LPSARI
Microcontrôleurs TP1

1. Mise en place des outils

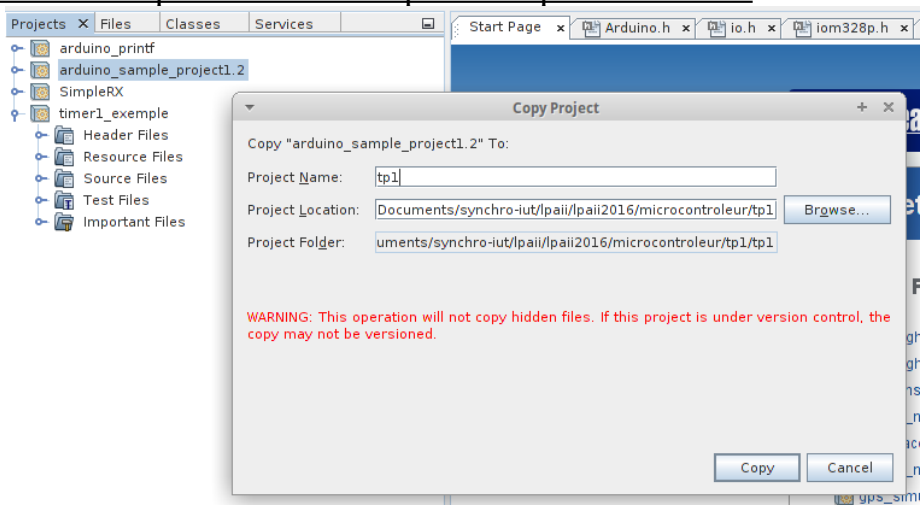
Nous allons travailler avec des microcontrôleurs *Atmel ATMega368P* sur plate-forme arduino.
La chaîne de développement est celle fournie par l'IDE arduino. Elle est basée sur la chaîne libre avr-libc : http://www.nongnu.org/avr-libc/
L'IDE à utiliser est NeatBeans à configurer pour arduino :
http://arlotto.univ-tln.fr/arduino/article/arduino-netbeans

Les projets donnés en exemple sont des projets NeatBeans (linux). Pour les utiliser, il faudra peut-être changer la version de l'IDE arduino ainsi que le nom de l'utilisateur dans le makefile. Pour télécharger le code exécutable, il faudra renseigner correctement le chemin du port série dans le makefile.

- Installez les outils nécessaires (arduino, NeatBeans,…) et faites fonctionner l'exemple
"arduino sample project" (http://arlotto.univ-tln.fr/ressources/arduino/)

- Quelle est la taille mémoire (Flash et RAM) occupée par ce programme ?
A quoi correspondent les sections .text , .data , .bootloader , .data, .bss, .noinit ?
Comment le programme est il transféré dans le micro ? Quelle est le nom du fichier transféré ? Quel format ?

Par la suite on repartira toujours du projet exemple que l'on copiera à l'aide de la fonction de copie de projet de NeatBeans. Veillez toujours à ce que le nom du projet soit identique au nom du répertoire qui le contient.



Pour la suite on pourra utiliser un shield comprenant 4 leds, 2 boutons et un potentiomètre.

2. Machine d'états

En utilisant des machines d'états, écrire un programme unique qui :
- allume une led lorsqu'on appuie sur une première fois sur un bouton et l'éteint lorsqu'on appuie de nouveau sur ce même bouton
- allume une autre led après 3 appuis sur le deuxième bouton et l'éteint après le quatrième appui.
- simule une régulation de température tout ou rien à hystérésis :
  la mesure est simulée par le potentiomètre : min -10°C maxi +50°C.
  la commande de chauffage par une led.
  l'appuie simultané sur un bouton à la mise sous tension permet de régler la valeur de consigne entre 10 et 30°C avec le potentiomètre.
- affiche (sur le port série) à chaque changement : température,consigne, état du chauffage.
- stocke la valeur de la consigne en EEPROM. Prévoir un retour à la valeur usine (21°C) par appui simultané sur les deux boutons à la mise sous tension.

En utilisant une sortie logique et l'oscilloscope, mesurez le temps de boucle de votre programme.
En utilisant le mode "single" de l'oscilloscope, relevez la forme de la tension lors de l'appui sur un bouton (rebonds).
En comparant le temps de boucle et le temps de stabilisation de l'état du bouton, expliquer pourquoi les rebonds ne sont en général pas un problème dès que le programme est assez conséquent.

3. Générer une interruption périodique

Faites fonctionner le projet "timer1interrupt".
Expliquer le fonctionnement à l'aide des documentations du microcontrôleur et de avr-lib.
Expliquez la macro _BV(). Comment mettre des bits à 1 ? à 0 ?
Quelle est la période de clignotement de la LED ?
Modifier le programme pour avoir une période interruption d'une seconde exactement.
Faire clignoter en plus une autre LED toutes les 12 secondes.
Modifiez le programme pour avoir une période interruption de 10ms.
Vérifiez le fonctionnement à l'oscilloscope.

4. Création d'une bibliothèque de temporisateurs

On se propose de créer une classe C++ permet de gérer efficacement des temporisations basées sur une interruption périodique.

Un temporisateur est une machine d'états qui comprend au moins les états suivants : START, RUNNING, DONE/IDLE et un compteur.

Au départ le temporisateur est dans l'état IDLE. Pour le démarrer on le place dans l'état START et on charge le compteur avec la durée souhaitée.
Périodiquement (dans une interruption) on décrémente alors le compteur (état RUNNING) et lorsqu'il devient égal à 0 l'état passe à DONE.
Des fonctions permettent de connaître l'état de la tempo, de l'arrêter, de la prolonger,….
Chaque temporisateur est un objet de cette classe. On peut ainsi disposer de multiples temporisations indépendantes basées sur le même timer physique.
La période de interruption définie la résolution des temporisateurs. La taille du compteur fixe la durée maximale.

La classe est ainsi définie :

```
class TimerDelay {
public:
    TimerDelay();
    TimerDelay(uint16_t delay);
    TimerDelay(uint16_t delay , uint16_t period);
    void setPeriod(uint16_t period );
    void start();
    void start(uint16_t delay);
    void reStart();
    void reStart(uint16_t delay);
    void halt();
    bool isRunning();
    bool isDone();
    bool isHalted();
    void periodicCall(); // to be called periodicaly
    void debug();
private:
    uint16_t delay;
    uint16_t counter ;
    uint8_t state ;
    uint16_t period ;

};
```

Les constructeurs permettent de créer des temporisateurs soit avec une durée par défaut (10 secondes par exemple) soit en spécifiant les durées.
La fonction `periodicCall()` doit être appelée périodiquement dans l'interruption. Elle gère l'évolution de l'état de la temporisation.
Les autres méthodes permettent d'agir ou de connaître l'état de la temporisation.

- Donner la relation entre la période de l'interruption, la durée maximale possible et la taille du compteur. Que vaut l'incertitude sur la durée de temporisation

effectivement réalisée (jitter) ? Montrez qu'il existe un compromis durée maximale / incertitude.
- Complétez et commenter le projet "timer_tempo_edt" fourni.
- Changez la période de l'interruption et mettre en évidence le jitter.
- Ajoutez la possibilité de rendre une temporisation "auto réarmable".

5. DataLogger

On souhaite créer une application d'acquisition et d'horodatage d'une température (ici simulée par un potentiomètre par exemple [0-5V] ↔ [-40°C ; +40°C]). On travaillera au dixième de degré. La température est mesurée toutes les 5 secondes. Les données seront stockées sur une carte SD.
Les paramètres de l'acquisition seront modifiables par la lecture d'un fichier sur la carte SD. Par défaut l'intervalle temporel est de une minute.
Le programme surveille également que la température reste comprise entre deux seuils (SH et SB).
Lorsque la température dépasse le seuil haut SH pendant plus de DH un défaut "haut" est enregistré. De manière symétrique, on a un défaut si on est en dessous de SB pendant plus d'une durée DB.
Le format de stockage est le suivant :
T;date;time;temp;défaut\n
Les défauts possible sont : ok , high, low.
Exemple : T;2015-04-12;15:48:42;25.1;ok;
Prévoir un intervalle variable entre 10s et 15minutes. -30°C< SH<30°C, -30°C< SB<30°C, 1s<DB<15minutes , 1s<DH<15minutes

Pour cet exercice on utilisera un shield avec une carte SD et une horloge temps réel. Les librairies (SD et RTC) seront intégrées à votre projet.