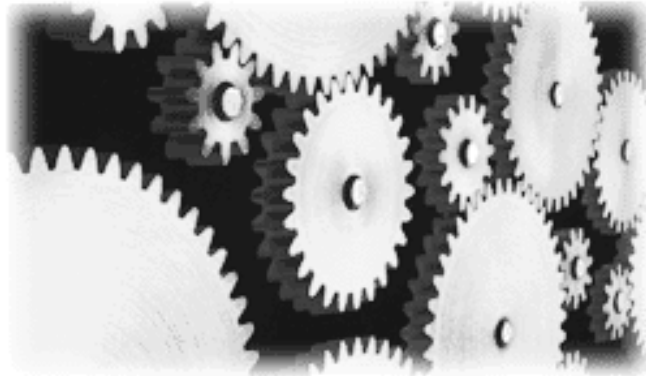


# Programmation C sur PIC (EasyPIC5 - MikroC)

Geii - S2 - II2 - Travaux pratiques

7 juin 2010

F. Morain-Nicolier  
<http://pixel-shaker.fr>



# Table des matières

<b>1</b>	<b>Prise en main</b>	<b>2</b>
1.1	Préambule . . . . .	2
1.2	Observation de la maquette . . . . .	2
1.3	Création d'un premier projet . . . . .	3
1.4	Exécution pas-à-pas, déboguage . . . . .	3
<b>2</b>	<b>Des leds, des leds</b>	<b>5</b>
2.1	Rappels . . . . .	5
2.2	Exemple . . . . .	5
2.3	Exercices . . . . .	5
<b>3</b>	<b>Ports en entrée/sortie</b>	<b>6</b>
3.1	Sélection des E/S . . . . .	6
3.2	Scrutation de niveau . . . . .	7
3.3	Attente de fronts . . . . .	7
3.3.1	Changement d'état d'une diode . . . . .	7
3.3.2	Compteur . . . . .	7
3.4	En attendant la fin du TP . . . . .	7
<b>4</b>	<b>Multiplexage - afficheurs sept segments</b>	<b>8</b>
4.1	Matériel . . . . .	8
4.2	Logiciel - test . . . . .	8
4.3	Logiciel - multiplexage . . . . .	9
4.4	Logiciel - avec interruptions . . . . .	9
4.5	Si vous avez encore du temps . . . . .	10
<b>5</b>	<b>Afficheur GLCD - capteur de température DS1820 - liaison <i>one-wire</i></b>	<b>11</b>
5.1	Affichage sur le GLCD . . . . .	11
5.2	Communication <i>one-wire</i> avec le capteur DS1820 . . . . .	12
5.2.1	Mise en place . . . . .	12
5.2.2	Communication . . . . .	13
5.2.3	Décodage de la température . . . . .	14
5.3	Supplément . . . . .	14
<b>6</b>	<b>Timer et interruption - mesure de temps et production de signaux périodiques</b>	<b>15</b>
6.1	Éléments de documentation . . . . .	15
6.2	Test de fonctionnement . . . . .	16
6.3	Mesure de durée d'exécution d'un morceau de code . . . . .	17
6.4	Production d'un signal périodique . . . . .	17
<b>7</b>	<b>Interfaçage d'un écran tactile</b>	<b>18</b>

# TP 1

## Prise en main

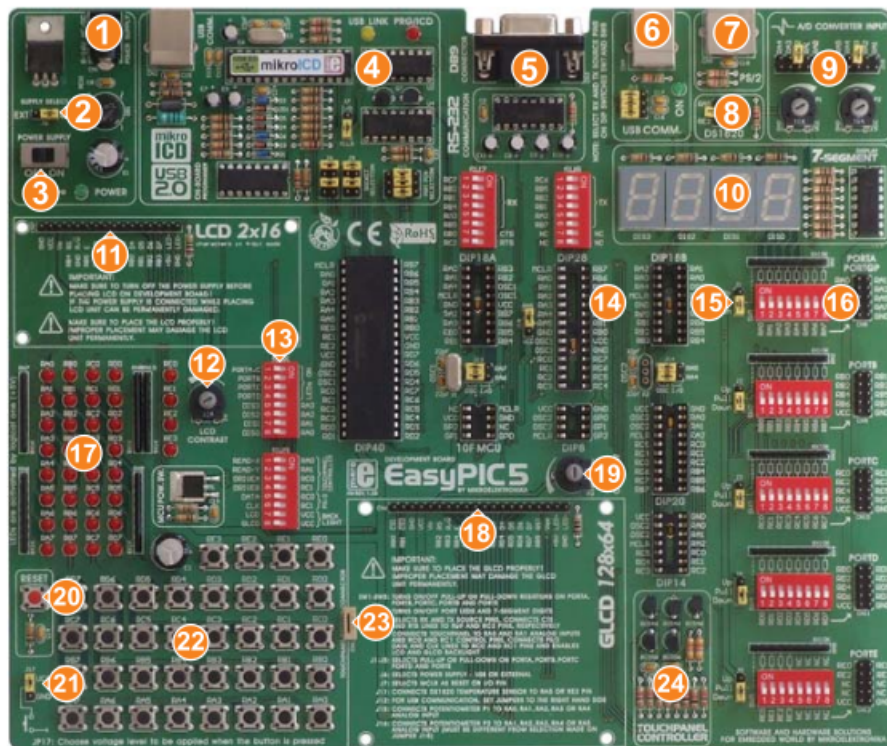
**Objectifs** Identifier les éléments de la maquette, créer un projet, le compiler et l'exécuter sur la maquette, observer les fichiers créés, tracer le programme et des variables.

### 1.1 Préambule

- une documentation est disponible dans `C:\doc_mikroc for PIC\` ;
  - sur le site du constructeur (<http://www.mikroe.com/>), vous pourrez consulter divers éléments de documentation et télécharger une version d'essai de l'environnement (IDE et compilateur).
- Vous devez vous connecter en usant le *login* `ge1` (sans mot de passe).

### 1.2 Observation de la maquette

Voici une vue de maquette comportant des numéros.



**Manipulation** Identifier les éléments suivant sur cette reproduction (donner les numéros) :

- *Switch* d'alimentation,
- programmeur USB (*mikroICD*),
- micro-contrôleur PIC,
- bouton de *reset*,
- diodes de contrôle des ports A à E,
- boutons poussoir de test de ports A à E,
- afficheurs 7 segments multiplexés,

## 1.3 Création d'un premier projet

Vous disposez d'un petit fascicule (en anglais) intitulé *Creating the first project in mikroC for PIC*. Ce texte vous indique la marche à suivre pour créer et compiler un projet.

**Manipulation** Suivre les instructions (jusqu'à l'exécution - *run*) avec les recommandations suivantes :

- Nom du projet : **tp1a**
- Chemin du projet : D:\rep\_perso\tp1. Ce dossier doit être créé en remplaçant **rep\_perso** par un nom qui vous est propre.
- Le type (*device*) de PIC est à lire sur la puce elle-même.
- Le programme à saisir est le suivant :

```
void main() {
    PORTC = 0; // Init PORTC
    TRISC = 0; // PORTC as output
    while(1) {
        PORTC = ~PORTC; // toggle PORTC
        Delay_ms(1000); // 1s delay
    }
}
```

**Manipulation**

- Aller dans le dossier de votre projet.
- Ouvrir les fichiers **.hex**, **.mcl**, **asm**, **.lst**. Que contiennent ces fichiers, quelle est leur utilité?
- Par quelles instructions assembleur sont traduites les lignes suivantes?
  - `PORTC = 0;`
  - et `TRISC = 0;`

## 1.4 Exécution pas-à-pas, déboguage

Créer un nouveau projet (toujours dans votre dossier **tp1**) nommé **tp1b**. Le programme à saisir est le suivant :

```
void main() {
    int k;
    PORTC = 0; // Init PORTC
    TRISC = 0; // PORTC as output
    for (k = 0; k < 256; k++) {
        PORTC = k;
    }
}
```

**Manipulation** Pour utiliser le débogueur (voir la photocopie jointe) :

- Modifier les options du projet pour activer le débogueur ICD (dans *Project Setup Window*, sur la gauche) :
  - cocher **mikroICD Debug** sous **Build Type**,
  - vérifier que **mikroICD Debugger** est activé sous **Debugger**.
- Compiler le projet (*Build* - CTRL+F9).
- Programmer la puce (*Program* - F11).
- Lancer le débogueur (*Start Debugger* - F9)

**Manipulation** Exploiter le débogueur pour :

- Suivre en pas-à-pas l'exécution du programme (*Step Into* par exemple). Vérifier l'allumage correct des diodes du port C.
- Ajouter le suivi des variables PORTC et k lorsque vous êtes en mode pas-à-pas. Contrôler la bonne évolution des valeurs de ces variables.
- Expérimenter les autres possibilités du débogueur.

Name	Description	Function key
Debug	Starts Debugger.	[F9]
Run/Pause debugger	Runs or pauses program execution and debugging.	[F6]
Toggle Breakpoints	During debugging, program is executed until a breakpoint is reached. At that point, the <i>Toggle Breakpoints</i> option sets new breakpoints or removes those already set at the current cursor position. To view all the breakpoints, select <i>Run &gt;View Breakpoints</i> from the drop-down menu. Double click on an item from the list locates the breakpoint.	[F5]
Run to cursor	Program is executed until the cursor position is reached.	[F4]
Step Into	Executes the current C/Pascal/Basic (single- or multi-cycle) program line, then halts. If the program line is a routine call, steps into the routine and halts at the first instruction within it.	[F7]
Step Over	Executes the current C/Pascal/Basic (single- or multi-cycle) program line, then halts. If the program line is a routine call, enters the routine and halts at the first instruction following the call.	[F8]
Flush RAM	Flushes current PIC microcontroller RAM. All RAM memory values will be changed according to values in the <i>Watch</i> window.	-
Stop Debugger	Stops Debugger.	[Ctrl+F2]
Step Out	Executes all remaining program lines within the subroutine. It halts immediately upon exit from the subroutine.	[Ctrl+F8]
Disassembly View	Instead of program written in high-level program language ( <i>Basic</i> in this very case), the assembly version of the same program will appear. Each program line written in Basic is divided in assembly instructions executed by the microcontroller.	[Alt+D]

# TP 2

## Des leds, des leds

### 2.1 Rappels

On rappelle qu'en C le «ou booléen» se fait par `||`, le «et booléen» par `&&`. Nous aurons besoin du «ou bit à bit» `|` et du «et bit à bit» `&`. Soit le contenu d'un registre `B` sur huit bits,

b7	b6	b5	b4	b3	b2	b1	b0
1	1	1	0	0	0	1	1

#### Préparation

1. Vous désirez mettre à 1 le bit b2 sans changer les autres bits, comment faites-vous ?
2. Vous désirez mettre à 0 le bit b6 sans changer les autres bits comment faites-vous ?

### 2.2 Exemple

On vous donne un programme C qui fait clignoter une led (poids faible mais à gauche) sur le port C.

```
void main(void){
    TRISC = 0; // Tous les bits du PORTC en sortie.
    PORTC = 0;
    while(1) {
        PORTC = 0x01;
        Delay_ms(1000);
        PORTC = 0x00;
        Delay_ms(1000);
    }
}
```

#### Manipulation

Écrire ce programme pour l'essayer, le compiler, le charger et l'exécuter. Modifier-le pour faire clignoter la led RC1.

### 2.3 Exercices

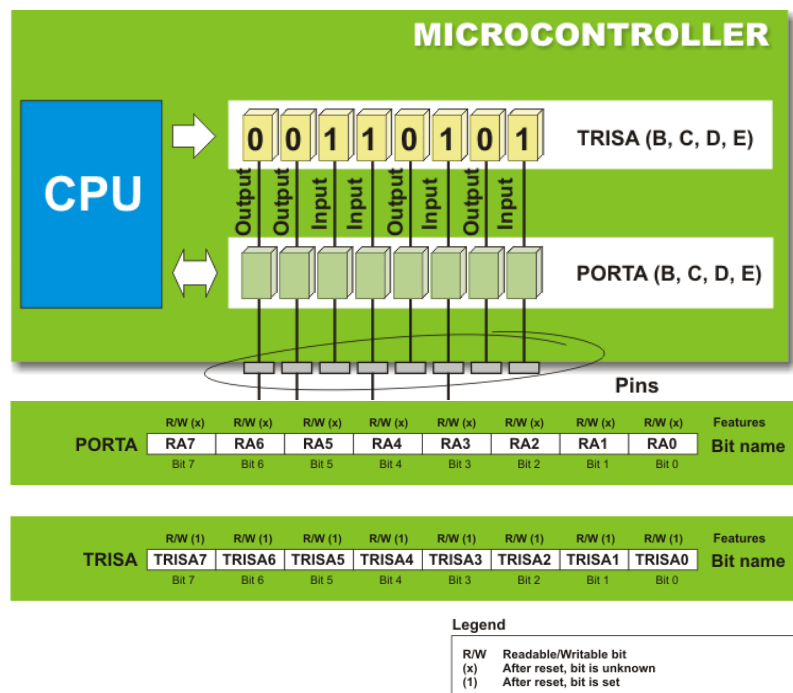
1. Écrire un chenillard simple : une led se déplaçant sur le PORTC (de haut en bas) et en utilisant le même type de temporisation que dans le programme exemple.
2. Écrire un chenillard double : un chenillard de haut en bas et simultanément un autre de bas en haut qui se croisent.
3. Écrire un chenillard à entassement (1 led se déplaçant et s'accumulant en bas).

**Note :** les opérateurs de décalage sont `>>` et `<<` pour respectivement le décalage à droite et le décalage à gauche.

# TP 3

## Ports en entrée/sortie

Les ports A, B, C, D et E sont des ports d'entrée/sortie dont chaque bit peut-être utilisé soit en entrée, soit en sortie, de façon indépendante. Ainsi chaque port possède un satellite : TRISA, TRISB, TRISC, TRISD et TRISE qui permet de déterminer le sens de chaque bit (**0** : *Output*, **1** : *Input*).



Figures issues de *PIC Microcontrollers - Programming in C*<sup>1</sup>.

par exemple :

```
void main (void){
    ANSEL = ANSELH = 0; // Toutes le ports E/S sont configurés comme numériques
    PORTA = 0; // RAZ des broches du port A
    TRISA = 0b00000100; // Toutes les broches du port A sauf PORTA.2 sont des sorties
}
```

### 3.1 Sélection des E/S

Écrire un programme qui positionne en entrée les 4 bits de poids faibles du port B et en sortie les autres. Le programme doit ensuite en permanence recopier les 4 bits de poids faibles vers les 4 bits de poids forts.

1. <http://www.mikroe.com/en/books/pic-books/mikroc/>

## 3.2 Scrutation de niveau

Écrire un programme qui comporte les éléments suivants.

1. Un compteur binaire sur le port B (256 états). Placer une temporisation de 100ms entre chaque état.
2. Un appui sur PORTA.2 doit remettre à zéro le compteur.
3. Les accès individuels aux bits seront effectués à l'aide de masques.

Modifier ensuite le programme pour que le comptage n'ai lieu que lorsque PORTA.1 est à zéro.

## 3.3 Attente de fronts

### 3.3.1 Changement d'état d'une diode

Réaliser un programme avec le cahier des charges suivant.

1. En début de programme la diode `PORTB.0` doit être allumée.
2. Ensuite, un front montant appliqué sur `PORTA.0` provoquera un changement d'état de la diode.
3. Les accès au bits seront effectués à l'aide des accès directs (en écrivant `PORTB.F2` pour accéder au bit 2 du port B par exemple). Puis dans un second temps avec des masques.

### 3.3.2 Compteur

Reprendre le compteur précédent (*cf* scrutation par niveau 3.2) avec les changements suivants (un changement à la fois).

1. Un front descendant sur `PORTA.2` provoquera la RAZ du compteur.
2. Puis, seul un front montant sur `PORTA.1` incrémentera le compteur.

Que pensez-vous de la gestion de plusieurs attentes de front ?

## 3.4 En attendant la fin du TP

Soit le programme suivant :

```
void main (void){
  ANSEL = ANSELH = 0; // Toutes le ports E/S sont configurés comme numériques
  PORTA = 0x55;
}
```

1. Quelles sont les diodes allumées du port A ?
2. Proposer un schéma compatible avec cette constatation.
3. Proposer un schéma où les diodes seraient allumées en positionnant les bits à 0 du port.



```
// Fichier principl
#include "Display_utils.h"

void main () {
    unsigned short i;
    INTCON = 0; // Disable GIE, PEIE, INTE, RBIE, TOIE
    TRISA = 0;
    PORTA = 0;
    TRISD = 0;
    PORTD = 0;
    while (1) {
        for (i = 0; i <= 9u; i++){
            PORTA = 0; // Turn off all 7seg displays
            PORTD = mask(i); // bring appropriate value to PORTD
            PORTA = 1; // turn on appropriate 7seg. display
            Delay_ms(1000);
        }
    }
}
```

```
// Fichier Display_utils.h

unsigned short mask(unsigned short num) {
    switch (num) {
        case 0 : return 0x3F;
        case 1 : return 0x06;
        case 2 : return 0x5B;
        case 3 : return 0x4F;
        case 4 : return 0x66;
        case 5 : return 0x6D;
        case 6 : return 0x7D;
        case 7 : return 0x07;
        case 8 : return 0x7F;
        case 9 : return 0x6F;
    }
}
```

1. Compiler et exécuter le programme. Que fait ce programme ?
2. Le modifier pour afficher le compteur sur DIS1, puis sur DIS2, puis sur DIS3.

### 4.3 Logiciel - multiplexage

L'objectif est d'afficher un compteur sur DIS1 et DIS0 - donc de 00 à 99. Il sera **indispensable** de basculer entre les deux afficheurs selon l'algorithme suivant :

```
Faire plusieurs fois:
- activer DIS0
- afficher le chiffre de poids faible
- tempo
- activer DIS1
- afficher le chiffre de poids fort
- tempo
```

1. Écrire et tester un programme qui affiche un compteur de 00 à 99 sur les deux afficheurs - en s'appuyant sur l'algorithme donné.
2. Modifier les valeurs des temporisation - la méthode est-elle robuste ? simple ?

### 4.4 Logiciel - avec interruptions

Voici une autre méthode pour réaliser le multiplexage. Le fichier est téléchargeable à l'adresse suivante : <http://pixel-shaker.fr/fr/enseignements/geii-programmation-pic-en-c-easypic5-mikroc>.

```

/*
 * Project name:
 *   Display7Seg_02 (Usage of 2 7Seg. displays)
 * Copyright:
 *   (c) MikroElektronika, 2005-2008.
 * Description:
 *   This code demonstrates displaying numbers (0,1,2..99) on two 7-segment
 *   displays. Each digit is on for 1 second.
 * Test configuration:
 *   MCU:          PIC16F877A
 *   Dev.Board:    EasyPIC5
 *   Oscillator:   HS, 08.0000 MHz
 *   Ext. Modules: -
 *   SW:          mikroC v8.0
*/

#include "Display_utils.h"

unsigned short digit_no, digit10, digit1, digit, i;

void interrupt() {
  if (digit_no==0) {
    PORTA = 0;          // Turn off all 7seg displays
    PORTD = digit1;    // send mask for ones digit to PORTD
    PORTA = 1;          // turn on 1st 7 seg., turn off 2nd
    digit_no = 1;
  } else {
    PORTA = 0;          // Turn off all 7seg displays
    PORTD = digit10;   // send mask for tens digit to PORTD
    PORTA = 2;          // turn on 2nd 7 seg., turn off 1st
    digit_no = 0;
  }
  TMR0 = 0;            // clear TMRO
  INTCON = 0x20;       // clear TMR0IF and set TMR0IE
}

void main() {
  OPTION_REG = 0x80;   // Timer0 settings
  TMR0       = 0;
  INTCON     = 0xA0;   // Disable PEIE,INTE,RBIE,T0IE
  PORTA     = 0;      // clear PORTA (make sure both displays are off)
  TRISA     = 0;      // designate PORTA pins as output
  PORTD     = 0;      // clear PORTD (make sure LEDs are off)
  TRISD     = 0;      // designate PORTD pins as output
  do {
    for (i = 0; i<=99; i++) {          // count from 0 to 99
      digit   = i % 10u;
      digit1  = mask(digit);           // prepare ones digit
      digit   = (char)(i / 10u) % 10u;
      digit10 = mask(digit);           // prepare tens digit
      Delay_ms(1000);
    }
  } while (1);                        // endless loop
}

```

1. Tester le programme.
2. Le comparer avec la version précédente - en particulier, comment est réalisé le basculement ?
3. Expliquer le fonctionnement de ce programme - en donnant son algorithme par exemple.

## 4.5 Si vous avez encore du temps

1. Modifier le programme pour stocker les valeurs des segments dans un tableau (au lieu d'utiliser une fonction).

## TP 5

# Afficheur GLCD - capteur de température DS1820 - liaison *one-wire*

### 5.1 Affichage sur le GLCD

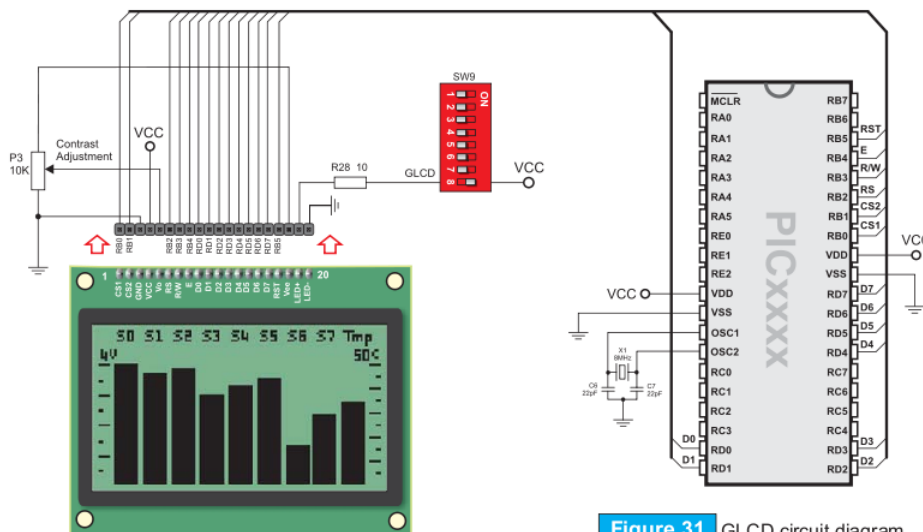


Figure 31 GLCD circuit diagram

Tester le programme suivant :

```
void my_glcd_init(){
    ANSEL = ANSELH = 0; // AN pins as digital
    Glcd_Init(&PORTB, 0, 1, 2, 3, 5, 4, &PORTD);
    Glcd_Set_Font(FontSystem5x8, 5, 8, 0x20); // Sélection police de caractères
    Glcd_Fill(0x00);
}
void main(){
    my_glcd_init();
    Glcd_Write_Text("Hello world!", 0, 0, 1);
}
```

1. Écrire "Hello world!" en noir sur fond blanc. Pour cela consulter la documentation des fonctions `Glcd_Fill` et `Glcd_Write` dans l'aide intégrée (*QHelp*).
2. Modifier le programme pour placer (approximativement) la phrase au milieu de l'écran.
3. La fonction `sprintf` est fréquemment utilisée pour formater un affichage. Elle s'utilise comme suit (voir l'aide) :

```
sprintf(&chaine, format, arg1, arg2, ...)
```

où `chaîne` est une chaîne de caractères (*i.e.* un tableau de `char`) qui sera modifiée. `format` est une chaîne de caractères contenant des caractères ordinaires et des spécifications de format du type `%0[taille][type]`, `[taille]` étant le nombre de chiffres utilisés pour l’affichage et `[type]` étant `d` pour des entiers signés et `u` pour des entiers non-signés. La fonction `sprintf` ne fonctionne qu’avec des `int`.

Compléter le programme suivant pour afficher "t = 20,5"

```

----- text[10];
void main(){
    unsigned int val = 20;
    unsigned int dec = 5;
    my_glcd_init();
    sprintf(text, "t = %d,%d", val, dec);
    Glcd_Write_Text(text, 0, 0, 1);
}

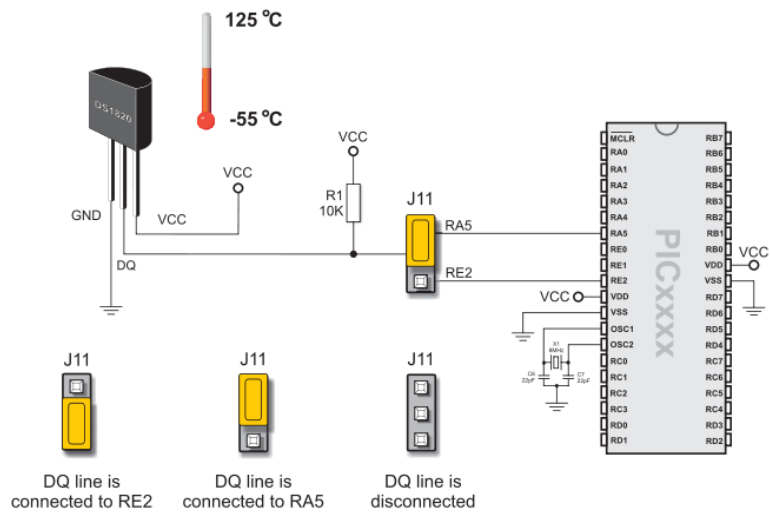
```

## 5.2 Communication *one-wire* avec le capteur DS1820

DS1820 digital thermometer is convenient for environmental temperature measurement. It can measure temperature in the range between -55°C and 125°C with 0.5°C accuracy. It must be properly placed in the 3-pin socket provided on the EasyPIC5, with its rounded side directed to the right, as marked on the board (refer to the Figure 41 below). Otherwise, the DS1820 could be permanently damaged. The DS1820 data pin can be connected to either RA5 or RE2 pin, which is determined by the jumper J11.



Figure 40 DS1820 connector



Le capteur de température DS1820 s’appuie sur le protocole *one-wire* pour communiquer sa mesure. Comme son nom l’indique, un seul fil est nécessaire (même si plusieurs périphériques sont utilisés).

### 5.2.1 Mise en place

- Vérifier que le commutateur (*switch*) J11 est placé en position RE2, le déplacer si nécessaire. La ligne DQ est ainsi connectée à la broche 2 de PORTE.
- En consultant la documentation de la librairie (*QHelp, OneWire Library*), donner les caractéristiques principale du protocole *one-wire*.

## 5.2.2 Communication

La librairie est composée de trois fonctions : `Ow_Reset()`, `Ow_Write()` et `Ow_Read()`. Pour lire et afficher une température, il faut suivre les étapes suivantes :

- Envoi de la commande `CONVERT_T` au capteur (mesure de la température)
- Envoi de la commande `READ_SCRATCHPAD` au capteur (placement de la température dans le *buffer* du capteur)
- Lecture du *buffer*
- Affichage.

Compléter le programme suivant (en vous aidant de la documentation) :

```
void main(){
  unsigned int temp;
  my_glcd_init();
  while(1) {
    // Step a)
    Ow_Reset(____, ____);
    Ow_Write(____, ____, 0xCC); // on s'adresse à tous les périphériques one-wire
    Ow_Write(____, ____, 0x44); // Envoi de la commande CONVERT_T
    Delay_us(120); // attente mesure

    // Step b)
    Ow_Reset(____, ____);
    Ow_Write(____, ____, 0xCC); // on s'adresse à tous les périphériques one-wire
    Ow_Write(____, ____, 0xBE); // Envoi de la commande READ_SCRATCHPAD

    // Step c)
    temp = Ow_Read(____, ____);

    // Step d)
    // a écrire

    delay_ms(100);
  }
}
```

Quel affichage obtenez-vous ?

### 5.2.3 Décodage de la température

La mesure envoyée par le capteur est codée comme indiqué par l'extrait de la *datasheet* du DS1820 :

#### TEMPERATURE REGISTER FORMAT Figure 2

LS Byte	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	$2^{-1}$
MS Byte	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
	S	S	S	S	S	S	S	S

#### TEMPERATURE/DATA RELATIONSHIP Table 2

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+85.0°C*	0000 0000 1010 1010	00AAh
+25.0°C	0000 0000 0011 0010	0032h
+0.5°C	0000 0000 0000 0001	0001h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1111	FFFFh
-25.0°C	1111 1111 1100 1110	FFCEh
-55.0°C	1111 1111 1001 0010	FF92h

\*The power-on reset value of the temperature register is +85°C

Il est donc indispensable de la décoder avant de l'afficher.

1. Quels bits contiennent la partie entière de la température ?
2. Quels bits contiennent la partie décimale de la température ?
3. Quels bits contiennent le signe de la mesure ?
4. Quelle est la précision de la mesure ?
5. Créer deux variables `temp_int` et `temp_dec` déclarée en **unsigned int** destinées à contenir respectivement la partie entière et la partie décimale. On négligera de gérer le signe.
6. Affecter ces deux variables avec les parties entière et décimale, déterminée à partir de `temp`. Vous pourrez utiliser des opérateurs de masque (&) et de décalage (>>, <<).
7. Corriger l'affichage en utilisant la fonction `sprinti` utilisée comme suit :

```
sprinti(text, "t = %02u,%01u (Celsius)", temp_int, temp_dec);
```

8. Que pensez-vous de la précision de la mesure de température ?

## 5.3 Supplément

Au choix :

1. Utiliser les fonctions de la librairie du GLCD pour réaliser un affichage graphique de la température en fonction du temps.
2. Lire en détail la documentation du DS1820 pour réaliser une lecture de température avec une résolution supérieure à 9bits (voir p. 3 de la *datasheet* et le projet `oneWire` dans les *examples*).

# TP 6

## Timer et interruption - mesure de temps et production de signaux périodiques

### 6.1 Éléments de documentation

OPTION_REG	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Features
	<b>RBPU</b>	<b>INTEDG</b>	<b>T0CS</b>	<b>T0SE</b>	<b>PSA</b>	<b>PS2</b>	<b>PS1</b>	<b>PS0</b>	<b>Bit name</b>
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

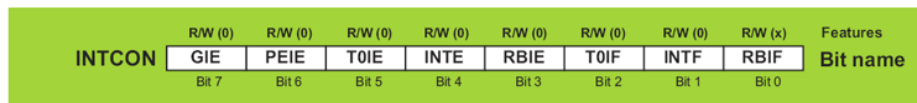
#### Legend

R/W Readable/Writable bit  
(1) After reset, bit is set

- RBPU - PORTB Pull-up enable bit**
  - 0 - PORTB pull-up resistors are disabled.
  - 1 - PORTB pins can be connected to pull-up resistors.
- INTEDG - Interrupt Edge Select bit**
  - 0 - Interrupt on rising edge of the INT pin (0-1).
  - 1 - Interrupt on falling edge of the INT pin (1-0).
- T0CS - TMR0 Clock Select bit**
  - 0 - Pulses are brought to TMR0 timer/counter input through the RA4 pin.
  - 1 - Timer uses internal cycle clock (Fosc/4).
- T0SE - TMR0 Source Edge Select bit**
  - 0 - Increment on high-to-low transition on the TMR0 pin.
  - 1 - Increment on low-to-high transition on the TMR0 pin.
- PSA - Prescaler Assignment bit**
  - 0 - Prescaler is assigned to the WDT.
  - 1 - Prescaler is assigned to the TMR0 timer/counter.
- PS2, PS1, PS0 - Prescaler Rate Select bit**
  - Prescaler rate is adjusted by combining these bits. As seen in the table, the same combination of bits gives different prescaler rate for the timer/counter and watch-dog timer, respectively.

PS 2	PS 1	PS 0	TMR 0	WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

When PSA bit is cleared, prescaler is assigned to TMR0 timer/counter as illustrated on the figure below:



- **GIE - Global Interrupt Enable bit** - controls all possible interrupt sources simultaneously.
  - 1 - Enables all unmasked interrupts.
  - 0 - Disables all interrupts.
- **PEIE - Peripheral Interrupt Enable bit** acts similar to the GIE it, but controls interrupts enabled by peripherals. It means that it has no impact on interrupts triggered by the timer TMR0 or by changing the state of PORTB or the RB0/INT pin.
  - 1 - Enables all unmasked peripheral interrupts.
  - 0 - Disables all peripheral interrupts.
- **TOIE - TMR0 Overflow Interrupt Enable bit** controls interrupt enabled by TMR0 overflow.
  - 1 - Enables the TMR0 interrupt.
  - 0 - Disables the TMR0 interrupt.
- **INTE - RB0/INT External Interrupt Enable bit** controls interrupt caused by changing the logic state of the RB0/INT input pin (external interrupt).
  - 1 - Enables the INT external interrupt.
  - 0 - Disables the INT external interrupt.
- **RBIE - RB Port Change Interrupt Enable bit**. When configured as inputs, PORTB pins may cause an interrupt by changing their logic state (no matter whether it is high-to-low transition or vice versa, the fact that something is changed only matters). This bit determines whether an interrupt is to occur or not.
  - 1 - Enables the port B change interrupt.
  - 0 - Disables the port B change interrupt.
- **TOIF - TMR0 Overflow Interrupt Flag bit** registers the timer TMR0 register overflow, when counting starts at zero.
  - 1 - TMR0 register has overflowed (bit must be cleared from within the software).
  - 0 - TMR0 register has not overflowed.
- **INTF - RB0/INT External Interrupt Flag bit** registers the change of the RB0/INT pin logic state.
  - 1 - The INT external interrupt has occurred (must be cleared from within the software).
  - 0 - The INT external interrupt has not occurred.
- **RBIF - RB Port Change Interrupt Flag bit** registers any change of logic state of some PORTB input pins.
  - 1 - At least one of the PORTB general purpose I/O pins has changed state. Upon reading PORTB, the RBIF bit must be cleared from within the software.
  - 0 - None of the PORTB general purpose I/O pins has changed the state.

## 6.2 Test de fonctionnement

Saisir et tester le programme suivant :

```

unsigned int cnt;

void interrupt() {
    cnt++;           // Increment value of cnt on every interrupt
    TMR0 = 96;
    INTCON = 0x20;  // Set T0IE, clear T0IF
}

void main() {
    OPTION_REG = 0x84; // Assign prescaler to TMR0
    ANSEL = 0;        // Configure AN pins as digital I/O
    ANSELH = 0;
    TRISB = 0;       // PORTB is output
    PORTB = 0xFF;    // Initialize PORTB
    TMR0 = 96;       // Timer0 initial value
    INTCON = 0xA0;   // Enable TMR0 interrupt
    cnt = 0;         // Initialize cnt

    while (1) {
        if (cnt == 400) {
            PORTB = ~PORTB; // Toggle PORTB LEDs
            cnt = 0;        // Reset cnt
        }
    }
}

```

1. Que se passe-t-il?

2. Expliciter la configuration des registres `OPTION_REG` et `INTCON`.
3. A quels endroits est-il possible de modifier le code pour allonger la période de clignotement ?
4. Modifier les bits du *prescaler* selon la table fournie pour allonger la période de clignotement.

## 6.3 Mesure de durée d'exécution d'un morceau de code

Voici une fonction que l'on souhaite tester :

```

unsigned int div10(unsigned int A){
  unsigned int Q; /* the quotient */
  Q = ((A >> 1) + A) >> 1; /* Q = A*0.11 */
  Q = ((Q >> 4) + Q)      ; /* Q = A*0.110011 */
  Q = ((Q >> 8) + Q) >> 3; /* Q = A*0.00011001100110011 */
  /* either Q = A/10 or Q+1 = A/10 for all A < 534,890 */
  return Q;
}

```

1. Écrire un programme utilisant cette fonction pour afficher le résultat (sur le GLCD) de la division par 10 du nombre 171.
2. On souhaite maintenant connaître la durée d'exécution de cette fonction en utilisant le *timer* `TMR0`. Écrire un programme qui mesure cette durée, avec l'algorithme suivant :
  - (a) Initialisation du *timer*
  - (b) Appel de la fonction
  - (c) Lecture du timer
  - (d) Calcul de la durée (en fonction de la fréquence d'horloge et du *prescaler*).
  - (e) Affichage de la durée mesurée.

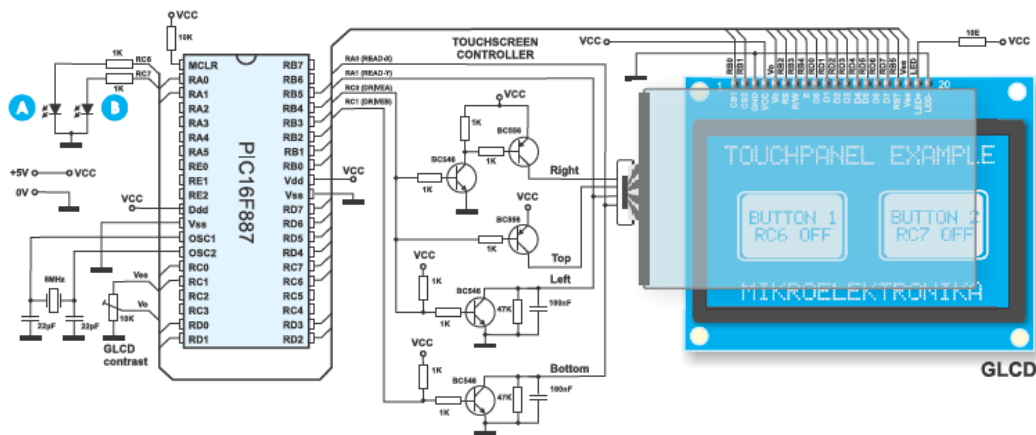
## 6.4 Production d'un signal périodique

1. Générer un signal de fréquence 1 KHz sur `PB0`. Pour cela :
  - (a) calculer la valeur de pré-division,
  - (b) calculer la valeur de comptage,
  - (c) écrire le programme.
2. Générer un signal de fréquence 1 KHz de rapport cyclique 1/4.

## TP 7

# Interfaçage d'un écran tactile

L'objectif est d'interfacer l'écran tactile pour commander l'allumage et l'extinction d'une LED. Voici le schéma détaillant la connexion de l'écran au microcontrôleur :



Divers ressources sont récupérable à l'adresse

<http://pixel-shaker.fr/fr/enseignements/geii-programmation-pic-en-c-easypic5-mikroc> :

- un article (en français) sur le fonctionnement et l'utilisation de l'écran tactile ;
- un premier programme (à terminer) d'interfaçage : `touchscreen1.c`
- un second programme à tester (en fin de TP) : `touchscreen2.c`

1. Lire le document, en particulier le paragraphe "principe de fonctionnement" et expliquer pourquoi seuls quatre fils sont nécessaires pour interfacier l'écran tactile.
2. Expliquer alors le fonctionnement des fonctions `GetX()` et `GetY()` dans le programme `touchscreen1.c`
3. Placer le fichier `touchscreen1.c` dans un projet et compléter le programme pour dans une boucle infinie, afficher (sur le GLCD) les coordonnées `x` et `y` acquises via les fonctions `GetX()` et `GetY()`.
  - Quelles sont les valeurs min et max que peuvent prendre ces coordonnées ?
  - Quelle est l'orientation des axes ?
4. A partir des coordonnées `x` et `y`, calculer les coordonnées `x_screen` et `y_screen` correspondant à la position courante dans le GLCD.

5. Ajouter le code suivant en début de la fonction main :

```
Glcd_Fill(0); // Clear GLCD
Glcd_Write_Text("TOUCHPANEL EXAMPLE",10,0,1);
//Display Buttons on GLCD:
Glcd_Rectangle(8,16,60,48,1);
Glcd_Rectangle(68,16,120,48,1);
Glcd_Box(10,18,58,46,1);
Glcd_Box(70,18,118,46,1);
Glcd_Write_Text("BUTTON1",14,3,0);
Glcd_Write_Text("RC6 OFF",14,4,0);
Glcd_Write_Text("BUTTON2",74,3,0);
Glcd_Write_Text("RC7 OFF",74,4,0);
```

Vérifier que deux “boutons” sont dessinés. Ajouter alors le code nécessaire dans la boucle infinie pour qu’une pression sur le “bouton 1” provoque l’extinction de la LED n°6 du PORTC, et qu’un appui sur le “bouton 2” commande sont allumage.

6. Télécharger et tester le programme du fichier touchscreen2.c