République **A**lgérienne **D**émocratique et **P**opulaire
Ministère de L'**E**nseignement **S**upérieur et de la **R**echerche **S**cientifique

UNIVERSITE **M**OULOUD **M**AMMERI DE **T**IZI-**O**UZOU

FACULTE DE **G**ENIE **E**LECTRIQUE ET D'**I**NFORMATIQUE
DEPARTEMENT D'ELECTRONIQUE

Mémoire de Fin d'Etude de MASTER ACADEMIQUE

Spécialité : **Electronique biomédicale**

Présenté par
CHAIB Youcef
GACEM CHAOUCHE Farhat

Mémoire dirigé par **Dr AIT OUAMER Farid**

Thème

Programmation d'un microcontrôleur et commande d'un moteur pas à pas

Mémoire soutenu publiquement le 18 septembre 2014 devant le jury composé de :

M	**ACHOUR**	Président
M	**OUALOUCHE**	Examinateur
M	**LAZRI**	Examinateur

Laboratoire LCPQ (laboratoire chimique et physique quantique)

Remerciements

Nous remercions Allah le tout puissant qui nous a donné la force, la volonté et la patience durant toutes ces années d'étude. Il nous a permis, à la fin, de réaliser de ce projet.

Nous sommes reconnaissants à notre pays, l'Algérie, qui nous a offert une école, un lycée et une université et qui nous ont formé.

Je tiens à remercier énormément l'encadreur Dr. AIT-OUAMER Farid pour m'avoir encadrer et diriger. J'ai exploité les conseils, les aidés et les orientations pour nous mener à un bon terme à la réalisation de cette recherche.

J'exprime ma reconnaissance au personnel du laboratoire LPCQ (laboratoire de physique et chimique quantique) y compris son directeur, Hamid Bouzar, qui nous a ouvert la porte de bienvenu.

Ma gratitude va à tous les enseignants qui ont participé à ma formation durant ce cycle d'étude et en particulier ceux du département d'Electronique.

Nous tenons aussi à remercier tous ce qui ont contribué de loin ou de près à la réalisation de ce travail.

Un grand merci aux membres du jury.

Dédicace

C'est avec une grande émotion que je dédie ce modeste travail de fin d'étude aux personnes qui me sont les plus chers.

Je citerai :

Mon père et ma mère pour leurs grand amour et leurs sacrifices.

Mes chers frères : Djamel, Krimo, Hocine et Redouane.

Ma chère sœur Cylia.

Mes cousins et cousines et toute la famille CHAIB.

Mes camarades de la faculté, et tous ceux qui m'ont aidé de près ou de loin à la réalisation de ce travail.

A tous mes amis.

CHAIB Youcef

Dédicace

C'est avec une grande émotion que je dédie ce modeste travail de fin d'étude aux personnes qui me sont les plus chers.

Je citerai :

Mon père et ma mère pour leurs grand amour et leurs sacrifices.

Mon cher frère Samir.

Mes chères sœurs : Fadhila et Samira.

Mes cousins et cousines et toute la famille GACEM CHAOUCHE .

Mes camarades de la faculté, et tous ceux qui m'ont aidé de près ou de loin à la réalisation de ce travail.

A tous mes amis.

GACEM CHAOUCHE Farhat

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Nous vivons dans un monde global où les avancées techniques, technologiques et scientifiques sont rapides. L'Algérie, à travers ses instituts, ses universités et ses usines doit participer à ce développement. Un sujet pratique et expérimental pour un mémoire de master devient attrayant s'il utilise la technologie moderne. Pour ce faire, un choix de sujets abordables peut se faire en tenant compte de l'environnement dans lequel on fonctionne et en utilisant des ressources accessibles et existantes.

On a opté pour un sujet pratique, prometteur et pluridisciplinaire intitulé " Programmation d'un microcontrôleur et interface à un moteur pas à pas ". Ce sujet combine un microcontrôleur de type Atmel programmable en langage C via un programmateur et d'une interface à un moteur pas à pas. Le programme est écrit de façon à positionner un chariot en une position donnée. Le mécanisme adopté est basé sur le bras d'un scanner.

Tout en tentant de respecter les délais de temps et pour mener à terme pareil tâche, plusieurs ressources sont requises et l'utilisation de plusieurs disciplines (électronique, mécaniques, physiques, informatique) est essentielle. Ces techniques d'ingénieries et cette expertise sont capitales pour mener à bien les projets futurs dans notre discipline que nous allons confronter dans nos carrières à venir.

Bien qu'on s'est procuré du commerce quelques composants (puce et autres), on a aussi choisi, là où c'est possible, l'utilisation de produits de recyclage (moteurs, mécanismes mécaniques, composant électroniques, et autres). Cette philosophie est en accord avec la tendance international du respect de la terre, de l'environnement et de la réutilisation d'objets qui paraissent obsolètes.

Ce mémoire est divisé en cinq grandes sections. Après cette introduction, on discutera du microcontrôleur Atmega8 de chez Atmel, des moteurs pas à pas, de l'utilisation d'une commande infrarouge à distance. On introduira ensuite dans une autre section la plateforme gratuite de programmation et de développement. Puis, On présentera les tests et résultats obtenus. On finira par une conclusion générale et quelques perspectives.

Les moteurs, en général, sont connus de longue date. Ils peuvent être considérés comme la force derrière le développement et l'industrialisation. Les moteurs électriques de

précision et de positionnement ont pris un essor grandissant grâce à l'électronique numérique et ce à partir de 1960. Plusieurs types de moteurs sont utilisés et les moteurs pas à pas est un choix impératif lorsqu'un déplacement ou une rotation par pallier ou incrémental est demandé.

Les moteurs pas à pas en combinaison avec des systèmes mécaniques sont des mécanismes de positionnement de précision. On les retrouve dans plusieurs domaines de l'industrie et de la recherche comme la robotique, l'automatisme, l'astronomie, l'instrumentation médicale et autre. Plusieurs périphériques informatiques comme l'imprimante, disque dur, lecture de disquette et scanner les utilisent. Ces derniers sont une source formidable pour le recyclage de composants de grande qualité. Ils peuvent facilement être adaptés et réutilisés sur d'autres projets. C'est l'une des tactiques utilisées pour la conception de notre table de travail et d'expérimentation.

En effet, le moteur pas à pas qu'on a utilisé provient d'un scanner. Ce choix n'est pas fortuit, vu que de nos jours, les scanners peuvent avoir une précision de quelques dizaines de micromètres. Entre autre, le mécanisme d'entraînement par pignons et courroie nous a été très utile. Ce moteur de marque MITSUMI M35SP-7 a 5 fils. Donc c'est un moteur pas à pas unipolaire.

La carte électronique et le système optique de ce scanner ne nous servent pas. D'où la nécessité d'œuvrer à la confection d'un système de commande de notre conception pour contrôler ce moteur. Il existe plusieurs choix de configuration qui majoritairement nécessitent un microcontrôleur. L'intégration dans l'industrie des semi-conducteurs et la performance des puces réduits énormément le nombre de composants électroniques nécessaires. Aussi, l'acquisition sur le marché local et leur disponibilité nous impose des contraintes. Heureusement que plus plusieurs types de microcontrôleurs existent et peuvent être facilement acquis.

Il existe plusieurs familles de microcontrôleurs. Les microcontrôleurs Atmel (AVR), les pics (Microchip), les microcontrôleurs à cœur ARM, les microcontrôleurs Motorola et autres. Ces microcontrôleurs sont des circuits intégrés programmable sous une des plateformes de développement existantes et peuvent être programmés en plusieurs langages (C, C++, java, basic, assembleur, etc...). Ceci devrait faciliter la tâche de commande des moteurs.

Pour des raisons d'appréciations, nous avons choisi pour ce travail le microcontrôleur Atmega8 de la famille AVR. Celui-ci est basé sur le 8-bit processeur Intel. Il suffit de mentionner ici que ce microcontrôleur contient un jeu de 131 instructions. Additionnellement la compagnie Atmel offre un logiciel de programmation en C gratuit (AVR Studio) qui est très accessible et à une courbe d'apprentissage assez courte. Pour compléter le système de développement, il faut en rajouter un circuit électronique, i.e. un programmateur, permettant de flasher ce type de microcontrôleur.

Pour le programmateur, on distingue deux types d'interfaces : l'interface série et l'interface parallèle. Pour ce projet, on a besoin d'une interface de communication entre le microordinateur où le développement du logiciel et la simulation se font et le microcontrôleur.

L'utilisation du port parallèle ne nous intéresse pas car on a besoin de faire appel à des cartes de programmation dédiées aux ports parallèle et qui nous sont inaccessibles. Donc, il était raisonnable d'utiliser le port série et de réaliser nous même un programmateur moyennant quelques diodes, des résistances et un fil.

Notre tâche consiste, alors, à interfacer le microcontrôleur à un moteur pas à pas via un circuit intégré ULN2803 après avoir programmé, compilé et flashé le microcontrôleur par un programmateur à travers le port série RS232.

Afin de nous permettre d'écrire les routines nécessaires pour commander ce moteur pas à pas, plusieurs programmes tests ont étés développés. Ils seront discutés dans ce mémoire dans le chapitre consacré aux tests et résultats. Ce processus d'apprentissage est nécessaire vu que ce type de microcontrôleur est rarement utilisé dans nos instituts bien qu'il soit très employé à travers le monde. On ajoutera dans ce mémoire la procédure l'installation du logiciel de développement d'AVR Studio. On touchera aussi à quelques notions de base de la programmation en C.

Avec l'avènement de l'Internet, l'accès à l'information n'est plus une commodité propriétaire. Nous l'avons utilisé pleinement. Nous avons, pour ce mémoire, accéder à plusieurs documents, datasheet, sites, images et circuits que nous avons adapté ou réadapté et qui peuvent avoir ou ne pas avoir une restriction de copyright. On a fait un effort considérable pour les référencer dans la liste des références. Toute omission ou oubli n'est pas intentionnel. Nous tenons à remercier tous les auteurs, même anonymes, sans lesquels ce modeste projet n'aurait pas eu lieu.

Chapitre 1 :

Les microcontrôleurs AVR

1) Introduction

Les microcontrôleurs de la famille AVR d'Atmel possèdent de nombreuses caractéristiques. Bien qu'ayant un cœur similaire, les différents modèles se distinguent en termes de vitesse, mémoire, nombre d'entrées/sorties mais aussi au niveau de fonctions particulières. Ceci donne l'option de choisir le type de microcontrôleur selon de l'utilisation qui est requise.

Les microcontrôleurs de la famille Atmega sont basés sur la technologie CMOS ayants un bus à 8 bits et fondés sur l'architecture RISC. Ceci leurs permet d'exécuter des instructions durant un seul cycle d'horloge. Ainsi l'Atmega réalise des opérations s'approchant de 1 MIPS par MHZ. Ils sont conçus pour des systèmes à faible consommation électrique et un niveau électronique assez simplifié [2].

2) Histoire des microcontrôleurs

Les microcontrôleurs ont été inventés à Intel en 1969. Quand une société japonaise approcha Intel pour construire des circuits intégrés pour leurs calculatrices, Marcien Huff a utilisé son expérience antérieure sur le PDP-8 de proposer une solution alternative : un circuit intégré (IC) programmable.

Frederico Faggin transforma cette idée en réalité et Intel acheta la licence de la société japonaise (Busicom) pour créer le microprocesseur 4004 de 4 bits capable de 6000 d'opérations par seconde. Il fut rapidement suivi par le 8-bit 8008 en 1972. Les efforts d'Intel ont été imités par son compétiteur farouche (Motorola) mettant au marché la série 6800 avec le 8-bit et MOS Technologie pour seulement 25 $ chacun. Lançant ainsi l'intégration de milliards de transistors sur une puce de quelques cm^2.

Les microcontrôleurs utilisés dans notre laboratoire peuvent être acquit au prix de 3,50 $ pour une quantité de 100. Ils peuvent fonctionner à des fréquences allant jusqu'à 20 MHz, et contiennent jusqu'à 16 Ko de mémoire Flash programmable, 512 octets de EEPROM, et 1 Ko de SRAM. Ils sont aussi dotés de nombreux périphérique comme un ADC, une communication série, PWM, et JTAG [2].

3) Architectures Harvard et Van Neumann

L'architecture, dite architecture de Von Neumann, est un modèle pour microcontrôleur qui utilise une structure de stockage unique pour conserver à la fois les instructions et les

données requises ou générées par le calcul. La séparation entre le stockage et le processeur est implicite dans ce modèle. Une même instruction permet l'accès au programme ou aux données, cependant pas simultanément.

Cette architecture est maintenant principalement utilisée pour la conception des processeurs d'ordinateurs (PC, MAC). Ce type d'architecture nécessite plusieurs cycles d'horloge pour exécuter une instruction (recherche d'instruction, décodage, lecture opérande, exécution).

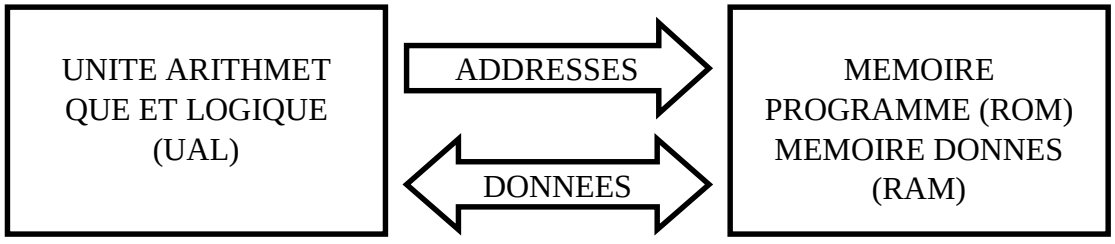


Figure 1-1: Architecture Van Neumann [3].

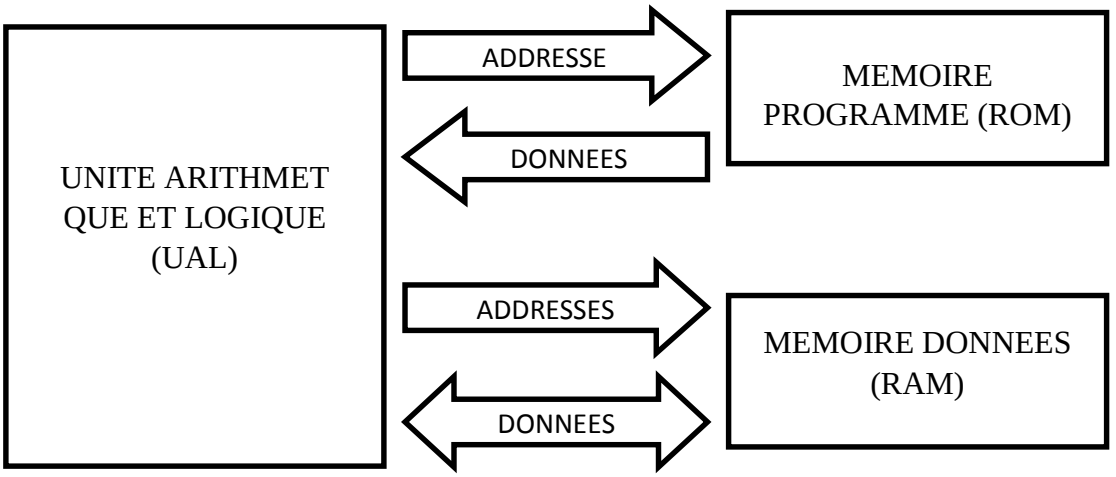


Figure 1-2 : Architecture Harvard [3].

L'architecture de type Harvard est une conception de microprocesseur qui sépare physiquement la mémoire de données et la mémoire programme. L'accès à chacune des deux mémoires s'effectue via deux bus distincts. Cette structure permet un accès simultané aux données et aux instructions l'exécution des programmes est plus rapide. En revanche elle nécessite des instructions différentes pour accéder à la mémoire programme et à la mémoire de

données. Cette architecture très employée pour la conception des processeurs de traitement de signal (DSP) est de plus en plus utilisée pour les microcontrôleurs d'usage généraux.

Ce type d'architecture, plus complexe à fabriquer permet une exécution du programme en PIPELINE (recherche d'instruction, décodage, lecture opérande, exécution en un cycle d'horloge) [3].

4) Avantages et inconvénients des microcontrôleurs

On cite ci-dessous les avantages et inconvénients de ces deux architectures

- Avantages

Diminution de l'encombrement du matériel et du circuit imprimé

Simplification du tracé du circuit imprimé (plus besoin de tracer de bus !)

Augmentation de la fiabilité du système nombre de composants diminués connexions composants/supports et composant circuit imprimé diminués

Intégration en technologie MOS, CMOS, ou HCMO (diminution de la consommation)

Le microcontrôleur contribue à réduire les coûts à plusieurs niveaux:

moins cher que les composants qu'il remplace

Diminution des coûts de main d'œuvre (conception et montage)

Environnement de programmation et de simulation évolués

- Inconvénients

le microcontrôleur est souvent surdimensionné devant les besoins de l'application

Investissement dans les outils de développement

Écrire les programmes, les tester et tester leur mise en place sur le matériel qui entoure le microcontrôleur

Les microcontrôleurs les plus intégrés et les moins coûteux sont ceux disposant de ROM programmables par masque

Incompatibilité possible des outils de développement pour des microcontrôleurs de même marque.

Fabrication uniquement en grande série >1000

Défaut relatif car il existe maintenant systématique des versions OTPROM un peu plus chère [4].

5) Le choix du microcontrôleur :

Atmel offre une vaste gamme de microcontrôleurs, et afin de choisir un microcontrôleur adéquat à notre projet. Pour cela nous avons fixé notre choix sur l'Atmega8 vu que le nombre de ports d'entrée/sortie est suffisant, ainsi que la mémoire flash qui a une capacité de 8Kbytes.

6) Présentation du microcontrôleur Atmega8

- Brochage

Le brochage du microcontrôleur Atmega8 est montré sur la figure 1-3. A gauche de cette figure on montre une photo de la puce à une échelle 2,5 à 3 fois de ses dimensions réelles et a droite, le label de chacun des 28 pins ainsi que leurs fonctions alternatives.

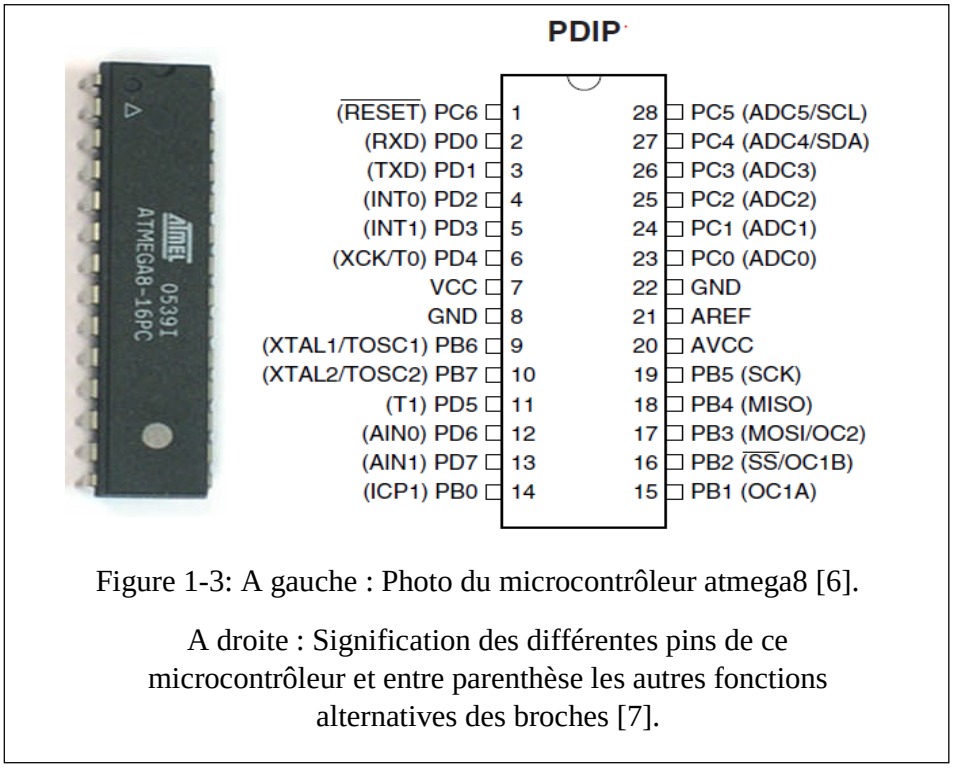


Figure 1-3: A gauche : Photo du microcontrôleur atmega8 [6].

A droite : Signification des différentes pins de ce microcontrôleur et entre parenthèse les autres fonctions alternatives des broches [7].

- Architecture

Le cœur AVR combine un riche jeu de 131 instructions avec 32 registres spéciaux travaillants directement avec l'Unité Arithmétique et Logique (ALU), qui représente le registre d'accumulateur A (B ou D) dans les microcontrôleurs classiques.

Ces registres spéciaux permettent à deux registres indépendants d'être en accès directs par l'intermédiaire d'une simple instruction et d'être exécutés en un seul cycle d'horloge. Cela signifie que pendant un cycle d'horloge ALU exécute l'opération et le résultat est stocké en arrière dans le registre de sortie, le tout dans un cycle d'horloge. L'architecture résultante est plus efficace en réalisant des opérations jusqu'à dix fois plus rapidement qu'avec des microcontrôleurs conventionnels CISC.

Les registres spéciaux sont dit aussi registre d'accès rapide et 6 des 32 registres peuvent être employés comme trois registre d'adresse 16 bits pour l'adressage indirects d'espace de données (X, Y & Z). Le troisième Z est aussi employé comme indicateur d'adresse pour la fonction de consultation de table des constantes.

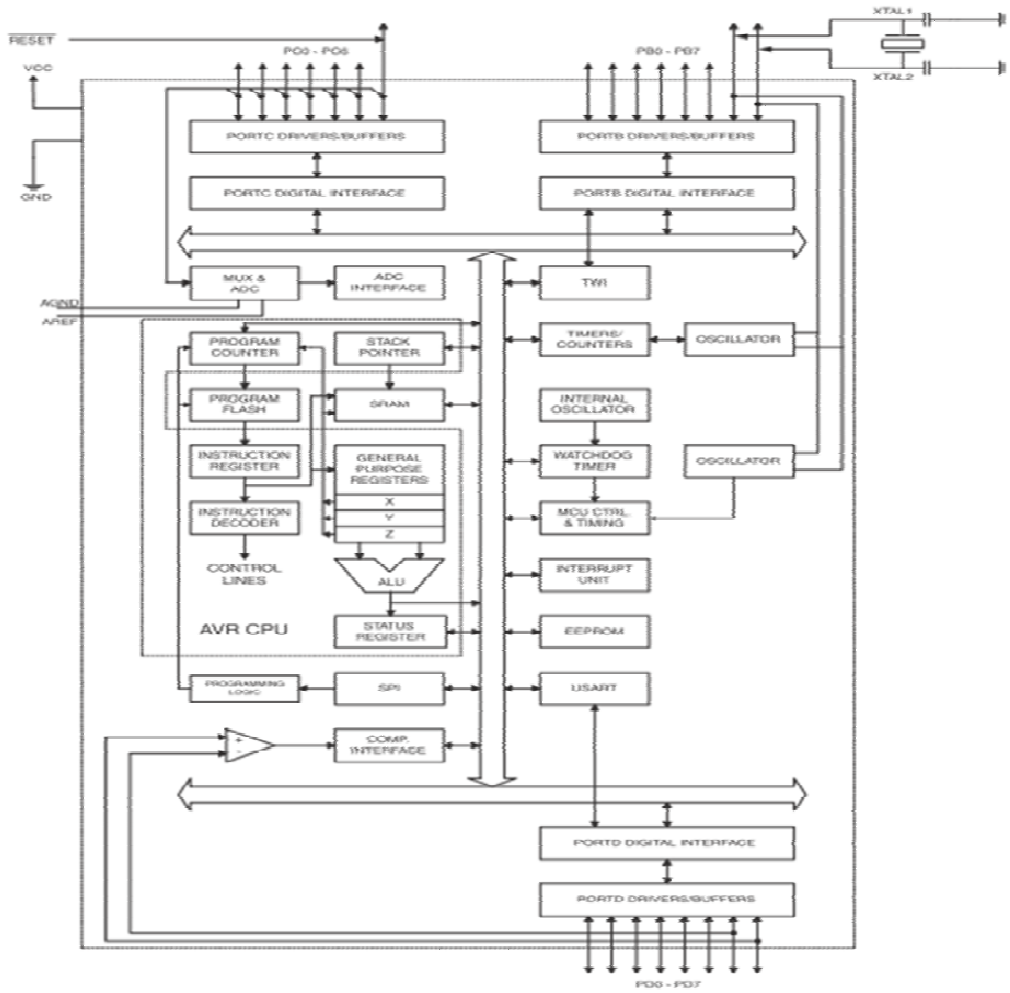


Figure 1-4 : Architecture interne de l’atmega8 montrant les différents blocs [1].

Les 32 registres sont détaillés dans le tableau 1 qui suit avec l’adresse effective dans la mémoire SRAM : Les informations sont diffusées par un bus de donnée à 8 bits dans l’ensemble du circuit.

Le microcontrôleur possède aussi un mode sommeil qui arrête l'unité centrale en permettant à la SRAM, les Timer/Compteurs, l’interface SPI d’interrompre la veille du système pour reprendre le fonctionnement.

Lors de l’arrêt de l’énergie électrique, le mode économie sauve le contenue des registres et gèle l'oscillateur, mettant hors de service toutes autres fonctions du circuit avant qu’une éventuelle interruption logicielle ou matérielle soit émise. Dans le mode économie, l'oscillateur du minuteur continue à courir, permettant à l'utilisateur d’entretenir le minuteur RTC tandis que le reste du dispositif dort. Le dispositif est fabriqué en employant la technologie de mémoire à

haute densité non volatile d'ATMEL. La figure 1-4 montre l'architecture interne du microcontrôleur Atmega8.

Les 32 registres sont détaillés dans le tableau 1 qui suit avec l'adresse effective dans la mémoire SRAM : Les informations sont diffusées par un bus de donnée à 8 bits dans l'ensemble du circuit.

La mémoire FLASH est reprogrammable par le système avec l'interface SPI ou par un programmateur de mémoire conventionnel non volatile [5].

Bit 0 à 7	Adresse	Registres spéciaux
R0	$00	
R1	$01	
Rn	$xx	
R26	$1A	Registre x partie basse
R27	$1B	Registre x partie haute
R28	$1C	Registre y partie basse
R29	$1D	Registre y partie haute
R30	$1E	Registre z partie basse
R31	$1F	Registre z partie haute

Tableau 1-1 : Adressage des 32 registres [5].

7) Les principales caractéristiques de l'Atmega8

Le microcontrôleur Atmega8 dispose :

De 28 broches numériques entrées /sorties, se trouvent sur les ports B, C et D dont 3 peuvent être utilisées en sorties PWM (largeur d'impulsion modulée) reparties selon l'ordre suivant : OC1B (PB2), OC1A (PB1), OC2 (PB3).

De 8 entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques). Ces entrées sont reparties sur les ports B, C, et D.

De 32 registres de travails universels qui dialoguent directement avec l'unité centrale UAL (Unité Arithmétique et Logique).

D'une mémoire flash de 8Kbytes. D'une mémoire SRAM de 1Kbytes. D'une mémoire de type EEPROM de 512 bytes.

Il dispose aussi de trois compteurs (Timer 0, Timer 1, Timer 2), le Timer 0 et le Timer 2 sont à comptage 8bits, et le Timer 1 à comptage 16 bits. Chaque Timer peut être utilisé pour générer deux signaux PWM [1].

Certaines broches peuvent avoir plusieurs fonctions différentes choisies par programmation, on distingue :

- PWM

Pour l'utilisation de la PWM, l'Atmega8 a 3 broches qui peuvent servir à cette fonction qui sont les broches OC1B (PB2), OC1A (PB1) et OC2 (PB3).

- Convertisseur Analogique/Numérique

Il possède un convertisseur Analogique/Numérique d'une résolution de 8bits, ce convertisseur peut être utilisé à travers 8 entrées multiplexées.

- Port série (USART)

Emission/Réception série via les deux broches TXD (PD1) / RXD (PD0).

- Gestion bus I2C

Ce bus est exploité via les deux broches SDA (PC4)/ SCL (PC5).

- Comparateur Analogique

Le comparateur analogique intégré dans l'Atmega8 peut être utilisé à travers les deux broches AIN0(PD6) et AIN1(PD7), ce comparateur peut déclencher une interruption [5].

- WatchDog Timer programmable

L'Atmega8 possède un compteur dit de chien de garde programmable pour générer des interruptions à la fin de son comptage et il peut être utilisé comme étant un compteur.

Gestion d'interruptions (plusieurs sources possibles) : En résumé -Interruptions liées aux entées INT0(PD2) et INT1(PD3).

- Interruption sur changement d'état des broches PCINT0 à PCINT27
- Interruption liées aux Timers 0, 1 et 2 (plusieurs causes configurables).
- Interruption liée au comparateur analogique.
- Interruption de fin de conversion ADC.
- Interruptions du port série USAR.
- Interruption du bus I2C [8].

- Les mémoires

Trois types de mémoires sont utilisés dans les microcontrôleurs Atmega. On trouve la mémoire EEPROM, la mémoire de donnée (SRAM) et la mémoire de programme (FLASH).

- La mémoire morte

La mémoire morte est de type EEPROM d'accès plus complexe contiendra la configuration du programme et les données importantes qui seront sauvé pendant l'absence de courant électrique. On peut écrire jusqu'à 100,000 fois dans l'EEPROM. La taille de l'EEPROM est fonction du modèle de microcontrôleur Atmega (de 256 bits à 8 Ko).

L'**EEPROM** est une mémoire programmable est effaçable électriquement. La particularité de cette mémoire et de pouvoir garder les informations stockées longtemps même hors tension.

L'Atmega8 contient 512 Bytes de données dans la mémoire EEPROM. Elle est organisée comme un espace de donné séparé, dans lequel des octets simples peuvent être lus et écrits. L'EEPROM a une endurance d'au moins 100,000 cycles d'écriture/effacement.

L'espace mémoire de l'EEPROM est accessible par l'utilisation de registres spéciaux d'accès. Le temps d'accès en écriture dans l'EEPROM est donné dans la table suivante :

Symbole	Nbre de cycle d'oscillateur RC interne	Temps typique de programmation
Ecriture EEPROM	8448	8.5 (ms)

Tableau 1-2 : Temps d'accès en écriture [1].

- La mémoire de donnée

La mémoire de donnée contient les 32 registres de travail, les 64 registres de commande et la mémoire SRAM pour les variables du programme de 1KBytes pour le modèle Atmega 8

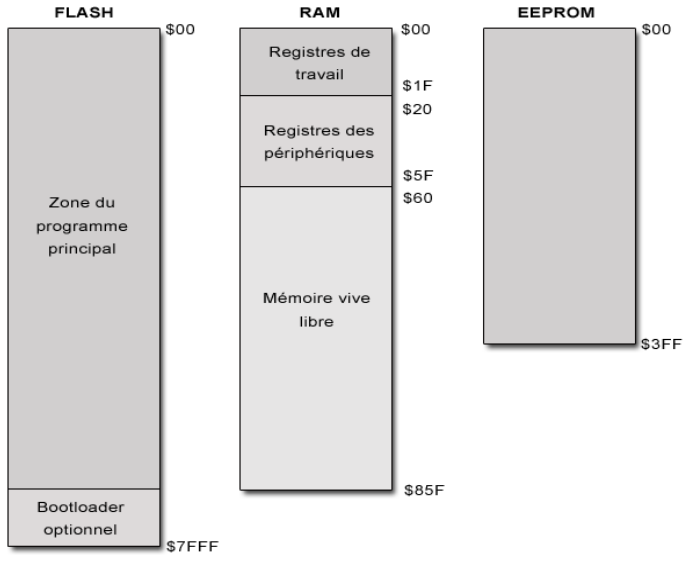


Figure 1-5 : Les différents types de mémoire et leurs adresses [4].

- La mémoire programme

La mémoire programme (flash) permet de stocker et de faire fonctionner le microcontrôleur, il contient de 4 à 256 Ko de programme selon le modèle du microcontrôleur. Le nombre d'écriture sur cette mémoire est limité à 10.000, largement suffisant pour la majorité

des applications. L'Atmega8 à 8 KBytes de flashs programme. Le schéma de la figure 1-5 montre les trois types de mémoires de L'Atmega8 [4].

- Le processeur CPU

Le CPU a pour mission de rechercher les instructions en mémoire, de les décoder et de les exécuter. Il est composé de plusieurs éléments. Le CPU peut être divisé en 3 blocs principaux :

CPU = UAL + UC + registres CPU

L'unité arithmétique et logique (UAL) est chargée des calculs +, -, *, /, AND, OR, NOT. L'unité de commande chargée de traduire puis d'exécuter les commandes. Son rôle est d'aller chercher une information en mémoire centrale, d'analyser cette instruction (décodage), d'exécuter cette instruction, de localiser l'instruction suivante. En plus on un décodeur d'instruction, un séquenceur et des circuits de commande.

- Les registres

- Les registres systèmes

Les registres systèmes permettent de programmer le microcontrôleur selon le choix d'utilisation que le programmeur veut en faire. Ils sont aux nombres de 4, mais seul les deux premiers sont importants. Le fonctionnement du microcontrôleur est modifié lors de la programmation de la mémoire FLASH.

Registre MCUCR (MCU control register)

C'est le registre de contrôle de l'unité centrale du microcontrôleur. Ce registre détermine le fonctionne en mode sommeil et la gestion des interruptions externe 0 et 1.

SE : (sleep enable) Mise en sommeil de l'unité centrale

Adresse	7	6	5	4	3	2	1	0
$35	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	IC00
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E

Tableau 1-3 : Les 8 bits du registre MCUCR [5].

SM2	SM1	SM0	Mode sommeil (sleep)
0	0	0	Fonctionnement normal
0	0	1	ADC à réduction bruit
0	1	0	Arrêt alimentation
0	1	1	Alimentation sauvée
1	0	0	Réservé
1	0	1	Réservé
1	1	0	En pause
1	1	1	En pause étendue

Tableau 1-4 : Etat des bits SM (2. 1. 0) du registre MCUCR [5].

ICS01	ICS00	Mode de déclenchement	ICS11	ICS10
0	0	Sur niveau bas	0	0
0	1	Sur changement de niveau (0/1 OU 1/0)	0	1
1	0	Sur front descendant	1	0
1	1	Sur front montant	1	1

Tableau 1-5 : Etats des bits ICS (01. 00. 11. 10) du registre MCUCR [5].

SM (2.1.0) : (sleep mode) Choix du mode sommeil (arrêt ou ralentie). Les modes de sommeil ne seront pas étudiés mais nous présentons les modes possibles dans le tableau qui suit :

ICS11, ICS10 : (Interrupt Control Sense), définition de prise en compte de l'interruption externe INT1 et ICS01, ICS00 pour l'interruption externe INT0.

- Registre d’état SREG

Le registre SREG sert principalement pour les opérations arithmétiques. Il est le reflet des états des manipulations mathématiques et permet d'effectuer des tests afin d'effectuer des éventuels branchements au programme.

Adresse	7	6	5	4	3	2	1	0
$3F	I	T	H	S	V	N	Z	C
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E

Figure 1-6 : Les bits du registre SREG [5].

I : (Global Interrupt Enable) sert à activer (1) ou interdire (0) les sources d'interruptions.

T : (Copy Storage) ce bit tampon de manipulation les instructions BLD et BST.

H : (Half Cary) signale qu'une demi-retenue lors d'une instruction arithmétique.

S : (Sign bit) bit de signe résultant d'un OU exclusif avec le bit N et V.

V :(Overflow bit) bit de dépassement de capacité des opérations arithmétique et logique.

N :(Negative bit) le résultat de la dernière manipulation arithmétique est négatif.

Z : (Zéro bit) le résultat de la dernière manipulation arithmétique est égal à zéro.

C : (Cary bit) l'opération arithmétique à donner lieu à une retenue.

- Registre de pointeur de pile

Le registre de pointeur de pile est utilisé par les instructions PUSH et POP de gestion de pile. La gestion de pile utilise les deux registres 8 bits qui suivent pour former une adresse 16 bit.

Adresse	15	14	13	12	11	10	9	8
$3D	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
Adresse	7	6	5	4	3	2	1	0
$3E	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0

Tableau 1-7 : Les bits du registre pointeur de pile [5].

L'adresse ainsi crée doit être positionnée en général en fin de la mémoire SRAM. La pile est décrémenté avec l'instruction PUSH et incrémenté avec POP, donc le positionnement en fin de SRAM ne pose aucun problème de taille en général, mais attention aux boucles trop grande avec des PUSH qui pourrait arriver dans l'espace de stockage des variables programmes ou système.

Registre MCUSR

Ce registre permet de connaître le statut du microcontrôleur.

JTD : (JTAG interface disable), quand ce bit est à 0, on permet l'interface JTAG.

JTAG = interface de programmation des Atmegas.

ISC2 : (interrupt sense control), il est activé par la broche externe INT2 si le registre SREG est activé pour les interruptions.

Adresse	7	6	5	4	3	2	1	0
$34	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E

Tableau 1-8 : Les bits de registre MCUSR [5].

JTRF : (JTAG Reset Flag), ce bit est mis à 1 si un Reset est causé par la logique JTAG du Registre reset choisi par l'instruction JTAG AVR_RESET. Ce bit est remis à 1 lors de la mise sous tension, ou en écrivant un 0 dedans.

WDRF : (Watchdog Reset Flag), est mis à 1 pour activer le chien de garde. Le bit est remis à 0 par un reset ou par l'écriture d'un 0.

BORF : (Brown-out Reset Flag), ce bit est mis à 1 si une Panne d'électricité partielle arrive. Le bit est remis par un reset ou par l'écriture d'un 0.

EXTRF et **PORF** : (External Reset Flag & Power-on Reset Flag) Détermine la source d'un reset, le décodage peut se résumer ainsi :

Source du reset	EXTRF	PORF
WatchDog	0	0
Mise sous tension	0	1
Reset externe	1	0
Mise sous tension	1	1

Tableau 1-9 : Etat des bites EXTRF et PORF [5].

- Oscillateur externe

L'oscillateur externe à quartz est généralement utilisé dans les montages électroniques pour générer des fréquences allant de 4 à 20 MHz.

Ce quartz est connecté sur les broches XTAL1 et XTAL2 avec un amortisseur capacitif de l'ordre de 12 à 22 pF.

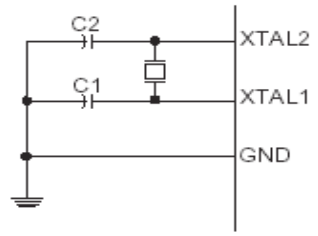


Figure 1-6 : Brochage de l'oscillateur à quartz [5].

- Oscillateur interne (Timer/Counter)

Pour les microcontrôleurs AVR avec les pins d'oscillateur de Timer/Counter (TOSC1 et TOSC2), le cristal est relie directement entre les pins. En programmant le fusible de CKOPT, l'utilisateur peut permettre les condensateurs internes sur XTAL1 et XTAL2 enlevant de ce fait le besoin de condensateur externe. L'oscillateur est optimisé pour l'usage d'un cristal de montre à 32.768 kHz. L'application d'une source extérieure d'horloge à TOSC1 n'est pas recommandée

L'oscillateur de Timer/Counter emploie le même type d'oscillateur en cristal que l'oscillateur de base fréquence interne. Les condensateurs ont la même valeur nominale de 36 pF [5].

Chapitre 2 :

Les moteurs pas à pas

1) Introduction

Bien que les premiers moteurs pas à pas remontent aux années 1930, Ceux-ci ne sont vraiment développés que vers 1970 grâce au développement conjugué de l'électronique de puissance et, surtout, grâce à l'apparition de l'électronique numérique à forte intégration.

Le moteur pas à pas assure le rôle d'interface entre le mode des systèmes numérique et leur environnement mécanique. Il permet de produire des déplacements incrémentaux, en boucle ouvert, avec un couple de positionnement à l'arrêt. Le moteur pas à pas est un convertisseur électromécanique ayant pour fonction la transformation d'une information électrique en une action mécanique pouvant être un déplacement linéaire ou plus classiquement angulaire. Ils sont très utilisés dans toutes les applications mécaniques ou l'on doit contrôler simplement la position ou la vitesse d'un système en boucle ouverte.

Contrairement aux moteurs à courant continu, ils ne nécessitent pas de boucle d'asservissement et sont plus simple à commander. Leur commande est relativement simple car on n'a pas besoin d'accessoires tels que des codeurs pour connaître la position car chaque impulsion du système de commande les fait avancer d'un pas.

La génération d'un nombre fini d'impulsion constitue une commande de position, alors que la fréquence de ces mêmes impulsions contrôle la vitesse de rotation.

Ces moteurs sont par exemple utilisés dans les imprimantes jet d'encre ou laser, pour positionner les têtes d'impression ou pour l'avancée du papier.

2) Moteur pas à pas

Il existe trois types de moteur pas à pas: les moteurs à aimants permanents, les moteurs à réluctance variable et les moteurs hybrides. Nous verrons plus loin que les moteurs à aimants permanents se subdivisent également en deux catégories. Malgré les différences qui existent entre les moteurs,

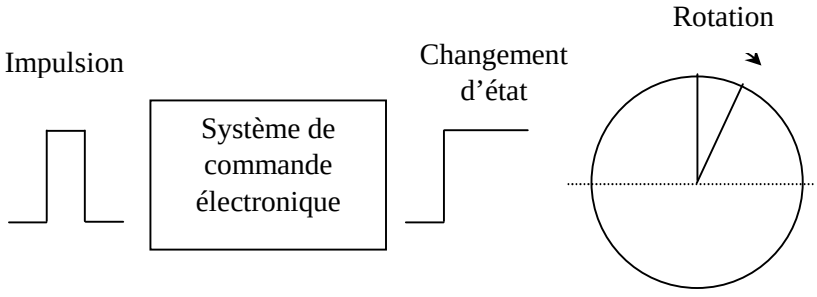


Figure 2-1 : Diagramme simplifié du principe de fonctionnement d'une rotation incrémental pour moteur [14].

le résultat recherché est l'avance d'un seul pas, c'est-à-dire la rotation de leur axe suivant un angle déterminé à chaque impulsion.

Cet angle, qui varie selon la constitution interne du moteur, est en général compris entre 0° et 90°. Les moteurs les plus couramment rencontrés présentent des pas de :

0, 9° : soit 400 pas par tour,
1, 8° : soit 200 pas par tour,
3, 6° : soit 100 pas par tour,
7, 5° : soit 48 pas par tour,
15° : soit 24 pas par tour [10].

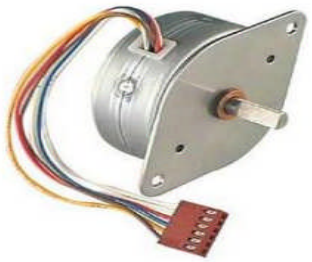


Figure 2-2 : Moteur pas à pas ayant six fils [15].

Il est évident que les moteurs pas à pas, de par leur technologie, présentent une très grande précision et une durée de vie quasi illimitée, l'usure mécanique étant pratiquement inexistante (absence de frottements).

Les moteurs pas à pas existent en différentes tailles qui varient de 1 cm à plus d'une dizaine de centimètres. Tout dépendra des applications dans lesquelles ils seront utilisés.

Le plus petit moteur, par exemple, sera destiné au déplacement des têtes de lecture dans les lecteurs de disquettes ou dans les disques durs où un couple très faible est requis. Par contre, le déplacement d'un bras de robot demandera un couple nettement plus important, donc un moteur de diamètre élevé. Signalons que le couple est exprimé le plus souvent en kilogrammes par centimètre (Kg/cm), ce qui définit le poids en kilogrammes que pourra soulever l'axe d'un moteur pourvu d'un bras de longueur exprimée en centimètres.

La valeur de la tension d'alimentation varie dans de grandes proportions, elle peut aller de 3V à plusieurs dizaines de volts. De même, selon la résistance ohmique de leurs bobinages, le courant consommé s'étendra dans une gamme allant de quelques dizaines de milliampères à plusieurs ampères. Alors on peut dire que plus le courant sera élevé, plus le couple sera important.

En ce qui nous concerne, les détaillants commercialisent la plupart du temps, sauf exception rare, des moteurs de type standard, de consommation moyenne et possédant un couple de puissance relativement bas. Ce faible couple pourra être compensé par l'adjonction d'une démultiplication constituée de pignons. La vitesse de rotation diminuera mais le couple augmentera dans de fortes proportions, ce qui constituera une solution intermédiaire étant donné le prix prohibitif des moteurs pas à pas de forte puissance.

Nous allons maintenant aborder plus en détail ce qui différencie chaque type de moteur, leur technologie et les principaux types de commande [9].

3) Caractéristiques des moteurs pas à pas

Il y a deux sortes de caractéristiques : les caractéristiques mécaniques qui dépendent essentiellement du moteur, et les caractéristiques électromagnétiques et électromécaniques qui dépendent du circuit de commande. Les constructeurs ont l'habitude de fournir ces dernières pour des circuits-types, à partir desquels on doit extrapoler.

Les Caractéristiques mécaniques, électriques et électromécaniques sont :

a) Taille : C'est la dimension mécanique du moteur. Elle consiste en un diamètre (ou une cote sur plat pour les moteurs carrés) et une longueur. Pour les moteurs hybrides, les cotes sont souvent données en pouces, ce qui fait qu'un moteur de 2, 3 pouces de diamètre et de 2 pouces de longueur est désigné par "taille 23 longueur 2 pouces".

b) Nombre de pas par tour : Il correspond au nombre de systèmes de pôles. Il est donné pour une commande dite "en plein pas" et vaut 4 fois le nombre de systèmes de pôles pour les moteurs diphasés.

c) Précision du pas : C'est la tolérance non-cumulative de la position des pas par rapport à leur place théorique. Elle est donnée en % de l'intervalle angulaire entre deux pas.

d) Inertie du rotor : C'est le moment d'inertie donné en g. cm2.

e) Couple résiduel : C'est le couple qu'il faut fournir au moteur non alimenté pour vaincre l'attraction magnétique rotor-stator. Les fabricants y incluent aussi le frottement des paliers.

f) Résistance et inductance de phase : C'est la valeur ohmique de chaque enroulement et l'inductance de chaque enroulement non couplé aux autres (tous les autres en circuit ouvert), respectivement.

g) Courant de phase : C'est l'intensité nominale moyenne par phase. Elle est limitée par des considérations d'échauffement et par la saturation du circuit magnétique. Ces deux limitations agissent différemment selon le mode de commande. Cette valeur ne constitue pas la limite supérieure mais plutôt correspond à la performance optimum en service continu.

h) Couple de retenue : C'est le couple qu'il faut appliquer pour faire décrocher le moteur (le déplacer de façon permanente de sa position initiale). Cette valeur est généralement donnée pour un moteur à l'arrêt mais sous tension.

La constante du couple K_c est donnée par la relation :

$$K_c = \frac{C_r}{I_c} \quad (1)$$

où C_r est le couple de retenue publié par le constructeur et I_c est l'intensité composée respectivement. L'intensité composée est le module de la somme vectorielle des intensités des enroulements alimentés. Dans le cas de la commande en plein pas, elle vaut :

$$I_c = \sqrt{2}\ I_{ph} \quad (2)$$

I_{ph} étant l'intensité par phase.

Dans le cas de la commande en demi-pas, on a alternativement un puis deux enroulements alimentés. Dans ce cas, l'intensité composée varie un pas sur deux entre I_{ph} et $\sqrt{2}\ I_{ph}$ et le couple de retenue varie aussi dans la même proportion [19].

i) Couple dynamique : C'est la valeur du couple résistant qui provoque le décrochage en marche. Cette valeur diminue quand la vitesse augmente mais cette diminution est moindre pour une commande raffinée que pour un circuit simple. Voir les documentations données en exemple.

j) Puissance nominale : C'est la puissance qu'on peut tirer au régime de puissance maximum en alimentant avec un circuit simple donné en référence. Cette valeur peut être augmentée dans de grandes proportions selon le facteur de marche et la complexité du circuit de commande.

k) Force contre-électromotrice : Ce paramètre très important est rarement donné dans les documentations. Il a cependant une grande importance pour la conception des schémas de commande. Heureusement, il peut être facilement mesuré. Il peut aussi être estimé assez exactement comme suit.

La constante de vitesse est

$$K_v = \frac{E}{\omega} \quad (3)$$

Où E est la force contre-électromotrice et ω la vitesse angulaire de l'arbre du moteur. En MKSA, E est en Volts et ω en radian/s.

4) Type de moteur pas à pas

a) Le moteur à aimant permanent

Le rotor comporte des aimants permanents radiaux et le stator est constitué de bobinages. Lorsque chaque phase statorique est alimentée séquentiellement, le flux statorique réagit avec le flux rotorique des aimants ce qui tend à aligner les deux flux le long d'un axe de réluctance minimum (loi d'attraction des pôles de noms contraires). Le rotor se met en mouvement rotatif pas à pas pour suivre l'alimentation successive des phases statoriques. Ces moteurs sont assez efficaces, souples et délivrent un couple relativement important même lorsque le moteur n'est plus alimenté. Cependant, l'inertie du rotor est assez élevée ce qui limite leur gamme de vitesse. L'alimentation du bobinage A va créer un NORD en 1 et un SUD en 3, donc une rotation d'un "pas" de 90° dans le sens trigonométrique.

En alimentant successivement A, B, C, D on obtient une rotation pas par pas de 90° dans le sens trigonométrique. En alimentant C, B, A, D, la rotation est inverse. Si l'on maintient l'alimentation même réduite d'un bobinage, le rotor est maintenu à l'arrêt par un couple puissant.

Grâce à l'aimant, même si l'on coupe l'alimentation de toutes les bobines, le rotor reste dans sa position avec un couple de maintien plus faible mais non négligeable [17].

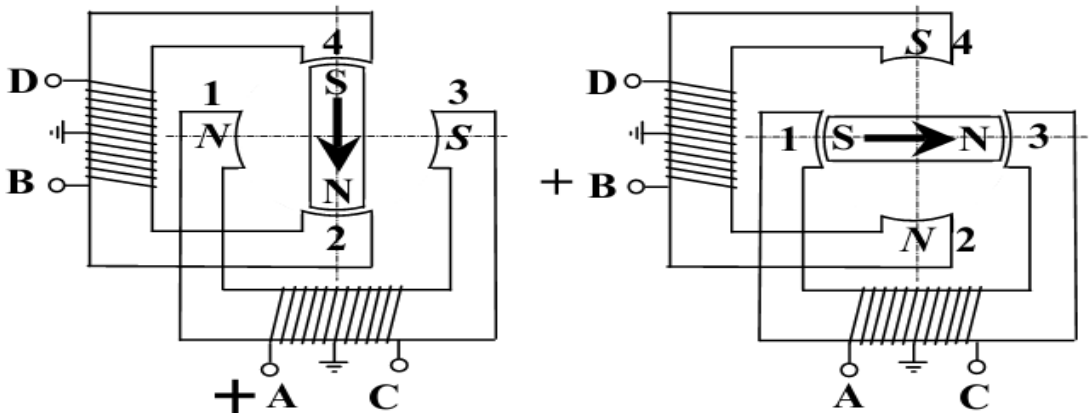


Figure 2-3 : Alimentation des bobines du moteur [17].

b) Les moteurs à aimant permanent bipolaires et unipolaires

b-1) Les moteurs à aimant permanent bipolaires

Ils ont en général 4 fils et se commandent en inversant le sens du courant dans une des deux bobines, qui sont alimentés une fois dans un sens, une fois dans l'autre sens. Ils créent une fois un pôle nord, une fois un pôle sud d'où le nom de bipolaire [18].

Chaque bobine ne possède que deux fils. Cela complique un peu l'électronique de commande mais permet de renforcer le couple du moteur (ou de réduire son encombrement pour un même couple par rapport à un moteur unipolaire).

La commande du moteur s'effectue par envoie successif, une à une, des combinaisons suivantes :

Séquence	I1>0	I1<0	I2>0	I2<0	Angle
1	1	0	0	0	0°
2	0	0	1	0	90°
3	0	1	0	0	180°
4	0	0	0	1	270°

Tableau 2-1 : Séquence d'alimentation de l'enroulement 1 du moteur bipolaire [18].

Séquence	I1>0	I1<0	I2>0	I2<0	Angle
1	1	0	1	0	45°
2	0	1	1	0	135°
3	0	1	0	0	225°
4	1	0	0	1	315°

Tableau 2-2 : Séquence d'alimentation de l'enroulement 1 et 2 du moteur bipolaire [18].

En mode 1, on alimente 1 enroulement à la fois et en mode 2, on alimente 2 enroulements à la fois.

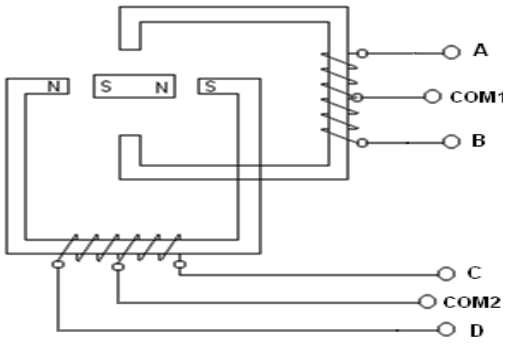


Figure 2-5 : Alimentation d'un moteur unipolaire [18].

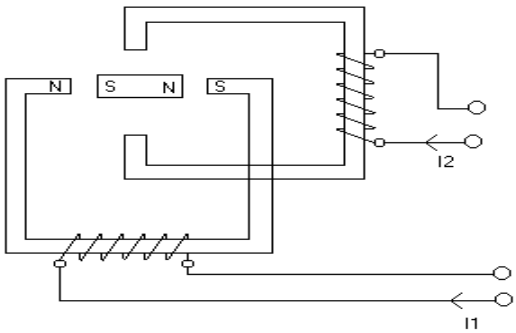


Figure 2-4 : Alimentation d'un moteur bipolaire [18].

b-2) Les moteurs à aimant permanent unipolaires

Ils ont en général six fils, dont deux sont reliés au milieu des bobines. Ils se commandent en les alimentant tour à tour,

Les bobinages d'un moteur unipolaire sont alimentés toujours dans le même sens par une tension unique d'où le nom d'unipolaire. Il possède un point milieu qui est une connexion centrale sur chaque enroulement. Généralement, on relie ensemble les points milieu de chaque bobine. Cela permet une simplification de la commande des moteurs [18].

La séquence d'alimentation des enroulements est suivant le tableau suivant.

Séquence	A	B	C	D	Angle
1	1	0	0	0	0^o
2	0	0	1	0	90^o
3	0	1	0	0	180^o
4	0	0	0	1	270^o

Tableau 2-3 : Séquence d'alimentation les enroulements unipolaires [18].

c) Les modes de commande d'un moteur pas à pas à aimant permanent

c-1) Fonctionnement en pas entier

Dans ce mode de fonctionnement, les bobines sont alimentées l'une après l'autre dans un sens puis dans l'autre. L'aimant permanent suit le déplacement du champ magnétique créé par ces bobines et s'oriente selon une de ses quatre positions stables. Comme le rotor est aimanté, lorsque le moteur n'est pas alimenté le flux magnétique dû à l'aimant permanent va à lui seul créer un couple résiduel ou couple de détente, en se mettant dans l'axe de l'une des bobines.

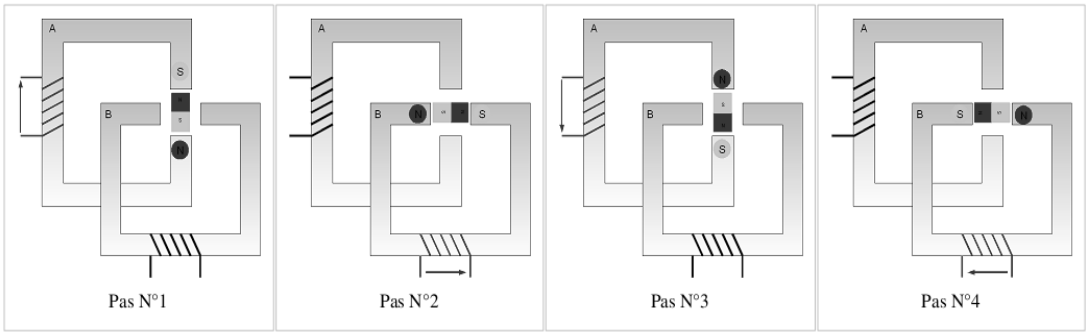


Figure 2-6 : Séquence d'alimentation des enroulements en mode pas entier [11].

c-2) Fonctionnement en mode fort couple "High Torque" (fort couple)

Pour augmenter l'intensité du flux magnétique créé par le stator, et donc le couple moteur, on peut alimenter les deux bobines en même temps, en faisant varier uniquement le sens du courant dans chacune d'entre elles. Le rotor prendra donc également l'une des quatre positions de la vidéo suivante, suivant le sens d'alimentation de chacune des bobines.

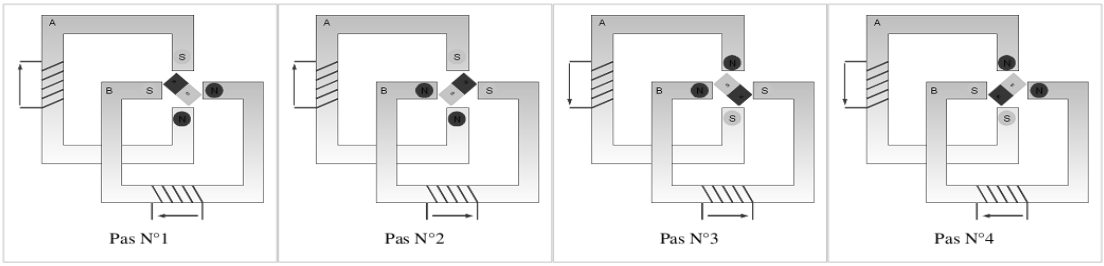


Figure 2-7 : La séquence d'alimentation des enroulements en mode 'couple fort [11].

c-3) Fonctionnement en mode demi-pas :

Pour augmenter le nombre de positions stables et donc de pas du moteur à aimant permanent, on peut combiner les deux modes précédents dans un mode de commande appelé « demi-pas ».

Les moteurs pas à pas à aimant permanent ont un couple moteur important, mais une résolution (nombre de pas par tour) faible, et une fréquence de rotation faible. La commande de ces moteurs pas à pas nécessite de contrôler le sens du courant dans chaque bobine

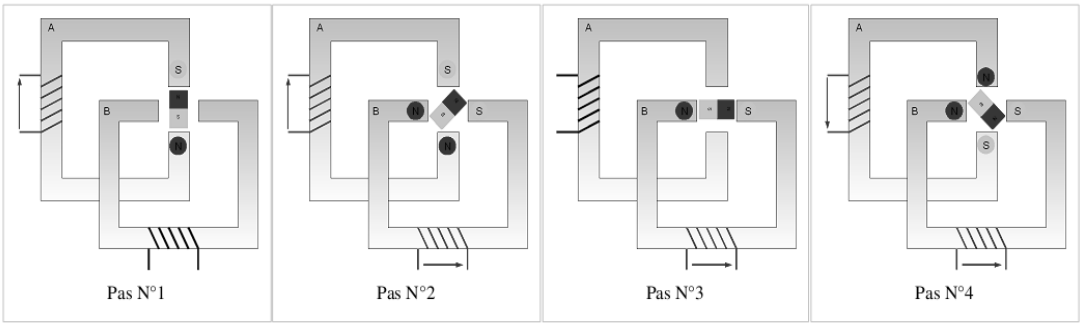


Figure 2-8 : La séquence d'alimentation les enroulements en mode demi-pas [11].

d) Caractéristiques

- Faible résolution : nombre de pas/tour peu important ;
- Couple d'utilisation plus élevé par rapport au moteur à reluctance variable ;
- Présence d'un couple résiduel lorsque le moteur est hors tension [12].

Avantages du moteur à aimant permanent :

- Bon marché
- Dimensions réduites
- Bon rendement
- Bon amortissement des oscillations
- Grand angle de pas (nombre de pas faible : 48)

Inconvénients du moteur à aimant permanent :

- Puissance faible
- Paliers en bronze ou plastique (pas de roulement)
- Couple résiduel sans courant
- Vitesse faible [14].

e) Moteur à réluctance variable

Le moteur à réluctance variable diffère d'un moteur à aimant permanent par son rotor. Le rotor d'un moteur à réluctance variable est en matériau magnétique non aimanté. Le stator est également un bobinage à plusieurs phases, mais ici le rotor est un cylindre métallique garni de dents dont le nombre est différent de celles du stator.

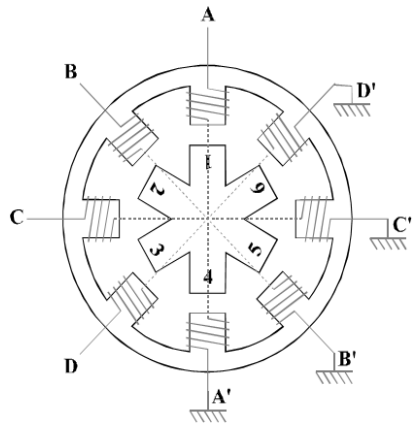


Figure 2-9 : Moteur à réluctance variable [12].

Lorsqu'une phase est alimentée, la loi de moindre réluctance tend à aligner la dent rotorique la plus proche sur la dent statorique dont le bobinage est alimenté.

Stator 3 phases et rotor 4 dents, pas de 30° lorsque l'enroulement 2 est alimenté, la dent D qui est la plus proche va s'aligner (rotation dans le sens trigonométrique) de manière à offrir une moindre réluctance. La rotation sera d'un angle de 30°.

Si le rotor ne conserve pas d'aimantation rémanente, il n'y a pas de couple lorsque le moteur n'est plus alimenté. La faible inertie possible du rotor permet à ces moteurs de travailler à grande vitesse aux faibles charges.

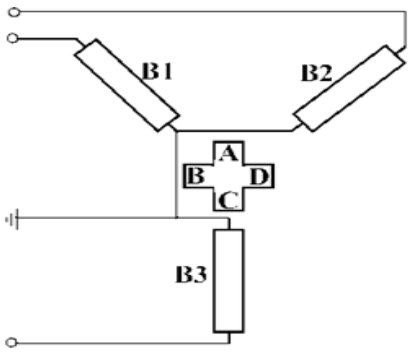


Figure 2-10 : Alimentation de moteur réluctance variable [17].

Ces moteurs sont sensibles aux oscillations et résonances dues au manque de souplesse [17].

Dans l'exemple présenté ci-dessous, on détermine le nombre de dents Z_S du stator, le nombre de dents Z_R du rotor, le nombre de pas N_p dans un tour et la valeur du pas sachant que :

$$N_p = \frac{Z_S \times Z_R}{|Z_S - Z_R|}$$

Caractéristiques :

- Grand nombre de pas par tours (bonne résolution)
- Construction simple mais délicate
- Couple développé faible
- Absence de couple résiduel avec le moteur hors tension

Notion de réluctance ℜ

Les milieux ferromagnétiques sont des conducteurs de flux: par exemple, dans un transformateur, le flux créé par l'enroulement primaire est conduit par le milieu magnétique vers l'enroulement secondaire.

La réluctance ℜ d'un milieu quelconque traduit le fait que ce milieu s'oppose plus ou moins à la conduction du flux magnétique. Les matériaux ferromagnétiques étant de bons conducteurs de flux ont une réluctance ℜ faible. L'air, mauvais conducteur de flux, a une réluctance élevée (entrefer des moteurs).

Règle du flux maximal

Quand un circuit peut se déplacer librement, la position d'équilibre stable que prend ce circuit est telle que le flux magnétique à travers ce circuit est maximal [16].

f) Moteurs hybrides : Le rotor est constitué par deux pièces en fer doux ayant chacune n pôles séparés par un aimant permanent magnétisé dans le sens de l'axe du rotor. Le nombre m de pôles du stator est différent de celui du rotor. Le rotor se déplace pour que le flux qui le traverse soit maximum. En mode pas entier, les bobines sont alimentées paire par paire alternativement avec inversion à chaque pas. Il est nécessaire d'avoir un rotor polarisé pour imposer le sens de rotation à chaque commutation.

Pour le modèle présenté, (stator avec deux paires de bobines et rotor avec rotor à deux fois trois pôles) à chaque pas, la direction du champ induit par le stator tourne de 30° en mode "pas entier" et de 15° en mode "demi-pas".

5) Comparaison des différents types de moteurs pas à pas

- Les moteurs à aimant permanent

Ils ont un couple élevé en raison de l'incorporation d'un aimant sur le rotor.

Étape angles disponibles sont grandes.

La taille est conditionnée par le rotor à aimant permanent et le terrain, vous pouvez obtenir de petites tailles.

Parce que l'aimantation du rotor, ce moteur un couple résiduel ou un couple de freinage.

L'effet d'amortissement généré par l'aimantation des limites de la plage de vitesses.

- Les moteurs à réluctance variable

Ne comprenant pas un aimant permanent, le rotor peut être fabriqué avec un petit diamètre, et donc la taille du moteur est également réduite.

Avec un petit diamètre de rotor, le moment d'inertie de cette offre également une plage dynamique élevée (haute vitesse), et un couple de départ.

L'angle de pas est limité, le diamètre de pôles de stator limite le nombre de bobines qu'il peut contenir.

Il n'a pas de couple participation résiduelle quand il est éteint, ce qui est souvent un inconvénient.

Ils sont beaucoup moins de couple, mais en la rendant plus facile.

- Les moteurs hybrides

Possibilité d'obtenir des angles de petit pas, sans l'aide d'un grand nombre de phases. Ils ont une bonne gamme dynamique que les moteurs à aimants permanents ou à réluctance variable.

Ces mesures sont contenues sans atteindre le moteur VR en raison de l'emplacement de l'aimant permanent dans le stator et produit un couple élevé produit par des moteurs à réluctance vigueur que VR.

Le Tableau suivant donne une comparaison entre les divers types de moteurs pas à pas.

Type de moteur pas à pas	Moteur à réluctance variable	Moteur à aimants permanents	Moteur hybride
Résolution (nbre de pas par tour)	Bonne	Assez peu élevée 100pas/tr	La plus élevée 400pas/tr
Couple moteur	Faible	Elève	Le plus élevée
Sens de rotation	Il dépend de l'ordre d'alimentation des phases	Il dépend de l'ordre d'alimentation des phases du sens de courant dans les bobines	Il dépend de l'ordre d'alimentation des phases du sens de courant dans les bobines
Fréquence de travail	grande	Fiable	Grande
Puissance	Quelques Watts	Quelques dizaines de Watts	Quelques kWatts

Tableau 2-4 : Comparaison entre les différant types de moteurs pas à pas.

Chapitre 3 :

Environnement de programmation

Pour garder la fluidité et l'organisation du texte, on a préféré de diviser ce chapitre en deux parties (I et II).

Partie I : La programmation en langage C

1) Introduction

Le C est un langage très populaire. Il est l'un des langages les plus connus et les plus utilisés qui existent, utilisé pour programmer une grande partie des logiciels que nous connaissons.

A l'époque où les ordinateurs pesaient des tonnes et faisaient la taille d'une maison, on a commencé à inventer un langage de programmation appelé l'Algol. En évoluant ensuite, on a créé un nouveau langage appelé le CPL, qui évolua lui-même en BCPL, qui prit ensuite le nom de langage B.

Puis en 1990, on en est arrivé à créer et normaliser un autre langage encore, appelé le langage C. Ce dernier, s'il a subi quelques modifications, reste encore un des plus utilisés aujourd'hui. Un peu plus tard en 1999, on a proposé de rajouter des améliorations au langage C, qui a été normalisé par ISO ; nommé C99. Le langage C n'est pas un vieux langage, au contraire, il est encore très utilisé aujourd'hui. Il est à la base des plus grands systèmes d'exploitation tels Unix (Linux et Mac OS) ou Windows [26].

2) Le langage de programmation

Un langage de programmation est, d'un point de vue mathématique, un langage formel, c'est-à-dire un ensemble de suites (appelés mots) de symboles choisis dans un ensemble donné (appelé alphabet) qui vérifient certaines contraintes spécifiques au langage (syntaxe).

Dans le cas des langages de programmation, on trouve dans l'alphabet plusieurs symboles :

– des mots-clés. Ex : main, int, if, for . . .

– des opérateurs. Ex : =, <, & . . . ,

– des chiffres et des lettres permettant d'identifier des variables ou est constantes,

– des caractères spéciaux : Ex : accolades, crochets, point-virgule, tiret bas Un programme est un mot du langage, c'est-à-dire une suite de symboles vérifiants les contraintes dictées par la syntaxe du langage.

Comme le but d'un langage est quand même de se faire comprendre de la machine, on a plusieurs possibilités :

- Soit on utilise un langage de bas niveau : on écrit un programme directement dans le langage machine (ou langage natif) compréhensible par le processeur, mais rarement intelligible puisque rares sont les personnes qui parlent le binaire couramment.

- Soit on utilise un langage de haut niveau : on écrit un programme dans un langage inintelligible pour le processeur, qui sera ensuite "traduit" en langage machine afin que le processeur l'exécute.

Ces langages permettent de décrire des taches sans se soucier des détails sur la manière dont la machine l'exécute. Deux stratégies sont utilisées pour les langages de haut niveau. La différence réside dans la manière dont on traduit le programme, qui est à l'origine dans un ou plusieurs fichiers texte appelés fichiers source ou code source.

- Soit on écrit un programme dans un langage interprété. Le code source sera traduit mot à mot ou instruction par instructions dans le langage machine propre au processeur par un programme auxiliaire : l'interpréteur. L'humain fait une requête, cette requête est traduite par l'interpréteur, qui détermine la stratégie d'exécution, le processeur l'exécute puis passe à la requête suivante. Chaque exécution du programme nécessite l'utilisation de l'interpréteur.

- Soit on écrit un programme dans un langage compilé. Le code source sera traduit (une bonne fois pour toute) dans le langage machine propre au processeur par un programme compilateur.

Contrairement aux interpréteurs, les compilateurs lisent entièrement le code source avant de le traduire, détermine la stratégie d'exécution de toutes les instructions, puis génère un fichier binaire exécutable. Le fichier produit n'a plus besoin d'être lancé par un autre programme. Il est autonome. Cela permet de garantir la sécurité du code source [25].

3) Compilation d'un programme

Le C est un langage compilé. Le fichier source, écrit dans un fichier texte doit être traduit dans le langage machine. Le C fait partie des langages faisant appel à un compilateur.

Un compilateur est un programme qui traduit une instance d'un langage source, dans un autre langage, appelé le langage cible, en préservant la signification du texte source. Ce principe général décrit un grand nombre de programmes déférents et ce que l'on entend par signification du texte source dépend du rôle du compilateur.

Lorsqu'on parle de compilateur, on suppose aussi en général que le langage source est, pour l'application envisagée, de plus haut niveau que le langage cible, c'est-à-dire qu'il présente un niveau d'abstraction supérieur. En pratique, un compilateur sert le plus souvent à traduire un code source écrit dans un langage de programmation en un autre langage, habituellement un langage d'assemblage ou un langage machine.

Un compilateur fonctionne par analyse-synthèse, c'est-à-dire qu'au lieu de remplacer chaque construction du langage source par une suite équivalente de constructions du langage cible, il commence par analyser le texte source pour en construire une représentation intermédiaire appelée code objet à partir duquel il construit le code exécutable (instance autonome du langage machine) ou un programme équivalent au code source écrit dans le langage cible, si celui-ci n'est pas le langage machine.

Il est donc naturel de séparer « au moins conceptuellement, mais aussi en pratique » le compilateur en une partie avant (ou frontale), parfois appelée "souche", qui lit le texte source et produit la représentation intermédiaire, et une partie arrière (ou finale), qui parcourt cette représentation pour produire le texte cible. Dans un compilateur idéal, la partie avant est indépendante du langage cible, tandis que la partie arrière est indépendante du langage source. Certains compilateurs effectuent de plus sur la forme intermédiaire des traitements substantiels, que l'on peut regrouper en une partie centrale, indépendante à la fois du langage source et de la machine cible. On peut ainsi écrire des compilateurs pour toute une gamme de langages et d'architectures en partageant la partie centrale, à laquelle on attache une partie avant par langage et une partie arrière par architecture.

Voici les quatre phases de la compilation :

Phase 1 - Le traitement par le préprocesseur :

Le fichier source est analysé par le préprocesseur qui effectue les transformations purement textuelles : le remplacement des chaînes de caractères, l'inclusion d'autre fichiers sources, la suppression des commentaires.

Phase 2 - La compilation :

La compilation proprement dite traduit le code source en assembleur, c'est-à-dire une suite d'instructions qui utilise des mnémoniques : des mots courts correspondants un à un à des octets. Trois phases d'analyse sont effectuées avant la traduction proprement dite : le découpage du programme en lexèmes (analyse lexicale), la vérification de la correction de la syntaxe du programme (analyse syntaxique), l'analyse des structures de données (analyse sémantique).

Phase 3 - L'assemblage :

Cette opération traduit le code assembleur en fichier binaire, compréhensible directement par le processeur. Cette étape est en général faite directement après la compilation. Le fichier obtenu est appelé fichier objet. Il contient également les informations nécessaires à l'étape suivante.

Phase 4 - L'édition de liens :

Un programme est souvent séparé dans plusieurs fichiers sources pour des raisons de clarté mais aussi parce que l'on fait souvent appel à des bibliothèques (Library, en anglais. . .) de fonctions standards déjà écrites. Une fois le code source assemblée, il faut donc lier entre eux tous les fichiers objets créés. L'édition de ces liens produit alors le fichier exécutable.

Ces déférentes étapes expliquent que les compilateurs fassent toujours l'objet de recherches, particulièrement pour des questions d'optimisation [25].

4) Caractéristique du langage C

Il existe de nombreux langages de programmation de haut niveau comme le C, le Pascal, ou le Basic. Ils sont tous excellents et conviennent pour la plupart des tâches de programmation. Toutefois, les professionnels placent le langage C en tête de liste pour plusieurs raisons :

Il est souple et puissant. Le langage C est utilisé pour des projets aussi variés que des systèmes d'exploitation, des traitements de textes, des graphiques, des tableurs ou même des compilateurs pour d'autres langages.

Lorsqu'une nouvelle architecture (nouveau processeur, nouveau système d'exploitation, etc…) apparaît, le premier langage disponible est généralement le C car contrairement à d'autres, il est facile à porter. De plus, un compilateur C est souvent disponible sur les ordinateurs, ce qui n'est pas le cas pour les autres langages.

Avec la norme ANSI, le C est devenu un langage portable. Cela signifie qu'un programme C écrit pour un type d'ordinateur (un PC IBM, par exemple) peut être compilé pour tourner sur un autre système (comme un DEC VAX) avec très peu ou aucune modification.

Le langage C contient peu de mots. Une poignée d'expressions appelées mots clés servent de bases pour l'élaboration des fonctions. On pourrait penser, à tort, qu'un langage possédant plus de mots clés (quelquefois appelés mots réservés) pourrait être plus puissant. Lorsque vous programmerez avec ce langage, vous vous apercevrez que vous pouvez réaliser n'importe quelle tâche.

Le langage C est modulaire. Son code peut (et devrait) être écrit sous forme de sous-programmes appelés fonctions. Ces fonctions peuvent être réutilisées pour d'autres applications ou programmes. Si vous passez des informations à ces fonctions, vous obtenez du code réutilisable.

Comme vous pouvez le constater, le choix du C en tant que premier langage de programmation est excellent. Vous avez certainement entendu parler de C++. Ce langage s'appuie sur une technique de programmation appelée programmation orientée objet.

C++ était initialement une version améliorée du C, à savoir un C disposant de fonctions supplémentaires pour la programmation orientée objet. Le C++ est aujourd'hui un langage à part entière. Si vous êtes amenés à étudier ce langage, ce que vous aurez appris du C vous aidera grandement.

Un autre langage, également basé sur C, a été l'objet d'une attention toute particulière. Il s'agit de Java. Il existe de nombreuses similitudes entre ces deux langages [26].

5) Comparaison avec d'autres langages

- L'Assembleur (ASM)

Inventé en 1945 par John Von Neumann, l'assembleur fut le premier vrai langage de programmation mais aussi le plus proche du langage machine (il s'adresse directement au processeur) et de ce fait, l'un des plus compliqués. Les instructions sont de type MOV, ADD, PUSH, POP, INT, bref rarement plus de 5 lettres, ce qui les rend difficile à mémoriser ; il faut aussi bien comprendre le fonctionnement de la pile et des différents registres du processeur (prise de tête garantie). Il n'y a pas d'instructions toutes faites : il faut d'abord mettre les bonnes données dans les bons registres avant d'appeler les interruptions adéquates. De plus, il existe un

langage assembleur par famille de processeur. Il présente néanmoins l'avantage de donner des exécutables de taille réduite [25].

- Le BASIC

Le BASIC (Beginners All purpose Simple Intructions Code) est apparu en 1964. Sans doute le langage de programmation le plus simple au monde. Il permet de créer des programmes très facile et constitue ainsi une initiation à la programmation, surtout que c'est bien le seul usage qu'on pourrait encore lui trouver. En effet, le BASIC est loin d'être un langage "puissant" [26].

- Le JAVA

Tirant ses origines du début des années 90, né officiellement en 1995, ce langage présente l'avantage d'être portable ; on peut exécuter un programme JAVA sous Windows, Mac, Linux, tout. Cette portabilité est dû à une particularité de l'implémentation la plus répandue (comprenez : de Sun) du langage JAVA : celui-ci n'est pas compilé en code machine comme les autres langages mais dans un langage intermédiaire dit "ByteCode".

Il faut alors ce qu'on appelle une "machine virtuelle JAVA" (JVM ou J2RE pour les connaisseurs) pour l'exécuter. Seule cette machine virtuelle change selon les systèmes ; une fois qu'elle est installée sur la machine, on peut y exécuter tous les programmes JAVA.

On regrettera toutefois que le concept de machine virtuelle soit quasiment imposé (il existe des compilateurs qui compilent Java en langage machine mais ceux-ci ne permettent pas autant de chose que la JVM). En plus d'être souvent inutile, l'exécution dans la JVM plombe souvent les performances des programmes [26].

Partie II : Installation et configuration des logiciels de développement

1) AVR Studio

Pour réaliser la programmation de l'Atmega, nous avons utilisé le logiciel AVR Studio qui permet d'écrire des programmes informatiques en langage C et de programmer des microcontrôleurs. Et pour envoyer le ce programme nous avons utilisé le logiciel WinAVR qui contient la commande AVRdude.

Les logiciels d’Atmel que nous utilisons sont distribués gratuitement sous certaines conditions décrites dans les fichiers à télécharger sur Internet :

http://winavr.sourceforge.net/links.html: winAVR

http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725: AVR Studio et SP2.

a) Installer WinAVR:

a. Exécuter WinAVR-20081205-install.exe

b. Choisir la langue. Pour la même configuration qu'en laboratoire : Français. OK.

c. Suivant. Accepter les conditions du contrat de licence. J'accepte.

d. Choisir un dossier d'installation (C:\ winAVR-20081205). Suivant.

e. Choisir les composants par défaut. Installer.

f. Suivant. Terminer l'installation. Fermer.

b) Installer AVR Studio 4. 13:

a. Exécuter aStudio4b528.exe

b. Next. Accepter les conditions du contrat de licence. Next.

c. Choisir le répertoire d'installation (C:\Program Files\Atmel\AVR Tools). Next.

d. Sélectionner l'option USB. Next. // Pour ce qui utilise le programmateur STKxxx.

e. Démarrer l'installation. Install. Finish.

c) Installer le Service Pack AVR Studio 4. 13 SP2:

b. Exécuter AVR Studio4.13SP2.exe.

b. Sélectionner l'option USB. Next.

c. Terminer l'installation. Finish.

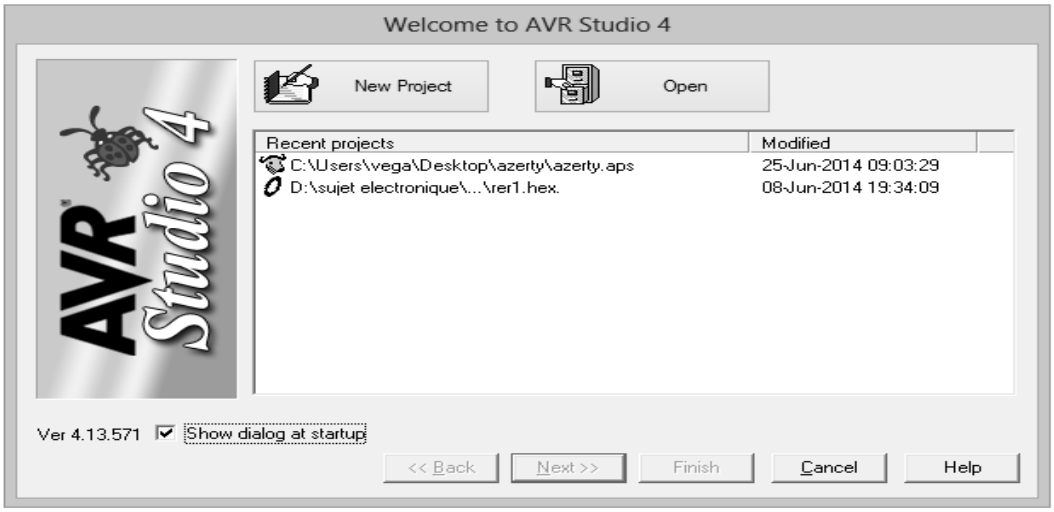


Figure 3-1 : Ecran de bienvenu.

Après l'installation de logiciel AVR Studio, on démarre AVR Studio 4 et on crée un nouveau projet. On clique sur New Project dans l'écran de bienvenu (Figure 3-1) ou alors Project->New Project depuis le menu.

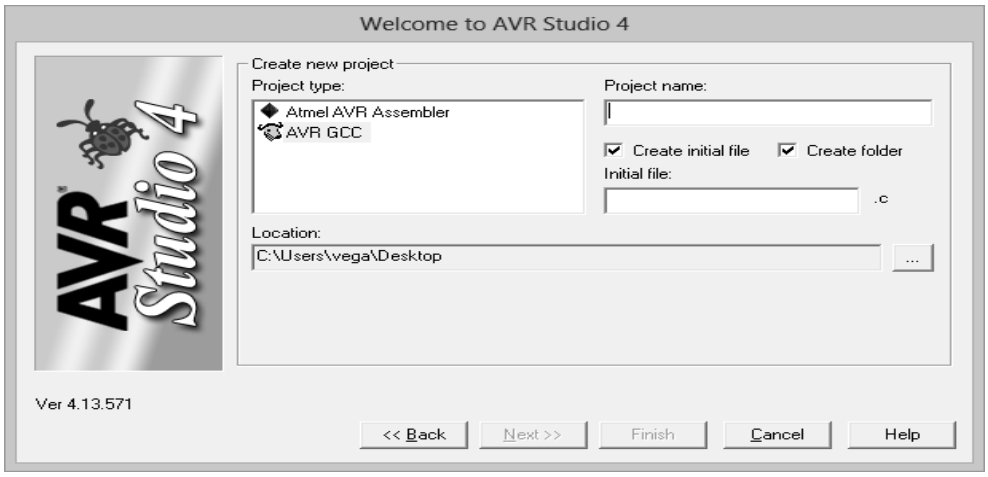


Figure 3-2 : Sélection du type de projet.

Dans la boîte de dialogue "New Project" (Figure 3-2), deux types de projets vous sont proposés.

• Atmel AVR assembler, qui comme son nom l'indique permet d'écrire des programmes en assembleur.

• AVR GCC, qui utilisera GCC pour compiler le code, ceci vous permettra d'écrire le code en C.

Sélectionner le projet AVR GCC (Figure 3-2), puis compléter le champ Project Name. Le nom du fichier initial est, par défaut, identique à celui du projet. Vous avez cependant la possibilité de le modifier, puis cliquer sur Next.

Une nouvelle boîte de dialogue apparaît, cette dernière vous permet de sélectionner le débuggeur ainsi que le microcontrôleur à utiliser.

Dans notre cas, nous utiliserons " AVR Simulator " et le système pour sélectionner le microcontrôleur dont on est besoin. Une fois la sélection faite, cliquer sur Finish.

La fenêtre de l'environnement de "travail" apparaît (Figure 3-4), à gauche se trouve l'arborescence des fichiers utilisés dans votre projet, alors que sur la droite sont présentés les différents éléments du microcontrôleur.

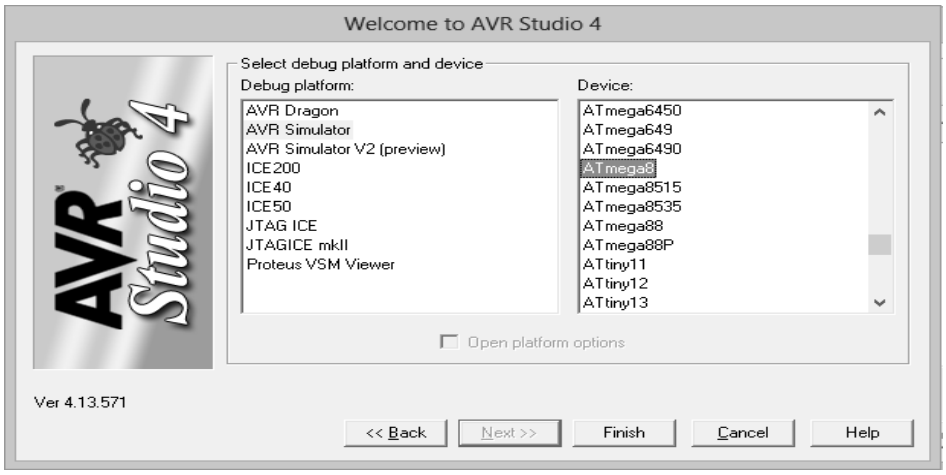


Figure 3-3 : Boite de dialogues pour sélectionner le modèle du microcontrôleur.

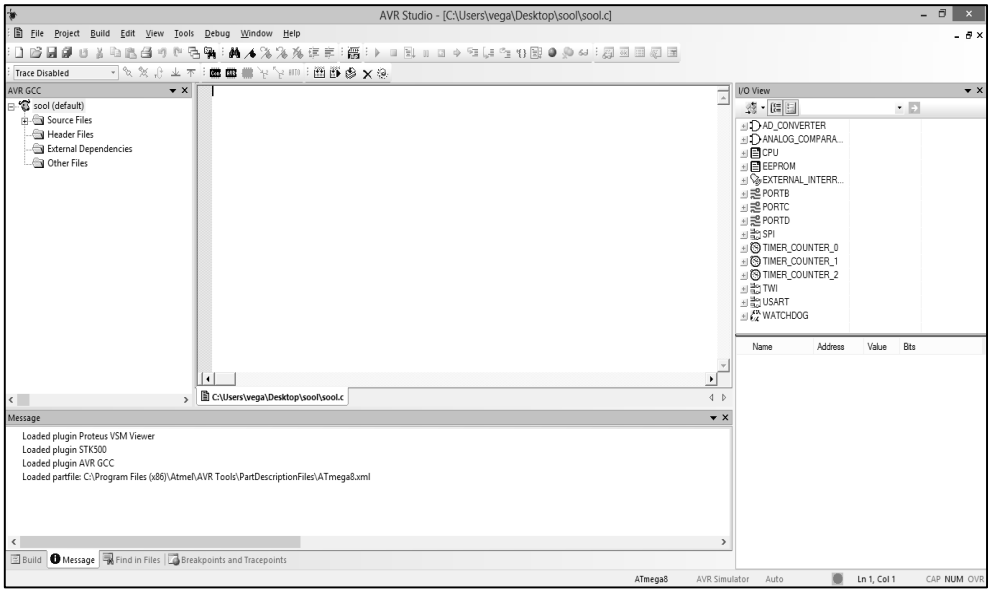


Figure 3-4 : Fenêtre principale de l'AVR Studio 4.

Après avoir écrit le programme en C dans le fichier principal, nous pouvons effectuer une compilation ou bien build du projet, depuis les menus Build -> Build ou en cliquant sur le bouton encerclé sur la (Figure 3-5) ou appuyez sur la touche F7 [22 & 23].

2) AVRdude

AVRdude est un paquet exécutable. Et il se charge de transférer toute l'information dans les processeurs AVR. Plusieurs programmateurs du matériel sont entreposés dans une base de données. Cette base de données peut être modifiée en ajoutant le nouveau matériel ou un processeur AVR si ce n'est pas déjà inscrit.

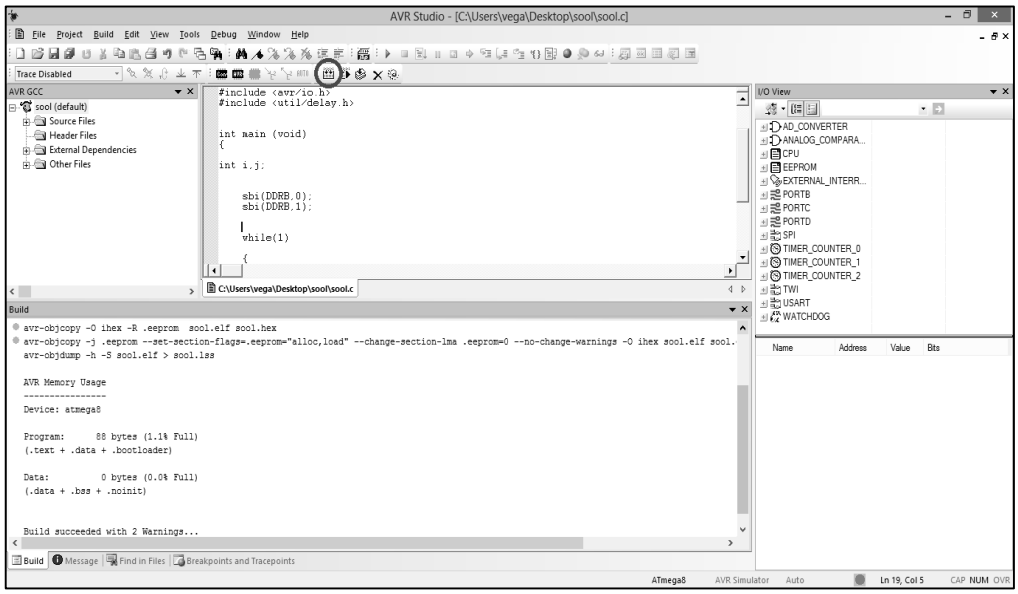


Figure 3-5 : Exemple d'un programme après compilation.

Après l'installation de logiciel AVR Studio 4, on programme et compile notre code pour le microprocesseur. L'exécutable doit être envoyé ou flashé sur le microcontrôleur. Il nous faut un programmateur et son logiciel. Le programmateur sera discuté plus bas. Le programme standard utilisé pour ce type de microcontrôleur est AVRdude qui d'ailleurs s'installe automatiquement lors de l'installation de AVR Studio. Ce programme se trouve dans le répertoire C:\WinAVR\bin de notre installation AVR Studio 4.

AVRdude est intégré dans l'environnement winAVR et toutes les opérations décrites ci-dessous sont automatisées en utilisant un script (voir l'exemple ci-bas).

On crée un script qu'on sauvegarde avec une extension « .BAT ». Le script utilisé pour nos programmes est :

```
cd . .
cd . .
cd C:\WinAVR\bin\
                rem  chercher le chemin du fichier binaire.
del C:\WinAVR\bin\rer1.hex
```

```
                rem  supprimer le fichier afin de le remplacer par un neuf
copy C:\Users\vega\Desktop\azerty\default\test1.hex C:\WinAVR\bin\rer1.hex
                rem  copier le fichier en hex pour remplacer par le fichier
                rem  supprimé avec la commande précédente
avrdude -p atmega8 -P com1 -c dasa -e -V -U flash:w:rer1.hex
                rem  flasher le atmega8 à travers les port com1
cmd
```

Dans cette commande on remplace l'ancien fichier par un nouveau fichier qu'AVR Studio crée à chaque nouvelle compilation. Ce fichier contient notre code en hexadécimale (programme) qui doit être envoyé au microcontrôleur.

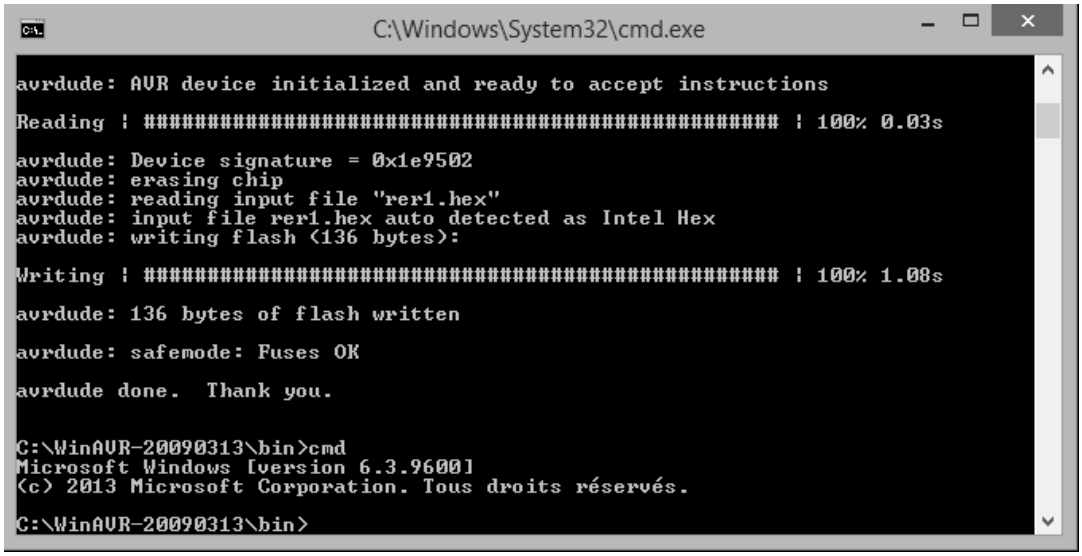


Figure 3-6 : Fenêtre 'cmd' après l'envoi du script de flashage.

Ecriture dans l'EEPROM

L'écriture dans l'EEPROM est rendue facile grâce à l'AVRdude, avec quelques commandes on peut écrire manuellement le programme à l'intérieure de la mémoire. Voici une commande qui permette de communiquer avec l'EEPROM :

avrdude -p ATMEGA8 –P com1 –c dasa –t

-p : indiquer le système AVR (choisir le microcontrôleur à programmer).

-P : indiquer la connexion port.

-c : indiquer le type de programmateur.

-t : Entrer le le mode terminal.

w : Ecrire dans l'EEPROM.

r : Lire le contenue de l'EEPROM.

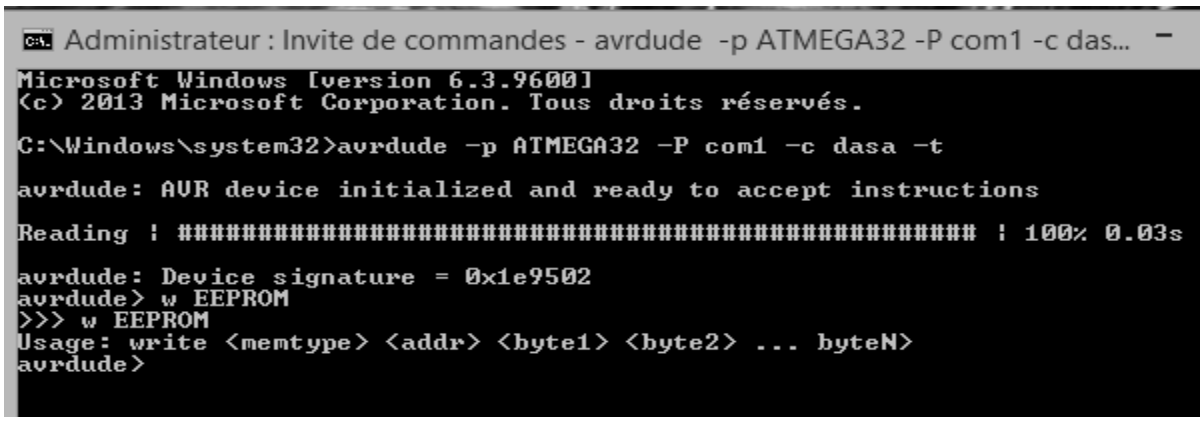


Figure 3-7 : Fenêtre interactive d'exécution de commande avec le programme AVRdude.

3) Programmateur

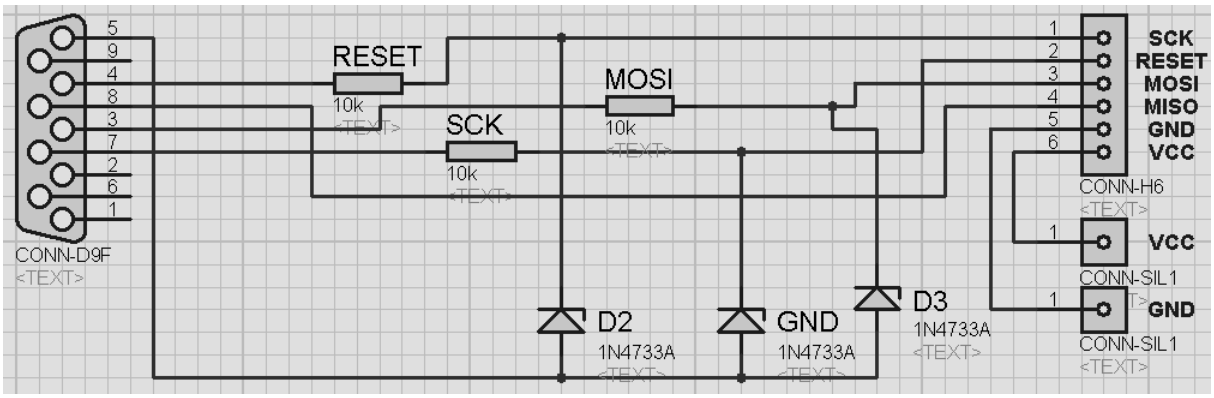


Figure 3-8 : Schéma du programmateur AVR dasa réalisé sous ISIS.

Nous avons besoin d'un circuit électronique pour pouvoir flasher le microcontrôleur, en faisant un lien entre l'ordinateur et le microcontrôleur et pour cela on utilise un programmateur avec le port série RS232.

a) Connexion des composants

A l'aide de la fenêtre SCHEMATIC de ARES qui est montré dans la (Figure 3-8), on réalise les connexions entre les composants électroniques qu'on a chargés à partir des bibliothèques ISIS.

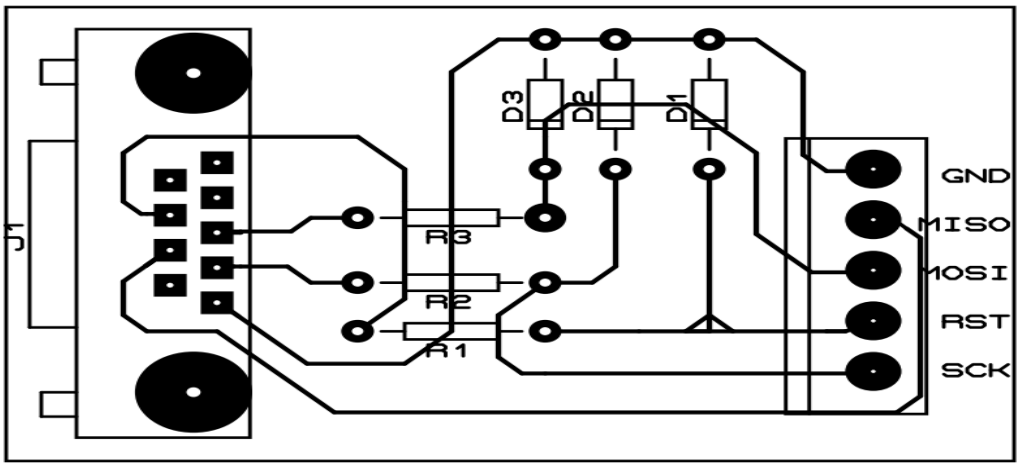


Figure 3- 9 : Fenêtres du schéma sur ARES de positionnement des différents composants électroniques du programmateur dasa.

b) L'outil de visualisation en 3D :

Cette outil (visionneur 3D) d'ARES permet de voir la carte telle qu'elle sera en réalité. C'est une aide très utile lors de la conception et la création du circuit.

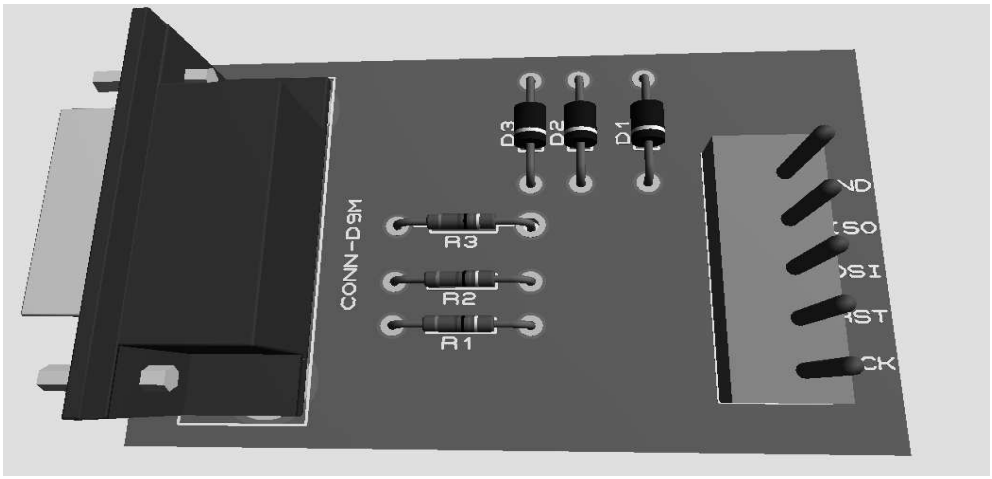


Figure 3-10 : Fenêtres du schéma sur ARES et visualisation 3D du programmateur dasa.

c) Le circuit imprimé :

En utilisant le logiciel proteus, on a réalisé le circuit imprimé spécifiquement avec ARES, afin de l'implanter sur une carte électronique.

Chapitre 4 :

Tests, résultats et applications

1) Introduction

Dans cette partie on va voir comment interfacer le microcontrôleur avec d'autres composants électroniques et comment on va relier la partie machine et logiciel. On mentionne ici que dans ce qui suit les schémas sont réalisés avec proteus et Fritzing.

Les tests performés ont suivi un processus de difficulté croissante. Ce qui veut dire qu'on a commencé par les tests de base pour arriver à une plateforme fonctionnel. Cette procédure en forme de manipulation et de travaux pratiques s'est faite pendant plusieurs semaines et durant même les jours de ramadan durant l'été 2014. Des dizaines de manipulation ont été essentielles pour nous mener à apprendre ce microcontrôleur et en utilisant la plateforme de développement en C.

On citera aussi que les travaux ont été performés au laboratoire de recherche (LPCQ) de Hasnaoua à l'UMMTO. Bien qu'on n'ait pas un appareillage coûteux ou sophistiqué, on a utilisé les moyens existants pour mener à bien nos tests. Par exemple, l'alimentation d'une ancienne unité centrale d'un microordinateur nous a été utile pour générer des tensions continues, stables et nécessaires de 3V, 5V et 12V.

2) Configuration des pins en sorties et en entrées

a) Configuration des pins en sorties

Le clignotement d'une LED est un test qui permette de configurer (programmer) les ports de microcontrôleur en sorties. Ce montage (figure dessous) permet aussi le calcul de la fréquence du microcontrôleur en mesurant le nombre de ON /OF moyennant un chronomètre.

Dans notre test on a compté avec un chronomètre 50 (ON /OF) ou 2x 50 = 100 périodes T en 61 secondes. Donc 1 T dure 0,61 second. En utilisant le programme test 1-b (voir appendice) avec un timer1 et un prescalaire de 1024. La règle de trois nous permet d'obtenir la fréquence f_c de travail de notre microcontrôleur.

Les correspondances : $f_c \rightarrow 1s$ et $5000 \times 1024 cycle \rightarrow 0{,}61s$, donnent $f_c = \dfrac{5000 \times 1024}{0{,}61} = 8{,}4\ Mhz$.

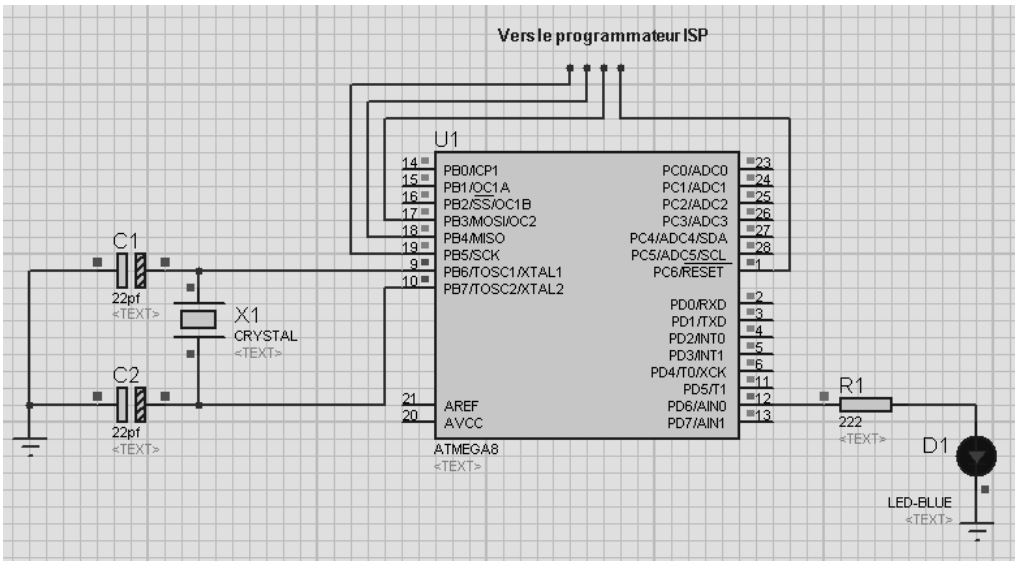


Figure 4-1 : Schéma électronique pour tester la LED et calculer la fréquence du microcontrôleur.

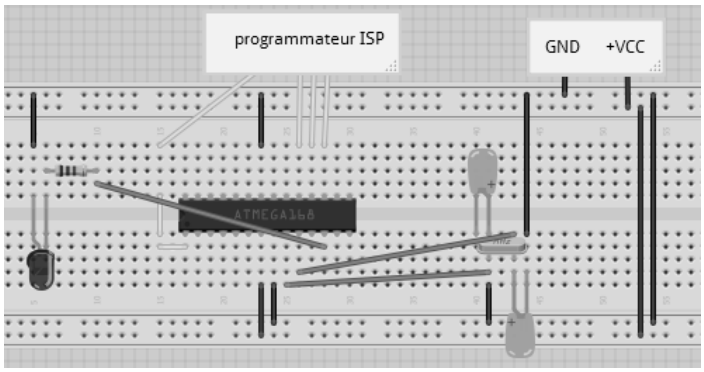
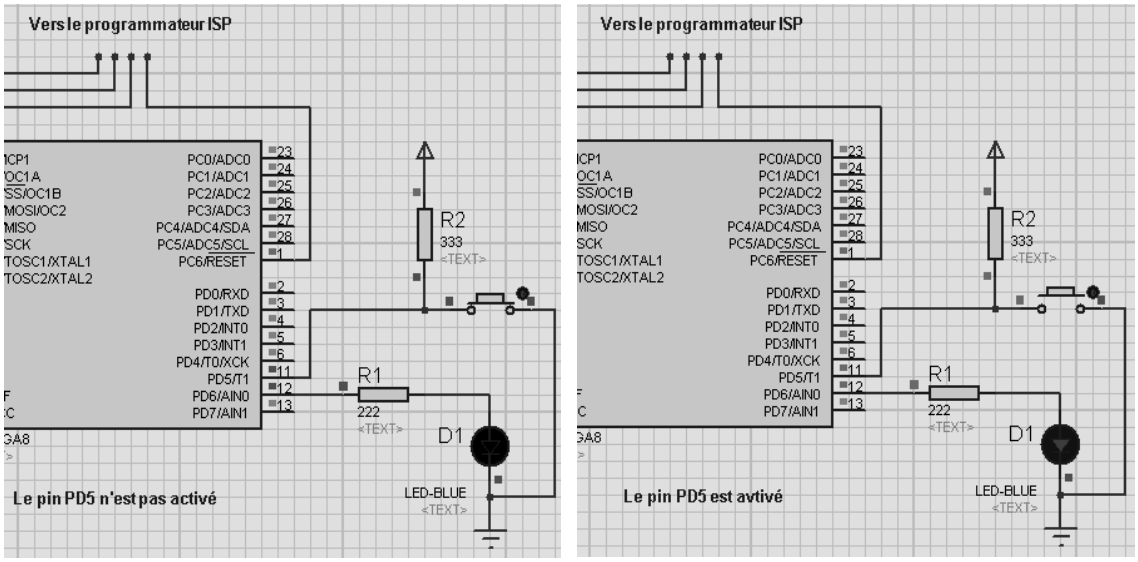


Figure 4-2 : Montage électronique du circuit ‘test led’ sur platine d’essai.

b) Configuration des pins en entrées

Ici on va voir comment programmer un port (pins) en entrée (voir le schéma électronique de la figure 4-3).

On voit bien que lorsque le pin PD5 est activé le microcontrôleur va exécuter un programme qui allume la LED. Donc en utilisant des pins du microcontrôleur en entrée on peut commander les sorties de ce même microcontrôleur.



Figures 4-3 : Montage électronique du circuit led test pin en entée avec proteus. A gauche PD5 n'est pas activé est la LED est éteint et l'inverse pour la partie de droite.

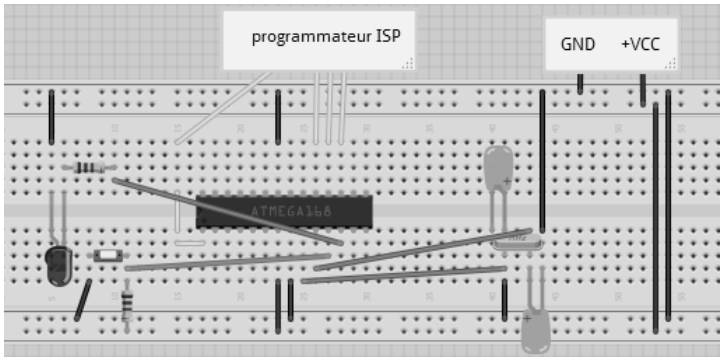


Figure 4-4 : Montage électronique test pin en entrée sur la platine d'essai.

La figure 4-4 est un montage électrique qui permet de tester les pins d'entrées et de sorties sur une platine d'essai. On notera que tous nos circuits ont été testés sur la platine d'essai ; i.e. des tests réels et non de simulation seulement.

3) Circuit de puissance ULN2083

a) Description

L'ULN2803 est un circuit de puissance et forte intensité il est constitué de transistors Darlington. Ce dispositif se compose de huit paires Darlington de type NPN qui comporte des sorties à haute tension avec cathode commune.

L'estimation de courant collecteur de chaque paire Darlington est de 500 mA.

Les paires de Darlington peuvent être reliées en parallèle pour des possibilités de courant plus élevé [26].

b) Brochage de l'ULN2803

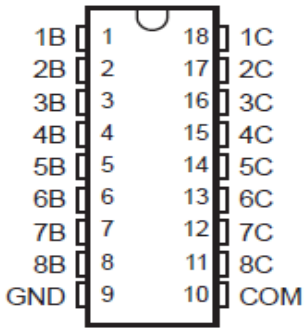


Figure 4-5: Brochage de l'ULN2803 [26].

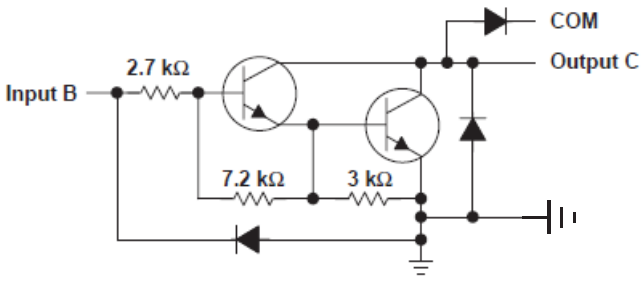


Figure 4-6 : Schéma électronique d'une paire de transistor de l'ULN2803 [26].

c) Schéma électrique d'une paire

Ce circuit comporte des diodes de roue libre pour éliminer les surtensions transistors (donc permet de commander des charges inductives, relais, bobinage moteur).

d) Caractéristiques de l'ULN2803

Un courant de sortie maximal allant jusqu'à 500mA.

Une tension de sortie maximale allant jusqu' à 50V.

Une tension d'entrée de 0. 5V jusqu'à 30V [26].

e) Principe de fonctionnement

Le fonctionnement est très simple. Pour chaque étage, on a :

Entrée numérique	Sortie	Etat
HAUT	0V	MARCHE
BAS	+V	ARRET

Tableau 4-1 : Fonctionnement de chaque paire [26].

Ainsi, la sortie de puissance est le reflet inversé de l'entrée de commande et l'élément électrique commandé est en MARCHE sur le niveau HAUT.

4) Moteur et engrenages

a) Schéma du montage

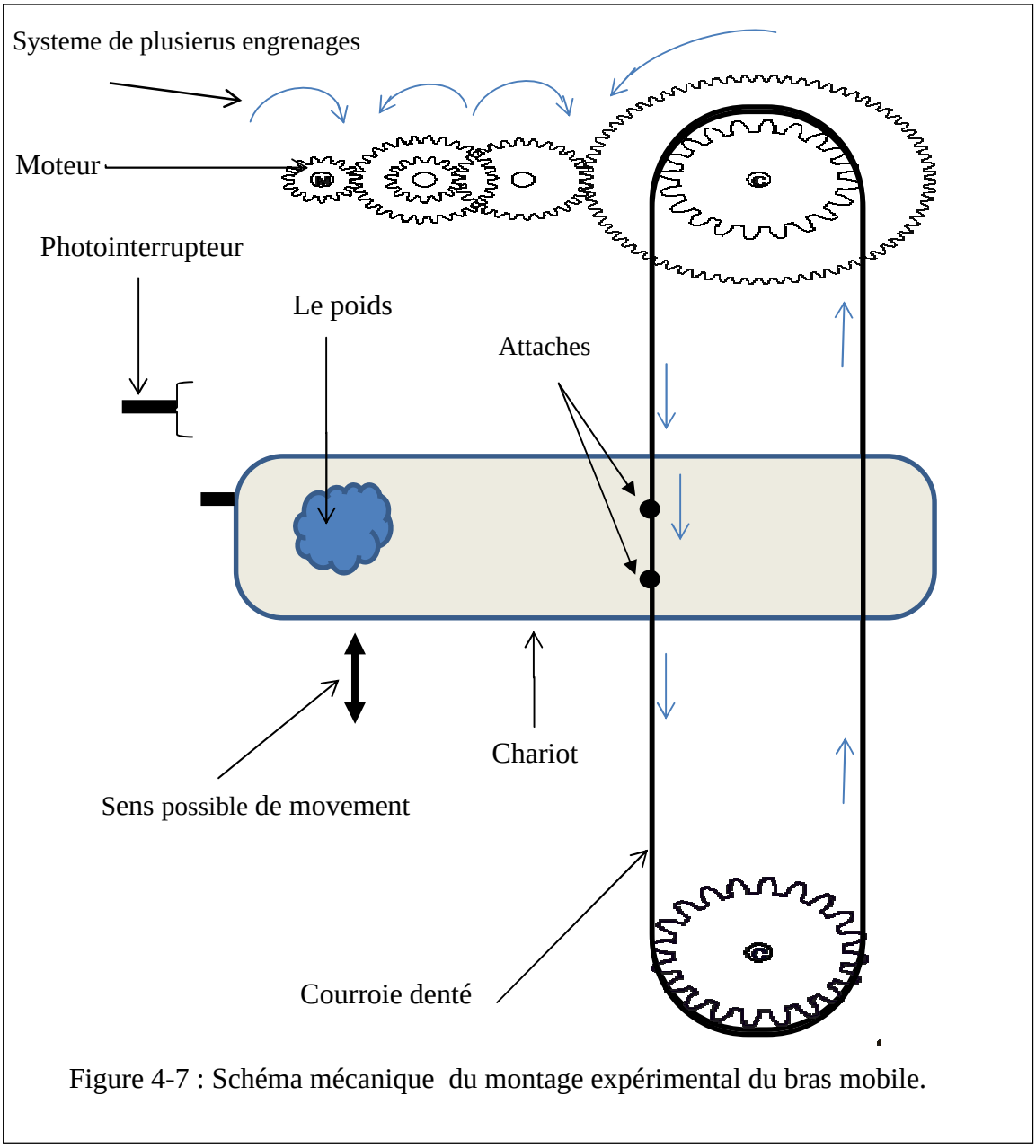


Figure 4-7 : Schéma mécanique du montage expérimental du bras mobile.

Les engrenages permettent de transmettre un mouvement de rotation. Lorsqu'on met en contact deux roues dentées, il suffit d'en faire tourner une pour que la deuxième se mette aussi à tourner. Si les deux roues ont le même nombre de dents, elles tournent à la même vitesse. Si une roue à deux fois moins de dents que l'autre, elle tournera deux fois plus vite.

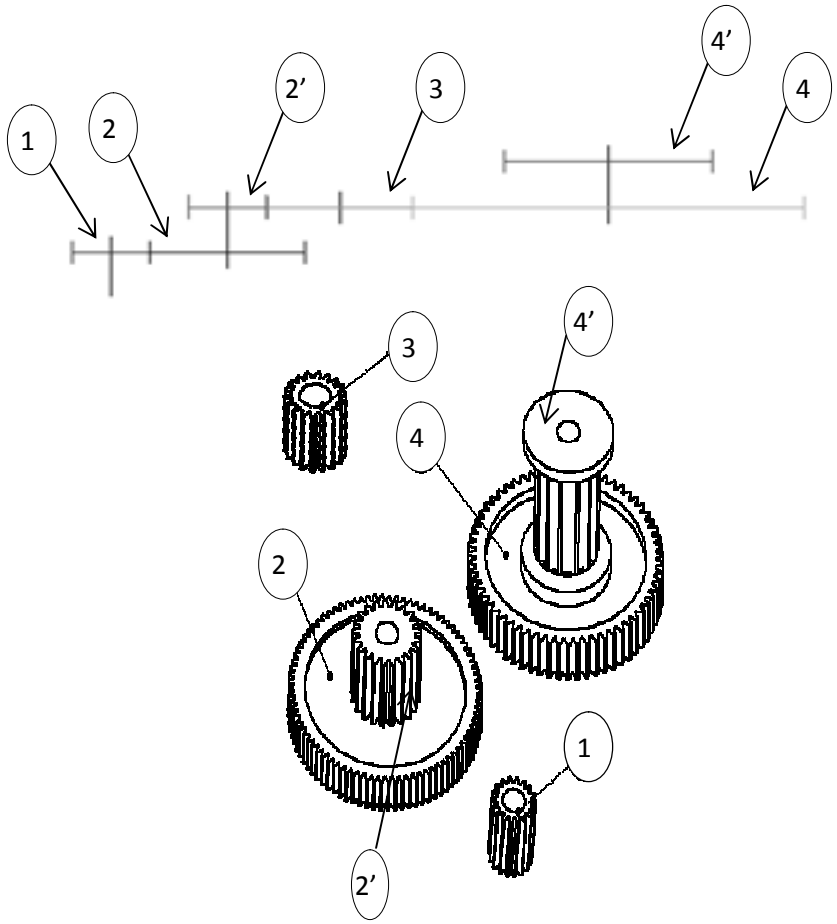


Figure 4-8 : Positions relatives des différents engrenages de notre système d'entraînement, et leurs vue en perspectives [27].

b) Caractéristiques des pignons

Système	Pignon (nbre de dents)	Roue (nbre de dents)	diamètre (D) des pignons et roue (mm)
Pignon-moteur (1)	Zp1=15	/	5
Pignon-roue (2) / (2')	Zp2=15	Zr2=30	16 / 8
Pignon (3)	Zp3=28	/	14
Pignon-roue (4) / (4')	Zp4=20	Zr4=75	32/16

Tableau 4-2 : Valeurs du nombre de dents et du diamètre des systèmes d'engrenages dans notre table de travail.

c) Calcul de la vitesse transmise par le moteur

La vitesse est un paramètre réglable qui est relié à la durée de la pulsation d'un pas. Pour calculer la vitesse angulaire ω du moteur, on calcule la vitesse linéaire v du test 03. On effet, les mesures donne un déplacement d'un pas de $40\,\mu m = 0{,}04\,mm$ en $5\,ms$ et donc $v = 8\ mm/s$.

La relation reliant la vitesse linéaire v et la vitesse angulaire ω pour un cylindre denté (engrenage) m est : $v_m = \omega_m R_m$. La vitesse angulaire ω n'est autre que la vitesse ω_1 du pignon 1 attaché au moteur. En utilisant le tableau des valeurs des diamètres du (Tableau :4-2), on calcule les vitesses angulaires de chaque pignon de notre dispositif. On obtient : $\omega_1 = 3{,}2\ rad/s$, $\omega_2 = \omega'_2 = 2\ rad/s$, $\omega_3 = 1{,}14\ rad/s$ et $\omega_4 = \omega'_4 = 0{,}5\ rad/s$.

d) Calcul du moment du moteur

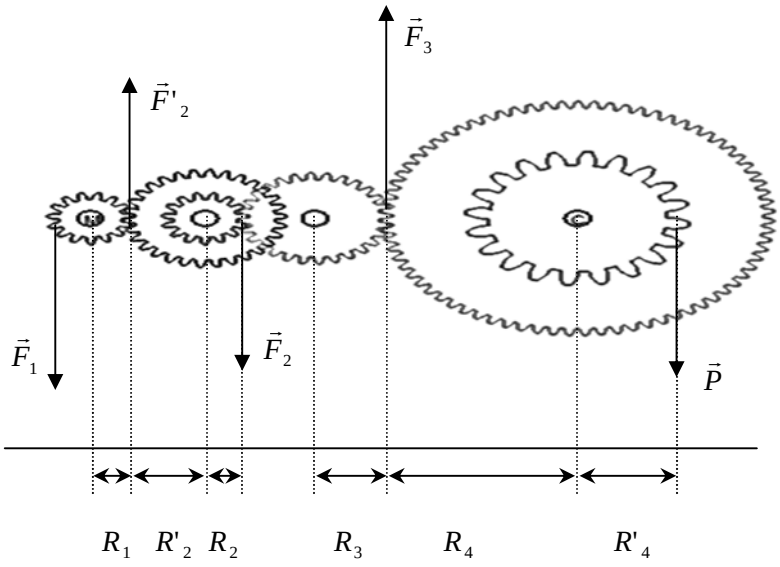


Figure 4-9: Transmission de la force (couple) du point P au moteur.

Pour faire ce calcule, on attache une masse $m = 700\,g$ (une bouteille d'eau) par un fil au chariot. Ce fil est parallèle à la droite sortant de la roue de rayon R'_4 . On commence par un délai de pas assez large qui permet de faire avancer la masse. On réduit ce délai de temps jusqu'à arriver à l'état d'équilibre où la masse est immobile. On calcule alors les forces et les moments de forces.

La physique de base définit le moment d'une force par $\vec{M}_i = \vec{r}_i \times \vec{F}_i$. Pour des forces perpendiculaires aux rayons, on a $M_i = r_i\ F_i$. A l'équilibre, on a : $\sum \vec{M}_i = \vec{0}$. Ceci donne :

$$r'_4\ P = r_4\ F_3 = r_3\ F_2 = r'_2\ F'_2 = r_1\ F_1 = Couple\ du\ moteur$$

Sachant que $P = 7\,N$ et les diamètres ceux du tableau 4-2, on obtient :

$F_1 = r'_4 P / r_1 = 22{,}4\,N$, $F_2 = r'_4 P / r_3 = 8\,N$, $F'_2 = r'_4 P / r'_2 = 14\,N$

et $F_3 = r'_4 P / r_4 = 3{,}5N$.

Le moment exercé sur le moteur est $M_1 = r_1 \ F_1 = 56 \times 10^{-3} \ mN$.

e) Commande du moteur pas à pas à 5 fils

Le moteur à 5 fils. Le milieu des deux bobines étant commun. A ce point central, on applique l'alimentation et successivement on connecte un des 4 autres fils à la terre et dans un certain ordre. Si l'ordre est correct, le moteur tourne d'un angle qu'on appelle un pas.

Généralement lorsqu'on commande des moteurs, on fait appel à des circuits de puissances comme le transistor, le pont H (LD293), ULN2803 ou relais.

f) Montage électronique de commande pin par pin et commande de moteur pin par pin (mode monophasé)

Ce test permet de faire un pas avec le moteur on faisant activer pin par pin (alimenter chaque fil du moteur). Le programme suivant fait tourner aussi le moteur dans deux sens.

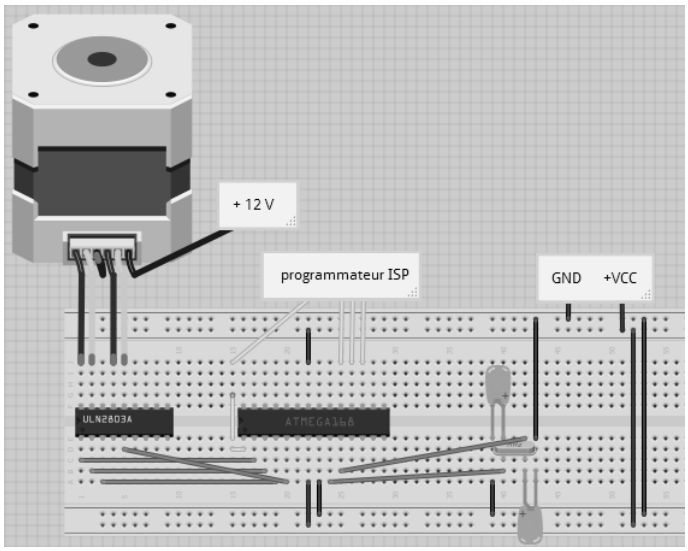


Figure 4-10 : Montage électronique commande de moteur sur la platine d'essai.

g) Rotation par pas complet

Les 4 fils sont reliés à l'ATMEGA8 par l'intermédiaire d'ULN2803, ces 4 fils sont des bobines connectées à 1c, 2c, 3c et 4c de l'ULN2803. Dans ce mode, seul le monophasé est excité pour faire tourner le moteur. Le mode pas à pas se fait en 4 étapes (1c, 2c, 3c et 4c) et se répète pour faire des cycles complets.

La fonction est définit par la séquence qui se trouve dans le tableau suivant :

Cycle	1C	2C	3C	4C
Pas 1	HAUT	BAS	BAS	BAS
Pas 2	BAS	HAUT	BAS	BAS
Pas 3	BAS	BAS	HAUT	BAS
Pas 4	BAS	BAS	BAS	HAUT

Tableau 4-3 : Séquence de rotation par pas complète.

h) Commande en mode biphasé

Le schéma électronique est le même que précédemment.

Ce mode conduit avec l'excitation de deux phases (2 bobines sont excitées). Ce mode dispose de 4 cycles, cependant le moteur fonctionne avec une puissance plus élevée que celle du mode à une phase. Pour ces deux modes, le moteur tourne de la même façon.

La séquence de commande double pins est :

Cycle	1C	2C	3C	4C
Pas 1	HAUT	HAUT	BAS	BAS
Pas 2	BAS	HAUT	HAUT	BAS
Pas 3	BAS	BAS	HAUT	HAUT
Pas 4	HAUT	BAS	BAS	HAUT

Tableau 4-4 : Séquence double phases.

i) Commande par demi pas

Ce mode est la combinaison de l'excitation monophasée et l'excitation double phase. Tous les deux sont alimentées alternativement afin de tourner le moteur de demi pas, ce mode de fonctionnement a 8 cycles.

La commande par demi-pas fournit une bonne résolution mais vitesse réduite de moitié.

Le tableau suivant montre la séquence :

Cycle	1C	2C	3C	4C
(1/2)PAS 1	HAUT	BAS	BAS	BAS
(1/2)PAS 2	HAUT	HAUT	BAS	BAS
(1/2)PAS 3	BAS	HAUT	BAS	BAS
(1/2)PAS 4	BAS	HAUT	HAUT	BAS
(1/2)PAS 5	BAS	BAS	HAUT	BAS
(1/2)PAS 6	BAS	BAS	HAUT	HAUT
(1/2)PAS 7	BAS	BAS	BAS	HAUT
(1/2)PAS 8	HAUT	BAS	BAS	HAUT

Tableau 4-5 : Séquence de demi pas.

j) Commande du moteur dans deux sens via la fonction « movh »

On a crée cette fonction dont le but est de minimiser le programme (taille de programme) et de faciliter les calculs. On a utilisé la fonction « movh ». On changeant les paramètres, on arrive à positionner le chariot en différentes locations prédéfinies.

On peut, par exemple, déclarer un tableau de « pas » et un tableau de « temps » comme suit :

```
uint16_t cbuf[6]={4800, 3900, 3000, 2500, 1500, 800} ;
uint16_t timebuf[6]={800, 390, 300, 2500, 500, 80} ;
```

Le moteur va à chaque position et s'arrête durant le délai correspond. Ce cycle se répète, en faisant une loop, pour chaque position. On a ajouté aussi un retour à la position initiale après chaque étape. Cette manipulation a été visualisée avec le bras du scanner. Le photointerrupteur servira à changer l'état d'un pin pour arrêter le moteur et donc pour l'initialiser.

5) Photointerrupteur

Le photointerrupteur (opto-coupleur) en fourche associe dans un même boîtier plastique une LED qui émet de la lumière infrarouge et un phototransistor qui détecte la lumière émise par la LED.

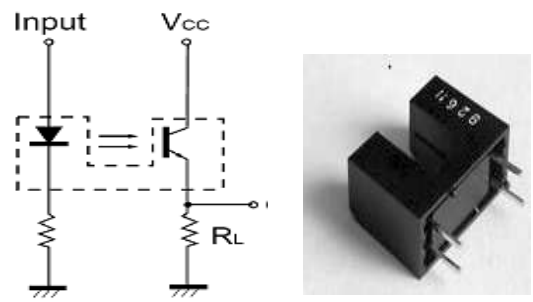


Figure 4-11 : Photo et circuit electrique d'un photointerrupteur [28].

Les deux composants sont face à face et séparés par une fente dans laquelle il est possible de faire passer une roue encodée ou un objet rotatif quelconque qui bloque ou laisse passer la radiation IR. L'ensemble forme une barrière photoélectrique miniature.

Ce composant est aisément interfacé au microcontrôleur. Il suffit d'en rajouter, comme pour toute LED ou transistor, de résistances.

Les valeurs typiques de résistances sont à utiliser avec ce type composant. En général une valeur de $100\,\Omega$ pour la résistance en série avec la LED et une valeur de $4{,}7\,k\,\Omega$ pour celle avec le phototransistor.

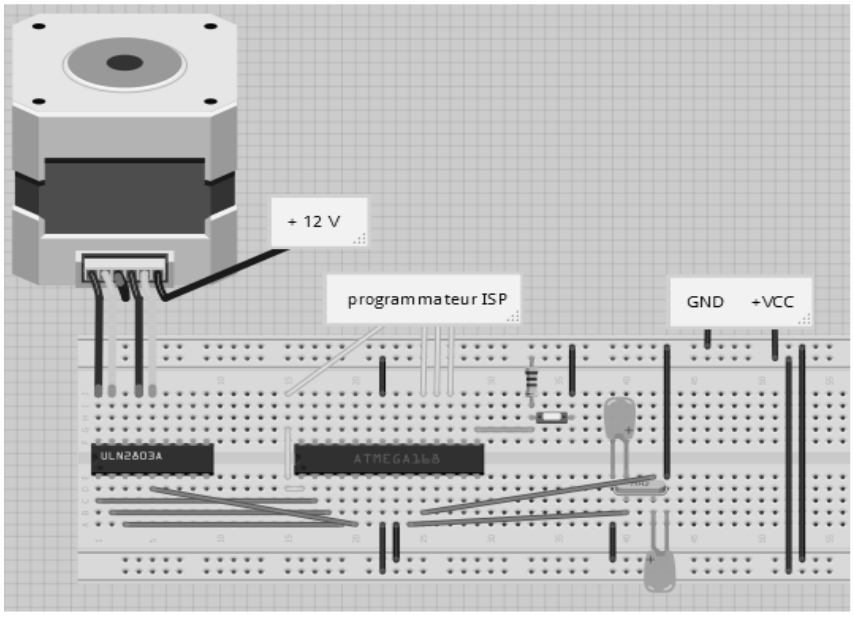


Figure 4-12 : Circuit pour tester le moteur dans deux sens avec un pin d'entrée.

6) Commande à distance

Il existe plusieurs manières techniques pour effectuer une commande à distance, comme la télécommande TV ou MP3.

a) Télécommande

La télécommande est un dispositif électronique généralement de taille réduite. Elle sert à commander un autre dispositif à distance sans utiliser un lien par câble. Cette commande à distance peut être réaliser par infrarouge ou par ondes radio. On a choisi l'infrarouge.

b) Télécommande hertzienne

L'onde porteuse appelé « onde hertzienne » ou onde radio est un signal porteur électromagnétique servant à véhiculer des informations électriques non visibles la portée est bien supérieur à celle de l'infrarouge, ces ondes franchissent aussi des petits obstacles comme le mur. Elles balaient un angle assez grand, ce qui permet de ne pas viser directement l'objet à piloter.

c) Télécommande à infrarouge

L'infrarouge est émis par une diode infrarouge. C'est composant électronique qui au passage d'un courant électrique (environ 20 mA) produit une lumière ayant un spectre de longueur d'onde invisible à l'œil nu, et se situant au-dessous de rouge dit infrarouge (800-1000 nm). Ce système fonctionne sur des distances de quelques mètres avec une transmission de l'ordre de 30 à 40 KHZ.

L'information transmise est de forme binaire, ne peut prendre que deux valeurs 0 ou 1 cette suite binaire forme une trame.

d) Détail d'une trame de donné de la télécommande à infrarouge

Une trame est une série de bit comprenant l'information complète à émettre. Dans le code RC5, elle se compose d'une suite de 14 bits, dans notre cas la trame émise par la télécommande MP3 est sur 32 bits.

La télécommande a besoin d'un autre outil pour qu'elle soit active c'est le récepteur.

Il existe plusieurs types de récepteurs infrarouges. Nous avons utilisé un TSOP.

e) Description du détecteur “Thin Small Outline Package” (TSOP)

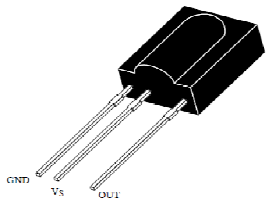
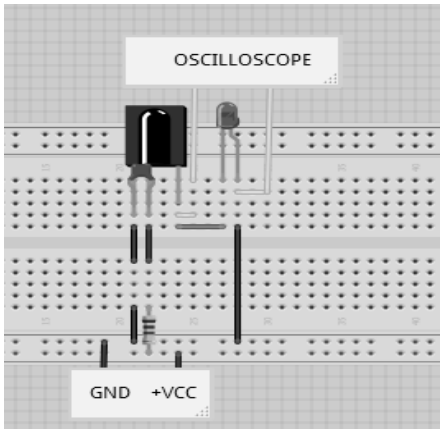


Figure 4-13 : Le détecteur IR TSOP [29].

Figure 4-14 : Télécommande MP3.



4-15 : Circuit test du TSOP avec l’oscilloscope.

Les TSOP17xx sont des récepteurs conçus pour les systèmes de télécommande infrarouges. La PIN diode et le préamplificateur sont assemblés sur un armateur du fil, le paquet époxy conçu en tant que filtre infrarouge. Le signal de la sortie démodulée peut être directement décodé par un microcontrôleur. Le récepteur infrarouge standard de télécommande supporte tous les codes de transmission.

f) Circuit de test du détecteur TSOP

Le montage électronique de la figure 4-15 permet de tester le TSOP en envoyant un signal par télécommande, la LED rouge va s’allumer (ou plutôt clignoter vite).

Grâce au logiciel scope et la sortie de la carte son d’un ordinateur, nous avons réussi à décoder les trames envoyées par une télécommande MP3. Le signal suivant, par exemple, est un signal envoyé par la touche (ch+).

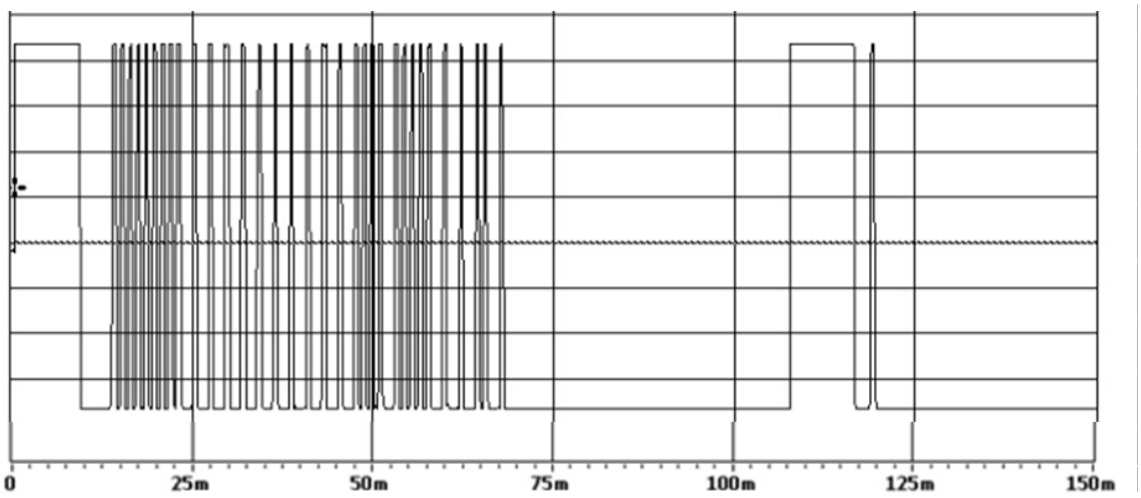


Figure 4-16 : Un exemple de la trace du signal de l'oscilloscope de la trame envoyée par la touche (ch+) de la télécommande.

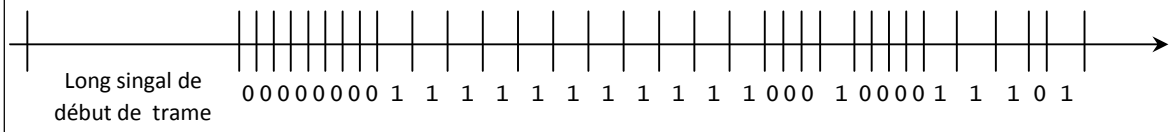


Figure 4-17 : Décodage binaire de la trame correspondant à la touche (ch+).

La suite binaire correspondante est : 0000 0000 1111 1111 1110 0010 0001 1101, soit (00 FF A2 5D) en hexadécimale.

Les premiers 16 bits sont les bites d'identification de la télécommande, les 16 autres sont les bites qui correspondent à la touche (ch+). On remarque aussi que les 8 derniers bits sont les compléments binaires des 8 avant derniers bits. Pour la touche (ch+), chaque bit de 0001 1101 est inversé pour donner 1110 0010. Cette particularité donne un dispositif de vérification des bits transmis.

Le tableau suivant représente les codes envoyés par chaque touche de la télécommande.

Touche	Code en binaire	Code en Hexadécimal	Touche	Code en binaire	Code en Hexadécimal
0	0000 0000 1111 1111 0110 1000 1001 0111	00 FF 68 97	+100	0000 0000 1111 1111 1001 1000 0110 0111	00 FF 98 67
1	0000 0000 1111 1111 0011 0000 1100 1111	00 FF 30 CF	+200	0000 0000 1111 1111 1011 0000 0100 1111	00 FF B0 4F
2	0000 0000 1111 1111	00 FF 18 E7	Ch+	0000 0000 1111 1111	00 FF E2 1D

	0001 1000 1110 0111			1110 0010 0001 1101	
3	0000 0000 1111 1111 0111 1010 1000 0101	00 FF A7 85	Ch-	0000 0000 1111 1111 1010 0010 0101 1101	00 FF A2 5D
4	0000 0000 1111 1111 0001 0000 1110 1111	00 FF 10 EF	Ch	0000 0000 1111 1111 0110 0010 1001 1101	00 FF 62 9D
5	0000 0000 1111 1111 0011 1000 1100 0111	00 FF 38 C7	EQ	0000 0000 1111 1111 1001 0000 0110 1111	00 FF 90 6F
6	0000 0000 1111 1111 0101 1010 1010 0101	00 FF 5A A5	Next	0000 0000 1111 1111 0000 0010 1111 1101	00 FF 02 FD
7	0000 0000 1111 1111 0100 0010 1011 1101	00 FF 42 BD	Play	0000 0000 1111 1111 1100 0010 0011 1101	00 FF C2 3D
8	0000 0000 1111 1111 0100 1010 1011 0101	00 FF 4A B5	Prev	0000 0000 1111 1111 0010 0010 1101 1101	00 FF 22DD
9	0000 0000 1111 1111 0101 0010 1010 1101	00 FF 52 AD	Vol-	0000 0000 1111 1111 1110 0000 0001 1111	00 FF E01F

Tableau 4-6 : Ce tableau montre la trame envoyée en binaire et en hexadécimal de chaque touche de la télécommande après son décodage.

g) Circuit de commande à distance

Lorsqu' une commande est envoyée, le récepteur TSOP décode la trame et comme le signal est relié à un pin du microcontrôleur, le programme analyse les bits reçus. Selon l'interprétation de la commande, une action peut être entreprise. Par exemple, on fait tourner le moteur d'un certain nombre de pas dans un sens donné pour une commande spécifié.

On a utilisé une librairie existante pour le décodage de la trame. Cette librairie a été crée au niveau du laboratoire et est une adaptation de routines facilement téléchargeable du net. On doit respecter une certaine procédure dans l'AVR studio pour pouvoir compiler notre programme avec cette librairie. L'utilisation de bibliothèque simplifie énormément la programmation et nous évite de recréer ce qui existe déjà. Il suffit alors d'ajouter quelques lignes de codes pour agir lors de la réception d'une trame.

On utilise une particularité du microcontrôleur. On configure Timer/Counter1 en mode ICP (input capture mode). Dans ce mode et selon la configuration désirée, une interruption est générée dès qu'un change d'état (montant, descendant ou les deux). Le pin PB0 devient ICP1 et doit être relié au pin de sortie de la TSOP (voir le schéma de la figure 4-18).

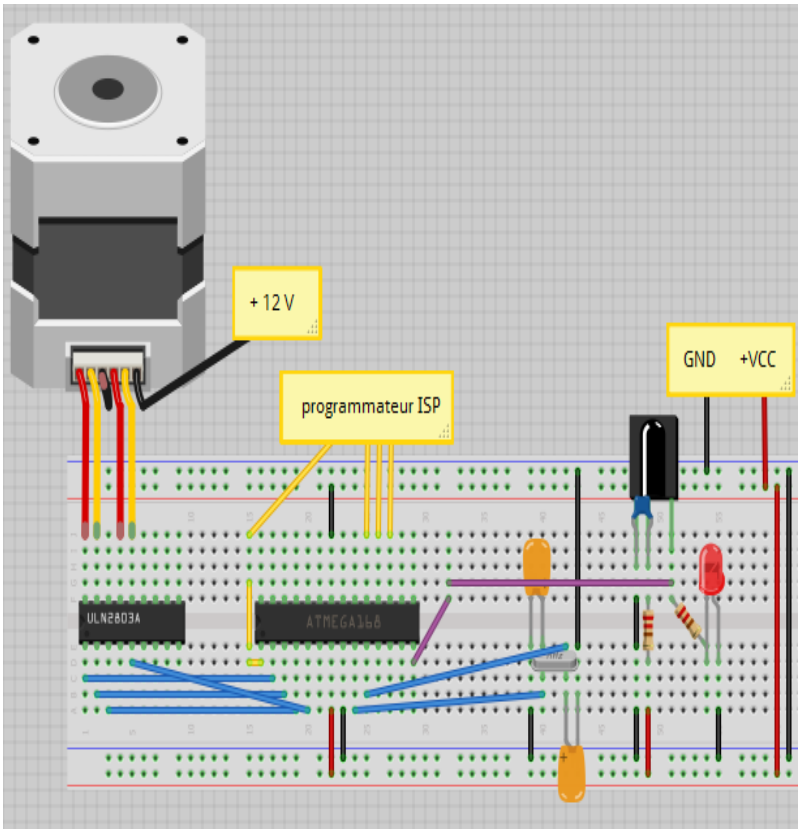


Figure 4-18 : Montage électronique de commande à distance.

7) Conclusion

Tous les éléments du montage ont été décrits, configurés et testés. Le montage final consiste à faire déplacer un chariot le long d'un axe. Ce mouvement est géré par un moteur pas à pas qui transmet son mouvement de rotation précis par une chaîne d'engrenages au chariot. On a commandé ce système à distance en utilisant une télécommande banale qu'on a programmée. Tout le système électronique est centré autour d'un microcontrôleur de type Atmega8. Plusieurs caractéristiques ont été mesurées et sont conforme à l'expectation. La plateforme simple et gratuite de développement AVR Studio a été utilisée pour le développement du logiciel en C qu'on a flashé dans l 'EEPROM via un programmateur monté localement. Les systèmes réalisés constituent une base essentielle et utile pour des applications plus complexes applicable dans plusieurs domaines techniques et technologiques.

Conclusion générale

Ce travail constitue une base essentielle d'ingénierie. La philosophie et stratégie utilisées d'un point de vue pédagogique et technique sont fondamentales pour atteindre l'objectif souhaité. On mentionnera ici quelques concepts qu'on a appris en réalisant ce projet de mémoire : travail de groupe, système pluridisciplinaire, coopération interdépartementale, discussion et ouverture d'esprit sur d'autres possibilités, entraînement d'entrepreneur, adaptation de ressources existantes, réutilisation de composants et recyclage, utilisation de plateforme performante mais gratuite, recherche biobibliographique, création d'un document technique, préparation d'une défense orale et d'une présentation pratique.

On insistera sur le recyclage comme moyen local permettant de récupérer des composants qu'on adapte à un projet donné. Cette démarche offre à l'étudiant et au chercheur la possibilité d'avoir les moyens basiques, de qualité et peu coûteux. Les instituts doivent offrir d'autres ressources pour complémenter cette tendance.

Entre autre, une analyse d'autocritique est nécessaire pour nous permettre de combler les lacunes rencontrées et nous permettre d'avancer dans nos carrières. On citera la commande de la langue française ou même anglaise pour accéder et utiliser les ressources, le manque d'expérience dans la rédaction de document et l'insuffisance de manipulations pratiques durant notre formation.

Néanmoins, en réalisant ce travail, nous avons globalement exploité une plateforme complète de développement pour microcontrôleur en langage C. On a aussi interfacé celui-ci à d'autres composants électroniques. L'asservissement d'une télécommande, à distance et sans contact filaire, comme un moyen de recevoir un input qu'on utilise programmatiquement pour performer une tâche donnée est un acquis essentiel. L'interface d'un contrôleur au monde extérieur via un contact filaire direct (LED par exemple) ou via un circuit de puissance (transistor) permet de translater un besoin technique en lignes de programmation. Ces dispositifs sont nécessaires dans beaucoup d'applications et dans plusieurs domaines.

Ce travail nous a permis de tester et vérifier quelque performances et caractéristiques des microcontrôleurs AVR de chez ATMEL. Ils présentent des avantages réels et des ressources d'applications efficaces. Un simple programme test, par exemple, nous a permis de trouver la fréquence interne de 8 *MHz*. Ce microcontrôleur en combinaison avec la plateforme de développement en C et un programmateur nous donne une plateforme complète, efficace et accessible. Moyennant une base technique et une préparation adéquate, la courbe d'apprentissage de ce système nous parait courte. La combinaison de tout ces facteurs nous amène à conclure que cette plateforme devance d'autres et devrait nous aider à migrer à d'autre système encore plus performant ; i.e. microcontrôleurs à 32 bit et à cœur d'ARM.

Le type de moteur étudié, de part sa simplicité et son acquisition, nous permet d'avoir plusieurs perspectives d'applications. Pour le commander, on a performé plusieurs tests et pris plusieurs données. On a vérifié les trois modes (pas, demi pas et couple fort) ainsi que leurs caractéristiques. Pour le mode demi pas la vitesse est réduite de moitié alors la résolution est double. Pour le mode couple fort, la puissance du système est meilleure. On a aussi observé que le système étudié est précis donnant une résolution de mouvement de translation de 40 µm pour le mode couple fort. On a fait aussi fait varier le temps des pulses pour changer la vitesse.

En plus, on a utilisé un moyen efficace, abordable et d'interface facile. Une télécommande IR banale, une librairie et quelques lignes de codes dans notre logiciel permettent de l'utiliser comme solution. L'avantage de ce type d'input est l'interface sans fil. Cette télécommande nécessite l'utilisation d'un détecteur TSOP. L'analyse de la trame envoyée, en utilisant un logiciel qui transforme une carte son d'un PC en oscilloscope de fortune, et son décodage nous donne une assise technique solide pour d'autre type d'analyses s'appuyant sur des techniques similaires. Les résultats de cette analyse sont clairement détaillés dans ce mémoire.

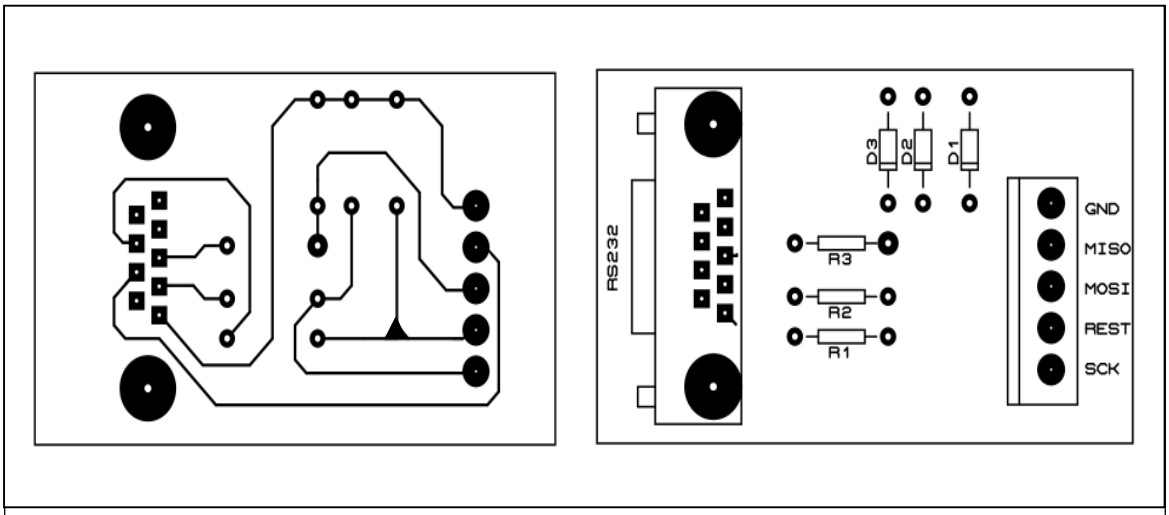
En assemblant ses fondamentaux, on a réussi à l'appliquer à un chariot mobile via un système d'engrenages et de courroie. Le moteur pas à pas fait tourner une roue dentée. Le positionnement de ce chariot se fait automatiquement selon le design de notre programme et l'interprétation des trames de la télécommande IR. Quelques tests ont été performés et des mesures ont été prises. Les résultats sont discutés dans le texte de notre mémoire. Une démonstration sera faite durant la présentation orale de ce travail.

Ce type d'expérimentation ouvre des perspectives dans plusieurs domaines et peuvent servir à plusieurs applications destinées aux différents domaines de l'industrie, de la médecine, de la recherche et de l'enseignement.

On donnera un exemple simple d'application dans notre filière 'électronique biomédical' : le lit d'un scanner. Cette manipulation peut être adaptée, moyennant une électronique de puissance, à ce lit pour qu'il soit positionné automatiquement et précisément en des positions prédéterminées afin de prendre des données qui sont ensuite traitées et visualisées.

Appendices

1- Schémas électroniques de carte de programmateur



Schémas électroniques de la carte du programmateur.

2- Ci-bas, on inclus plusieurs portions des programmes qu'on a utilisé.

```
#define F_CPU 8000000UL
#define sbi(x, y) (x |= (1<<y));                 /* set bit y in byte x */
#define cbi(x, y) (x &= (~ (1<<y)));             /* clear bit y in byte x */
#define toggle(x, y) (x ^= (1<<y));
#include <avr/io. h>
#include <util/delay. h>
#include "tsopdecode. h"     // ionclure la biblio de la télécommande

// TEST:  Pin en sortie
int main (void) {
       sbi (DDRD, 6);              // le pin PD6 est en sorties
       while (1) {
             sbi (PORTD, 6);       // allumer le pin 6
             _delay_ms(500);       // attendre 500 milli seconde
             cbi (PORTD, 6);       // éteindre le pin 6
             _delay_ms (500);
       }
       return 0;
 }
// TEST:   calcul de fréquence
      if (test == 18)                          // timer1 sans interruption
      {
            sbi(DDRD, 7);                      // sortie de pin 7
            TCCR1B |= (1 << CS10) | (1 << CS12); // un présalaire 1024
```

```
            While (1) {
                  if (TCNT1 >= 5000) {          // compteur TCNT1 atteint
                        toggle (PORTD, 7);      // allumer et éteindre PD7
                        TCNT1 = 0;              // Reset timer value
                  }
            }
      }

//  TEST : Pin en entrée
int main (void) // test entrée
{
      sbi (DDRD, 6);          // pin PD6 en sortie cbi (DDRD, 5);
      sbi (PORTD, 5);         // pin PD5 en entrée
      while (1) {
            if (bit_is_clear (PIND, 5)) { // si le pin PD5 est éteint "0"
                  cbi (PORTD, 6); // pin PD6 est à "0"
            }
            else { // sinon
                  sbi (PORTD, 6); // le pin PD6 est à "1" (allumé)
            }
      }
}

// TEST : commande monophasé
int main (void) {
      int j ; int s1 ;
      while(1) {
            for (j = 0; j < 800; j++) { // pour 800 pas
                  s1 = (j%4);             // le port sortie est s1= j modulo 4
                  sbi (PORTB, s1);
                  _delay_ms(5);
                  cbi (PORTB, s1) ;  // Moteur tourne dans un sens
            }
            for (j = 0; j < 800; j++) {
                  s1 = (j%4);
                  sbi (PORTB, (3-s1));
                  _delay_ms(5);
                  cbi (PORTB, (3-s1));  // Moteur tourne dans le sens opposé
            }
      }
}

// TEST : commande double phase
int j, int s1, int s2;
      While (1) {
            for (j = 0; j < 800; j++) {
                  s1 = (j%4); //
                  s2 = (((j%4) +1) %4); //
                  sbi (PORTB, s1);  sbi (PORTB, s2);
                  delay_ ms (5);
                  cbi (PORTB, s1); cbi (PORTB, s2);
            }
            for (j = 0; j < 800; j++) {
                  s1 = ((800-j) %4);
                  s2 = ((((800-j) %4) +1) %4);
                   sbi (PORTB, s1); sbi (PORTB, s2);
                  _delay_ ms (5);
                   cbi (PORTB, s1);  cbi (PORTB, s2);
            }
```

```
      }

// TEST : commande demi-pas
int main (void)
 {
Int j ;
Int s1, s2 ;
      While (1) {
            for (j = 0; j < 800; j++) {
                  s1 = (j%8)/2;
                  s2 = (((j%8) +1)%8)/2;
                  sbi (PORTB, s1); sbi (PORTB, s2);
                  _delay_ms(5);
                  cbi (PORTB, s1); cbi (PORTB, s2);
            }
            for (j = 0; j < 800; j++) {
                  s1 = ((800-j)%8)/2;
                  s2 = ((((800-j)%8) +1)%8)/2;
                  sbi (PORTB, s1); sbi (PORTB, s2);
                  _delay_ms (5);
                  cbi (PORTB, s1); cbi (PORTB, s2);
            }
      }

//  TEST  : la fonction « movh » et position initial
{
uint16_t cbuf[6]={4800, 3900, 3000, 2500, 1500, 800} ; // tableau de pas
uint16_t timebuf[6]={800, 390, 300, 2500, 500, 80} ; // tableau de temps
sbi(DDRD, 7);            // pin en sortie
cbi(DDRD, 6);     // pin en entrée
sbi(PORTD, 6);     //
      for (j = 0; j < 200; j++) {
            moveh_1(1);
      }
      While (bit_is_clear (PIND, 6)) {
      moveh_1 (0);
      }
numstep = 0; // position initial
      for (i = 0; i < 6; i++) {
            for (j = 0; j < cbuf[i]; j++) moveh_1(1);
      }
Cbi (PORTD, 7);
_delay_ms (timebuf[i]);
sbi (PORTD, 7);
      while (bit_is_clear (PIND, 6)) moveh_1(0);
}
void moveh_1(unsigned char idire )
{
      int s111, s222;
            if(idire == 1) {
                  numstep++;
            }
      if(idire == 0) {
            numstep--;
      }
      s111 = (numstep%8)/2;
      s222 = (((numstep%8)+1)%8)/2;
      sbi(PORTD, s111); sbi(PORTD, s222);
      _delay_ms(10);
```

```
      cbi(PORTD, s111); cbi(PORTD, s222);
      return ;
}

// TEST : commande à distance
{
sbi(DDRD, 7);           //output pin
cbi(DDRB, 0);           //input pin for icp
timer2_init();
      while(1)
      {
       }                                        // end of test
                        // icp;
{
sbi(DDRD, 7);           // pin en sortie
cbi(DDRB, 0);           //pin en entrée pour l' icp
timer2_init();
while(1){
      if(ico1 == 1) {
            ico1 = 0;
            comma1 = 0;
            comma1 = tsopdecode ();
       }
 }
 }
```

3- DATASHEET Atmega8 (1ère page seulement résumant ses caractéristiques essentielles) [1]

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 130 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 8K Bytes of In-System Self-Programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 512 Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 1K Byte Internal SRAM
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler, one Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Three PWM Channels
  - 8-channel ADC in TQFP and MLF package
    Eight Channels 10-bit Accuracy
  - 6-channel ADC in PDIP package
    Eight Channels 10-bit Accuracy
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Five Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-lead PDIP, 32-lead TQFP, and 32-pad MLF
- Operating Voltages
  - 2.7 - 5.5V (ATmega8L)
  - 4.5 - 5.5V (ATmega8)
- Speed Grades
  - 0 - 8 MHz (ATmega8L)
  - 0 - 16 MHz (ATmega8)
- Power Consumption at 4 Mhz, 3V, 25°C
  - Active: 3.6 mA
  - Idle Mode: 1.0 mA
  - Power-down Mode: 0.5 µA

8-bit AVR® with 8K Bytes In-System Programmable Flash

ATmega8
ATmega8L

2486O–AVR–10/04

4- DATASHEET MOTEUR pas à pas utilisé [20]

MITSUMI

Stepping Motors
M35SP-7

Stepping Motors

OUTLINE

"M35SP-7" has the thinnest body among the 35mm outer diameter model series.
Output torque characteristics-holding torque : 29.4mN·m, pull-out torque : 18.1mN·m/200pps, and pull-in torque : 17.6mN·m/200pps (6V DC).
With these torques this motor materializes an excellent size performance.

FEATURES

1. Compact size and high output torque.
2. Superior running quietness and stability.
3. Step angle : 7.5°.
4. Excellent responsiveness acquired.

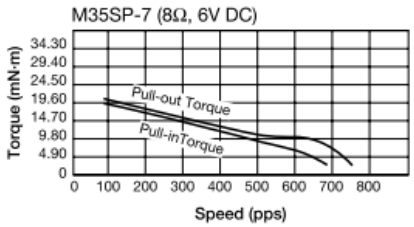
USES

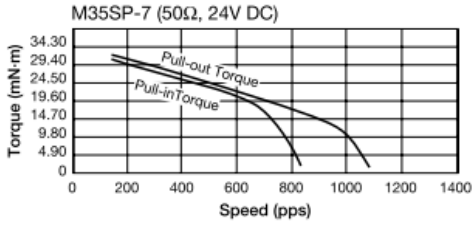
Printers, typewriters, word processors, facsimiles, and such.

SPECIFICATIONS

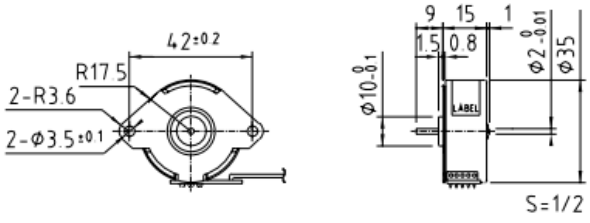
Items	M35SP-7	
Rated Voltage	DC 6V	DC 24V
Working Voltage	DC 5.4~6.6V	DC 21.6~26.4V
Rated Current/Phase	807mA max.	517mA
No. of Phase	4 Phase	4 Phase
Coil DC Resistance	8Ω/phase±7%	50Ω/phase±7%
Step Angle	7.5°/step	7.5°/step
Excitation Method	2-2 Phase excitation (Unipolar driving)	
Insulation Class	Class E insulation	Class E insulation
Holding Torque	29.4mN·m	34.3mN·m
Pull-out Torque	18.1mN·m/200pps	31.4mN·m/200pps
Pull-in Torque	17.6mN·m/200pps	30.9mN·m/200pps
Max. Pull-out Pulse Rate	770pps	1,050pps
Max. Pull-in Pulse Rate	710pps	850pps

CHARACTERISTICS





DIMENSIONS



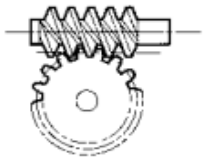
Unit : mm, General tolerance : ±0.5

5- Rappel et formulaire de mécanique [21]

RAPPEL DE MÉCANIQUE

Système roue/vis

$$J = J_V + \frac{1}{R^2} J_r$$



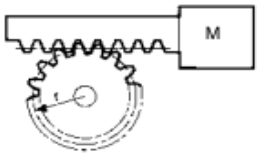
J_v = Inertie de la vis considérée comme un cylindre de diamètre égal au diamètre primitif.

J_r = Inertie de la roue considérée comme un cylindre plein de diamètre égal a un diamètre primitif

R = Rapport de réduction

Crémaillère

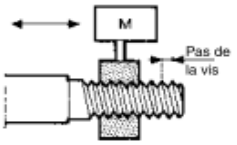
$$J = Mr^2 + \frac{mr^2}{2}$$



M = Masse en translation
m = Masse pignon

Dispositif vis/écrou

$$J = \frac{MP^2}{4\pi^2} + \frac{mr^2}{2}$$



M = Masse en translation
m = Masse de la vis
r = Rayon moyen de la vis

Inertie

Calcul des inerties ramenées au moteur

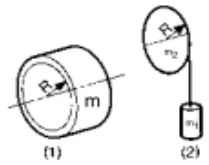
➜ Cylindre

$$J = \frac{mR^2}{2}$$

➜ Jante - Poids/poulie

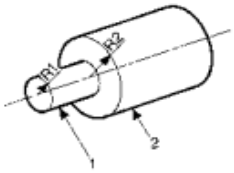
$$J = mR^2$$

$$J = mR^2 + \frac{mR^2}{2}$$



➜ Cylindres coaxiaux (arbres épaulés)

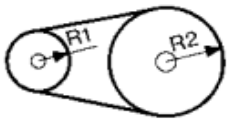
$$J = \frac{M_1R_1^2}{2} + \frac{M_2R_2^2}{2}$$



M1 = Masse du cylindre 1
M2 = Masse du cylindre 2

Transmission pour courroie (ou chaîne)

$$J = \frac{M_1 + 2m + M^2}{2} R1^2$$

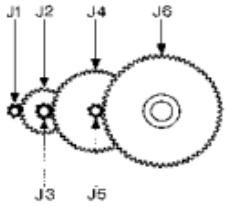


M_1 = Masse poulie moteur
M_2 = Masse poulie menée

m = Masse courroie

Si la poulie menée reçoit aussi le moment d'inertie Jc d'une charge on a alors :

$$J = \frac{M_1 + 2m + M_2}{2} R_1^2 + J_C \left(\frac{R_1}{R_2}\right)^2$$



Cas d'un réducteur

$$J = \frac{1}{R^2} J_c + J_r$$

J_c = Inertie de la charge entraînée en sortie du réducteur
J_r = Inertie du réducteur
R = Rapport de réducteur

➜ Nota :

L'inertie du réducteur se calcule étage par étage, chaque roue étant considérée comme un cylindre.

$$J_r = J_1 + \left(\frac{1}{R_1}\right)^2 (J_2 + J_3) + \left(\frac{1}{R_1}\right)^2 (J_4 + J_5) + \ldots\ldots$$

En pratique le calcul de l'inertie des 2 premiers trains, voire très souvent du premier donne une valeur approchée suffisante.

6- DATASHEET ULN2803 (1ere page seulement) [27]

TOSHIBA ULN2803,04APG/AFWG

TOSHIBA Bipolar Digital Integrated Circuit Silicon Monolithic

ULN2803APG,ULN2803AFWG,ULN2804APG,ULN2804AFWG
(Manufactured by Toshiba Malaysia)

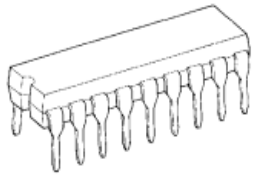
8ch Darlington Sink Driver

The ULN2803APG / AFWG Series are high−voltage, high−current darlington drivers comprised of eight NPN darlington pairs.
All units feature integral clamp diodes for switching inductive loads.
Applications include relay, hammer, lamp and display (LED) drivers.

Features

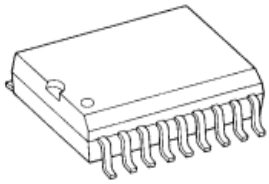
- Output current (single output)
  500 mA (max)
- High sustaining voltage output
  50 V (min)
- Output clamp diodes
- Inputs compatible with various types of logic.
- Package Type−APG : DIP−18pin
- Package Type−AFWG : SOL−18pin

ULN2803APG
ULN2804APG



DIP18-P-300-2.54F

ULN2803AFWG
ULN2804AFWG

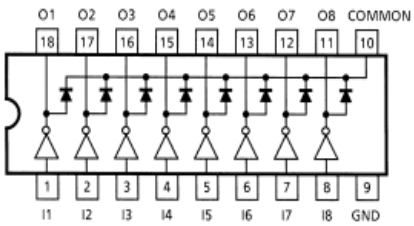


SOL18-P-300-1.27

Weight
DIP18−P−300−2.54F: 1.478 g (Typ.)
SOL18−P−300−1.27 : 0.48 g (Typ.)

Type	Input Base Resistor	Designation
ULN2803APG / AFWG	2.7 kΩ	TTL, 5 V CMOS
ULN2804APG / AFWG	10.5 kΩ	6~15 V PMOS, CMOS

Pin Connection (top view)



7- DATASHEET TSOP [30]

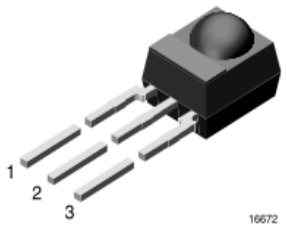
VISHAY

www.vishay.com

TSOP22.., TSOP24.., TSOP48.., TSOP44..

Vishay Semiconductors

IR Receiver Modules for Remote Control Systems



FEATURES

- Low supply current
- Photo detector and preamplifier in one package
- Internal filter for PCM frequency
- Improved shielding against EMI
- Supply voltage: 2.5 V to 5.5 V
- Improved immunity against ambient light
- Insensitive to supply voltage ripple and noise
- Material categorization:
  For definitions of compliance please see www.vishay.com/doc?99912

Pb-free

e3

RoHS COMPLIANT

HALOGEN FREE

GREEN (5-2008)

MECHNICAL DATA

Pinning for TSOP44.., TSOP48..:

1 = OUT, 2 = GND, 3 = V_S

Pinning for TSOP22.., TSOP24..:

1 = OUT, 2 = V_S, 3 = GND

DESCRIPTION

The TSOP22.., TSOP48.., TSOP24.. and TSOP44.. series are miniaturized IR receiver modules for infrared remote control systems. A PIN diode and a preamplifier are assembled on lead frame, the epoxy package contains an IR filter.

The demodulated output signal can be directly connected to a microprocessor for decoding.

The TSOP24.., TSOP44.. are optimized to suppress almost all spurious pulses from energy saving lamps like CFLs. The AGC4 used in the TSOP24.. and TSOP44.. may suppress some data signals. The TSOP22.., TSOP48.. are legacy products for all common IR remote control data formats. Between these four receiver types, the TSOP24.., TSOP44.. are preferred. Customers should initially try the TSOP24.., TSOP44 in their design.

These components have not been qualified according to automotive specifications.

PARTS TABLE

AGC		LEGACY, FOR LONG BURST REMOTE CONTROLS (AGC2)		RECOMMENDED FOR LONG BURST CODES (AGC4) (1)	
Carrier frequency	30 kHz	TSOP4830	TSOP2230	TSOP4430	TSOP2430
	33 kHz	TSOP4833	TSOP2233	TSOP4433	TSOP2433
	36 kHz	TSOP4836	TSOP2236	TSOP4436 (2)(3)(4)	TSOP2436 (2)(3)(4)
	38 kHz	TSOP4838	TSOP2238	TSOP4438 (5)(6)(7)	TSOP2438 (5)(6)(7)
	40 kHz	TSOP4840	TSOP2240	TSOP4440	TSOP2440
	56 kHz	TSOP4856	TSOP2256	TSOP4456 (7)(8)	TSOP2456 (7)(8)
Package		Mold			
Pinning		1 = OUT, 2 = GND, 3 = V_S	1 = OUT, 2 = V_S, 3 = GND	1 = OUT, 2 = GND, 3 = V_S	1 = OUT, 2 = V_S, 3 = GND
Dimensions (mm)		6.0 W x 6.95 H x 5.6 D			
Mounting		Leaded			
Application		Remote control			
Best remote control code		(2) RC-5 (3) RC-6 (4) Panasonic (5) NEC (6) Sharp (7) r-step (8) Thomson RCA			

Note

(1) We advise try AGC4 first if the burst length is unknown.

Bibliographie

[1] - Datasheet atmega8, http://www.atmel.com/images/atmel-2486-8-bit-avr-microcontroller-atmega8_l_datasheet.pdf

[2] - http://fr.wikipedia.org/wiki/Intel_4004

[3] - Christian Dupaty, Systèmes à Microcontrôleurs V1.1,Ecole Nationale Supérieure des mines SAINT-ETEINNE

[4] - Jérôme VICENTE, Les microcontrôleurs, ver 4.0, École Polytechnique Universitaire de Marseille Dpt ME - Otion SIIC 2 ème, année 2005-2006

[5] - Microcontrôleur ATMEL ATMEGA www.reality.be/elo/labos2/files/ AtMega32DocFr.pdf

[6] - http://shop.rabtron.co.za/catalog/atmega8-p-3029.html

[7] - http://www.futurlec.com/Atmel/ATMEGA8.shtml

[8] - http://forum.arduino.cc/index.php?action=dlattach;topic=100906.0;attach=16634

[9] - Moteur pas à pas et PC 2.Eme, Edition, Patrice OGUIC, Edition Dunod, Paris 2004.

[10] - S.CHARI, moteur pas à pas, http://idrissicours.freevar.com/ idrissicours/ 2STE/ADC/Convertir/covertir3p-%20%20www.idrissicours.on.ma.pdf

[11] - Moteur pas à pas, http://fr.wikipedia.org/wiki/Moteur_pas_%C3%A0_pas

[12] - ZENATI A/Hakim & KERROUCHE Samir ,Commande de deux moteurs pas à pas via le port parallèle avec Delphi,Université Abderrahmane Mira de Bejaia,2008/2009

[13] - IDDIR Hayet, Etude et réalisation d'un kit électronique du type système embarque à base de microcontrôleur application au contrôle d'un bras de robot à 4 degrés de libertés, Université M'Hamed BOGUERRA Boumerdes, juin 2013

[14] - http://meteosat.pessac.free.fr/Cd_elect/mot_pas_a_pas/motpap.htm

[15] - http://www.lycee-ferry-versailles.fr/si-new/3_2_convertir/moteurs_elec/motorisation_gwenola_eleve_2.pdf

[16] - http://gdumenil.free.fr/gfichier/cira/cours/moteurpaspas.pdf

[17] - ALEXANDRE CORNET Ciney, MOTEUR PAS A PAS,www.meltinfo.com/pdf/le-moteur-pas-à-pas

[18] - http://www.technologuepro.com/TP-miniprojet-electronique/miniprojet-5-Commande-un-moteur-pas-a-pas.pdf

[19] - Jean-Marc Delaplace, Le moteur pas à pas, le 12 novembre 14

[20] - Datasheet M35SP-7, http://vinvin.tf/projects/M35SP-7T.pdf

[21] - http://educypedia.karadimov.info/library/cours_moteur_pas_a_pas.pdf

[22] - Installation studio avr, http://home.roboticlab.eu/fr/examples/setup/windows

[23] - BRUNEAUX Jérôme, Installation des outils de programmation pour le microcontrôleur ATMEL AtMega128 (Guide de programmation de base), INRIA Rhône, Alpes

[24] - LES DIFFÉRENTS LANGAGES DE PROGRAMMATION, http://www.lopr.net/informatique.php?n=8

[25] - MATHIEU Nebra, apprenez à programmer en c.2010

[26] - Datasheet ULN2803, https://www.sparkfun.com/datasheets/IC/uln2803a.pdf

[27] - thomase , TD scanner, 2011, http://home.iooner.me/shake/Cours%20Me%CC%81ca%20-%20Elec%20%5BFrance%5D/Cours%20Terminale/Cours%20TD%20&%20TP%20Elec%20PDF/Se%CC%81quence%205%20MPP/TD%20Scanner/TD%20scanner.pdf

[28] - http://www.festo-didactic.com/int-fr/services/symboles/electrical-engineering-din-en-60617/semi-conducteurs-et-tubes-electroniques/optocoupleur-a-fente-pour-detection-d-obturation,represente-avec-obturation-mecanique-barriere-photoelectrique.htm?fbid=aW50LmZyLjU1Ny4xNi4zMi4xMjgxLjczMTU

[29] - http://arduinoguides.blogspot.com/2012/05/tsop-ir-receiver.html

[30] - Datasheet TSOP, www.vishay.com/docs/82459/tsop48.pdf