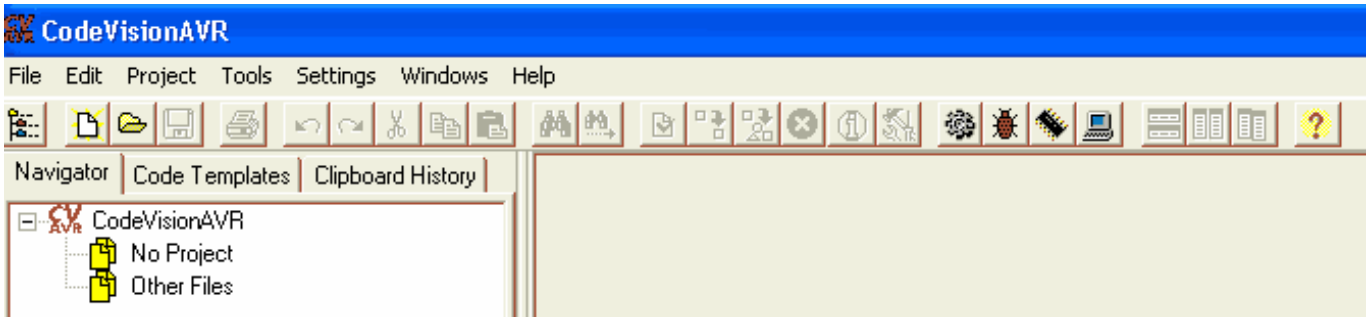
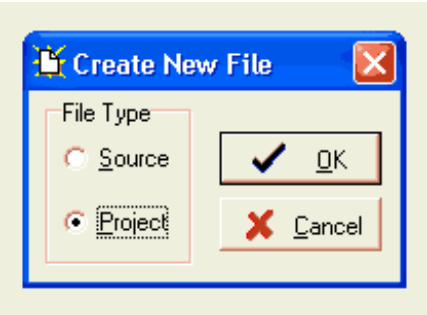
**Mise en oeuvre de CodeVisionAVR**

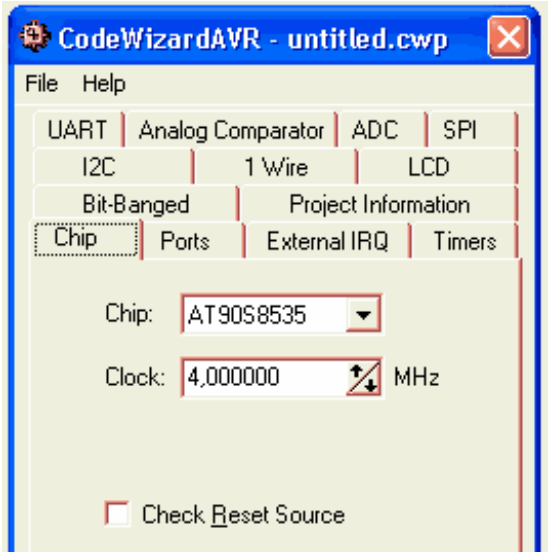
1) Lancer CodeVision AVR




2) Faire « File new », sélectionner « Project » puis « OK »




3) Répondre « oui » au CodeWizard (celui-ci permet d'initialiser les ressources du microcontrôleur et de générer la partie de programme C correspondant).



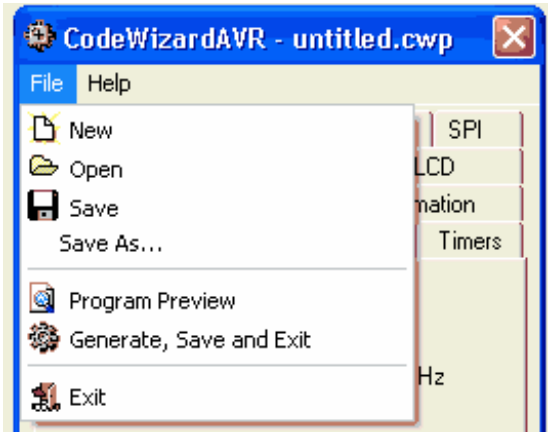

4) Choisir le composant utilisé, sa fréquence d'horloge, initialiser les ports, timers utilisés ...

5) Enregistrer : File => generate => save and exit. L'outil de développement propose alors la création de trois fichiers de base :

   - le programme principal avec l'extension **.c** (nom_projet.c)

- le fichier projet avec l'extension **.prj** (nom_projet.prj)
- le fichier CodeWizard project avec l'extension **.cwp** (nom_projet.cwp)

**!!! Ne pas dépasser huit caractères dans les noms de fichiers**




A cette étape CodeVision a généré le projet avec le programme C incluant les initialisations des périphériques.

6) Ecrire le reste du programme.

7) Vérifier la syntaxe : **Project => Check Syntax**.

8) Compiler : **Project => Compile** (crée le fichier assembleur en faisant appel à AVRC Compiler).

9) Assembler : **Project => Make** (crée le fichier assembleur **et** le fichier éxécutable en faisant appel à AVRC Compiler + AVRASM 32).

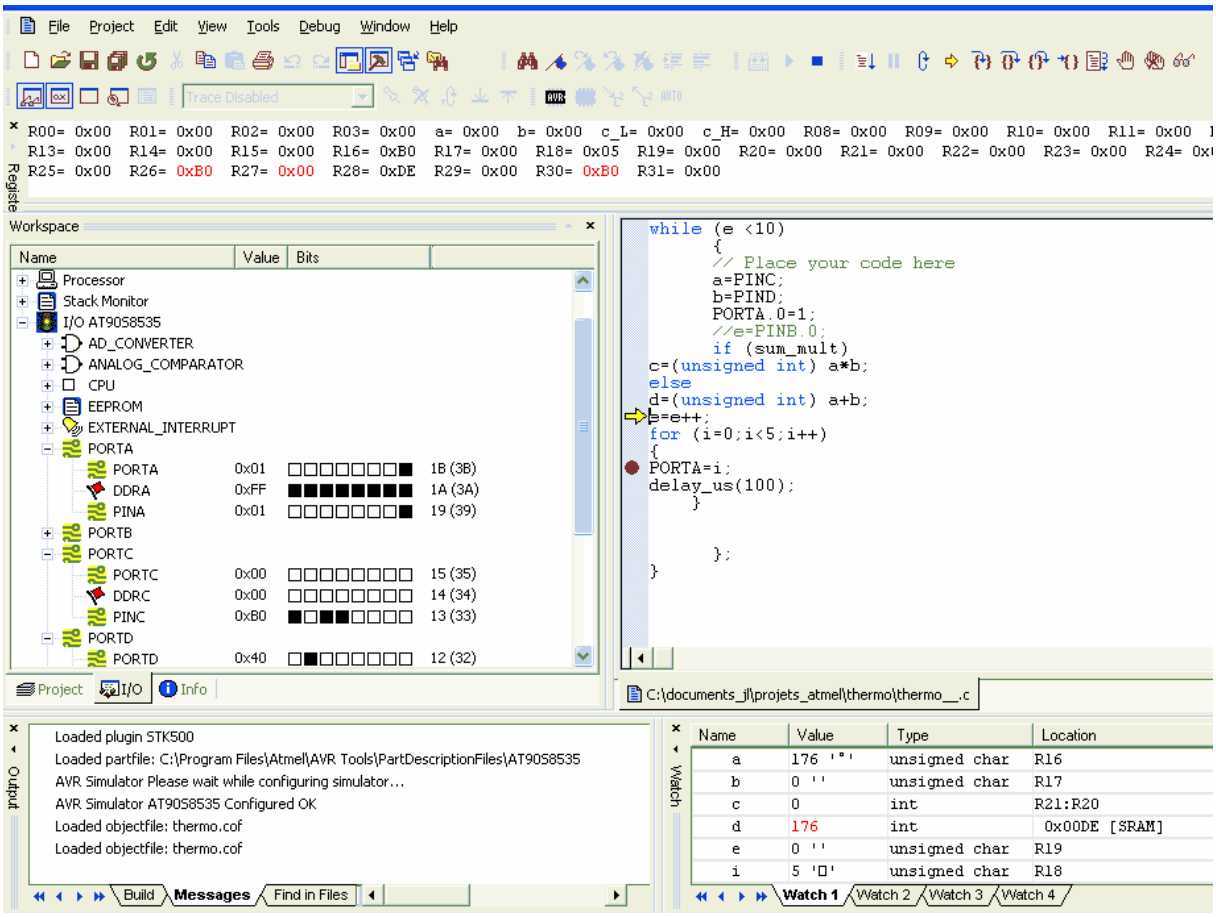
   Remarque : on peut passer directement de l'étape 7 à l'étape 9.

Une fois l'étape 9 terminée et si pas d'erreur un fichier **nom_projet.cof** a été généré. C'est ce fichier qui sera ouvert dans AvrStudio pour le deboggage.

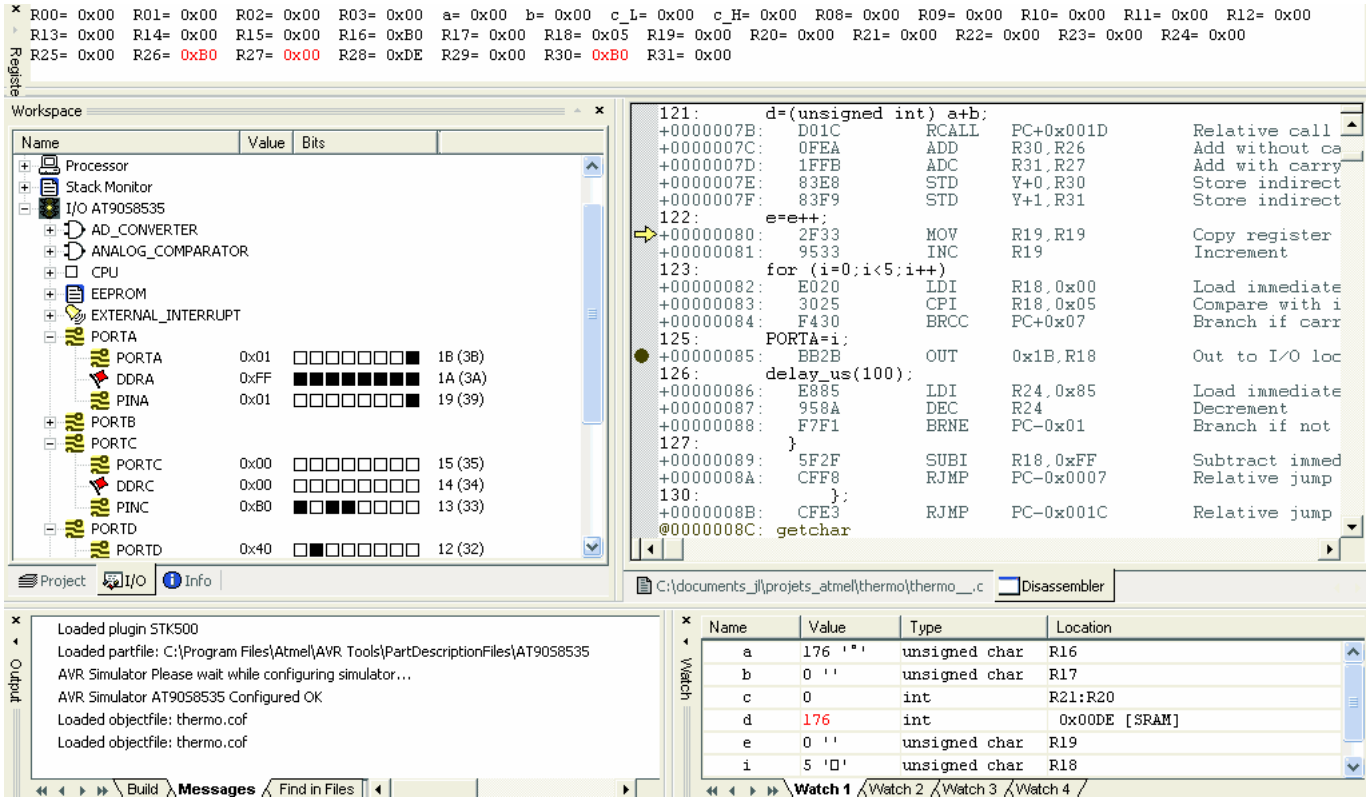
10) Lancer AvrStudio à partir de l'environnement codeVision (icône coccinnelle). Dans AvrStudio choisir l'option « open » de la boîte de dialogue. Sélectionner le fichier **nom_projet.cof** puis l'ouvrir. Le programme en C apparaît dans une fenêtre ainsi que les ressources du microcontrôleur dans une autre fenêtre. (Si AvrStudio n'apparaît pas, indiquer son chemin d'accès dans CodeVision : **settings => debugger** )

11) Deboggage :

Le déboggage s'effectue à partir des mêmes outils de AvrStudio comme pour un programme assembleur. Les instructions peuvent être exécutées en pas à pas au niveau du C. Si l'on veut visualiser l'exécution des instructions assembleur, cliquer sur la fenêtre « désassembleur ». La visualisation des variables peut se faire dans la fenêtre « Watch » après avoir sélectionné la variable et cliqué sur l'icône « lunette ».




Et avec la fenêtre de désassemblage :



Remarque : si on modifie le programme dans CodeVision et qu'on le recompile, AvrStudio propose de recharger le programme.

12) Quelques instructions utiles :

- lecture d'un port (PORTA par exemple)

char a ; // déclaration type de la variable dans la zone de déclaration des variables
**a= PINA ;** (respecter la casse)

- lecture d'un bit d'un port (bit 0 du PORTA)

bit a ; // déclaration type de la variable
**a=PINA.0 ;**

- écriture d'un port (PORTA par ex.)

**PORTA=0xFC ;**

- écriture à 1 du bit d'un port (bit 0 du PORTA)

**PORTA.0=1 ;**

13) La condition « IF .. else »

```
Bit e ; //déclaration variable e de type bit
Char a, b ;              // a, b caracteres sur 8 bits
Int c, d;                // c, d entiers sur 16 bits
If (e)                   //si e=1 on exécute c=
c=(unsigned int) a*b ;
Else                     // sinon on exécute d=
d=(unsigned int) a+b;
```

Remarque 1: on peut mettre directement un test sur un bit d'entrée de port
Exemple :

```
If (PINB.0)
C=
```

Remarque 2: on peut remplacer le bit du port par un nom plus explicite.
Exemple :

```
#define sum_mul PINB.0
if (sum_mul)
....
```

autre exemple:

```
if (a <b)
c= .....;
else
d= ..... ;
```

14) Structure "While"

- While (1){liste d'instructions}; //exécute tout le temps les instructions de la liste.
- While (b){liste d'instructions}; // si b de type bit, exécute tout le temps les instructions de la liste tant que b=1.
- While (c<10){liste d'instructions}; //exécute les instructions de la liste tant que c<10.

15) Boucle « For »

```
For (i=0 ; i<10 ; i++)
{liste d'instructions};  // la liste d'instructions est exécutée 10 fois.
```

16) Fonction « Switch ..case »

```
Remplace plusieurs "if ...else »
switch (var)
{
case « 1 » : b=10 ;
case « 2 » : b=5 ;
.....
}
```

17) Divers

- **localisation de variables** (imposer l'adresse physique aux variables)

exemple1 : unsigned char a@0x60;  // la variable a d'un octet sera stockée en RAM à l'adresse $60.
exemple2 : register unsigned char b@10;  // la variable b d'un octet sera stockée dans le registre R10.

- **Introduction d'instructions en assembleur :**

#asm ("sei\nop\.....\cli")
Les instructions sont insérées dans l'ordre d'apparition dans le programme en C.

- **Appel d'un sous-programme assembleur à partir du C :**

On définit une fonction :

```
int sum_ab (int a, int b)   // retourne la somme des deux entiers a et b
#asm                        // début du sp assembleur
ldi temp, $7F
inc temp1
...
#endasm
```

Appel de la fonction dans le programme principal :

```
void main (void)
{
int res ;
res= sum_ab (8, 12);
}
```

18) Création de librairies

a) Création du **header** (en-tête)
Dans **CodeVision => File=>New=>Source**, taper les différents protocoles de librairies.
Exemple :

```
#pragma used +     (évite les warnings si librairie non utilisée dans le projet)
Int sum (int a, int b) ;
```

```
Int mul (int a, int b) ;
…
#pragma used –
```

Faire « enregistrer sous » et choisir le répertoire « inc » avec l'extension **.h** (exemple : perso.h)

b) Création du fichier **.c**

```
Int sum (int a, int b)
{
return a+b ;
}
Int mul (int a, int b)
{
return a*b ;
}
```

Faire « enregistrer sous » dans un répertoire en perso.c puis faire « File=>Convert to Library » avec l'extension **.Lib**.
Dans le programme principal faire :
# include <perso.h>