Synthèse STS SE

Méthode de développement de programme en langage C avec PICC et PROTEUS sur systèmes embarqués à base de microcontrôleur MICROCHIP

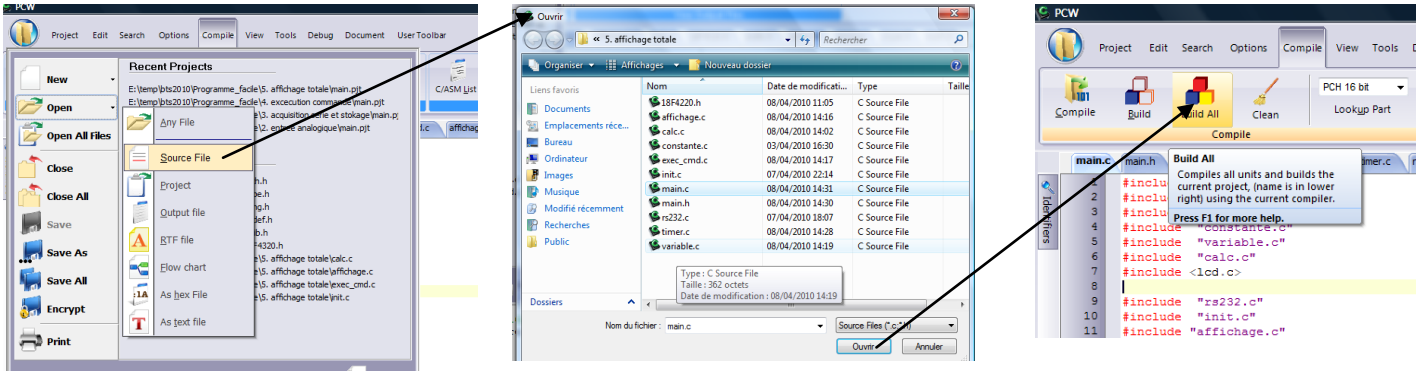
Programmation facile

Pré requis : langage C, PICC, PROTEUS.

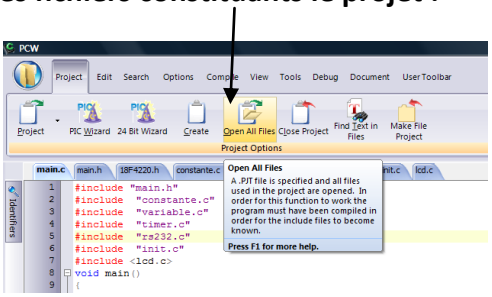
Utilisation :

Vous disposez des projets correspondants à chacune des applications. Le nom du dossier à utiliser est indiqué au niveau du titre de chaque projet. Vous trouvez dans chaque dossier les fichiers langage C pour PICC et le fichier schéma pour PROTEUS. Pour utiliser ses projets :

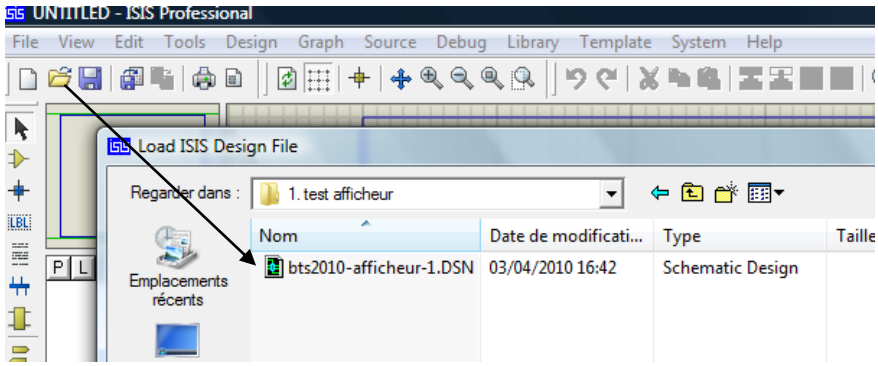
- **PICC : charger le fichier « main.c » puis le compiler.**



Il est possible de voir tous les fichiers constituants le projet :



- **PROTEUS : vous chargez le fichier schéma.**



Mise en situation :

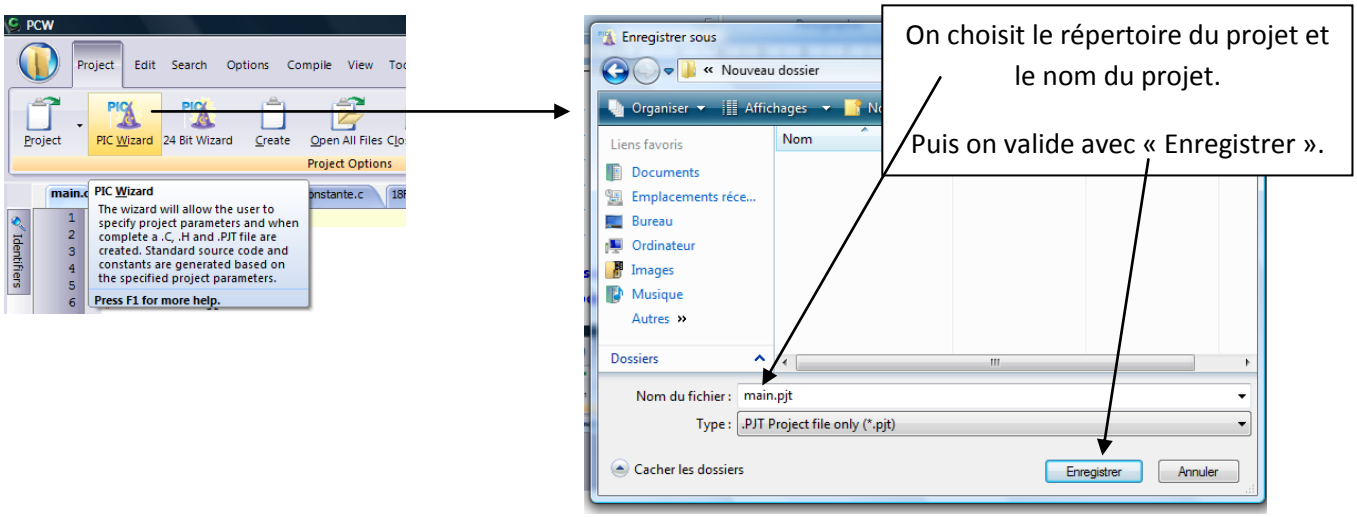
Le système, que vous avez à développer va définir les ressources nécessaires à ce microprocesseur :

- **Entrée-sorties logiques.**
- **Entrée-sorties analogiques.**
- **Liaisons série : RS232, SPI, USB…**
- **Timer.**
- **Interruption.**

Nous allons nous attacher à un système pour lequel le choix d'un microcontrôleur PIC18F4220 s'impose. Les contraintes sont les suivantes.

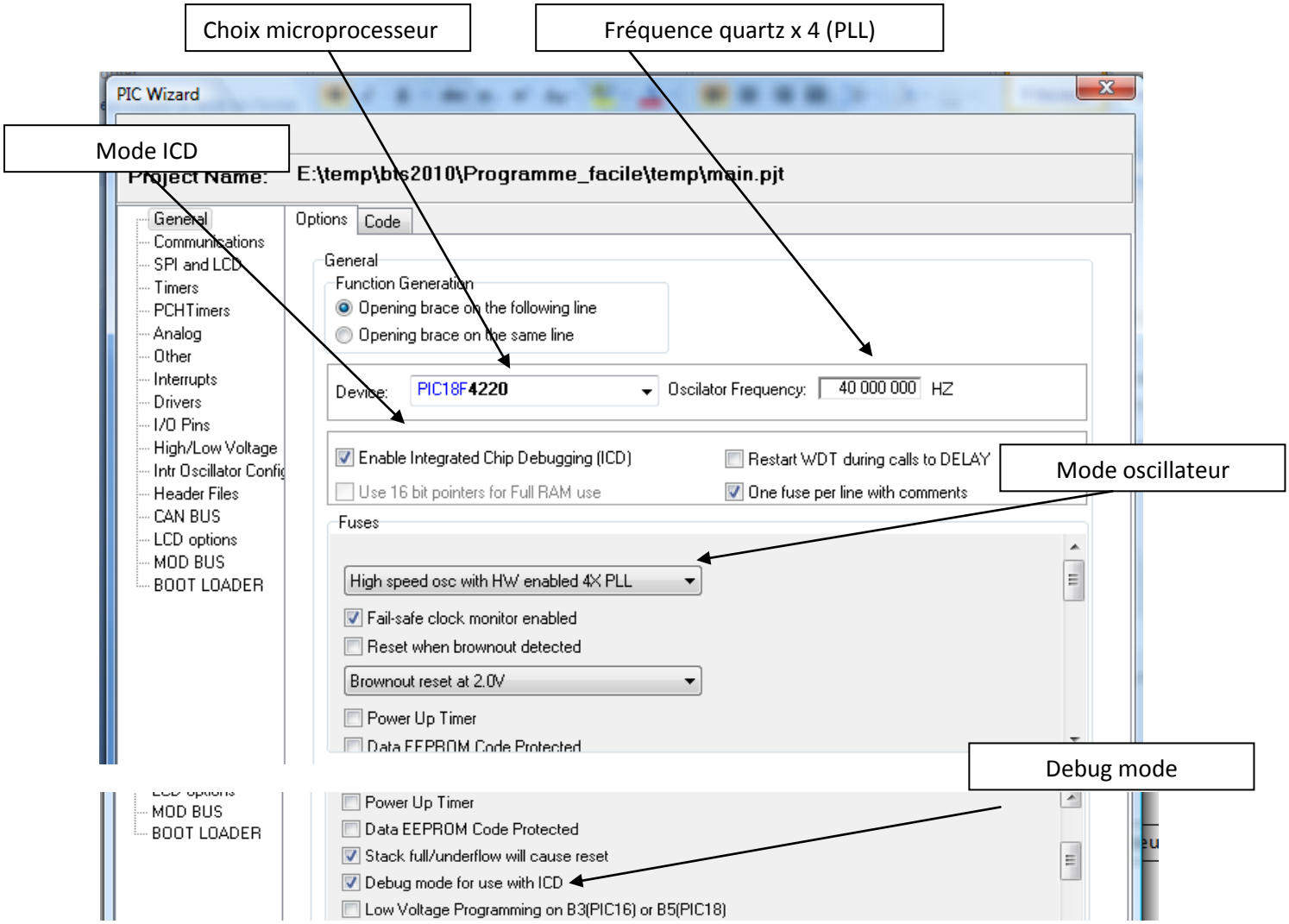
- **1 liaison ICD pour la programmation du microcontrôleur.**
- **1 Entrée analogique avec Vref+ et Vref-.**
- **1 horloge temps réel de l'ordre de 1 kHz pour la gestion des claviers et des temporisations.**
- **1 liaison avec micro-ordinateur PC - RS232 avec PC 9600 bauds, 1 stop bit pas de parité.**
- **1 liaison SPI.**
- **1 afficheur alpha numérique à bus parallèle.**

Dans un premier temps, nous devons définir les contraintes de chacune des fonctions afin de pouvoir avec le wizard de PICC produire la configuration du microcontrôleur.

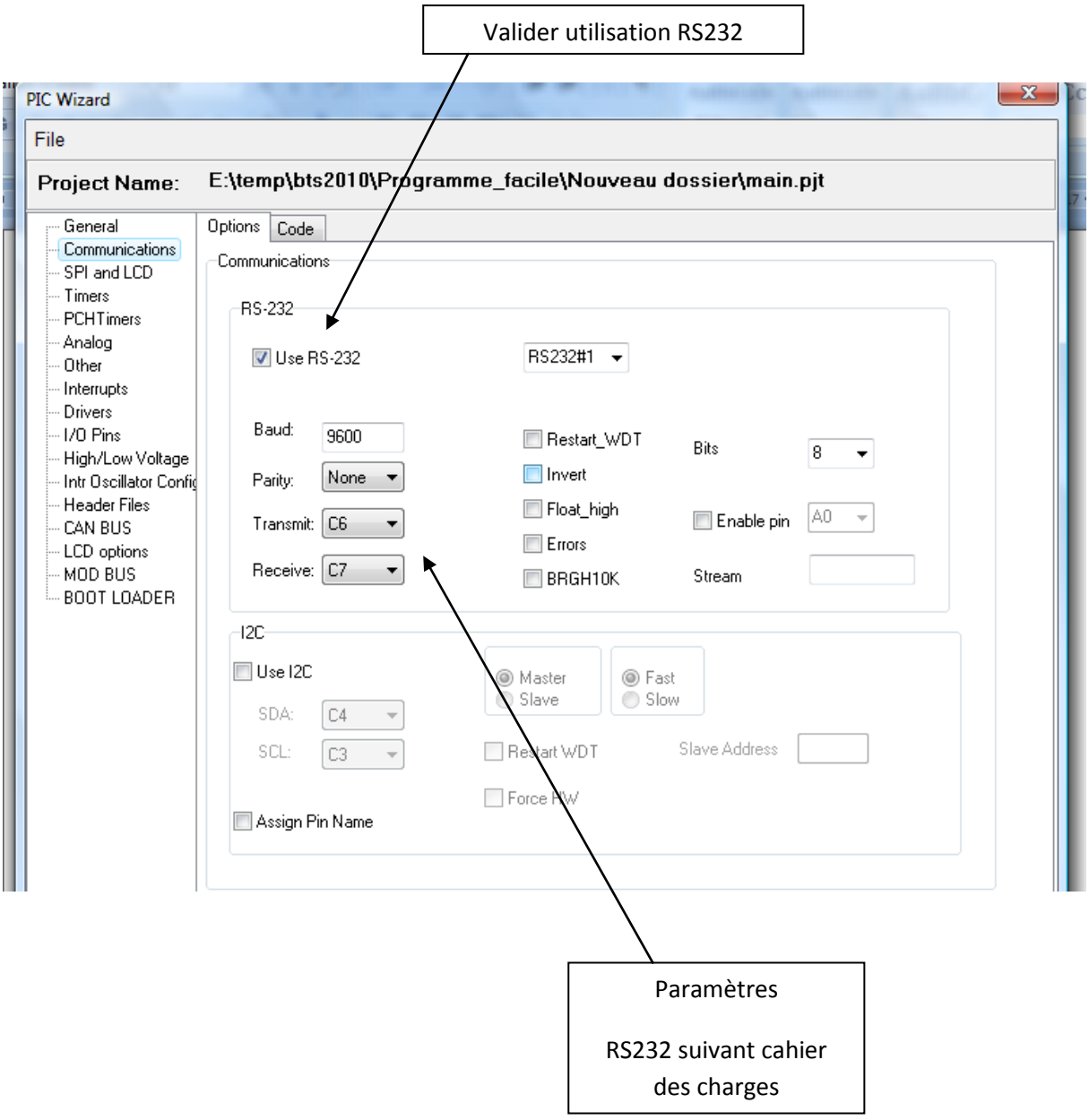


On définit ensuite les différentes options à configurer.

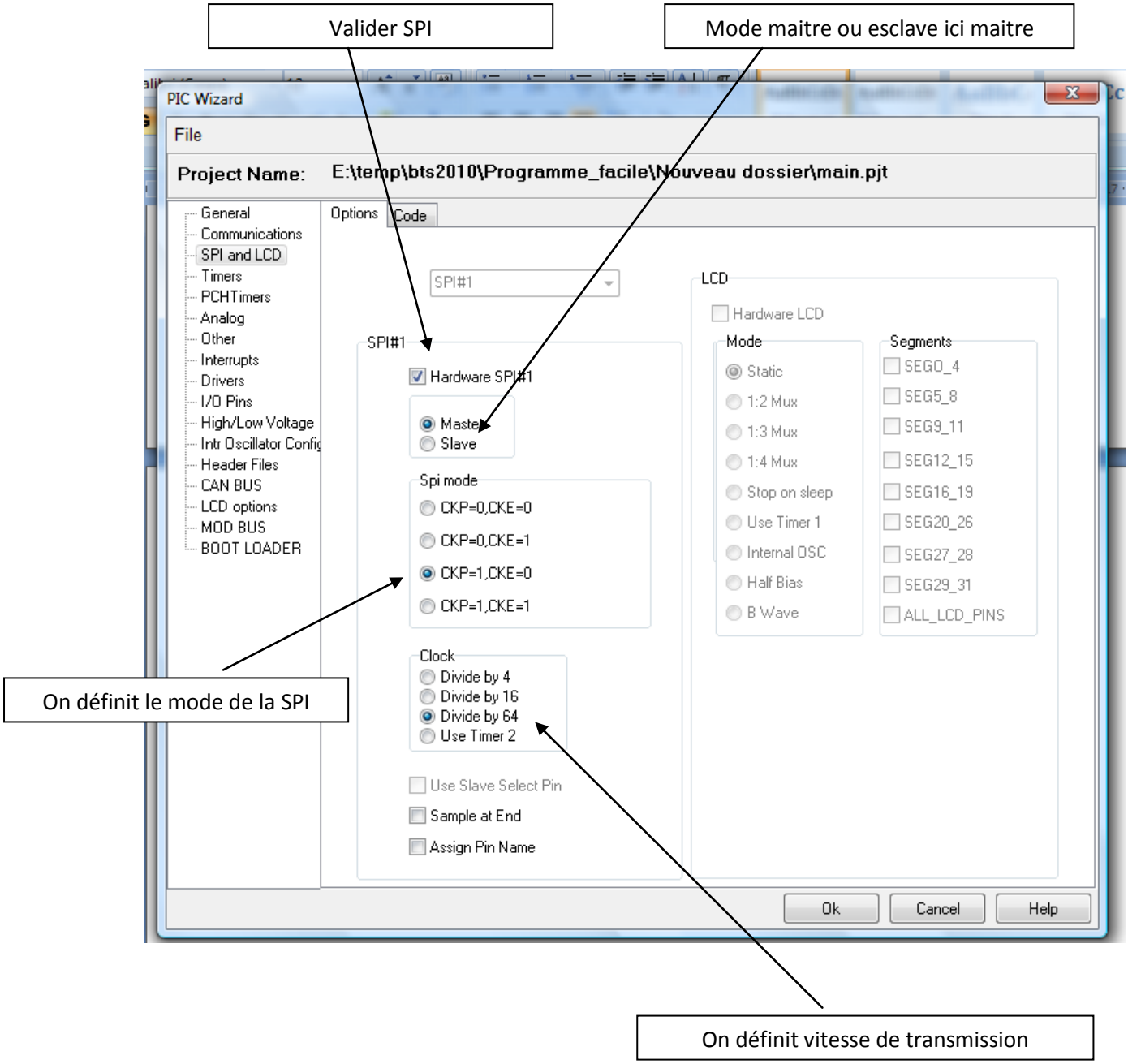
Page générale :



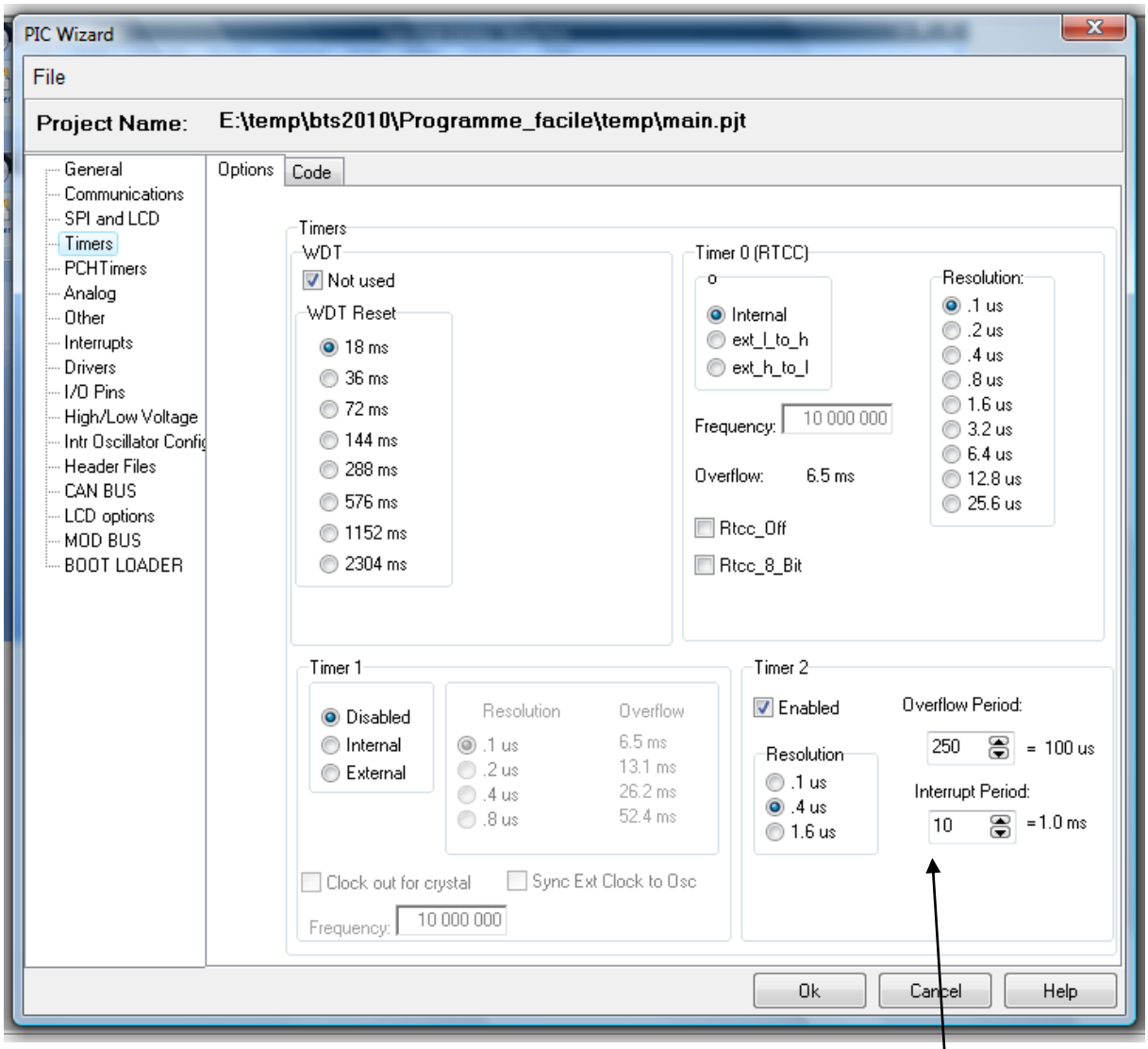
Ressource : 1 liaison RS232 avec PC (9600 bauds, 1 stop bit et pas de parité) onglet communication.



Ressource : 1 SPI – onglet SPI et LCD.



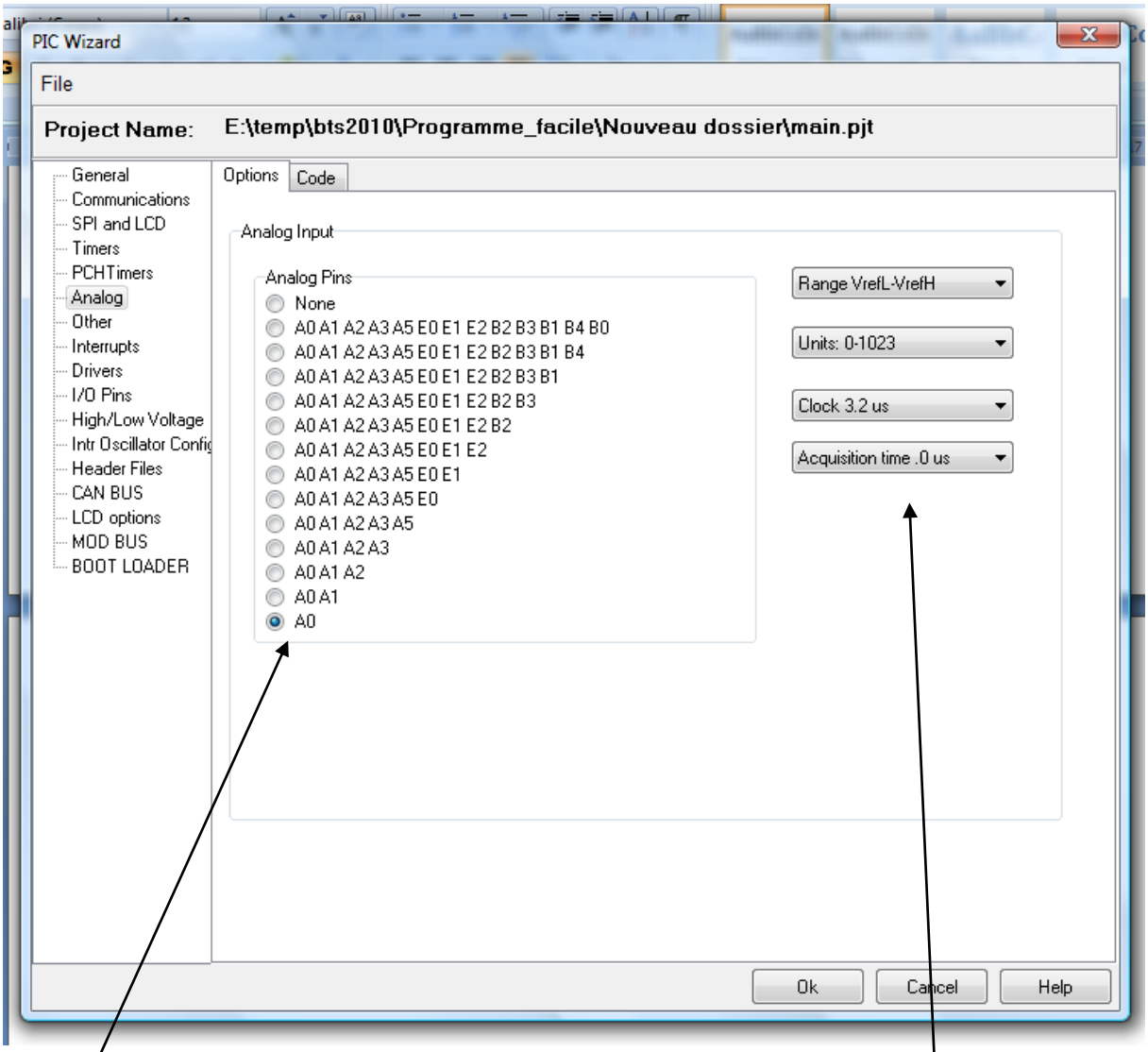
Ressource : 1 horloge temps réel de l'ordre de 1 kHz pour la gestion des claviers et des temporisations – onglets TIMERS



On utilise le Timer 2, on règle la période d'interruption à 1 ms.

Il faudra au niveau de l'onglet Interrupts activer l'interruption sur le Timer 2.

Ressource : 1 Entrée analogique avec Vref+ et Vref-, on utilise l'entrée A0 – onglet analog.



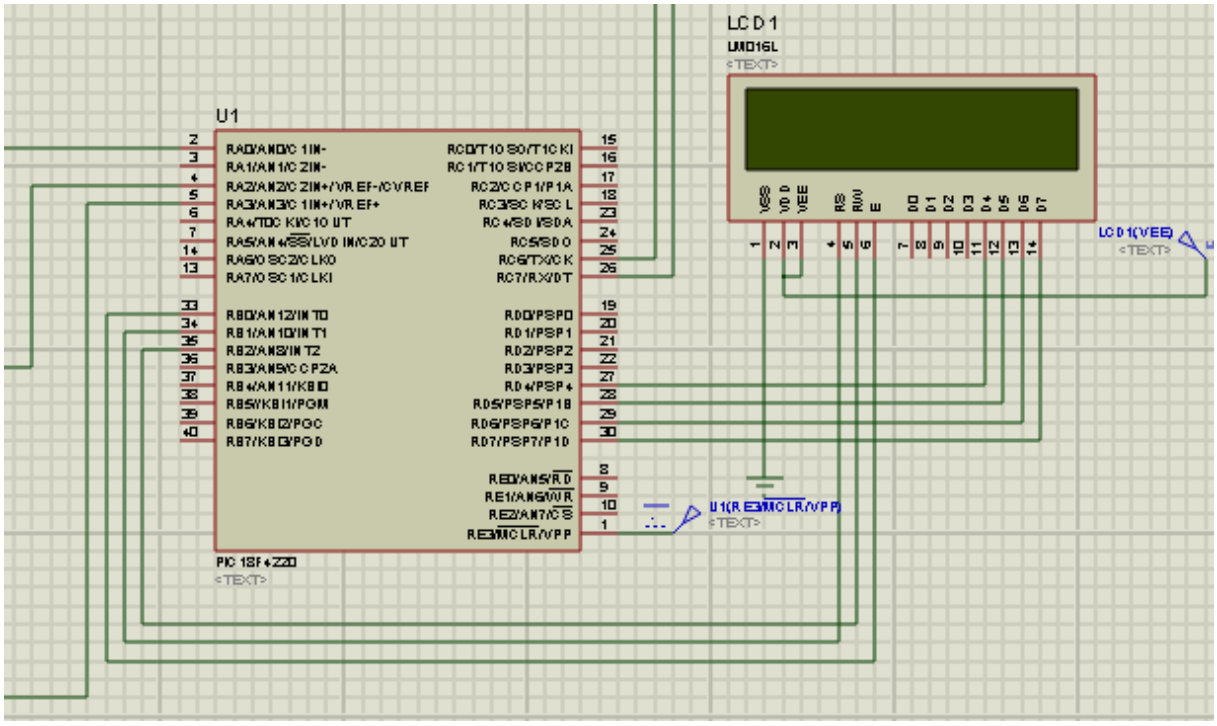
Une seule entrée A0

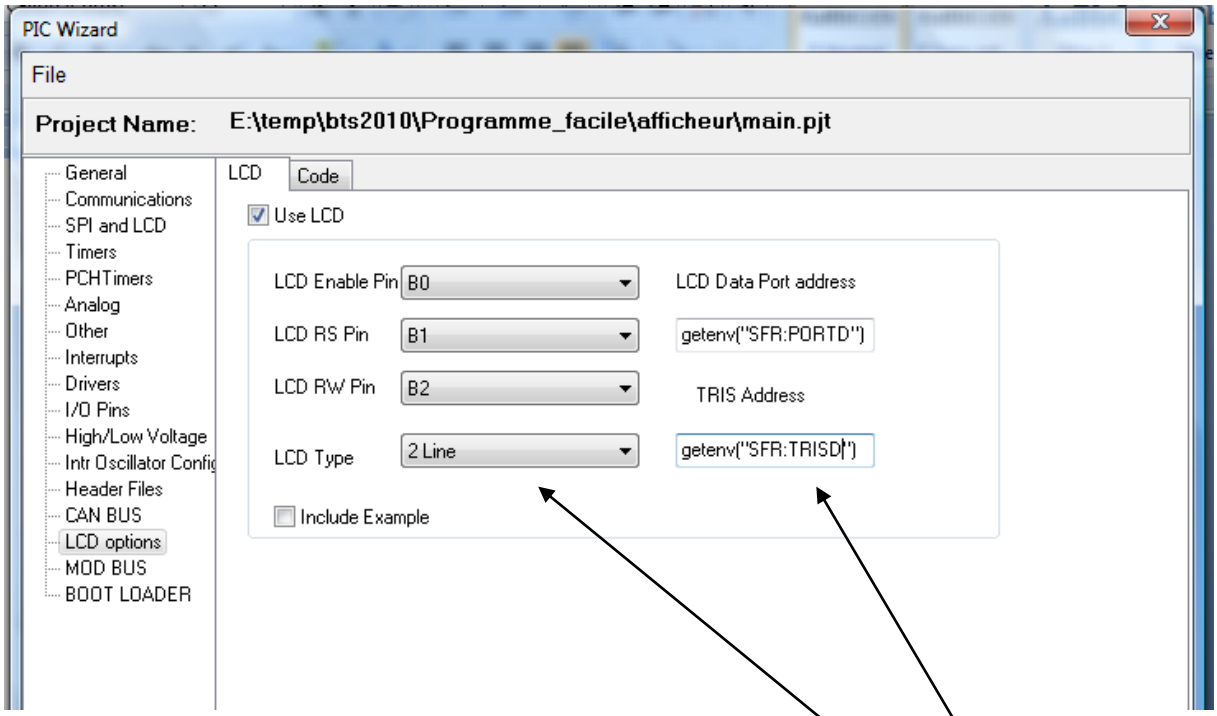
On définit les paramètres :

- 10 bits,
- Range …

Ressource : 1 afficheur alpha numérique à bus parallèle – onglet LCD option.

Les composants sont reliés comme ceci :

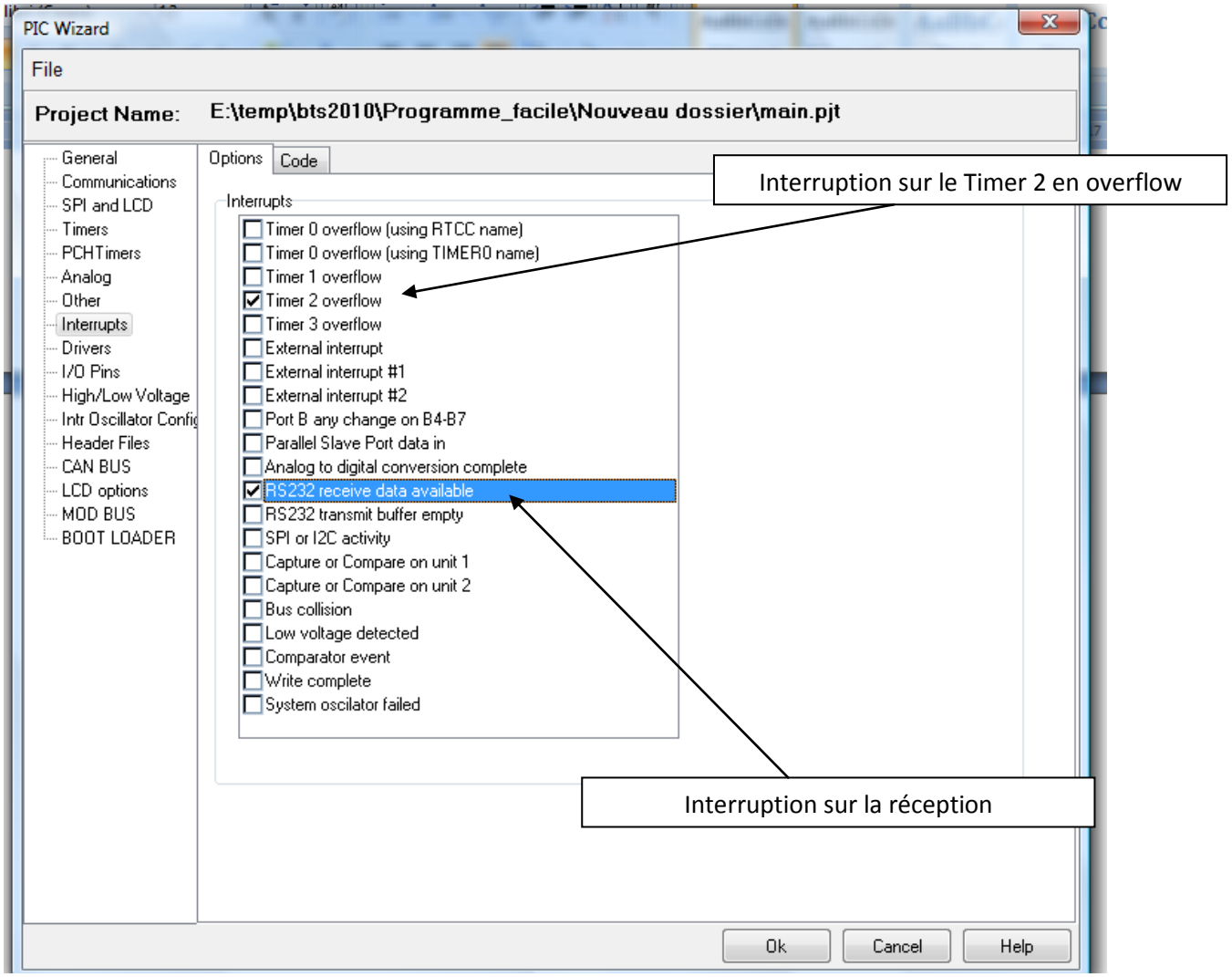




Attention à l'écriture du Port de donnée.

Définir commande et donnée

Ressources interruption – onglet Interrupt.



La configuration est terminée, il suffit de valider avec Ok. Le programme de base est produit automatiquement.

Programme obtenu :

```
#include "E:\temp\bts2010\Programme_facile\Nouveau dossier\main.h"
#int_TIMER2
void  TIMER2_isr(void)
{


}
#int_RDA
void  RDA_isr(void)
{


}
#define LCD_ENABLE_PIN PIN_B0
#define LCD_RS_PIN PIN_B1
#define LCD_RW_PIN PIN_B2
#define LCD_DATA_PORT getenv("SFR:PORTD")
#define LCD_TYPE 2
#define LCD_TRIS_LOCATION getenv("SFR:TRISD")
#include <lcd.c>
void main()
{
  lcd_init();
  setup_adc_ports(AN0|VREF_VREF);
  setup_adc(ADC_CLOCK_DIV_8|ADC_TAD_MUL_0);
  setup_psp(PSP_DISABLED);
  setup_spi(SPI_MASTER|SPI_H_TO_L|SPI_XMIT_L_TO_H|SPI_CLK_DIV_64);
  setup_wdt(WDT_OFF);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_16|RTCC_8_bit);
  setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DIV_BY_4,250,10);
  setup_timer_3(T3_DISABLED|T3_DIV_BY_1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  enable_interrupts(INT_TIMER2);
  enable_interrupts(INT_RDA);
  enable_interrupts(GLOBAL);
//Setup_Oscillator parameter not selected from Intr Oscillator Config tab

// TODO: USER CODE
}
```

Interruption Timer 2

Interruption RS232 en réception

Définition du câblage afficheur

Inclusion librairie lcd

Initialisation des ressources du microcontrôleur

Le traitement principal est ici

Réorganisation du programme :

Je propose de découper le programme en plusieurs fichiers, afin de faciliter la lecture et le développement (ce n'est pas une obligation).

1. **Fichier Timer.c**

```
#int_TIMER2
void  TIMER2_isr(void)
{
}
```

Interruption Timer 2

2. **Fichier RS232.c**

```
#int_RDA
void  RDA_isr(void)
{
}
```

Interruption RS232 en réception

3. **Fichier constante.c : nous déclarerons les constantes et les équivalences dans ce fichier.**

```
#define LCD_ENABLE_PIN PIN_B0
#define LCD_RS_PIN PIN_B1
#define LCD_RW_PIN PIN_B2
#define LCD_DATA_PORT getenv("SFR:PORTD")
#define LCD_TYPE 2
#define LCD_TRIS_LOCATION getenv("SFR:TRISD")
```

Définition du câblage afficheur

4. **Fichier init.c : nous mettrons les fonctions d'initialisation.**

```
Void init(void)
 {
 setup_adc_ports(AN0|VREF_VREF);
  setup_adc(ADC_CLOCK_DIV_8|ADC_TAD_MUL_0);
  setup_psp(PSP_DISABLED);
  setup_spi(SPI_MASTER|SPI_H_TO_L|SPI_XMIT_L_TO_H|SPI_CLK_DIV_64);
  setup_wdt(WDT_OFF);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_16|RTCC_8_bit);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DIV_BY_16,170,1);
  setup_timer_3(T3_DISABLED|T3_DIV_BY_1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  enable_interrupts(INT_TIMER2);
  enable_interrupts(INT_RDA);
  enable_interrupts(GLOBAL);
 }
```

Initialisation des ressources du microcontrôleur

5. **Fichier variable.c : nous mettrons les variables globales.**

```
/* on déclare les variables globales */
....
/* pour donner la valeur initiale des variables globales*/
Void init_variable(void)
{
  ...
}
```

Le programme principal devient (il est très court et facilement lisible) :

```
#include "main.h"
#include  "constante.c"
#include  "variable.c"
#include <lcd.c>
#include  "timer.c"
#include  "rs232.c"
#include  "init.c"
void main()
{
  lcd_init();
  init();
  Init_variable();
 while (true)
  {
                    <-------------------------------
  }
}
```

Le traitement principal est ici, avec une boucle infinie.

Le plus difficile est fait, il ne reste qu'à écrire le reste.

Un conseil : programme tel qu'il est structuré permet d'avancer fonction par fonction.

Fonction affichage : vérification (dossier : 1. test afficheur).

On utilise la fonction printf(…) pour afficher «Cela fonctionne ».
Sous Picc, on ajoute dans la fonction main() et on compile le programme.

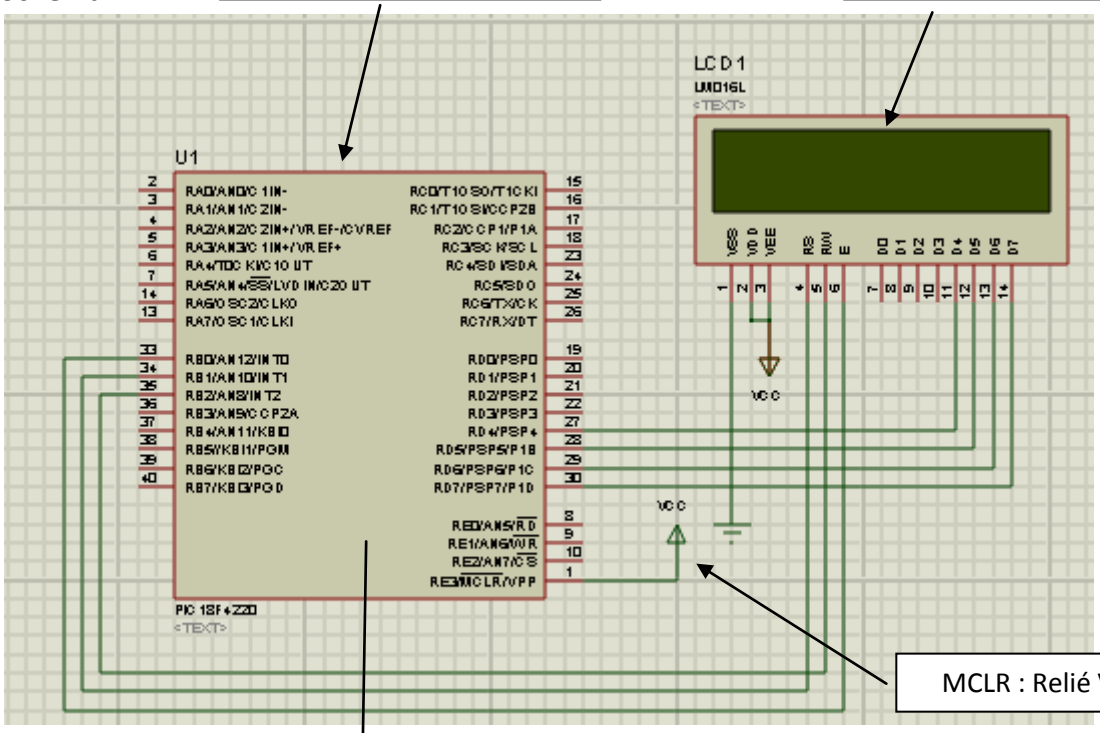
```
void main()
{
  Init_variable();
  lcd_init();
  init();
  printf(lcd_putc ,"Cela fonctionne");
 while (true)
  {

  }
}
```

Indique que la fonction printf est dirrigé vers le flux lcd_putc (afficheur LCD)

Schéma :

Composant : PIC18F4220

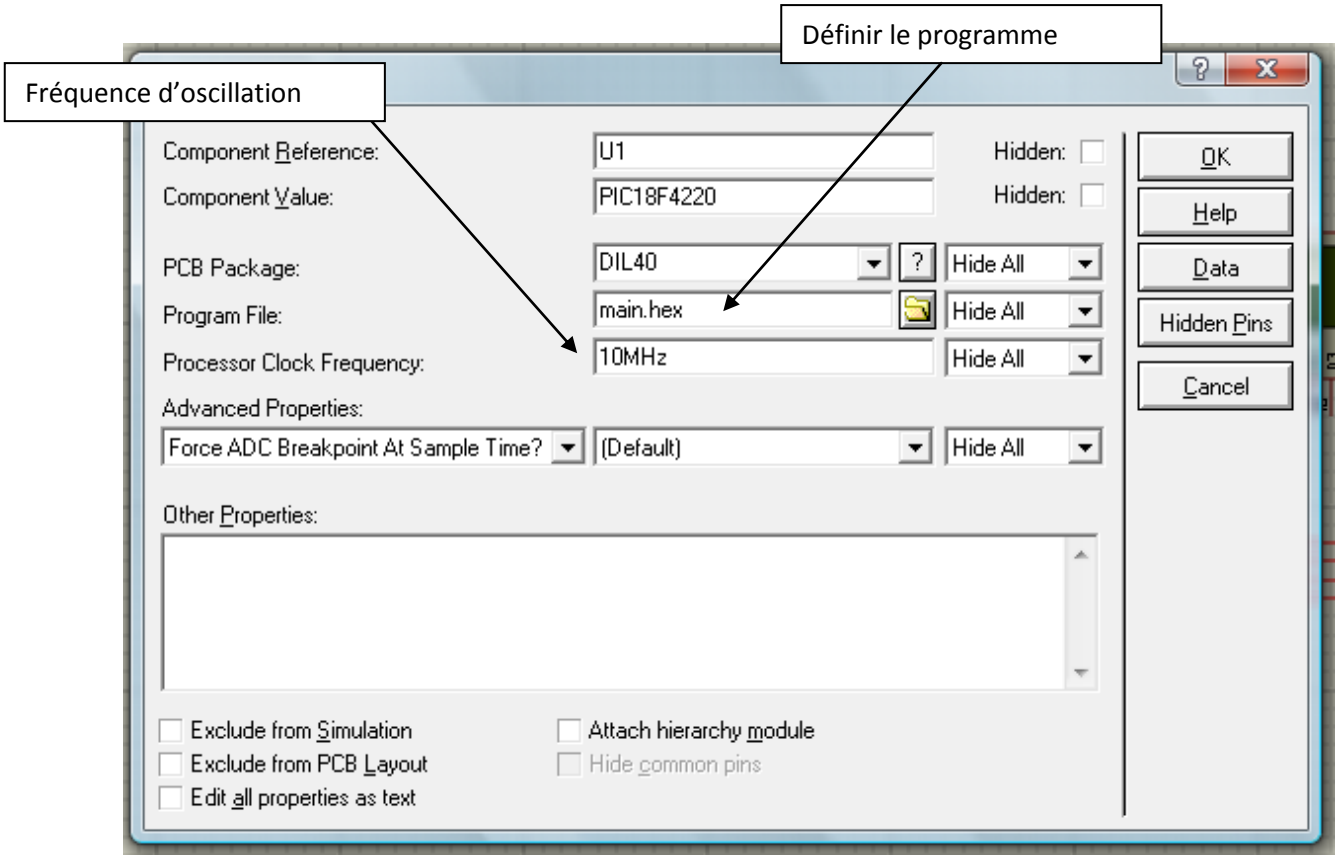
Composant : LM016L



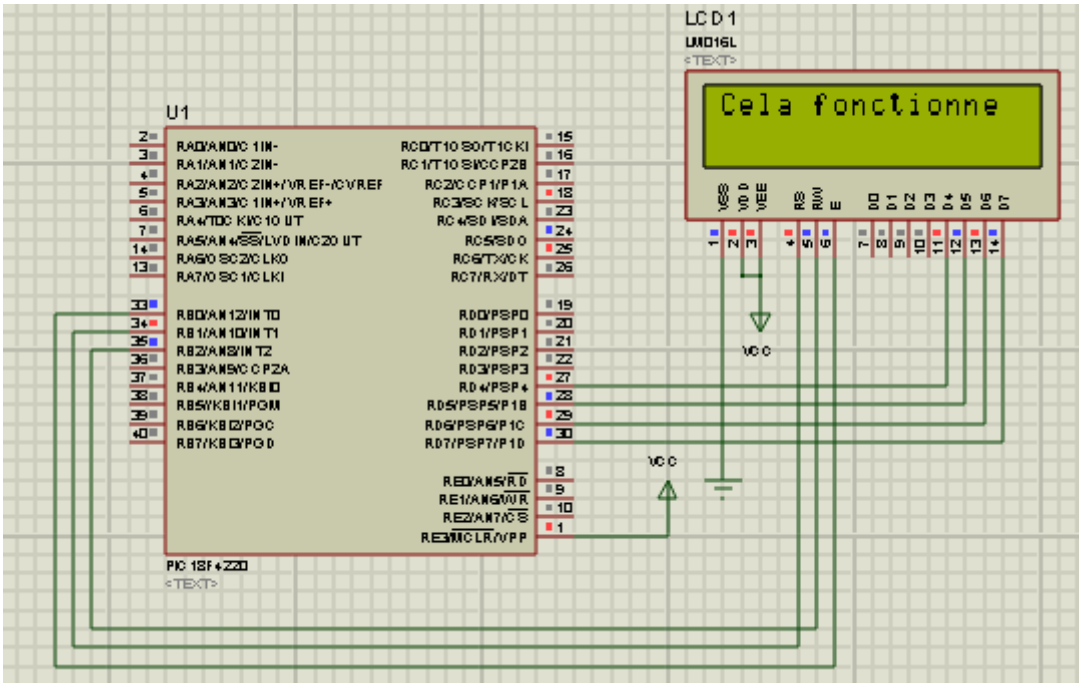
MCLR : Relié VPP à VCC

PIC18F4220 : configuration

Configuration du microcontrôleur sous Proteus.



Et voici le résultat :

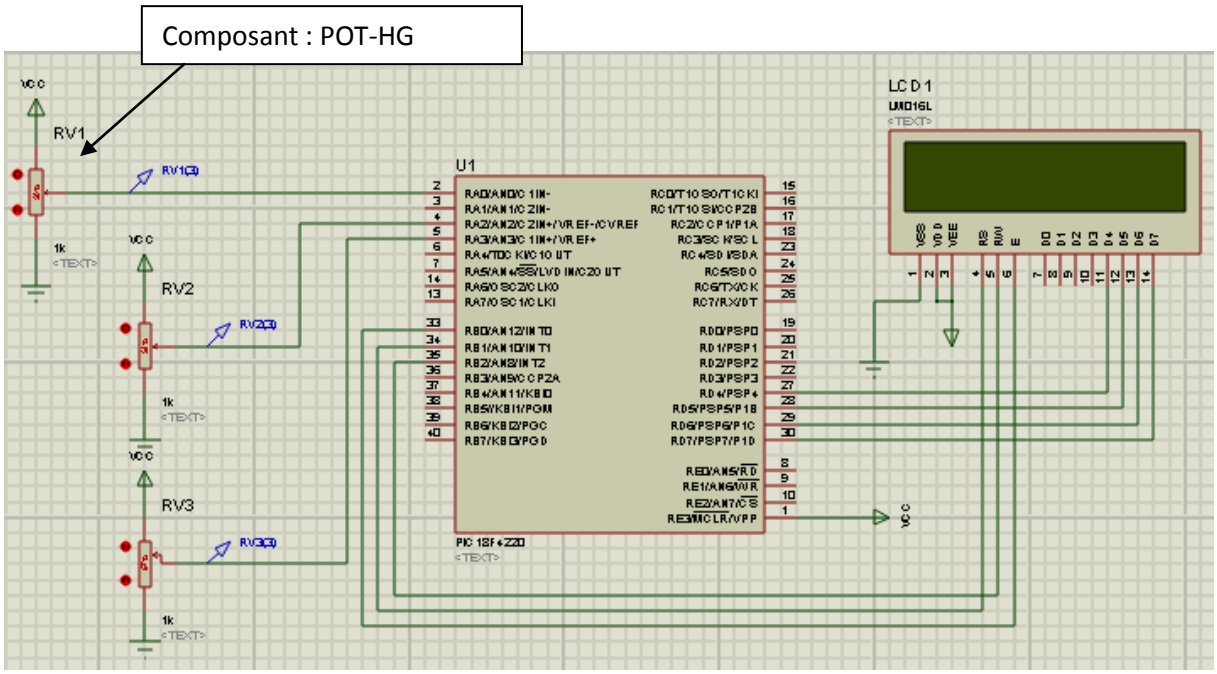


Tout va bien, nous pouvons poursuivre.

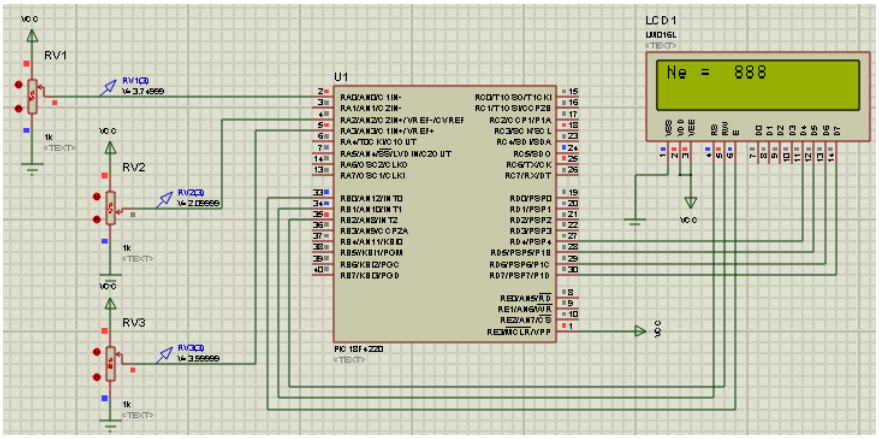
Fonction conversion analogique-numérique (dossier : 2. entree analogique).

A0 = entrée analogique, A2 = Vref- et A3 = Vref+.

Schéma sous PROTEUS :



Nous allons afficher sur l'afficheur le nombre en sortie du convertisseur analogique-numérique correspondant à la tension analogique en entrée A0 en fonction des tensions de références.



La fonction main() devient :

```
void main()
{
 unsigned long value;
 lcd_init();
 init();
 init_data();
while (true)
 {
  value = read_adc();  /*lecture de l'entrée analogique*/
  lcd_gotoxy(1,1);      /*déplacement curseur*/
  printf(lcd_putc,"Ne = %4Lu", value);  /*Affichage avec formatage du résultat*/
 }
}
```

Liaison RS232 avec PC – réception données (dossier : 3. acquisition serie et stokage).

Le protocole entre le PC et le système à micro-processeur est décrit ci-dessous :

Une commande se présente comme une suite de code ASCII. Le format est le suivant :

CMDstart	*CMD1*	*V1*	*V2*	*V3*	*V4*	*V5*	*V…*	END

Les commandes ne sont pas sensibles à la casse, les espaces sont ignorés, il est au plus composé de 20 code ASCII.

- **CMDstar : R, T ou N.**
- **CMD1 : C, M.**
- **V1 : soit un chiffre ou les codes de commande ? (demande de valeur), ! (réponse message reçue).**
- **V2, V3 … : chiffre.**
- **END : '/0'. (code ascii correspondant à 0)**

Exemples :

- **TC6520'/0' : le PC envoie température de consigne de 65,2°**
- **TM ?'/0' : le PC demande la température mesurée.**
- **TM7145'/0' : le boitier de commande répond température mesurée 71,45°.**
- **RM ?'/0' : le PC demande la résistance mesurée.**

La température est codée en centième de degré.

Les données vont arriver octet par octet par l'interface série. Il faut donc créer un tableau de caractère dans lequel seront stockés les octets avant d'être exécutés.

- **3 variables à placer dans « variable.c » sont créées. Elles sont initialisées à la mise sous tension avec la fonction initi_variable().**

Voici le fichier « variable.c » modifié :

```
char buffer_reception_pc[20];
unsigned int ptr__buffer_cmd_pc;
short commande_presente_pc;
void init_data(void)
        {
                ptr__buffer_cmd_pc = 0;
                commande_presente_pc=false;
        }
```

Chaque octet reçu par l'interface produit une interruption, le programme d'interruption série est le suivant :

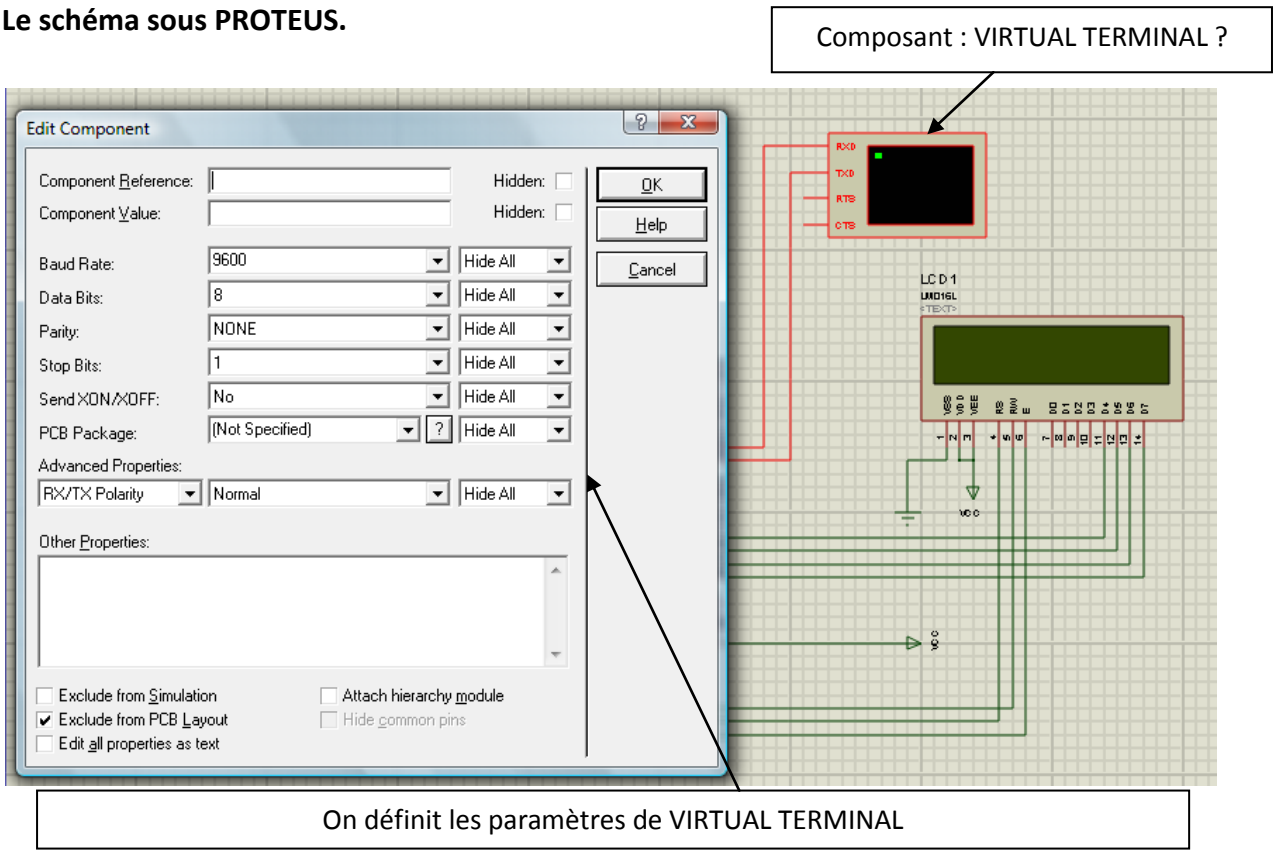
```
#int_RDA
void  RDA_isr(void)
   {
      char cmde;
      cmde = getch(); /*lecture du caractere*/
      /*la casse n'est pa sensible*/
      if ((cmde>'a') && (cmde<'z')) cmde =  toupper(cmde); /*si caractere est une lettre de a .. z
                           alors majuscule*/
         if ((cmde=='T') | (cmde=='R') | (cmde=='N'))
           {
              ptr_buffer_cmd_pc = 1;
              buffer_reception_pc[0] = cmde;
           }
           else if (cmde != ' ') /*espace est ignoré*/
                {
                if (( ptr_buffer_cmd_pc != 0) && (ptr_buffer_cmd_pc <20))
                 {
                  if (cmde == '\0') /* so 0 alors fin de commande */
                       {
                        commande_presente_pc = true; /*une nouvelle commande est presente*/
                        buffer_reception_pc[ptr_buffer_cmd_pc] = cmde;
                        ptr_buffer_cmd_pc = 0;
                       }
                   else
                       {
                        buffer_reception_pc[ptr_buffer_cmd_pc] = cmde;
                        ptr_buffer_cmd_pc ++;
                       }
                }
             }
  }
```

Pour effectuer les tests avec PROTEUS, nous allons afficher sur la deuxième ligne de l'afficheur la chaine reçue lorsqu'elle est conforme.

Nous allons modifier la fonction main().

```
void main()
{
  unsigned long value;
  lcd_init();
  init();
  init_data();
 while (true)
  {
   value = read_adc();
   lcd_gotoxy(1,1);
   printf(lcd_putc,"Ne=%4Lu", value);
   if (commande_presente_pc)
    {
     lcd_gotoxy(1,2);
     printf(lcd_putc,buffer_reception_pc);
     commande_presente_pc = false;
     ptr_buffer_cmd_pc = 0;
    }
  }
}
```

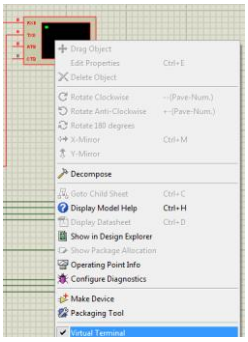
Le schéma sous PROTEUS.

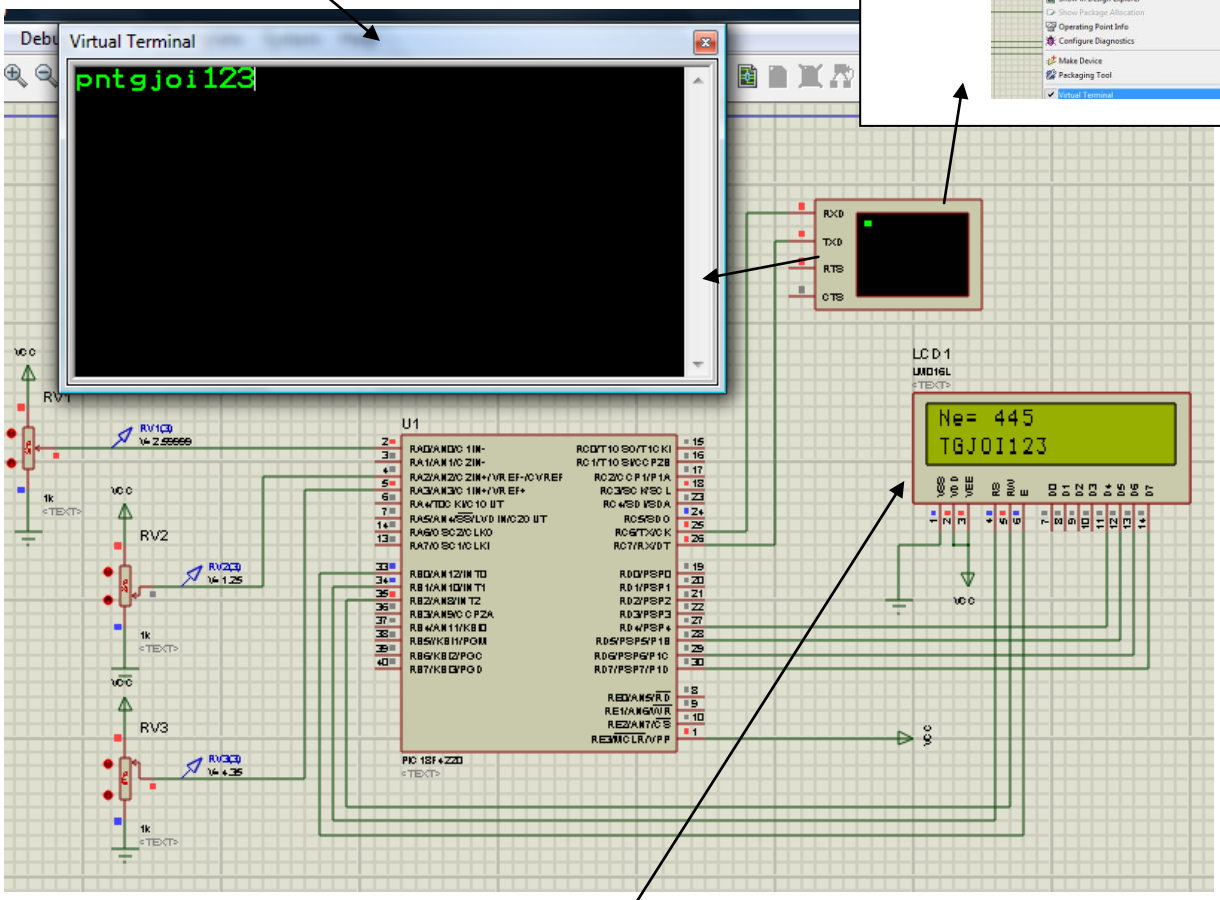


Résultat obtenu.

Une chaine de caractère est saisie dans VIRTUAL TERMINAL
Le caractère '\0' est saisi en appuyant la touche CTRL puis le chiffre 0.

Si VIRTUAL TERMINAL n'apparait pas activer le (touche droite) :





La chaine TGJOI123 est affichée dès que CTRL 0 est saisi.
Les minuscules sont transformées en majuscule.
La commande commence avec t.

Exécution d'une commande (dossier : 4. excecution commande).

On ajoute un fichier « exec_cmde.c ».

Le programme suivant est écrit dans ce fichier :

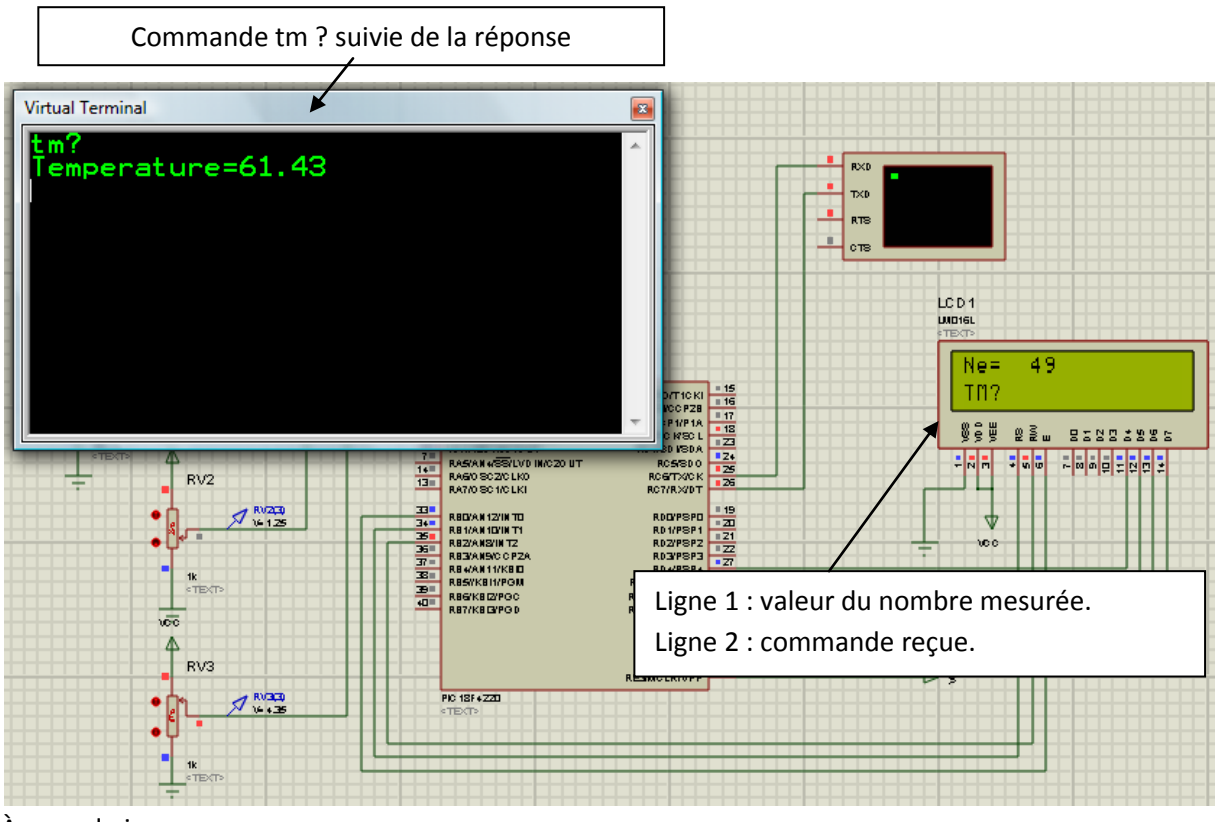
```
void exec_cmd_pc(char *cmde)
 {
  unsigned long value;
  switch (cmde[0]) {
   case 'T':
    switch (cmde[1]) {
      case 'C': /*commande temperature consigne*/
      break;
      case 'M': /*commande temperature mesuree*/
       if (cmde[2] == '?')
        {
          putchar(13); /*pour ESSAI retour a la ligne*/
          value = read_adc();
          value = value * 3000 / 1024 + 6000;
          printf("Temperature=%5.2w", value);
          putchar(13); /*pour ESSAI retour a la ligne*/
        }
      break;}
     break;
   case 'R':
     break;
   case 'N':
     break;}
 }
```

Le fichier « main.c » est modifié :

```
#include "main.h"
#include  "constante.c"
#include  "variable.c"
#include <lcd.c>
#include  "timer.c"
#include  "rs232.c"
#include  "init.c"
#include "exec_cmd.c"
```

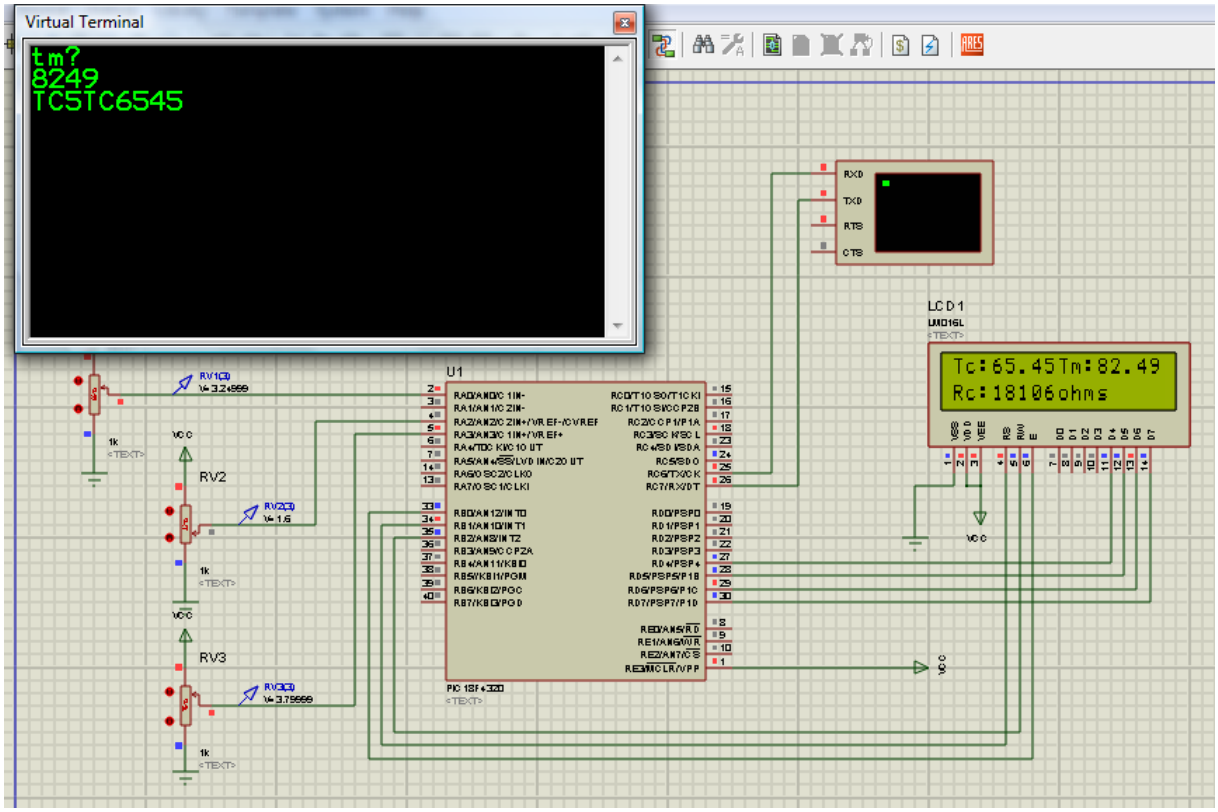
```
void main()
{
  unsigned long value;
  lcd_init();
  init();
  init_data();
 while (true)
  {
   value = read_adc();
   lcd_gotoxy(1,1);
   printf(lcd_putc,"Ne=%4Lu", value);
   if (commande_presente_pc)
    {
     lcd_gotoxy(1,2);
     printf(lcd_putc,"                "); /*efface affichage la commande*/
     lcd_gotoxy(1,2);
     printf(lcd_putc,buffer_reception_pc); /*affiche la commande*/
     exec_cmd_pc(buffer_reception_pc);
     commande_presente_pc = false;
     ptr_buffer_cmd_pc = 0;
    }
  }
}
```

Seule la commande « tm ? » est implantée. La simulation donne le résultat suivant :



À vous de jouer.

Complément – (dossier : 5. affichage total).



La température mesurée est affichée en permanence.

Les commandes actives depuis Virtual Terminal sont :

- TC? , NC?, TC6790 et TM?.

A vous de jouer.