REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE FERHAT ABBAS DE SETIF
FACULTÉ DE TECHNOLOGIE
DEPARTEMENT D'ELECTROTECHNIQUE

MEMOIRE

Présenté Pour l'obtention du Diplôme de

MAGISTER EN ELECTROTECHNIQUE

Option : Machines et Commandes Electriques

Par

RAISEMCHE Aziz

Thème

Conception et programmation d'une armoire de commande assistée par ordinateur

Soutenu le ... / ... / ... devant le Jury

M. MOSTEFAI	**Prof**	**Université de Sétif**	**Président**
M. KHEMLICHE	**M.C**	**Université de Sétif**	**Rapporteur**
A. KHELLAF	**Prof**	**Université de Sétif**	**Examinateur**
M. ABDELAZIZ	**M.C**	**Université de Sétif**	**Examinateur**
K.E. HEMSAS	**M.C**	**Université de Sétif**	**Examinateur**

Remerciements

Je tiens à remercier mon encadreur : Dr. Mabrouk Khemliche, Maître de Conférences à l'UFAS, pour m'avoir accueilli au sein de son équipe et pour toute sa confiance. Aussi je tiens à le remercier pour la grande liberté qu'il m'a accordée durant ce travail, ainsi que de ses qualités pédagogiques et humaines envers tous ses étudiants.

Mes remerciements vont aussi aux membres de jury qui ont accepté de juger mon travail.

Je tiens particulièrement à présenter mes sincères remerciements au responsable de la station de pompage de la commune de Boutaleb pour leurs aides qui ont contribué au bon déroulement de cette étude.

J'ai sincèrement apprécié durant ces années la bonne et chaleureuse ambiance. Je tiens à saisir cette occasion pour remercier tout le corps des étudiants en Magister et je leur souhaite du succès dans tout ce qu'ils entreprendront.

Que ceux qui se sentent oubliés, trouvent dans cette phrase ma profonde gratitude et mes remerciements pour l'aide et le soutien apporté durant ces années. Je souhaite aussi remercier toutes les personnes qui m'ont nourri durant ma vie par leur savoir leur gentillesse et leur dévouement. Je ne saurai décrire en quelques mots ma gratitude.

Je suis immensément reconnaissant à mes parents qui m'ont soutenu tout au long de ma vie. Je leur dois beaucoup. Qu'ils trouvent dans ce manuscrit toute ma reconnaissance et le signe que je suis enfin arrivé au bout.

Sommaire

Introduction générale

L'eau, en tant que ressource stratégique et vitale pour l'Algérie, nécessite une bonne gestion afin d'optimiser son exploitation. Il est alors indispensable de faire appel à des techniques efficaces de supervision au niveau des installations hydrauliques et de leurs accessoires. En effet la technologie d'enregistrement des données provenant d'une pompe permet de contrôler le fonctionnement et le rendement d'une station de pompage pendant toute l'année.

La pompe représente l'une des éléments les plus importants dans une station de pompage, mais le problème qui se pose c'est que le prix de l'énergie (électricité) est cher pendant les heures de pointe, c'est pour cela qu'il est préférable de procéder au pompage hors de cette période de temps. Notre but est de concevoir un système qui permet la gestion et l'optimisation du fonctionnement des stations de pompage.

L'idée de ce projet est nouvelle, en effet il existe sur le marché des milliers de modèles de cartes de commande, mais cette approche est basé sur l'utilisation des ordinateurs standard que n'import qu'elle d'entre nous à chez lui. D'une part, leurs disponibilités et leur faible coût nous permettra une conception à budget minime, et d'autre part, la puissance et la rapidité d'exécution des processeurs du moment pourra prendre en charge le traitement des données lors d'une communication entre les modules de commandes.

En plus l'originalité de notre étude est d'essayer à partir d'une carte simple et surtout non coûteuse d'optimiser le fonctionnement des ouvrages d'extraction d'eau dans les stations de pompages. Mais aussi notre système permet le stockage des données de consommation durant toute une année. Ces données peuvent être transmises à un PC à l'aide d'un simple câble d'imprimante (LPT/DB25) pour un éventuel traitement externe (traçage des courbes de modulation…).

Le point fort aussi de ce mémoire c'est l'utilisation du plus connus des langages de programmation qui est sans nul doute « le langage C ». Il est utilisé particulièrement par les informaticiens pour la conception des programmes et des logiciels et c'est dommage de ne pas voir ce type de langage intégré dans le programme d'étude en électrotechnique dans la mesure où cela ouvrira des portes pour de nombreuses applications dans la commande programmable.

Ce langage possède une base de donné très riche et facile à manié, certainement plus intéressante à manipuler que les langages assembleur utilisé pour programmer les microcontrôleurs (PIC, Motorola, Atmel, etc…). De ce fait, on démontrera plus tard la simplicité de la programmation avec le langage C cela en réalisant une application typique de commande par PC pour un simple démarrage statorique d'un moteur asynchrone triphasé. On notera que cet exemple à été l'objet d'une participation durant un congrès international en informatique appliquée à Bordj Bou Arreridj [ICAI09].

Maintenant que nous avons présenté l'idée générale de notre projet, nous allons présenter la démarche que nous avons envisagée en indiquant le contenu des différentes parties qui constituent cet ouvrage : Le premier chapitre consistera, pour un premier temps, à faire une présentation des différents éléments utilisés dans les commandes industrielles. Dans le deuxième chapitre nous commencerons par la présentation globale des principaux commande en C qui nous permettrons de réaliser notre programme de commande par la suite une bonne partie sera consacrée pour décrire l'exemple précédemment citée ainsi que l'étude de la procédure de conception de notre système pour un bon fonctionnement des pompes en vue de l'optimiser. Dans le troisième chapitre nous attaquerons la partie simulation du système, la démarche se veut avant tout progressive. Nous partirons donc des montages de base des différents modules qui ont été présentés dans le schéma synoptique puis nous allons essayer de les améliorer suivant les simulations et les expériences que nous décrirons dans cette partie.

Ainsi, le premier et le deuxième chapitre qui ont pour but de se familiariser avec les systèmes de commande et de programmation constituent la base puisqu'ils sont indispensables à la compréhension du montage. Dans l'annexe nous décrirons la réalisation pratique, cette présentation débutera par le traçage du schéma électrotechnique et des circuits imprimés ainsi que la programmation.

Chapitre I

Introduction à la Commande Industrielle des Moteurs

1 - Introduction :

La commande industrielle désigne l'ensemble des méthodes qui permettent de contrôler les performances d'un appareil électrique, d'une machine ou d'un système. Appliquée aux moteurs, la commande industrielle contrôle le démarrage, l'accélération, le sens de rotation, la vitesse, la décélération et l'arrêt des parties tournantes.

Dans ce chapitre, nous étudierons la commande industrielle des moteurs (Continus /Alternatifs). On se limitera à l'étude des circuits élémentaires car le montage industriel est souvent trop élaboré pour permettre une représentation simple et des explications faciles. Toutefois, les principes de base que nous examinons s'appliquent à tout système de commande, quelque soit sa complexité.

2 - Dispositifs Industriels de Commande :

Tout circuit de commande comprend quelques composants de base raccordés entre eux de façon à assurer le contrôle désiré du moteur. Leurs dimensions peuvent varier selon la grosseur du moteur à commander, mais leur principe de fonctionnement reste le même. Avec seulement une dizaine de dispositifs de base, on réalise des montages de commande très complexes [1]. Voici les principaux dispositifs :

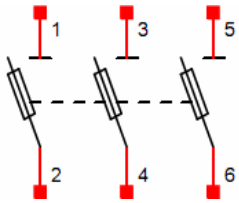
1. Sectionneurs
2. Disjoncteurs manuels
3. Commutateurs à cames
4. Boutons poussoirs
5. Relais de phase
6. Relais thermiques
7. Contacteurs magnétiques
8. Temporisateur
9. Lampes témoins
10. Interrupteurs de fin de course
11. Divers (Résistances, réactances, transformateurs, etc…)

2.1 - Sectionneurs :

Les sectionneurs isolent le circuit du moteur de celui de la source. Ils doivent pouvoir supporter indéfiniment le courant nominal ainsi que les courants de court-circuit pendant de courtes périodes. Ils comportent des contacts à couteaux et des fusibles. Ils s'ouvrent et se ferment manuellement.

Sectionneur tétrapolaire
à fusible

Sectionneur tripolaire
à commande rotative



Symbole

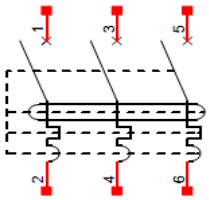
Figure 1.1 - Différents types de sectionneurs

2.2 - Disjoncteurs manuels :

Les disjoncteurs sont conçus pour ouvrir et fermer manuellement le circuit d'un moteur et pour ouvrir le circuit automatiquement si le courant dépasse une limite prédéterminée. On peut réenclencher le disjoncteur après une ouverture anormale. Souvent, on utilise le disjoncteur manuel au lieu d'un sectionneur [1].

Disjoncteur différentiel
bipolaire

Interrupteur tripolaire
de puissance



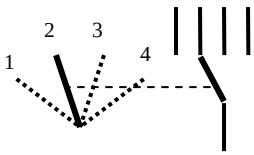
Symbole

Figure 1.2 - Différents types de disjoncteurs

2.3 - Commutateurs à cames :

Ces commutateurs comprennent une série de contacts fixes et autant de contacts mobiles actionnés par la rotation manuelle d'un arbre à cames. On les utilise pour la commande manuelle des moteurs de grues, calandres, pompes, etc.

Différent commutateur à cames



Symbole

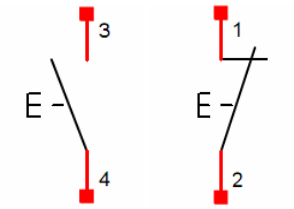
Figure 1.3 - Différents types de commutateurs

2.4 - Boutons poussoirs :

Les boutons-poussoirs sont des commutateurs actionnés par une pression du doigt et qui ouvrent ou ferment deux ou plusieurs contacts. Habituellement, ils ouvrent ou ferment momentanément un circuit.

Bouton d'arrêt d'urgence

Boutons poussoirs



NO NF

Symbole

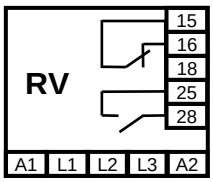
Figure 1.4 - Différents types de boutons poussoirs

2.5 - Relais de phase :

C'est un organe de commande qui se déclenche en cas de coupure d'une des trois phases ou dans le cas d'un déphasage prolongé. Dans un réseau triphasé, il surveille simultanément l'ordre des phases, l'absence d'une phase avec un taux de régénération maximum soit 70% de la tension affichée en face avant par le potentiomètre, la baisse symétrique en tension des 3 phases inférieure à 20 % de la valeur préréglée. Lorsque les 3 phases sont en ordre direct, le relais de sortie est excité et visualisé par une LED. Le relais de sortie retombe (LED éteinte) après une temporisation T, réglable en face avant de 0,2 à 10 s.

Relais de phases Asymétriques

Relais d'ordre de phases



Symbole

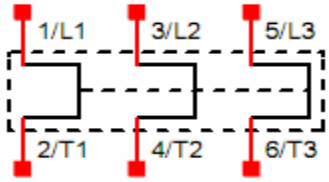
Figure 1.5 - Différents types de relais de phases

2.6 - Relais thermiques :

Les relais thermiques (ou relais de surcharge) sont des dispositifs de protection dont les contacts s'ouvrent ou se ferment lorsque la chaleur créée par le passage d'un courant dépasse une limite prédéterminée. Leur fonctionnement est temporisé car la température ne peut pas suivre instantanément les variations du courant [1].

Relais thermique tripolaire

Relais thermique électronique

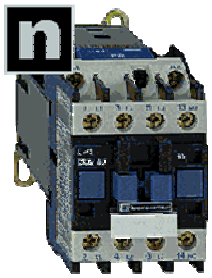


Symbole

Figure 1.6 - Différents types de relais thermiques

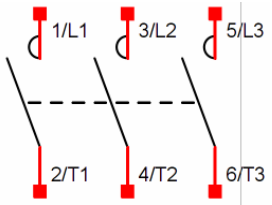
2.7 - Contacteurs magnétiques :

Les contacteurs magnétiques sont de gros relais destinés à ouvrir et à fermer un circuit de puissance. On les utilise dans la commande des moteurs dont la puissance est entre 0,5 kW et plusieurs centaines de kilowatts. Comme pour les moteurs, la grosseur et les dimensions principales des contacteurs sont standardisées par les organismes de normalisation [1].



Contacteur tripolaire

Contacteur tétrapolaire



Symbole

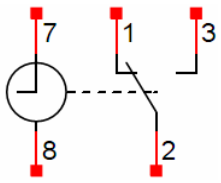
Figure 1.7 - Différents types de contacteurs

2.8 - Temporisateur :

Le temporisateur électrique est un composant souvent utilisé dans l'industrie pour ajouter un délai dans un circuit électrique. Ce délai est parfois nécessaire pour retarder le départ ou l'arrêt d'un moteur. Le temporisateur électrique est en effet un relais auquel on peut retarder les effets. Le réglage du délai se fait à l'aide d'un bouton situé sur le temporisateur.

Temporisateur analogique
(Temporisation de 0,05 sec à 300 heures)

Temporisateur numérique
(Temporisation programmable)



Symboles

Figure 1.8 - Différents types de temporisateurs

2.9 - Lampes témoins :

Les lampes témoins servent à indiquer l'état d'un système de contrôle. Ils servent comme indicateur pour l'opérateur, ils peuvent :

- Affiche plusieurs couleurs par LED extrêmement visibles sur chaque voyant
- Plusieurs configurations pour des applications différentes
- Affiche une à cinq couleurs, en fonction du modèle
- Compatible avec les sorties vers automates programmables ou autres commandes logiques

Il est très important d'intégrer des voyants dans un système de commande et plus encore il est important de connaitre le code des couleurs sur une armoire, cela pour faciliter l'analyse du fonctionnement et surtout en cas de défaillance, le tableau suivant nous montre les différents codes de couleur des voyants :

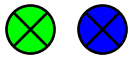
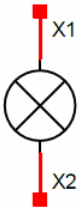
Vert/Bleu 	Moteur en marche (en service)
Rouge 	Moteur à l'arrêt (hors service)
Jaune/Orange 	Défaut (Pb d'une phase, surintensité,…)
Blanc 	Présence du courant (armoire alimentée)

Tableau 1.1 – Code des couleurs pour les voyants lumineux

Voyants bleus

Voyants jaunes



Symbole

Figure 1.9 - Différents types de voyants

3 - Les Schémas en Électrotechnique :

3.1 - Respect de la normalisation :

Le schéma électrique est un moyen de représentation des circuits et d'installations électriques, c'est donc un langage qui doit être compris par tous les électriciens [10]. Des

règles de représentation sont à respecter, elles sont répertoriées dans des normes nationales, européennes et internationales :

Monde → ISO (CEI/CEI 439-1)

Europe → CEN (CENELEC/EN50439-1)

France → AFNOR (UTE/NF EN 50439-1)

3.2 - Quelques règles essentielles sur les schémas :

Un schéma électrique représente à l'aide de symboles graphiques, les différentes parties d'un réseau, d'une installation et d'un équipement qui sont reliées et connectées fonctionnellement. Un schéma électrique à pour but d'expliquer le fonctionnement de l'équipement, fournir les bases d'établissement des schémas de réalisation et faciliter les essais et la maintenance.

3.2.1 - Principe de marquage des bornes :

Dans les représentations schématiques on doit suivre des normes pour faciliter la lecture des schémas pour cela on doit respecter ce qui suit :

- Le marquage des diverses bornes d'un même circuit doit caractériser leur appartenance au même circuit.
- Le marquage des bornes des circuits d'alimentation (bobines, impédances …) doit être alphanumérique.
- Le marquage des bornes des éléments de contacts doit être numérique.
- Chaque marquage ne doit apparaître qu'une seule fois.

Pour un élément simple : Les deux extrémités d'un élément simple sont distinguées par des nombres de référence successifs, par exemple 1 et 2. S'il existe des points intermédiaires à cet élément, on les distingue par des nombres supérieurs en ordre croissant à ceux des extrémités.

Pour un groupe d'éléments : Pour un groupe d'éléments semblables, les extrémités des éléments seront désignées par des lettres de référence. Exemple : U, V, W pour les phases d'un système alternatif triphasé.

3.2.2 - Principe de marquage des contacts :

Contacts principaux : Les bornes sont repérées par un seul chiffre de 1 à 6 (tripolaire), de 1 à 8 (tétrapolaire).

Contacts auxiliaires : Ils sont repérés par un nombre de deux chiffres. Le chiffre des unités indique la fonction du contact :

1-2, contact à ouverture ;

3-4, contact à fermeture ;

5-6, 7-8, contact à fonctionnement spécial.

3.2.3 - Principe d'identification des éléments :

Chaque gamme d'élément se rapporte à un repère (lettre) selon le tableau suivant :

DESIGNATION	REPERE	DESIGNATION	REPERE	DESIGNATION	REPERE
- alternateur - batterie d'accumulateur - générateur	**G**	Appareil mécanique de connexion pour circuit de puissance : - sectionneur - disjoncteur - interrupteur-sectionneur	**Q**	Avertisseurs - lumineux - sonore	**H**
Appareil mécanique de connexion pour circuit de commande : - interrupteur-sélecteur - bouton poussoir - commutateur - interrupteur de position	**S**	Les contacteurs et relais auxiliaires : - les contacteurs et relais auxiliaires temporisé - auxiliaires et contacteurs à accrochage - relais polarisé	**KA**	Condensateurs	**C**
Transformateur	**T**	Contacteur de puissance	**KM**	Moteur	**M**
Dispositif de protection : - coupe-circuit à fusible - parafoudre - relais thermique - relais magnétique - relais magnétothermique	**F**	Appareil mécanique actionné électriquement : - électro-aimant - embrayage - frein électromécanique	**Y**	- résistance - shunt - thermistance - potentiomètre - résistance variable - varistance	**R**

Tableau 1.2 – Repère des éléments d'un schéma

3.3 - Types de schémas :

Afin de faciliter l'étude et la réalisation des schémas, puis pour l'exécution et le dépannage de l'installation, l'électrotechnicien a besoin de différents schémas, dont :

1 - le diagramme synoptique

2 - le diagramme unifilaire

3 - le diagramme schématique

4 - le diagramme des connexions

3.3.1 - Diagramme synoptique ("block diagram") :

Le diagramme synoptique comprend un groupe de rectangles représentant chacun un dispositif de commande et comprenant une courte description de sa fonction. On réunit les rectangles par des flèches pour indiquer la direction de la puissance électrique.

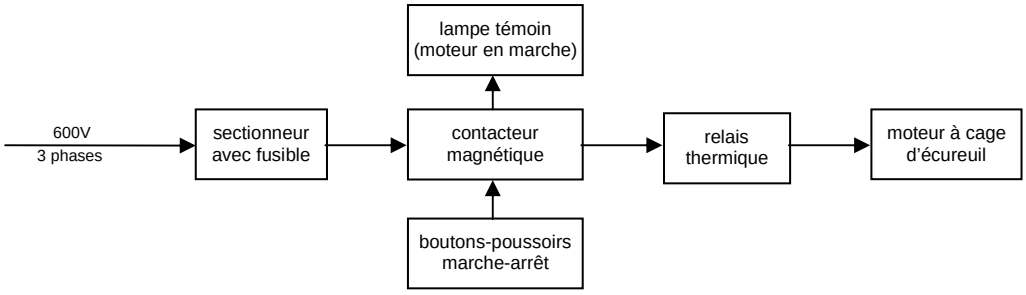


Figure 1.10 – Diagramme synoptique d'une MAS à cage

3.3.2 - Diagramme unifilaire ("one-line diagram") :

Le diagramme unifilaire est semblable au diagramme synoptique sauf que les composants sont représentés pat leur symbole plutôt que par des rectangles. Les symboles donnent une idée de la nature des composants de sorte que le diagramme unifilaire révèle plus d'information. Une seule ligne relie les divers composants quelque soit le nombre de conducteurs réellement utilisés [1].

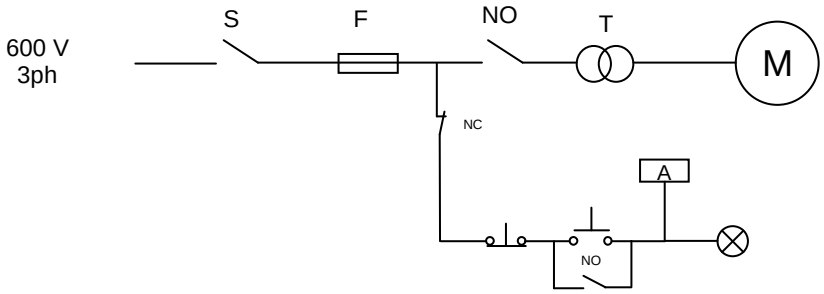


Figure 1.11 – Diagramme unifilaire d'une MAS à cage

3.3.3 - Diagramme schématique ("schematic diagram") :

Le diagramme schématique montre toutes les connexions électriques entre les composants sans respecter leurs positions respectives ni la disposition de leurs bornes. Ce genre de diagramme est indispensable quand on doit localiser un défaut dans un circuit de commande, ou quand il faut connaître son fonctionnement en détail.

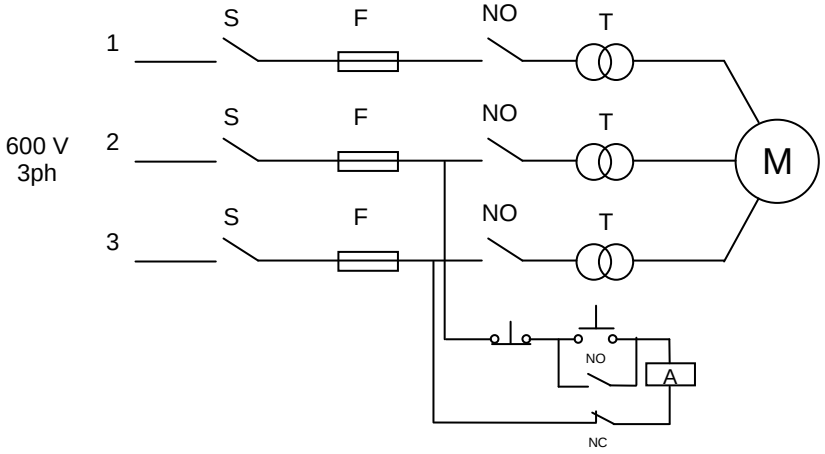


Figure 1.12 – Diagramme schématique d'une MAS à cage

3.3.4 - Diagramme des connections ("wiring diagram") :

Le diagramme des connexions montre les connexions entre les composants en tenant compte de la disposition physique des bornes et parfois de la couleur des fils. On utilise ces diagrammes lors de l'installation ou quand il faut identifier les fils pour localiser une panne, par exemple [1].

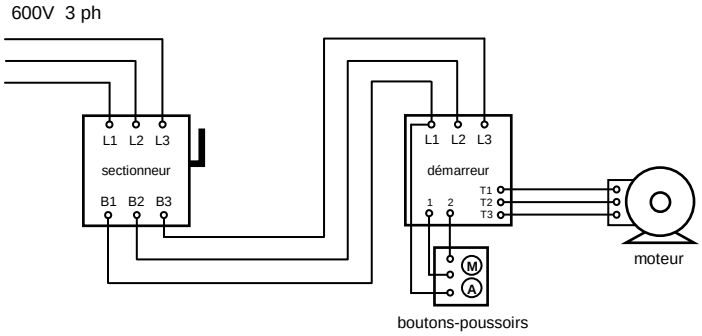


Figure 1.13 – Diagramme des connexions d'une MAS à cage

3.3.5 - Diagramme développé :

Il est généralement utilisé pour représenter la partie commande d'une installation. Les symboles des différents éléments d'un même appareil sont séparés et disposés de manière à ce que le tracé de chaque circuit puisse être facilement suivi. Dans le cas de représentation d'une commande, les éléments du diagramme comportent un repère (§ 3.3.2) qui est lié à la partie puissance de la même installation, on n'aura qu'à faire la liaison pour comprendre le parcours global du schéma de puissance et sa commande [12].

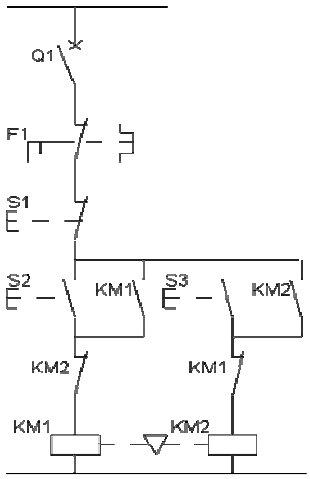


Figure 1.14 – Diagramme développé d'une MAS à cage

4 - Démarrage Industriel des Moteurs :

Plusieurs procédés industriels requièrent un démarrage lent afin d'atteindre graduellement le régime de fonctionnement normal. Dans d'autres cas, on ne peut pas brancher un moteur directement sur la ligne car le courant de démarrage risque de causer une chute de tension inacceptable non seulement pour l'usager mais aussi pour ses voisins

raccordés sur la même ligne. Dans ces circonstances, il faut limiter le couple de démarrage ou le courant de démarrage en réduisant la tension aux bornes du moteur [1].

On trouve plusieurs procédés de démarrage, les plus courants sont :

1. Démarrage Direct
2. Démarrage Statorique
3. Démarrage Rotorique
4. Démarrage Étoile Triangle
5. Démarrage par Autotransformateur

4.1 - Démarrage Direct :

C'est un procédé de démarrage simple mais brutal, obtenu en un seul temps, le stator du moteur est couplé directement sur le réseau [2]. Le couple est énergique, l'appel de courant est important (5 à 8 fois le courant nominal) [4].

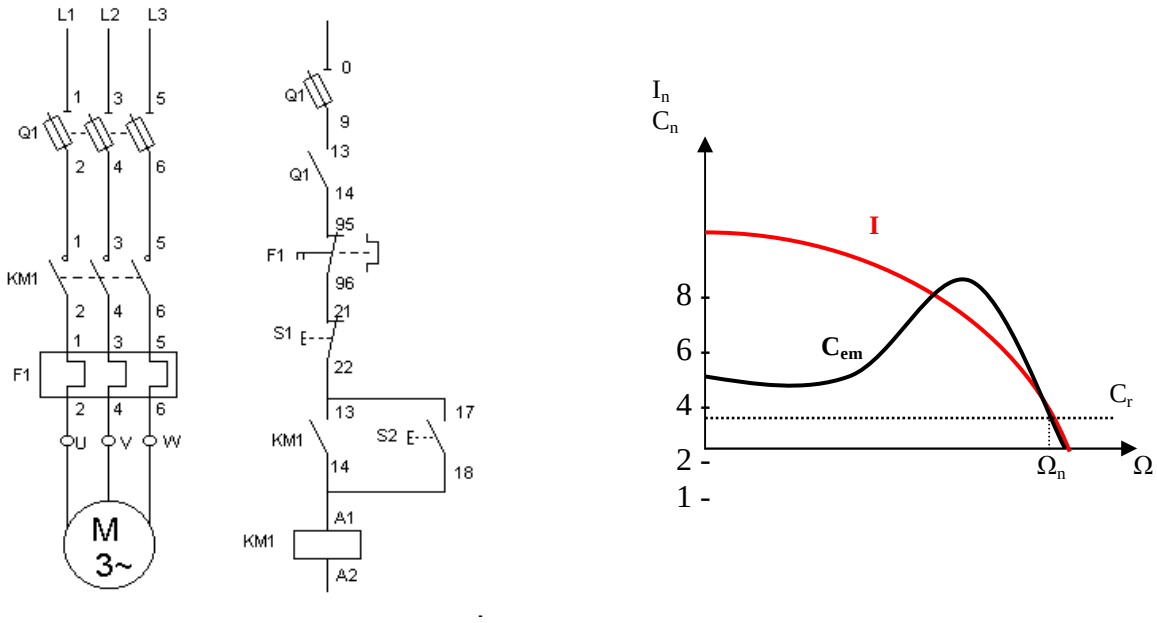


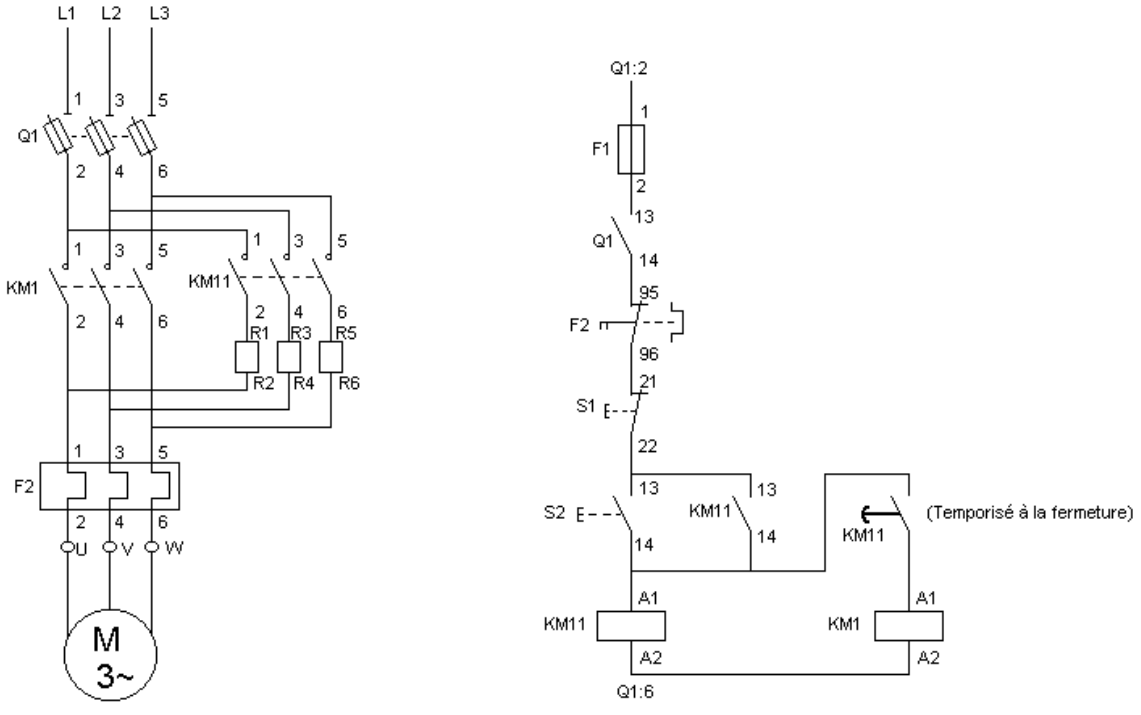
Figure 1.15 – Schéma d'un démarrage direct

Une impulsion sur S2 alimente le relais (KM1) : les contacteurs KM1 se ferment et le relais est auto-alimenté → le moteur démarre, l'arrêt est obtenu par une impulsion sur S1. Le moteur démarre sur ses caractéristiques "naturelles". Au démarrage, le moteur se compose comme un transformateur dont le secondaire (rotor) est en court-circuit, d'où la pointe de courant au démarrage [6].

Avantages	**Inconvénients**	**Utilisations**
- Simple - Économique - C_d = 1,5 à 2 C_n - Démarrage rapide	- Démarrage brutal - I_d = 4 à 8 I_n	- Machine de petite et moyenne puissance (P < 5 kW) - Démarrage des compresseurs - Motopompes

4.2 - Démarrage Statorique :

Plusieurs dispositifs permettent de réduire la tension aux bornes des enroulements du stator pendant la durée du démarrage du moteur ce qui est un moyen de limiter l'intensité du courant de démarrage. Cette réduction se fait en intercalant des potentiomètres ou des résistances (rhéostats) en série avec les enroulements du stator, ou en utilisant un autotransformateur [1], ou dans certain cas, par des systèmes à base de semi conducteurs (Gradateur, redresseur commandée,…). Le courant de démarrage est limité, sans que le couple ne soit réduit.



La figure suivante montre un schéma démarrage statorique :

Figure 1.16 – Schéma d'un démarrage statorique

Fonctionnement de la partie puissance :

Le démarrage s'effectue en 2 temps et dure entre 2 et 3s :

1er temps → on met en série avec chaque phase du stator une résistance (fermeture de KM11)

2ème temps → on court circuite les résistances (fermeture de KM1)

Fonctionnement de la partie commande :

1er temps → Impulsion sur S2 : le relais KM11 est activé et les contacteurs KM11 (partie puissance) se ferment. Le relais est auto-alimenté.

2ème temps → Le contacteur temporisé KM11 se ferme, entraînant l'alimentation du relais KM1 : Les contacteurs de puissance KM1 court-circuitent les résistances.

* L'arrêt est obtenu par une impulsion sur S1

Avantages	Inconvénients	Utilisations
- Pas de coupure d'alimentation pendant le démarrage - Utilisé pour tout types de moteurs - Si on augmente la tension de 2, le couple augmente de 4.	- Perte de puissance dans les résistances - Faible réduction de courant - Installation volumineuse	- Machines à papier - Machines à textile

4.3 - Démarrage Rotorique :

C'est le même principe que le démarrage statorique sauf que le dispositif de démarrage est relié aux enroulements du rotor, c'est pour ça que le démarrage rotorique nécessite un moteur à bagues (rotor bobiné).

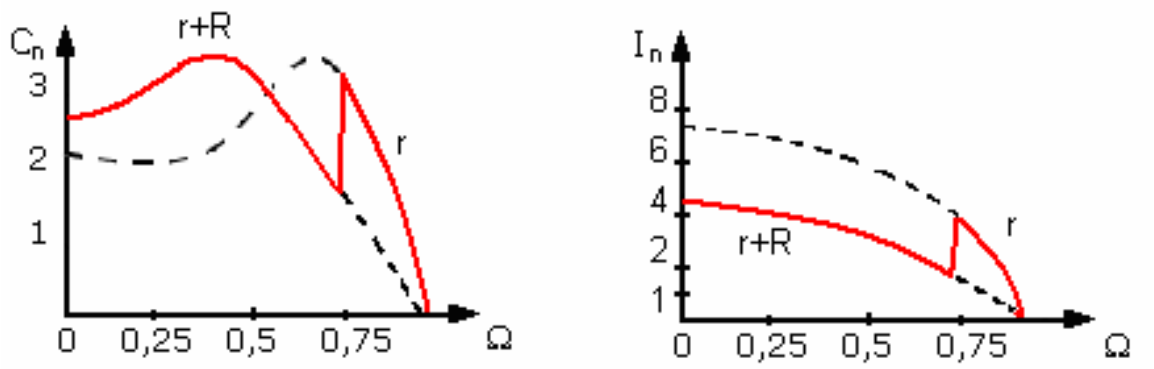


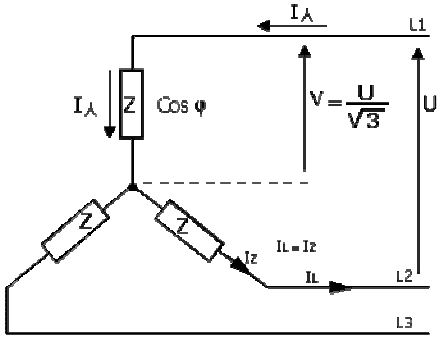
Figure 1.17 – Couple et courant d'un démarrage rotorique

Avantage	Inconvénient	Utilisation
- Démarrage linéaire	- Procédé lent	- Moteur

4.4 - Démarrage Étoile-Triangle :

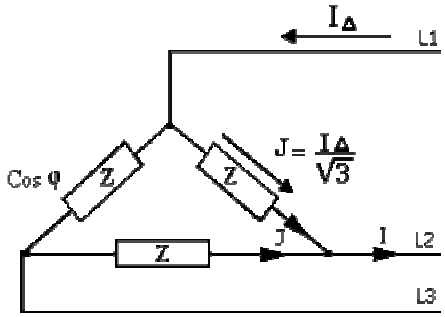
Le démarrage étoile-triangle est très utilisé en électrotechnique pour la mise en marche des moteurs électriques asynchrones triphasés [6]. Son but est de réduire le courant de démarrage par trois, cela en modifiant l'alimentation des enroulements du moteur, du montage étoile au montage triangle, comme le montre la figure :

1ère temps – Montage Étoile



$I_Y = \frac{\sqrt{3}}{3}\frac{U}{Z}$	$P_Y = \frac{U^2 \cos\varphi}{Z}$

2ème temps – Montage Triangle



$I_\Delta = \sqrt{3}\frac{U}{Z}$	$P_\Delta = \frac{3\,U^2 \cos\varphi}{Z}$

Figure 1.18 - Structure d'un démarrage étoile-triangle

Cette propriété est utilisée pour les démarrages des moteurs asynchrones triphasés où au premier temps les enroulements sont couplés en étoile (In et P trois (3) fois plus faible) et au deuxième temps on effectue le couplage triangle. Il en résulte de la même façon que le couple de démarrage en étoile est trois fois plus faible qu'en triangle.

Ce type de démarrage s'applique uniquement au moteur possédant 6 bornes dans la plaque de connexion. En effet, c'est impossible de réaliser ce type de montage avec une plaque à 3 bornes.

La figure suivante montre un schéma de démarrage étoile triangle :

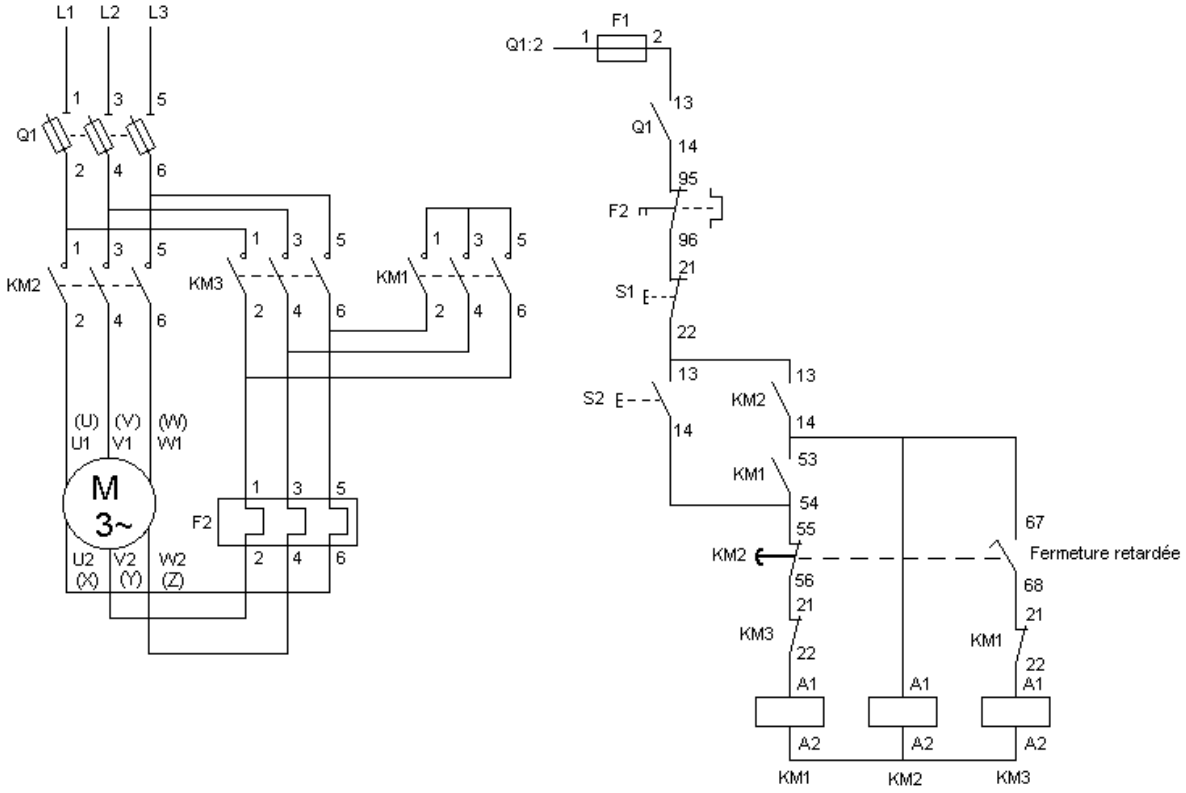


Figure 1.19 – Schéma d'un démarrage étoile-triangle

Le démarrage s'effectue en deux étapes et dure 3 à 7 secondes :

1er temps : couplage Etoile (Y) → Le courant de démarrage I_d est réduit par rapport au démarrage direct. (I_d = 1,3 à 2,6 I_n)

2ème temps : couplage Triangle (Δ) → Quand le moteur est lancé, on passe au couplage triangle. La surintensité qui en résulte est moins importante qu'en démarrage direct et le moteur atteint sa vitesse nominale à pleine tension.

Fonctionnement de la partie commande :

1er temps → Une impulsion sur S2 alimente le relais KM1. Les contacteurs KM1 se ferment et le relais KM2 est activé : il y a auto-alimentation (KM2 13-14 est fermé). Les contacteurs de puissance KM1 et KM2 étant fermés, on a un couplage étoile.

2ème temps → Au bout de 3 secondes, le contacteur à ouverture temporisée (KM2 55-56) s'ouvre, entraînant avec un léger retard la fermeture du contacteur 67-68 : Le relais KM3 est alors alimenté. Les contacteurs KM2 et KM3 sont donc fermés : c'est le couplage Triangle. L'arrêt est obtenu par une impulsion sur S1.

L'évolution du courant et du couple est comme suit :

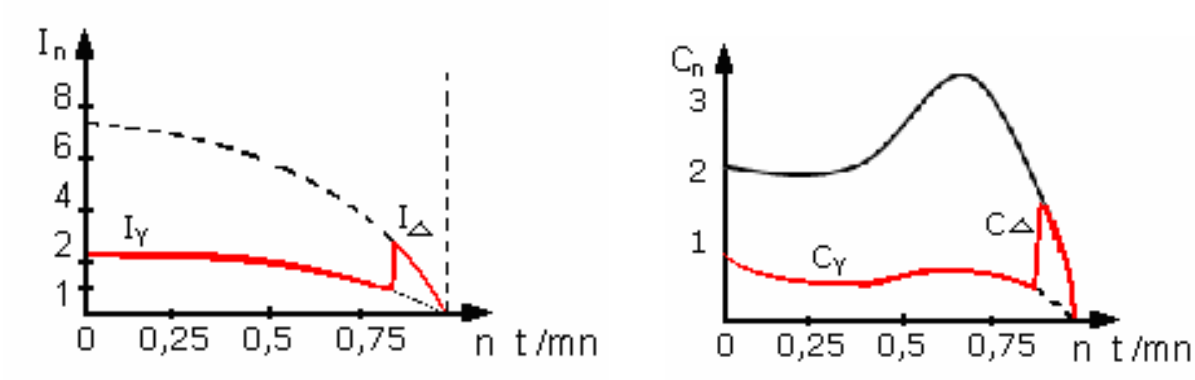


Figure 1.20 – Couple et courant d'un démarrage étoile triangle

Lors du couplage étoile, chaque enroulement est alimenté sous une tension 3 fois plus faible, de ce fait, le courant et le couple sont divisés par 3. Lorsque les caractéristiques du courant ou du couple sont admissibles, on passe au couplage triangle. Le passage du couplage étoile au couplage triangle n'étant pas instantané, le courant est coupé pendant 30 à 50 ms environ. Cette coupure du courant provoque une démagnétisation du circuit magnétique. Lors de la fermeture du contacteur triangle, une pointe de courant réapparaît brève mais importante (magnétisation du moteur) [5].

Avantages	**Utilisations**
- Bon rapport couple/courant - Réduction importante du courant de démarrage	- Moteurs AC - Machines démarrant à vide - Ventilateurs et pompes centrifuges - Machines outils - Machine à bois
Inconvénients	
- Couple au démarrage faible -Pas de possibilité de réglage de la vitesse - Apparition de phénomènes transitoires au changement du couplage - Puissance limitée (< 75 KW)	

4.5 - Démarrage par Autotransformateurs :

Dans le démarrage par autotransformateur, on effectue le même type que le démarrage étoile triangle (on a en plus le choix du rapport des tensions en choisissant le rapport de transformation) mais les phénomènes transitoires du démarrage étoile triangle ne vont plus exister car le courant n'est jamais coupé. Dans un premier temps, on démarre le moteur sur un autotransformateur couplé en étoile. De ce fait, le moteur est alimenté sous une tension réduite réglable. Avant de passer en pleine tension, on ouvre le couplage étoile de l'autotransformateur, ce qui met en place des inductances sur chaque ligne limitant un peu la pointe et presque aussitôt, on court-circuite ces inductances pour coupler le moteur directement au réseau [5].

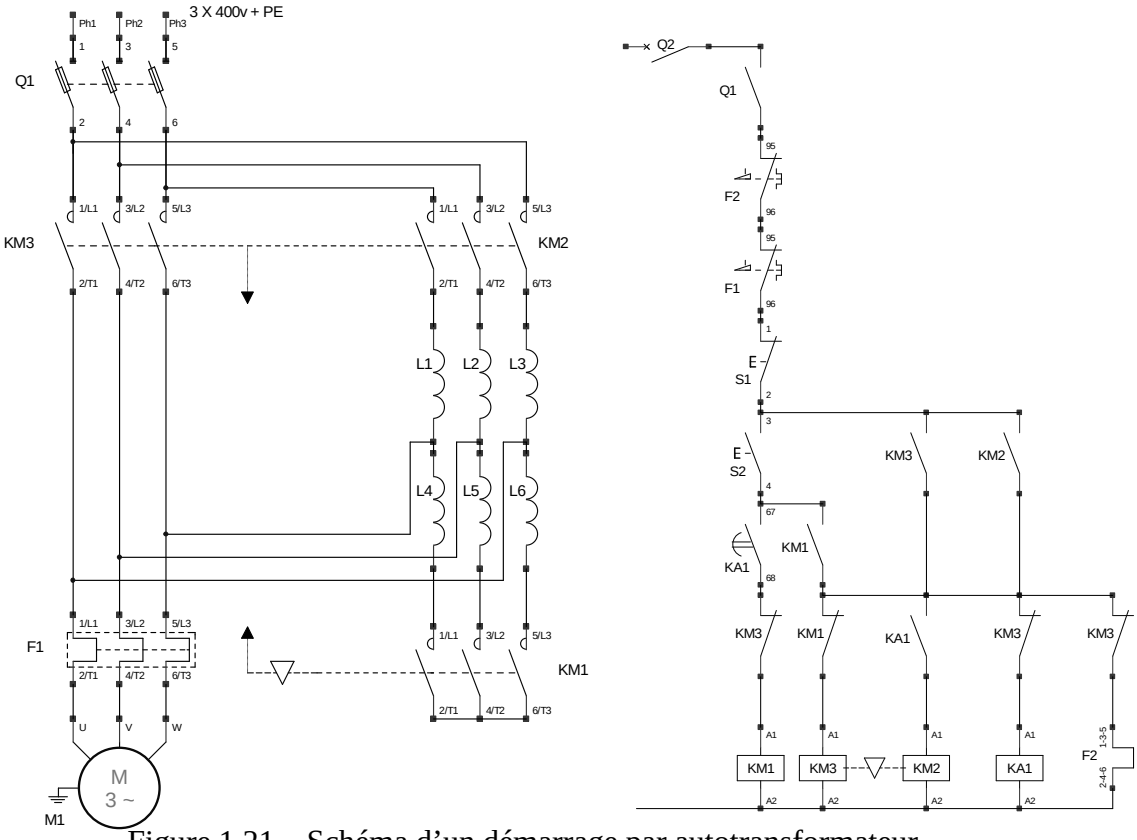


Figure 1.21 – Schéma d'un démarrage par autotransformateur

5 - Freinage Industriel des Moteurs :

Pour arrêter un moteur, on doit appuyer sur le bouton d'arrêt afin de couper l'alimentation. En coupant l'alimentation, la vitesse diminue graduellement sous l'effet des pertes par frottement, le moteur prend un certain temps pour s'arrêter. Dans certaines applications, il faut freiner le moteur rapidement, parmi les méthodes de freinage:

1. Freinage Rhéostatique
2. Freinage par Inversion

3. Freinage par Injection de Courant Continu
4. Freinage par Récupération d'Énergie

5.1 - Freinage rhéostatique (dynamique) :

Le freinage rhéostatique consiste à ouvrir le circuit d'alimentation et de raccorder les enroulements du moteur aux bornes d'une résistance afin que la puissance emmagasinée dans le moteur soit dissipée dans cette résistance. En pratique, on choisit la résistance telle que le courant de freinage, soit environ deux fois le courant nominal (Le moteur fonctionne dans le quadrant 2 ou 4). La tension diminue progressivement au fur et à mesure que le moteur ralentit. Le courant décroît également. Par conséquent, le couple de freinage diminue et s'annule lorsque le moteur cesse de tourner. Pour un freinage plus rapide, on diminue la résistance de freinage R. La figure 1.22 montre un schéma du freinage rhéostatique.

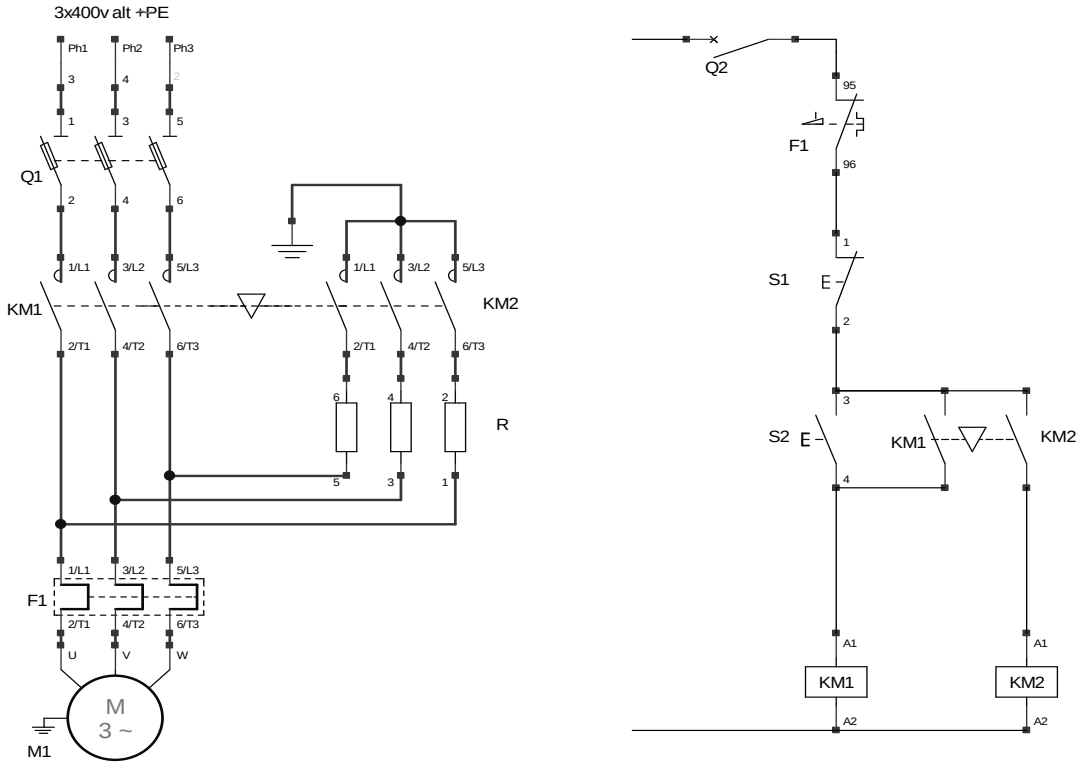


Figure 1.22 – Schéma d'un freinage rhéostatique

En appuyant sur le bouton de marche S2, la bobine du contacteur KM1 enclenche ses contacts de puissance, ce qui alimente le moteur. La résistance de freinage est mise hors circuit par le contact de KM1 branché en série.

En appuyant sur le bouton d'arrêt, les contacts de puissance KM1 s'ouvrent et le contacteur KM2 se ferme. A ce moment, l'induit débite son courant dans la résistance de freinage R. Le moteur s'arrête lorsque le courant s'annule.

5.2 - Freinage par inversion (contre-courant) :

On peut freiner un moteur asynchrone rapidement en changeant le sens de rotation de son champ tournant, cela en permutant deux fils (phases) d'alimentation, ça inverse brutalement le sens du courant dans le stator. Le schéma suivant montre un schéma d'un freinage par Inversion [12].

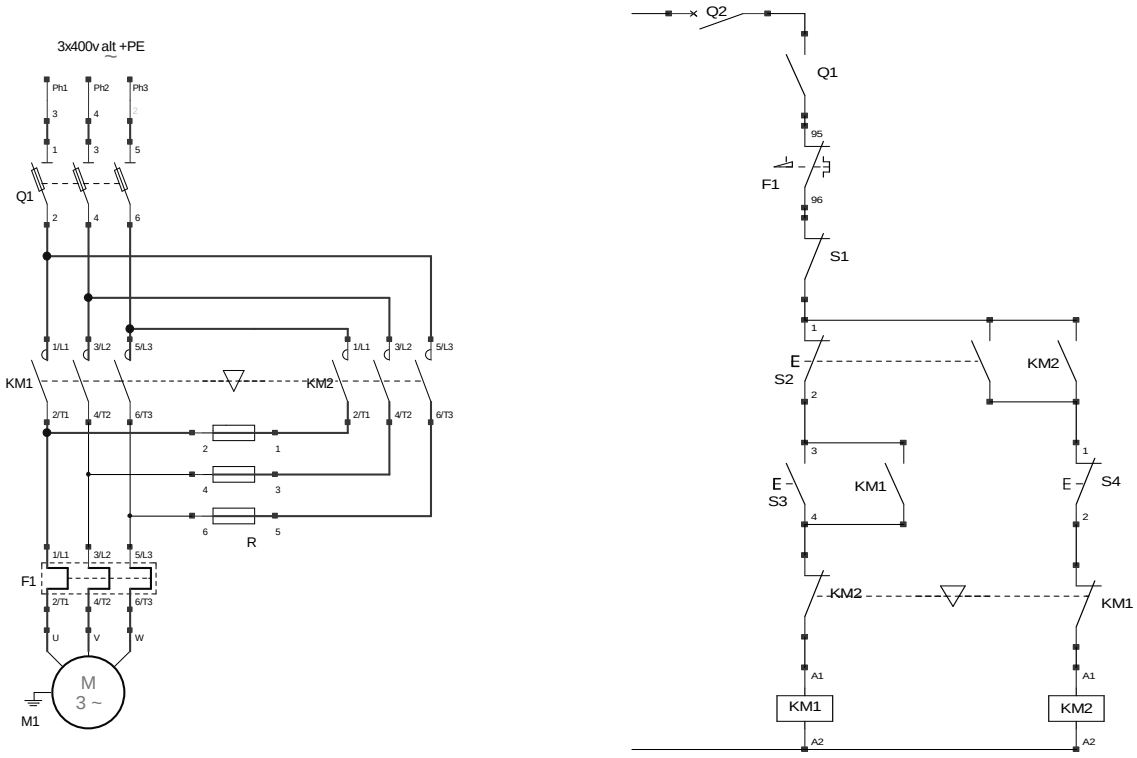


Figure 1.23 – Schéma d'un freinage par inversion

Lors du freinage, il y a ouverture de KM1 puis fermeture de KM2 : le moteur est alimenté par un champ statorique inverse. Les pointes de courant sont très importantes et il est conseillé d'insérer un jeu de résistances pour limiter ce courant. KM2 doit s'ouvrir dès l'arrêt du moteur, pour éviter un redémarrage en sens inverse : il est donc nécessaire de prévoir un capteur détectant l'absence de rotation (capteur centrifuge) [4].

Il existe plusieurs types de détecteurs de rotation, dont le détecteur du type à induction qui fonctionne sur le principe du moteur asynchrone, ou aussi, le détecteur du type centrifuge qui utilise la force centrifuge créée par la rotation.

5.3 - Freinage par injection de courant continu :

L'idée générale pour freiner un moteur asynchrone est de faire baisser la fréquence d'alimentation, pour le faire travailler en alternateur plutôt qu'en moteur, le cas extrême est l'injection de courant continu (fréquence = 0). On arrive dans ce cas particulier à un alternateur qui a un rendement nul, l'intégralité de la puissance est alors dissipée dans le rotor sous forme de pertes joules. L'alimentation en courant continu du stator crée un champ fixe dans la machine qui s'oppose au mouvement. C'est la méthode la plus efficace pour freiner la machine, mais les contraintes en courant sont également très sévères. Le contrôle de l'intensité du courant continu permet de contrôler le freinage [12].

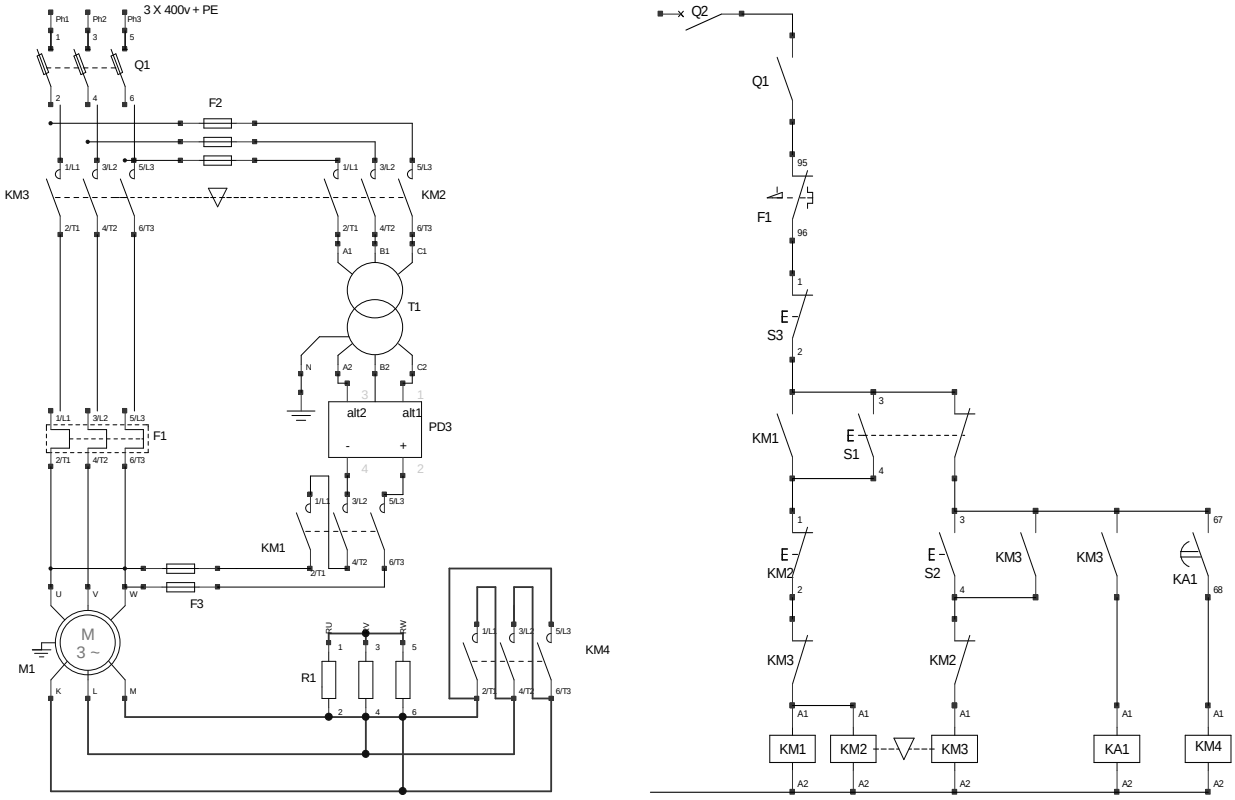


Figure 1.24 – Schéma d'un freinage par injection de courant continu

Lorsque l'ordre de démarrage est donné, KM3 se ferme et le moteur se met en rotation. Lorsque l'arrêt est demandé, KM3 s'ouvre, puis KM1 et KM2 se ferment. Un courant continu (limité par la résistance R1) est envoyé dans le stator. Le moteur se comporte comme un alternateur en court-circuit. L'inducteur, constitué par 2 phases du stator, produit un champ magnétique fixe.

Des courants rotoriques apparaissent : l'énergie cinétique est transformée en pertes par effet Joule au niveau du rotor : le moteur ralentit. Lorsque le moteur est à l'arrêt, KM2 s'ouvre

[9]. Puisqu'en courant continu il n'y a pas d'inductance pour limiter le courant, il faut une tension beaucoup plus faible que la tension normale. Le contacteur envoie 12V ou 24V entre 2 phases d'un moteur 380V pendant quelques secondes : ça freine très bien ! Parfois sur certains moteurs il vaut mieux se contenter de 6V pour ne pas freiner trop brutalement.

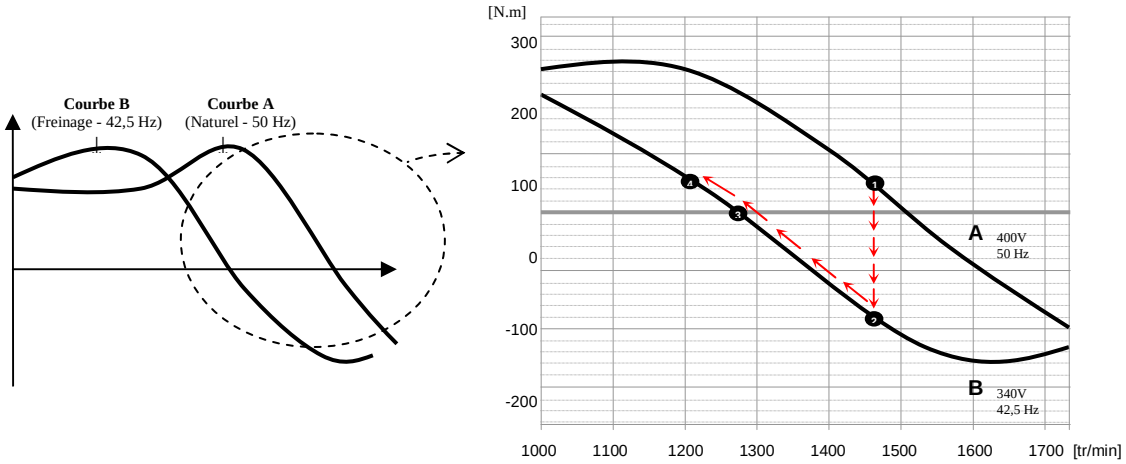
5.4 - Freinage par récupération d'énergie :

Cette méthode est basée sur la dissipation d'une partie de l'énergie de freinage dans le réseau, cette méthode se fait selon l'exemple suivant :

Supposons qu'un moteur soit raccordé à un réseau de 400V, 50Hz. Il entraîne une charge qui exige un couple constant de 50 N.m (point 1). Supposons que l'on réduit brusquement la tension et la fréquence à 85% de leurs valeurs nominales de sorte que la courbe A soit subitement remplacée par la courbe B.

Les nouvelles conditions d'alimentation sont : E = 400x0,85 = 340V, f = 50x0,85 = 42,5Hz.

Comme la vitesse du moteur ne change pas instantanément, l'état de fonctionnement passe du point 1 au point 2. Dans ces conditions, le moteur exerce un couple négatif de 180 N.m, celui-ci s'ajoute au couple de charge (50 N.m), de sorte que le couple total de freinage devient 230 N.m, cela induit logiquement une diminution rapide de la vitesse. À mesure que la vitesse diminue, le couple exercé par le moteur décroît progressivement, en suivant la courbe B. Rendu à 1275 tr/min (point 3), le couple s'annule, mais sa vitesse continue à décroître rapidement à cause du couple imposé par la charge. Après ce point, le moteur développe un couple positif qui augmente progressivement, jusqu'au moment où il devient égale à celui de



la charge (point 4) [1].

Figure 1.25 – Étapes du freinage par récupération d'énergie

Lorsque le point de fonctionnement passe du point 2 au point 3, une partie de l'énergie du rotor est retournée au réseau, car, durant cet intervalle, le moteur fonctionne en génératrice asynchrone. Donc, ce freinage est accompagné d'une certaine récupération d'énergie.

Vue de la diversité des procédés de démarrage et de freinage, on propose le récapitulatif suivant :

	DEMARRAGE DIRECT	DEMARRAGE ÉTOILE TRIANGLE	DEMARRAGE STATORIQUE	DEMARRAGE ROTORIQUE	DEMARRAGE PAR AUTO TRANSFORMATEUR	DEMARRAGE ÉLECTRONIQUE
Courant de démarrage	4 à 8 I_n	2,5 I_n	2 à 4 I_n	5 I_n	3/4,5/6 I_n	1,5 à 2 I_n
Baisse de I_d en %	0 %	75 %	50 %	30 %	40 %	10 à 50% (en 100ms)
Couple au démarrage	0,6 à 1,5 C_n	0,2 à 0,5 C_n	0,6 à 0,85 C_n	0,4 à 0,85 C_n	0,4 à 0,85 C_n	<2,5 C_n
Commande	1 temps	2 temps	2 temps (ou +)	2 temps (ou +)	3 temps	électroniques
Avantages	- démarreur simple et économique -couple au démarrage important	-économiques -bon rapport couple/courant	-possibilités de réglages des valeurs au démarrage	-très bon rapport couple/courant -possibilité de réglage des valeurs au démarrage	-bon rapport couple/courant - possibilités de réglages des valeurs au démarrage	-démarrage sans à coup -montée progressive en vitesse -limitation de l'appel de courant au démarrage
Inconvénients	-pointe de courant très importante -démarrage brutal	-couple de démarrage faible -coupure d'alimentation au changement de couplage -moteur 6 bornes	-faible réduction de la pointe de courant au démarrage -nécessite des résistances volumineuses	-moteur à bague plus onéreux	-nécessite un auto transformateur onéreux -présente des risques de réseau perturbé	-prix

Tableau 1.3 – Résumé des procédés de démarrage

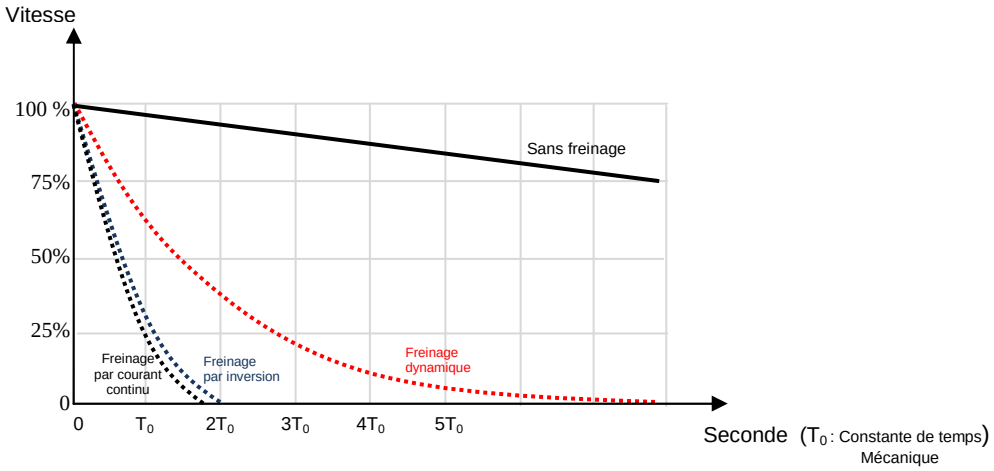


Figure 1.26 – Taux de décroissance de la vitesse selon le genre de freinage utilisé [1]

6 - Conclusion :

Dans ce chapitre, nous avons vu qu'avec une dizaine de dispositifs de base, on peut réaliser des systèmes de commande allant du plus simple démarreur de moteur aux contrôleurs de procédés industriels les plus complexes. Certain de ces dispositifs, comme les sectionneurs, disjoncteurs, contacteur, relais thermique sont introduits directement dans le circuit de puissance pour interrompre, mesurer ou modifier le courant de l'appareil commandé.

D'autres dispositifs sont utilisés dans le circuit de commande à basse tension. Ces composants comprennent les boutons-poussoirs, les interrupteurs spéciaux, les lampes témoins, les relais de commande qui peuvent comporter plusieurs contacts. Ils sont interconnectés de façon à réaliser l'intelligence du système de commande et les fonctions de signalisation.

Pour le chapitre II, on utilisera les dispositifs industriels vus précédemment, ainsi que les principes de démarrage détaillés précédemment tout ça sera commandé par un dispositif informatisé assisté par PC.

Chapitre 2

Conception et programmation du système

2.1 - Introduction

L'informatique a pris une place importante dans le fonctionnement des procédés industriels. Il est donc judicieux d'introduire des techniques à base de microprocesseurs telle que les DSP, INTEL, AMD et autres, parallèlement à ça, on doit analyser et concevoir les logiciels chargés du pilotage et du contrôle des machines à partir du schéma d'ensemble défini par le cahier des charges. Bien sûr, tout ça ne peut être réalisé sans un algorithme générant les fameux créneaux de commande, qui sont des signaux logiques successifs exprimés par des 1 et des 0 et qui assure l'ouverture et la fermeture des différents éléments du système industriel tels que les semi-conducteurs, les contacteurs et les relais de commande qu'on détaillera dans ce chapitre.

On va se focaliser pour l'instant sur le dispositif qui gère les signaux de commande. On pourrait bien sûr, utiliser les systèmes habituels à base de carte DSP ou DSPACE et qui sont pilotés par des logiciels spécifiques comme Matlab, Labview et qui ne pourrait être réalisables sans la présence d'un PC. Mais aux lieux de ça, on pourrait générer les signaux de commande en utilisant directement les entrées/sorties de l'ordinateur, en particulier le port imprimante connue sous le nom de LPT (Local Parallel Transmition).

2.2 - Généralités sur la commande industrielle par PC

La commande par ordinateur offre une grande fiabilité et une flexibilité optimale. Pour la périphérie intégrée dans la commande, le PC industriel offre une fonctionnalité multitâche en temps réel [13]. En plus de l'utilisation d'un ordinateur, des interfaces matérielles et des algorithmes d'asservissement en ligne doivent être adaptés en vue de piloter le système. Choisir et concevoir des algorithmes de commande permettant d'imposer des spécifications dynamiques appropriées tout en respectant certaines contraintes.

La logique du câblage se fait sous différentes formes et on trouve plusieurs structures standards pour les interfaces PC/Actionneur [12], [16], [17], on cite l'AS-i, le FIP, le RLI qui sont reliées en réseau avec les deux protocoles standards TCP/IP. Habituellement, on trouve des dizaines de systèmes reliés en réseau mais on ne va donner que les deux structures les plus utilisées :

Réseau FIPWAY : c'est un réseau local industriel assurant la communication entre les différents automates programmables. Il sert de bus de synchronisation entre automates. Il est dérivé de la norme FIP.

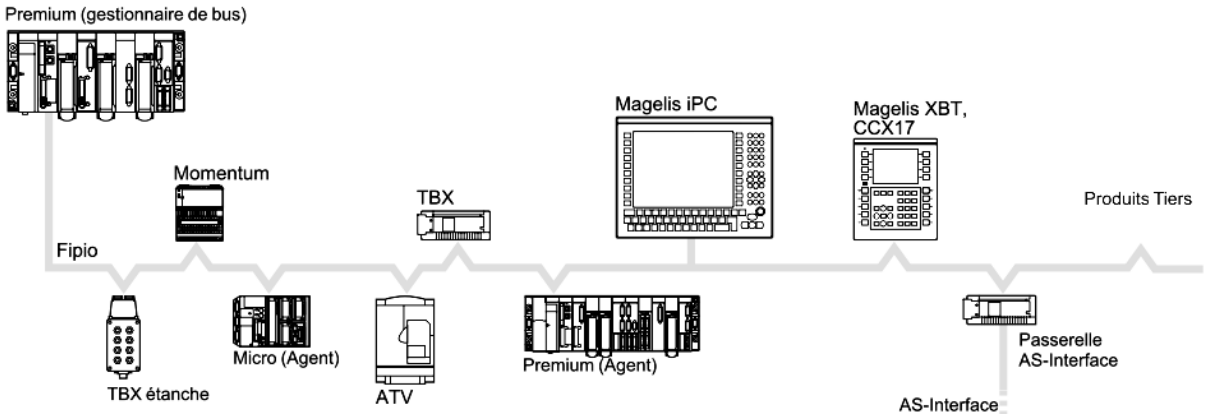


Figure 2.1 - Structure d'un réseau local industriel (Fipway) [14]

Réseau MOBUS/JBUS : C'est un réseau local industriel qui répond aux architectures Maitre/Esclave. Le bus est composé d'une station Maitre et de stations Esclave. Seule la station Maître peut être à l'initiative de l'échange (la communication directe entre stations Esclave n'est pas possible).

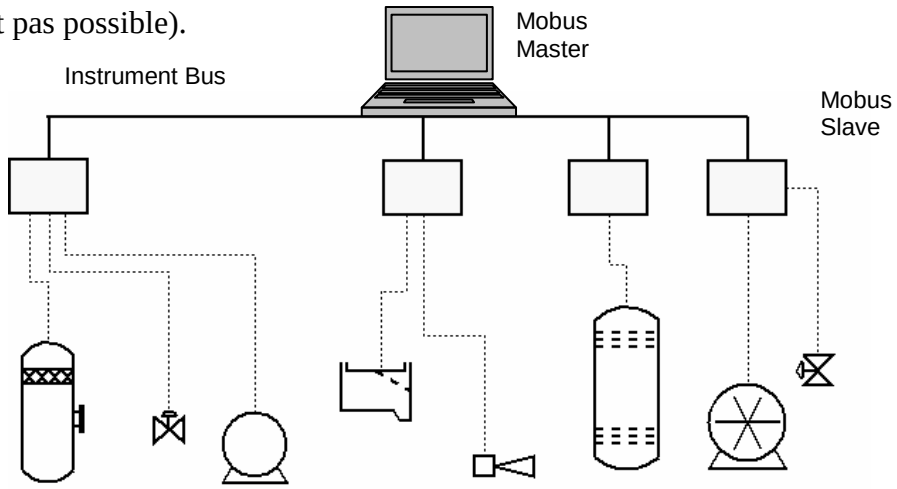


Figure 2.2 - Structure d'un réseau local industriel (Modbus) [15]

2.3 - Exemple de commande par PC - démarrage statorique programmé

2.3.1 - Principe : L'idée est de contrôler ce processus par des codes binaires générés par le port imprimante. On va utiliser la sortie du port afin qu'il affecte un code en 8 bits (8 sorties simultanées) et qui activera le relais alimentant l'organe de commande des différents contacteurs. L'organe de commande est en réalité une bobine alimenté sous 220V avec un noyau mobile relié à des lames assurant la fermeture de la ligne triphasée.

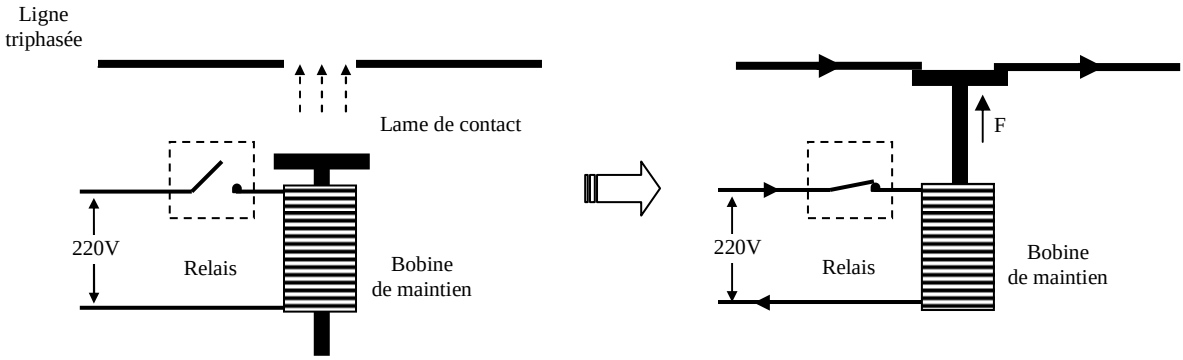


Figure 2.3 – Principe de commande par relais/contacteur

2.3.2- Application à un démarrage statorique d'une MAS à cage :

Cette application est un peu spéciale car elle traite les étapes de conception d'un système de démarrage statorique informatisé à base d'un microprocesseur INTEL 486 en utilisant son port parallèle (LPT) destiné généralement à la gestion des imprimantes. Le but est donc de créer des signaux de commande numérique capables de piloter n'importe quel actionneur électromagnétique, en l'occurrence dans notre cas, un moteur asynchrone à cage. L'algorithme de commande sera écrit en langage C sous Windows Me seulement (on verra pourquoi dans le § 2.4.1). L'interface de commande sera complétée par des contacteurs et relais qui constitueront la partie puissance. Comme tout système de commande, le cœur de l'installation contient en général au moins un microprocesseur ou microcontrôleur, dans notre cas c'est le µp Intel 486 qui est relié par ses périphériques d'entrée/sortie. Pour la partie puissance, on aura des coupleurs d'interfaces (Relais, contacteurs 1 et 2) et bien sûr un moteur asynchrone à démarrage statorique. La chaîne globale est schématisée par la figure 2.4.

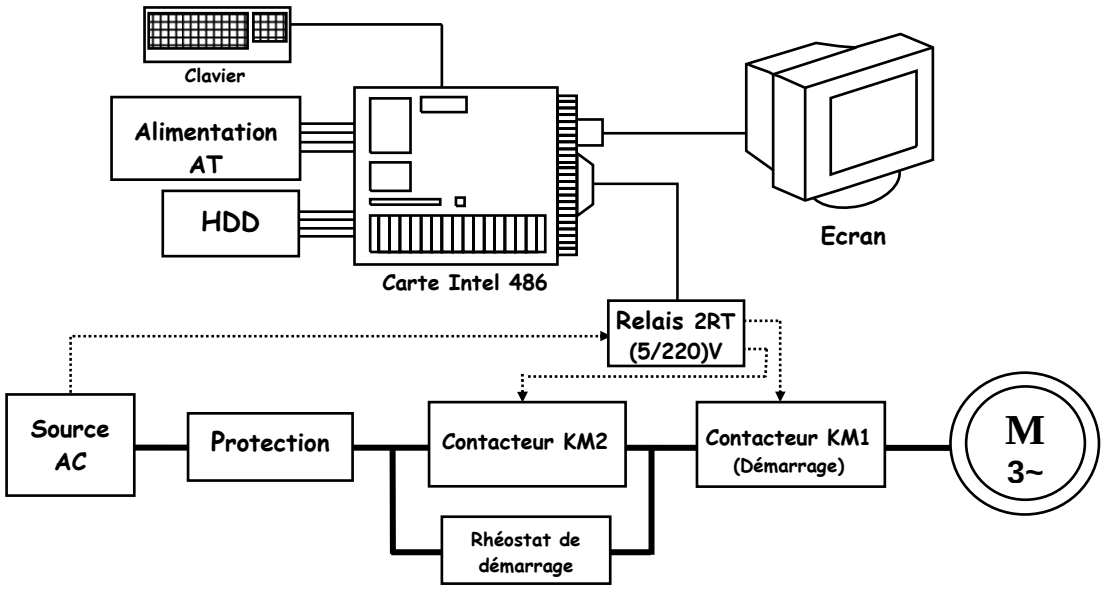


Figure 2.4 - Diagramme synoptique du montage

Un relais électromécanique peut assurer une isolation galvanique dans la mesure où le circuit de commande et celui de puissance sont câblés sur des circuits séparés. En effet, il faut isoler la partie commande (trait fin) de la partie puissance (trait gras) afin d'éviter les retours d'énergie, pour cela un relais de type 2RT fait l'affaire (figure 2.5). Le principe de ce relais est qu'il se comporte comme un interrupteur commandable, c'est-à-dire pour une tension de 0V il s'ouvre et pour 5V il se ferme, ce qui se traduit respectivement par un 0 et un 1 en langage binaire. Il peut fonctionner sous 220V et 2A max avec un temps de réponse de 5 millisecondes.

Figure 2.5 - Le relais 2RT

Figure 2.6 - µP Intel 486 DX2 50MHz

2.3.3 - La carte INTEL 486 :

L'Intel 80486 (figure 2.6) est un microprocesseur CISC fabriqué par Intel® qui fait partie de la famille des x86. Du point de vue de l'architecture, c'est une grande amélioration. Il y a un cache d'instruction et de donnée unifiée intégrée, en option, une unité de calcul en virgule flottante intégrée (FPU), et une unité d'interface de bus améliorée. En outre, le cœur du processeur peut soutenir un rythme d'exécution d'une instruction par cycle. Ces améliorations permettent approximativement de doubler la vitesse d'exécution à 100MHz [5]. Les éléments suivants constituent la carte Intel 486 :

- Le processeur Intel® i486™ DX2™
- 7 interfaces ISA pour port RS232, LPT, VGA, IDE
- 3 Interfaces VESA pour extension ISA
- 8 mémoires extérieures de 4MO
- Une prise AT (clavier)

Les éléments suivants constituent les périphériques de la carte :

- Carte E/S intégrant un port LPT, IDE, COM
- Carte VGA
- Un disque dur de 512 MO
- Une alimentation AT de 200W

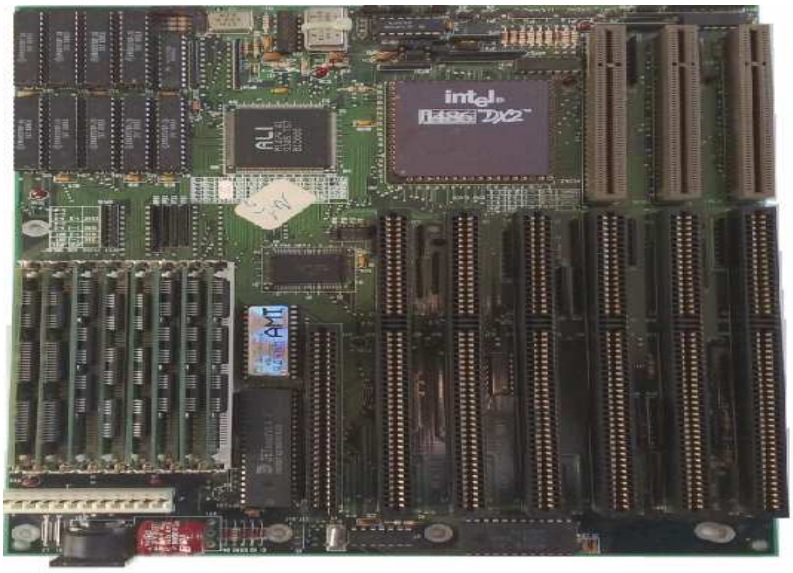


Figure 2.7 - Photo de la carte mère avec son processeur Intel 486

2.3.4 – Programmation du port d'imprimante sous DOS:

Pour commencer à programmer le port, nous allons utiliser le système DOS. Sous DOS, nous avons des commandes pour accéder directement au port LPT mais ces programmes ne fonctionnent pas sur les systèmes basés sur Windows NT, XP, Vista ou versions supérieures. En effet, pour des raisons de sécurité, ces versions de Windows ne permettent pas d'accéder directement au port. Pour programmer le port parallèle sous XP, nous avons besoin d'écrire des pilotes en mode noyau (voir annexe). Maintenant, si on prend une fiche imprimante LPT plus connue sous le nom de DB-25. Le nombre 25 représente en général, le nombre de connexions existantes et dans notre cas c'est une prise à 25 broches. Le tableau ci-dessous nous donne l'aspect de ce type de prise avec la désignation de chaque broche.

Pin N°	Nom	Désignation
1	STR - Strobe	Balayage
2	D0 - Data bit 0	Bit de données 0
3	D1 - Data bit 1	Bit de données 1
4	D2 - Data bit 2	Bit de données 2
5	D3 - Data bit 3	Bit de données 3
6	D4 - Data bit 4	Bit de données 4
7	D5 - Data bit 5	Bit de données 5
8	D6 - Data bit 6	Bit de données 6
9	D7 - Data bit 7	Bit de données 7 (poids fort)
10	ACK	Acquittement
11	Busy	Occupé (lecture des données)
12	Paper Out	Plus de papier
13	Select	Sélection

14	Auto feed	Saut de page
15	Error	Erreur
16	Reset	Réinitialisation
17	Select Input	Sélection de l'entrée
de 18 à 25	GND	Masse

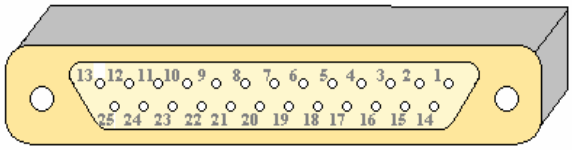


Tableau 2.1 - Brochage pour une liaison LPT femelle

On aura donc sur ce type de prise 8 broches de données bidirectionnelles allant du pin 2 au pin 9, contrairement au port série cela nous donnera une transmission de 8 signaux binaires simultanés, en d'autres termes on aura un mot binaire de 1 Octet en transmission parallèle. En plus, vu qu'on a une communication bidirectionnelle, on doit avoir un système permettant de choisir la direction de transmission de données, soit pour saisir les données ou soit pour les transmettre. Ce protocole est symbolisé par I/O c'est-à-dire input/output.

2.3.5 - Détection du port parallèle :

Pour connaître la disponibilité des ports imprimantes d'un ordinateur, nous allons utiliser l'emplacement mémoire ou adresse du port. Si on exécute le code suivant, on obtiendra les adresses des ports LPT disponibles [17].

Code : C

```
#include <stdio.h>
#include <dos.h>
void main()
{
    unsigned int far *ptraddr;     /* Pointeur pour localiser
                                   l'adresse des ports LPT */
    unsigned int adresse;          /* Adresse du Port */
    int a;
    ptraddr=(unsigned int far *)0x00000408;
    clrscr();

    for (a = 0; a < 3; a++)
    {
        adresse = *ptraddr;
        if (adresse == 0)
            printf("Port non trouvé pour LPT%d \n",a+1);
```

```
        else
            printf("L'adresse assignée à LPT%d est 0x%X
                    \n",a+1,adresse);
        ptraddr++;
    }
    getch();
}
```

On aura les données suivantes sur l'écran de l'ordinateur :

Code C

```
     L'adresse assignée à LPT1 est 0x3BC
     L'adresse assignée à LPT1 est 0x378
     L'adresse assignée à LPT1 est 0x278
```

2.3.6 - Principe du montage :

Comme tout le monde le sait, lors du démarrage, le moteur asynchrone fonctionne comme un transformateur où le primaire (stator) est alimenté par la source et le secondaire (rotor) est en court-circuit, l'intensité du courant est alors très importante.

$$I_d = 6\ I_n \qquad (1)$$

Pour diminuer cette contrainte électrique on doit insérer en série un rhéostat de démarrage. En effet, les enroulements du moteur seront alimentés à travers des résistances qui provoqueront une chute de tension U_R=RI, alors le moteur sera alimenté sous une tension :

$$U_M = U_{ligne} - RI \qquad (2)$$

Au fur et à mesure que le moteur prend de la vitesse le courant diminue ainsi que la tension U_R. Dans ce procédé de démarrage, le moteur est couplé suivant la tension du réseau, soit en Y, soit en Δ. Dans les deux cas chaque résistance est placée en série avec un enroulement. La figure qui suit représente le schéma d'un démarrage statorique en deux temps à un sens de rotation pour un moteur triphasé asynchrone à cage de moyenne puissance.

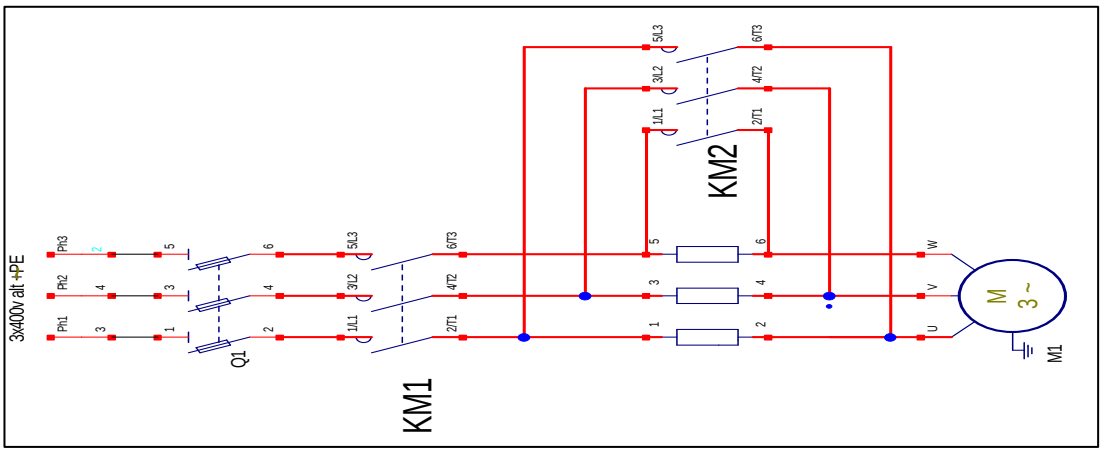


Figure 2.8 – Schéma d'un démarrage statorique

Avec Q1 : Le disjoncteur principal ; KM1 : Le contacteur 1 ; KM2 : Le contacteur 2

Alors l'idée est de commander ce processus par des codes binaires générés par la carte vue précédemment. Le principe de cette commande est d'utiliser le port parallèle afin qu'il affecte le code [00000001] qui activera le relais alimentant la bobine du contacteur KM1. Le contacteur restera fermé pendant 5 secondes, juste assez pour que le moteur s'amorce. Puis un autre code [00000011] actionnera le relais qui alimentera la bobine du contacteur KM2, ce déclanchement causera l'élimination du rhéostat par court-circuit. Une pression sur une touche du clavier nous donnera le code [00000000] qui logiquement annulera l'alimentation des bobines des deux contacteurs d'ou un freinage progressif du moteur. L'organigramme de la figure 2.9 aidera sûrement à mieux comprendre ce qu'on a dit.

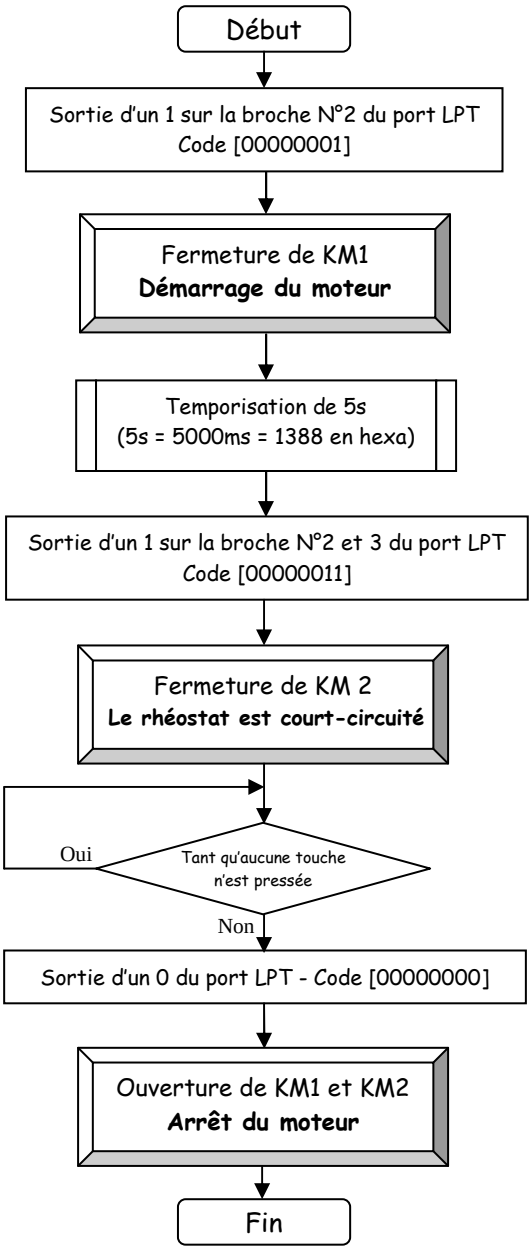


Figure 2.9 – Organigramme de commande

2.3.7 - L'algorithme de commande :

Le programme qui gère le démarrage et le fonctionnement du moteur est écrit en langage C car il est plus évolué et plus pratique que le langage assembleur, ça se traduit par la simplicité de ses instructions et la capacité d'accéder directement aux entrées/sorties de cette carte ou de n'importe qu'elle autre carte. Dans notre cas, les instructions principales seront :

- Pour faire entrer un bit au port parallèle, on doit utiliser l'instruction **[inportb]**
- Pour faire sortir un bit du port parallèle, on doit utiliser l'instruction **[outportb]**
- Pour faire une temporisation, on utilise l'instruction **[delay]**
- Pour afficher un message sur écran, on utilise l'instruction **[printf]**

Bien sûr ça parait facile, mais il faut synchroniser le processeur avec le port LPT, pour cela on doit mettre les sorties (les bits de commande) dans la case mémoire dont l'adresse est celle du port LPT de la carte, en l'occurrence c'est 0x378 (0x : veut dire hexadécimal).

Vue la longueur du programme, on ne tient compte que des lignes principales, celles qui prennent en charge la commande. Le programme simplifié sous VisualC++ sera comme suit :

Code : C

```
#include"conio.h"
#include"dos.h"
#include"stdio.h"

#define PORT 0x378  /*Affectation de l'adresse du port LPT*/
#define TDS 0x1388  /*Donne une temporisation de 5 sec*/

void main()
 {
   printf("Démarrage du moteur: statorique de %ld secondes,
TDS");
   outportb(PORT,0x01);  /*sortie du code %00000001*/
   delay(TDS);      /*temporisation de 5 sec*/
   outportb(PORT,0x03);
   printf("Fin du démarrage, le moteur tourne à plain
régime");
    while(!kbhit())
   {
     Printf(Appuyer sur une touche pour arrêter le moteur");
     outportb(PORT,0x00); /*sortie du code %00000011*/
   }
   return 0;
 }
```

2.3.8 - Le banc d'essai :

La réalisation expérimentale a été testée en atelier. On a utilisé le montage le plus simple possible, en prenant un moteur asynchrone triphasé à cage et à démarrage statorique pour un sens de rotation. La photo du montage est représentée dans la figure 2.10.

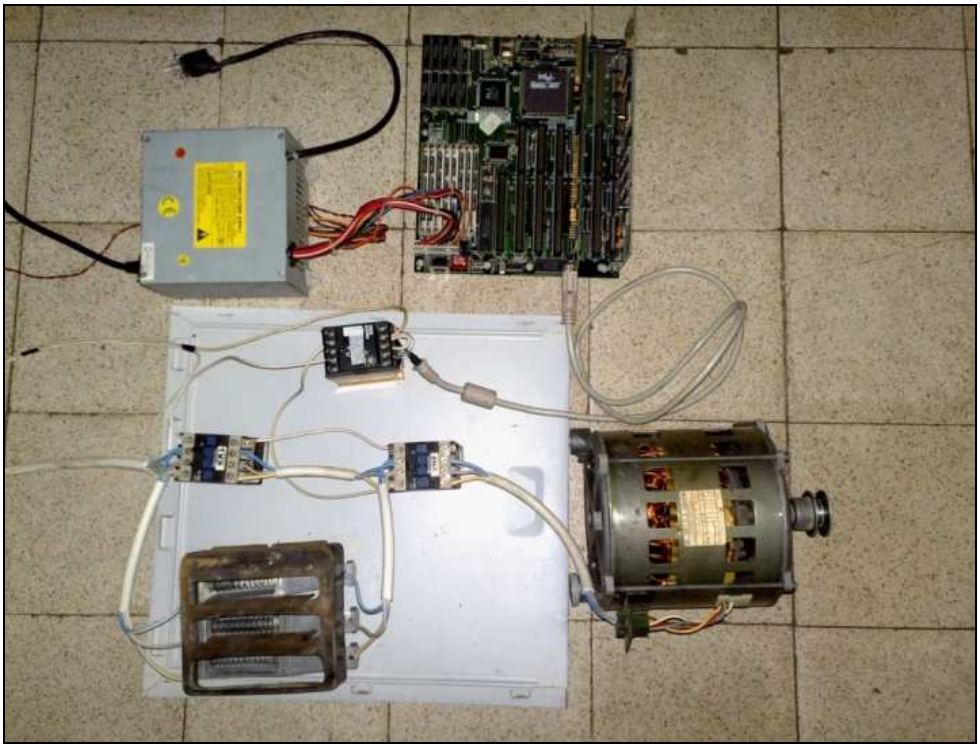


Figure 2.10 – Photo du banc d'essai

Le démarrage se passe bien, les 5 secondes d'amorçage sont respectées, le moteur arrive à 70% de sa vitesse nominale lors de la fermeture du deuxième contacteur. L'augmentation de la vitesse continue pendant 5 secondes jusqu'à se stabiliser à la vitesse de fonctionnement nominale qui est de 3000 tr/min.

Cette exemple nous donne un léger aperçu de nos projets en vue de leur réalisation. En effet, les programmes précédents on été compilés avec Turbo C sous Windows Melinium mais on utilisera pour ce projet le compilateur de Visual C++ et on changera DOS pour passer à Windows XP.

On devra en plus faire quelques rectifications au sujet du contrôle des ports PC. Tout cela va être détaillé dans les paragraphes qui suivent.

2.4 - L'algorithme d'acquisition en ligne :

2.4.1 - Le problème d'accès au port sous Windows :

Pour programmer n'importe quel port, il est préférable d'utiliser l'environnement DOS. L'avantage c'est que sous DOS on dispose de commandes qui nous donnent un accès direct vers les ports du PC mais cette procédure ne peut fonctionner sous des systèmes basés sur les Windows XP, NT ou une autre version ultérieure, cela pour des raisons de sécurité d'accès.

Pour programmer un port sous ces systèmes, on a besoin d'écrire sous un mode nommé « Mode Noyau », qui existe déjà sous Linux et Windows Me mais reste absent sous XP. La solution est : si l'on veut exécuter le même programme on doit utiliser Visual C++ au lieu de Turbo C. En plus, on doit créer une application en suivant ces étapes [17]:

- démarrer Visual C++, choisir File --> New
- Sur l'onglet « Projects », on choisi « MFC AppWizard (exe) » et on nomme notre projet « ParallelPort »
- Dans la fenêtre suivante on choisi « Dialog based » et on clic sur « Next » en laissant les autres options par défaut
- On clic sur « Finish » puis « OK » pour avoir une fenêtre avec deux boutons et une phrase disant « TODO : Place Dialog control here », on doit supprimer cette phrase.
- On fait défiler le menu contextuel sur le bouton « OK », puis on sélectionne « Propriété » et on change la légende vers « &EXIT »
- Si on exécute cette application en cliquant sur cette icône :  , cela ne nous donnera aucune erreur et aucun avertissement du coup on peut accéder à la fenêtre de la figure 2.11 :

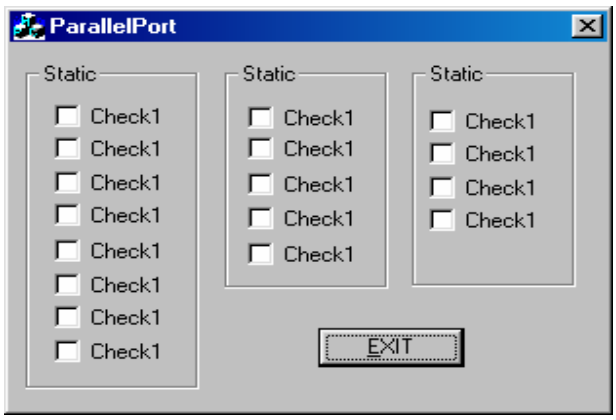


Figure 2.11 - Boite de dialogue de notre application

Ensuite, on fait un clic droit sur la zone marquée « statique » et on sélectionne « propriétés ». On modifie les légendes vert les données, les statuts et le contrôle, respectivement. On fait un clic droit sur la première case Check 1 et on change la légende de "Pin 2" et l'ID vers IDC_Pin2. De même on change les légendes des données du groupe incluant : les pins 2 à 9, état du port des pins 10, 11, 12, 13 et 15, le contrôle du Port de pins 1, 14, 16 et 17. Puis on change l'ID en conséquence (IDC_Pin2, IDC_Pin3, etc ...).

Code : C

```
void CParallelPortDlg::ChangePin(int pin)

{
int data_register, new_register;
UpdateData(TRUE);
data_register=_inp( DATA );
new_register=0;

if( m_pin2==TRUE ) new_register |= 0x01;
if( m_pin3==TRUE ) new_register |= 0x02;
if( m_pin4==TRUE ) new_register |= 0x04;
if( m_pin5==TRUE ) new_register |= 0x08;
if( m_pin6==TRUE ) new_register |= 0x10;
if( m_pin7==TRUE ) new_register |= 0x20;
if( m_pin8==TRUE ) new_register |= 0x40;
if( m_pin9==TRUE ) new_register |= 0x80;

_outp(DATA, new_register);
}

void CParallelPortDlg::ChangeControl()

{
int control_register, new_register;
UpdateData(TRUE);
control_register = _inp( CONTROL );
new_register = control_register;

if( m_pin1== 0 ) new_register &= 0xFE;
    else new_register |= 0x01;
if( m_pin14==0 ) new_register &= 0xFD;
    else new_register |= 0x02;
if( m_pin16==0 ) new_register &= 0xFB;
    else new_register |= 0x04;
if( m_pin17==0 ) new_register &= 0xF7;
    else new_register |= 0x08;

_outp(CONTROL, new_register);
}
```

La fenêtre de conception étant terminée, la prochaine partie est le codage. Nous avons placé des contrôles dans la boîte de dialogue. Pour obtenir les valeurs de ces contrôles, nous avons besoin de variables associées. Pour cela, dans le menu contextuel on sélectionne « ClassWizard » puis « Variables membres ». On obtiendra une liste de contrôle d'identité. On sélectionne chaque ID séparément et on clique sur ajouter une variable. On tape le nom de variable comme m_pin1. m_pin10, m_pin11, etc ... et dans la catégorie type de variable on met : BOOL. Cela nous donnera la configuration suivante :

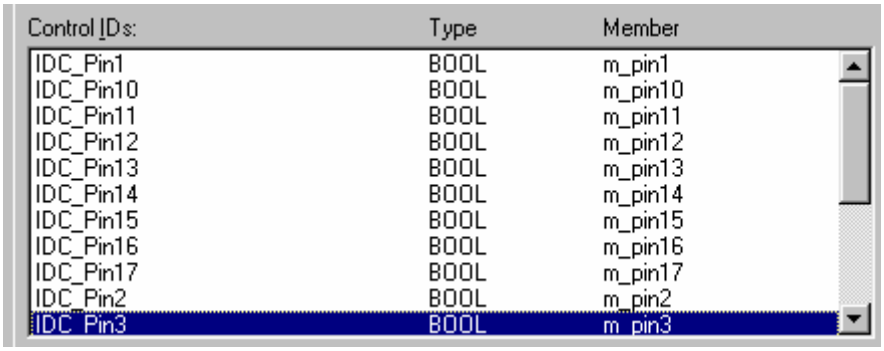


Figure 2.12 - Boite de dialogue de notre application

Maintenant, c'est la dernière étape (la plus importante), c'est celle qui donne l'accès au port sous XP. L'astuce est d'utiliser le fichier DLL « InpOut32 » est qui peut envoyer des données au port parallèle (On peut télécharger ce fichier gratuitement à partir de http://logix4u.net).

Pour savoir comment utiliser les fichiers DLL dans VC++, nous allons convertir notre projet pour l'activé sous XP.

- On ajoute ces deux lignes dans le fichier « ParallelPortDlg.cpp » après les directives de préprocesseur.

Code : C

```
Bref Inp32 _stdcall (portaddr courte);
void Out32 _stdcall (portaddr court, datum courte);
```

- On copie le dossier du projet : le fichier DLL « inpout32.lib » et le fichier « inpout32.dll » obtenu en compilant le code source.
- Dans le menu Projet/Paramètres/Tab_link/object/library_modules, on met « inpout32.lib »
- Maintenant, notre système devrait fonctionner sans problème.

2.4.2 - La logique de programmation sous langage C:

Il existe de nombreux langages de plus ou moins haut niveau en informatique dans lesquels nous pouvons écrire notre programme, en voici quelques-uns par exemple : le C, le C++, Java, Visual Basic, Delphi, etc ...

Pour notre programme on a choisi le langage C car c'est un langage normalisé qui comprend des instructions de préprocesseur, des instructions C et des symboles dont la sémantique peut varier selon le contexte, c'est vrai que ça demande une initiation longue, mais on va bien devoir passer par là. Petit à petit, on expliquera les étapes de conception du programme d'acquisition des données de l'historique de fonctionnement de la pompe, en s'aidant bien sur de ce qui a été fait dans l'exemple précédent (§2.3). Bien sur on ne va pas mettre toutes les instructions du langage C, mais on se limitera sur ce qu'on a besoin dans nos travaux, avec ce qui suit on apprendra suffisamment de choses pour commencer à réaliser des programmes de plus en plus complexes. Le but étant pour nous d'essayer de nous débrouiller tous seuls dans n'importe quel programme écrit en C.

2.4.3 - Le préprocesseur et les bibliothèques de bases de données :

Un préprocesseur en C est un programme qui procède à des transformations sur un code source, avant l'étape de compilation (on parle alors de pré-compilateur) ou d'interprétation proprement dite [21]. Ce sont des commandes en C qu'on doit mettre au début de n'importe quel programme sinon le programme ne pourra pas se compiler.

Le préprocesseur inclut des bibliothèques de bases de données. Ces bases de données contiennent les informations de chaque mot clé et de chaque instruction du langage C. Il y a en des dizaines mais c'est rare qu'un programme les utilise tous en même temps. Elles sont précédées par la directive du préprocesseur « #include ».

Les bibliothèques les plus utilisées dans la plupart des programmes sont :

<stdlib.h> : Génère de nombres pseudo-aléatoires, l'allocation de mémoire

<stdio.h> : Fournit les capacités centrales d'entrée/sortie

<stddef.h> : Définit plusieurs types et macros utiles, comme « null »

<math.h> : Calcule des fonctions mathématiques courantes

<assert.h> : Fait le diagnostic à l'exécution

2.4.4 - Les principales instructions :

Il existe de nombreuses façons d'expliquer les bases d'un langage de programmation. On choisi une approche progressive mais concrète et basée sur l'expérimentation. Les informations données font référence à la définition du langage C selon la norme ISO/IEC 9899:1999 (appelée aussi C99).

Le langage C comprend plus de 300 instructions et plus de 50 symboles. Pour plus de détails, il est conseillé de se munir d'un livre de référence. Les instructions les plus utilisées et qui seront utilisées dans nos programmes sont détaillées dans le tableau suivant :

Instruction	**Description**	**Symbole**	**Description**
include	Inclure une base de donnée	#	Directive de préprocesseur
define	Définir des variables ou des fichiers	;	Fin d'une instruction
void	Fonction vide	{	Début d'une fonction
scanf	Saisir un message sur l'écran	}	Fin d'une fonction
printf	Imprimer un message sur l'écran	!	Négation
delay	Temporisation (en ms)	0x	Hexadécimal
inport	Lecture de données sur un port	%	Obtenir le reste d'une division
inportb	Lecture binaire sur un port	&&	ET logique
outport	Ecriture de données sur un port	||	OU logique
outportb	Ecriture binaire sur un port	\n	Retour à la ligne (≡ Entrée)
return 0	Exécution réussi du programme	\t	Tabulation de 5 caractères
return X	Erreur (X≠0)	\a	alerte:bip sur le haut parleur
kbhit	Pression d'une touche du clavier	\b	effacer un caractère
if ….. else	Boucle de condition : si …. sinon	\\	imprime le caractère
while	Boucle : tant que	++	Incrémentation
for	Boucle : pour	--	Décrémentation
getchar	Récupération d'un seul caractère	%ld	Nombre entier
putchar	Affichage d'un seul caractère	%lf	Nombre décimal
main	Fonction principale	" … "	Message
clrscr	Effacement du contenu de l'écran	/* … */	Commentaires

Tableau 2.2 – Les principales instructions en langage C

2.4.5 - Lire et écrire dans des fichiers :

Heureusement, on peut lire et écrire dans des fichiers en langage C. Les fichiers seront écrits sur le disque dur de l'ordinateur : l'avantage est donc qu'ils restent les mêmes si on arrête le programme ou l'ordinateur.

Pour lire et écrire dans des fichiers, nous allons avoir besoin de réutiliser tout ce que nous avons appris jusqu'ici : pointeurs, structures, pointeurs de structures, chaînes de caractères etc. Pour une bonne compréhension de ce chapitre, il faut donc avoir une petite base en programmation si c'est possible en langage C.

Pour lire et écrire dans des fichiers, nous allons nous servir de fonctions situées dans le fichier source nommée **stdio.h** (.h est une extension qui veut dire header). Elle contient plusieurs fonctions, notamment des fonctions faites pour travailler sur des fichiers.

Tous les fichiers sources que nous allons utiliser : stdlib.h, stdio.h, math.h, string.h..., sont ce qu'on appelle des fichiers source standards. Ce sont des fichiers livrées avec l'IDE et qui ont la particularité de fonctionner sur tous les OS. On peut donc les utiliser partout, qu'on soit sous Dos, Windows, Linux, Mac ou autre. Les fichiers source standards ne sont pas très nombreux et ne permettent de faire que des choses très basiques.

2.4.5.1 - Création des fichiers :

Maintenant que les bons fichiers source sont inclus, nous allons pouvoir s'attaquer aux choses les ardues. Voici ce qu'il faut faire à chaque fois dans l'ordre quand on veut ouvrir un fichier (que ce soit pour lire ou pour écrire dedans) :

- On appelle la fonction d'ouverture de fichier **fopen** qui nous renvoie un pointeur sur le fichier.
- On vérifie si l'ouverture a réussi (c'est-à-dire si le fichier existait) en testant la valeur du pointeur qu'on a reçu. Si le pointeur vaut NULL, c'est que l'ouverture du fichier n'a pas marché, dans ce cas on ne peut pas continuer (il faut afficher un message d'erreur).
- Si l'ouverture a marché (si le pointeur est différent de NULL), alors on peut lire et écrire dans le fichier à travers des fonctions que nous verrons un peu plus loin.
- Une fois qu'on a terminé de travailler sur le fichier**,** il faut penser à le "fermer" avec la fonction **fclose**.

Nous allons dans un premier temps apprendre à nous servir de **fopen** et **fclose**. Une fois que nous saurions faire cela, nous apprendrons à lire le contenu du fichier et à écrire dedans. Voir justement le prototype de la fonction fopen :

Code : C

```
FILE* fopen(const char* nomDuFichier, const char* modeOuverture);
```

Cette fonction attend 2 paramètres :

- Le nom du fichier à ouvrir.
- Le mode d'ouverture du fichier, c'est-à-dire une indication qui dit si vous voulez juste écrire dans le fichier, juste lire dans le fichier, ou les deux à la fois.

Cette fonction renvoi un pointeur sur FILE. C'est un pointeur sur une structure de type FILE. Cette structure est définie dans stdio.h. La fonction fopen renvoi un FILE*. Il est extrêmement important de récupérer ce pointeur, pour pouvoir ensuite lire et écrire dans le fichier. Le pointeur est initialisé à NULL dès le début. Je vous rappelle que c'est une règle fondamentale que d'initialiser ses pointeurs à NULL dès le début si on n'a pas d'autre valeur à leur donner. Si on ne le fait pas, on risque des plantages par la suite.

Notez que la forme de la structure peut changer d'un OS à l'autre (elle ne contient pas forcément les mêmes sous-variables partout). Pour cette raison, on ne modifiera jamais le contenu d'un FILE directement, on passera par des fonctions qui manipulent le FILE à notre place.

Maintenant, nous allons appeler la fonction fopen et récupérer la valeur qu'elle renvoi dans le pointeur "fichier". Mais avant ça, il faut savoir comment se servir du second paramètre, le paramètre "modeOuverture". En effet, il y a un code à envoyer qui indiquera à l'ordinateur si on ouvre le fichier en mode de lecture seule, d'écriture seule, ou des deux à la fois, ce qui semble logique.

Voici les modes d'ouvertures possibles :

- **"r" : lecture seule**. On pourra lire seulement le contenu du fichier. Le fichier doit être créé au préalable.

- **"w" : écriture seule**. On pourra seulement écrire dans le fichier. Si le fichier n'existe pas, il sera créé.

- **"a" : mode d'ajout**. On écrit dans le fichier en partant de la fin. On peut rajouter donc du texte à la fin du fichier. Si le fichier n'existe pas, il sera créé.

- **"r+" : lecture et écriture**. On pourra lire et écrire dans le fichier. Le fichier doit être créé au préalable.

- **"w+" : lecture et écriture, avec suppression du contenu au préalable**. Le fichier est d'abord vidé de son contenu puis on écrit et on lit ensuite dedans. Si le fichier n'existe pas, il sera créé.

- **"a+" : ajout en lecture / écriture à la fin**. On écrit et on lit du texte à partir de la fin du fichier. Si le fichier n'existe pas, il sera créé.

Et encore, pour chaque mode qu'on a vu par exemple si vous rajoutez un "b" après le premier caractère ("rb", "wb", "ab", "rb+", "wb+", "ab+"), alors le fichier est ouvert en mode binaire. On a déjà fort à faire avec ces 6 modes d'ouverture à retenir personnellement, j'utilise souvent "r" (lecture), "w" (écriture) et "r+" (lecture et écriture). Le mode "w+" est un peu dangereux parce qu'il vide de suite le contenu du fichier, sans demande de confirmation. Il ne doit être utilisé que si vous voulez réinitialiser le fichier. Le mode d'ajout ("a") peut être utile dans certains cas, juste pour rajouter des informations à la fin du fichier.

Le code suivant ouvre le fichier test.txt en mode "r+" (lecture / écriture) :

Code : C

```
int main(int argc, char *argv[])
{
    FILE* fichier = NULL;
    fichier = fopen("test.txt", "r+");
    return 0;
}
```

Le pointeur "fichier" devient alors un pointeur sur "test.txt". Le fichier conçu doit être situé dans le même dossier que notre exécutable (.exe).

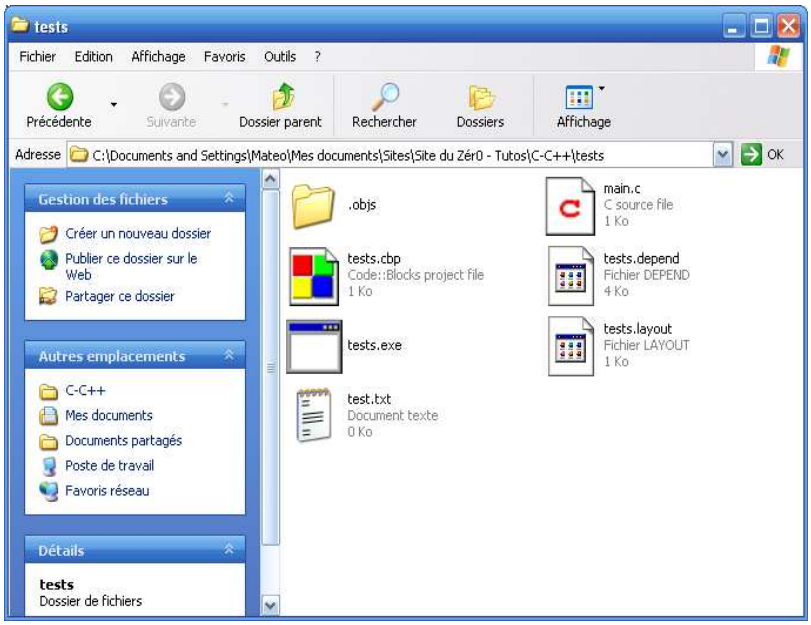


Figure 2.13 - Plan de création du fichier « Test » sous Windows

Tester l'ouverture du fichier

Le pointeur "fichier" devrait contenir l'adresse de la structure de type FILE qui sert de descripteur de fichier. Celui-ci a été chargé en mémoire par la fonction fopen(). A partir de là, nous avons 2 possibilités :

- Soit l'ouverture a réussi, et on peut continuer (c'est-à-dire commencer à lire et écrire dans le fichier).
- Soit l'ouverture a échoué parce que le fichier n'existait pas ou était utilisé par un autre programme. Dans ce cas, on doit arrêter de travailler sur le fichier.

Juste après l'ouverture du fichier, il faut absolument vérifier si l'ouverture a réussi ou pas**.** Pour faire ça, c'est très simple : si le pointeur vaut NULL, l'ouverture a échoué sinon l'ouverture a réussi. On va donc suivre systématiquement le schéma suivant :

Code : C

```
int main(int argc, char *argv[])
{
    FILE* fichier = NULL;
    fichier = fopen("test.txt", "r+");
    if (fichier != NULL)
    {
        // On peut lire et écrire dans le fichier
    }
```

```
        else
        {
            // On affiche un message d'erreur si on veut
            printf("Impossible d'ouvrir le fichier test.txt");
        }
        return 0;
    }
```

On fait toujours ça lorsqu'on ouvre un fichier et si on ne le fait pas et que le fichier n'existe pas, on risque un plantage du programme par la suite.

Fermeture du fichier : Si l'ouverture du fichier a réussi, on peut lire et écrire dedans.

Une fois qu'on a fini de travailler avec le fichier, il faudra le "fermer". On utilise pour cela la fonction **fclose** qui a pour rôle de libérer la mémoire, c'est-à-dire supprimer notre fichier chargé dans la mémoire vive.

Son prototype est :

Code : C

```
    Int fclose(FILE*pointeurSurFichier);c
```

Cette fonction prend un paramètre : le pointeur sur le fichier. Elle renvoie un **int** qui indique si elle a réussi à fermer le fichier. Cet **int** vaut :

- 0 : si la fermeture a marché
- EOF : si la fermeture a échoué. EOF est un **define** situé dans **stdio.h** qui correspond à un nombre spécial, utilisé pour dire soit qu'il y a eu une erreur, soit qu'on est arrivé à la fin du fichier. Dans le cas présent cela signifie qu'il y a eu une erreur.

Enfin, le schéma que nous allons suivre pour ouvrir et fermer un fichier sera le suivant :

Code : C

```
    int main(int argc, char *argv[])
    {
        FILE* fichier = NULL;
        fichier = fopen("test.txt", "r+");
        if (fichier != NULL)
        {
            // On lit et on écrit dans le fichier
                    // ...
```

```
            fclose(fichier); // On ferme le fichier qui a été
ouvert
        }
        return 0;

    }
```

On n'a pas mis le else ici pour afficher un message d'erreur si l'ouverture a échoué.

Il faut toujours penser à fermer le fichier une fois que l'on a fini de travailler avec, cela permet de libérer de la mémoire. Si on oublie ça, notre HDD risque à la fin de se saturer. Sur de petit programme ce n'est pas flagrant, mais sur un gros programme ça devient plus sérieux.

Oublier de libérer la mémoire, ça arrive. Ca nous arrivera d'ailleurs très certainement. Notre programme se mettra alors à utiliser plus de mémoire que nécessaire sans que vous n'arriviez à comprendre pourquoi. Bien souvent, il s'agit simplement d'un ou deux trucs comme des petits fclose oubliés.

2.4.5.2 - Lire et écrire dans le fichier :

Maintenant que nous avons écrit le code qui ouvre et ferme le fichier, nous n'avons plus qu'à insérer le code qui lit et écrit dedans. Nous allons commencer par voir comment écrire dans un fichier (ce qui est un peu plus simple), puis nous verrons ensuite comment lire dans un fichier.

Ecrire dans le fichier : Il existe plusieurs fonctions capables d'écrire dans un fichier. Ce sera à vous de choisir celle qui est la plus adaptée à votre cas.

Voici les 3 fonctions que nous allons étudier :

- fputc : écrit un caractère dans le fichier (un seul caractère à la fois).
- fputs : écrit une chaîne dans le fichier
- fprintf : écrit une chaîne "formatée" dans le fichier, fonctionnement quasi-identique à printf

Lire dans un fichier : Nous pouvons utiliser les fonctions suivantes :

- fgetc : lit un caractère
- fgets : lit une chaîne
- fscanf : lit une chaîne formatée

Se déplacer dans un fichier : On a parlé d'un terme "curseur" virtuel tout à l'heure. Nous allons l'étudier maintenant plus en détail. A chaque fois qu'on ouvre un fichier, il existe en effet un curseur qui indique notre position dans le fichier. Nous pouvons imaginer que c'est exactement comme le curseur dans un éditeur de texte (tel bloc-notes). Il indique où nous sommes dans le fichier, et donc où nous allons écrire.

En résumé : le système de curseur nous permet d'aller lire et écrire à une position précise dans le fichier. Il existe 3 fonctions à connaître :

- ftell : indique à quelle position vous êtes actuellement dans le fichier
- fseek : positionne le curseur à un endroit précis
- rewind : remet le curseur au début du fichier (c'est équivalent à demander à la fonction fseek de positionner le curseur au début).

ftell : position dans le fichier : Cette fonction est très simple à utiliser. Elle renvoie la position actuelle du curseur sous la forme d'un long. Le prototype de ftell est le suivant ::

Code : C

```
long ftell(FILE* pointeurSurFichier);
```

Le nombre renvoyé indique donc la position du curseur dans le fichier.

fseek : se positionner dans le fichier : La fonction fseek permet de déplacer le "curseur" d'un certain nombre de caractères (indiqué par déplacement) à partir de la position indiquée par origine. Le prototype de fseek est le suivant :

Code : C

```
int fseek(FILE* pointeurSurFichier, long deplacement, int origine);
```

- Le nombre déplacement peut être un nombre positif (pour se déplacer en avant), nul (= 0) ou négatif (pour se déplacer en arrière).

- Quant au nombre origine, vous pouvez mettre comme valeur l'une des 3 constantes (généralement des defines) listées ci-dessous :

  - **seek_set** : indique le début du fichier.
  - **seek_cur** : indique la position actuelle du curseur.
  - **seek_end** : indique la fin du fichier.

Voici quelques exemples pour bien comprendre comment utiliser déplacement et origine

- Le code suivant place le curseur 2 caractères après le début :

Code : C

```
fseek(fichier, 2, SEEK_SET);
```

1

- Le code suivant place le curseur 4 caractères avant la position courante :

Code : C

```
fseek(fichier, -4, SEEK_CUR);
```

On remarque que déplacement est négatif car on se déplace en arrière.

- Le code suivant place le curseur à la fin du fichier :

Code : C

```
fseek(fichier, 0, SEEK_END);
```

Si on écrit après avoir fait un fseek qui mène à la fin du fichier, cela rajoutera vos informations à la suite dans le fichier (ça complètera notre fichier).

En revanche, si on place le curseur au début et qu'on écrit, cela écrasera le texte qui se trouvait là. Il n'y a pas de moyen "d'insérer" de texte dans le fichier (à moins de coder soi-même une fonction qui lit les caractères d'après pour s'en souvenir avant de les écraser). Mais maintenant le problème qui reste à résoudre c'est comment on sait à quelle position on doit aller lire et écrire dans le fichier ? Alors ça, c'est nous qui gérons cette étape.

Si c'est un fichier que nous avons nous-mêmes écrit, nous savons comment il est construit. On sait donc où aller chercher nos informations (par exemple le temps de fonctionnement de la pompe est en position 0, le nombre de mise en marche/arrêt de la pompe est en position 50 etc...)

On fera une mise au point plus tard dans lequel nous comprendrons (si ce n'est pas déjà le cas) comment on fait pour aller chercher l'information qui nous intéresse. N'oublions pas que c'est nous qui définissons comment notre fichier est construit.

La fonction fseek peut se comporter bizarrement sur des fichiers ouverts en mode texte. En général, on l'utilise plutôt pour se déplacer dans des fichiers ouverts en mode binaire.
Quand on lit/écrit dans un fichier en mode texte, on le fait généralement caractère par caractère. La seule chose qu'on se permet en mode texte avec fseek c'est de revenir au début ou à la fin.

rewind : retour au début : Cette fonction est équivalente à fseek pour revenir à la position 0 dans le fichier. Si on a eu un magnétoscope un jour dans notre vie, eh bien c'est le même nom de la touche qui permet de revenir en arrière.

Le prototype est :

Code : C

```
void rewind(FILE* pointeurSurFichier);
```

Aussi étonnant que cela puisse paraître, ce chapitre ne vous a enseigné aucune nouvelle connaissance en langage C. Car on n'a pas encore parlée des pointeurs, des tableaux, des structures. Ici, nous n'avons fait qu'utiliser des fonctions de stdio, une bibliothèque standard. Nous avons donc fait l'étude d'une bibliothèque.

En fait, le langage C est vite appris pour ce qui est de la base. Le plus long est certainement d'arriver à comprendre les pointeurs, à ne pas confondre valeur et adresse en analogie avec l'assembleur, Motorola ou PIC.

Mais, une fois que c'est fait, on est en théorie capable de tout programmer. Le tout est de savoir utiliser des bibliothèques**,** c'est-à-dire faire ce qu'on vient de faire ici : apprendre à utiliser des fonctions de ces bibliothèques.

Sans bibliothèque, un programme ne peut donc rien faire. Même le **printf** nous n'aurions pas pu le faire, vu qu'il est situé dans stdio.

Si on arrive à comprendre et à retenir tout ce qu'on a vu dans ce chapitre alors ça facilitera la lecture du programme principale du système qu'on va concevoir. La prochaine partie sera entièrement dédiée à la conception de l'armoire et du programme.

2.4.5.3 - Création de fichiers graphiques :

Pour créer une courbe ou un diagramme sous VC++, on utilisera deux librairies simultanément : **SDL** et **FMOD**. Il faut savoir que la SDL est tout particulièrement adaptée à la création de jeux graphique, mais ça ne veut pas dire qu'elle est faite uniquement pour ça.

Nous allons réaliser une visualisation des données en SDL, plus précisément, on dessinera la courbe de fonctionnement du système à partir des données recueillies. C'est sûr que ce n'est pas aussi facile que ça mais on va essayer d'expliquer seulement les commandes principales qui sont nécessaires pour ce projet.

Pour commencer, on doit créer un programme qui accueillera et traitera l'image de la courbe. Il y a un outil sous VC++ qui gère tout ça, il ne nous reste plus qu'a écrire les commandes pour dialoguer avec ce programme.

On utilisera FMOD (qui possède d'immense capacité). Elle peut lire aussi des CD, effectuer des enregistrements, des modifications, etc... Nous allons ici nous intéresser à un de ses modules, appelé **DSP**. Ce sont justement celles qui nous donnent des informations sur les données, afin que nous puissions visualiser la courbe. Pour cela, il va falloir activer ce module :

Code : C

```
FGRAPH_DSP_SetActive(FGRAPH_DSP_GetFFTUnit(), 1);
```

Récupérer les données : Maintenant que le module DSP est activé, on peut faire appel à la fonction FGRAPH_DSP_GetSpectrum(). Il n'y a aucun paramètre à lui donner. En revanche, elle offre des choses très intéressantes : un pointeur vers un tableau de 512 float. Il nous faudra donc déclarer un pointeur vers float. On peut ensuite parcourir ce tableau pour obtenir les valeurs de chacune des cases :

Code : C

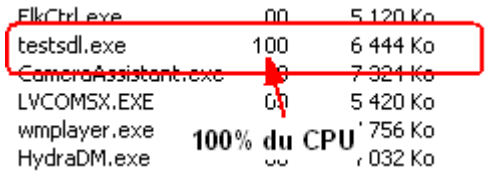
```
donnée[0] // première case (à gauche)
donnée [1]
donnée [2]
...
donnée [509]
donnée [510]
donnée [511] // dernière case (à droite)
```

Chaque case est un nombre décimal compris entre 0 (rien) et 1 (maximum). Notre travail va consister à afficher une barre plus ou moins grande en fonction de la valeur que contient chaque case du tableau. Par exemple, si la valeur est 0.5, nous devrons tracer une barre dont la hauteur correspondra à la moitié de la fenêtre destinée à accueillir le graph. Si la valeur est 1, elle devra faire toute la hauteur de la même fenêtre. Mais les barres doivent bouger au fur et à mesure du temps. Comme les données changent tout le temps, il faut mettre à jour le graphique. En effet, le tableau de 512 floats que nous renvoie FMOD change toutes les 25 ms (pour être à jour par rapport aux nouvelles données actuelles). Il va donc falloir dans le code de relire le tableau de 512 floats (en refaisant appel à FGRAPH_DSP_GetSpectrum()) toutes les 25 ms, puis mettre à jour le graphique en barres.

Vue les informations concernant la gestion du temps de SDL, on va faire un résumé des différentes commandes dont on a besoin [16]:

- **SDL_Delay** : permet de mettre en pause le programme un certain nombre de millisecondes.
- **SDL_GetTicks** : retourne le nombre de millisecondes écoulées depuis le lancement du programme.
- **SDL_WaitEvent** : Cette fonction mettait en "pause" le programme (un peu à la manière de SDL_Delay) tant qu'il n'y avait pas d'évènement.
- **SDL_PollEvent** : elle renvoie une valeur qu'il y ait eu un évènement ou pas. On dit que la fonction n'est pas "bloquante" : elle ne met pas en pause le programme, la boucle infinie va donc tourner indéfiniment tout le temps.

Actuellement, notre programme tourne en boucle indéfiniment à la vitesse de la lumière (enfin presque). Il consomme donc 100% du CPU. Si on fait CTRL + ALT + SUPPR (onglet "Processus"), on verra ça sous Windows :



Comme nous pouvons le voir, notre CPU est utilisé à 100% par notre programme testsdl.exe. Il vaut mieux utiliser le moins de CPU possibles pour que l'utilisateur puisse faire autre chose sans que son PC ne "rame". On va reprendre exactement le même code source mais on va lui ajouter en plus un **SDL_Delay** pour patienter le temps qu'il faut afin que ça fasse 30ms. C'est comme si on effectue un échantillonnage de 30 ms pour chaque acquisition.

Insérer le graphe dans le fichier test.txt :

Pour mettre la courbe dans l'historique d'impression, on utilisera cette instruction comme commande. À la fin on aura, dans notre fichier, une courbe mise à jour chaque instant. Un exemple de ce fichier sera donné dans la section : impression de l'historique de fonctionnement.

2.5 - Conception de l'armoire de commande :

Dans un premier lieu, on va essayer de définir qu'est ce qu'une station de pompage ainsi que le principe de fonctionnement des pompes utilisées, afin de créer notre cahier des charges.

Pour un deuxième lieu, on va s'intéresser à l'étude de la variation de la consommation d'eau. Cette dernière est différente d'une période à une autre. Les courbes de modulation des consommations journalières et mensuelles vont nous permettre de dégager un organigramme de fonctionnement suivant les tarifs d'énergie de l'électricité (Jour/Pointe/Nuit). Enfin, une partie sera consacrée à la mise en équation de la station de pompage.

2.5.1 - Description de la station de pompage :

Une station de pompage est un système hydromécanique destiné à pomper l'eau potable d'un niveau géographique bas à un niveau plus haut [18]. On distingue dans une station de pompage : un poste de transformation, un groupe électropompe, une armoire de commande et de protection, un conduit de raccordement de protection, un réservoir, des accessoires divers (vanne, clapets, ….).

Le principal élément dans une station est les pompes, leur rôle est de refouler une quantité d'eau d'un niveau bas à un niveau haut. C'est pour cela qu'une pompe se compose de deux parties essentielles qui sont : l'aspiration et le refoulement.

Mais il diffère d'une pompe à une autre suivant des critères bien définis comme le débit, la pression, la hauteur, le rendement et la puissance [19]. Pour cela, on va étudier les pompes centrifuges qui ont des performances très variables ce qui leur confèrent à des utilisations diverses : Transfert de fluide (débit important), transmission de puissances (pression importante).

Une pompe centrifuge est construite pour répondre à des conditions précises de

fonctionnement : débit Q à élever à une hauteur H. Quand la pompe ne comporte qu'une seule cellule elle est dite monocellulaire et se compose d'une roue et d'une volute ou corps de pompe qui joue le rôle de diffuseur de la pompe multicellulaire. Le problème se pose sur l'économie d'énergie pour une station de pompage. Tout d'abord, nous allons analyser des données tirées d'un document qui nous a été donné par un entrepreneur, ce document présente des données de consommation d'eau enregistrées à la station de pompage de la commune de Boutaleb.

Pour montrer que la consommation d'eau se diffère d'une saison à une autre. Nous avons tiré quelques valeurs dans les tableaux suivants.

Consommation maximale	92 000 m^3
Consommation minimale	18 800 m^3
Consommation moyenne	55 400 m^3

Tableau 2.3 : Exemple de consommation du mois d'août (Saison d'été)

Consommation maximale	68 400 m^3
Consommation minimale	16 400 m^3
Consommation moyenne	42 400 m^3

Tableau 2.4 : Exemple de consommation du mois de janvier (Saison d'hiver)

On constate d'après les deux tableaux que la consommation d'eau varie en passant d'une saison à une autre c'est à dire en été on consomme plus qu'en hiver. Mais aussi en remarque que la quantité d'eau consommée varie de façon remarquable durant toute la journée.

D'après une étude personnelle, on trouve que la consommation d'eau augmente pendant les heures de pointe et diffère d'une période à une autre, en plus le prix de l'énergie est différent pendant les trois périodes du jour. Notre but consiste alors à optimiser le fonctionnement des pompes suivant les tarifs, c'est-à-dire qu'on veut commander le démarrage et l'arrêt des moteurs.

D'une part, si le niveau d'eau atteint son niveau bas, il faut que les moteurs démarrent. D'autre part, il faut essayer de minimiser le temps de fonctionnement des

motopompes durant la période critique définie préalablement où le tarif de l'énergie est cher (heures de pointe). Par la suite, il faut démarrer les moteurs de telle sorte qu'on puisse avoir un réservoir plein au début de la période critique. Pour assurer cet objectif on va faire une étude sur l'optimisation.

2.5.2 - Etude de l'optimisation du fonctionnement d'une station de pompage :

C'est une étude faite, d'une part pour contrôler l'énergie consommée par les motopompes durant une période bien déterminée (consommation journalière) et d'autre part pour assurer que notre unité de contrôle (microcontrôleur) est capable de stocker certaines données concernant la consommation d'eau en fonction du temps, d'où le traçage de la courbe de modulation annuelle.

EXEMPLE :

Heure	00 → 04	04 → 08	08 → 12	12 → 16	16 → 20	20 → 24
Consommation (m^3)	5 000	20 000	52 000	39 000	27 000	20 000

Tableau 2.5 - Tableau de mesures journalier

Nous avons partagé les heures du jour en six périodes, chaque période comporte quatre heures. Notre unité de contrôle va calculer la quantité d'eau consommée pendant cette période et par la suite nous obtiendrons une courbe de modulation journalière représentée par la figure 2.14.

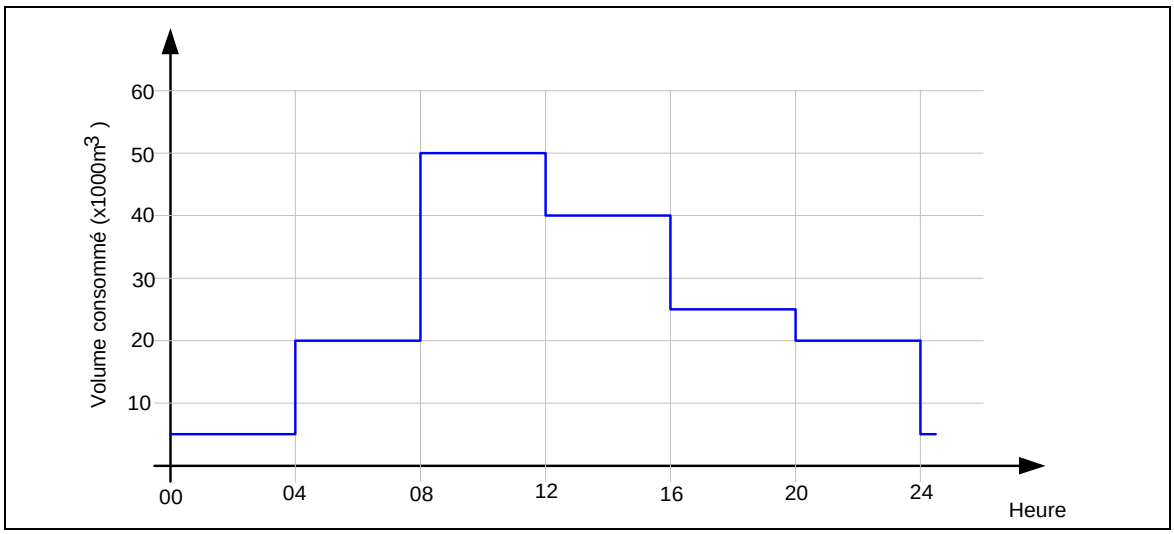
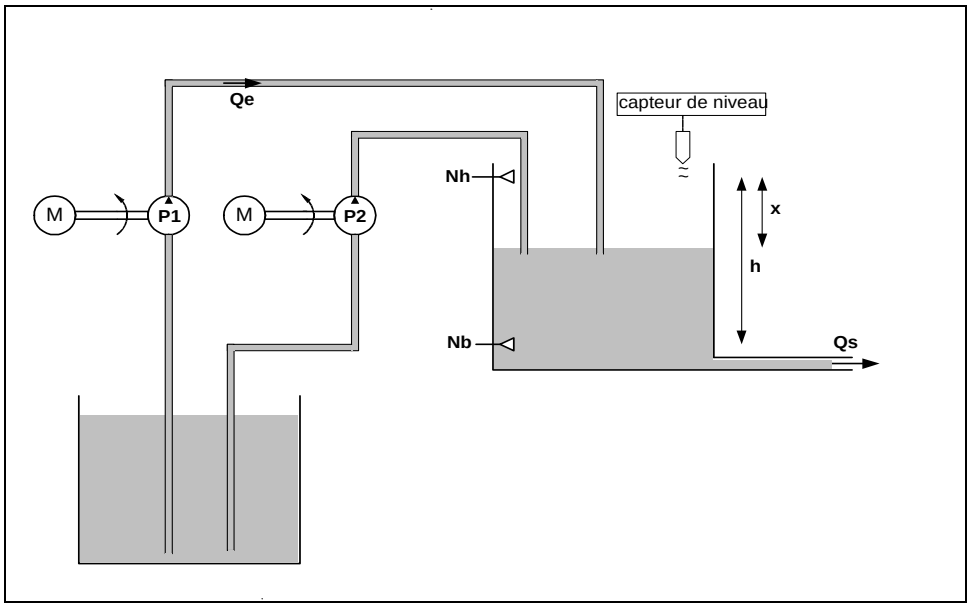


Figure 2.14 - Courbe de modulation journalière

Cette étude va être répétée chaque jour, ensuite une courbe de consommation moyenne va être déduite et stockée en mémoire qui servira comme base de données pour les calculs d'optimisation.

Après avoir enregistré les données de chaque mois, l'unité de contrôle va stocker ces données en mémoire pour un éventuel traitement externe, nous avons prévu alors une liaison avec un PC. Après avoir tracé la courbe de modulation, et pour attaquer la partie programmation, il faut tout d'abord identifier certains paramètres qui seront utiles pour notre étude et cela en suivant le schéma de la station de la figure 2.15 (qu'on a simplifié au maximum).



P1,P2 : Pompes
M : moteur
Qe : Débit des pompes
Qs : Débit de consommation
h : Hauteur du réservoir
x : Volume vide
Nh : capteur de niveau haut
Nb : capteur de niveau bas

Figure 2.15 - Schéma simplifié de la station de pompage

Etant donnée la hauteur (h), la surface du réservoir (S) et le volume d'eau pompée (débit des pompes constant Q_e) on peut déterminer par un simple calcul la quantité consommée d'eau pendant un temps t_2-t_1.

$$\mathbf{Q_S = S.[x(t_2) - x(t_1)] - Qe} \qquad (3)$$

2.5.3 - Algorithme d'optimisation :

Pour optimiser le fonctionnement d'une station de pompage d'eau notre algorithme sera muni de plusieurs conditions, on prendra ce qui suit comme un cahier des charges :

Les règles de base :

- Si Nb=1 alors P1=1 ;

- Si Nh=1 alors P1=0 ;

Si le niveau bas est atteint alors la pompe principale P1 doit forcément fonctionner pour répondre aux besoins des consommateurs. Si le réservoir est plein c'est à dire que le niveau haut est atteint alors il n'aura pas de pompage.

Les règles de sécurité :

- Si P1= 1 et x(t=15mn) - x(t=0) > 0 alors P2=1
- Si P1= 1 et Si P2=1 et x(t=15mn) - x(t=0) < Q1 alors P2 = 0

Si la pompe P1 fonctionne et le niveau du réservoir diminue pendant un temps de fonctionnement de 15mn alors la pompe de secours P2 doit fonctionner.

Si P1 et P2 fonctionnent en même temps et si la consommation est inférieure au débit de la pompe P1 alors il faut arrêter la pompe de secours.

Les règles d'optimisation :

Sachant que le temps de fonctionnement t_f nécessaire pour le remplissage du réservoir est estimé à une durée bien déterminée notre unité de contrôle va vérifier est ce que le volume d'eau dans le réservoir suffira à la consommation durant la période de pointe sinon elle entamera le pompage à l'instant heure de pointe H_p moins t_f.

Si nous sommes dans la période de pointe et le niveau dans le réservoir est bas alors la pompe principale P1 doit fonctionner le temps nécessaire juste pour remplir la quantité de consommation prévue pour cette période. Notre unité de contrôle doit toujours entamer une période jour par un réservoir plein. On obtient donc les règles suivantes :

- Si t = H_p- t_f et Qs (H_p) > (h-x) alors P1=1
- Si t = H_p et N_b=1 alors P1=1 jusqu'à (h-x) > Qs (H_p)
- Si t =Hj – t_f et N_h=0 alors P1=1

Remarque : Les règles d'optimisation peuvent être aléatoires pendant le fonctionnement alors que les règles de base et de sécurité sont toujours fixes.

En conclusion, nous pouvons dire que la consommation d'eau varie d'une saison à une autre, nous voulons alors optimiser le fonctionnement des pompes à des conditions précises, c'est à dire commander le démarrage et l'arrêt des moteurs. Pour assurer cet objectif nous réalisons un montage à base d'un PC standard dont le programme de commande sera basé sur l'ensemble des algorithmes de fonctionnement qui ont été établis dans ce chapitre.

2.5.4 - Conception de l'armoire de commande :

Dans cette partie nous présenterons une description de l'armoire pour répondre aux spécifications de notre cahier de charges et en abordant la conception de chaque partie du système afin d'obtenir une schématisation complète et précise. Notons que nous avons utilisé le logiciel XRelais 3.1 pour la conception du schéma.

Notre montage, comme l'indique la figure 2.16, est muni de plusieurs unités qui assurent le bon fonctionnement du système. La liaison entre l'armoire et le PC se fait à travers des relais 5/220 V ainsi que d'un câble LPT bidirectionnel.

Vue le faible courant délivré par le port LPT, on doit impérativement utiliser un étage amplificateur, soit avec des transistors, soit avec un circuit spécial à 8 entrées/sorties comme l'ULN 2803.

En effet, on trouve l'unité de traitement et de contrôle des données qui est le PC et des actionneurs qui assurent la commande des deux pompes (relais).

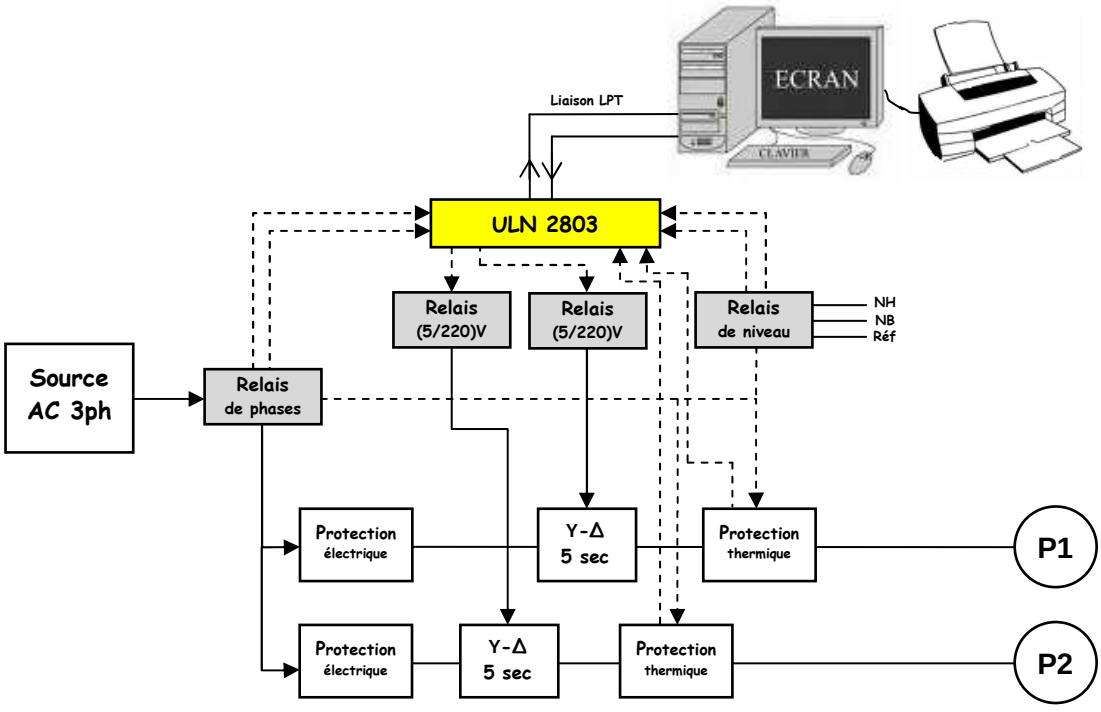


Figure 2.16 - Diagramme synoptique du système

La liaison à la norme LPT :

Les liaisons parallèles permettent la communication entre deux systèmes numériques en 8 liaisons parallèles simultanées et bidirectionnelles. Les niveaux logiques et électriques sont détaillés dans le tableau 8. Rappelons que ces chiffres ne sont pas systématiques et qu'il convient de les vérifier auprès du constructeur.

	0 logique	1 logique
Entrée DC	0 à 0,8 V 0,4 mA max	2 à 5 V 8 mA max
Sortie DC	0 à 0,4 V 0,4 mA max	2,5 à 5 V 8 mA max

Tableau 2.5 - Niveaux logiques et niveaux électriques TTL [20]

Pour pouvoir dialoguer avec le PC, on utilise un câble imprimante. Le PC utilise son circuit interne de communication parallèle (se trouvant dans n'importe quelle carte mère), intégrant des tests de tension électrique permettant de différencier et évitant tout risque d'erreur côté PC.

Le standard IEEE 1284 (1994) [21] définit 5 modes d'opérations pour un port parallèle : SPP, EPP, ECP, Nibble Mode et Byte Mode. Nous allons utiliser le protocole standard (SPP, Standard Parallel Port).

Dans une communication parallèle, les données sont transférées de l'émetteur au receveur par groupe de 8 bits, à travers 8 lignes. La transmission va ainsi être très rapide, 50 à 100 KB de données par seconde (beaucoup plus vite qu'en série qui est limitée à 10 KB) mais les prises sont plus encombrantes et, du fait que le protocole de communication parallèle a été étendu (EPP et ECP), les périphériques peuvent contenir plus d'électronique (plus chers) pour implémenter ce protocole. Mais tout cela ne posera pas un problème majeur dans la conception de notre système.

L'ULN 2803 :

D'âpres la documentation technique de l'ULN 2803, un réseau de 8 Darlington devrait permettre le passage de 50V/500mA. Nous allons l'utiliser à 5V/500mA pour exciter les bobines internes des relais. Les détails du circuit sont représentés par la figure 2.17 :

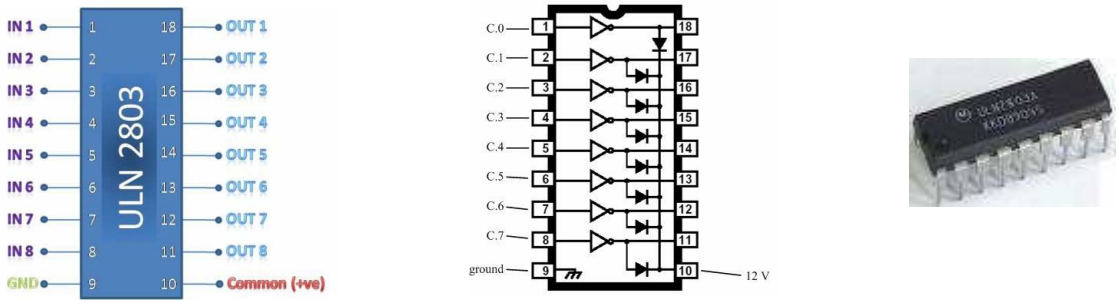


Figure 2.17 - Spécifications de l'ULN 2803 [21]

Pour implanter l'ULN2803 au système il faudra surement lui confectionner un circuit imprimé, incluant des diodes de protection avec leurs résistances (560 Ω) ainsi que des résistances de protection du port LPT (100 KΩ).

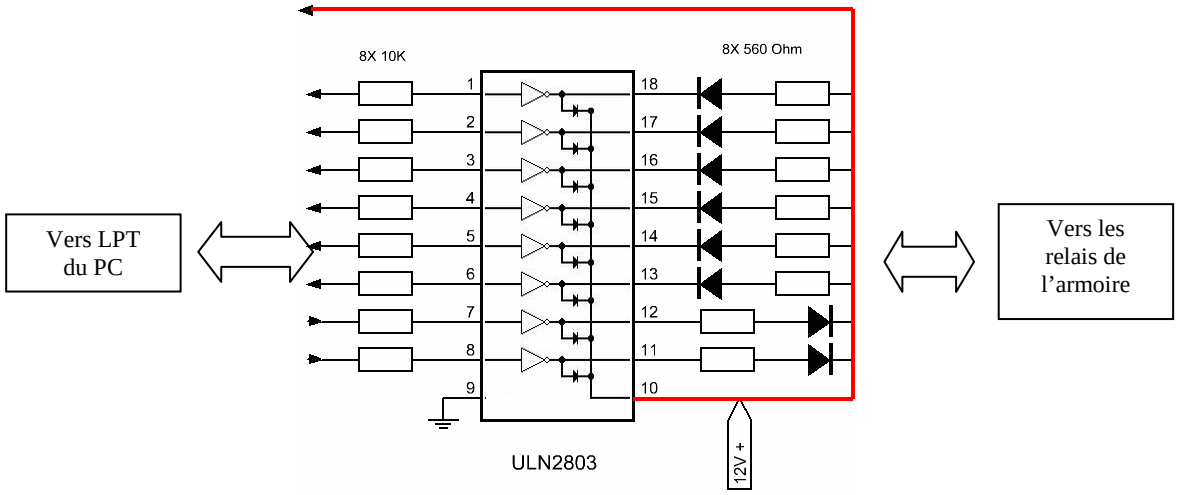


Figure 2.18 - Branchement de l'ULN 2803 dans le système

Les relais :

Dans notre application nous avons utilisé deux relais dans l'un est principal et l'autre de secours. Ces 2 sorties sont disponibles sur connecteur à vis. Le relais qu'on a utilisé dans l'exemple au début du chapitre (§2.3.2) et un relais qui ne consomme pas énormément de courant (10 mA) c'est pour ça qu'on n'a pas utilisé un circuit d'amplification, mais vue que les modèles de pompes consomment des pointes de courant importantes (200A) on devra utiliser des éléments imposant que le choix des relais doit être adapté aux caractéristiques des contacteurs et plus précisément à leurs bobines de maintien.

La logique de câblage :

Au cours de l'étude du thème qui nous est proposé et que nous présenterons, nous serons amenés à produire des schémas électriques répondant aux normes de sécurité. Toutes les mesures de sécurité ont leur importance mais certaines règles de base doivent être connues par cœur. Les normes en Algérie étant de concevoir une commande alimentée par une phase et le neutre. Les consignes seront comme suit :

- Toute armoire électrique doit pouvoir être consignée et déconsignée. En tête de l'équipement on doit trouver un interrupteur sectionneur (cadenassable si possible) permettant de consigner ou de déconsigner l'équipement tout entier de l'extérieur de l'armoire électrique.
- Les différents états de l'équipement doivent être signalés par des voyons lumineux dont les couleurs ont une signification définie:
  - verrine incolore : présence tension sur le système
  - verrine orange : système consigné
  - verrine rouge : système déconsigné
- Tout équipement ou système électrique doit être doté d'un appareil de protection des personnes (disjoncteur ou interrupteur) suivant le schéma de liaison à la terre auquel il est soumis. Les protections des biens et matériels sont dimensionnés en fonction des récepteurs à alimenter.
- Le coffret électrique doit être alimenté par un câble 5 conducteurs (3 phases + neutre + PE) ou 4 conducteurs (3 phases + PE) raccordé directement en amont de l'interrupteur sectionneur général sans passer par les borniers.
- L'entrée du câble d'alimentation dans le coffret doit être faite par presse-étoupe.

- Le passage de ce câble dans les goulottes est interdit et son identification doit être claire.
- Suivant la fonction et la nature du réseau dans lequel le conducteur est placé, celui-ci a une couleur définie par la norme:
  - noir: circuit de puissance (alternatif et continu)
  - rouge: circuit de commande (alternatif)
  - bleu: circuit de commande (continu)
  - bleu clair : neutre des circuits de puissance (quand ils ne sont pas utilisés pour la mise à la terre)
  - orange: circuit de commande en permanence sous tension
  - vert/jaune: protection électrique
- Chaque conducteur doit porter un repère correspondant à celui du schéma électrique.
- C'est le plus simple à mettre en œuvre. Tous les conducteurs toujours soumis au même potentiel portent un même numéro. Le numéro change lorsqu'il y a possibilité d'ouverture du circuit.

L’impression de l’historique de fonctionnement :

Vue que j’étais indisponible pour la période d’installation de l’armoire, on n’a pas pu installer le système sur site comme prévue, mais le système a été testé en atelier (sans le câblage de puissance). La commande entre relais/contacteurs fonctionne parfaitement, elle montre que les contacteurs de démarrage réagissent instantanément aux directives des relais et qu’il n’y a aucun court-circuit dans la partie commande. De ce fait, vu que le système n’a pas été installé sur site, le rapport ne peut être généré. Mais théoriquement le rapport simplifié du fonctionnement sera similaire à celui qui suit :

Historique de fonctionnement, test.txt

21/03/2010 au 28/03/2010

nbr dém P1 : 6
temps fct P1/jours: 15 h

nbr dém P2 : 2
temps fct P2/jours: 5 h

Quantité moy déb/jour : 171 460 m^3

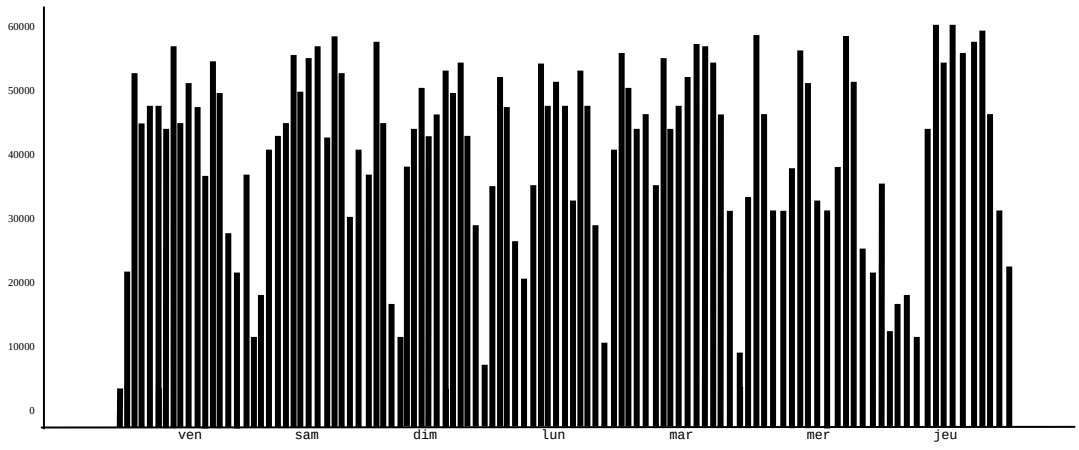
phase : ok
déphase : ok
R therm 1: ok
R therm 2: ok

default : 0

Q déb/sem [m^3]

dimanche	lundi	mardi	mercredi	jeudi	vendredi	samedi
173 265	171 033	170 979	169 836	171637	173 006	170 469

Q semaine = 1 200 225 m^3



Le graphe qu'on aura a l'impression est semblable à celui de la figure 2.15, on pourra varier le temps d'échantillonnage pour avoir une période plus longue entre chaque mesure vue qu'il n'y a pas de changement d'état pour le débit d'eau pendant le démarrage des pompes.

La largeur de chaque barre est fixée préalablement à 2 heures de fonctionnement, cela implique que pour chaque jour on aura 12 barres représentatives du débit d'eau. Ce graphe est

juste un test fait à partir des résultats de la station de pompage pour l'année 2009, qui nous ont été gentiment restitué par le responsable de la station. Ainsi on verra que les heures de pointes sont situées au milieu de la journée vue qu'il faut alimenter la population en eau pour qu'ils puissent l'utiliser le soir, ainsi on aura un fonctionnement des deux pompes P1 et P2 pour avoir le rendement maximal de la station. En revanche la station ne débite pas une grande quantité d'eau au milieu de la nuit.

Ces résultats sont juste une partie de ce qu'on peut mettre sur l'historique de fonctionnement d'une station de pompage, on pourra mettre plus de détails si cela nous donne envie comme par exemple affichage du débit d'eau, la quantité maximale ou minimale de l'eau débitée, l'heure de chaque démarrage et arrêt, etc…

2.6 - Conclusion :

L'étude et la conception des montages était pour nous une expérience très enrichissante du fait que nous avons vécu un cas réel de conception, il nous a fallut pour ça apprendre un nouveau langage de programmation.

Les expériences que nous avons menées durant ce projet nous ont montré que l'étude de la partie programmation est obligatoire et c'est elle qui commande tout le fonctionnement de l'armoire. Une approche pratique a été envisagé mais malheureusement, vue mon indisponibilité, nous avons donc pu tester notre montage sur le site.

Pour le dernier chapitre on va simuler tout le système sous Matlab pour vérifier son bon fonctionnement, ainsi élargir la durée et déterminer ses performances et ses limites.

Chapitre 3

Simulation du dispositif sous MATLAB

3.1 - Introduction :

Comme tout projet, l'expérimentation en labo ou en atelier est une étape obligatoire pour valider un prototype mais la simulation numérique s'impose comme une deuxième approche dans la plupart des disciplines [25]. Elle ne remet nullement en cause l'efficacité de l'expérimentation mais elle s'efforce à apporter des explications aux phénomènes étudiés pour mieux comprendre et prévoir des perspectives futures.

Le logiciel MATLAB joue plus que jamais un rôle respectable dans le monde des logiciels de simulation. Possédant une large base de données et incluant l'outil Simulink pour les simulations électrotechniques, on pourra l'utiliser pour reproduire le comportement d'un relais de commande, d'un relais de phase, d'un relais de niveau, d'un contacteur électrique ou thermique et de tout ce qui compose notre dispositif.

Dans ce dernier chapitre on essayera de modéliser le système qu'on a réalisé. En premier lieux on créera une boite noire chaque partie du dispositif puis on passera à la modélisation du système global. On finira par exposer les résultats de simulation ainsi que les conclusions et perspectives.

3.2 - Modélisation des éléments du système :

L'outil Simulink sera employé pour développer un modèle mathématique représentant chaque élément, on utilisera pour cela les blocs contenus dans Simulink. On reproduira fidèlement le comportement électromécanique et électromagnétique des relais et des contacteurs en prenant compte des caractéristiques techniques du constructeur.

3.2.1 - Modèle du disjoncteur :

Vue qu'il y a plusieurs types de disjoncteurs on est obligé de mettre en place un modèle universel rassemblant tous les cas de protection, je fais allusion bien sur au disjoncteur magnétothermique différentiel. Il fonctionne par différence entre 2 champs magnétiques, chaque phase et neutre forme un bobinage autour d'un tore. Quand tout va bien, les phases et neutre ont chacun le même champ magnétique et donc ils s'annulent les uns les autres. Quand un défaut ou une fuite survient, les champs magnétiques ne sont plus les mêmes et vont former un autre champ magnétique dans une bobine dite de détection et ainsi attirer un contact mobile qui ouvrira le circuit.

Cela se traduit par le système d'équations suivant :

Si $H_{PH} = H_N$ alors disjoncteur fermé

Si $H_{PH} \neq H_N$ alors disjoncteur ouvert

La coupure d'un circuit en charge implique la formation systématique d'un arc électrique entre les contacts. Le courant I circule à travers l'arc, ce qui retarde la coupure, mais qui génère entre les contacts une tension U_{arc}, dite tension d'arc, qui s'oppose à la tension du réseau Er qui l'a créé. Le courant diminue, jusqu'à se couper, dès que la tension d'arc est supérieure à celle du réseau, conséquence de l'équation du circuit :

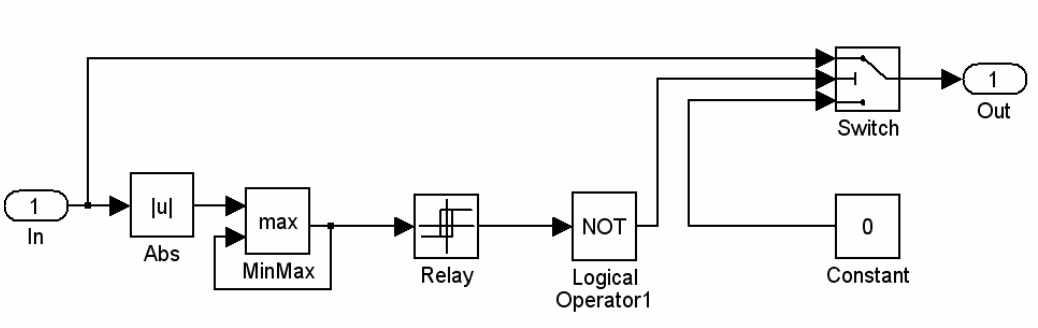


Figure 3.1 - Modèle du disjoncteur sous Matlab/Simulink

3.2.2 - Modèle du contacteur :

Lorsque sa bobine est alimentée, elle crée un champ magnétique qui attire les contacts liés mécaniquement et ainsi ferme le circuit. La bobine est branchée sur le circuit de commande de ce fait, le contacteur sera un organe de puissance et son modèle mathématique sera à base d'interrupteurs.

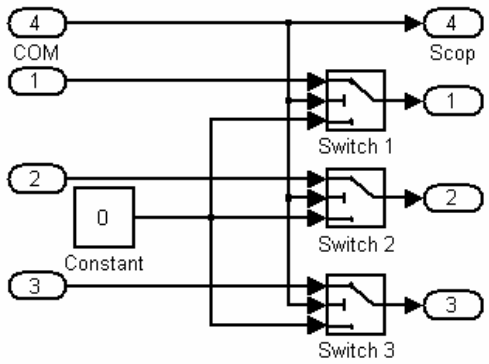


Figure 3.2 - Modèle du contacteur sous Matlab/Simulink

3.2.3 - Modèle du relais de phases :

Le relais détecte l'ordre incorrect des phases ainsi que la perte de phase, cela permet de couper l'alimentation si la tension est incorrecte. Le relais opère lorsque toutes les phases sont présentes, l'ordre des phases est correct et que chaque tension de phase soit dans la tolérance désirée (± 10% ou ± 15%). De ce fait, on peut réaliser un modèle en comparant une référence

sinusoïdale avec celles du réseau ainsi que des comparateurs en cascade relié directement aux interrupteurs internes du relais.

En plus, lors d'un déphasage des tensions du réseau, le relais ne s'ouvre pas instantanément. En effet, le déphasage peut être temporaire causé par une perturbation, cela n'infecte pas le fonctionnement de la pompe mais peut s'avérer fatal lors d'un délai supérieur à 10s. De ce fait, une temporisation de contrôle sera prise en considération pour l'élaboration du modèle.

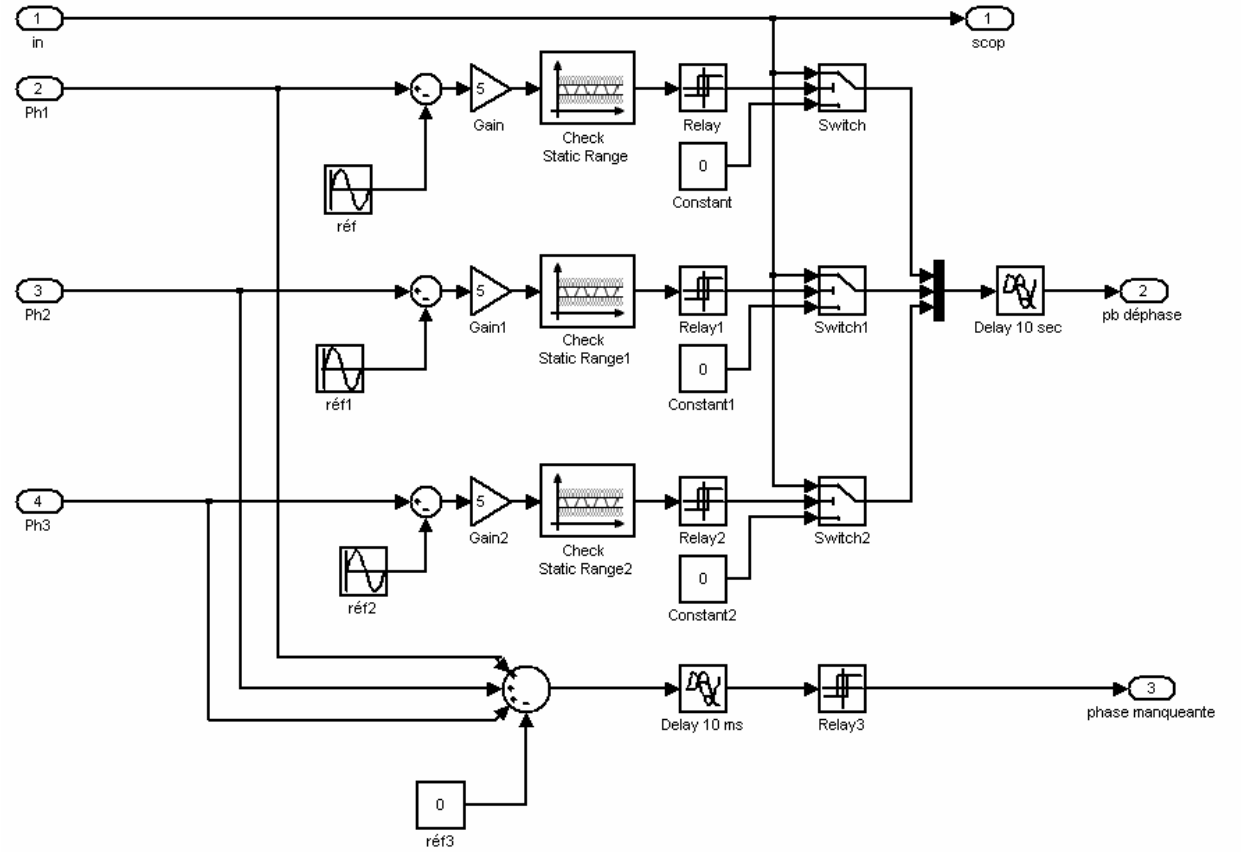


Figure 3.3 - Modèle du relais de phase sous Matlab/Simulink

3.2.4 - Modèle du relais de niveau :

Le principe de fonctionnement du relais de niveau est très spécial, il est basé sur une logique combinatoire à base de portes logiques et de bascules, de ce fait son modèle se fera à base de portes AND, OR et NOT.

Le relais comporte 3 sondes : une de niveau haut (NH), une de niveau bas (NB) et une sonde de référence (Réf). Grace à la conduction de l'eau, les deux sondes NH et Réf se touchent en cas où le réservoir est plein, on aura donc la fermeture du relais activant ainsi la pompe. Le niveau d'eau baissera et la sonde NH ne sera plus immergée mais le relais ne

s'ouvrira pas pour autant, ça attendra jusqu'à ce que le niveau atteigne la sonde NB et c'est à ce moment-là que le relais s'ouvrira arrêtant ainsi le fonctionnement de la pompe. Le réservoir étant vide, il va se remplir à nouveau jusqu'à atteindre la sonde NH et le processus recommencera de nouveau. Ce procédé est très utilisé dans les stations de pompage car ça assure la mise en marche et l'arrêt automatique des pompes d'une manière économique. La figure 3.4 représente la manière la plus utilisée pour le câblage d'un relais de niveau.

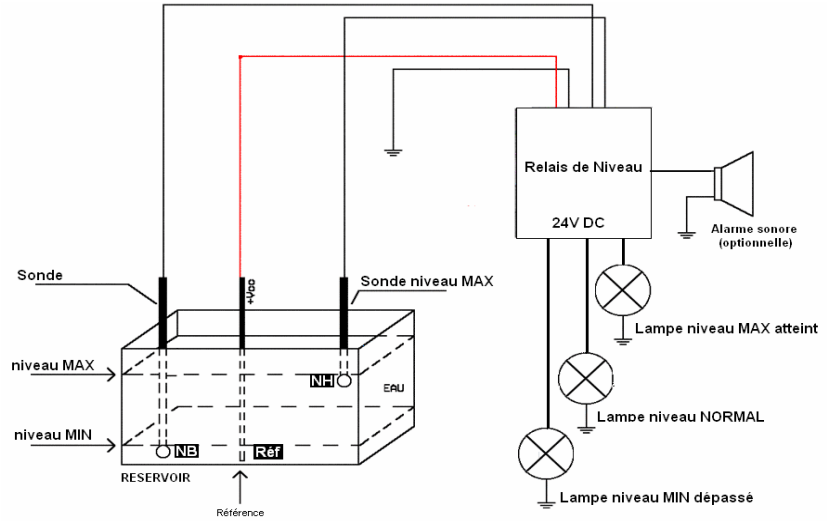


Figure 3.4 - Schéma de câblage d'un relais de niveau

Vue la séquence d'ouverture et de fermeture de ce genre de relais, il nous faudra élaborer une table de vérité qui nous servira à concevoir le modèle du relais. En suivant ce qui a été dit plus haut, la table de vérité sera facile à créer, elle sera détaillée dans le tableau 3.1.

Les entrées N1 (Niveau MIN) et N2 (Niveau MAX) sont à "1" s'il n'y a pas contact avec l'eau. Le tableau suivant reflète, en fonction des états des entrées, les actions que le relais doit effectuer.

Niveau MAX (N2)	Niveau MIN (N1)	Etat du réservoir	Actions associées	Lampe niveau MAX atteint	Lampe niveau NORMAL	Lampe niveau MIN dépassé
N2 = 0	N1 = 0	Vide	Q4 = 1 NB dépassé	Q3 = 0	Q2 = 0	Q1 = 1 Allumée
N2 = 0	N1 = 1	Normal	Niveau normal	Q3 = 0	Q2 = 1 Allumée	Q1 = 0
N2 = 1	N1 = 0	Impossible	-	-	-	-
N2 = 1	N1 = 1	Plein	Q4 = 1 NH atteint	Allumée Q3 = 1	Q2 = 0	Q1 = 0

Tableau 3.1 - États logiques en fonction des niveaux d'eau

La table de vérité étant faite, il ne restera plus que construire le modèle sous Matlab en utilisant le bloc logique se trouvant dans : Simulink/Math Operations/Logical Operator.

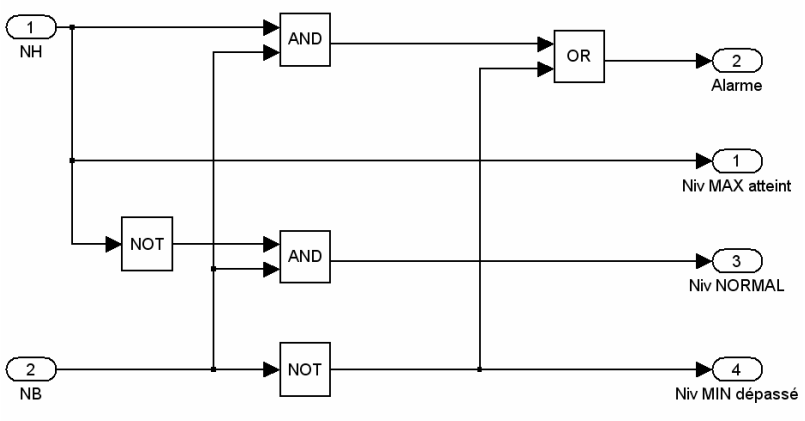


Figure 3.5 - Modèle du relais de niveau sous Matlab/Simulink

3.2.5 - Modèle du relais thermique :

En cas de surcharge, le relais thermique n'agit pas directement sur le circuit de puissance. Un contact du relais thermique ouvre le circuit de commande de la bobine de maintien ainsi le contacteur coupe le courant dans le récepteur. Il faut aussi ajouter une temporisation comme pour le relais de phases.

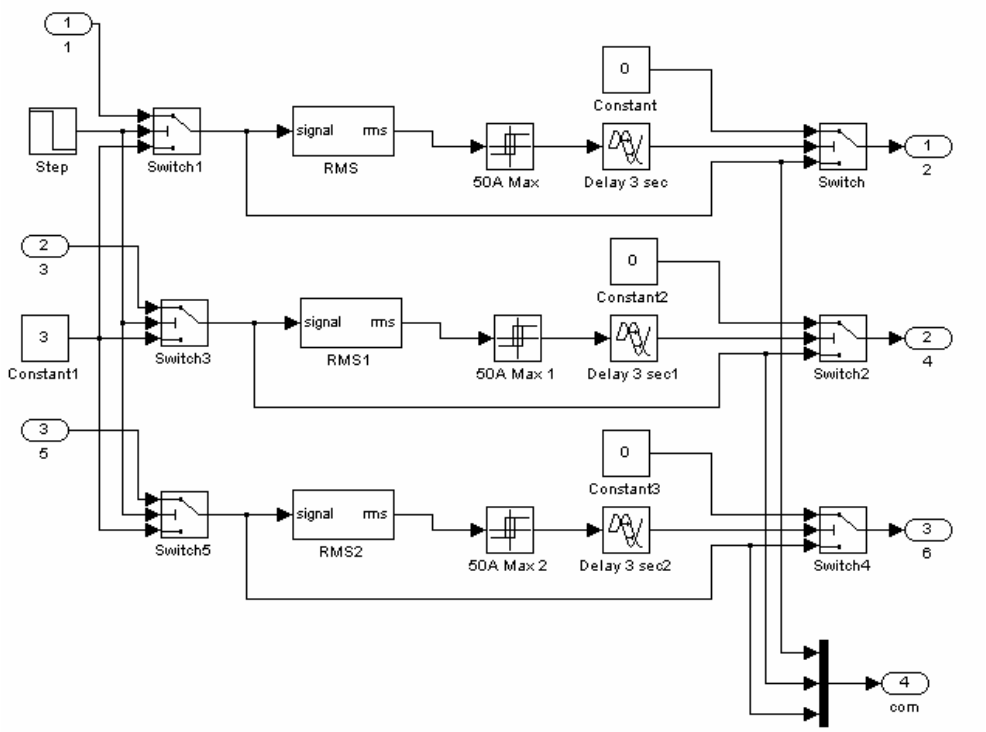


Figure 3.6 - Modèle du relais thermique sous Matlab/Simulink

3.2.6 - Modèle de la pompe :

Vue qu'une pompe est un moteur asynchrone à cage, on utilisera directement le bloc de la MAS inclue dans la bibliothèque de Matlab ainsi que le modèle mathématique couramment utilisé.

Modèle pour un démarrage rhéostatique Modèle pour un démarrage Y- Δ

Figure 3.7 - Modèle d'une pompe sous Matlab/Simulink

3.3 - Simulation du l'exemple du chapitre 2 :

Nous allons simuler l'exemple du chapitre 2. On utilisera les modèles vus précédemment pour modéliser le démarrage statorique. Le modèle de simulation est comme suit :

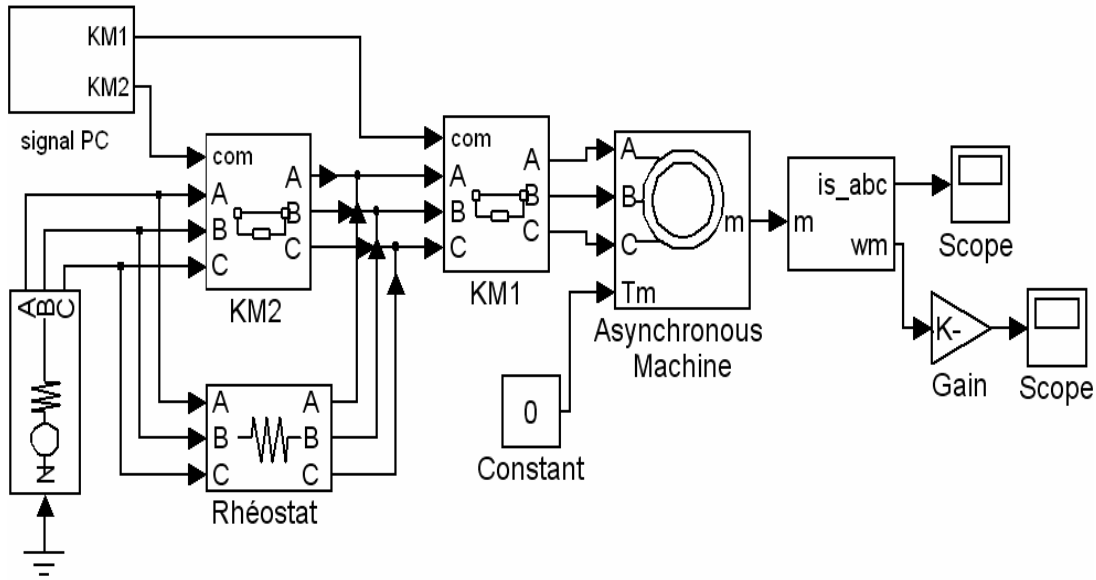
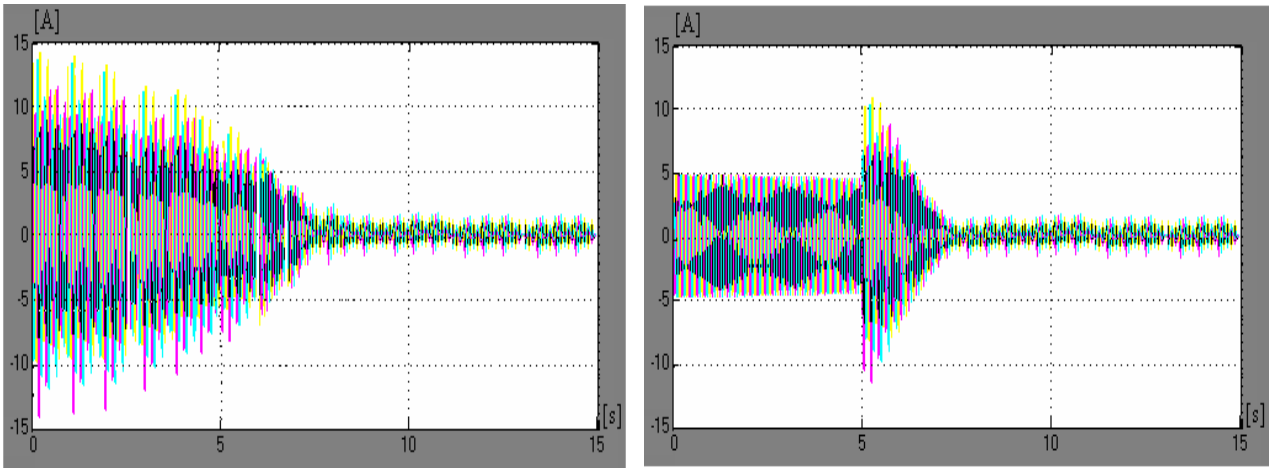


Figure 3.8 - Simulation de l'exemple sous Matlab/Simulink

La comparaison entre les résultats de simulation avec et sans démarrage statorique sont représentés par la figure 3.9 :

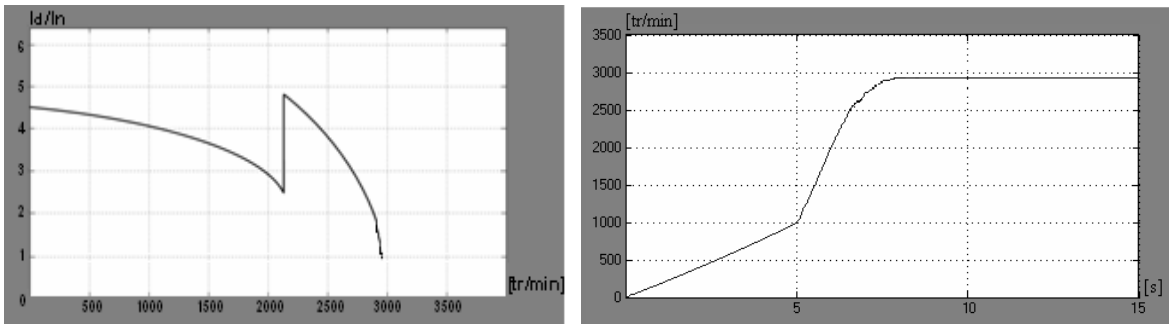


3.9.a - Sans rhéostat 3.9.b - Avec rhéostat

Figure 3.9 - Allure des courants du régime transitoire et établi

La tension appliquée aux bornes du moteur varie pendant le démarrage, le courant diminue dans les 5 premières secondes au fur et à mesure que le moteur accélère jusqu'à atteindre 1000 tr/min et lors de la fermeture de KM2 on voie que le retour à l'état de démarrage sans rhéostat avec un certain régime transitoire dû au phénomène de court circuit causé par la fermeture des contacteurs.

La courbe de la figure 3.10 représente la relation entre le courant de démarrage et le courant nominal, avec et sans rhéostat.



3.10.a - Relation entre le courant de démarrage et le courant nominal

3.10.b - Evolution de la vitesse du moteur

Figure 3.10 - Evolution du courant et de la vitesse

On voie nettement que le démarrage se fait à 4,5 I_n soit une réduction de 30% de la contrainte.

Les résultats de simulation confirme le bon fonctionnement de l'exemple cité au début du chapitre 2, cela indique qu'une commande assistée par ordinateur est fiable et peut être élargie vers un système plus complexe comme celui que l'on va voir dans la section suivante.

3.4 - Simulation de l'armoire commandée par PC:

Vue que l'on a le modèle de chacun des éléments du système, il ne restera plus qu'à créer des boites noires et de les relier suivant le schéma synoptique de la figure 2.16 (chapitre 2). Le reste des composants dans le schéma comme par exemple l'ULN 2803, sera reproduit sous forme de groupement parallèle de gain d'une valeur de 50 et les signaux de commande seront générés par des sources préprogrammées variant entre 0 et 5 volts induisant ainsi l'excitation des bobines de déclenchements internes des relais de commande.

L'impression de l'historique de fonctionnement ne sera pas modélisée car ça n'entre pas dans la commande des deux pompes et ça représente une partie optionnelle mais qui n'en demeure pas un moyen pratique pour suivre le fonctionnement du système.

La figure 3.11 représente le modèle de notre système sous Matlab, on préfère faire une simulation seulement pour la pompe 1 car la pompe 2 n'est qu'une pompe auxiliaire dans le système et une seule section suffira.

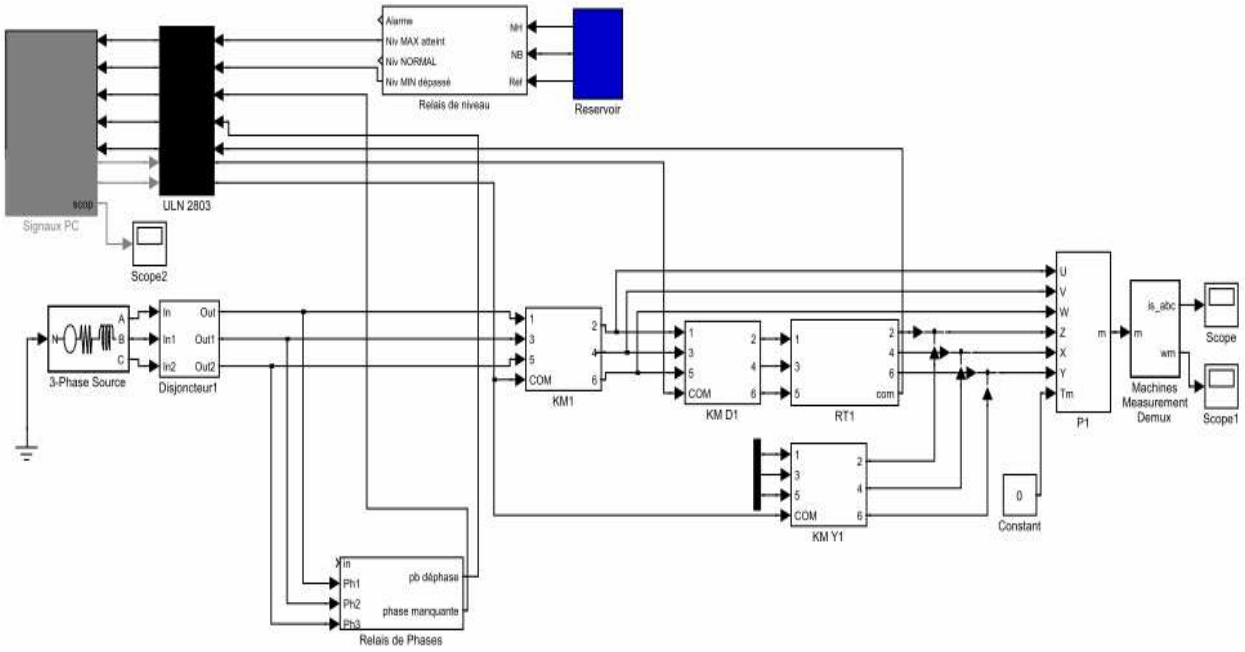


Figure 3.11 - Simulation du système sous Matlab/Simulink

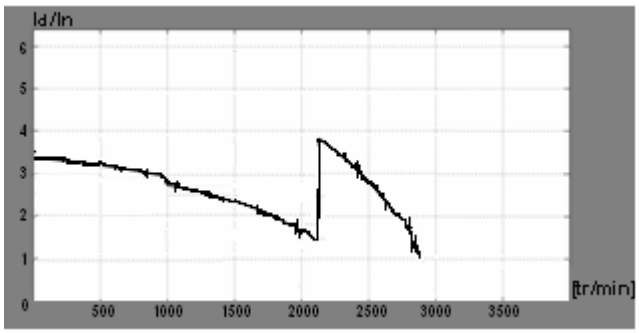
Le moteur est alimenté en U V W et bouclé en Z X Y étoile (voir figure 3.11). Quand les contacts temporisés de KM1 basculent, KM D1 décolle, et KM Y1 est excité. Le moteur fonctionne donc en triangle et le relais thermique de protection est alors en circuit. Le relais thermique doit être calculé pour [I_n /1,732].

Ce démarrage se réalise en deux temps :

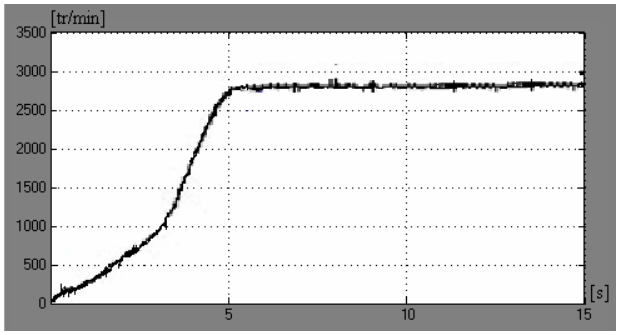
- **1er temps :** les enroulements vont être couplés en étoile, ce qui impose une tension réduite aux bornes de chaque enroulement (tension simple V = U/1,732). Le courant absorbé sera donc lui aussi réduit.
- **2ème temps :** au bout de quelques secondes, on couple les enroulements du moteur en triangle, chaque enroulement est maintenant soumis à la tension réseau entre phases U.

Le moteur fini son démarrage jusqu'à son point nominal.

Comme pour l'exemple précédent, on va étudier l'évolution du courant ainsi que de la vitesse pendant le régime de démarrage et le régime établie.



3.12.a - Relation entre le courant de démarrage et le courant nominal



3.12.b - Evolution de la vitesse du moteur

Figure 3.12 - Evolution du courant et de la vitesse

En comparaison avec les résultats de l'exemple précédent, on remarque trois choses :

- On voit bien que le courant de démarrage a été nettement réduit (voir figures 3.13, 3.14).
- Des ondulations sont présentes au niveau de la vitesse, cela est due au modèle mathématique qu'on a construit sous Matlab mais ça n'influe nullement sur le principe de fonctionnement.
- Le temps de démarrage est plus rapide vue qu'un démarrage Y-Δ n'a pas besoin de rhéostat.

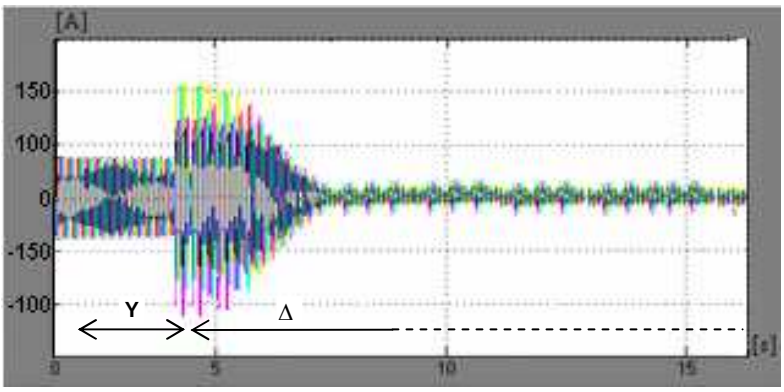


Figure 3.13 - Allure des courants pour le démarrage Y-Δ

$I_{Y\ max} = 80A$	$I_{Y\ eff} = 56A$	$t_{d\ Y} = 3s$
$I_{\Delta\ d\ max} = 120A$	$I_{\Delta\ d\ eff} = 84A$	$t_{d\ \Delta} = 3s$
$I_{\Delta\ n} = 30A$	$I_d/I_n \approx 2,6$	$t_d = t_{d\ Y} + t_{d\ \Delta} = 6s$ à 2850 tr/min

La figure 3.14 représente les signaux de commande injectés au relais. Pour donner plus de clarté dans la qualité de la figure, on a préféré les représenter sous forme de schémas au lieu de donner la visualisation sous Matlab. Le chronogramme de fonctionnement pour la pompe 1 sera comme suit :

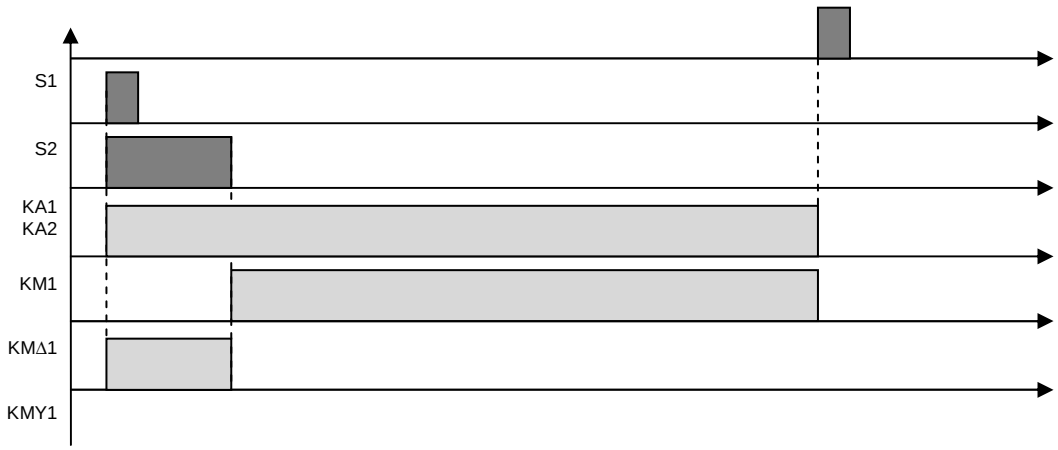


Figure 3.14 - Chronogramme du démarrage étoile triangle pour la Pompe 1

Ces créneaux de commande ne sont qu'une infime partie de celle généré par notre algorithme, car la plus grande partie des signaux est localisée dans la partie acquisition des données du fonctionnement (Data in) et surtout pour le suivie en temps réel du dispositif.

3.5 - Conclusion :

Les résultats de simulation, quoique très utiles et souvent indispensables, habituellement ne permettent pas à eux seuls de bien comprendre les phénomènes qui caractérisent le comportement du système. De ce fait, les modèles de boites noirs que l'on a créé au début peuvent être perfectionnés et être généralisés pour tous les types d'applications dans la simulation de schémas électrotechniques sous Matlab.

En fait, on pourra faire encore mieux en élaborant des modèles pour le reste des actionneurs existants sur le marché et qui n'ont pas été utilisés dans ce mémoire, comme par exemple : sectionneur, interrupteur fin de course, temporisateurs, électrovannes, capteurs, etc …

Conclusion générale et perspectives

Partant d'une étude générale des stations de pompage telle que la description de ces équipements et leur principe de fonctionnement, on a pu montrer l'importance de l'optimisation de fonctionnement des ouvrages d'extraction d'eau. Ceci a été fait par une étude complète comportant des courbes de modulation sur la consommation d'eau en fonction de temps.

Les résultats ont été satisfaisants et ont montré un bonne mise en condition du procédé de démarrage. En effet, l'introduction d'un système informatisé à l'intérieur d'un dispositif de commande industrielle n'a pas eu d'influence néfaste sur son fonctionnement, au contraire, avec ce procédé on a fait l'acquisition en ligne de l'installation en vue de suivre son fonctionnement et transmettre le maximum d'informations aux techniciens à partir de l'armoire de commande.

Comme tout prototype, les erreurs ne sont pas exclues. Les deux montages que l'on a réalisé ne sont pas parfaits à 100% et on ne pourra les vérifier qu'en faisant des tests à long termes sous différentes contraintes induisant ainsi tout les cas qu'on pourra trouver dans une situation réelle sur site.

Du point de vue consentions, on pourra utiliser des processeurs spécialisés pour l'acquisition comme les DSP, les FPGA/EPLD et les cartes PCI DSAPCE, mais qui ne pourront nullement concurrencer les processeurs INTEL ou AMD dans leur puissance et leur vitesse. La difficulté étant de pouvoir modifier les cartes mères standards pour en faire des stations d'études autonomes.

Du point de vue programmation, on pourra utiliser le langage C++, semblable qu'au C mais avec plus d'aisance dans sa logique d'exécution. Mais le plus intéressant, c'est que grâce à ce langage, on aura la possibilité de créer des logiciels de simulation ou d'acquisition sous Windows et de tester la fiabilité d'une armoire de commande avant de réaliser son câblage, pour prévenir les courts-circuits ou les défauts de liaison, ce qui arrive souvent. En fait, ce logiciel aura une base de données des éléments utilisés dans les dispositifs industriels et il ne suffira plus qu'à choisir les blocs nécessaires et suivre les consignes du cahier des charges.

Références bibliographiques

[1] - T. Wildi, Gilbert Sybille, « Électrotechnique - 4ème édition » [Commande Industrielle des Moteurs], Série de Boeck, 2005

[2] - P. Graftieaux, « Schémas en Électrotechnique », Série AGS, 1999

[3] - J. B. Delambre, « Principales Règles de Schéma Électrique et de Câblage », Support de Cours, Université d'Amiens, 1999

[4] - B. Ben Ghanem, « Démarrage et Freinage des Moteurs Asynchrones Triphasés», Support de Cours, École d'Ingénieurs de Monastir, 2008

[5] - F. Fabrice, « Démarrage des Moteurs Asynchrones », Dossier Ressource Génie Électrique, 2001

[6] - Positron-libre.com, « Le Démarrage Étoile Triangle », 1999

[7] - J. Lebanere, « Cours Freinage des Moteurs », Maxicours RCS, Paris, 2008

[8] - www.techno-science.net/Machine-asynchrone/Encyclopdie-scientifique-en-ligne.com

[9] - P. Abati, « Freinage par Injection de Courant Continu », Support de Cours, Génie électrique Aix-Marseille, 2008

[10] - O, Lejeune, « Schéma d'Électricité Industrielle et d'Électrotechnique », Olivier Lejeune (2005)

[11] - http://www.typonrelais.com, http://www.micrelec.fr

[12] - R. Bourgeois, D Cogniel, « Mémotech Plus Électrotechnique 7ème édition», Edition Casteilla, Paris, 2007

[13] - P. Bigras, « Gpa-781 Commande Par Ordinateur », Notes De Cours, École de Technologie Supérieure de Québec, 2002

[14] - Fiche Télémécanique, « Plate-forme d'automatisme Modicon Premium », France, 2005

[15] - Elec_Intro, « industrial modbus », Revue d'électronique, N°224, Chine, 2009

[16] - D. Mateo, « Apprenez à Programmer en C », www.siteduzero.com, 2005

[17] - H. Perla, « Parallel Port Programming with C », Revue Electrosoft, N°56, 2005

[18] - K. Sinna, R. Goularchi, « Les Stations de Pompage d'eau », Mémoire d'Ingéniorat, Département de Génie Electrique, Tunisie, 2006

[19] - A. Zobeiri, «Investigations of Time Dependent Flow Phenomena in a Turbine and a Pump-Turbine of Francis Type: Rotor-Stator Interactions and Precessing Vortex Rope», Thèse de Doctorat, École Polytechnique Fédérale de Lausanne, France, 2009

[20] - M. Hubin, «Paramètres des circuits logiques», Support de Cour, CNRS, France, 2010 http://pagespersoorange.fr/michel.hubin/physique/logique/chap_log9.htm

[21] - Datasheet de l'ULN 2803A

[22] - M. Gook, «PC Hardware Interfaces - 1ère édition », Edition Alist, UK, 2004.

[23] - T. Gantar, «Hydraulic Behaviour of Unconventional Radial Pump Stages», Université de Ljubljana, Faculté d'Ingénieur de Mécanique, Slovenia, 2008

[24] - G. Patri, « Des outils classiques à la commande numérique », Revue aérospatiale, N° hors série, France, 1990

[25] - P. Marcotte, « Étude sur la modélisation mathématique informatisée de la commande d'un robot afin d'améliorer sa précision d'opération », Thèse de Doctorat, Université Laval, 1994

[26] - P.M. Grojean, « Conception Assistée par Ordinateur de Commande en Temps Réel d'Automatismes Complexes », 3ème édition, France, 1999

[27] - C. Cyrot, « Conception assistée par ordinateur de systèmes de commande robuste temps réel: applications aux procédés mécaniques flexibles »,

[28] - L. Soete, H. Coeckelberghs, J. Pinte, « Les Systèmes de Programmation Assistée par Ordinateur des Machines-Outils à Commande Numérique: Une Synthèse de Leurs Caractéristiques », Centre de Recherche du SIFM, Bruxelles, 2002

[29] - D. Meyer, « Péri-PC 2, Plus de 20 Circuits d'Interface Spécifiques pour RS232, USB et le Port Parallèle », Edition GEODIF, France, 2001

[30] - Revues Elektor, N°66, « Les périphériques d'acquisitions PC », Février 1998

N°124, « Programmation et gestions des entrées/sorties », Mars 2003

N°130, « La commande et le langage de programmation», Septembre 2003

[31] - Y. Magda, « Complete Practical Measurement System using a PC », Edition GEODIF, UK, 2006

Annexe

Schéma électrotechnique de l’armoire de commande de notre dispositif, il a été réalisé à laide du logiciel XRelais 3.1 :

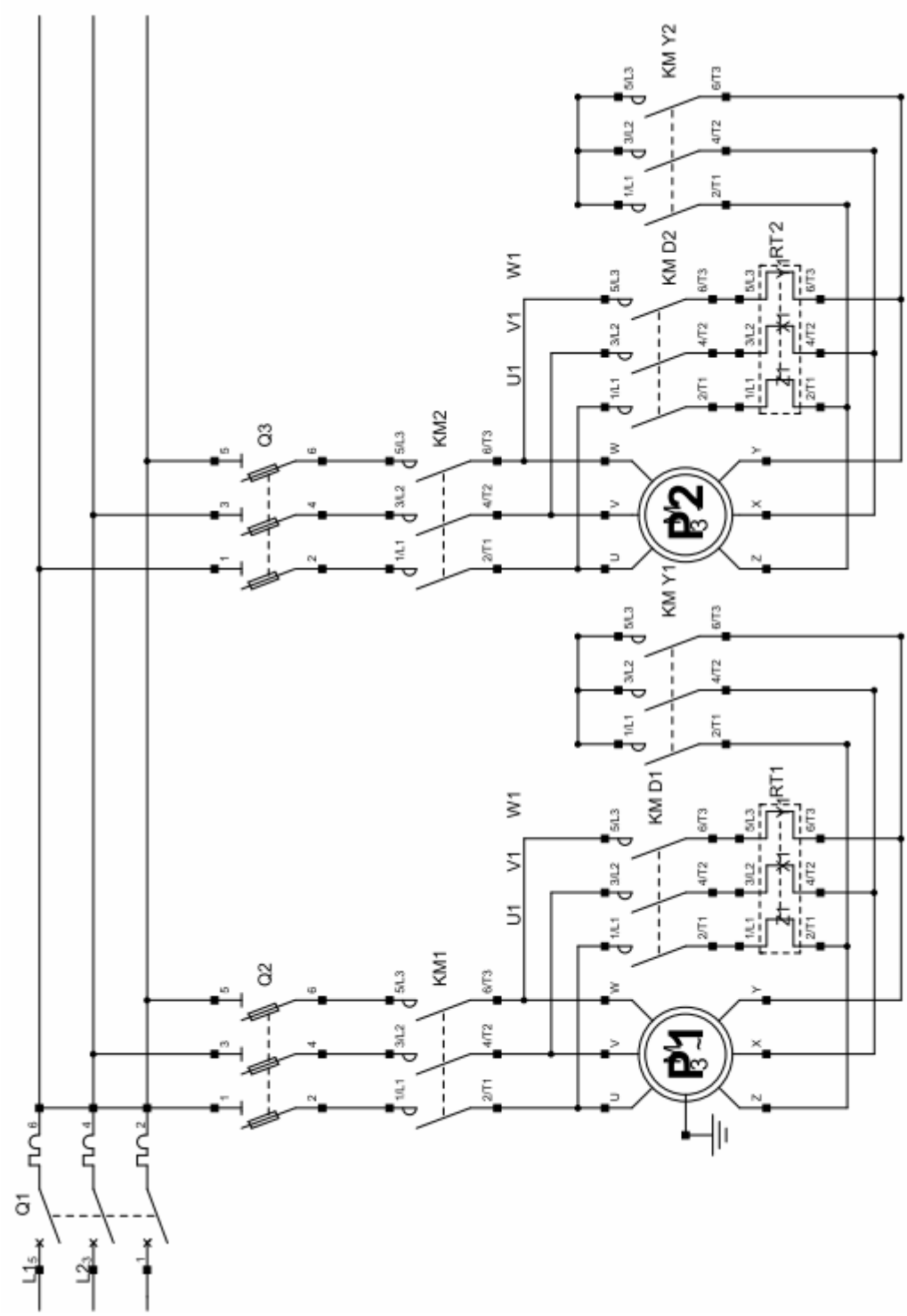


Fig. A1- Schéma de puissance de l’armoire de commande

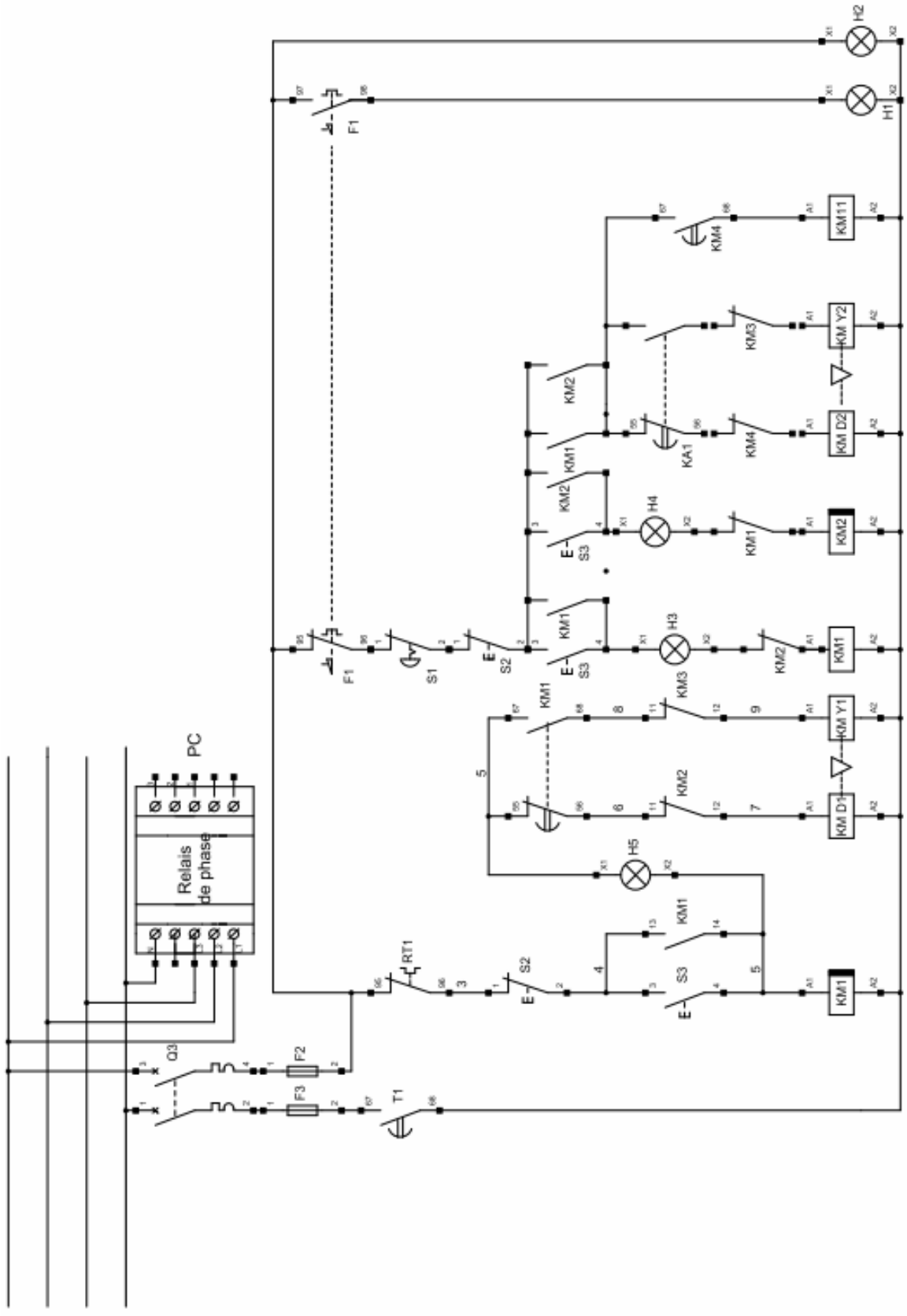


Fig. A2- Schéma de puissance de l’armoire de commande

Photos prises lors de la conception de l'armoire, il faut dire qu'elle n'est fini qu'à 70% et le câblage de la partie commande sera différent de celui dans les photos.

Fig. A3.a - Façade de l'armoire

Fig. A3.b - Face arrière de la façade

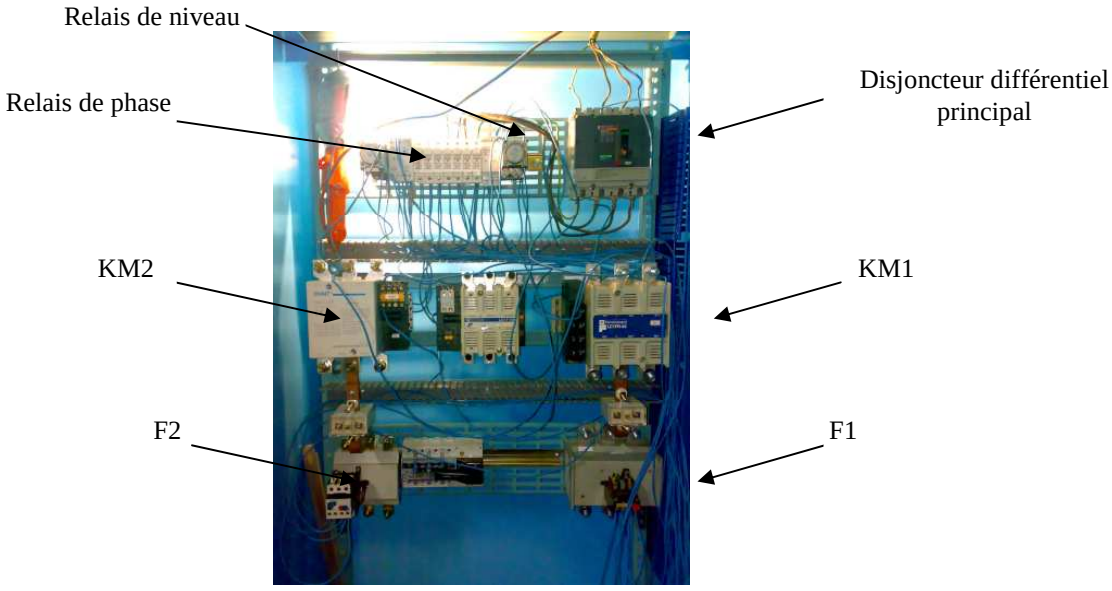


Fig. A3.c - Intérieur de l'armoire

Fig. A3- Photo de l'armoire finie à 70%

Une vidéo est disponible sur ce lien : http://www.youtube.com/watch?v=TE0miZRazbE

L’amplification est utilisée avec un ULN 2803 se fera avec cette configuration avec des protections en amont et aval du circuit :

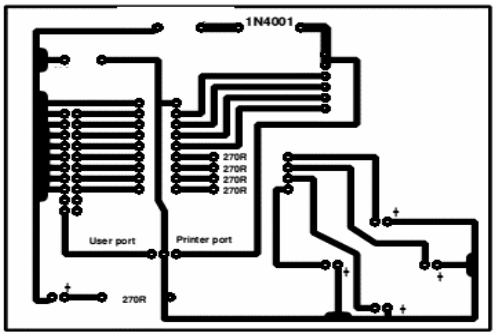


Fig. A4- PCB du la plaque d’amplification

Pour des raisons d’expérimentation il faut toujours réaliser les montages électroniques sous une plaque de test ou avec, dans notre cas, une plaque à pastilles de groupement de 3 comme est représenté sur la Fig. A4.

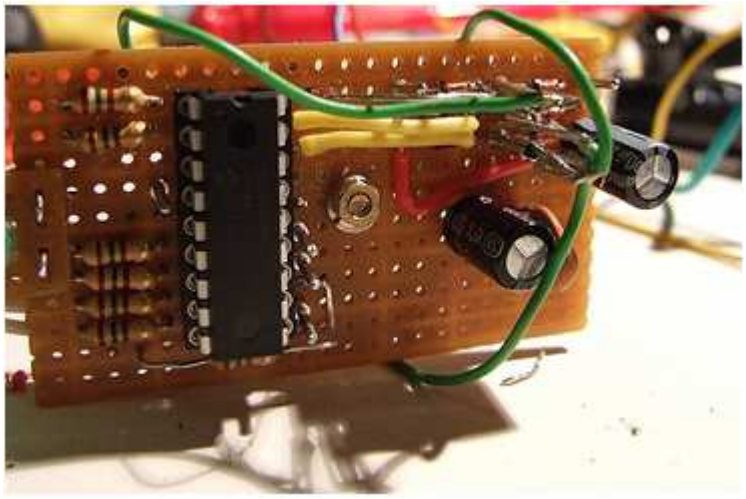


Fig. A5- Photo du circuit d’amplification à base de l’ULN 2803A

Le programme qui suit est écrit en C et il représente la quasi-totalité de l'algorithme, la plus grande difficulté été dans tout ce qui est acquisition et visualisation. La partie qui traite la communication vers l'armoire de commande étant la plus facile.

Code : C

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "test.h"
#include <SDL/SDL.h>
#include <FMOD/fmod.h>

#define PORT 0x378  /*Affectation de l'adresse du port LPT*/
#define TDS 0x0BB8  /*temporisation de 3 sec pour le démarrage étoile ou triangle*/

void main()
 {
   outportb(PORT,0x01);   /*sortie du code %00000001*/
   delay(TDS);            /*temporisation de 3 sec*/
   outportb(PORT,0x03);
   printf("Fin du démarrage, le moteur tourne à plain régime");
    while(!kbhit())
   {
     outportb(PORT,0x00); /*sortie du code %00000011*/
   }
   return 0;
 }
pixel* BMP(char* nomDuFichier){
unsigned char chaine1[54] = "";/*on met à l'intérieur l'entête du fichier qui fait
54*/ octets
unsigned char chaine[LONGEURX*LARGEURY*3];/*chaque pixel étant sur 3 octet*/
pixel Im[LONGEURX][LARGEURY];/*je déclare une matrice de la taille longueur largeur*/
FILE* fichier = NULL;
int i=0;//les variables d'incrémentation
int j=0;
fichier = fopen(nomDuFichier, "r");/*ouverture du fichier*/
if (fichier != NULL)
{
fgets(chaine1,54,fichier);/*la chaine ne sert qu'a incrémenter le pointeur pour qu'il
se positionne juste avant les informations concernants les pixels de l'image*/
fgets(chaine,3*LONGEURX*LARGEURY,fichier);//je met les pixels dans une chaine


for(j=0;j<LARGEURY;j++){
for(i=0;i<LONGEURX;i++){
/*le format bmp commence par le pixel en bas à gauche donc j 'ai fait un petit chgt de
variable pour commencer par en haut à gauche.
chque 3 octets forment un pixel*/
Im[i][LARGEURY-j-1].R=chaine[3*i+j*LONGEURX];
Im[i][LARGEURY-j-1].G=chaine[3*i+1+j*LONGEURX];
Im[i][LARGEURY-j-1].B=chaine[3*i+2+j*LONGEURX];

}}
return Im;/*on renvoie en sortie un pointeur sur le tableau*/
}
else
{

printf("Impossible d'ouvrir le fichier test.txt");
}

//PS: j ai choisi fgets comme fct parce qu'elle renvoie des char ( 1 octet ) en sortie
       }
       void main(){
       pixel im[LONGEURX][LARGEURY];
```

```
       im=BMP("image.bmp");
       int i=0;
       int j =0;
       for(i=0;i<LONGEURX-1;i++){
       for(j=0;j<LARGEURY-1;j++){
       printf("rouge=%d\t",im[i][j].R);
       printf("vert=%d\t",im[i][j].G);
       printf("bleu=%d\t\n",im[i][j].B);
       }

       Printf(Appuyer sur une touche pour arrêter le moteur");
Printf (nbr dém P1 : "%d\n")
Printf (temps fct P1/jours: "%d\n" h)
Printf (nbr dém P2 : "%d\n")
Printf (temps fct P2/jours: "%d\n" h)
Printf (Quantité moy déb/jour : "%d\n" m^3)
if (i = 0 ; i < 4 ; i+
       {
       Printf (phase: ok déphase: ok R therm 1: ok R therm 2: ok)
else
       Printf (phase : pointeur1 déphase : pointeur2 R therm 1: pointeur3 R therm 2:
       pointeur5)
       }
Printf (default : "%d\n")

avec test.h :

typedef struct pixel pixel;
#define LONGEURX 500
#define LARGEURY 1500
struct pixel


begin_message_map (cparallelportdlg, cdialog)
     //{{afx_msg_map (cparallelportdlg)
     on_wm_syscommand()
     on_wm_paint()
     on_wm_querydragicon()
     on_wm_timer()
     //}}afx_msg_map
     //code added by me from here.
     on_command_range(idc_pin2, idc_pin9, changepin)
     on_command (idc_pin14, changecontrol)
     on_command (idc_pin16, changecontrol)
     on_command (idc_pin17, changecontrol)
     on_command (idc_pin1, changecontrol)
     //code added by me till here
end_message_map()


#define LARGEUR_FENETRE         512 /* DOIT rester à 512 impérativement car il y a 512
barres (correspondant aux 512 floats) */
#define HAUTEUR_FENETRE         400 /* Vous pouvez la faire varier celle-là par
contre*/
#define RATIO                   (HAUTEUR_FENETRE / 255.0)
#define DELAI_RAFRAICHISSEMENT  25 /* Temps en ms entre chaque mise à jour du graphe.
25 ms est la valeur minimale. */

void setPixel(SDL_Surface *surface, int x, int y, Uint32 pixel);


int main(int argc, char *argv[])
{
    SDL_Surface *ecran = NULL;
    SDL_Event event;
    int continuer = 1, hauteurBarre = 0, tempsActuel = 0, tempsPrecedent = 0, i = 0, j
= 0;
    float *spectre = NULL;

    /* Initialisation de FMOD
       ----------------------
```

```
    On charge FMOD, la musique, on active le module DSP et on lance la lecture
    de la musique */

    FSOUND_Init(44100, 4, 0);
    FSOUND_STREAM* musique = FSOUND_Stream_Open("Hype_Home.mp3", 0, 0, 0);
    if (musique == NULL)
    {
        fprintf(stderr, "Impossible d'ouvrir la musique");
        exit(EXIT_FAILURE);
    }
    FSOUND_DSP_SetActive(FSOUND_DSP_GetFFTUnit(), 1);
    FSOUND_Stream_Play(FSOUND_FREE, musique);




  /* Initialisation de la SDL
     ------------------------

    On charge la SDL, on ouvre la fenêtre et on écrit dans sa barre de titre.
    On récupère au passage un pointeur vers la surface ecran, qui sera la seule
    surface utilisée dans ce programme */

    SDL_Init(SDL_INIT_VIDEO);
    ecran = SDL_SetVideoMode(LARGEUR_FENETRE, HAUTEUR_FENETRE, 32, SDL_SWSURFACE |
SDL_DOUBLEBUF);
    SDL_WM_SetCaption("Visualisation spectrale du son", NULL);

      /* Initialisation du tableau */

int main(int argc, char *argv[])
{
    int tableau[4], i = 0;
    for (i = 0 ; i < 7 ; i++)
    {
        tableau[i] = 0;
    } /* Affichage de ses valeurs pour vérifier*/
    for (i = 0 ; i < 4 ; i++)
    {
        printf("%d\n", tableau[i]);
    }

    return 0;
}


      /* Boucle principale */

    while (continuer)
    {
        SDL_PollEvent(&event); /* On doit utiliser PollEvent car il ne faut pas
attendre d'évènement
                                  de l'utilisateur pour mettre à jour la fenêtre*/
        switch(event.type)
        {
            case SDL_QUIT:
                continuer = 0;
                break;
        }

        /* On efface l'écran à chaque fois avant de dessiner le graphe (fond noir */
        SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 0, 0, 0));


        /* Gestion du temps
           -----------------
        On compare le temps actuel par rapport au temps précédent (dernier passage
dans la boucle)
        Si ça fait moins de 25 ms (DELAI_RAFRAICHISSEMENT), alors on attend le
        temps qu'il faut pour qu'au moins 25 ms se soit écoulées. On met ensuite à
```

```
        jour tempsPrecedent avec le nouveau temps */

        tempsActuel = SDL_GetTicks();
        if (tempsActuel - tempsPrecedent < DELAI_RAFRAICHISSEMENT)
        {
            SDL_Delay(DELAI_RAFRAICHISSEMENT - (tempsActuel - tempsPrecedent));
        }
        tempsPrecedent = SDL_GetTicks();



        /* Dessin du phraphique
           ------------------------

        C'est la partie la plus intéressante. Il faut réfléchir un peu à la façon
        de dessiner pour y arriver, mais c'est tout à fait faisable

        On récupère le pointeur vers le tableau de 512 floats via
FGRAPH_DSP_GetSpectrum()
        On travaille ensuite pixel par pixel sur la surface ecran pour dessiner les
barres.
        On fait une première boucle pour parcourir la fenêtre en largeur.
        La seconde boucle parcourt la fenêtre en hauteur pour dessiner chaque barre.
        */

        spectre = FGRAPH_DSP_GetSpectrum(); /* On récupère le pointeur vers le tableau
de       512 floats */

        SDL_LockSurface(ecran); /* On bloque la surface ecran car on va directement
        modifier ses pixels */


         /* BOUCLE 1 : on parcourt la fenêtre en largeur (pour chaque barre verticale)
*/
        for (i = 0 ; i < LARGEUR_FENETRE ; i++)
        {
        /* On calcule la hauteur de la barre verticale qu'on va dessiner.
        spectre[i] nous renvoie un nombre entre 0 et 1 qu'on multiplie par 4 pour
        zoomer
        afin de voir un peu mieux (comme je vous avais dit). On multiplie ensuite
        par HAUTEUR_FENETRE pour que la barre soit agrandie par rapport à la taille
        de la fenêtre */
        hauteurBarre = spectre[i] * 4 * HAUTEUR_FENETRE;

        /* On vérifie que la barre ne dépasse pas la hauteur de la fenêtre
        Si tel est le cas on coupe la barre au niveau de la hauteur de la fenêtre */
        if (hauteurBarre > HAUTEUR_FENETRE)
        hauteurBarre = HAUTEUR_FENETRE;

        /* BOUCLE 2 : on parcourt en hauteur la barre verticale pour la dessiner */
        for (j = HAUTEUR_FENETRE - hauteurBarre ; j < HAUTEUR_FENETRE ; j++)
        {
         /* On dessine chaque pixel de la barre à la bonne hauteur.
         On fait simplement varier chacun dans un sens différent

j ne varie pas entre 0 et 255 mais entre 0 et HAUTEUR_FENETRE. Si on veut l'adapter
proportionnellement à la hauteur de la fenêtre, il suffit de faire le calcul
j / RATIO, où RATIO vaut (HAUTEUR_FENETRE / 255.0)
                J'ai dû réfléchir 2-3 minutes pour trouver le bon calcul à faire, mais
c'est du niveau de tout le monde. Il suffit de réfléchir un tout petit peu */

                setPixel(ecran, i, j, SDL_MapRGB(ecran->format, 255 - (j / RATIO), j /
RATIO, 0));
            }
        }

        SDL_UnlockSurface(ecran); /* On a fini de travailler sur l'écran, on débloque
la surface */

        SDL_Flip(ecran);
    }
```

```
    /* Le programme se termine.
    On désactive le module DSP, on libère la musique de la mémoire
    et on ferme FMOD et SDL */
    FSOUND_DSP_SetActive(FSOUND_DSP_GetFFTUnit(), 0);
    FSOUND_Stream_Close(musique);
    FSOUND_Close();

    SDL_Quit();

    return EXIT_SUCCESS;
}


/* La fonction setPixel permet de dessiner pixel par pixel dans une surface */
void setPixel(SDL_Surface *surface, int x, int y, Uint32 pixel)
{
    int bpp = surface->format->BytesPerPixel;

    Uint8 *p = (Uint8 *)surface->pixels + y * surface->pitch + x * bpp;

    switch(bpp) {
    case 1:
        *p = pixel;
        break;

    case 2:
        *(Uint16 *)p = pixel;
        break;

    case 3:
        if(SDL_BYTEORDER == SDL_BIG_ENDIAN) {
            p[0] = (pixel >> 16) & 0xff;
            p[1] = (pixel >> 8) & 0xff;
            p[2] = pixel & 0xff;
        } else {
            p[0] = pixel & 0xff;
            p[1] = (pixel >> 8) & 0xff;
            p[2] = (pixel >> 16) & 0xff;
        }
        break;
    case 4:
        *(Uint32 *)p = pixel;
        break;
    }
       return 0;
 }
```

Résumé *- Ce mémoire traite les étapes de conception d'un système de commande informatisé à base d'un ordinateur standard en utilisant son port parallèle (LPT) destiné généralement à la gestion des imprimantes. Le but est donc de créer des signaux de commande numérique capables de piloter n'importe quel actionneur électromagnétique, en l'occurrence dans notre cas, deux pompes hydrauliques. L'algorithme de commande sera écrit en langage C sous Windows XP et l'interface de commande sera complétée par des contacteurs et relais qui constitueront respectivement la partie puissance et la partie commande.*
Mots clés : *Commande par PC, pompe hydraulique, station de pompage, langage C*

Abstract *- This work deals the conception stage of computerized control system based on a basic computer using parallel port (LPT) which it is generally used to printer management. The aim is to create numerical control signals able to drive any electro-magnetically actuator by occurrence in our case two hydraulic pomp. The control algorithm will be write in C language under Windows XP, and interfacing control will be completed with the contactors and relays which made respectively the power and command part.*
Key words : *PC control, hydraulic pump, Purification station, C language*

ملخص : هذه الأطروحة تتناول مراحل تصميم نظام السيطرة على جهاز الكمبيوتر على أساس معيار الكمبيوتر باستخدام منفذ متوازي *(LPT)* المقصود عموما لإدارة الطابعات والهدف هو خلق إشارات التحكم الرقمية يمكن أن تحكم أي صمام الكهرومغناطيسي وهي في حالتنا ، و مضختين مائيتين . يتم كتابة الخوارزمية في اللغة C تحت ويندوز XP و واجهة التحكم .سيتم الانتهاء من الملامسات والقاطعات على التوالي والتي تشكل جزء القدرة وجزء التحكم.
المفاتيح : التحكم بالحاسوب ، المضخة المائية ، محطة التصفية ، اللغة C