François SCHNEIDER – Lycée Victor-Hugo BESANÇON.

STS SE

Développement de microcontrôleurs Microchip avec PICC
validation fonctionnelle PROTEUS

Utilisation Wizard et PROTEUS Simulation Validation

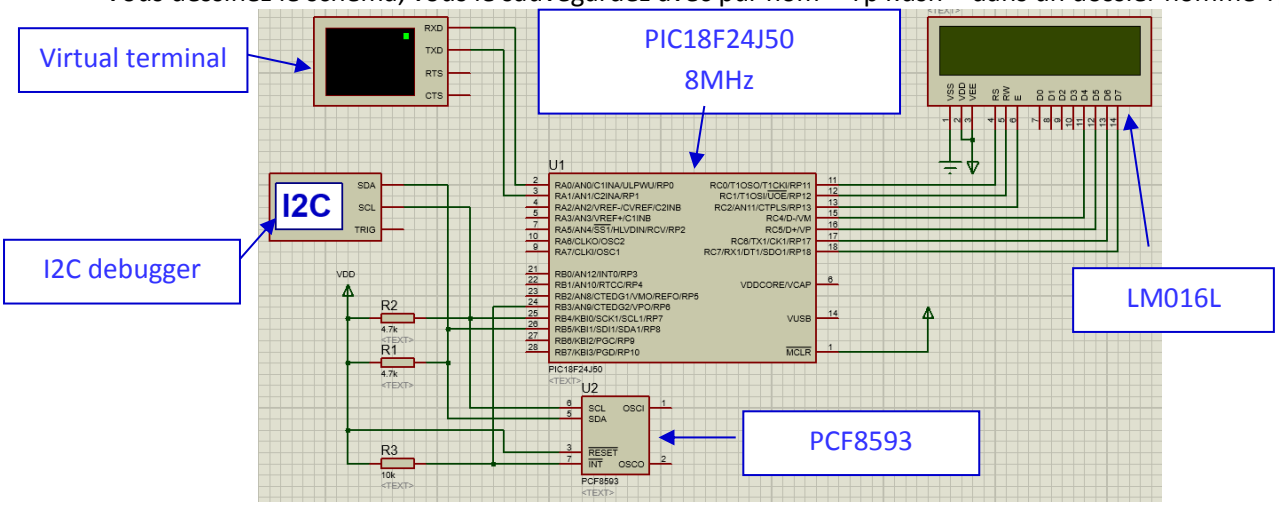
Interface I2C

Interruption

Prérequis : langage C, PROTEUS ISIS simulation d'un microprocesseur.

I. Le projet.

Le schéma du montage est donné page suivante avec le nom des composants en bleu (librairie ISIS). Vous dessinez le schéma, vous le sauvegardez avec pur nom « Tp4.dsn » dans un dossier nommé Tp4.



Nous trouvons 2 composants, qui appartiennent à la famille des « Instruments » : .

Le fonctionnement est assez simple :

- Le circuit PCF8593 est un circuit horloge, il permet d'avoir accès à tout moment à la date et à l'heure. La sortie INT/ produit un signal périodique carré de fréquence 1Hz.
- La liaison série 57600 bauds, pas de parité, 8 bits, 1 bit de stop.
- Chaque seconde le processeur lit la date et l'heure. Il affiche ces informations sur l'afficheur LCD et les envoie sur la liaison RS232 sous la syntaxe :
  - 'date jj/mm/aaaa\r\n'
  - 'time hh :mm :ss\r\n'

  \r étant le caractère de code ASCII 0x0D (CR).

  \n étant le caractère de code ASCII 0x0A (LF).

Première partie : afficheur lcd, liaison série.

Nous constatons que la première partie du I. du projet Tp3 ressemble à ce projet. Dans un premier temps nous allons donc faire le projet uniquement pour la partie afficheur lcd et RS232 à partir du projet Tp3. Lorsqu'un projet a été validé, il faut penser à le réutiliser.

Les différences sont :

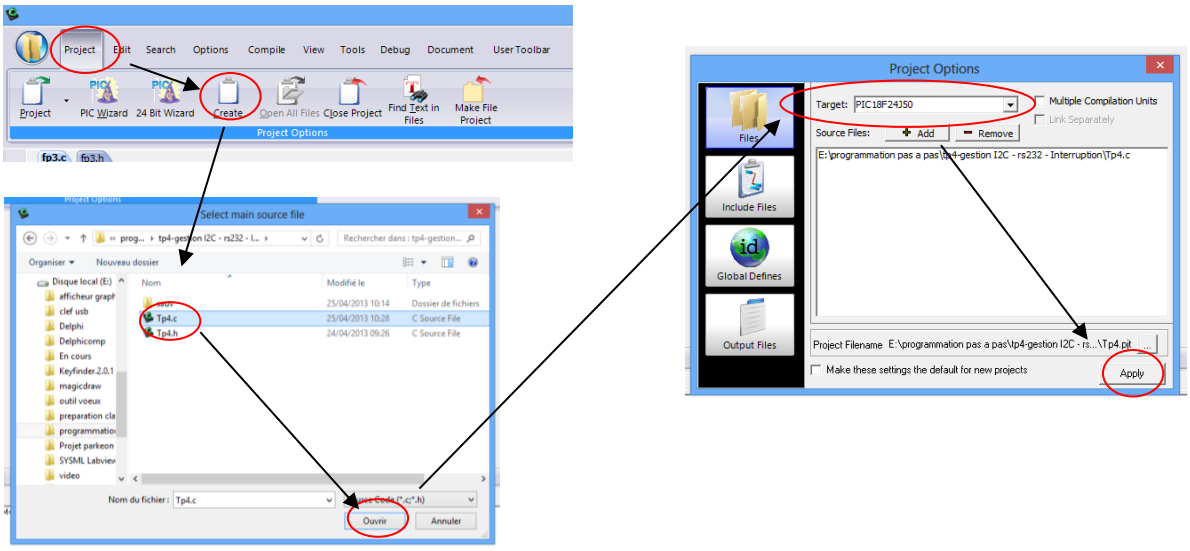
- Absence du circuit horloge.
- Câblage différent de l'afficheur et de l'interface RS232.
- Le microcontrôleur est différent mais très proche : taille mémoire différente.

a. Création du dossier projet : nous créons un dossier Tp4.

b. Proteus : il suffit de copier le fichier Tp3.dsn dans le nouveau dossier, de le renommer Tp4.dsn et de modifier le schéma pour la partie afficheur et liaison série.

c. PICC :
   - Copier les fichiers Tp3.c, Tp3.h, les renommer en Tp4.c et Tp4.h.
   - Créer manuellement un nouveau Tp4.pjt à partir des fichiers Tp4.c et Tp4.h.

Nous devons maintenant créer manuellement le projet.



Votre projet est créé, vous pouvez travailler avec de la même façon, que s'il avait été créé avec le wizard. Cette méthode est utile lorsque vous récupérez un projet qui ne fonctionne pas en le rechargeant.

- Modifier les fichiers programmes pour les adapter au nouveau schéma. Ils deviennent :

TP4.c	Tp4.h

#include <Tp4.h>
#define LCD_ENABLE_PIN PIN_C2
#define LCD_RS_PIN PIN_C0
#define LCD_RW_PIN PIN_C1
#define LCD_Data4 PIN_C4
#define LCD_Data5 PIN_C5
#define LCD_Data6 PIN_C6
#define LCD_Data7 PIN_C7
#include <lcd.c>
 	
#PIN_SELECT U2TX = PIN_A0
// la sortie TX de l UART2 est affectee a PIN_A0
#PIN_SELECT U2RX = PIN_A1
// l'entree RX de l UART2 est affectee a PIN_A1
#use rs232(UART2,baud=57600,parity=N,bits=8)
//ici nous definissons les fonctions pour l UART 2
 	
VOID main()
{
lcd_init () ;
output_high(PIN_A0);
printf (lcd_putc, "Tp3 - afficheur LCD\n");
printf("Tp3 - interface serie ");
WHILE (TRUE)
{
if (kbhit()) // on teste si une donnee est presente.
{
lcd_putc(getc()); // on lit la donnee et on l'affiche.
}
}
}
	#include <18F24J50.h>
	#device ICD=TRUE
	#device adc=16
	
	#FUSES NOWDT //No Watch Dog Timer
	#FUSES WDT128 //Watch Dog Timer uses 1:128 Postscale
	#FUSES PLL2 //Divide By 2(8MHz oscillator input)
	#FUSES NOXINST //Extended set extension and Indexed Addressing mode disabled (Legacy mode)
	#FUSES DEBUG //Debug mode for use with ICD
	#FUSES HSPLL //High Speed Crystal/Resonator with PLL enabled
	#use delay(clock=8000000)

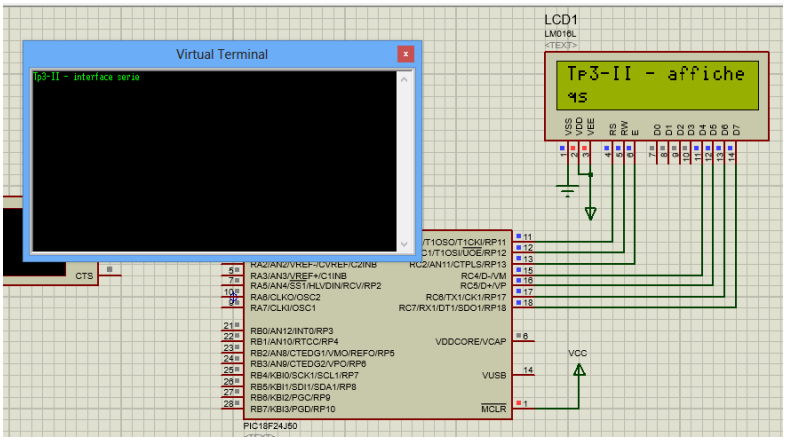
Nous simulons le fonctionnement à l'aide de Proteus. Il faut penser à paramétrer le processeur (fréquence, fichier programme) et les paramètres de 'Virtual Terminal'. Cela fonctionne correctement, nous allons pouvoir réaliser la librairie pour le circuit horloge.

Deuxième partie : projet global sans gestion du signal INT/.

Afin de ne pas perdre le travail effectué, nous allons :

- Créer un nouveau schéma sous Proteus nommé Tp4-II.dsn. Il suffit d'enregistrer le schéma actuel sous ce nouveau nom dans le même dossier.
- Créer un nouveau projet sous PICC dans le même dossier. Pour ce, il faut enregistrer le fichier Tp4.C en Tp4-II.c et créer un nouveau projet nommé Tp4-II.pjt en manuel à partir de ce fichier (voir section précédente).

Une fois ceci réalisé, vous vérifiez le bon fonctionnement en pensant à changer le fichier programme du processeur dans Proteus (pensez au niveau du programme dans PICC à changer le message affiché). Vous obtenez ceci :



Bien sûr, lorsque vous tapez des codes dans 'Virtual terminal', ils sont affichés sur l'afficheur LCD.

Compléter le schéma : vous complétez le schéma avec le composant PF8593, les résistances de tirage et 'l'ICD debugger'.

Modifier le programme : après analyse des besoins, nous constatons que nous aurons différentes fonctions à créer. Afin de faciliter la programmation, nous allons créer des variables globales pour les grandeurs date et heure.

Ces variables prennent des valeurs sur 8 bits (entiers non signés) car leurs valeurs ne dépassent jamais 255 sauf pour l'année, qui peut prendre une valeur plus grande que 255 et sera donc codée sur 16 bits :

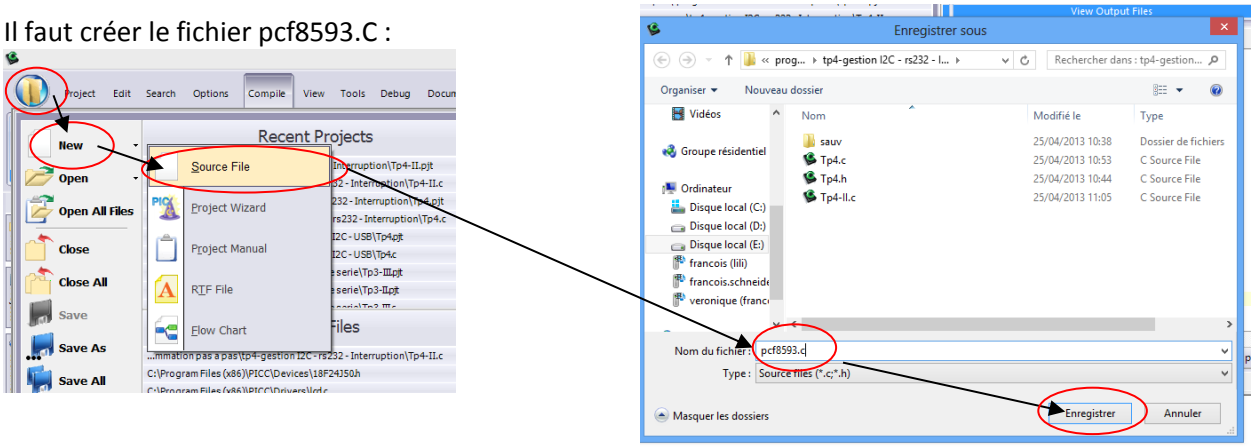
```
UNSIGNED int heure;
UNSIGNED int minute;
UNSIGNED int seconde;
UNSIGNED int jour;
UNSIGNED int mois;
UNSIGNED long annee;
```

Les fonctions dont nous aurons besoin sont :

- Configurer le composant pcf8593 : Void Init_pcf8593(void)
- Mettre la date du composant à jour :
  - Void Ecrire_date_pcf8593(void)
- Mettre le composant à l'heure :
  - Void Ecrire_heure_pcf8593(void)
- Lire la date et l'heure :
  - Void lire_date_heure_pcf8593(void)

Nous allons donc créer un fichier bibliothèque pour ce composant, que nous allons nommer pcf8593.c.

Il faut créer le fichier pcf8593.C :



Le fichier pcf8593.c est créé.
Vous déclarez le fichier librairie dans le fichier tp4-II.C en ajoutant la directive #include <pcf8593.c> sous la directive #include <lcd.c>. Vous pouvez compiler à nouveau votre programme, il ne doit pas y avoir d'erreur.

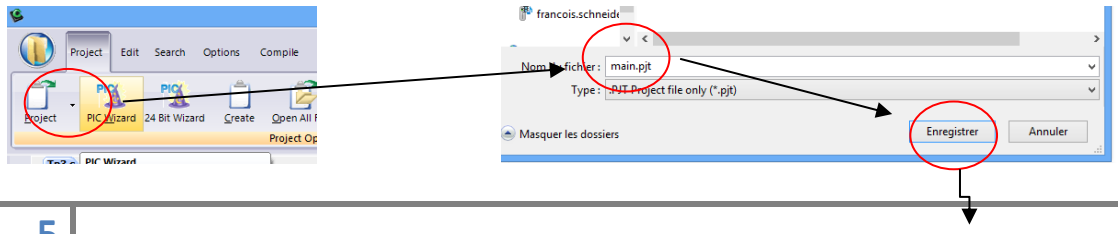
Quelques rappels sur l'interface I2C :

- L'interface I2C est un bus synchrone bidirectionnel. Il n'est pas full-duplex.
- Nous rencontrons des maitres et des esclaves. Il peut être multi-maitres mais cela complique la gestion. Ici nous avons un seul maitre.
- Chaque esclave possède une adresse unique.
- Les différents composants sont reliés au bus à travers des signaux collecteurs ouverts, il faut donc des résistances de tirage. Ces résistances se calculent en fonction des composants présents et de la vitesse du bus.

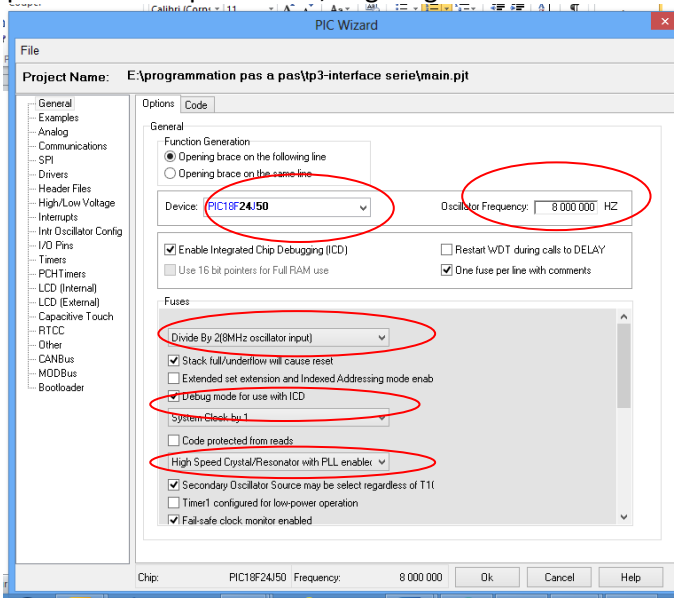
Pour plus d'information voir : https://fr.wikipedia.org/wiki/I2C.

1. Configurer la liaison I2C et test. Nous allons utiliser le wizard pour compléter le programme actuel.

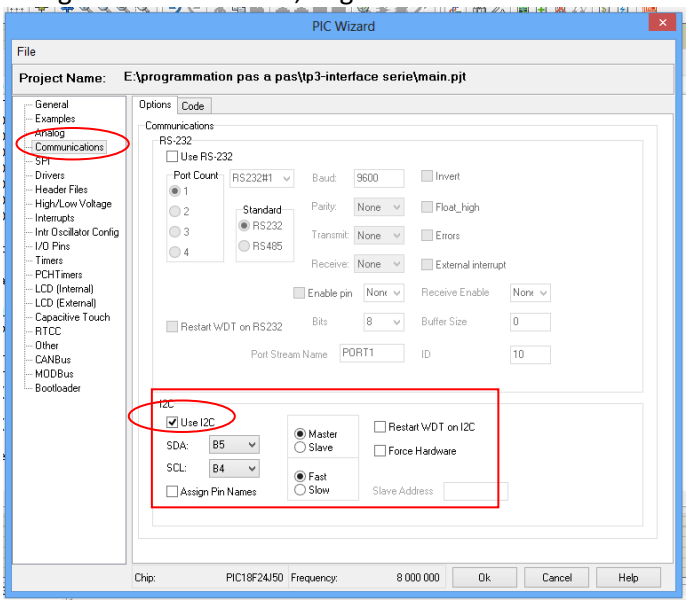
Nous lançons le wizard et laissons le nom du projet par défaut : 'main.pjt', cela évitera, en cas d'erreur de manipulation, d'écraser le projet en cours d'utilisation.



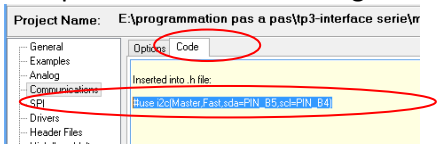
Nous configurons les paramètres du processeur, onglet ‘general’.



Nous allons ensuite configurer l’interface I2C, onglet ‘Communication’.



Maintenant nous allons voir le code produit en sélection l’onglet en haut ‘Code’.



Vous sélectionnez la directive #use I2C….

Vous la copiez dans le presse-papier en faisant au clavier ‘CTRL C’.

Vous quittez le wizard avec le bouton ‘Cancel’.

Et copiez la directive dans le fichier Tp4-II.C avant la ligne #include <pcf8593.c>.

Le programme est prêt pour utiliser l’interface I2C.

Nous allons tester les fonctions associées à l'interface I2C. Les principales fonctions sont :

- I2C_START() : pour émettre un start.
- I2C_write(data) : pour écrire un octet.
- I2C_read() : pour lire un octet.
- I2C_stop : pour émettre un stop.

Nous plaçons un oscilloscope au niveau de la liaison SPI, le câblage est le suivant :

- Voie A : INT.
- Voie B : SCL.
- Voie C : SDA.

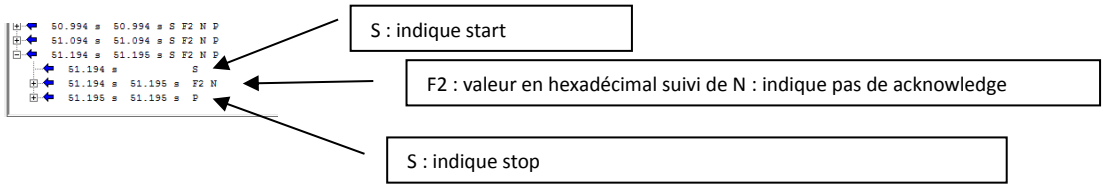
Nous synchroniserons l'oscilloscope sur SCL et sur front négatif.

Je vous propose de remplacer la fonction main() par :

```
VOID main()
  {
  lcd_init () ;
  output_high(PIN_A0);
 // printf (lcd_putc, "Tp3-II - afficheur LCD\n");
 //printf("Tp3-II - interface serie ");
  WHILE (TRUE)
    {
    I2c_start() ;
    i2c_write(0xF2);
    I2c_stop();
    delay_ms(100);
    /*
    if (kbhit())   // on teste si une donnee est presente.
    {
     lcd_putc(getc()); // on lit la donnee et on l'affiche.
    }
   */
    }
  }
```

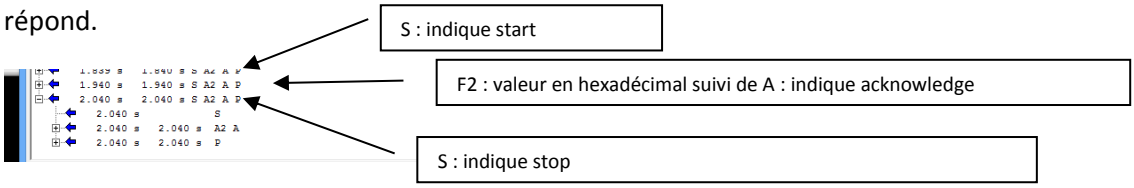
Vous lancez la simulation, les données reçues par 'l'I2C debugger' évoluent.

Vous passez en mode pause (en bas à gauche) et constatez au niveau de l'I2C debugger :

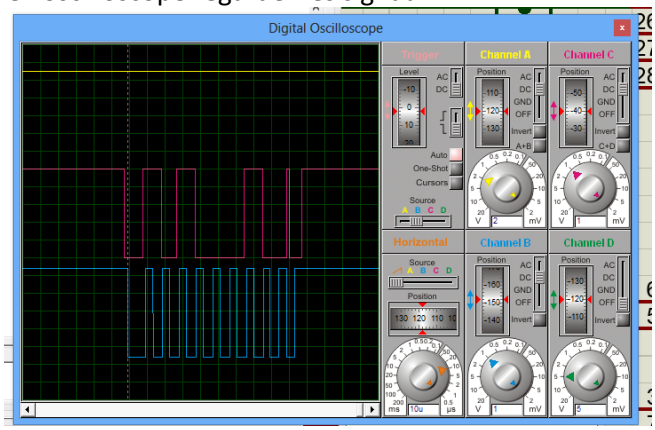


Le fait qu'il n'y ait pas de 'acknowledge' indique qu'aucun circuit d'adresse F2 est présent. Puisque la première valeur envoyée sur le bus est l'adresse de l'esclave que le maitre désire contacter.

Nous allons changer cette adresse par l'adresse du pcf8593 : 0xA2. Cette fois le circuit pcf8593 répond.



Nous pouvons à l'aide de l'oscilloscope regarder les signaux :



Nous reconnaissons en début le start et en fin le stop.
Nous allons passer à la partir 2.

2. Programmation du circuit PCF8593.

Nous allons compléter le fichier pcf8593 comme ceci :

PCF8593.c

```
UNSIGNED int heure;
UNSIGNED int minute;
UNSIGNED int seconde;
UNSIGNED int jour;
UNSIGNED int mois;
UNSIGNED long annee;

const unsigned long base_annee = 2012;

Void Init_pcf8593(void)
{
}
Void Ecrire_date_pcf8593(void)
{
}
Void Ecrire_heure_pcf8593(void)
{
}
Void lire_date_heure_pcf8593(void)
{
}
```

Il nous faut écrire les différentes fonctions, nous allons commencer par la fonction d'initialisation. Si nous regardons la documentation du composant Il faut configurer le registre à l'adresse 0.

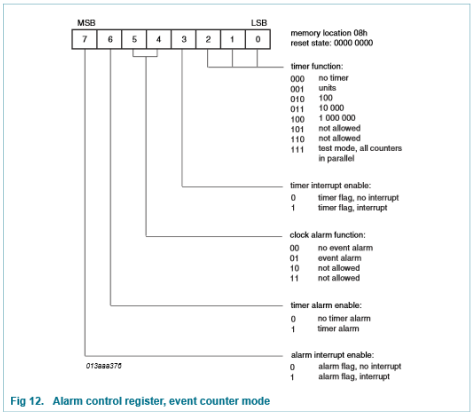


Fig 12. Alarm control register, event counter mode

Il nous faut écrire la valeur 0 dans ce registre. Nous allons utiliser le mode ci-dessous :

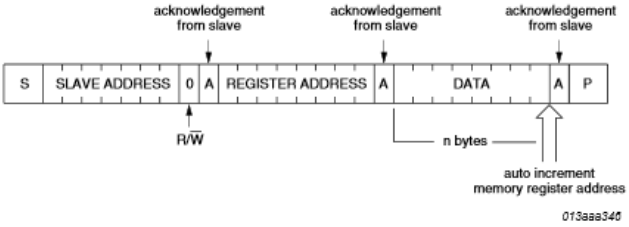


Fig 18. Master transmits to slave receiver (WRITE mode)

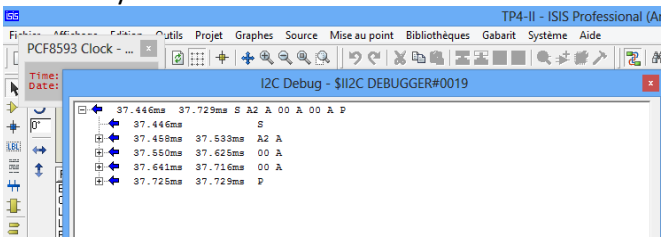
Remarque : le premier mot transmis contient l'adresse de l'esclave ainsi que pour le bit de poids faible l'indication de lecture ou écriture. Le programme du PCF8593 devient :

```
void init_pcf8593(VOID)
{
  I2C_START (); // start
  I2C_WRITE (0XA2); // j'adresse en ecriture le composant 0xA2 PCF8593
  I2C_WRITE (0); // adresse du registre de configuration a ecrire 0
  I2C_WRITE (0); // valeur a mettre dans le registre de configuration
  I2C_STOP ();   stop
}
```

Vous modifiez la fonction void init_pcf8593(VOID) dans pcf8593.c.
Vous modifiez la fonction main() du programme principal :

```
VOID main()
 {
 lcd_init () ;
 Init_pcf8593();
 output_high(PIN_A0);
// printf (lcd_putc, "Tp3-II - afficheur LCD\n");
//printf("Tp3-II - interface serie ");
 WHILE (TRUE)
   {
   /*
   if (kbhit())   // on teste si une donnee est presente.
   {
    lcd_putc(getc()); // on lit la donnee et on l'affiche.
   }
  */
  }
 }
```

Vous lancez la simulation et analysez le résultat.



Une seule trame I2C est transmise et correspond à la trame d'initialisation de l'horloge.
Si nous regardons l'oscilloscope nous constatons que le signal INT/ présente une oscillation de période 2s, alors que nous avions prévu 1s.

Cela est à vérifier sur le circuit réel, il se peut que nous rencontrions un bug de proteus.

Nous allons écrire la fonction de lecture du circuit pcf8593 : Void lire_date_heure_pcf8593(void)
L'organisation des registres internes est la suivante :

control/status		
hundredth of a second	1/10 s	1/100 s
seconds	10 s	1 s
minutes	10 min	1 min
hours	10 h	1 h
year/date	10 day	1 day
weekdays/months	10 month	1 month
timer	10 day	1 day
alarm control		
hundredth of a second alarm	1/10 s	1/100 s
alarm seconds		
alarm minutes		
alarm hours		
alarm date		
alarm month		
alarm timer		

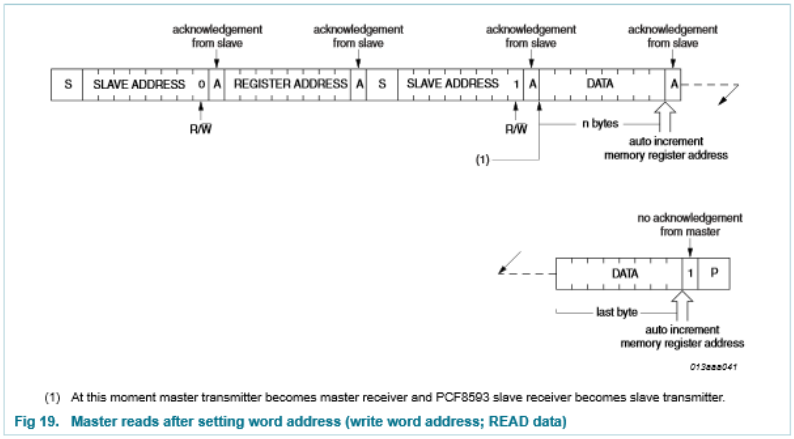
Nous constatons, que :

Les secondes se situent à l'adresse 2, et que les registres suivants présentent minutes, …

Les données sont codées en BCD, si nous voulons les stocker en décimal, il faudra effectuer un conversion bcd → décimal.

2 registres contiennent des informations multiples.

La lecture peut être effectuée en commençant par un registre choisi et qu'ensuite la lecture se fait par incrémentation automatique du numéro de registre. Voici les chronogrammes :



(1) At this moment master transmitter becomes master receiver and PCF8593 slave receiver becomes slave transmitter.

Fig 19. Master reads after setting word address (write word address; READ data)

Le principe est le suivant :

- Start
- Envoi de l'adresse du composant avec r/w = 0
- Envoi de l'adresse du registre par exemple 2 pour les secondes
- Start (ceci s'appelle un restart)
- Envoi de l'adresse du composant avec r/w = 1
- Lecture du registre dont l'adresse a été envoyée plus haut
- Lecture … l'adresse s'auto incrémente
- stop

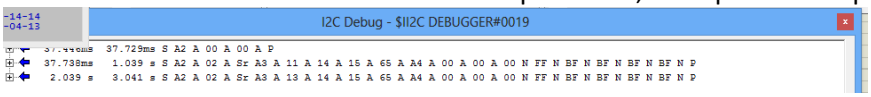
Ceci donne la fonction :

```
Void lire_date_heure_pcf8593(void)
{
  I2C_START ();    // start
  I2C_WRITE (0XA2); // j'adresse en lecture le composant 0xA2 PCF8593
  I2C_WRITE (2);   // j'indique que je lis le registre  2 = seconde
  I2C_START ();    // restart
  I2C_WRITE (0XA3); // j'adresse en lecture le composant PCF8593
  seconde = I2C_READ ();  // 2 on lit les secondes (BCD)
  minute = I2C_READ ();  // 3 lecture minute
  heure = I2C_READ ();   //4
  jour = I2C_READ ();    //5
  mois = I2C_READ ();   //6
  I2C_STOP ();
}
```

Vous copiez cette fonction dans votre projet.
Vous ajoutez au niveau de la fonction main dans la boucle while :

```
 WHILE (TRUE)
     {
     lire_date_heure_pcf8593();
     delay_ms(1000);
     /*
     if (kbhit())   // on teste si une donnee est presente.
     {
      lcd_putc(getc()); // on lit la donne et on l'affiche.
     }
     */
     }
```

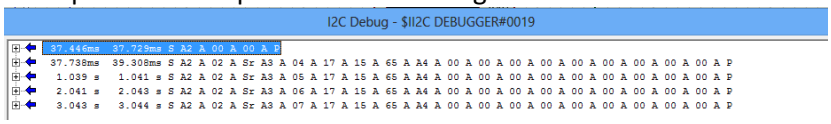
Vous compilez et testez en simulant. Nous constatons un problème, le stop ne se fait pas ?



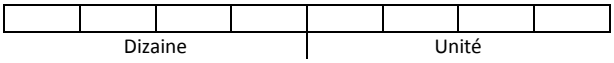
Il existe une solution en lisant les registres du pcf8593 jusqu'aux derniers. Ce problème est à vérifier mais semble un problème dû à Proteus. La fonction devient :

```
Void lire_date_heure_pcf8593(void)
{
  I2C_START ();    // start
  I2C_WRITE (0XA2); // j'adresse en lecture le composant 0xA2 PCF8593
  I2C_WRITE (2);   // j'indique que je lis le registre  2 = seconde
  I2C_START ();    // restart
  I2C_WRITE (0XA3); // j'adresse en lecture le composant PCF8593
  seconde = I2C_READ ();  // 2 on lit les secondes (BCD)
  minute = I2C_READ ();  // 3 lecture minute
  heure = I2C_READ ();   //4
  jour = I2C_READ ();    //5
  mois = I2C_READ ();   //6
   I2C_READ (); // 7 voir en pratique.
  I2C_READ (); // 8
  I2C_READ (); // 9
  I2C_READ (); // 10
  I2C_READ (); // 11
  I2C_READ (); // 12
  I2C_READ (); // 13
  I2C_READ (); // 14
  I2C_READ (); // 15
  I2C_STOP ();
```

La simulation donne un résultat cohérent notamment nous voyons les secondes augmenter à chaque lecture. Il est possible de repérer chacun des registres ci-dessous :



Nous devons maintenant compléter la fonction précédente pour obtenir chacune des grandeurs codées en décimal et non pas en BCD.
Nous devons créer une fonction, qui convertit un nombre BCD en décimal, le principe et le suivant :



Pour traduire ce nombre en codé décimal, il faut calculer 10 x Dizaine + Unité. Or ce nombre est un mot de 8 bits, il faut opérer ainsi :

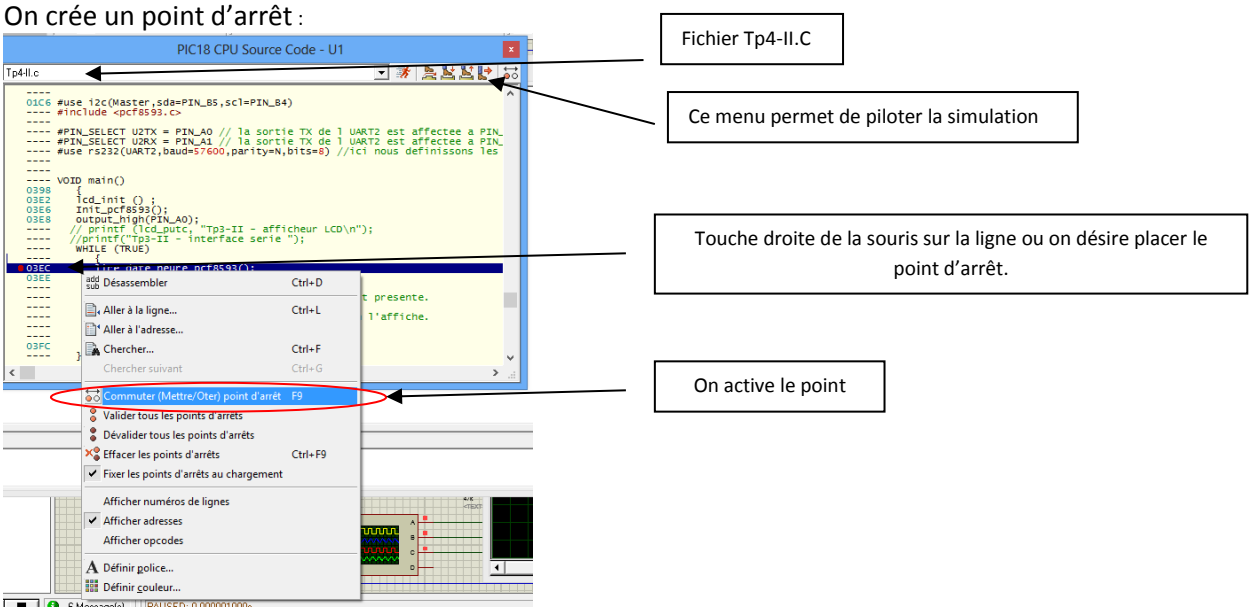
result = ((bcd>>4) * 10) + (bcd & 0x0F);

Comme cette opération de conversion va s'appliquer à plusieurs grandeurs, il est plus simple de créer une fonction :

unsigned int bcdtodecimal(UNSIGNED int bcd)
{
UNSIGNED int result;
result = ((bcd>>4) * 10) + (bcd & 0x0F);
RETURN result;
}
	La fonction reçoit un entier non signé en entrée
	
	Cette fonction renvoie un entier non signé.

La directive return indique la grandeur que la fonction renvoie.
Copier cette fonction dans pcf8503.c et appliquer aux variables heure, minute et seconde.

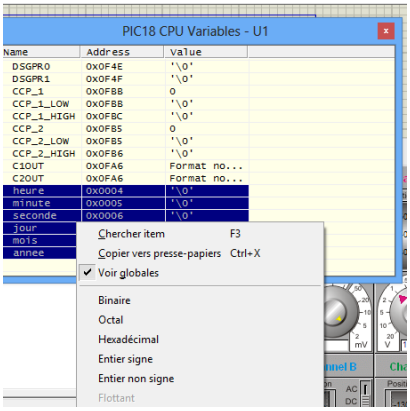
Pour vérifier le bon fonctionnement nous allons utiliser **Proteus** avec des points d'arrêt dans le programme.
On lance la simulation avec : 
On crée un point d'arrêt :



Ce menu présent en haut à droite de la fenêtre permet de piloter la simulation.

On peut à nouveau lancer la simulation avec  , le programme s'arrête sur la ligne du point d'arrêt.
Ensuite on avance en pas à pas en entrant dans les fonctions avec  ..

Il faut cependant définir la façon dont les variables sont affichées : On sélectionne le type « entier non signé ».



Nous allons regarder l'évolution des variables en avançant le programme.
Vous vérifiez que le fonctionnement est correct.
Nous allons compléter le programme pour obtenir la date.

Après avoir effectuez « jour = I2C_READ (); //5 », la variable 'jour' contient le contenu de l'adresse 5 :

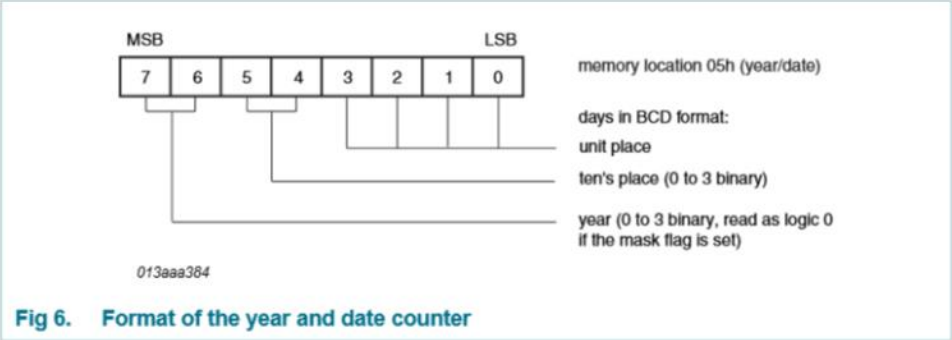


Fig 6. Format of the year and date counter

L'année est représentée par les Bits 6 et 7. Pour obtenir l'année, il suffit de décaler 6 fois à droite.
Le jour est représenté en BCD par les bits 5 à 0. Pour obtenir le jour il suffit de mettre à 0 les bits 6 et 7 et de convertir de BCD en décimal.

Les lignes de programme suivantes permettent d'obtenir l'année et le jour :

```
jour = I2C_READ ();    //5
annee = jour;
annee = (annee>>6) + base_annee;  // decalage droite pour garder annee et on ajoute 2012
jour = bcdtodecimal  (jour&0x3F);  // masquage pour garder uniquement jour
```

Vous intégrez ces lignes dans le programme et vous validez le fonctionnement.

Après avoir effectué «mois = I2C_READ ();//6», la variable 'mois' contient le contenu de l'adresse 6 :

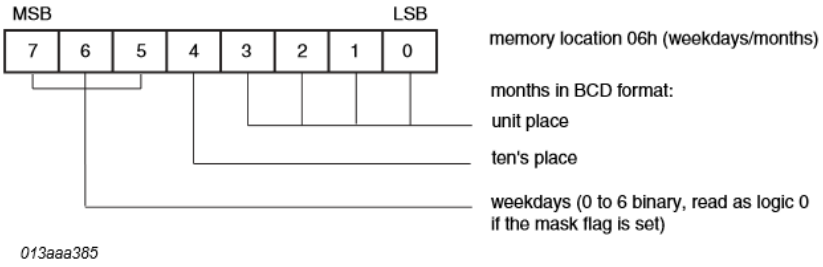


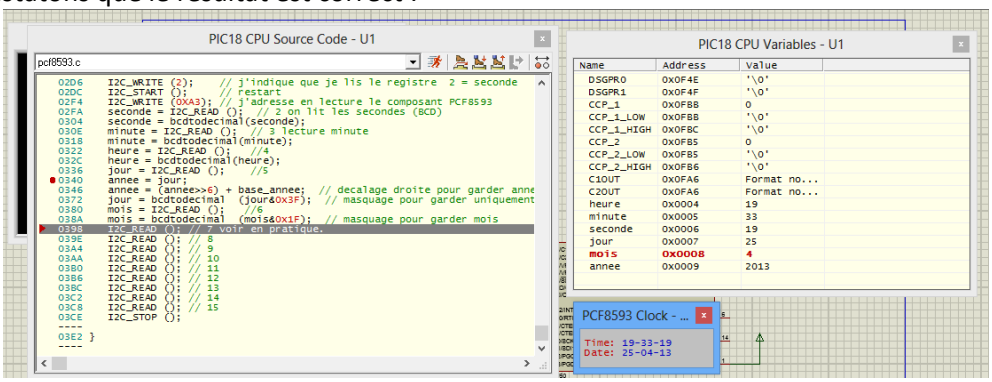
Fig 7. Format of the weekdays and month counter

Le mois est représenté en BCD par les bits 4 à 0. Pour obtenir le mois il suffit de mettre à 0 les bits 5, 6 et 7 et de convertir de BCD en décimal.

Les lignes de programme suivantes permettent d'obtenir l'année et le jour :

```
mois = I2C_READ ();   //6
mois = bcdtodecimal  (mois&0x1F);  // masquage pour garder mois
```

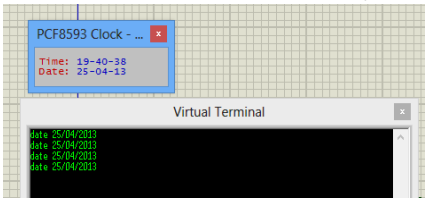
Vous intégrez ces lignes dans le programme et vous validez le fonctionnement.
Nous constatons que le résultat est correct :



La fonction est terminée, nous allons compléter le programme principal pour envoyer les commandes à la liaison série. Pour cela nous allons utiliser la fonction printf.
La fonction while devient :

```
WHILE (TRUE)
    {
    lire_date_heure_pcf8593();
    printf("date %02u/%02u/%04Lu\r\n",jour,mois,annee);  // afficher la date.
    delay_ms(1000);
    /*
    if (kbhit())   // on teste si une donnee est presente.
    {
     lcd_putc(getc()); // on lit la donnee et on l'affiche.
    }
    */
    }
```

Vous modifiez la fonction main() et validez son fonctionnement, cela donne :



Il ne vous reste plus qu'à faire la même chose pour l'heure.

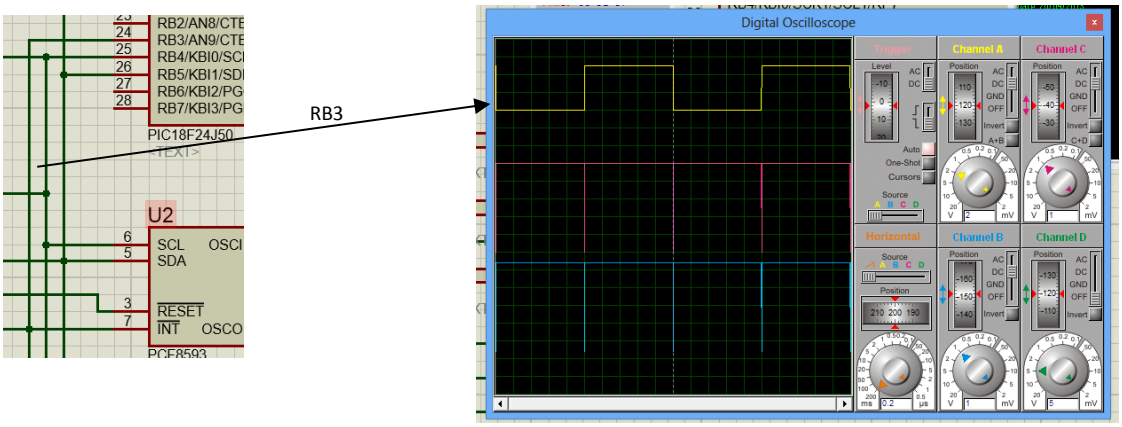
Si vous le désirez-vous pouvez aussi afficher la date et l'heure sur l'écran LCD.

Remarque : le programme possède un défaut important, lorsque le processeur effectue la fonction delay_ms(1000), il ne peut rien faire d'autre. Il faut trouver une solution pour supprimer cette fonction.

Le signal INT/ produit par le circuit va nous permettre de tester si une information de date est présente. Nous allons donc utiliser ce signal. Deux solutions vont être développées ci-dessous :

- Par scrutation : le programme lit l'état du bit et effectue alors la tâche qui lui est attribuée.
- Par interruption : lorsqu'un évènement se produit sur le bit par exemple un front montant, alors le processeur termine la tâche qu'il est en train de réaliser pour aller effectuer une tâche attribuée à cette interruption.

Troisième partie : projet global gestion du signal INT/ par scrutation.



Le signal INT relié à l'entrée RB3 du processeur présente un signal périodique de fréquence. La détection du front montant sur ce signal permet d'acquérir l'information horaire sans bloquer le processeur pendant l'exécution de la temporisation.
Comment détecter le front sur RB3 ? Plusieurs solutions sont possibles, je vous propose celle-ci :
Si RB3 est à l'état haut et que RB3 était l'instant précédent à l'état bas alors RB3 vient d'avoir un front montant.

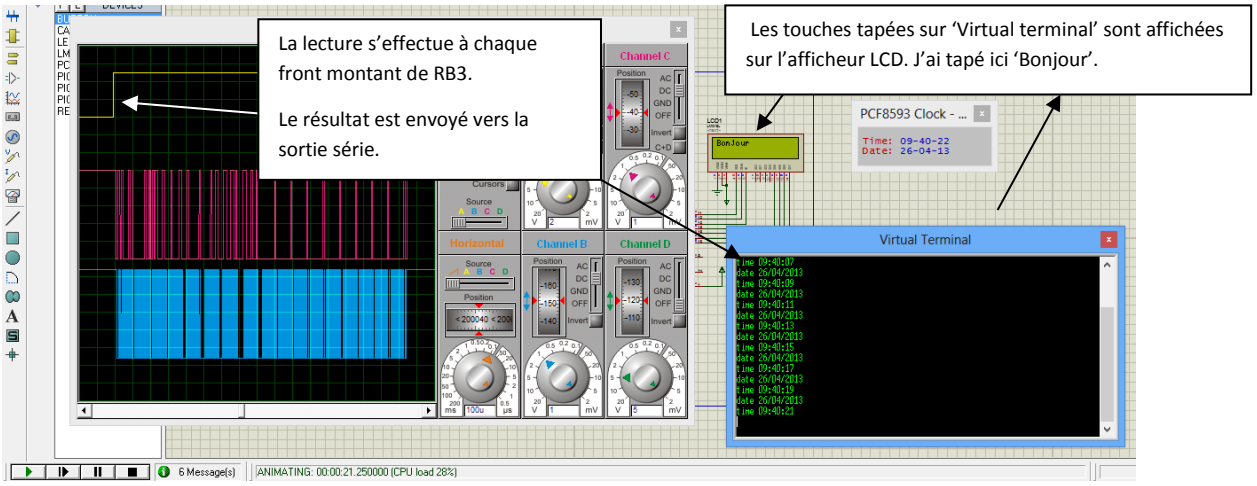
Application en langage C :

```
VOID main()
  {
  short mem_int_pcf8593, anc_int_pcf8593;
  lcd_init () ;
  Init_pcf8593();
  output_high(PIN_A0);
 mem_int_pcf8593 = input(PIN_B3);
  WHILE (TRUE)
    {
     anc_int_pcf8593 = mem_int_pcf8593;
     mem_int_pcf8593 = input(PIN_B3);
    if ((mem_int_pcf8593==1) & (anc_int_pcf8593==0))
     {
     lire_date_heure_pcf8593();
     printf("date %02u/%02u/%04Lu\r\n",jour,mois,annee);  // afficher la date.
     printf("time %02u:%02u:%02u\r\n",heure,minute,seconde);  // afficher la date.
     }
    /*if (kbhit())   // on teste si une donnee est presente.
     lcd_putc(getc()); // on lit la donnee et on l'affiche.*/
   }
  }
```

Remarques :

- L'instruction delay_ms(1000) est supprimée.
- Les instructions ajoutées sont indiquées en rouge.
- Le programme n'est donc plus bloquant. Dans la boucle d'autres taches peuvent être effectuées. Il est possible de lire une information en provenance de la liaison RS232 en activant le texte en bleu en même temps que la lecture et l'envoi de la date vers la liaison RS232.

Il ne reste plus qu'à valider cela. Voici le résultat en ayant activé les lignes en bleu.



Nous avons donc l'impression que les deux tâches sont exécutées en simultané.

Quatrième partie : projet global gestion du signal INT/ par interruption.

Une troisième solution existe en utilisant les ressources interruption du microcontrôleur. Le microcontrôleur 18F24J50 possède plusieurs sources d'interruption, qui ont chacune un degré de priorité, cela veut dire que si plusieurs interruptions arrivent en simultané elles vont être traitées dans un ordre prédéfini.

Les interruptions ont plusieurs origines :

- Un évènement extérieur dû à un état ou changement d'état d'un signal :
    - Certaines broches sont affectées à ce fonctionnement : par exemple l'évènement RESET.
    - Le microcontrôleur 18F24J50 possède des ressources interruption, qui sont affectables à des broches choisies par exemple :
        - "#pin_select INT1=PIN_B3" permet d'affecter l'interruption INT1 à la borne B3.
- Un évènement extérieur lié à une ressource UART,….
- Un évènement intérieur comme un timer…
- ….

Nous allons voir comment utiliser la borne B3 avec l'interruption INT1.

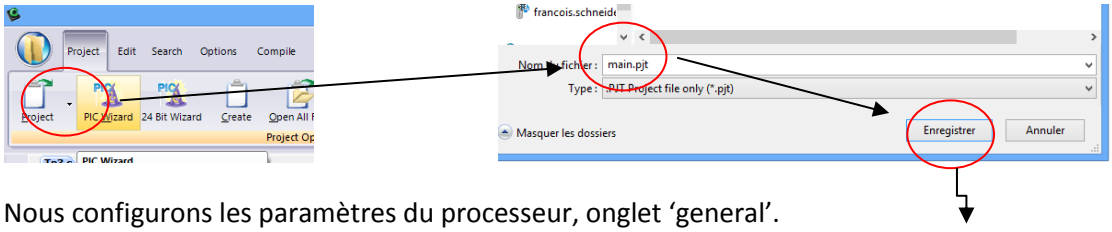
Avec le microcontrôleur PI18F24J50, il faut :

- Affecter la ressource : #pin_select INT1=PIN_B3 (vous trouvez la liste des affectations possibles dans l'aide de #pin_select).
- Configurer et activer l'interruption INT1.
- Ecrire le programme d'interruption.
- Activer le masque d'interruption global.

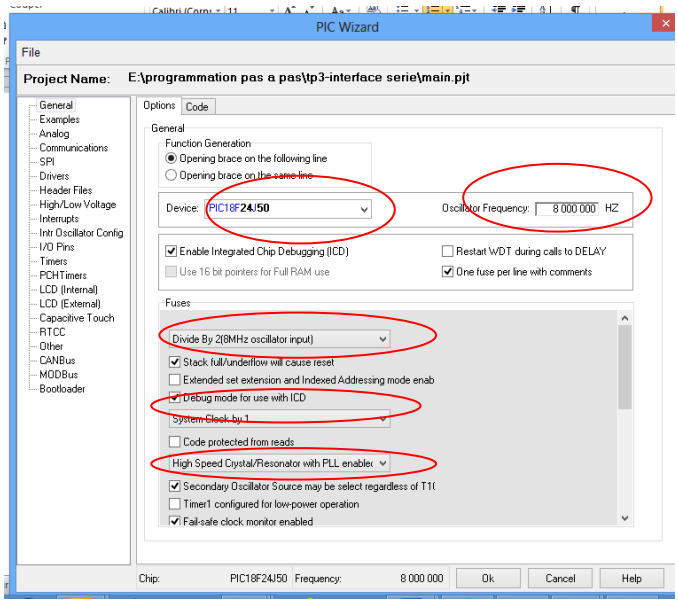
Pour cela nous allons utiliser le wizard. Dans un premier temps, je vous conseille d'enregistrer le fichier Tp3-II.c en Tp3-III.C et de créer manuellement un nouveau projet pour ce fichier. Ceci vous permet de garder Tp3-II.C.

Vous devez au niveau de Proteus indiquer le nouveau fichier source.

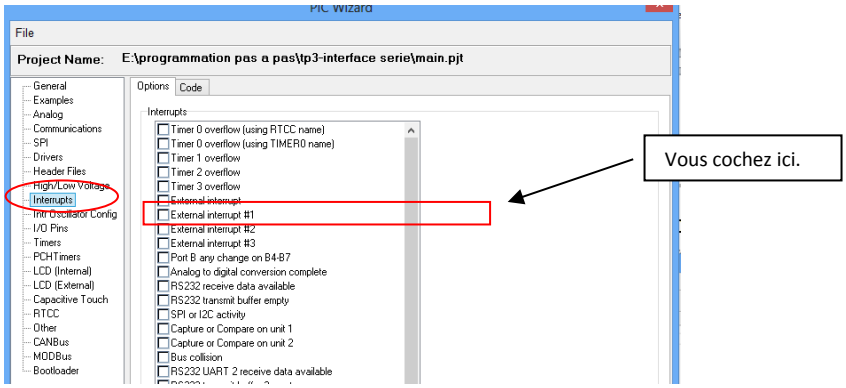
Nous lançons le wizard et laissons le nom du projet par défaut : 'main.pjt', cela évitera, en cas d'erreur de manipulation, d'écraser le projet en cours d'utilisation.



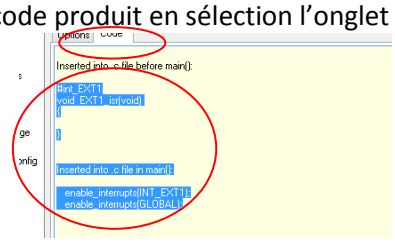
Nous configurons les paramètres du processeur, onglet 'general'.



Nous allons ensuite configurer l'interruption sur INT1, onglet 'Interrupts'.



Maintenant nous allons voir le code produit en sélection l'onglet en haut 'Code'.



Vous sélectionnez la zone en bleue et la copiez dans le presse-papier en faisant au clavier 'CTRL C'.

Vous quittez le wizard avec le bouton 'Cancel'.

Et intégrez la zone copiée dans le fichier Tp3-III.C.

Pensez à ajouter la ligne #pin_select INT1=PIN_B3 avant la fonction interruption.

Ce programme devient :

```
#include <Tp4.h>
#define LCD_ENABLE_PIN PIN_C2
#define LCD_RS_PIN PIN_C0
#define LCD_RW_PIN PIN_C1
#define LCD_Data4 PIN_C4
#define LCD_Data5 PIN_C5
#define LCD_Data6 PIN_C6
#define LCD_Data7 PIN_C7

#include <lcd.c>

#use i2c(Master,sda=PIN_B5,scl=PIN_B4)
#include <pcf8593.c>

#PIN_SELECT U2TX = PIN_A0 // la sortie TX de l UART2 est affectee a PIN_A0
#PIN_SELECT U2RX = PIN_A1 // la sortie TX de l UART2 est affectee a PIN_A1
#use rs232(UART2,baud=57600,parity=N,bits=8) //ici nous definissons les fonctions pour l UART 2


#pin_select INT1=PIN_B3
#int_EXT1
void  EXT1_isr(void)
{
     lire_date_heure_pcf8593();
     printf("date %02u/%02u/%04Lu\r\n",jour,mois,annee);  // afficher la date.
     printf("time %02u:%02u:%02u\r\n",heure,minute,seconde);  // afficher la date.
}

VOID main()
  {
  lcd_init () ;
  Init_pcf8593();
  output_high(PIN_A0);
  enable_interrupts(INT_EXT1);
  enable_interrupts(GLOBAL);
  WHILE (TRUE)
    {
    if (kbhit())   // on teste si une donnee est presente.
     lcd_putc(getc()); // on lit la donnee et on l'affiche.
    }
  }
```

En rouge, les lignes ajoutées. Vous pouvez tester le programme, il a le même fonctionnement qu'en scrutation, sauf, qu'il n'est plus nécessaire de tester la ligne RB3.

Il est bien sur possible d'utiliser les entrées interruption pour des boutons ou d'autres éléments.

Dans le programme ci-dessus, il y a une erreur, qui consiste à transmettre ou à afficher des informations dans un programme interruption. Le risque étant de commencer à transmettre un message vers l'interface série, une interruption prend la main et envoie à son tour un message vers la même interface… On constate qu'il y aura un mélange des messages.

En principe dans une interruption, on effectue l'acquisition des entrées et ensuite on indique qu'une donnée est présente à l'aide d'une variable. Cette variable est appelée drapeau. Le programme ci-dessous vous donne un exemple.

```
#include <Tp4.h>
short date_presente; // creation d une variable qui sert de drapeau
#define LCD_ENABLE_PIN PIN_C2
#define LCD_RS_PIN PIN_C0
#define LCD_RW_PIN PIN_C1
#define LCD_Data4 PIN_C4
#define LCD_Data5 PIN_C5
#define LCD_Data6 PIN_C6
#define LCD_Data7 PIN_C7

#include <lcd.c>

#use i2c(Master,sda=PIN_B5,scl=PIN_B4)
#include <pcf8593.c>

#PIN_SELECT U2TX = PIN_A0 // la sortie TX de l UART2 est affectee a PIN_A0
#PIN_SELECT U2RX = PIN_A1 // la sortie TX de l UART2 est affectee a PIN_A1
#use rs232(UART2,baud=57600,parity=N,bits=8) //ici nous definissons les fonctions pour l UART 2


#pin_select INT1=PIN_B3
#int_EXT1
void  EXT1_isr(void)
{
     lire_date_heure_pcf8593();
     date_presente = true; // on indique qu une interruption vient d avoir lieu
}

VOID main()
  {
  date_presente = false;
  lcd_init () ;
  Init_pcf8593();
  output_high(PIN_A0);
  enable_interrupts(INT_EXT1);
  enable_interrupts(GLOBAL);
  WHILE (TRUE)
    {
    if (date_presente)  // on effectue l action liee a l interruption
     {
      date_presente = false;  // on indique que l action a ete faite
      printf("date %02u/%02u/%04Lu\r\n",jour,mois,annee);  // afficher la date.
      printf("time %02u:%02u:%02u\r\n",heure,minute,seconde);  // afficher la date.
     }
    if (kbhit())   // on teste si une donnee est presente.
     lcd_putc(getc()); // on lit la donnee et on l'affiche.
   }
  }
```

Cette méthode est beaucoup plus élégante et efficace que les deux précédentes.

C'est terminé.