUpsPortA

原形	`void Expander_Set_PullUpsPortA(char ModuleAddress, char Data);`
戻り値	なし
解説	このルーチンはポートAの端子をプルアップ又はプルダウンするようポートエキスパンダを設定する。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Set_PullUpsPortA(0,$FF);`

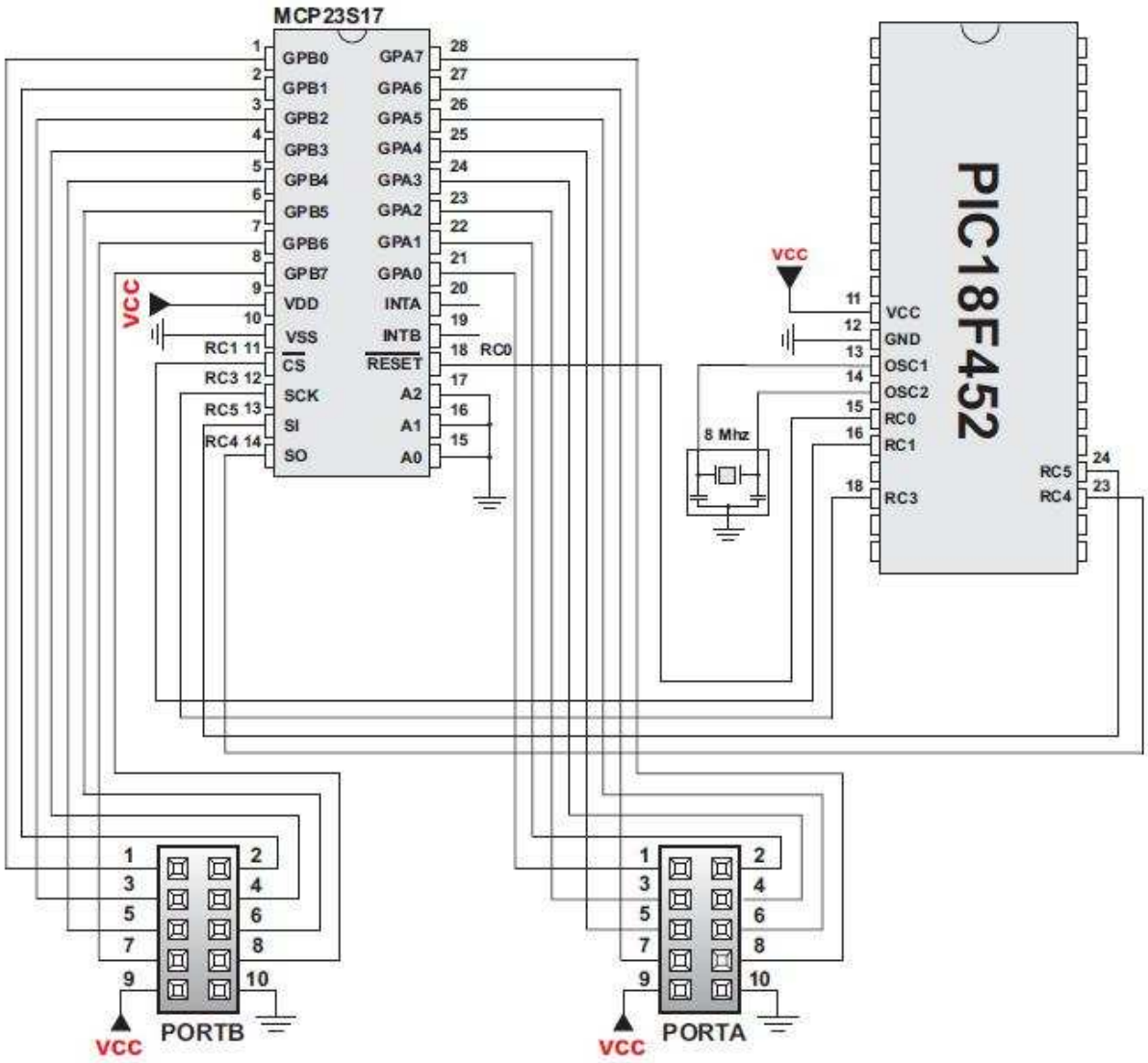
Expander_Set_PullUpsPortB

原形	`void Expander_Set_PullUpsPortB(char ModuleAddress, char Data);`
戻り値	なし
解説	このルーチンはポート B の端子をプルアップ又はプルダウンするようポートエキスパンダを設定する。
必要事項	ポートエキスパンダは初期化されねばならない。Expander_Init を見よ。
例	`Expander_Set_PullUpsPortB(0,$FF);`

Library Example

```
unsigned char  i;

void main(){
  ADCON1 |= 0x0f;
  TRISB = 0x00;
  LATB  = 0xFF;
  Delay_ms(2000);

  Spi_Init();     // Initialize SPI module

  Expander_Init(0, &PORTC, 0, &PORTC, 1);      // initialize port expander

  Expander_Set_DirectionPortA(0, 0);        // set expander's porta to be output

  Expander_Set_DirectionPortB(0,0xFF);         // set expander's porta to be input
  Expander_Set_PullUpsPortB(0,0xFF);
  // set pull ups to all of the expander's portb pins

  i = 0;
  while(1) {
      Expander_Write_PortA(0, i++);            // write i to expander's porta
      LATB = Expander_Read_PortB(0);
  // read expander's portb and write it to PIC's LATB
      Delay_ms(20);
  }
}
```

Hardware Connection



SPI　Lcd　Library(4Bit　interface)

　mikroC は、SPI インターフェース経由でよく使われる LCD（４ビットインターフェース）と通信するためのライブラリを提供する。
PIC 及び SPI　LCD とのハードウェア接続を示す図をこの章終わりに掲示する。

注）Spi_Init()；が SPI　LCD を初期化する前に呼び出されねばならない。

Library　Routines

```
Spi_Lcd_Config
Spi_Lcd_Init
Spi_Lcd_Out
Spi_Lcd_Out_Cp
Spi_Lcd_Chr
Spi_Lcd_Chr_Cp
Spi_Lcd_Cmd
```

Spi_Lcd_Config

原形	`void Spi_Lcd_Config(char DeviceAddress, unsigned char * rstport, unsigned char rstpin, unsigned char * csport, unsigned char cspin);`
戻り値	なし
解説	あなたが指定する端子設定（リセット端子及びチップセレクト端子）で、SPI インターフェース経由で LCD を初期化する。
必要事項	Spi_Init;が SPI　LCD を初期化する前に呼び出さねばならない。
例	`Spi_Lcd_Config(0, &PORTB, 1, &PORTB, 0);`

Spi_Lcd_Init

原形	`void Spi_Lcd_Init();`
戻り値	なし
解説	デフォルト端子設定（この章の終わりの接続回路を見よ）によるポートでLCDを初期化する。
必要事項	Spi_Init;がSPI　LCDを初期化する前に呼び出さねばならない。
例	`Spi_Lcd_Init();`

Spi_Lcd_ Out

原形	`void Spi_Lcd_Out(char row, char column, 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）にテキストを出力する。両文字変数とリテラルはテキストとして通される。
必要事項	LCDのポートは初期化されねばならない。Spi_Lcd_ConfigまたはSpi_Lcd_Initを見よ。
例	`Spi_Lcd_Out(1, 3, "Hello!");`

Spi_Lcd_Out_Cp

原形	`void Spi_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を出力する。両文字変数とリテラルはテキストとして通される。
必要事項	LCDのポートは初期化されねばならない。Spi_Lcd_ConfigまたはSpi_Lcd_Initを見よ。
例	`Spi_Lcd_Out_Cp("Here!");` // 現在のカーソル位置に“Here!”を出力する。

Spi_Lcd_Chr

原形	`void Spi_Lcd_Chr(char Row, char Column, char Out_Char);`
戻り値	なし
解説	指定された LCD 上の列、行（変数 row と col）に character を出力する。両文字変数とリテラルはテキストとして通される。
必要事項	LCD のポートは初期化されねばならない。Spi_Lcd_Config または Spi_Lcd_Init を見よ。
例	`Spi_Lcd_Chr(2, 3, "i");`

Spi_Lcd_Chr_Cp

原形	`void Spi_Lcd_Chr_CP(char Out_Char);`
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。両文字変数とリテラルはテキストとして通される。
必要事項	LCD のポートは初期化されねばならない。Spi_Lcd_Config または Spi_Lcd_Init を見よ。
例	`Spi_Lcd_Chr_Cp("e");` // 現在のカーソル位置に“e”を出力する。

Spi_Lcd_Cmd

原形	`void Spi_Lcd_Cmd(char out_char);`
戻り値	なし
解説	LCD へコマンド（命令）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すことができる。有効なコマンドの完全な表を以下に示す。
必要事項	LCD のポートは初期化されねばならない。Spi_Lcd_Config または Spi_Lcd_Init を見よ。
例	`Spi_Lcd_Cmd(LCD_Clear);` // LCD 表示器を初期化する。

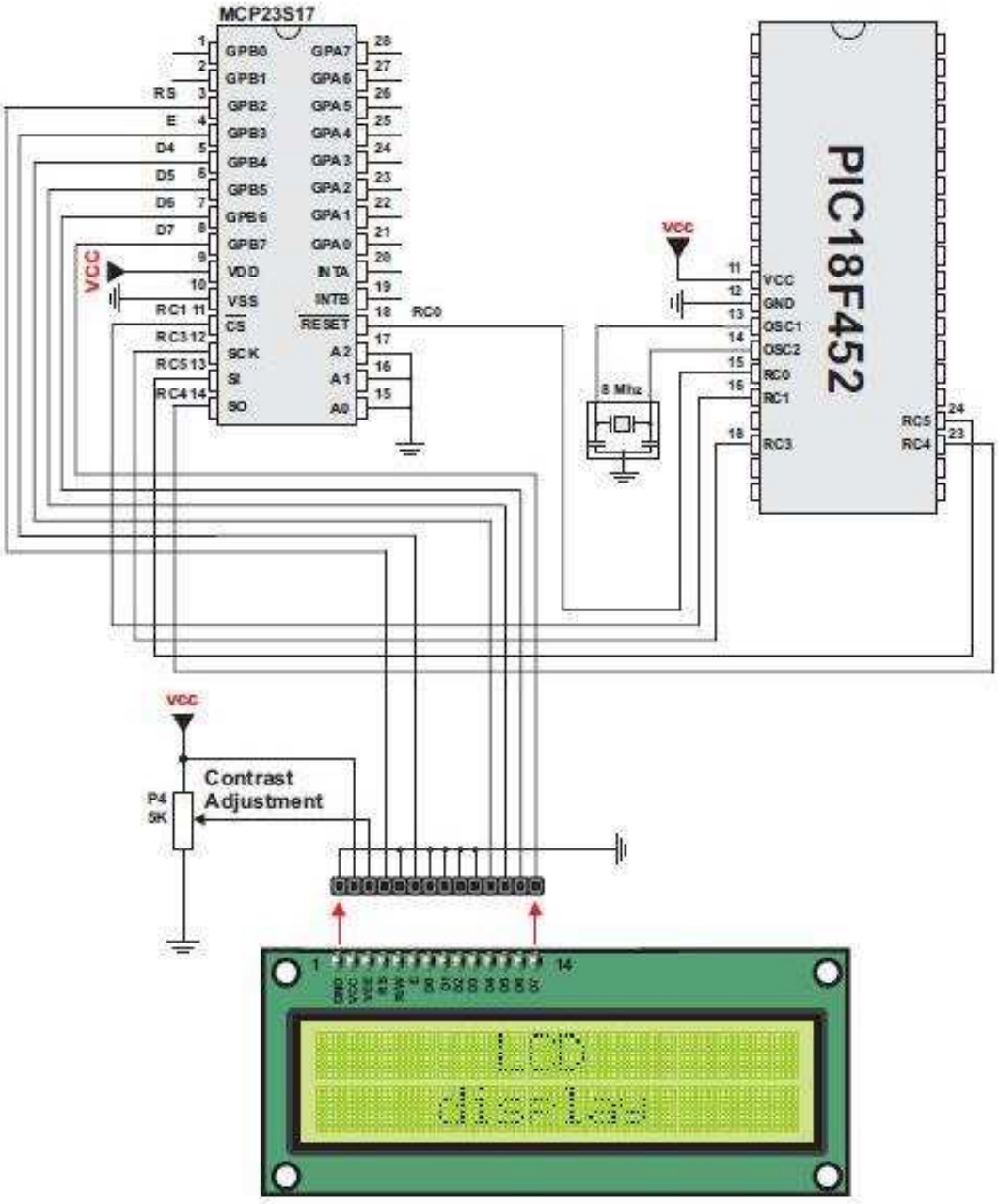
LCD Commands

LCD 命令	目的
LCD_FIRST_ROW	カーソルを第１列へ移動
LCD_SECOND_ROW	カーソルを第２列へ移動
LCD_THIRD_ROW	カーソルを第３列へ移動
LCD_FOURTH_ROW	カーソルを第４列へ移動
LCD_CLEAR	ディスプレイをクリア
LCD_RETURN_HOME	カーソルをホームポジションに戻し、シフトした表示を原点へ戻す。
LCD_CORSOL_OFF	カーソルをオフする。
LCD_UNDERLINE_ON	アンダーラインのカーソルをオンする。
LCD_BLINK_CURSOL_ON	ブリンクカーソル　オン
LCD_MOVE_CURSOL_LEFT	データ RAM を変更せずにカーソルを左へ移動する。
LCD_ MOVE_CURSOL_RIGHT	データ RAM を変更せずにカーソルを右へ移動する。
LCD_TURN_ON	LCD 表示オン
LCD_ TURN_Off	LCD 表示オフ
LCD_SHIFT_LEFT	表示 RAM を変えずに左へ表示をシフト
LCD_SHIFT_ RIGHT	表示 RAM を変えずに右へ表示をシフト

Library Example

```
char *text = "mikroElektronika";

void main() {
  Spi_Init();                    // initialize spi
  Spi_Lcd_Init();                // initialize lcd over spi interface
  Spi_Lcd_Cmd(LCD_CLEAR);        // Clear display
  Spi_Lcd_Cmd(LCD_CURSOR_OFF);   // Turn cursor off
  Spi_Lcd_Out(1,6, "mikroE");    // Print text to LCD, 1st row, 7th column
  Spi_Lcd_Chr_CP('!');           // append !
  Spi_Lcd_Out(2,0, text);        // Print text to LCD, 2nd row, 3rd column
  Spi_Lcd_Out(3,1,"mikroE");     // for lcd with more than two raws
  Spi_Lcd_Out(4,15,"mikroE");    // for lcd with more than two raws
} //~!
```

Hardware Connection



SPI　Lcd8(8Bit　interface)　Library

mikroC は、ＳＰＩ経由でよく使われる８ｂｉｔ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 及び SPI　LCD の HW 接続を示す図をこの章の最後に掲示する。

注意：Spi_Init（）；が SPI　LCD８を初期化する前に呼び出されねばならない。

Library　Routines

```
Spi_Lcd8_Config
Spi_Lcd8_Init
Spi_Lcd8_Out
Spi_Lcd8_Out_Cp
Spi_Lcd8_Chr
Spi_Lcd8_Chr_Cp
Spi_Lcd8_Cmd
```

Spi_Lcd_8_Config

原形	`void Spi_Lcd8_Config(char DeviceAddress, unsigned char * rstport, unsigned char rstpin, unsigned char * csport, unsigned char cspin);`
戻り値	なし
解説	SPI 経由で、あなたが指定する端子設定（リセット端子とチップセレクト端子）で LCD を初期化する。
必要事項	Spi_Init()；が SPI　LCD8 を初期化する前に呼び出されねばならない。
例	`Spi_Lcd8_Config(0, &PORTB, 1, &PORTB, 0);`

Spi_Lcd8_Init

原形	`void Spi_Lcd8_Init();`
戻り値	なし
解説	デフォルト端子設定(この章の最後の接続図を見よ。)で、LCD に対しコントロールポート（ctrlport）とデータポート(dataport)を初期化する。
必要事項	Spi_Init()；が SPI　LCD8 を初期化する前に呼び出されねばならない。
例	`Spi_Lcd8_Init();`

Spi_Lcd8_Out

原形	`void Spi_Lcd8_Out(unsigned short row, unsigned short column, 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）にテキストを出力する。文字列変数とリテラル共にテキストとして通すことが可能である。
必要事項	LCD のポートは初期化されねばならない。Spi_Lcd8_Config または Spi_Lcd8_Init を見よ。
例	`Spi_Lcd8_Out(1, 3, "Hello!");` //　１行３文字目に“Hello！”を出力する

Spi_Lcd8_Out

原形	`void Spi_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。文字列変数とリテラル共に text として通すことが可能である。
必要事項	LCD のポートは初期化されねばならない。Spi_Lcd8_Config または Spi_Lcd8_Init を見よ。
例	`Spi_Lcd8_Out_Cp("Here!");` //　１行３文字目に“Here！”を出力する

Spi_Lcd8_Chr

原形	`void Spi_Lcd8_Chr(unsigned short row, unsigned short column, char out_char);`
戻り値	なし
解説	LCD 上の指定された列と行（(変数 row と col)）に character を出力する。文字列変数とリテラル共に character として通すことが可能である。
必要事項	LCD のポートは初期化されねばならない。Spi_Lcd8_Config または Spi_Lcd8_Init を見よ。
例	`Spi_Lcd8_Out(2, 3, "i");//` 2 行 3 文字目に“ i ”を出力する

Spi_Lcd8_Chr_Cp

原形	`void Spi_Lcd8_Chr_CP(char out_char);`
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。文字列変数とリテラル共に character として通すことが可能である。
必要事項	LCD のポートは初期化されねばならない。Spi_Lcd8_Config または Spi_Lcd8_Init を見よ。
例	`Spi_Lcd8_Chr_Cp("e");` // 現在のカーソル位置に“e”を出力する

Spi_Lcd8_Chr_Cp

原形	`void Spi_Lcd8_Cmd(char out_char);`
戻り値	なし
解説	LCD へコマンド（命令）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すことが可能である。有効なコマンドの完全な表を以下に示す。
必要事項	LCD のポートは初期化されねばならない。Spi_Lcd8_Config または Spi_Lcd8_Init を見よ。
例	`Spi_Lcd8_Cmd(LCD_Clear);` // LCD 表示器を初期化する

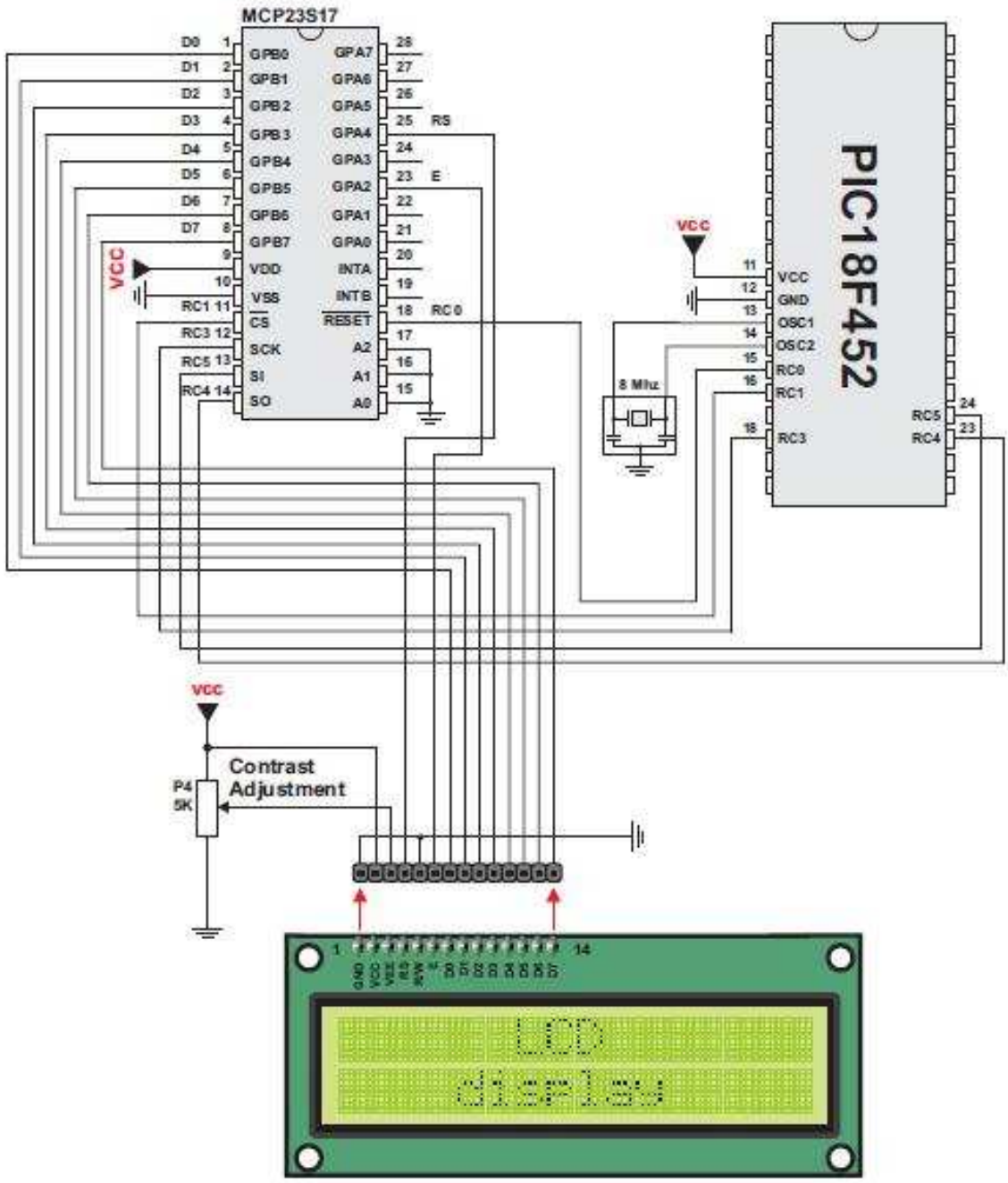
LCD　Commands

LCD 命令	目的
LCD_FIRST_ROW	カーソルを第１列へ移動
LCD_SECOND_ROW	カーソルを第２列へ移動
LCD_THIRD_ROW	カーソルを第３列へ移動
LCD_FOURTH_ROW	カーソルを第４列へ移動
LCD_CLEAR	ディスプレイをクリア
LCD_RETURN_HOME	カーソルをホームポジションに戻し、シフトした表示を原点へ戻す。
LCD_CORSOL_OFF	カーソルをオフする。
LCD_UNDERLINE_ON	アンダーラインのカーソルをオンする。
LCD_BLINK_CURSOL_ON	ブリンクカーソル　オン
LCD_MOVE_CURSOL_LEFT	データ RAM を変更せずにカーソルを左へ移動する。
LCD_ MOVE_CURSOL_RIGHT	データ RAM を変更せずにカーソルを右へ移動する。
LCD_TURN_ON	LCD 表示オン
LCD_ TURN_OFF	LCD 表示オフ
LCD_SHIFT_LEFT	表示 RAM を変えずに左へ表示をシフト
LCD_SHIFT_ RIGHT	表示 RAM を変えずに右へ表示をシフト

Library　Examples（デフォルト端子設定）

```
char *text = "mikroE";

void main() {
  Spi_Init();                          // initialize spi interface
  Spi_Lcd8_Init();                      // intialize lcd in 8bit mode via spi
  Spi_Lcd8_Cmd(LCD_CLEAR);              // Clear display
  Spi_Lcd8_Cmd(LCD_CURSOR_OFF);         // Turn cursor off
  Spi_Lcd8_Out(1,6, text);              // Print text to LCD, 1st row, 7th column...
  Spi_Lcd8_Chr_CP('!');                 // append '!'
  Spi_Lcd8_Out(2,0, "mikroelektronika");// Print text to LCD, 2nd row, 3rd column...
  Spi_Lcd8_Out(3,1, text);               // for lcd modules with more than two raws
  Spi_Lcd8_Out(4,15, text);             // for lcd modules with more than two raws
} //~!
```

Hardware Connection



SPI t6963c Graphic Lcd Library

PIC のために mikroC は、SPI インターフェース経由で東芝 T6963C グラフィック LCD（さまざまな大きさの）に描画し書き込むためのライブラリを提供する。

注意：Spi_Init（）；が SPI LCD８を初期化する前に呼び出されねばならない。

Library Routines

```
Spi_T6963C_Config
Spi_T6963C_writeData
Spi_T6963C_writeCommand
Spi_T6963C_setPtr
Spi_T6963C_waitReady
Spi_T6963C_fill
Spi_T6963C_dot
Spi_T6963C_write_char
Spi_T6963C_write_text
Spi_T6963C_line
Spi_T6963C_rectangle
Spi_T6963C_box
Spi_T6963C_circle
Spi_T6963C_image
Spi_T6963C_sprite
Spi_T6963C_set_cursor
Spi_T6963C_clearBit
Spi_T6963C_setBit
Spi_T6963C_negBit
Spi_T6963C_displayGrPanel
Spi_T6963C_displayTxtPanel
Spi_T6963C_setGrPanel
Spi_T6963C_setTxtPanel
Spi_T6963C_panelFill
Spi_T6963C_grFill
Spi_T6963C_txtFill
Spi_T6963C_cursor_height
Spi_T6963C_graphics
Spi_T6963C_text
Spi_T6963C_cursor
Spi_T6963C_cursor_blink
Spi_T6963C_Config_240x128
Spi_T6963C_Config_240x64
```

Spi_T6963C_Config

原形	`void Spi_T6963C_Config(unsigned int width, unsigned char height, unsigned char fntW, char DeviceAddress, unsigned char * rstport, unsigned char rstpin, unsigned char * csport, unsigned char cspin, unsigned char wr, unsigned char rd, unsigned char cd, unsigned char rst);`
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。この関数は、全ての SPI　T6963C ライブラリルーチンの前に呼び出されねばならない。 width－表示器の水平（x）ピクセル数 height－表示器の垂直（y）ピクセル数 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従い設定されねばならない。 data－データバスが接続されるポートアドレス cntrl－コントロールバスが接続されるポートアドレス wr－＊コントロールポートの！WR 線のビット数 rd－＊コントロールポートの！RD 線のビット数 cd－＊コントロールポートの！CD 線のビット数 rst－＊コントロールポートの！RST 線のビット数 DeviceAddress－デバイスアドレス ディスプレイＲAM： 　このライブラリは有効なＲAMの量を知りえない。このライブラリはパネル内のＲAMを切り分ける。完全なパネルはテキストパネルによる一つのグラフィックパネルである。プログラマは何枚のパネルを有するのかを知るため、ハードウェアを知らねばならない。
必要事項	Spi_Init()；が、ＳＰＩ東芝 T6963C グラフィック LCD を初期化する前に呼び出されねばならない。
例	`Spi_T6963C_Config(240, 64, 8, &PORTB, 1, &PORTB, 0, 0, 1, 3, 4, 0);` /* *　240 ピクセル幅 x64 ピクセル高さの表示器を初期化する *　キャラクタ幅 8bit *　PORTB.1 をリセット端子にする *　PORTB.0 をチップセレクト端子にする *　bit0 が！WR *　bit1 が!RD *　bit3 が！CD *　bit4 が RST *　チップイネーブル、予約済み、ライブラリ内で内部的に８x８フォントを設定 *　デバイスアドレスは 0 */

Spi_T6963C_writeData

原形	`void Spi_T6963C_writeData(unsigned char data);`
戻り値	なし
解説	SPI　T6963C コントローラへデータを書き込むルーチン。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_writeData(AddrL);`

Spi_T6963C_writeCommand

原形	`void Spi_T6963C_writeCommand(unsigned char data);`
戻り値	なし
解説	SPI　T6963C コントローラへコマンドを書き込むルーチン。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_writeCommand(T6963C_CURSOR_POINTER_SET);`

Spi_T6963C_setPtr

原形	`void Spi_T6963C_setPtr(unsigned int addr, unsigned char 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	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_setPtr(T6963C_grHomeAddr + start, T6963C_ADDRESS_POINTER_SET);`

Spi_T6963C_waitReady

原形	`void Spi_T6963C_waitReady();`
戻り値	なし
解説	このルーチンはステータスバイト値をプールし、レディとなるまでループする。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_waitReady();`

Spi_T6963C_fill

原形	`void Spi_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のメモリをバイト長のデータで満たす。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_fill(0x33,0x00FF,0x000F);`

Spi_T6963C_dot

原形	`void Spi_T6963C_dot(int x, int y, unsigned char color);`
戻り値	なし
解説	このルーチンは現在のグラフィック動作パネルを設定する。それはピクセルドット（x0,y0）を設定する。pcolor＝T6963C_［WHITE[BLACK］
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_dot(x0, y0, pcolor);`

Spi_T6963C_write_char

原形	`void Spi_T6963C_write_char(unsigned char c, unsigned char x, unsigned char y, unsigned char mode);`
戻り値	なし
解説	このルーチンは現在のテキスト動作パネルを設定する。 それは x 列、y 行に文字　c を書き込む。 mode＝T6963C_ROM_MODE_[OR｜EXOR｜AND]
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_write_char("A",22,23,AND);`

Spi_T6963C_write_text

原形	`void Spi_T6963C_write_text(unsigned char *str, unsigned char x, unsigned char y, unsigned char mode);`
戻り値	なし
解説	このルーチンは現在のテキスト動作パネルを設定する。 それは x 列、y 行に文字列　str を書き込む。 mode＝T6963C_ROM_MODE_[OR｜EXOR｜AND]
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_write_text("GLCD LIBRARY DEMO, WELCOME !", 0, 0, T6963C_ROM_MODE_XOR);`

Spi_T6963C_line

原形	`void Spi_T6963C_line(int px0, int py0, int px1, int py1, unsigned char pcolor);`
戻り値	なし
解説	このルーチンは現在のグラフィック動作パネルを設定する。 それは(x0,y0)から(x1,y1)へ直線を描画する。 pcolor＝T6963C_[WHITE[BLACK]
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_line(0, 0, 239, 127, T6963C_WHITE);`

Spi_T6963C_rectangle

原形	`void Spi_T6963C_rectangle(int x0, int y0, int x1, int y1, unsigned char pcolor);`
戻り値	なし
解説	このルーチンは現在のグラフィック動作パネルを設定する。 それは長方形(x0,y0)－(x1,y1)の輪郭を描画する。 pcolor＝T6963C_[WHITE[BLACK]
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_rectangle(20, 20, 219, 107, T6963C_WHITE);`

Spi_T6963C_box

原形	`void Spi_T6963C_box(int x0, int y0, int x1, int y1, unsigned char pcolor);`
戻り値	なし
解説	このルーチンは現在のグラフィック動作パネルを設定する。 それは長方形(x0,y0)－(x1,y1)の実線の箱を描画する。 pcolor＝T6963C_[WHITE[BLACK]
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_box(0, 119, 239, 127, T6963C_WHITE);`

Spi_T6963C_circle

原形	`void Spi_T6963C_circle(int x, int y, long r, unsigned char pcolor);`
戻り値	なし
解説	このルーチンは現在のグラフィック動作パネルを設定する。 それは円を描画する。中心は(x,y)、半径は r。 pcolor＝T6963C_[WHITE[BLACK]
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_circle(120, 64, 110, T6963C_WHITE);`

Spi_T6963C_image

原形	`void Spi_T6963C_image(const char *pic);`
戻り値	なし
解説	このルーチンは現在のグラフィック動作パネルを設定する。 それは MCU によりピクチャポインタでグラフィック領域を満たす。 MCU は表示ジオメトリに適合しなければならない。 例：240x128 表示器に対し、MCU は(240/8)*128＝3840 バイトの配列を持たねばならない。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_image(my_image);`

Spi_T6963C_sprite

原形	`void Spi_T6963C_sprite(unsigned char px, unsigned char py, const char *pic, unsigned char sx, unsigned char sy);`
戻り値	なし
解説	このルーチンは現在のグラフィック動作パネルを設定する。 それは MCU により指定されるウイッチピクチャでグラフィックの長方形領域(px,py)－(px+sx、py+sy)を満たす。 Sx と sy は画面のサイズでなければならない。 MCU は sx*sy バイトの配列を必要とする。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_sprite(76, 4, einstein, 88, 119);`

Spi_T6963C_set_cursol

原形	`void Spi_T6963C_set_cursor(unsigned char x, unsigned char y);`
戻り値	なし
解説	このルーチンはカーソルを列 x、行 y に設定する。
必要事項	ポートは初期化されねばならない。Spi_ T6963C _init を見よ。
例	`Spi_T6963C_set_cursor(cposx, cposy);`

Spi_T6963C_clearBit

原形	`void Spi_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	ポートは初期化されねばならない。Spi_ T6963C _init を見よ。
例	`Spi_T6963C_clearBit(b);`

Spi_T6963C_setBit

原形	`void Spi_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	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_setBit(b);`

Spi_T6963C_negBit

原形	`void Spi_T6963C_negBit(char b);`
戻り値	なし
解説	コントロールビットを否定する。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_negBit(b);`

Spi_T6963C_displayGrPanel

原形	`void Spi_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
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_displayGrPanel(n);`

Spi_T6963C_displayTxtPanel

原形	`void Spi_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
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_displayTxtPanel(n);`

Spi_T6963C_displaysetGrPanel

原形	`void Spi_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スタートアドレスを計算する。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_setGrPanel(n);`

Spi_T6963C_setTxtPanel

原形	`void Spi_T6963C_setTxtPanel(unsigned int n);`
戻り値	なし
解説	パネル番号ｎに対するテキストスタートアドレスを計算する。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_setTxtPanel(n);`

Spi_T6963C_panelFill

原形	`void Spi_T6963C_panelFill(unsigned int v);`
戻り値	なし
解説	全ての#n パネルをビットマップ v で満たす。（0 にクリア）
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_panelFill(v);`

Spi_T6963C_grFill

原形	`void Spi_T6963C_grFill(unsigned int v);`
戻り値	なし
解説	グラフィック#n パネルをビットマップ v で満たす。（0 にクリア）
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_grFill(v);`

Spi_T6963C_txtFill

原形	`void Spi_T6963C_txtFill(unsigned int v);`
戻り値	なし
解説	テキスト#n パネルを文字 v+32 で満たす。（0 にクリア）
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_txtFill(v);`

Spi_T6963C_cursol_height

原形	`void Spi_T6963C_cursor_height(unsigned int n);`
戻り値	なし
解説	カーソルサイズを設定する。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_cursor_height(n);`

Spi_T6963C_graphics

原形	`void Spi_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。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_graphics(1);`

Spi_T6963C_text

原形	`void Spi_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。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_text(1);`

Spi_T6963C_cursol

原形	`void Spi_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	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_cursor(1);`

Spi_T6963C_cursol_blink

原形	`void Spi_T6963C_cursor_blink(unsigned int n);`
戻り値	なし
解説	カーソル点滅をオン／オフする。
必要事項	GLCD は初期化されねばならない。Spi_ T6963C _Config を見よ。
例	`Spi_T6963C_cursor_blink(0);`

Spi_T6963C_config_240x128

原形	`procedure Spi_T6963C_Config_240x128();`
戻り値	なし
解説	mE　GLCD のためのデフォルト設定で GLCD（240 x 128 ピクセル）を基本とした T6963C を初期化する。
必要事項	Spi_Init ; が SPI 東芝 T6963C グラフィック LCD を初期化する前に呼び出さねばならない。
例	`Spi_T6963C_Config_240x128();`

Spi_T6963C_config_240x64

原形	`procedure Spi_T6963C_Config_240x64();`
戻り値	なし
解説	グラフィックのオン／オフを設定する
必要事項	mE　GLCD のためのデフォルト設定で GLCD（240 x 64 ピクセル）を基本とした T6963C を初期化する。
例	`Spi_T6963C_Config_240x64();`

Library　Examples

次の描画デモは SPI　T6963C　GLCD の高度な試験をする。

```
#include        "Spi_T6963C.h"

extern const char mc[] ;
extern const char einstein[] ;

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...
```

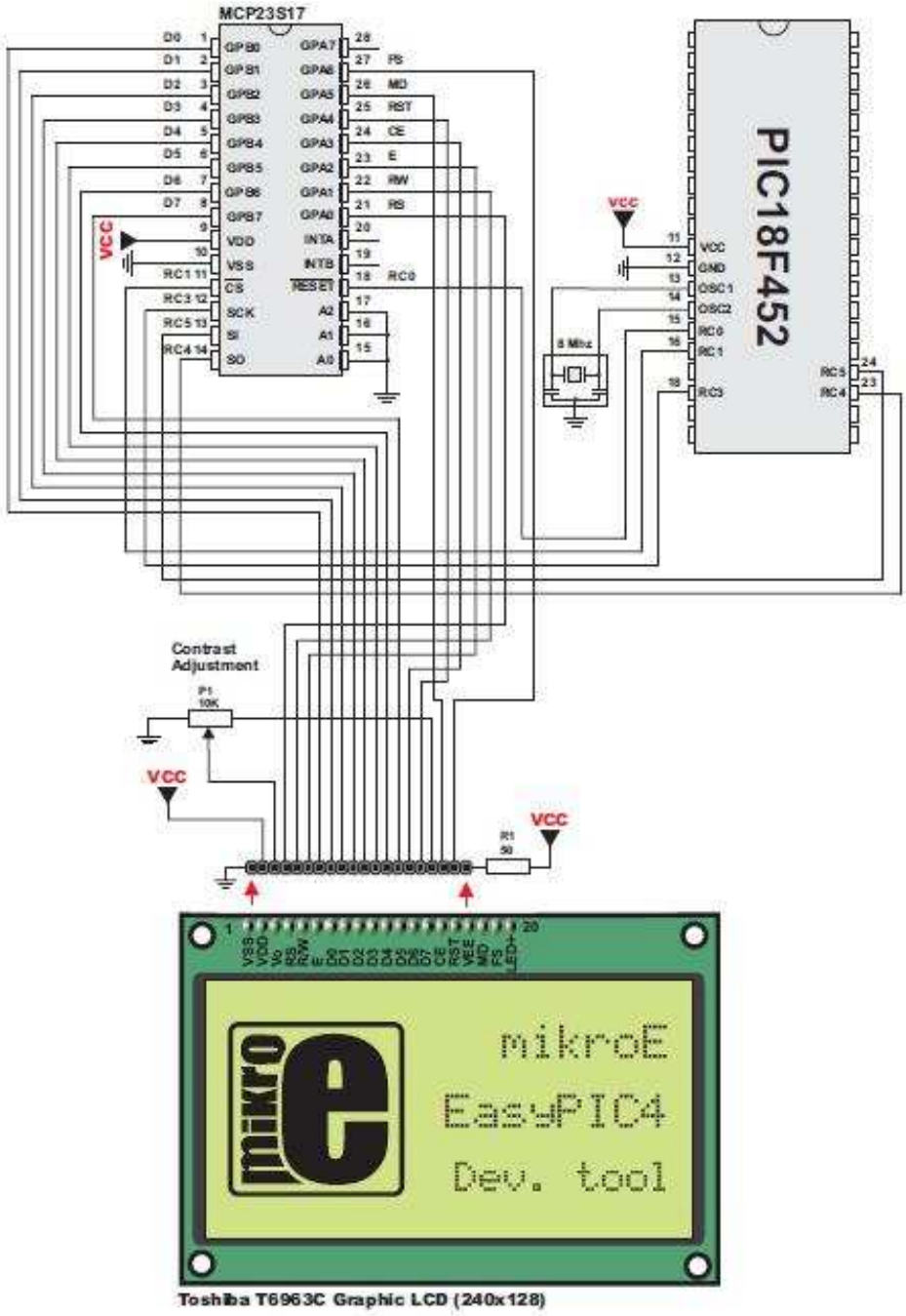
```
//continues...

        /*
         * 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 2 is !RD
         * bit 1 is C!D
         * bit 5 is RST
         */

        Spi_Init();
        Spi_T6963C_Init_240x128();

        /*
         * enable both graphics and text display at the same time
         */

        Spi_T6963C_graphics(1) ;
        Spi_T6963C_text(1) ;

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

        /*
         * text messages
         */

        Spi_T6963C_write_text(" GLCD LIBRARY DEMO, WELCOME !", 0,
0, Spi_T6963C_ROM_MODE_XOR) ;
        Spi_T6963C_write_text(" EINSTEIN WOULD HAVE LIKED mC", 0,
15, Spi_T6963C_ROM_MODE_XOR) ;

        /*
         * cursor
         */

        Spi_T6963C_cursor_height(8) ;        // 8 pixel height
        Spi_T6963C_set_cursor(0, 0) ;  // move cursor to top left
        Spi_T6963C_cursor(0) ;               // cursor off


//continued...
```

```
//continues...

        /*
         * draw rectangles
         */

        Spi_T6963C_rectangle(0, 0, 239, 127, Spi_T6963C_WHITE) ;
        Spi_T6963C_rectangle(20, 20, 219, 107, Spi_T6963C_WHITE) ;
        Spi_T6963C_rectangle(40, 40, 199, 87, Spi_T6963C_WHITE) ;
        Spi_T6963C_rectangle(60, 60, 179, 67, Spi_T6963C_WHITE) ;

        /*
         * draw a cross
         */

        Spi_T6963C_line(0, 0, 239, 127, Spi_T6963C_WHITE) ;
        Spi_T6963C_line(0, 127, 239, 0, Spi_T6963C_WHITE) ;

        /*
         * draw solid boxes
         */

        Spi_T6963C_box(0, 0, 239, 8, Spi_T6963C_WHITE) ;
        Spi_T6963C_box(0, 119, 239, 127, Spi_T6963C_WHITE) ;

        /*
         * draw circles
         */

        Spi_T6963C_circle(120, 64, 10, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 30, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 50, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 70, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 90, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 110, Spi_T6963C_WHITE) ;
        Spi_T6963C_circle(120, 64, 130, Spi_T6963C_WHITE) ;

        // draw a sprite
        Spi_T6963C_sprite(76, 4, einstein, 88, 119) ;


        Spi_T6963C_setGrPanel(1) ;  // select other graphic panel

        // fill the graphic screen with a picture
        Spi_T6963C_image(mc) ;


//continued...
```

```
//continues...

        for(;;)
                {

// if RB1 is pressed, toggle the display between graphic panel 0
// and graphic 1

                if(PORTB & 0b00000010)
                        {
                        panel++ ;
                        panel &= 1 ;
                        Spi_T6963C_displayGrPanel(panel) ;
                        Delay_ms(300) ;
                        }

                // if RB2 is pressed, display only graphic panel
                else if(PORTB & 0b00000100)
                        {
                        Spi_T6963C_graphics(1) ;
                        Spi_T6963C_text(0) ;
                        Delay_ms(300) ;
                        }

                // if RB3 is pressed, display only text panel

                else if(PORTB & 0b00001000)
                        {
                        Spi_T6963C_graphics(0) ;
                        Spi_T6963C_text(1) ;
                        Delay_ms(300) ;
                        }


            // if RB4 is pressed, display text and graphic panels

                else if(PORTB & 0b00010000)
                        {
                        Spi_T6963C_graphics(1) ;
                        Spi_T6963C_text(1) ;
                        Delay_ms(300) ;
                        }


//continued...
```

```
//continues...

                    // if RB5 is pressed, change cursor
                    else if(PORTB & 0b00100000)
                            {
                            curs++ ;
                            if(curs == 3) curs = 0 ;
                            switch(curs)
                                    {
                                    case 0:
                                            // no cursor
                                            Spi_T6963C_cursor(0) ;
                                            break ;
                                    case 1:
                                            // blinking cursor
                                            Spi_T6963C_cursor(1) ;
                                        Spi_T6963C_cursor_blink(1) ;
                                            break ;
                                    case 2:
                                            // non blinking cursor
                                            Spi_T6963C_cursor(1) ;

                                        Spi_T6963C_cursor_blink(0) ;
                                            break ;
                                    }
                            Delay_ms(300) ;
                            }

                    /*
                     * move cursor, even if not visible
                     */
                    cposx++ ;
                    if(cposx == Spi_T6963C_txtCols)
                            {
                            cposx = 0 ;
                            cposy++ ;
    if(cposy == Spi_T6963C_grHeight / Spi_T6963C_CHARACTER_HEIGHT)
                                    {
                                    cposy = 0 ;
                                    }
                            }
                    Spi_T6963C_set_cursor(cposx, cposy) ;

                    Delay_ms(100) ;
                    }
            }
```

Hardware Connection



Toshiba T6963C Graphic LCD (240x128)

Setjmp　Library

　このライブラリは、通常の関数呼出しと戻り値の規則をバイパスするための、関数と型定義を含む。
宣言された型は、呼出し環境を復活するために必要とされる情報を保持するのに適した配列型の jmp_buf である。

型宣言には、pic16 と pic18 ファミリ mcus のための setjmp16.h 及び setjmp18.h ヘッダーファイルがそれぞれ含まれる。
これらのヘッダーはコンパイラのインクルード・フォルダ内に見出される。
このライブラリの実行は pic16 と pic18 ファミリ mcus とでは異なる。
pic16 ファミリに対しては、Setjmp と Longjmp は setjmp16 ヘッダーファイル内で定義されたマクロとして実行され、pic18 対しては、setjmp ライブラリファイル内で定義した関数として実行される。
スタックポインタをリード／ライトすることはできないという pic16 ファミリの特性により、Longjmp 呼出しが発生した後、プログラムの実行はスタックの内容に依存する。
そういうわけで、pic16 ファミリのみ、setjmp と longjmp 関数の実行が ANSI　C 標準準拠ではないのである。

Library Routines

```
Setjmp
Longjmp
```

Setjmp

原形	`int setjmp(jmp_buf env);`
戻り値	もし戻り値が直接呼出しからならば、0 を返す。 もし戻り値が longjmp への呼出しからならば、非ゼロ値を返す。
解説	この関数は longjmp による後の使用のために jmp_buf 内の呼び出し位置を保存する。 変数 env は、呼出し環境を復活するために必要とされる情報を保持するのに適した配列型（jmp_buf）である。
必要事項	なし
例	`setjmp(buf);`

Longjmp

原形	`void longjmp(jmp_buf env, int val);`
戻り値	Longjmp は val を返すため setjmp に影響をもたらす。 もし val が 0 ならば、1 を返すであろう。
解説	Setjmp マクロの最新の呼出しにより、jmp_buf 内に保存した呼出し環境を復活する。 もしそれらがそのような呼出しをしていない、さもなくば setjmp 呼出しを含む関数が、仮に終了していたならば、その動作は未定義とされる。 変数 env：setjmp 呼出しに対応することにより、保存された情報を保持するための配列型（jmp_buf）である。 変数 val は char 型の値であり、それは setjmp に対応するものを返すであろう。
必要事項	Lonjmp 呼出しは、呼び出された setjmp が遭遇する関数から戻る前に発生しなければならない。
例	`longjmp(buf, 2);`

Library　Example

例は、setjmp と Longjmp 関数を使用しての、関数の相互呼出しを実際に説明するものである。
呼び出されると、Setjmp()は Longjmp()により後に使用するため、その jmp_buf 引数にその呼出し環境を保存する。
一方、Longjmp()は対応する jmp_buf 引数で Setjmp()の最新の呼出しにより保存した環境を復活する。
この例は mikroC　Setjmp　example フォルダ内で見つけることが可能である。

Time　Library

　タイムライブラリはＵＮＩＸタイムフォーマット内の時間計算のための関数と型定義を含む。
ＵＮＩＸタイムは“ｅｐｏｃｈ”から秒数をカウントする。
これは時分割で動作するプログラムにとって非常に便利である。つまり、２つのＵＮＩＸ時間値の差は、ローカル時計の精度内で、秒単位で計測された実時間の相違である。

“ｅｐｏｃｈ”とは何か？

　元来、それは１９７０年GMT（１９７０年１月１日ユリウス日）を始りとして定義された。
GMT、Ｇｒｅｅｎｗｉｃｈ　Ｍｅａｎ　Ｔｉｍｅとはイングランドにおける時間帯に対する伝統的な時間間隔である。
宣言された型は、時間と日の記録に適合した構造型であるＴｉｍｅＳｔｒｕｃｔである。
型宣言は、mikroC タイムライブラリ　Ｄｅｍｏ　ｅｘａｍｐｌｅフォルダ内に見つけることが可能な、ｔｉｍｅｌｉｂ．ｈ内に含まれる。

Library　Routines

```
Time_dateToEpoch
Time_epochToDate
Time_dateDiff
```

Time_dateToEpoch

原形	`long Time_dateToEpoch(TimeStruct *ts);`
戻り値	この関数は秒数を返す。
解説	この関数は unix　epoch を返す。ts で指定される時間構造体の１９７０年１月１日 0h00mn00s からの秒数
必要事項	なし。
例	`Time_dateToEpoch(&ts1);`

Time_epochToDate

原形	`void Time_epochToDate(long e, TimeStruct *ts);`
戻り値	この関数は 1970 年からの秒数を返す。
解説	unix epoch e（１９７０年からの秒数）を時間構造体 ts に変換する。
必要事項	なし。
例	`Time_epochToDate(epoch, &ts2);`

Time_dateDiff

原形	`long Time_dateDiff(TimeStruct *t1, TimeStruct *t2);`
戻り値	この関数は符号有り long として秒単位の時間差を返す。
解説	この関数は２つの日を比較し、符号有り long として秒単位の時間差を返す。 結果は、もし t1 が t2 の前なら正の値となり、t1 が t2 と同じならヌルであり、t1 が t2 の後なら負となる。
必要事項	注意 : この関数は、mikroC Time Library Demo example フォルダ内で見つける事が可能な timelib.h ファイル内のマクロとして実行される。
例	`diff = Time_dateDiff(&ts1, &ts2);`

Library　Example

例はTimeLibraryDemo（PIC　MCU用の簡略化されたｃライクなタイムライブラリ）を説明するものである。

```
#include        "timelib.h"

TimeStruct      ts1, ts2 ;
long    epoch ;
long    diff ;

void main()
        {

        ts1.ss = 0 ;
        ts1.mn = 7 ;
        ts1.hh = 17 ;
        ts1.md = 23 ;
        ts1.mo = 5 ;
        ts1.yy = 2006 ;

        /*
         * what is the epoch of the date in ts ?
         */
        epoch = Time_dateToEpoch(&ts1) ;

        /*
         * what date is epoch 1234567890 ?
         */
        epoch = 1234567890 ;
        Time_epochToDate(epoch, &ts2) ;

        /*
         * how much seconds between this two dates ?
         */
        diff = Time_dateDiff(&ts1, &ts2) ;
        }
```

Contact us

もしあなたが何らかのわれわれの製品で問題を体験している、又は追加情報を望むなら、われわれに知らせてください。

コンパイラの技術サポート

もしあなたが mikroC で何らかのトラブルを体験しているなら、どうか恥じる事無くわれわれに連絡を取って欲ください。それらの問題を解決することは、われわれ相互の利益になります。

学校及び大学のためのディスカウント（値引き）

mikroElectronika は、教育機関のために特別値引きを提供します。
もしあなたが mikroC を純粋な教育目的で購入したいならば、われわれに連絡してください。

輸送と納期の問題

もしあなたが、納期遅れ、またはわれわれの製品の配達に関する何らかの他の問題を報告したいなら、以下に示すリンクを使用してください。

MikroErektronika の販売業者になりたいですか？

MikroElektronika では、われわれは新しいパートナーを探していいます。もしあなたがわれわれの製品の販売者となってわれわれの手助けをしたいと考えるなら、われわれに知らせてください。

その他

もしあなたが何か他の質問、意見、提案があるなら、われわれに連絡を取ってください。

mikroElektronika
Admirala Geprata 1B
11000 Belgrade
EUROPE

Phone: + 381 (11) 30 66 377, + 381 (11) 30 66 378
Fax: + 381 (11) 30 66 379
E-mail: office@mikroe.com
Website: www.mikroe.com