INF1900

Projet initial de système embarqué
Travail pratique No. 4
Les registres spéciaux et les périphériques internes

Objectif: Explorer les ressources internes de l'AVR
Durée: Une semaine
Travail préparatoire: Aucun
Documents à remettre: Aucun fichier à remettre. Par contre, il s'agit d'un laboratoire sujet à un quiz de laboratoire la semaine prochaine et le code développé pour ce laboratoire pourra être réutilisé assez directement pour le fonctionnement du robot pour l'épreuve finale.
Présentation en classe: fichier LibreOffice Impress

«L'ordinateur obéit à vos ordres, pas à vos intentions»

-Anomyme

Introduction

Jusqu'ici, le projet a permis d'explorer le fonctionnement des ports du microcontrôleur et de la carte mère en général. Il est maintenant temps de regarder d'un peu plus près les ressources internes de l'AVR, spécialement celles qui nous seront utiles pour contrôler le robot.

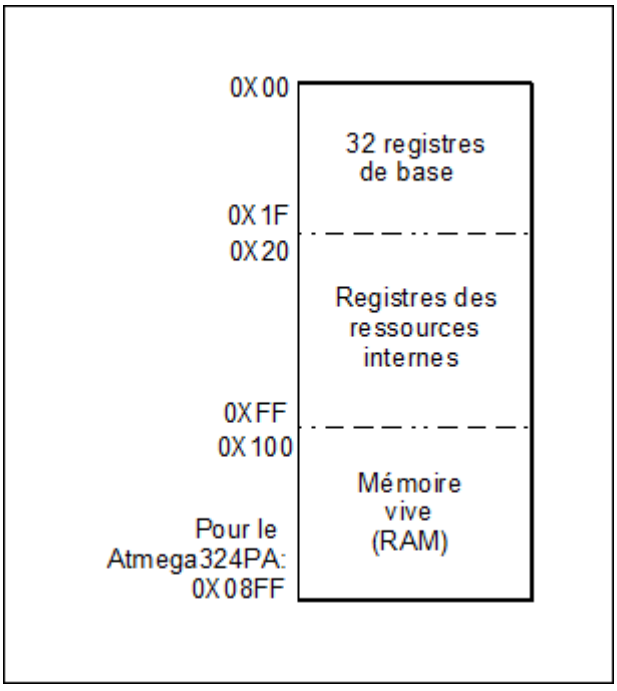
Les microcontrôleurs AVR disposent de plusieurs éléments de contrôle à l'intérieur même de la puce. La plupart de ces sous-systèmes peuvent être configurés et utilisés par des registres spéciaux placés à des adresses précises en mémoire. L'ajout de tels dispositifs matériels permet de leur déléguer certaines responsabilités et de libérer l'unité centrale pour lui permettre de faire d'autres traitements. Par contre, ces nouveaux périphériques internes doivent être gérés correctement et l'on doit toujours garder à l'esprit qu'ils travaillent en parallèle avec le reste du système. La synchronisation des activités devient donc importante.

Pour entreprendre cette étape, nous ferons un survol des mémoires et des périphériques internes de l'AVR. Pour les détails, il vous faudra lire la documentation du ATmega324PA de toute façon. Par la suite, vous referez certains exercices effectués durant les semaines précédentes, mais en réutilisant cette fois les ressources internes dédiées pour arriver aux mêmes résultats. Il y aura aussi des exercices qui introduiront de nouveaux concepts. Cependant, il faudra commencer par une présentation des mémoires de l'AVR puisqu'il s'agit probablement du meilleur point de départ pour arriver à saisir l'interaction avec les périphériques internes.

Mémoires de l'AVR ATmega324PA

Il y a 3 mémoires principales dans le microcontrôleur utilisé, soit la mémoire de programmation de type flash, la mémoire vive (*RAM*) et la mémoire morte (*ROM*). Chaque mémoire est indépendante des autres et est adressable séparément. La première est celle où le programme réside. Depuis le début du projet, c'est elle qui héberge vos programmes. Comme vous l'avez déjà utilisée depuis quelques semaines, il ne reste pas énormément de détails à ajouter sur cette mémoire de programmation. Le fait d'avoir une zone séparée pour loger le programme à exécuter permet l'utilisation d'une technologie spécifique de mémoire qui ne s'efface pas facilement. Ainsi, le programme peut résider longtemps à l'intérieur du circuit, et ce, même si l'alimentation est coupée, sans devoir être rechargé à chaque démarrage du microcontrôleur. Les ordinateurs PC procèdent de façon totalement différente et chargent les programmes constamment en mémoire vive. Mais les PC viennent avec un disque dur interne ce qui n'est pas le cas de notre carte mère...

Le second type de mémoire de l'AVR est une mémoire morte à laquelle on peut accéder par des instructions spéciales pour contenir des données pour une longue période de temps. Elle ne sera pas utilisée durant le cadre du projet puisqu'on lui préférera la mémoire externe disponible sur la carte mère et qui sera introduite la semaine prochaine. Il ne reste donc qu'à parler de la mémoire vive. On dit mémoire vive, mais en fait, il y a trois sections distinctes dans cet espace adressable comme le montre la figure ci-dessous tirée du livre de Christian Tavernier «Microcontrôleur AVR - Description et mise en oeuvre» (voir la section références).



Section de la mémoire vive du ATmega324PA

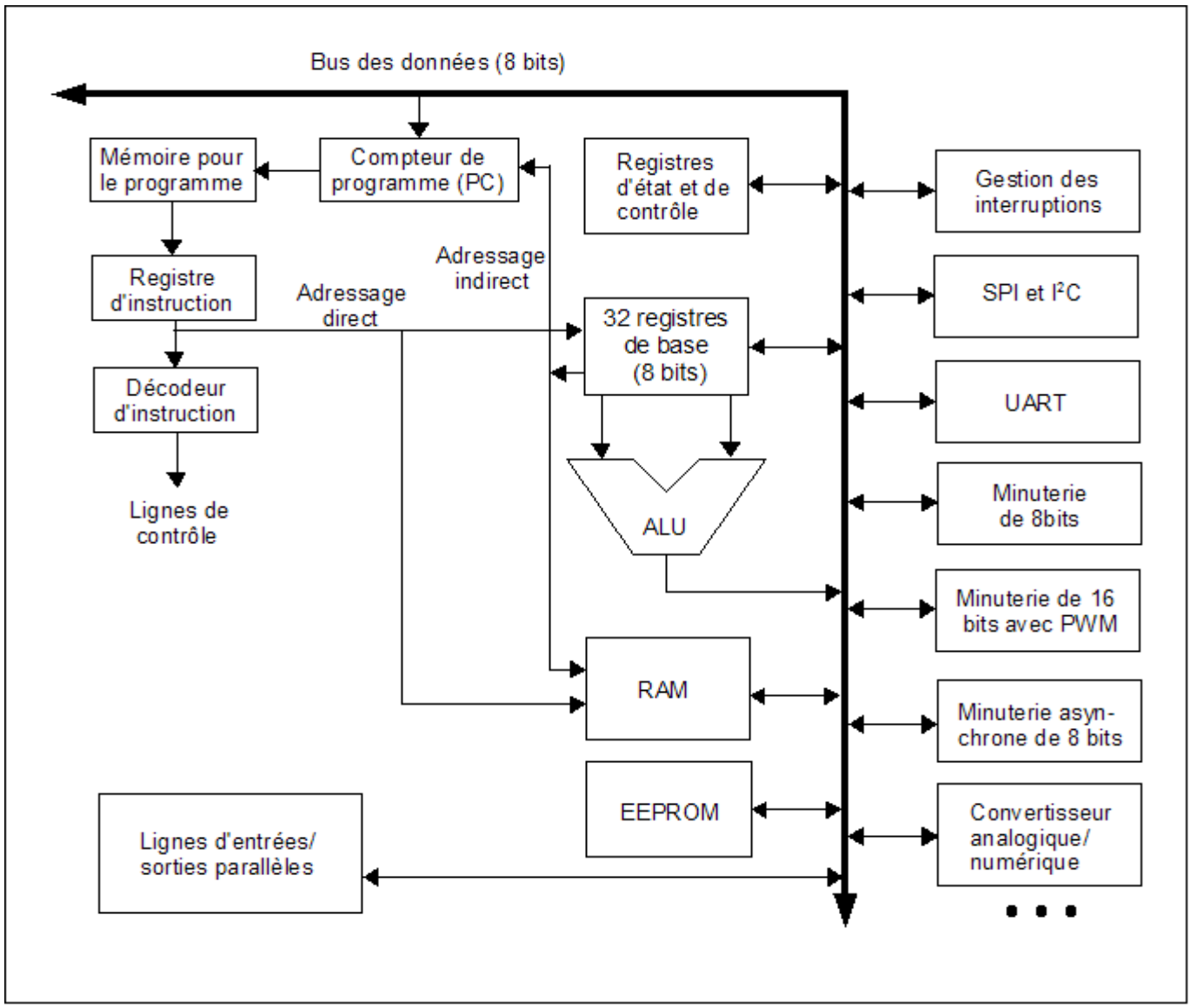
Comme on peut le voir, les registres de travail du processeur, au nombre de 32 (soit de 0 à 31 - 0x1F en hexadécimal), font partie de la mémoire vive! C'est une caractéristique assez particulière de la série AVR. La section suivante est réservée pour les registres spéciaux. Enfin, la dernière section est la «vraie» mémoire vive à usage général au sens où on l'entend le plus couramment. En réalité, du point de vue de la programmation, des mnémoniques sont associées aux adresses mémoires des deux premières sections. Ce que cette organisation signifie est que vous n'avez pas vraiment à savoir les détails précis des adresses des deux premières sections puisque vous y accéderez plutôt avec des noms de registres. Vous direz, par exemple, pour accéder au registre d'état (Status REGister):

```
uint8_t maChereVariable = SREG ;
```

En réalité, ce registre se situe dans la mémoire vive à l'adresse 0x3F, mais vous n'avez pas besoin d'en tenir compte. Si cette façon de voir vous déplaît, vous pouvez aussi considérer que la mémoire vive n'est constituée pratiquement que de la dernière section et que celle-ci commence à l'adresse 0x100. Une façon encore plus simple de voir la chose est de se dire que le compilateur verra lui-même à gérer l'utilisation des registres de base de même que de la dernière section de la mémoire vive et que nous accéderons aux registres spéciaux de la deuxième section par leur nom dans nos programmes tout simplement! Ces précisions devraient vous permettre de lire la documentation plus facilement sans trop vous perdre dans des détails utiles uniquement pour une utilisation spécialisée du microcontrôleur ou lorsque la programmation est en langage d'assemblage.

Les ressources internes, les interruptions et la scrutation

La figure suivante montre l'architecture interne de l'AVR (également tirée du livre de Christian Tavernier). Comme on peut le voir, la partie de gauche est semblable à ce qu'on peut trouver pour n'importe quel processeur avec un fichier de registres et une unité arithmétique et logique (UAL, *ALU* en anglais). Par contre, dans la section de droite, on peut voir certains blocs de périphériques. Dans le cadre du projet, nous aurons surtout besoin des minuteries (*timers*) et du UART. Plus tard, nous aurons aussi besoin du convertisseur analogique/numérique. Nous décrirons ici en quelques lignes les ressources utiles, mais il convient d'abord de parler des interruptions.



Architecture générale d'un microcontrôleur AVR

Mais en premier lieu, qu'est-ce qu'une interruption? L'interruption est l'une des deux méthodes pour savoir où en est un sous-système, l'autre étant la scrutation (*polling* en anglais). Ces concepts deviennent un peu plus clairs à partir d'exemples. Vous avez une montre à votre poignet et vous ne devez absolument pas manquer un rendez-vous important dans 2 heures. Il y a deux façons de vous y prendre pour gérer votre temps d'ici là tout en n'oubliant pas votre rendez-vous. La première consiste à regarder votre montre à intervalles plus ou moins réguliers pour savoir le temps qu'il reste avant votre rendez-vous. L'autre méthode est d'ajuster l'alarme sur votre montre pour qu'elle se mette à sonner juste avant votre rendez-vous pour vous avertir qu'il est temps de vous y rendre. La scrutation caractérise la première méthode alors que la seconde peut assez bien représenter ce qu'est une interruption. Avec la scrutation, le système principal demande à un sous-système de lui dire où il en est rendu dans sa tâche alors que par interruption, le sous-système avertit lui-

même la partie principale de la fin d'une étape importante (et peut-être finale) de sa tâche. Dans la programmation des microcontrôleurs, les 2 méthodes sont employées pour synchroniser des opérations, entre elles ou par rapport à un déroulement principal.

Depuis le début du projet, vous avez employé la scrutation pour arriver à savoir si le bouton-poussoir sur la carte avait été enfoncé par l'utilisateur pour changer la couleur de la DEL. Par contre, vous auriez pu avoir recours à la programmation par interruption pour arriver aux mêmes fins. C'est d'ailleurs ce que vous devrez faire plus loin comme exercice. La facilité de la programmation par scrutation vient du fait que le code suit une évolution prévisible puisque la succession des opérations est dictée ligne par ligne dans un déroulement linéaire. Avec les interruptions, ce cheminement est moins évident puisque le cours de l'évolution du programme peut être changé à tout moment pour répondre aux actions prévues par une interruption. Ce n'est pas tout. Une fois que les instructions d'une interruption ont été accomplies, il faut revenir au déroulement de ce qui était en cours juste avant l'interruption. Vous savez tous comment il peut être difficile de se replonger dans la lecture d'un document après avoir été interrompu par un coup de téléphone inattendu. Le principe est le même, il faut prendre le temps de noter où on en était avant l'interruption pour éventuellement pouvoir retourner au déroulement de l'action principale au point où on l'avait laissée. Dans le cas de la lecture d'un livre, on laissera un signet au bon endroit dans le livre. Dans le cas des processeurs, on sauvegarde le contenu de certains registres pour les recharger avec les bonnes valeurs après l'interruption. Heureusement, le compilateur que nous utiliserons ici se chargera pour nous de bien gérer lui-même les registres lors d'interruptions ce qui nous facilitera la tâche.

Pour en revenir à notre figure montrant l'architecture de l'AVR, on peut donc dire de façon générale que les périphériques de la partie droite et l'unité centrale sur la gauche peuvent se synchroniser en utilisant soit une approche par scrutation ou soit par interruption. Généralement, des registres spéciaux sont ajustés par programmation pour permettre ou non à certaines interruptions d'agir sur le déroulement du programme si on le souhaite. Si on préfère la scrutation, le programme ira voir lui-même l'état de certains autres registres pour voir ce qui se passe avec les périphériques internes.

Remarque très importante

Atmel est une compagnie qui a été acheté par la rival Microchip. En octobre de la même année la documentation du microcontrôleur a subi une révision. Malheureusement, certaines modifications ont été introduites qui font que la version 2.00 de la librairie AVRLibC ne fonctionnent par avec cette révision. Nous vous recommandons donc vivement de prendre la version 2015 du même document. Le bas de page de la documentation, vers la droite, vous indique le mois et l’année de la date de publication. Il vous faudra consulter régulièrement ce fichier PDF au cours de la session. Il est recommandé de la placer dans votre compte pour

éviter de le télécharger inutilement à répétition. Éviter également de le placer sous Git où il prendra trop d'espace inutilement.

Problèmes

Problème 1: Contrôle d'une del par interruption

Il vous faudra reprendre le code écrit pour le problème 2 de la semaine 2. Il vous faudra effectuer des modifications pour en arriver à un contrôle par interruption plutôt que par scrutation comme vous l'avez fait la première fois, sans trop le savoir:

- Il faut avoir un nouveau fichier «*include*» pour avoir accès aux éléments de programmation par interruptions. Les explications sont dans la documentation sur AVRLibC. Consultez cette section avant toute chose:
  <avr/interrupt.h>: Interrupts
- Il vous faudra adapter la déclaration de la variable décrivant l'état de la machine à état fini pour inclure l'attribut «volatile». CHAQUE FOIS QU'UNE VARIABLE GLOBALE EST MODIFIÉE DANS UNE ROUTINE D'INTERRUPTION, ELLE DOIT AVOIR CET ATTRIBUT «VOLATILE». Considérez-vous averti! Il se peut que vous ayez à modifier d'autres variables de la même manière. Les raisons de cette modification sont un peu complexes. Mentionnons simplement qu'elles ont à voir avec les optimisations de votre code effectuées par le compilateur:

```
volatile uint8_t etat = 0; // selon le nom de votre variable
```

- Il faudra préciser votre routine d'interruption. Nous utiliserons l'interruption externe zéro puisque le bouton-poussoir de la section «Interrupt» de la carte mère est relié au port D2. Il faudra s'assurer d'avoir un cavalier sur IntEn bien entendu. On utilisera l'instruction ISR pour contrôler une interruption. Voici les éléments les plus importants de la routine:

```
// placer le bon type de signal d'interruption
// à prendre en charge en argument
ISR ( 'modifier ici' ) {
    // laisser un delai avant de confirmer la réponse du
    // bouton-poussoir: environ 30 ms (anti-rebond)
    _delay_loop_ms ( 30 );
    // se souvenir ici si le bouton est pressé ou relâché
    'modifier ici'
    // changements d'états tels que ceux de la
    // semaine précédente
    'modifier ici'
```

```
        // Voir la note plus bas pour comprendre cette instruction et son
        rôle
        EIFR |= (1 << INTF0) ;
}
```

- Il est toujours plus facile d'avoir une routine d'initialisation qui ajuste bien des paramètres avant le début du traitement principal, surtout dans le cas présent:

```
void initialisation ( void ) {
        // cli est une routine qui bloque toutes les interruptions.
        // Il serait bien mauvais d'être interrompu alors que
        // le microcontroleur n'est pas prêt...
        cli ();

        // configurer et choisir les ports pour les entrées
        // et les sorties. DDRx... Initialisez bien vos variables
        'modifier ici'

        // cette procédure ajuste le registre EIMSK
        // de l’ATmega324PA pour permettre les interruptions externes
        EIMSK |= (1 << INT0) ;

        // il faut sensibiliser les interruptions externes aux
        // changements de niveau du bouton-poussoir
        // en ajustant le registre EICRA
        EICRA |= 'modifier ici' ;

        // sei permet de recevoir à nouveau des interruptions.
        sei ();
}
```

- Pour ce qui est de la boucle principale sans fin, elle devrait se contenter de générer les bons signaux de sortie pour que la DEL soit de la bonne couleur en sachant dans quel état se trouve le système.

- La routine ISR ne peut pas être interrompue durant son déroulement par une autre interruption (à part le *reset* et quelques autres, celles que la documentation sur le ATmega324PA qualifie de "type 2" - section AVR CPU Core – 8.8 Reset and Interrupt Handling, page 29 et suivantes). La documentation d'AVRLibC donne la même information. Bien entendu, l'utilisateur peut choisir le comportement inverse en utilisant "#define ISR_NOBLOCK" si l'on doit prendre en charge des interruptions réentrantes, ce qui n'est pas toujours désirable. La documentation du ATmega324PA mentionne:

«*When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.*»

- Donc, si un second signal d'interruption arrive durant l'exécution de ISR, l'AVR s'en souvient (le bit INTF0 est activé dans le EIFR) et la routine ISR sera exécutée une seconde fois, une fois la première terminée... Ces extraits de la documentation (pages 29 à 31) résument exactement le comportement du mécanisme des interruptions avec les AVR:

  «... *Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enable, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flags(s) will be set and remembered until the global interrupt enable bit is set, and will then be executed by order of priority.*»

  «*When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.*»

Problème 2: Jeu de réflexe avec l'utilisation d'une minuterie

Le défi est de concevoir un jeu de réflexe. Quand le microcontrôleur démarre, il attend 10 secondes et fait clignoter la lumière rouge pendant 1/10 de seconde. Quand la lumière est éteinte, le joueur doit peser sur le bouton aussitôt que possible. Si le joueur pèse sur le bouton à l'intérieur d'une seconde, la DEL devient verte. Si le joueur pèse sur le bouton passé une seconde ou ne pèse pas du tout, la lumière prend la couleur rouge. Dans un cas comme dans l'autre, la DEL conserve sa couleur indéfiniment. Il faut peser sur *reset* pour recommencer.

Le délai maximum d'une seconde pour permettre à l'utilisateur de peser sur le bouton sera calculé par le compteur de la minuterie qui générera une interruption après la période de temps allouée. Les autres délais (celui de 1/10 de seconde et de 10 secondes) peuvent être fait avec des fonctions de délais ou par la minuterie, comme bon vous semble. La détection du bouton-poussoir se fera également en détectant une interruption externe, tout comme pour l'exercice précédent. Voici une procédure:

- Lorsque survient une interruption, il faut surtout se souvenir qu'elle est survenue. On choisira donc deux variables globales de type volatile:

```
volatile uint8_t minuterieExpiree;
volatile uint8_t boutonPoussoir;
```

- Il faut deux routines d'interruptions très simples (mais il faut déterminer le type de signal que chacune doit gérer):

```
ISR ( 'modifier ici' ) {
        minuterieExpiree = 1;
}

ISR ( 'modifier ici' ) {
        boutonPoussoir = 1;
        // anti-rebond
        'modifier ici'
}
```

Pour arriver à avoir une interruption de la minuterie («timer»), on doit ajuster des registres. On placera tout ce qu'il faut dans une routine qu'on appellera au moment opportun. Il faudra fouiller dans la documentation d'Atmel pour savoir comment le tout peut être ajusté:

```
void partirMinuterie ( uint16_t duree ) {
        minuterieExpiree = 0;
        // mode CTC du timer 1 avec horloge divisée par 1024
        // interruption après la durée spécifiée
        TCNT1 = 'modifier ici' ;
        OCR1A = duree;
        TCCR1A = 'modifier ici' ;
        TCCR1B = 'modifier ici' ;
        TCCR1C = 0;
        TIMSK1 = 'modifier ici' ;
}
```

- Il faudra avoir, tout comme pour l'exercice précédent, une routine d'initialisation.

- Il y a peu à dire sur la routine principale. Vous pouvez faire passer les 10 secondes d'attente en utilisant la procédure de la semaine 2 (_delay_ms ou autre). La seule complication arrive lorsque l'on attend une réponse, soit de l'utilisateur ou soit de la minuterie. Voici ce qu'il faut faire:

```
do {
```

```
        // attendre qu'une des deux variables soit modifiée
        // par une ou l'autre des interruptions.
    } while ( minuterieExpiree == 0 && boutonPoussoir == 0 );

    // Une interruption s'est produite. Arrêter toute
    // forme d'interruption. Une seule réponse suffit.
    cli ();
    // Verifier la réponse
    'modifier ici'
```

Problème 3: Le PWM de façon matérielle

Lorsque votre robot avancera de façon autonome, il sera fastidieux de générer une onde PWM de la façon dont vous l'avez fait la semaine dernière (semaine 4) puisque le microcontrôleur sera occupé à faire bien d'autres choses. Il vaudra mieux utiliser les ressources internes pour arriver au même but.

Le timer1 du ATmega324PA possède un mode de fonctionnement qui permet de générer 2 signaux et de les faire sortir directement sur des broches du port D. On utilisera donc cette minuterie, mais cette fois, il n'y a pas lieu de générer d'interruption. Les moteurs peuvent fonctionner tout seuls sans autre assistance. Plus tard dans le projet, il faudra ajuster la vitesse des roues pour arriver à tourner, mais ce détail a peu d'intérêt pour l'instant. Par contre, tout comme la semaine passée, il vous faudra donner un signal de direction au circuit du pont en H. Le seul détail qui change par rapport à la semaine précédente est la fréquence du signal PWM. On se contentera d'une seule fréquence (donnée plus bas), mais on continuera de générer des signaux PWM de 0%, 25%, 50% et 75% et 100% pour des durées de 2 secondes chacun. Le rapport a/b du PWM sera réglé en passant correctement un ou des arguments de la fonction. Ces arguments permettront d'ajuster certains registres spéciaux.

- L'ajustement du PWM dans le timer1 se fait en ajustant des registres:

```
    void ajustementPWM ( 'modifier ici' ) {
        // mise à un des sorties OC1A et OC1B sur comparaison
        // réussie en mode PWM 8 bits, phase correcte
        // et valeur de TOP fixe à 0xFF (mode #1 de la table 17-6
        // page 177 de la description technique du ATmega324PA)
        OCR1A = 'modifier ici' ;
        OCR1B = 'modifier ici' ;

        // division d'horloge par 8 - implique une frequence de PWM fixe
        TCCR1A = 'modifier ici' ;
```

```
        TCCR1B = 'modifier ici' ;
        TCCR1C = 0;
}
```

- La routine principale (main) appellera tout simplement la procédure ajustementPWM et s'assurera que les ports de sortie (surtout le port D) soient bien ajustés. Il est toujours bon de consulter le haut de la page 15 du document d'Atmel pour avoir une figure illustrant les broches de l’AVR.
- Comme il y a deux moteurs sur le robot, il faut générer deux signaux PWM. Le timer1, tel qu'ajuster dans la routine ajustementPWV, en générera justement deux. Il est important de garder en tête le schéma de figure 16-1 à la page 131 puisqu'il montre que les OC1A et OC1B sont les sorties PWM.
- Une assignation correcte du registre TCCR1A fait en sorte que certains ports ne sont plus accessibles en sortie puisque OC1A et OC1B sont sélectionnées par le multiplexeur de la figure 17-4 à la page 176.
- Les minuteries 0 et 2 (timer0 et timer2) sont très semblables à la minuterie 1 dans leur fonctionnement de base mais il y a des différences avec certains modes plus avancés. La plus grande différences est peut-être que les minuteries 0 et 2 sont de 8 bits seulement mais elles peuvent aussi générer chacune deux signaux PWM directement sur des broches en sortie. Elles font l’objet des chapitres 16 et 19 respectivement plutôt que du chapitre 17.

Suivi logiciel

Le code développé pour résoudre ces trois 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 **absolument** besoin de ce code plus tard dans la session. De plus, la matière étudiée ici est sujette à une évaluation lors du quiz prévu prochainement. Elle en constituera même la matière principale.