INF1900

Projet initial de système embarqué
Travail pratique No. 5
Mémoires et protocoles de communication

Objectif: Comprendre l'accès à la mémoire externe de la carte mère et la communication par protocole RS232
Durée: Une semaine
Travail préparatoire: Aucun
Documents à remettre: Aucun, mais le travail pour ce laboratoire sera repris pour le travail pratique 9.
Présentation en classe: fichier LibreOffice Impress

> "Finalement tout se connecte – Les gens, les idées, les objets. La qualité des connexions est la clé de la qualité en soi."
>
> "Les détails ne sont pas les détails. Ils font la conception"
>
> - *Charles Eames*

Introduction

La semaine dernière, nous avons regardé l'ensemble des différents types de mémoire à l'intérieur même de l'AVR. En particulier, nous nous sommes attardés à l'utilisation de certains registres en mémoire pour accéder aux ressources internes disponibles. Les mémoires internes sont de plusieurs types et contrôlent à peu près tout le fonctionnement du microcontrôleur. Par contre, leur accès est grandement facilité par l'utilisation de mnémoniques dans le code et du fait que le manufacturier peut s'organiser pour faciliter l'adressage à l'intérieur même du circuit. Il en va autrement lorsque la mémoire se situe à l'extérieur du circuit. Il faut alors bien définir les signaux de contrôle et de données entre le microcontrôleur et la mémoire externe. De plus, les délais doivent être pris en considération pour synchroniser correctement les opérations entre les deux puces.

Il est important de constater que la majorité des petits systèmes embarqués n'ont pas toujours de disque dur ou d'accès à un réseau pour échanger ou stocker des données. Les mémoires deviennent donc des éléments importants, surtout pour l'ingénieur informaticien qui aborde le design de tels systèmes par les questions logicielles plutôt que par les aspects électriques ou mécaniques. De plus, l'établissement de ce qui est adressable, et de quelle

façon, est souvent au coeur des premières ébauches avec le reste de l'équipe de développement (voir le livre de Michael Barr, *Programming Embedded Systems in C and C++*, en référence).

Comme on ne fait ici qu'un survol, et que de toute façon la technologie des mémoires évolue très rapidement, on ne peut se permettre de couvrir tous les aspects de ce vaste sujet. Il est tout de même nécessaire de revenir sur certains concepts de base.

Traditionnellement, les mémoires ont été divisées en deux grandes catégories, les mémoires vives et les mémoires mortes. Cette division devient de plus en plus floue puisque les méthodes d'écriture et la persistance des données sont plus nombreuses et variées qu'à une époque encore pas si lointaine.

Les mémoires vives sont celles qui perdent leur contenu si le circuit n'est pas alimenté. On distingue les mémoires vives statiques et dynamiques. Les mémoires dynamiques nécessitent un circuit complexe qui rafraîchit leur contenu de façon périodique puisqu'elles tendent à se décharger rapidement. Par contre, elles peuvent stocker des quantités impressionnantes de données. Les mémoires vives statiques ne nécessitent pas de circuit de rafraîchissement. Elles sont par contre chères. Le fait qu'elles puissent être accédées très rapidement en fait de bonnes candidates pour former de la mémoire cache.

Les mémoires mortes ont subi une plus grande évolution au cours des dernières années. De la traditionnelle mémoire ROM que l'on «brûle» une seule fois, on a commencé par permettre la réécriture dans ce type de mémoire selon diverses technologies. La première évolution a vu l'utilisation d'ultraviolets pour effacer le contenu avant de permettre la réécriture par tension élevée (12 volts et parfois même plus). Puis, on a permis l'effacement et la réécriture de façon entièrement électrique (*Electrically Erasable Programmable Read-Only Memory* - EEPROM). C'est ce type de mémoire qui sera utilisé pour ce laboratoire. Une technologie assez différente est arrivée ces dernières années avec les mémoires de type *flash*. D'un point de vue de sa conception, ce type de mémoire fait partie des EEPROM. Les différences viennent du fait que la mémoire flash est plus rapide d'accès alors que les EEPROM peuvent subir un plus grand nombre de réécritures. L'accessibilité à plusieurs zones de mémoire a aussi grandement favorisé l'utilisation des mémoires flash.

Une autre façon de classer les mémoires est par les méthodes d'accès, soit série, soit parallèle. Accéder à une mémoire par un protocole série demande moins de broches et prend donc moins de place sur une carte. Le protocole d'accès peut cependant être plus complexe étant donné qu'il faut souvent envoyer l'adresse en premier, suivie de la donnée, et ce, bit par bit. Un protocole parallèle est plus simple puisqu'on peut compter sur des bus d'adresse et de donnée ce qui implique moins d'étapes lors de l'accès. Par contre, il peut y avoir plus de traces de métal sur la carte mère. La mémoire externe sur notre circuit n'a que 8 broches et doit donc être accédé avec un protocole série. Le protocole utilisé ici s'appelle I²C (ou *2-Wire*,

appelé aussi TWI par Atmel). D'autres protocoles série sont aussi répandus. Mentionnons simplement le protocole SPI (*Serial Peripheral Interface*). Ce protocole est aussi appelé 3-wire par certains manufacturiers. Il faut dire que beaucoup d'autres types de périphériques peuvent être accédés via ces protocoles, et non pas seulement des mémoires. Évidemment, d'autres protocoles sont plus connus (comme l'USB) parce qu'ils sont employés dans les PC ou des systèmes courants, mais ils sont plus complexes.

Alors, comment choisir la mémoire appropriée? La réponse variera beaucoup en fonction du système à concevoir. Les systèmes opérant en temps réel et qui demandent beaucoup de calculs pourront être très gourmands en mémoire statique dispendieuse. À l'autre extrême, il faut parfois stocker quelques octets seulement (comme le numéro de série, le numéro de révision du micrologiciel (*firmware*), un code d'accès spécial, etc..) dans des mémoires très petites. Entre ces extrêmes, toutes les situations sont possibles. Le coût est aussi un grand facteur. La petite mémoire de 512Kbits (65536 octets X 8 bits) sur votre carte mère coûte environ $2.10. Sa fiche technique peut être consultée par simple curiosité.

L'étude du protocole I²C permettant l'accès à la mémoire n'est pas un objectif direct du projet. C'est pourquoi nous vous fournissons une classe en C++ qui gère toutes les opérations mémoire. Vous pourrez regarder le code pour satisfaire votre curiosité si bon vous semble. Nous décrirons ici uniquement l'interface de programmation telle que présentée dans le fichier *memoire_24.h*.

Cette classe se nomme Memoire24CXXX et son constructeur n'a besoin d'aucun argument. Ce constructeur appelle lui-même la procédure *init* de telle sorte que vous ne devriez jamais avoir besoin de le faire puisqu'il n'y a aucune raison pour réinitialiser le bus I²C (ce ne serait pas le cas si vous aviez plus d'un périphérique I²C à contrôler). L'écriture peut se faire de deux façons. Un octet seul peut être écrit en précisant sa valeur et son adresse. Une autre possibilité est d'écrire un bloc d'octets (n'excédant pas 127) d'un seul coup en commençant à l'adresse spécifiée. La lecture se fait d'une manière similaire. On peut lire un octet à la fois ou en passant un vecteur de longueur précise pour lire un bloc continu.

```
class Memoire24CXXX
{
public:

     // le constructeur appelle init() decrit plus bas
     Memoire24CXXX();
     ~Memoire24CXXX();

     // procedure d'initialisation appelee par le constructeur
     // a appeler avant des lectures ou des ecritures
     void init();
     // la procedure init() initialize a zero le "memory bank".
```

```
    // appeler cette methode uniquement si l'adresse doit changer
    static uint8_t choisir_banc(const uint8_t banc);

    // deux variantes pour la lecture:
    // une donnee a la fois
    uint8_t lecture(const uint16_t adresse, uint8_t *donnee);
    // bloc de donnees : longueur doit etre de 127 et moins
    uint8_t lecture(const uint16_t adresse, uint8_t *donnee,
                                        const uint8_t longueur);

    // deux variantes pour l'ecriture:
    // une donnee a la fois
    uint8_t ecriture(const uint16_t adresse, const uint8_t donnee);
    // bloc de donnees : longueur doit etre de 127 et moins
    uint8_t ecriture(const uint16_t adresse, uint8_t *donnee,
                                        const uint8_t longueur);

private:
    // pour l'ecriture
    uint8_t ecrire_page(const uint16_t adresse, uint8_t *donnee,
                                    const uint8_t longueur);

private:
    // donnees membres
    static uint8_t m_adresse_peripherique;
    const uint8_t PAGE_SIZE;
};
```

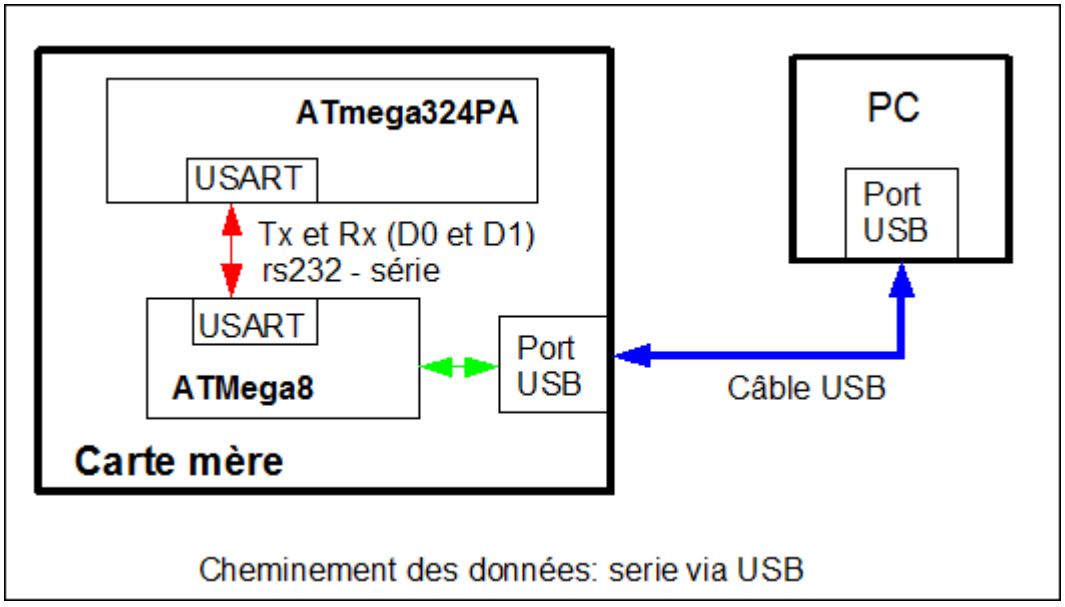
Voici les fichiers permettant l'accès à la mémoire: memoire_24.h et memoire_24.cpp. Il suffira d'ajuster votre Makefile pour les compiler avec votre code. Comme la mémoire sera utilisée, il faut qu'un cavalier soit présent sur MemEN.

Le bon vieux protocole de communication RS232 ne veut pas mourir... Les microcontrôleurs récents tendent à avoir des U(S)ART (Universal (Synchronous and) Asynchronous Reciever Transmetter) intégrés pour faciliter la communication RS232. Un U(S)ART est toujours basé sur un registre à décalage, le plus souvent de 8 bits. Comme l'information est envoyée bit par bit sur la ligne Tx du câble, le registre à décalage, tant en réception qu'en émission, devient l'élément électronique fondamental des équipements supportant le RS232.

Le RS232 est un protocole de communication fort simple comparé à des protocoles plus complexes comme l'USB. Dans notre cas, il nous servira surtout de façon sporadique pour aider au déverminage. Les exercices de ce travail pratique doivent donc être vus comme du

code que vous pourrez réutiliser lors de la mise au point de votre robot. On ne fera que recevoir quelques octets de la carte sans procédure de contrôle complexe.

Le site Wikipédia donne une bonne explication du RS232 qui est suffisante même si la version anglaise du site offre plus de détails. Dans la plupart des cas, les données qui sortent d'un USART passent par un circuit qui fait simplement ajuster la tension des signaux à -12 et +12 volts. Un connecteur de type DB9 ou DB25 permet également de faire cheminer les signaux dans un câble contenant quelques fils. Sur la carte mère, pour éviter l'ajout de ce circuit qui ajuste la tension et d'un connecteur DB9, on redirige les données via le port USB. De cette façon, on réduit la taille de la carte et toute la communication se fait par un seul câble comme le montre la figure. Le ATMega8 effectue la conversion de protocoles entre le RS232 et l'USB. Le code évoluant sur le ATmega324PA n'a pas à se soucier de cette redirection par l'USB. En effet, le même code pourrait servir dans un système avec communication directe vers un PC par câble RS232.



Cheminement des données: serie via USB

Problèmes

Problème 1: lecture et écriture en mémoire

Vous devrez faire un programme qui écrit la chaîne de caractères « ***P*O*L*Y*T*E*C*H*N*I*Q*U*E* *M*O*N*T*R*E*A*L*** » suivi d'un 0x00 en mémoire externe. La chaîne commencera à l'adresse 0x00. Faire en sorte que votre programme puisse aller relire la chaîne en question. Comparez la chaîne de caractères envoyée à la mémoire et celle relue. Il suffira de faire afficher la DEL en vert si elles sont identiques et en rouge dans le cas contraire. Cette démarche est un peu douteuse au départ, mais elle se raffinera avec le problème 3. Il faut prévoir un délai de 5 ms après l'écriture d'un octet en mémoire et l'accès

suivant à la mémoire. De cette façon, la mémoire peut compléter son cycle d'écriture sans problème.

Problème 2: Communication RS232

- Placer le cavalier DbgEN (mode actif). Les broches D0 et D1 du USART0 du ATmega324PA seront reliées au USART du ATMega8.
- La communication se fera à 2400 bauds, sans parité et les trames de 8 bits seront séparées par un bit d'arrêt. Du côté PC, vous n'avez pas à prendre ces détails en considération. Un petit programme, écrit spécifiquement pour ce cours, ajustera ce qu'il faut pour vous. Ce programme redirige les caractères ASCII affichables à l'écran (ou dans un fichier avec l'option -f). Il suffit de l'invoquer avec l'option de lecture:

```
% serieViaUSB -l
```

- Notez que le programme serieViaUSB bloquera tout simplement jusqu'à ce que des octets lui parviennent. Par contre, il affichera à l'écran les octets dès qu'ils arriveront sans autre commande de votre part.
- Pour ce qui est du code, la documentation de Atmel donne les procédures d'initialisation, de transmission et de réception. C'est le bon temps de copier sans sentiment de culpabilité! Il vous faudra peut-être ajuster quelques registres tout au plus:

```
void initialisationUART ( void ) {
        // 2400 bauds. Nous vous donnons la valeur des deux
        // premier registres pour vous éviter des
        complications
        UBRR0H = 0;
        UBRR0L = 0xCF;
        // permettre la reception et la transmission par le
        UART0
        UCSR0A = 'modifier ici' ;
        UCSR0B = 'modifier ici' ;
        // Format des trames: 8 bits, 1 stop bits, none parity
        UCSR0C = 'modifier ici' ;
}
```

- Il en va de même pour la procédure qui envoie un octet de la carte vers le PC. Il suffit de la trouver dans la fiche technique du ATmega324PA.

```
// De l'USART vers le PC

void transmissionUART ( uint8_t donnee ) {
        'modifier ici'
}
```

- Le chargé de laboratoire s'assurera que votre carte peut afficher à l'écran avec ce qui suit:

```
char mots[21] = "Le robot en INF1900\n";

uint8_t i, j;
for ( i = 0; i < 100; i++ ) {
     for ( j=0; j < 20; j++ ) {
               transmissionUART ( mots[j] );
     }
}
```

- Compiler votre code et le charger sur le microcontrôleur. Observer si le programme serieViaUSB affiche les octets à l'écran. Il n'est pas nécessaire de redémarrer le programme serieViaUSB à répétition pour réinitialiser la communication puisqu'il continuera toujours d'afficher à l'écran les octets qu'il reçoit sur le port RS232 sans se soucier d'une quelconque forme de protocole. Par contre, il peut être intéressant de presser le bouton de remise à zéro sur la carte mère pour redémarrer votre programme quelques fois de suite pour tester si la communication avec quelques séquences de suite se passe bien.

Problème 3: De la mémoire vers le PC par RS232

Avec le code utilisé pour l'exercice précédent, transmettre les valeurs inscrites en mémoire vers le PC. Le terminal sur le PC affichera le contenu de la mémoire *eeprom* dès que le bouton *reset* de la carte mère sera relâché. Cette mémoire sera lue de l'adresse 0x00 jusqu'à ce qu'une valeur lue soit égale à 0xFF. Lors de l'évaluation, le chargé de laboratoire utilisera une mémoire externe contenant un message secret.

Question intéressante

- Dans le cas du problème 2, combien de cycles d'horloge se produisent entre 2 caractères transmis. Serait-il mieux de transmettre par le port RS232 en contrôlant par un mode de scrutation ou d'interruption? Pourquoi?

Suivi logiciel

Le code développé pour résoudre ces problèmes devra être placé dans votre entrepôt Git. Les noms des fichiers et les répertoires où ils doivent être placés sont laissés à votre discrétion. Vous aurez besoin de ce code plus tard dans la session.

Pour aller plus loin avec le robot...

Généralement, après six semaines de travail avec le robot, certains étudiants commencent à développer une passion pour le robot et sa programmation. De plus, ils sont tentés d'adhérer

à un club étudiant (Poly eRacing, la machine des Jeux de Génie, Esteban, Elikos, Oronos, etc...) Dans bien des cas, les problèmes électroniques et de programmation à résoudre pour faire fonctionner le robot sont un peu les mêmes que ceux que ces clubs peuvent avoir à résoudre avec leurs propres systèmes. Une autre option est de s'intéresser à PolyFab.

En discutant avec les gens impliqués dans ces clubs, on se rend compte qu'on utilise souvent les mêmes fournisseurs de pièces et les mêmes références de base. Il s'agit d'un très bon point de départ pour ceux qui désirent aller plus loin avec un projet similaire (pour un club ou pour eux-mêmes). Il faut se rendre compte que ces bonnes adresses offrent des ressources qu'on peut avoir énormément de difficulté à trouver au coin de la rue... Les concepteurs du robot auraient perdu moins de temps s'ils avaient eu cette liste beaucoup plus tôt...

Enfin, pour ceux et celles qui voudraient approfondir leurs connaissances du fonctionnement du robot lui-même, il y a la section «fichiers» du site web qui présente l'ensemble des fichiers ayant servi à la conception du robot (plan, schéma, code source, etc...) Il faut mentionner que le robot est un projet à code source ouvert (*open source*) et qu'il peut être modifié ou adapté à des projets particuliers (y compris les projets pour les clubs étudiants). On pourra remarquer la contribution de quelques étudiants à cette section. Bien entendu, la complexité technique de ce qui est présenté dans cette section dépasse largement le cadre du cours INF1900, mais il s'agit de sujets qui ne sont pas hors de porté pour des gens ayant le désire d'aller plus loin dans le domaine.