François SCHNEIDER – Lycée Victor-Hugo BESANÇON.

STS SE

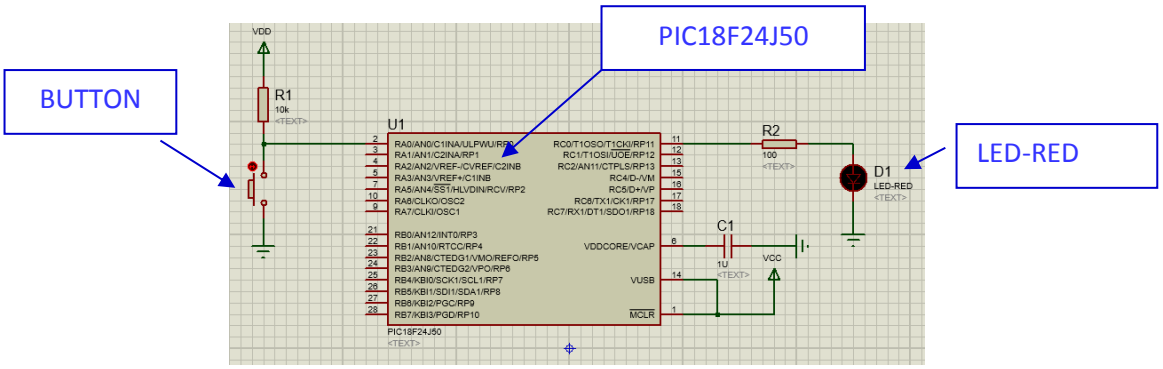
Développement de microcontrôleurs Microchip avec PICC
validation fonctionnelle PROTEUS

Premier programme
Utilisation Wizard et PROTEUS
Simulation Validation

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

I. Premier programme avec PICC, langage C, organisation d'un programme.

Le schéma du montage est donné ci-dessous avec le nom des composants en bleu (librairie ISIS). Vous dessinez le schéma, vous le sauvegardez avec pur nom « Tp1.dsn » dans un dossier nommé Tp1.



Le fonctionnement voulu est simple : un appui sur le bouton et la LED éclaire. Un quartz de 8MHz est monté en oscillateur principal.

PICC : le wizard, production du squelette du programme.

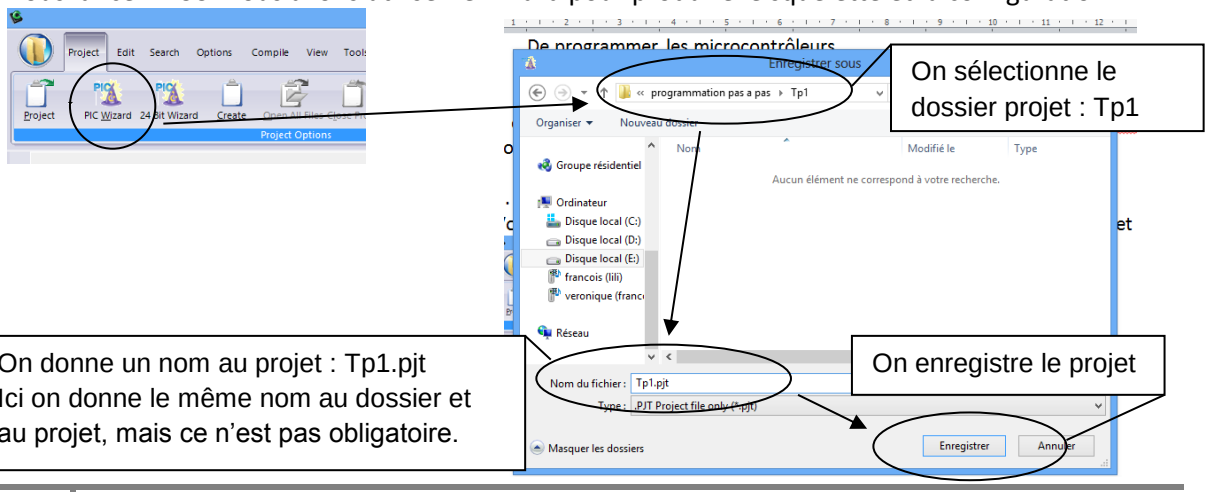
Nous allons utiliser le logiciel PICC pour produire le programme. La programmation se fait en langage C. PICC nous permet, pour les microcontrôleurs 8 et 16 bits de marque microchip :

- De produire le squelette et la configuration de base du programme.
- D'éditer le programme en langage C.
- De compiler le programme source pour obtenir le programme en langage machine. Deux versions sont produites :
  - .HEX : programme binaire simple.
  - .COF : programme binaire contenant les éléments pour la simulation ou l'émulation en pas à pas.
- De programmer les microcontrôleurs.
- De tester à l'aide d'une sonde les programmes dans la cible.

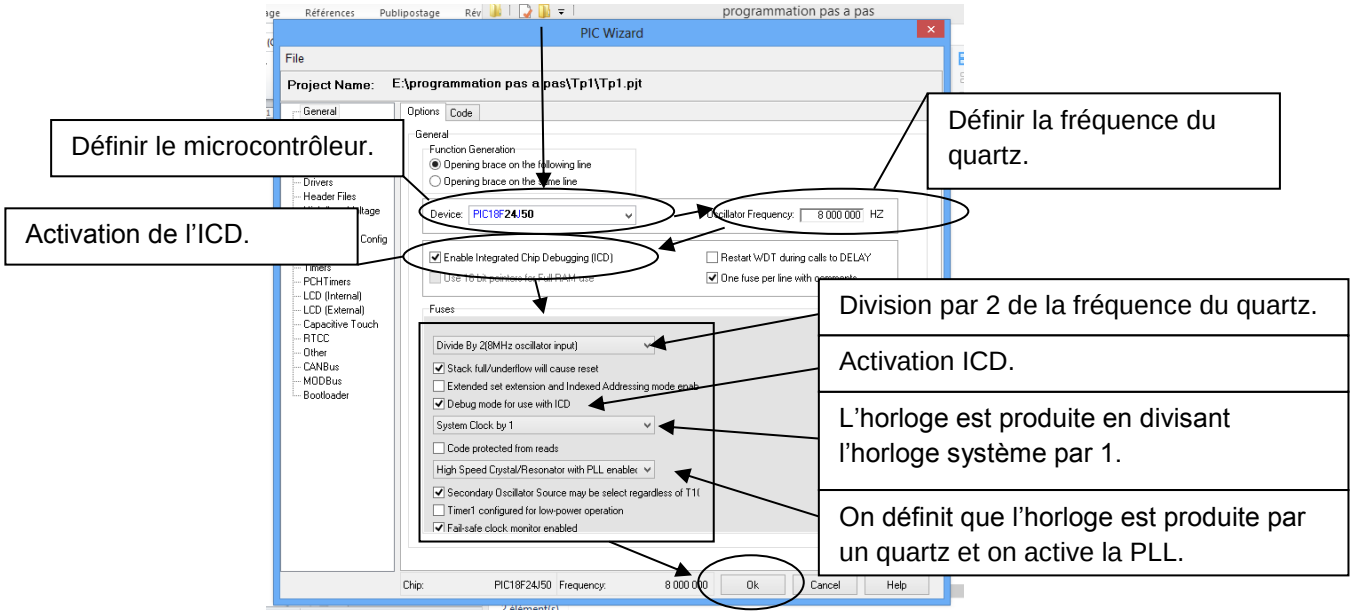
Dans ce document nous testerons les programmes par simulation avec Proteus. Vous devez avoir l'option pour la simulation des processeurs microchip, ici uniquement 8bits.

1. Création du squelette du programme en langage C avec PICC.

Vous lancez PICC. Nous allons utiliser le wizard pour produire le squelette et la configuration.

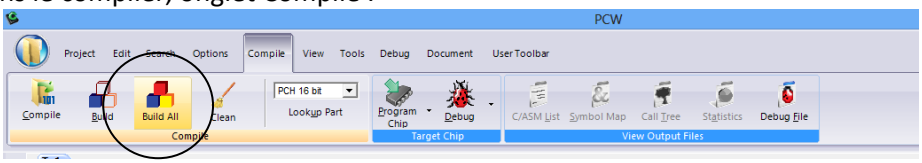


Nous obtenons la première page du Wizard, il faut alors définir les valeurs des différentes ressources utilisées. Ici seul l'ICD et l'oscillateur sont utilisés.

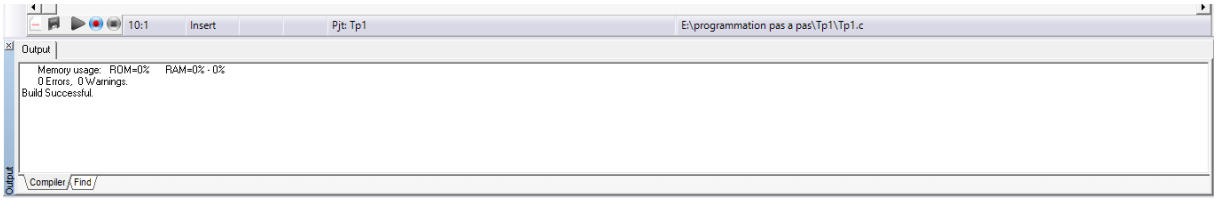


Remarque : pour avoir plus de détails sur la configuration de l'horloge, il faut se référer à la notice technique du circuit PI18F24J50.
En activant le bouton Ok, le squelette du programme est produit, deux fichiers sont créés.

Nous pouvons le compiler, onglet Compile :

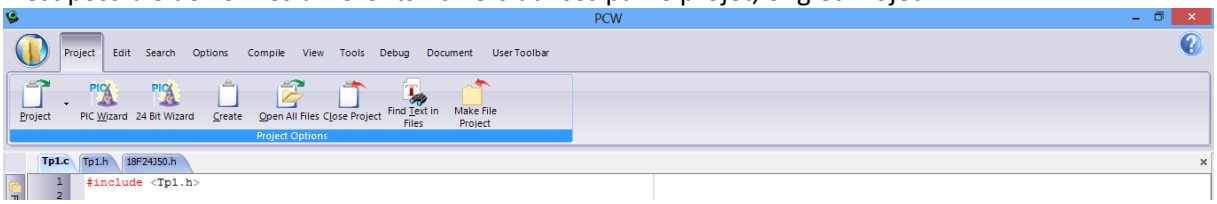


Le résultat est donné en bas de la fenêtre :



Nous voyons que la compilation s'est effectuée correctement.

Il est possible de voir les différents fichiers utilisés par le projet, onglet Projet :



Trois fichiers sont utilisés par le projet :

- Tp1.c : fichier produit par le wizard, contenant le squelette du programme.
- Tp1.h : fichier produit par le wizard contenant les déclarations.
- 18F24J50.h : fichier provenant de la librairie de PICC, contenant les déclarations particulières du composants 18F24J50. En jetant un coup d'œil, nous voyons l'affectation des ports, des registres, …

Le fichier Tp1.c

#include <Tp1.h>
 	
void main()
{
  setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
  setup_timer_4(T4_DISABLED,0,1);
  setup_comparator(NC_NC_NC_NC);// This device COMP currently not supported by the PICWizard
 	
  while(TRUE)
  {
  //TODO: User Code
  }
 	
}
	Nous trouvons ici l'organisation traditionnelle d'un programme en langage c :
	
	- Entête : avec la directive : #include <Tp1.h>
	- La fonction void main()
	- Dans cette fonction nous trouvons
	  - Au début l'initialisation de périphériques.
	  - La boucle While.

a. Entête : cette partie permet d'inclure des fichiers en principe de type .h, avec picc, il faut parfois inclure aussi des fichiers de type .c. Nous verrons par la suite le fichier Tp1.h.

   Nous pouvons dire que le langage C est un langage modulaire :

   - Il est très simple d'utiliser des librairies fournies avec le logiciel (par exemple LCD, USB, BUSCAN, …).
   - Il est possible de développer un projet à plusieurs, chacun est chargé d'une partie indépendante.
   - Il est possible de développer ses propres librairies et de les réutiliser par la suite, nous verrons ceci à la fin.

b. La fonction void main()

   Le langage C permet d'écrire un programme sous forme de fonction.

   - Une fonction peut recevoir des paramètres en entrées placés entre les ().
   - Chaque fonction peut renvoyer des valeurs. Lorsqu'elle n'en renvoie pas il est sage de placer void dans sa déclaration.

   La fonction main() est particulière, c'est la première fonction appelée lorsque le programme est lancé. Elle est donc la première fonction appelée après un reset.

   La boucle while (true) permet de créer une boucle perpétuelle. Le programme est à placer ici. Dans notre cas nous devons faire un programme, qui copie l'état du bouton sur la LED, il faut placer ce programme ici.

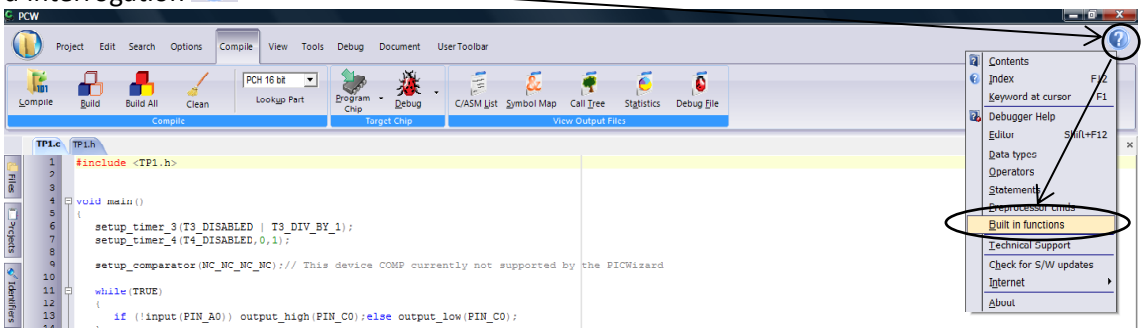
Le fichier Tp1.h

#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)	#include <18F24J50.h> inclut le fichier de configuration du processeur 18F24J50. C'est pour cette raison, qu'il se trouve associé au projet. Les directives #FUSE permettent de définir l'état des différents fusibles du processeur. En ayant un peu plus de connaissance, il est possible de modifier ces valeurs sans passer par le wizard.

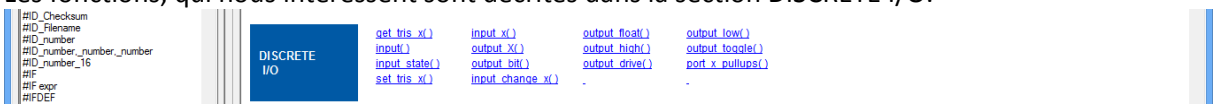
2. Ecriture du programme en langage C.

Il suffit de compléter la fonction main() au niveau de la boucle while() pour obtenir le fonctionnement voulu.

Nous allons utiliser la librairie de gestion de bits de PICC. Nous pouvons voir la description de ses fonctions en utilisant l'aide de PICC sur les fonctions de base. Pour cela cliquer sur le pont d'interrogation 



Les fonctions, qui nous intéressent sont décrites dans la section DISCRETE I/O.



Ce sont input(), output_low() et output_high(). Quelques questions :

Pour chacune des fonctions, donnez le rôle et indiquez les paramètres passés en entrée et les paramètres renvoyés en sortie :

- Input() :

- output_low() :

- output_high() :

La fonction void main() devient :

```
#include <Tp1.h>
void main()
{
  setup_timer_3(T3_DISABLED | T3_DIV_BY_1);
  setup_timer_4(T4_DISABLED,0,1);
  setup_comparator(NC_NC_NC_NC);// This device COMP currently not supported by the PICWizard
  while(TRUE)
  {
   If (!input (PIN_A0))       output_high (PIN_C0) ; else       output_low (PIN_C0) ;
  }
}
```

Commentaires :

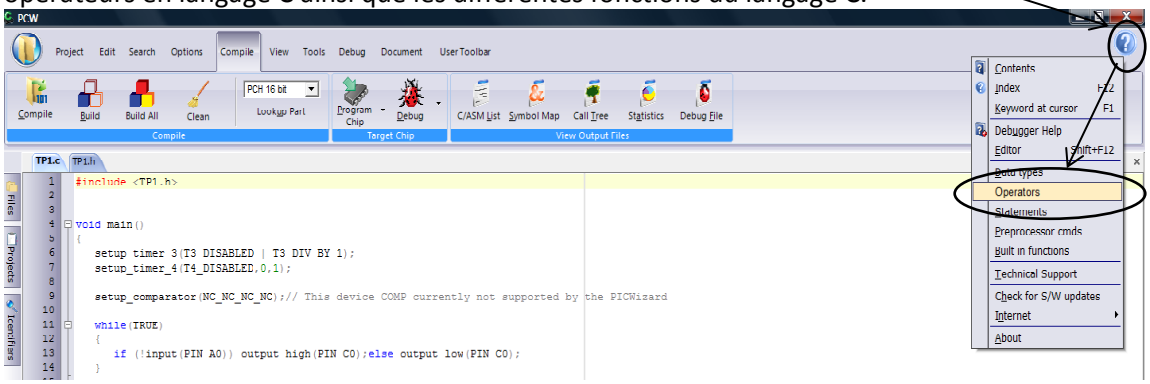
Nous utilisons la structure de contrôle if (test) then … ; else … ;

If vérifie si test est vrai, si c'est le cas alors il réalise l'instruction après *Then*, si ce n'est pas le cas il réalise l'instruction après *else*.

La fonction input() renvoie 0 ou 1 (en langage C, 0 = faux et 1 = vrai), or il faut en sortie un état haut lorsque le bouton est appuyé pour que la LED éclaire . Le ! correspond à l'opérateur inversion. Nous avons donc :

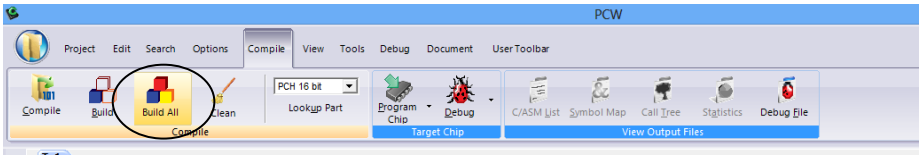
- Lorsque le bouton est appuyé, input(PIN_A0) vaut 0, l'opérateur ! inverse donc le test prend la valeur 1. Le test est donc vrai.

PICC vous fournit différentes aides à partir du point d'interrogation , notamment les différents opérateurs en langage C ainsi que les différentes fonctions du langage C.

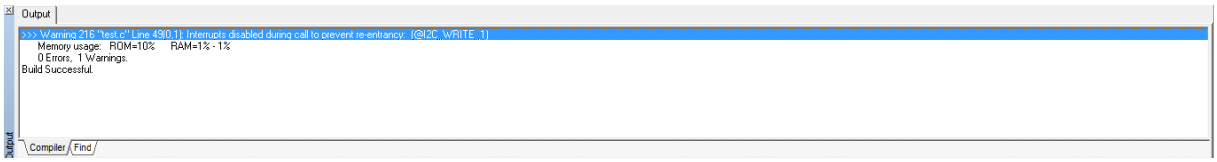


Il ne reste plus qu'à tester ce programme pour valider son fonctionnement. Je propose de le faire avec Proteus.

Dans un premier temps vous compilez le programme depuis l'onglet 'Compile' en cliquant sur le bouton 'Build'.

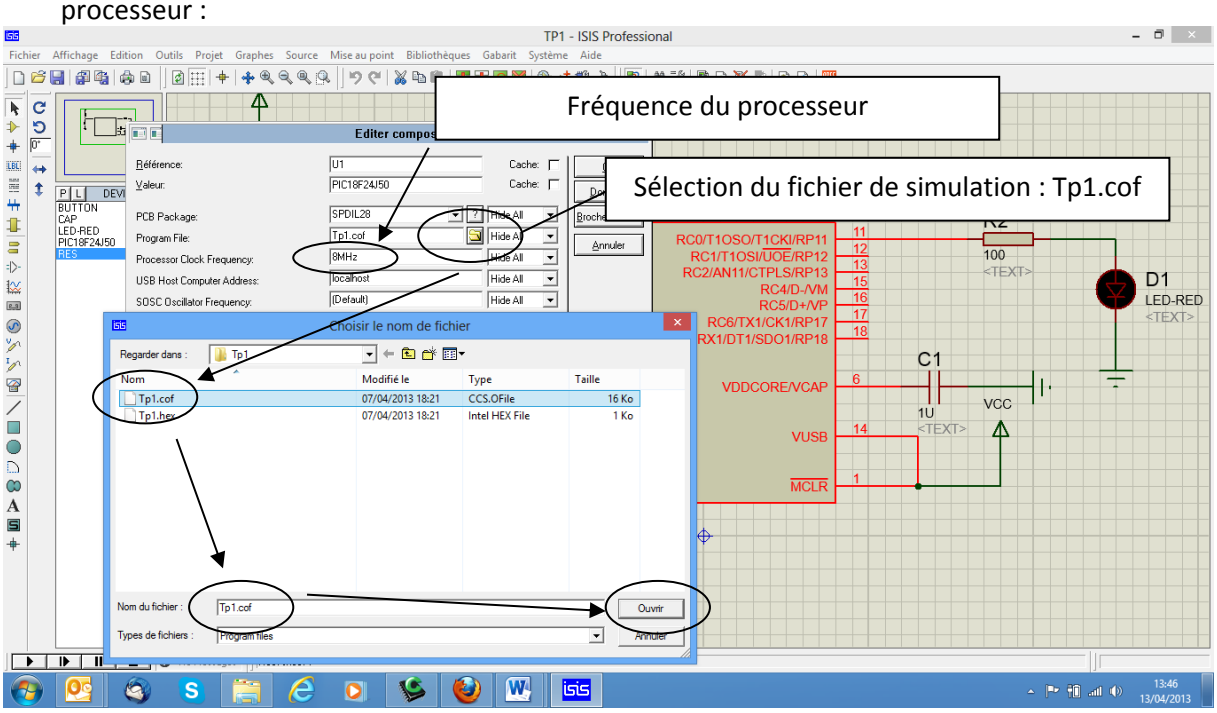


Voici le résultat de la compilation.

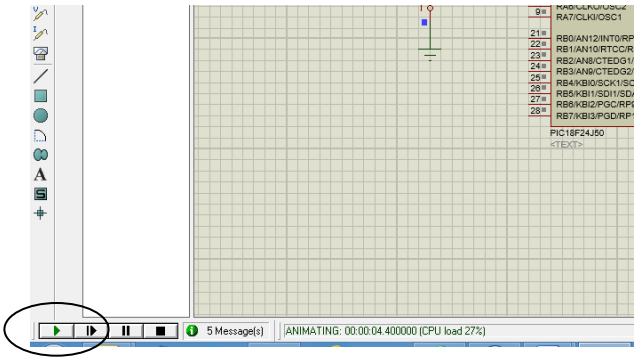


Le résultat est positif, vous pouvez continuer. Vous pouvez passer à Proteus :

- Vous définissez la configuration du microcontrôleur, pour cela vous double cliquer sur le processeur :



- Vous lancez la simulation et testez le montage en appuyant sur le bouton.

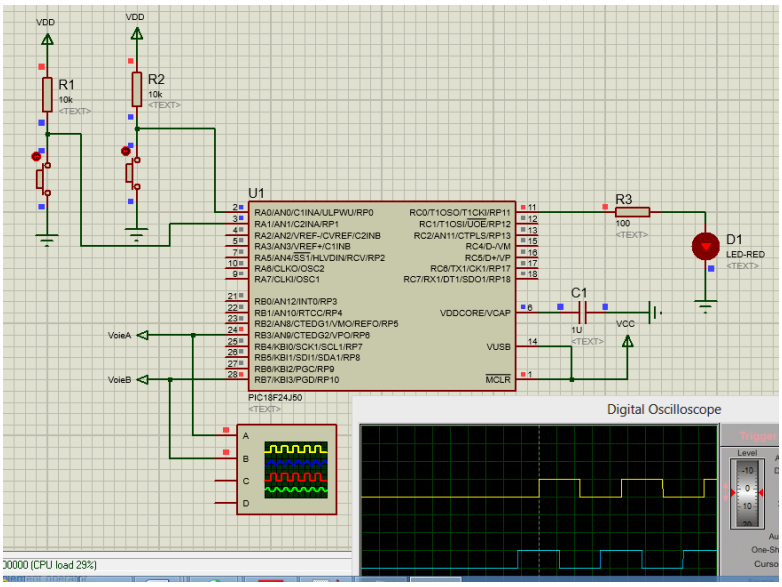


Remarque : Vous pouvez faire fonctionner le programme en pas à pas ou ajouter des points d'arrêt, soit en démarrant en mode pause bouton ⏯ ou en cliquant sur le bouton ⏸ alors que la simulation est déjà démarrée.

Vous pouvez à tout moment stopper le programme avec la touche ⏹.
Lorsque vous modifiez le programme avec PICC, il suffit de stopper la simulation et de la relancer pour que le nouveau programme soit pris en compte.

II. Programme de production des signaux de commande de phase d'un moteur pas à pas.

Le schéma du montage est donné ci-dessous avec le nom des composants en bleu (librairie ISIS). Vous dessinez le schéma, vous le sauvegardez avec pur nom « Tp1_II.dsn » dans un dossier nommé Tp1.



Les signaux VoieA et VoieB sont les signaux, qui seront appliqués au circuit de puissance d'alimentation d'un moteur pas à pas.
Le principe de fonctionnement est le suivant :

- Un appui sur le bouton câblé sur l'entrée A0, alors le moteur tourne de 10 pas en sens horaire.
- Un appui sur le bouton câblé sur l'entrée A1, alors le moteur tourne de 10 pas en sens anti-horaire.

Les signaux VoieA, VoieB vont prendre 4 états dans l'ordre :

Numéro	VoieA	VoieB
0	0	0
1	0	1
2	1	1
3	1	0

Il faut créer une variable *pas_encours* pour compter les pas. Cette variable est de type global car connue par tous les programmes.

Afin de produire du programme facilement réutilisable, nous allons créer dans le dossier projet un fichier librairie nommé « moteur.c ».

Dans ce fichier nous allons écrire 3 fonctions, qui sont :

- *void init_pas(void)*.
- *void rotation_un_pas(unsigned short sens)* : permet de tourner d'un pas dans le sens indiqué par sens.
- *void rotation_n_pas(unsigned int nombre_pas,unsigned long int duree_pas_us,unsigned short sens)* : permet d'avancer de nombre_pas, suivant sens et avec duree_pas_us.

Le fichier moteur .c est le suivant :

#define voieA PIN_B3
#define voieB PIN_B7
 	
volatile unsigned int pas_encours;
const unsigned short horaire=true;
const unsigned short antihoraire=false;
void init_pas(void)
{
pas_encours = 0;
output_low(voieA);
output_low(voieB);
}
void rotation_un_pas(unsigned short sens)
{
switch (pas_encours) {
case 0:
if (sens==horaire) {pas_encours = 1; output_high(voieB);}
else {pas_encours = 3; output_high(voieA);}
break;
case 1: ;
if (sens==horaire) {pas_encours = 2; output_high(voieA);}
else {pas_encours = 0; output_low(voieB);}
break;
case 2: ;
if (sens==horaire) {pas_encours = 3; output_low(voieB);}
else {pas_encours = 1; output_low(voieA);}
break;
case 3: ;
if (sens==horaire) {pas_encours = 0; output_low(voieA);}
else {pas_encours = 2; output_high(voieB);}
break;
default: init_pas();
break; }
}
void rotation_n_pas(unsigned int nombre_pas,unsigned long int duree_pas_us,unsigned short sens)
{
unsigned int i;
for (i=0;i<nombre_pas;i++)
{
rotation_un_pas(sens);
delay_us(duree_pas_us);
}
}
	Ces 2 lignes permettent d'affecter les broches du microcontrôleur.

Nous utilisons la structure de contrôle *switch* à voir dans l'aide.

Le programme principal Tp1_II.c devient :

```
#include <Tp1_II.h>
#include "moteur.c"


void main()
{
  SHORT mem_PIN_A0, mem_PIN_A1;
  SHORT old_PIN_A0, old_PIN_A1;
  setup_timer_3 (T3_DISABLED|T3_DIV_BY_1) ;
  setup_timer_4 (T4_DISABLED, 0, 1) ;
  setup_comparator (NC_NC_NC_NC); // This device COMP currently not supported by the
PICWizard
  init_pas();
  mem_pin_A0 = input (PIN_A0);
  mem_pin_A1 = input (PIN_A1);

 WHILE (TRUE)
  {
    old_pin_A0 =  mem_pin_A0;
    old_pin_A1 =  mem_pin_A1;
    mem_pin_A0 = input (PIN_A0);
    mem_pin_A1 = input (PIN_A1);
    IF ((mem_pin_A0==0) && (old_pin_A0== 1))
    {
      output_high (PIN_C0) ;
      rotation_n_pas(10,100,horaire); // on fait tourner le moteur de 10 pas en sens horaire, 100us
                                                  par pas
    }
    IF ((mem_pin_A1==0) && (old_pin_A1== 1))
    {
      rotation_n_pas(10,100,antihoraire); // on fait tourner le moteur de 10 pas en sens horaire,
                                                  100us par pas
      output_low (PIN_C0);
    }
  }
}
```

Les variables old_pin_A0, old_pin_A1, mem_pin_A0 et mem_pin_A1 sont ajoutées pour détecter un appui sur un bouton.
Vous n'avez plus qu'à tester.