



الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

جامعة وهران 2 محمد بن أحمد
Université d'Oran 2 Mohamed Ben Ahmed

معهد الصيانة والأمن الصناعي
Institut de Maintenance et de Sécurité Industrielle

Département de Maintenance en Instrumentation

MÉMOIRE

Pour l'obtention du diplôme de Master

Filière : Génie Industriel
Spécialité : Génie Industriel

Thème

**Réalisation d'un Automate Programmable à Base d'un
Microcontrôleur ARDUINO**

Présenté et soutenu publiquement par : NOGRARA LAHOUARI

Devant le jury composé de :

Nom et Prénom	Etablissement	Qualité
Mme.HAIMOUR RACHIDA	IMSI-Univ. D'Oran2	Présidente
Mr.ADDA NEGGAZ SAMIR	IMSI-Univ. D'Oran2	Encadreur
Mr.NEKROUF DJILALI	IMSI-Univ. D'Oran2	Examineur

Année 2020/2021



Remerciement

Avant tout, je remercie le bon Dieu le tout puissant de m'avoir donné le courage, la volonté et la patience durant toutes les années d'études et que grâce à lui ce travail a pu être réalisé.

Mes vifs remerciements vont en premier lieu, à mes chers parents, familles et camarades de m'avoir aidé, encouragés et soutenus tout au long de ces années et qui continuent de croire en moi en dépit de tout.

Comme je tiens à exprimer ma profonde gratitude à mon encadreur Mr. Adda Neggaz qui a cru en moi, ainsi que me soutenir et m'orienter tout au long de mon travail.

Qu'il trouve ici l'expression de ma reconnaissance.

Je remercie aussi les membres de jury qui m'ont fait l'honneur de juger ce Projet, j'espère être à la hauteur de leurs attentes.

Je suis très reconnaissant à tous les enseignants qui m'ont veillé au bon déroulement de mes formations tout au long de mes cursus. Qu'ils trouvent ici l'expression de mon respect et remerciements les plus sincères.

Un grand merci aussi à toute personne qui de près ou de loin a contribué à ce que ce modeste travail.

Résumé

Nous avons réalisé un automate programmable industriel à base d'un microcontrôleur Arduino, cet automate que nous avons conçu nous a permis de bien comprendre les concepts de l'automatisation.

Dans ce mémoire nous avons cité quelques généralités sur les automates programmables industriels et on a fait une étude sur la carte Arduino, puis On a conçu et simulé des différents exemples à l'aide de Grafcet et ladder et

Logiciel protéus, ensuite on a passé aux tests sur les plaques d'essai et à la réalisation final de notre système d'ascenseur.

L'automate que nous avons réalisé à répondu globalement aux même exigences attendues d'un automate classique, notamment la possibilité d'être programmé et reprogrammé directement via un port parallèle d'un microordinateur.

Summary

We have made an industrial programmable controller based on an Arduino microcontroller, the controller that we have designed allowed us to fully understand the concepts of automation.

In this thesis we listed some generalities on industrial programmable logic controllers and we did a study on the Arduino board, then we designed and simulated different examples using Grafcet , ladder and proteus software, then we moved on to testing & the finalizing our elevator system.

The PLC that we have produced has generally met the same requirements expected of a conventional PLC, in particular the possibility of being programmed and reprogrammed directly via a parallel port of a microcomputer

ملخص

لقد صنعنا وحدة تحكم صناعية قابلة للبرمجة تعتمد على متحكم أرد وينو، وحدة التحكم التي صممناها سمحت لنا بفهم مفاهيم الآلية تمامًا.

في هذه الرسالة، ذكرنا بعض العموميات حول وحدات التحكم المنطقية القابلة للبرمجة الصناعية وقمنا بدراسة على لوحة ارد وينو، و ladder و Grafcet ثم قمنا بتصميم ومحاكاة أمثلة مختلفة باستخدام

، ثم انتقلنا إلى اختبار لوحات الاختبار ووضع اللمسات الأخيرة على نظام المصعد الخاص بنا Proteus

لقد استوفى نظام التحكم المنطقي القابل للبرمجة الذي قمنا بإنتاجه بشكل عام نفس المتطلبات المتوقعة من نظام التحكم المنطقي القابل للبرمجة التقليدي ، ولا سيما إمكانية البرمجة وإعادة البرمجة مباشرة عبر منفذ متوازي لجهاز كمبيوتر صغير

TABLE DES MATIÈRES

INTRODUCTION GENERALE

.....	1
I .. AUTOMATES PROGRAMMABLES	2
I.1 Historique	3
I.2 Introduction	4
I.3 Définition d'un automate.....	5
I.4 Les avantages et les inconvénients d'un API	5
I.5 Domaines d'emploi des automates.....	5
I.6 Fonctionnement.....	6
I.7 Les caractéristiques principales d'un automate programmable.....	7
I.8 Structure D'un Automate Programmable.....	7
I.8.1 Structure générale des API :.....	7
I.8.2 Structure interne des API.....	8
I.9 Différents types des API.....	16
I.9.1 Type compact	16
I.9.2 Type modulaire.....	17
I.10 Choix d'un automate industriel	18
I.10.1 Automate fixe (dure)	18
I.10.2 Automate programmable	19
I.10.3 Automate flexible (douce).....	19
I.10.4 Automate totalement intégrée (TIA)	19
I.11 La Programmation (Logiciel / Langage / Réseaux).....	19
I.11.1 Ladder Diagramme LD.....	20
I.11.2 Le grafcet.....	20
I.11.3 Langage IL	20
I.12 Conclusion.....	20
II . LOGICIEL GRAFCET ET LADDER	22
II.1 Introduction	22
II.2 Ladder.....	23
II.2.1 Instructions de base pour le Ladder.....	23
II.2.2 Fonction de Ladder.....	25
II.2.3 Diagramme de la logique LADDER	25
II.3 Grafcet.....	27
II.3.1 Historique De Grafcet	27

II.3.2	Introduction	27
II.3.3	Avantage De Grafcet	28
II.3.4	Les concepts de base du Grafcet	28
II.3.5	Les règles de Grafcet	30
II.3.6	Description de Grafcet.....	31
II.4	Conclusion.....	32
III	Arduino	34
III.1	Introduction	34
III.2	Définition d'un Arduino	34
III.3	Les gammes de la carte Arduino	35
III.3.1	La carte Arduino UNO	35
III.3.2	La carte Arduino Lenardo	35
III.3.3	La carte Arduino Méga.....	36
III.3.4	La carte Arduino Méga ADK.....	36
III.3.5	La carte Arduino Due	37
III.3.6	La carte Arduino Nano	37
III.3.7	La carte Arduino Mini Pro	37
III.3.8	La carte Arduino Yun.....	38
III.4	Eléments des cartes Arduino	38
III.4.1	Catégories Matériel	38
III.4.2	Catégories logiciel.....	40
III.5	Applications.....	40
III.6	Logiciel de programmation l'IDE d'Arduino.....	41
III.7	La structure d'un programme.....	42
III.8	Le langage d'Arduino.....	43
III.9	Présentation générale de l'Arduino Méga 2560	44
III.9.1	Caractéristiques de l'Arduino Méga 2560.....	44
III.9.2	Caractéristique technique de la carte Arduino Méga 2560.....	45
III.9.3	Synthèse des caractéristiques	45
III.10	Conclusion.....	48
IV	SIMULATION ET CONCEPTION.....	50
IV.1	Introduction	50
IV.2	Exemplaire	50
IV.2.1	Exemple 1 : Exemple Allumage des LED Avec la Divergence.....	50
IV.2.2	Exemple Allumage des LED Avec la Convergence.....	53
IV.2.3	Exemple Allumage des LED avec Le Timer.....	56

Est un relais utilisé pour des fonctions de temporisations et retards.	56
IV.3 Etude théorique :	57
IV.3.1 Partie Simulation	57
IV.3.2 Partie électronique.....	69
IV.4 Conclusion.....	80
Conclusion Générale	81
ANNEXE OU.....	83
ANNEXE ET.....	84
ANNEXE TIMMER.....	85
ANNEXE ASCENSEUR : VERSION 1.....	86
ANNEXE ASCENSEUR VERSION 2.....	88
ANNEXE	92

LISTE DES FIGURES

Figure I.1: Photo historique montrant de : Dick Morley, Tom Boissevain, George Schwenk et Jonas Landau	4
Figure I.2: Principal automates programmable.	4
Figure I.3 : cycle de la tâche maître	6
Figure I.4: Structure générale des API.	8
Figure I.5:Architecture Simplifier de l'intérieur d'un automate programmable.....	9
Figure I.6:Structure interne de CPU.....	10
Figure I.7: Module CPU d'un API.	10
Figure I.8 : Structure de l'UC d'un API.....	11
Figure I.9: Modules d'E/S.....	13
Figure I.10: Exemple de structure d'un module d'entrée logique.....	14
Figure I.11 : Structure d'une sortie d'un module logique	14
Figure I.12: Exemple de structure d'un module de sortie logique.	15
Figure I.13: Exemple de structure d'une sortie logique.	15
Figure I.14:Console de programmation.	16
Figure I.15: Automate compact (Allen-Bradley).	17
Figure I.16 : Automate modulaire (Modicon).	17
Figure I.17 : Automate modulaire (Siemens).	18
Figure II.1 : Interface Logiciel LDmicro	22
Figure II.2:Un tableau des opérands utilisés pour ces instructions.	24
Figure II.3: langage Ladder (a) Porte ET simplifiée. (b) Porte OU simplifiée	24
Figure II.4:Minuterie de mise en marche (On-Delay Timer : RTO).....	24
Figure II.5 : Table logique d'un LADER	25
Figure II.6:Diagramme d'un Ladder logique simplifié.....	26
Figure II.7:Représentation d'une étape du grafcet et l'étape initiale.....	29
Figure II.8: Représentation normalisée des actions associées aux étapes du grafcet.....	29
Figure II.9 : Représentation normalisée des Transition.	29
Figure II.10: Représentation des Liaisons orientées.	30
Figure II.11:Schéma regroupant les règles et le fonctionnement de Grafcet.	31
Figure III.1 : Description de la carte Arduino.	34
Figure III.2 : La carte Arduino UNO.	35
Figure III.3:carte Arduino Lenardo.	35
Figure III.4: Carte Arduino Méga.	36
Figure III.5:carte Arduino Méga ADK.	36
Figure III.6: Carte Arduino Due.....	37
Figure III.7: Carte Arduino Nano.....	37
Figure III.8 : carte Arduino Mini Pro.....	38
Figure III.9:carte Arduino Yun.	38
Figure III.10 : Représentation des éléments d'une carte Arduino.....	39

Figure III.11 : Présentation IDE Arduino.....	41
Figure III.12: La barre d'outils de l'IDE d'Arduino	41
Figure III.13: Structure d'un programme	42
Figure III.14: Description de la carte Arduino MEGA 2560.	44
Figure III.15: illustration d'un grafcet d'allumage des LED Avec la Divergence.....	51
Figure III.16 : Ladder d'allumage Des LED Avec la Divergence	52
Figure III.17 : Interface De Logiciel Arduino avec le Code	52
Figure III.18: Schéma Proteus 8 d'allumage Des LED Avec la Divergence	53
Figure III.19: illustration d'un grafcet d'allumage des LED Avec la Convergence	54
Figure III.20: Ladder d'allumage Des LED Avec la Convergence.....	55
Figure III.21 : Schéma Proteus 8 d'allumage Des LED Avec Convergence	55
Figure III.22: Illustration Grafcet d'allumage Des LED Avec un Timer.....	56
Figure III.23: Ladder d'allumage Des LED avec un Timer	56
Figure III.24: Schéma Proteus 8 d'allumage Des LED Avec Le Timer	57
Figure III.25: Organigramme général de fonctionnement de l'ascenseur.....	58
Figure III.26: Organigramme de détection de la position(A) et de contrôle des portes(B).	59
Figure III.28: Organigramme de mouvement de la cabine	60
Figure III.29 : illustration d'un grafcet d'appel de la cabine.	61
Figure III.30: illustration d'un grafcet demande de l'étage.	61
Figure III.31: Ladder appelle de l'ascenseur.....	64
Figure III.32 : Ladder demande de l'ascenseur	65
Figure III.33 : Simulation d'un automate programmable d'un ascenseur a 3 étages avec Proteus 8.....	68
Figure III.34 : composé d'ULN2803.....	69
Figure III.35 : Schéma interne d'un relai.	70
Figure III.36 : Bouton poussoir (Droite : schéma avec câblage, Gauche : Pièce Original)	70
Figure III.37: image d'un afficheur à 7 segments.	71
Figure III.38: Chiffrement d'un afficher 7 segments	71
Figure III.39: Brochage standard d'un afficheur 7 segments CC et CA.....	72
Figure III.40: Le CD4511.....	73
Figure III.41: carte Arduino Méga 2560.	73
Figure III.42: Liquid Crystal Display 16*2.....	74
Figure III.43: Interrupteur de position (Capteur fin de course mécanique)	75
Figure III.44: Structure interne d'un moteur a courant continue	76
Figure III.45: Schéma interne d'une carte d'alimentation	76
Figure III.46: Image d'un Terminal Virtual	76
Figure III.47: Organigramme de la carte de commande	77
Figure III.48: Corde D'un ascenseur	78
Figure III.49 automate programmable d'un ascenseur a base d'un microcontrôleur ARDUINO	79

Introduction Générale

Le monde de l'industrie a connu ces dernières années des bouleversements importants, les machines automatiques ont été développées pour libérer l'homme de ses tâches quotidiennes, et aussi pour le remplacer dans l'exécution des travaux nombreux.

Les progrès de l'électronique et de l'informatique, ont donné naissance aux automates programmables industriels (API) et d'autres microcontrôleurs qui peuvent s'adapter et s'intégrer dans les processus industriels. Ils peuvent accomplir des tâches plus complexes, non seulement de contrôle, mais aussi de traitement de données, de circulation d'informations et de simulation. Et parmi ces nombreux exemplaires des automates programmable, en trouve Les ascenseurs et les montes charges

Les ascenseurs ont toujours été essentiels dans les bâtiments et sont devenus particulièrement importants dans le cas des immeubles de grande hauteur. Le transport vertical fait référence au problème qui se pose lorsqu'un passager souhaite se déplacer en ascenseur entre les étages d'un immeuble, Dans ce projet on cherche à résoudre cette problématique en proposant une maquette d'ascenseur. Notre contrôleur est basé sur une carte Arduino Méga

Le présent mémoire est organisé en quatre chapitres, une conclusion et quelques perspectives.

Dans le premier chapitre, nous allons donner des généralités sur les Automates Programmables industriels.

Le deuxième chapitre sera consacré à l'étude de la carte Arduino Méga 2560 autour duquel est architecturé notre système.

Dans le troisième chapitre, nous allons présenter Le mode de représentation et d'analyse Grafset, Et Le Langage de programmation Ladder

Dans le dernier chapitre, nous allons présenter la phase conception et simulation ainsi que la partie réalisation, des différents modules de notre automate.

Nous clôturons notre mémoire par une conclusion générale et quelques perspectives.

CHAPITRE I
AUTOMATES
PROGRAMMABLES

I.1 Historique

Un Automate Programmable Ou PLC (programmable logic Controller) est une unité informatique durcie industriellement qui exécute des fonctions de contrôle discrètes ou continues dans une variété d'environnements d'usine de traitement et d'usine. Il était à l'origine conçu comme un équipement de remplacement de relais pour l'industrie automobile.

L'histoire initiale de l'API remonte aux années 1960, lorsque les systèmes de contrôle étaient encore gérés à l'aide de la commande par relais. Pendant ce temps, les salles de contrôle se composaient de plusieurs murs contenant de nombreux relais, borniers et masse de fils.

L'aspect actuel de nombreux processus industriels est le résultat de nombreuses années de recherche et de travail acharné de personnes engagées à améliorer leur fonctionnalité, leur gestion et leur organisation. On pourrait rappeler la phrase «*la nécessité est la mère d'invention* », et cela correspondrait certainement au travail quotidien des ingénieurs et techniciens de contrôle travaillant dans les processus industriels dans les années 50 et 60. Cette nécessité a été à l'origine de dispositifs tels que le PLC et le DCS (Distributed Control System). [1]

Aujourd'hui, les API conservent les mêmes fonctionnalités de base et la même simplicité qui les rendaient si populaires auprès des fabricants. Cependant, grâce aux progrès continus de la technologie des processeurs et de la mémoire, les API continuent de diminuer en taille tout en augmentant en puissance et en vitesse. Les API modernes s'intègrent également de manière transparente aux systèmes ERP et MES, ainsi qu'aux puissants logiciels de surveillance des machines et SCADA, offrant de nouvelles façons aux fabricants d'améliorer l'efficacité et les performances de leurs opérations grâce à l'analyse des données de la machine.

Avant l'industrie utilisait des relais électromagnétiques et de systèmes pneumatiques pour la réalisation des parties commandes qui apparait d'être cher, sans flexibilité, et qui offre aucune communication, donc ils ont adopté l'utilisation de systèmes à base de microprocesseurs permettant une modification aisée des systèmes automatisés.



Figure I.1: Photo historique montrant de : Dick Morley, Tom Boissevain, George Schwenk et Jonas Landau

I.2 Introduction

Au départ, les industries utilisaient des relais pour contrôler les processus de fabrication. Les panneaux de contrôle des relais devaient être régulièrement remplacés, consommaient beaucoup d'énergie et il était difficile de comprendre les problèmes qui y étaient associés. Pour trier ces problèmes, un automate programmable (PLC) a été introduit.

Le PLC est un ordinateur numérique utilisé pour l'automatisation de divers processus électromécaniques dans les industries. Ces contrôleurs sont spécialement conçus pour survivre dans des situations difficiles et à l'abri de la chaleur, du froid, de la poussière et de l'humidité, etc... Le PLC se compose d'un microprocesseur qui est programmé en utilisant le langage informatique [2].

Les automates programmables ont été inventés par *Dick Morley* en 1964. Depuis lors, ils n'ont pas cessé de révolutionner les secteurs industriels et manufacturiers. Il existe une large gamme de fonctions des API telles que la synchronisation, le comptage, le calcul, la comparaison et le traitement de divers signaux analogiques.

Un API a la capacité de contrôler des processus beaucoup plus volumineux et plus complexes. Il peut être personnalisé en fonction de l'application et des besoins de l'utilisateur. [3]

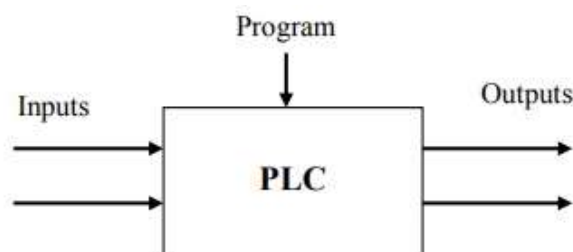


Figure I.2: Principal automates programmable.

I.3 Définition d'un automate

API (Automate Programmable Industriel) est un appareil électronique destiné à la commande de processus industriels par un traitement séquentiel. Qui comporte une mémoire programmable par un utilisateur automaticien à l'aide d'un langage adapté (Le langage List, Le langage Ladder...etc.) pour le stockage interne des instructions donnée afin de satisfaire un objectif donné. L'automate permet de contrôler, coordonner et d'agir sur l'actionneur, ce qui permet de simplifier la tâche d'automatisation des processus industriels [2].

I.4 Les avantages et les inconvénients d'un API

L'API s'est substituée aux armoires à relais en raison de sa souplesse, ils offrent de nombreux avantages par rapport aux dispositifs de commande câblés, comme :

- ✓ Accroître la productivité (rentabilité, compétitivité) du système ;
- ✓ Améliorer la flexibilité de production ;
- ✓ Améliorer la qualité du produit ;
- ✓ Réduire les couts de production ;
- ✓ Simplification du câblage ;
- ✓ Adaptation à des contextes particuliers tels que les environnements hostiles pour l'homme (milieu toxique, dangereux, nucléaire...);
- ✓ Améliorer les conditions de travail en éliminant les travaux répétitifs et adaptation à des tâches physiques pénibles pour l'homme (manipulation de lourdes charges ;
- ✓ Augmenter la sécurité ;
- ✓ Plus économique [4].

Ses inconvénients sont :

- ✓ Trop de travail requis dans les fils de connexion ;
- ✓ Le Cout élevé du matériel ;
- ✓ Le Besoin de formation pour maitriser bien le langage de programmation [4].

I.5 Domaines d'emploi des automates

On utilise les API dans tous les secteurs industriels pour la commande des machines (convoyage, emballage ...) ou des chaînes de production (automobile, agroalimentaire, ...) ou il peut également assurer des fonctions de régulation de processus (métallurgie, chimie ...). Il est de plus

en plus utilisé dans le domaine du bâtiment (tertiaire et industriel) pour le contrôle du chauffage, de l'éclairage, de la sécurité ou des alarmes [5].

I.6 Fonctionnement

Un API est un contrôleur basé sur un microcontrôleur avec plusieurs entrées et sorties. Il utilise une mémoire programmable pour stocker les instructions et exécuter des fonctions de contrôle des machines et des processus. L'automate exécute les fonctions logiques des relais, temporisateurs, compteurs et séquenceurs.

En effet avant d'exécuter quoi que ce soit, l'automate lit entièrement son programme ; et une fois l'exécution terminée recommence les mêmes opérations. On définit alors la notion de cycle et de temps de cycle.

Il existe plusieurs types de cycle, le plus répandu comprend 5 phases :

- **Phase 1** : Lecture ou Acquisition des entrées : Prise en compte des informations des modules d'entrées et écriture de leur valeur dans RAM (zone de DONNEE).
- **Phase 2** : Exécution du programme ou traitement des données : Lecture du programme (située dans la RAM programme) par l'unité de traitement, lecture des variables (RAM données).
- **Phase 3** : Traitement de toute demande de communication.
- **Phase 4** : Exécution du test d'auto-diagnostique (Gestion du système Autocontrôle de l'automate).
- **Phase 5** : Ecriture des sorties : Lecture des variables de sorties dans la RAM données et transfert vers le module de sorties.

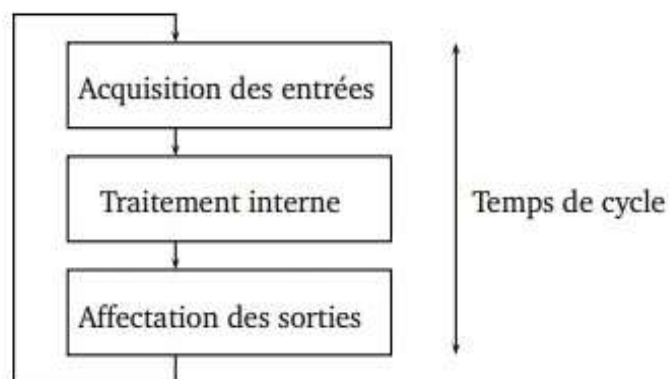


Figure I.3 : cycle de la tâche maître

Le temps de scrutation de chaque cycle est vérifié par un temporisateur appelé *Watchdog* (chien de garde) qui enclenche une procédure d'alarme en cas de dépassement de celui-ci (réglé par l'utilisateur) [6].

I.7 Les caractéristiques principales d'un automate programmable

- Compact ou modulaire.
- Tension d'alimentation.
- Taille de la mémoire.
- Sauvegarde (EPROM, EEPROM, pile, ...).
- Nombre d'entrées/sorties E/S.
- Modules complémentaires (analogique, communication).
- Langage de programmation [7].

I.8 Structure D'un Automate Programmable

I.8.1 Structure générale des API :

Un API comprend généralement des modules arrangés l'un à côté de l'autre, tels qu'une alimentation, une unité centrale (CPU) à base de microprocesseur dotée d'une carte de mémoire, des interfaces d'entrées et de sorties, des interfaces de communication, des cartes spéciaux et un dispositif de programmation. On peut effectivement considérer qu'il s'agit d'une unité contenant un grand nombre de relais, compteurs, temporisateurs et unités de stockage de données distincts (généralement EEPROM).

La figure suivante montre la disposition de base d'un API :

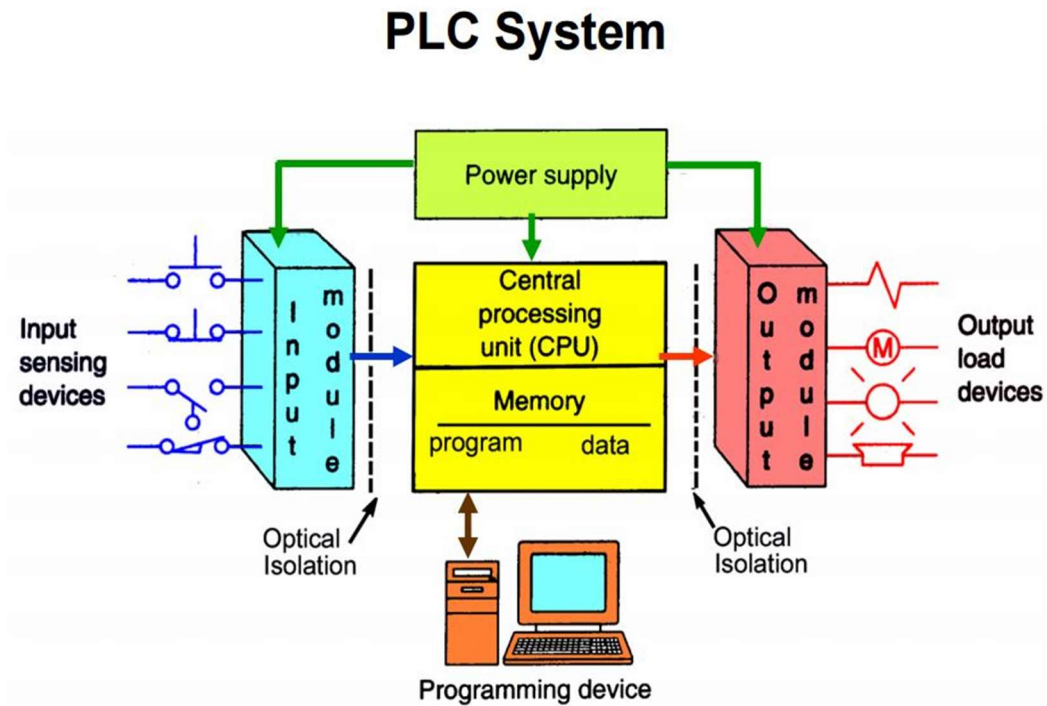


Figure I.4: Structure générale des API.

I.8.2 Structure interne des API

Un API se compose de trois grandes parties :

- L'unité centrale de traitement (CPU).
- Le module Entrée/Sortie (E/S).
- Module d'alimentation.

I.8.2.1 Module d'alimentation

Le bloc d'alimentation (Power Supply PS) est nécessaire pour convertir la tension d'entrée alternative (220V) du secteur en une tension continue (24V, 48V..) nécessaire au processeur et aux circuits des modules d'interface d'entrée et de sortie.

La puissance des alimentations varie entre un API et un autre et demandent un courant allant de 2A à 50A, en fonction du nombre d'interfaces d'E/S alimentées par cette alimentation.

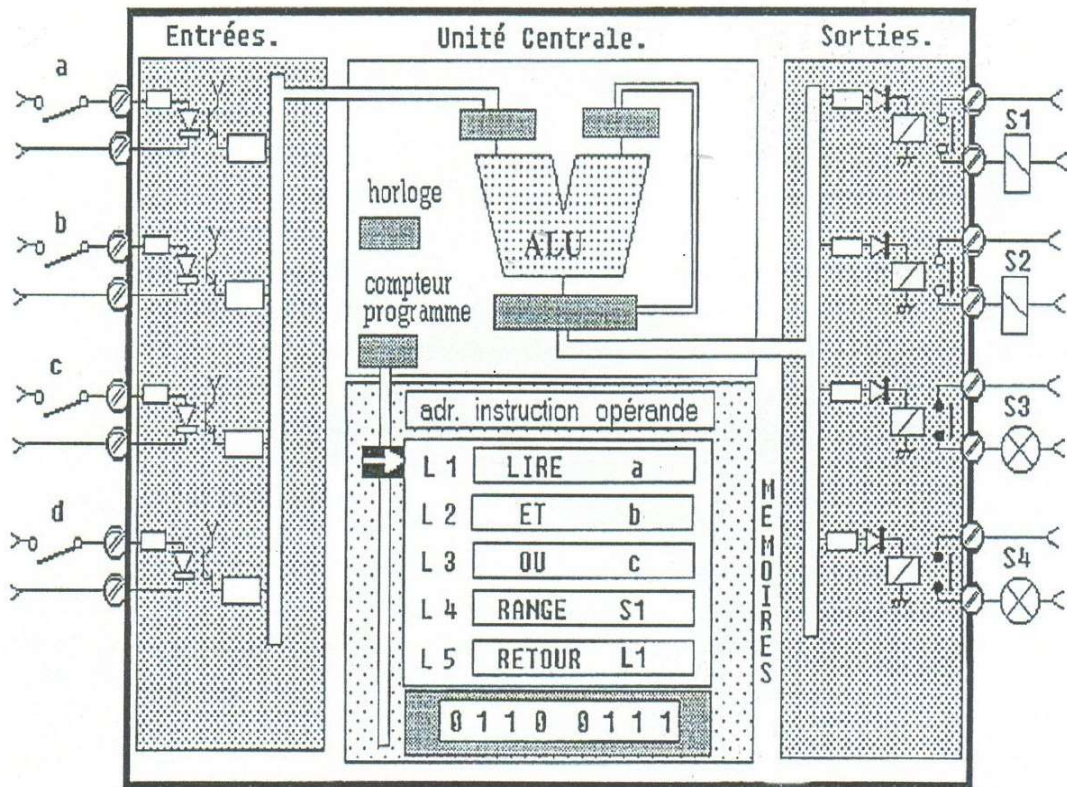


Figure I.5: Architecture Simplifier de l'intérieur d'un automate programmable.

I.8.2.2 L'unité centrale de traitement (CPU)

Le module CPU est l'unité contenant le microprocesseur. Cette unité interprète les signaux d'entrée et exécute les actions de commande en fonction du programme enregistré dans sa mémoire, communiquant les décisions sous forme des signaux d'actions aux sorties. Aussi, ce module contient une interface de programmation afin de communiquer avec la console de programmation suivant un protocole bien déterminé (Par exemple TCP/IP, MPI-bus ...etc.).

La mémoire de programme est l'endroit où le programme stocké contenant les actions de contrôle à exécuter par le microprocesseur, généralement c'est une mémoire ROM effaçable électriquement (EEPROM) d'une capacité varie du 4KB jusqu'au 50KB. Le rôle de l'UC est d'exécuter les programmes qui se trouvent dans la mémoire.

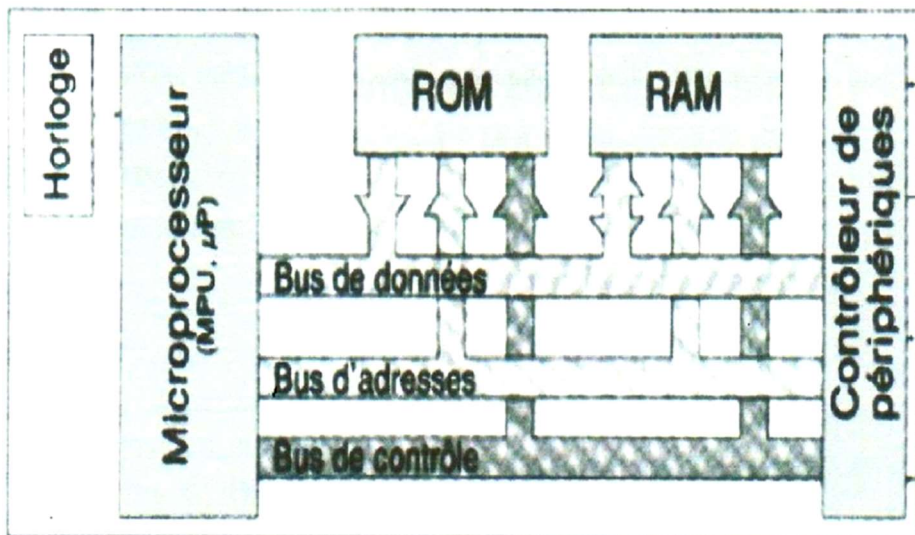


Figure I.6: Structure interne de CPU

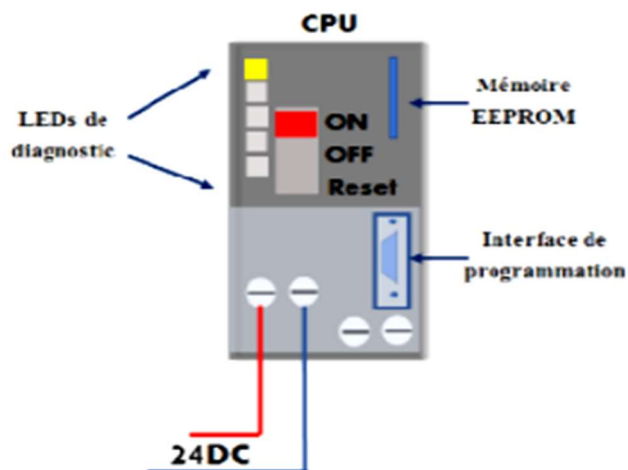


Figure I.7: Module CPU d'un API.

I.8.2.2.1 Le processeur

Il gère le fonctionnement de l'automate programmable et exécute les instructions du programme au rythme de son horloge. Il réalise toutes les fonctions logiques, arithmétiques et de traitement numérique (transfert, comptage, temporisation ...).

Les processeurs sont actuellement réalisés par des microprocesseurs. Au début de l'apparition des API, les processeurs étaient des processeurs câblés (réalisés par des circuits logiques séparés) ou des processeurs microprogrammés (qui utilisaient la logique programmable).

I.8.2.2.2 La mémoire centrale

La mémoire d'un automate comporte une zone programme, qui contient les programmes utilisateurs (à exécuter). Elle est en technologie RAM sauvegardée par pile ou batterie ou en EPROM, ou actuellement en EEPROM, elle comporte aussi une zone de donnée (ou mémoire de

données) et une zone réservée au moniteur ou système d'exploitation qui est le logiciel de gestion de l'API.

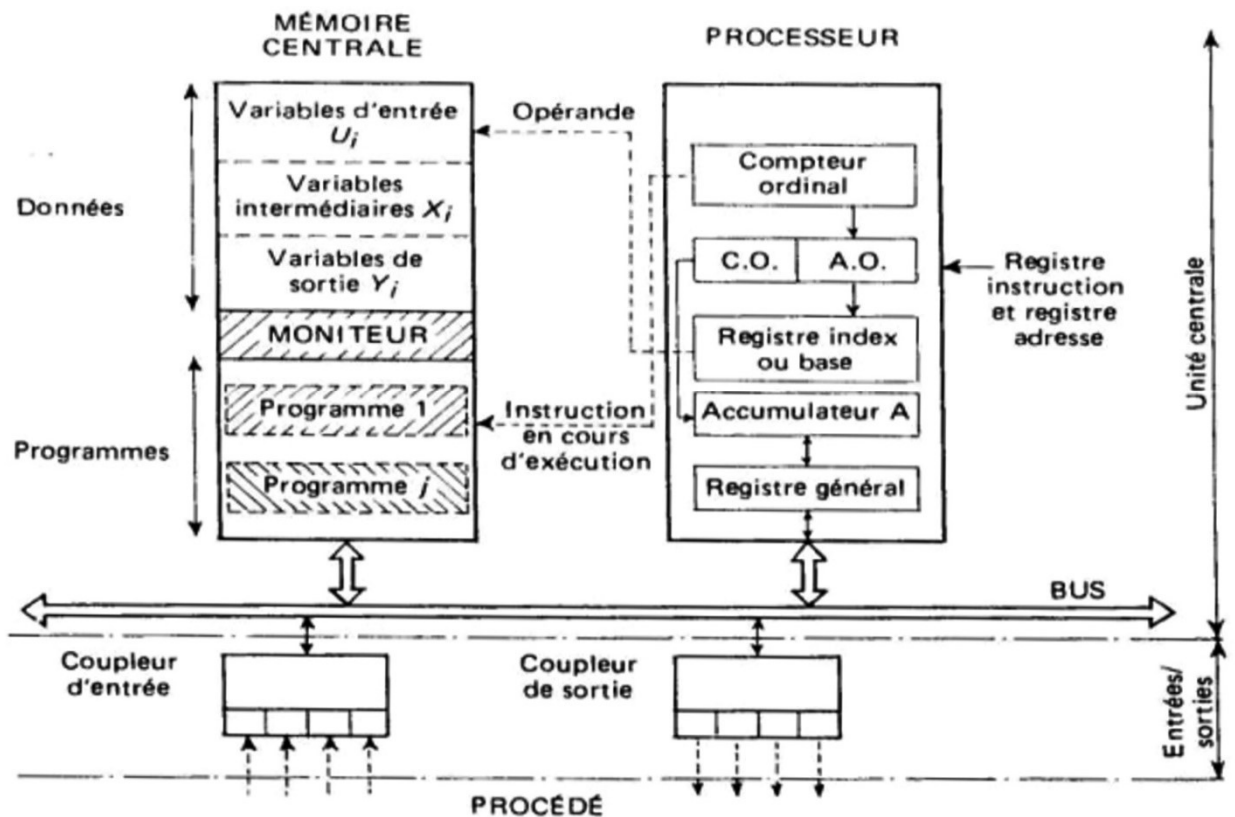


Figure I.8 : Structure de l'UC d'un API

I.8.2.2.3 Le Bus

C'est un ensemble de conducteurs sur lesquels circulent les signaux électriques. Il peut s'agir de pistes sur un circuit imprimé ou de fils dans un câble plat qui réalise la liaison entre les différents éléments de l'automate. Le bus est organisé en plusieurs sous-ensembles destinés chacun à véhiculer un type d'informations bien défini :

- **Bus de données :** Il peut réaliser des opérations entre des nombres et fournir des résultats sous forme de valeurs sur un nombre de bits bien défini.
- **Bus d'adresses :** Celui-ci transporte les adresses des emplacements mémoire. Pour que chaque mot puisse être localisé en mémoire,
- **Bus de contrôle :** Il sert, à informer les dispositifs mémoires s'ils vont recevoir des données à partir d'une entrée ou s'ils vont envoyer des données.

- **Bus système** : sert aux communications entre les ports d'entrées/sorties et l'unité d'entrées/sorties.

I.8.2.2.4 La mémoire

L'API accède aux données à traiter et aux instructions, c'est-à-dire au programme, qui lui explique comment traiter ces données. Ces informations sont stockées dans la mémoire de l'API qui est composée de :

- **Une mémoire morte (ROM, Read Only Memory)** : qui représente un espace de stockage permanent pour le système d'exploitation et les données figées utilisées par la CPU.
- **Une mémoire vive (RAM, Random Access Memory)** : utilisée pour le programme de l'utilisateur, et également pour les données. C'est à ce niveau que sont stockées les informations sur l'état des entrées et des sorties,
- **Une mémoire morte reprogrammable (EPROM, Erasable and Programmable Read Only Memory)** : elle est parfois employée pour stocker de manière permanente les programmes.

I.8.2.3 Interfaces d'entrée/sortie (E/S)

Les cartes d'E/S permettent au processeur de recevoir des informations de périphériques externes (capteurs) et de les communiquer aux périphériques externes (pré-actionneurs et actionneurs).

Plus de ces modules, on trouve des modules spéciaux d'E/S (carte PID, carte de comptage rapide ...etc.), ce type de cartes dotés des microprocesseurs, afin de simplifier les tâches et soulager le module CPU.

Les coupleurs d'entrées/sorties, appelés aussi interfaces d'entrées/sorties, sont des cartes électroniques qui assurent la liaison entre l'UC de l'automate programmable et la partie E/S.

Les coupleurs d'entrées reçoivent les signaux des capteurs et des commandes de l'opérateur (via le pupitre de commande) qu'ils traitent pour les rendre compatibles avec les caractéristiques internes de l'A.P.I.

Les coupleurs de sorties reçoivent les signaux de l'UC. Ils les amplifient et les rendent compatibles avec les pré-actionneurs commandés et les contrôles du pupitre de commande.

Pour la plupart des API, les coupleurs sont groupés avec l'UC, mais sont modulaires par carte ou par rack. Le coupleur accède d'une part au bus d'E/S, d'autre part au bornier qui se trouve le plus souvent sur la face avant de l'automate. Certains constructeurs proposent des E/S décentralisées

Les A.P.I. offrent une grande variété d'E/S comme :

- ✓ E/S T.O.R. (Tout Ou Rien c.à.d. binaires ou digital).
- ✓ E/S sur mot ou E/S numériques.
- ✓ E/S spéciales (modules intelligents) comme les E/S analogiques, les commandes d'axes, [8].

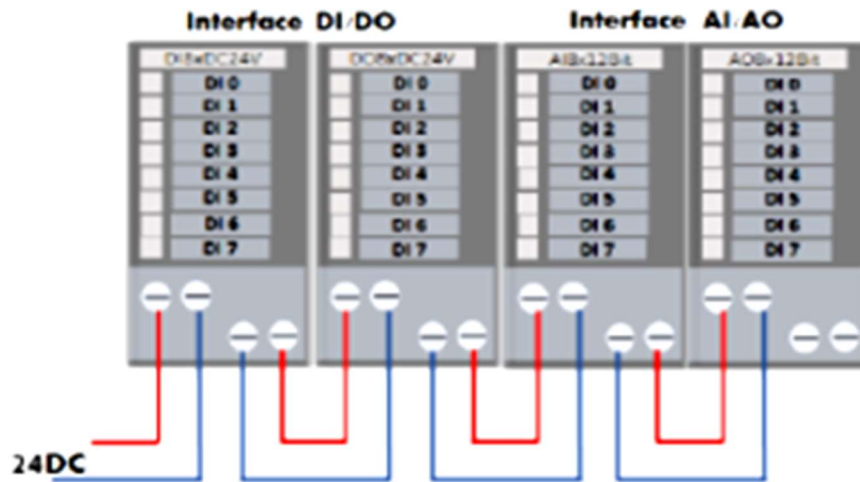


Figure I.9: Modules d'E/S.

a. Interfaces d'entrée

Ce sont des circuits spécialisés capables de recevoir en toute sécurité pour l'automate les signaux issus des capteurs ou de l'opérateur. Elles peuvent être :

- Logiques ou Tout Ou Rien : l'information ne peut prendre que deux états (vrai/faux, 0 ou 1 ...). C'est le type d'information délivrée par un détecteur, un bouton poussoir ...
- Numériques : l'information est contenue dans des mots codés sous forme binaire ou bien hexadécimale. C'est le type d'information délivrée par un ordinateur ou un module intelligent.
- Analogiques : l'information est continue et peut prendre une valeur comprise dans une plage bien déterminée. C'est le type d'information délivrée par un capteur (pression, température ...).[8].

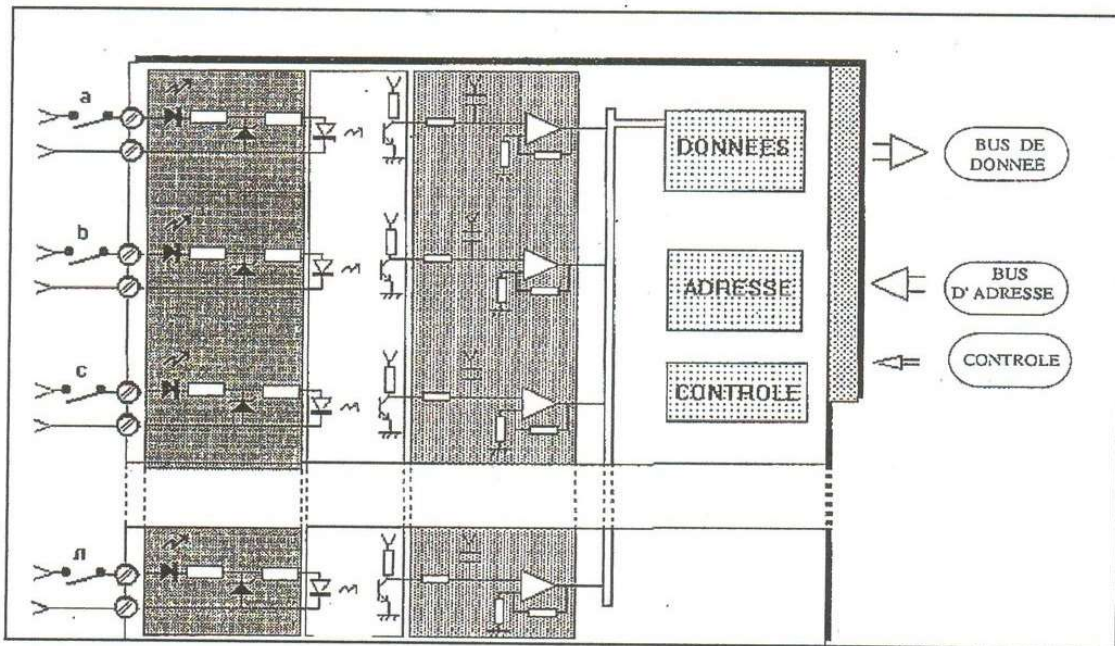


Figure I.10: Exemple de structure d'un module d'entrée logique.

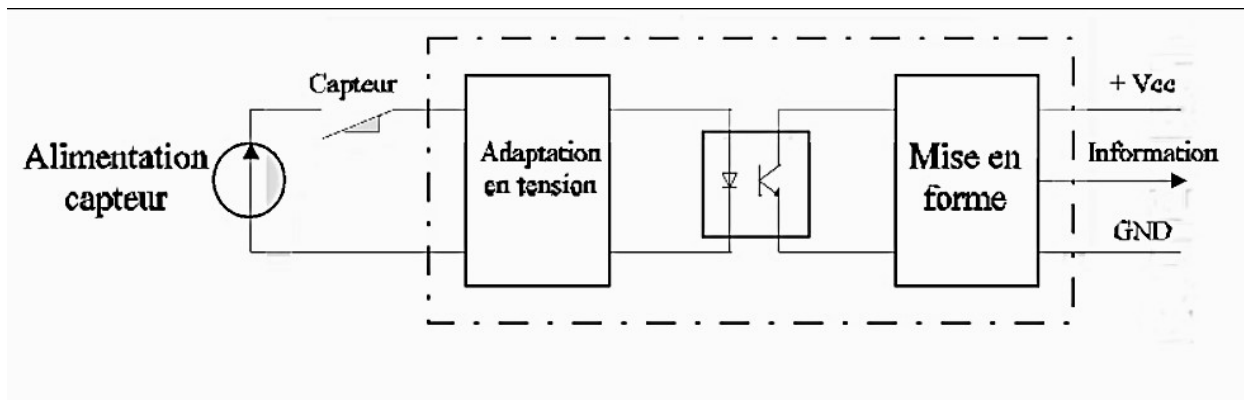


Figure I.11 : Structure d'une sortie d'un module logique

b. Interfaces de sortie

Ce sont des circuits spécialisés capables de commander en toute sécurité pour l'automate les circuits extérieurs. Elles peuvent être :

Logiques ou Tout Ou Rien : l'information ne peut prendre que deux états (vrai/faux, 0 ou 1 ...). C'est le type d'information délivrée par un détecteur, un bouton poussoir ...

Numériques : l'information est contenue dans des mots codés sous forme binaire ou bien hexadécimale. C'est le type d'information délivrée par un ordinateur ou un module intelligent [8].

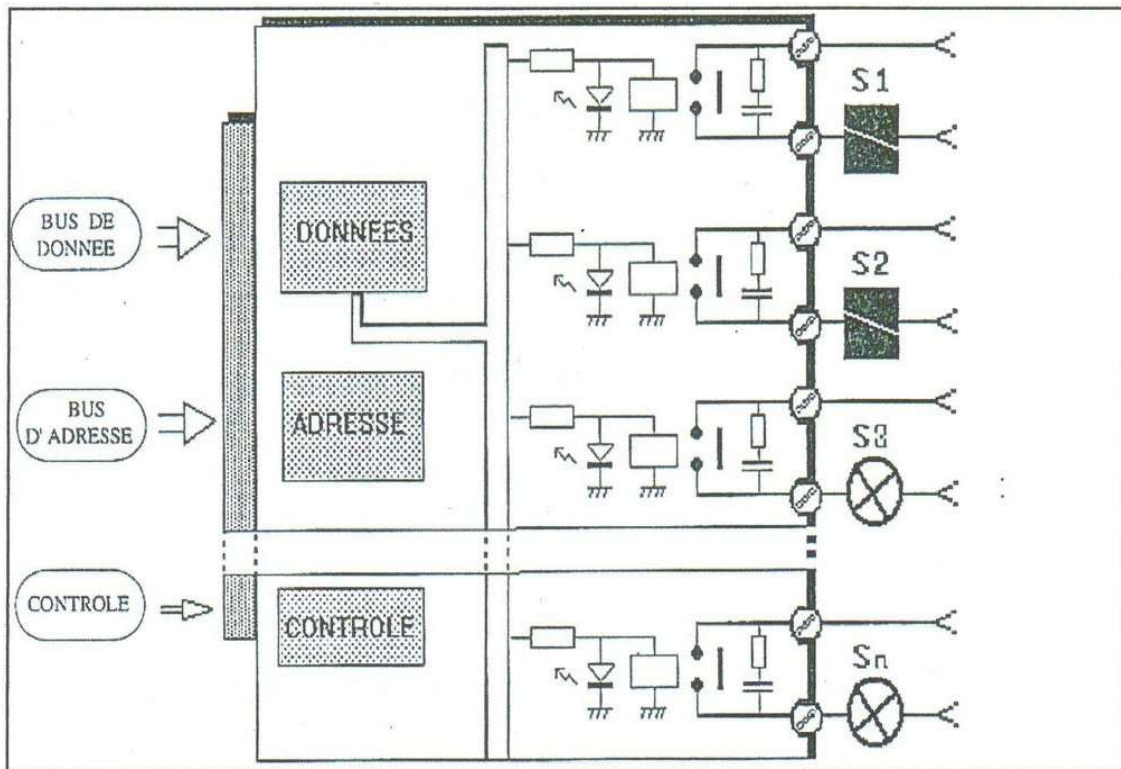


Figure I.12: Exemple de structure d'un module de sortie logique.

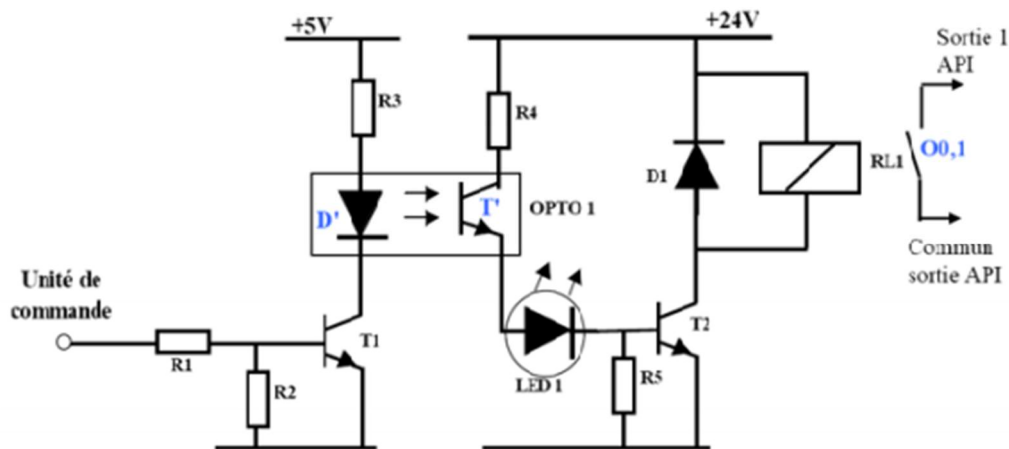


Figure I.13: Exemple de structure d'une sortie logique.

I.8.2.4 Console de programmation

Le dispositif de programmation est utilisé pour introduire le programme souhaité dans la mémoire programmable. Généralement le programme est développé dans un PC ou une console spéciale donnée par le constructeur, puis transféré dans la mémoire du CPU par l'intermédiaire d'un câble de communication adéquat (MPI-bus, TCP/IP...Etc.) [8].

La console de programmation est l'outil privilégié de la communication 'Homme-Machine' pour le développement, la mise au point et éventuellement l'exploitation des applications.

Le premier rôle de la console est de transformer le langage de programmation en instructions exécutables par l'automate. Elle permet lors de la mise au point du programme :

- ✓ La simulation pas à pas.
- ✓ La détection d'erreur de syntaxe.
- ✓ L'introduction, la correction et la modification des programmes (Editeur de textes).

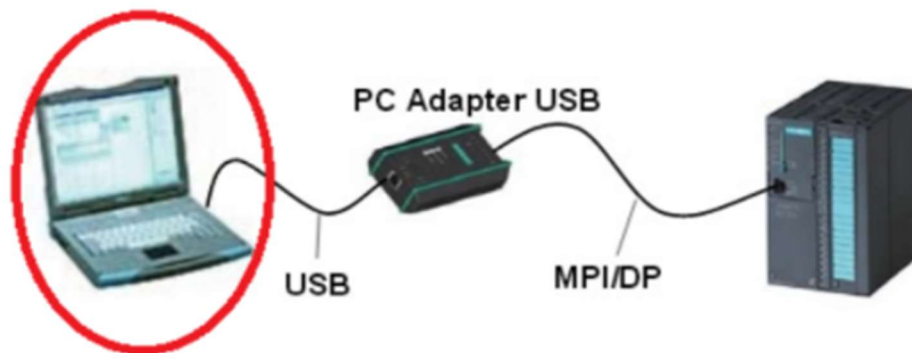


Figure I.14: Console de programmation.

I.9 Différents types des API

Les automates peuvent être de types compacts ou modulaire.

I.9.1 Type compact

On distinguera les modules de programmation (LOGO de Siemens, ZELIO de Schneider, MILLENIUM de Crouz...) des micros automates. Il intègre le processeur, l'alimentation, les entrées et les sorties. Selon les modèles et les fabricants, il pourra réaliser certaines fonctions supplémentaires (comptage rapide, ajout d'entrées/sorties analogiques ...) et recevoir des extensions en nombre limité. Ces automates, de fonctionnement simple, sont généralement destinés à la commande de petits automatismes. [9]



Figure I.15: Automate compact (Allen-Bradley).

I.9.2 Type modulaire

Le processeur, l'alimentation et les interfaces d'entrées/sorties résident dans des unités séparées (modules) et sont fixées sur un ou plusieurs racks contenant le "Fond de panier". Ces automates sont intégrés dans les automatismes complexes où puissants, où la capacité de traitement et flexibilité sont nécessaires.



Figure I.16 : Automate modulaire (Modicon).

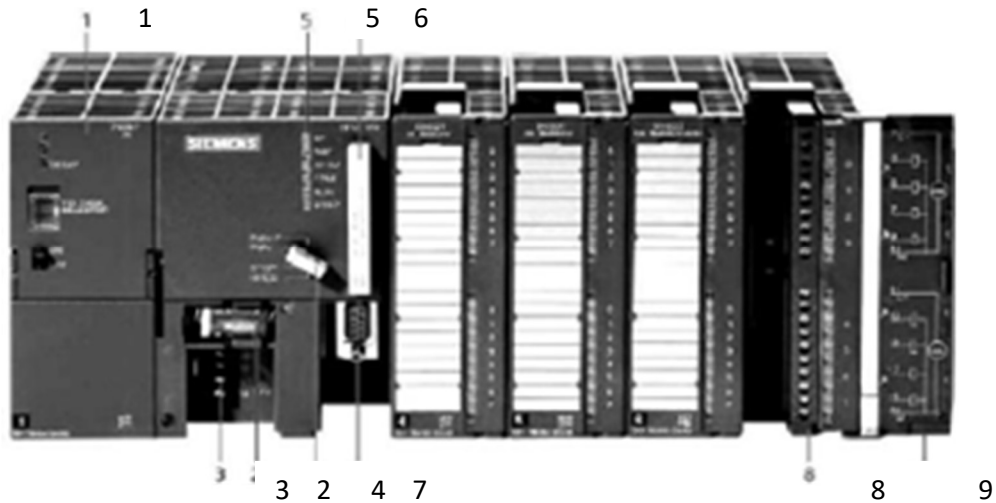


Figure I.17 : Automate modulaire (Siemens).

- | | |
|--|-------------------------------|
| 1. Module d'alimentation | 6. Carte mémoire |
| 2. Pile de sauvegarde | 7. Interface multipoint (MPI) |
| 3. Connexion au 24V cc | 8. Connecteur frontal |
| 4. Commutateur de mode (à clé) | 9. Volet en face avant |
| 5. LED de signalisation d'état et de défauts | |

I.10 Choix d'un automate industriel

On distingue plusieurs choix d'automate industrielle. Ils peuvent être répartis en quatre catégories principales :

- Automate fixe (dure).
- Automate programmable.
- Automate flexible (douce).
- Automate totalement intégrée (TIA).

I.10.1 Automate fixe (dure)

L'automatisation fixe fait référence à l'utilisation des équipements spéciaux pour automatiser une séquence fixe d'opérations de traitement ou d'assemblage. Il comprend :

- Assemblage mécanisé,
- Usinage de lignes de transfert,

- Manutention automatisée de matériel.

I.10.2 **Automate programmable**

Dans l'automatisation programmable, l'équipement de production est conçu avec la capacité de changer la séquence des opérations pour s'adapter à différentes configurations de produit. Il comprend : Robots industriels, Contrôleurs logiques programmables (API).

I.10.3 **Automate flexible (douce)**

Avec une automatisation flexible, plusieurs machines-outils sont reliées entre elles par un système de manutention et tous les composants du système sont contrôlés par un ordinateur central. Il comprend des bras de robot qui peuvent être programmés dans le but d'assumer plusieurs tâches

I.10.4 **Automate totalement intégrée (TIA)**

Plus une philosophie lancée par Siemens Automation and Drives qu'un système tangible, TIA comprend plusieurs concepts de base :

- ✓ Un environnement logiciel commun.
- ✓ Un système de gestion de données commun.
- ✓ Une méthode de communication commune. [10]

I.11 La Programmation (Logiciel / Langage / Réseaux)

L'écriture d'un programme consiste à créer une liste d'instructions permettant l'exécution des opérations nécessaires au fonctionnement du système. L'API traduit le langage de programmation en langage compréhensible directement par le microprocesseur. Ce langage est propre à chaque constructeur, il est lié au matériel mis en œuvre.

Chaque instruction du programme est composée de :

- l'opération à effectuer (la nature de l'opération est codée 1 ou 0).
- la variable sur laquelle l'opération va être effectué (variable de sortie, variable d'entrée, variable interne...).
- la nature de la variable (binaire, numérique, texte ...).

Chaque instruction est écrite dans une partie de la mémoire appelée adresse ou label. Il existe différents types de langage de programmation, les trois principaux sont :

I.11.1 Ladder Diagramme LD

Le langage LADDER est un schéma de relais qui ressemble aux schémas électriques, et permet de transformer rapidement une ancienne application faite de relais électromécaniques en un programme. C'est un langage graphique développé pour les électriciens. Ce langage est une représentation graphique d'équation booléennes combinant des contact (en entrée) et des relais (en sortie). Il utilise les symboles tels que : contacts, relais et blocs fonctionnels et s'organise en réseaux (labels). C'est le plus utilisé.

I.11.2 Le grafcet

Le Grafcet (graphe de commande étape transition) est un outil graphique de définition de l'automatisme séquentiel, en un nombre défini d'étapes, séparées par des conditions de transition. Il insiste d'un diagramme fonctionnel permettant de décrire, représenter et interpréter facilement un système automatisé

I.11.3 Langage IL

Le langage IL (instruction List), est un langage textuel de bas niveau. Les instructions opèrent toujours sur un résultat courant (ou registre IL). Un programme IL est une liste d'instructions dont chaque instruction doit commencer par une nouvelle ligne, et doit contenir un opérateur, compléter éventuellement par des modificateurs. Ce langage est particulièrement adapté aux applications de petite taille.

I.12 Conclusion

Dans ce chapitre nous avons présenté quelques notions de base et une description générale du matériel automate programmable. Un automate programmable est un outil adéquat pour les solutions d'automatisation. Il joue un rôle prépondérant pour assurer le bon fonctionnement de système commande dans différents domaines au moyen d'un programme qui est écrit et modifié à partir d'un terminal de programmations et de réglage.

CHAPITRE II
LOGICIEL GRAFCET ET
LADDER

II.1 Introduction

Maintenant que l'API a été introduit, passons à la programmation de l'API. Le premier, est toujours le langage de programmation le plus populaire, et la logique en échelle. À l'aide d'exemples, le langage est développé à partir du schéma de câblage du système de relais électromécanique. Après décrivant les symboles de base des différents processeurs couverts par ce texte, ils sont combinés dans un diagramme en échelle. La section suivante détaille le processus de numérisation d'un programme et accéder aux entrées et sorties physiques. Programmation avec le normalement

Le contact fermé fait l'objet d'une attention particulière car il est souvent mal appliqué par le novice programmeurs. Pour solidifier ces concepts, le démarrage / arrêt d'un appareil physique est envisagé. Le démarrage / arrêt est une application API très courante et se produit dans de nombreux autres contextes. La section optionnelle sur la conversion de relais en logique à relais API conclut le chapitre

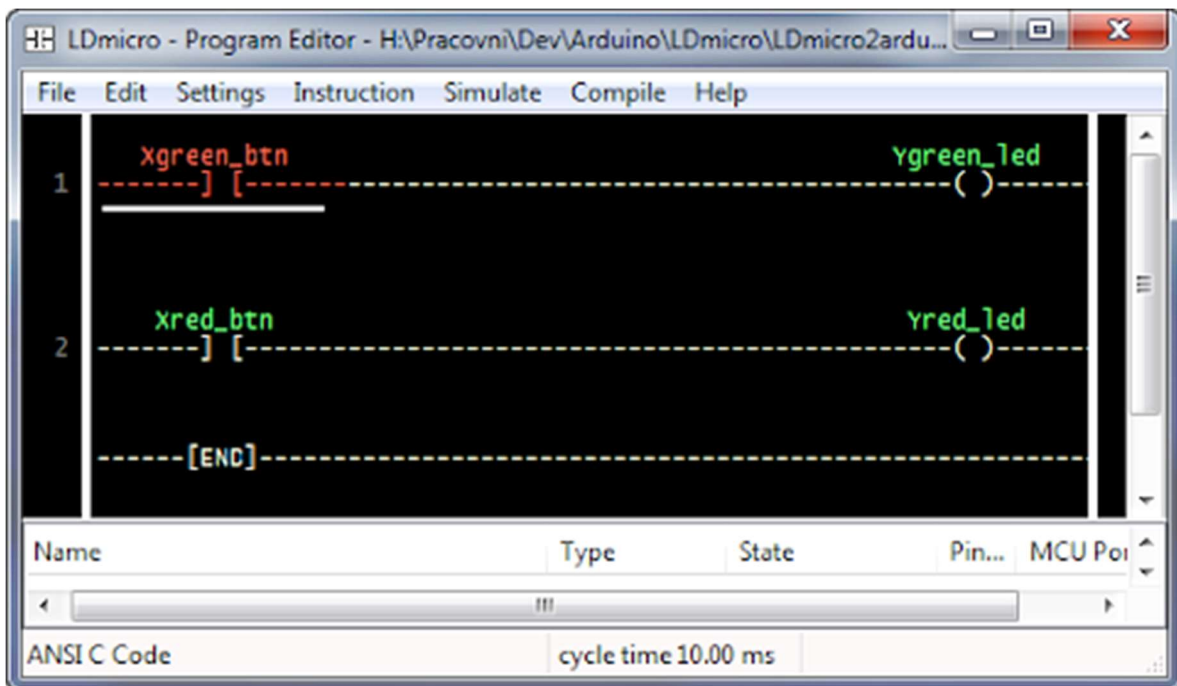


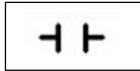
Figure II.1 : Interface Logiciel LDmicro

II.2 Ladder

II.2.1 Instructions de base pour le Ladder

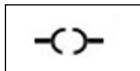
LADDER Programme est la base de la plupart des fonctions de contrôle. La logique à relais utilise des contacts de commutation ou de relais pour implémenter des expressions booléennes. Dans le passé, la logique en échelle était rendue possible avec des relais discrets, et était parfois appelée logique de relais. Aujourd'hui, la plupart des implémentations sont effectuées à l'aide d'un dispositif spécialisé basé sur un microprocesseur appelé : contrôleur logique programmable (API). Bien que les moyens de mise en œuvre parmi ces Instructions on trouve [11] :

II.2.1.1 Contact normalement ouvert :



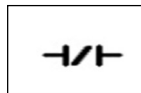
Cela peut être utilisé pour représenter n'importe quelle entrée de la logique de commande, telle qu'un interrupteur ou capteur, un contact d'une sortie ou une sortie interne. Une fois résolu, l'entrée référencée est examinée pour une condition vraie (logique 1). Si c'est vrai, le contact se ferme et autorisera logique à couler de gauche à droite. Si l'état est faux (0 logique), le contact est ouvert et la logique ne coulera pas de gauche à droite.

II.2.1.2 Bobine normalement ouverte :



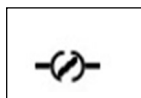
Cela peut être utilisé pour représenter n'importe quelle sortie TOR de la logique de commande. Lorsque "résolu" si la logique à gauche de la bobine est TRUE, la sortie référencée est TRUE (logique 1).

II.2.1.3 Contact normalement fermé :



Une fois résolue, l'entrée référencée est examinée pour une condition OFF. Si le statut est OFF (0 logique) l'alimentation (logique) circule de gauche à droite. Si l'état est ON, l'alimentation ne coulera pas.

II.2.1.4 Bobine normalement fermée :



Lorsqu'il est "résolu" si la bobine est un 0 logique, l'appareil est mis sous tension. Si le périphérique est logique 1, l'alimentation sera coupée.





Ladder	Instruction list	Structured text	Operands
	LD	:=	%I,%Q,%M,%S,%BLK,%•:Xk, %Xi, (True and False in instruction list or structured text)
	LDN	:=NOT	%I,%Q,%M,%S,%BLK,%•:Xk, %Xi, (True and False in instruction list or structured text)
	LDR	:=RE	%I,%Q,%M
	LDF	:=FE	%I,%Q,%M

Figure II.2:Un tableau des opérandes utilisés pour ces instructions.

II.2.1.5 Portes ET et OU :

Le ET est une condition logique fondamentale de base qui est facile à représenter directement dans la logique Ladder.

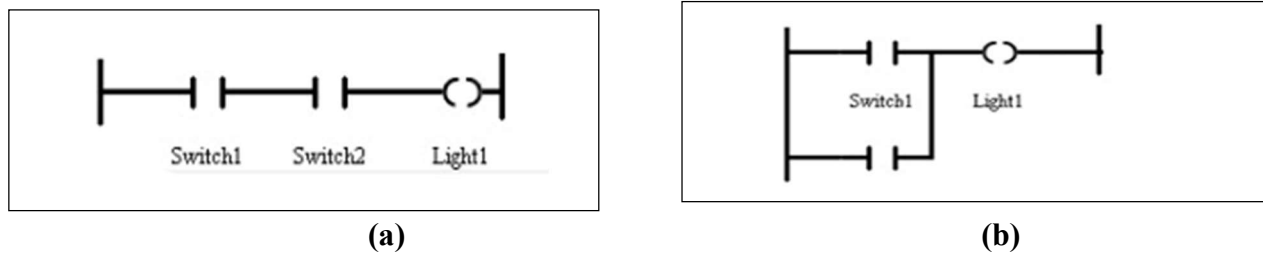


Figure II.3: langage Ladder (a) Porte ET simplifiée. (b) Porte OU simplifiée

II.2.1.6 Minuteries et compteurs :

Plusieurs fois, les programmes appelleront à prendre des mesures dans un programme de contrôle basé sur plus que les états des entrées et sorties TOR. Parfois, les processus devront s'allumer après un délai ou compter le nombre de fois qu'un interrupteur est touché. Pour faire ces simples tâches, les minuteries et les compteurs sont utilisés.

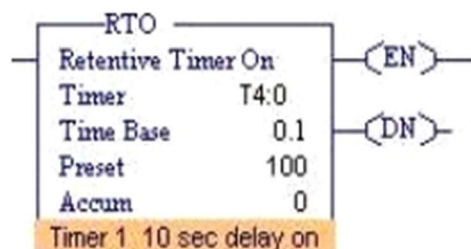


Figure II.4:Minuterie de mise en marche (On-Delay Timer : RTO).

II.2.2 Fonction de Ladder

Lors de l'étude de la logique, il est essentiel de commencer par les fonctions de base. Les valeurs d'entrée peuvent être combinées à l'aide des fonctions logiques ET, OU et OU exclusif (XOR). Les portes logiques utilisent l'électronique numérique pour mettre en œuvre ces fonctions. Chaque porte est considérée comme un circuit, généralement composé de transistors et de résistances de polarisation. A titre d'exemple, la puce à logique transistor-transistor (TTL) 7408 contient quatre portes ET à deux entrées dans un boîtier de circuit intégré (IC). Ces portes et d'autres types sur des circuits intégrés séparés peuvent être câblés ensemble pour mettre en œuvre un large éventail de logique numérique.

Nous pouvons construire simplement des fonctions logiques pour notre circuit de lampe hypothétique, en utilisant plusieurs contacts, et documentez ces circuits assez facilement et de manière compréhensible avec des échelons à notre « échelle » d'origine. Si nous utilisons la notation binaire standard pour le statut du interrupteurs et lampe (0 pour non actionner ou désexcité ; 1 pour actionner ou excité). Une table de vérité peut être faite pour montrer comment la logique fonctionne :

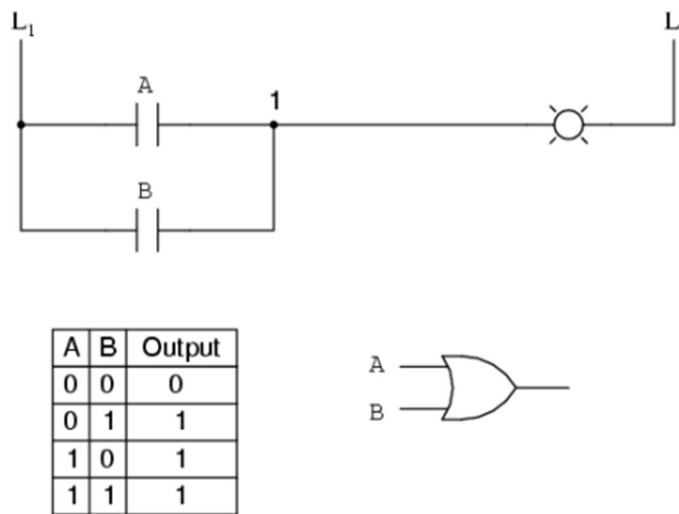


Figure II.5 : Table logique d'un LADER

II.2.3 Diagramme de la logique LADDER

Les 7 parties de base d'un Ladder sont détaillées ci-dessous [12] :

- *Rails* : Il y a deux rails dans un diagramme en échelle qui sont dessinés sous forme de lignes verticales descendant les extrémités les plus éloignées de la page. S'ils étaient

dans un circuit logique de relais, ils représenteraient les connexions actives et à zéro volt de l'alimentation électrique où le flux d'énergie va du côté gauche au côté droit.

- *Échelons (Rungs)* : Les échelons sont dessinés sous forme de lignes horizontales et relient les rails aux expressions logiques. S'ils se trouvaient dans un circuit logique de relais, ils représenteraient les fils qui relient l'alimentation aux composants de commutation et de relais. Chaque ligne est numérotée par ordre séquentiel croissant.
- *Entrées* : Les entrées sont des actions de contrôle externes telles qu'un bouton poussoir enfoncé ou un interrupteur de fin de course déclenché. Les entrées sont en fait câblées aux bornes de l'API et représentées dans le schéma à contacts par un *symbole de contact* normalement ouvert (NO) ou normalement fermé (NC).
- *Sorties* : Les sorties sont des dispositifs externes qui sont activés et désactivés comme un moteur électrique ou une électrovanne. Les sorties sont également câblées aux bornes de l'API et sont représentées dans le schéma à relais par un *symbole de bobine* de relais.

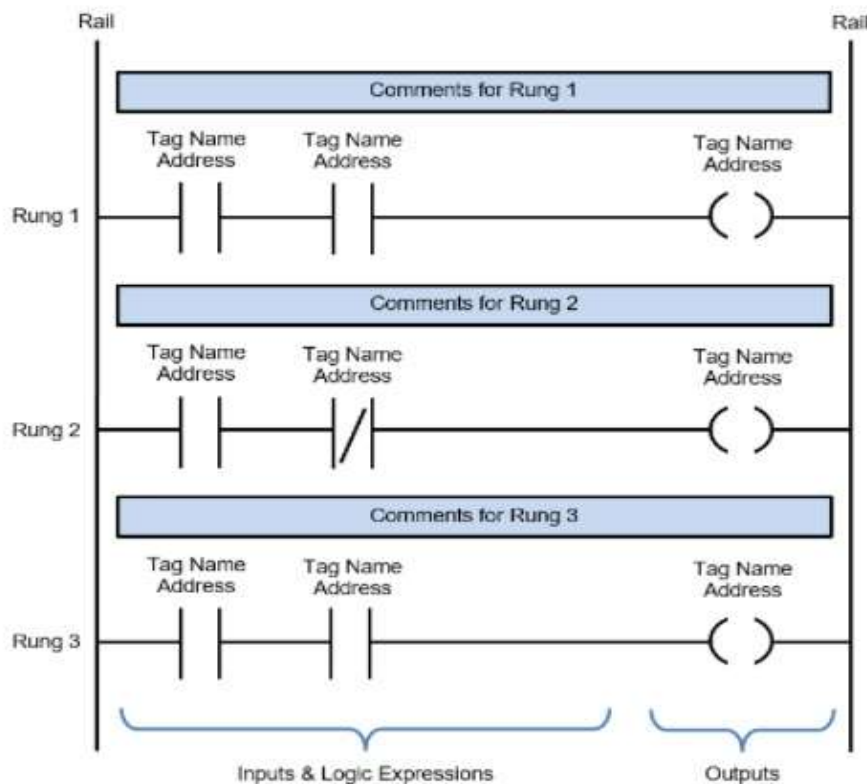


Figure II.6:Diagramme d'un Ladder logique simplifié.

- *Expressions logiques* : Les expressions logiques sont utilisées en combinaison avec les entrées et les sorties pour formuler les opérations de contrôle souhaitées.
- *Notation d'adresse et noms de point* : La notation d'adresse décrit la structure d'adressage de la mémoire d'entrée, de sortie et d'expression logique de l'API. Les noms de balises sont les descriptions attribuées aux adresses.
- *Commentaires* : Dernier point mais non le moindre, les commentaires sont une partie extrêmement importante d'un diagramme en échelle. Les commentaires sont affichés au début de chaque ligne et sont utilisés pour décrire les expressions logiques et les opérations de contrôle que la ligne, ou les groupes de lignes, exécutent.

La compréhension des schémas à contacts est beaucoup plus facile en utilisant des commentaires.

II.3 Grafcet

II.3.1 Historique De Grafcet

En 1975, un groupe d'universitaires et industriels de la section "Systèmes Logiques" de l'AFCEC (*Association Française de Cybernétique Economique et Technique*) se sont fixés l'objectif de définir un formalisme adapté à la représentation des évolutions séquentielles d'un système. Au début, le travail consista à dresser un état de l'art des différentes approches de modélisation du comportement de tels automatismes. L'analyse des avantages et inconvénients de ces outils a mené en 1977 à la définition du GRAFCET, ainsi nommé pour, à la fois marquer l'origine de ce nouvel outil de modélisation « AFCEC » et son identité Graphe Fonctionnel de Commande Etapes–Transitions. Les résultats de ces travaux furent l'objet d'une publication officielle dans la revue "*Automatique et Informatique Industrielle*" en décembre 1977, date que la communauté considère aujourd'hui comme correspondant à la date de naissance effective du GRAFCET.

II.3.2 Introduction

Grafcet (ou Graphe Fonctionnel de Commande Etape-Transition) est le point de départ du développement de tout projet d'automatisation. En effet, c'est un outil de représentation graphique du cahier des charges qui accompagnera le système automatisé, de sa conception à son exploitation. C'est un excellent outil pour, concevoir, documenter et démontrer le contrôle flux d'un système. Les primitives de base de la langue Grafcet sont décrites en détail, et le contrôle parallèle des constructions dans le Grafcet sont expliquées. C'est un langage

graphique représentant le fonctionnement d'un automatisme par un ensemble d'étapes auxquelles sont associées des actions, de transitions entre étapes et des liaisons orientées entre les étapes et les transitions.

En résumé le Grafcet est un outil graphique qui permet l'élaboration des séquences d'un système automatisé d'une façon littérale dès le début du projet sans exiger une connaissance approfondie des technologies utilisées.

II.3.3 Avantage De Grafcet

Ce qui fait de Grafcet un outil utile pour la conception contrôle de fabrication c'est son utilisation d'une notation graphique pour sa langue. La représentation graphique du contrôle facilite la compréhension, la documentation et le prototypage.

Grafcet a également des constructions parallèles inhérentes à la langue. Ces constructions permettent à l'utilisateur de se repentir systèmes de distribution d'une manière élégante. Pour terminer. Savoir a amélioré Grafcet avec une programmation conviviale environnement, parmi ces avantages :

- ✓ Simple.
- ✓ Accepté par tous ;
- ✓ Intelligible à la fois par les concepteurs et les exploitants ;
- ✓ Fournissant potentiellement des facilités de passage à une réalisation, à base matérielle et/ou logicielle de l'automatisme ainsi spécifié. [13]

II.3.4 Les concepts de base du Grafcet

En générale il existe 4 Concept de base du Grafcet, on distingue :

II.3.4.1 Les Etapes

Une étape symbolise un état ou une partie de l'état du système automatisé. L'étape possède deux états possibles : active représentée par un jeton dans l'étape ou inactive. L'étape i , représentée par un carré repéré numériquement, possède ainsi une variable d'état, appelée variable d'étape X_i .

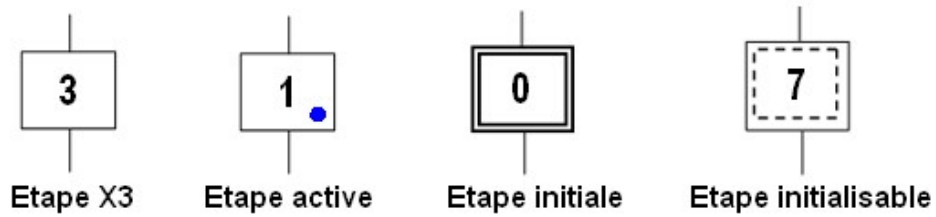


Figure II.7: Représentation d'une étape du grafcet et l'étape initiale.

La situation initiale d'un système automatisé est indiquée par une étape dite étape initiale et représentée par un carré double. [14]

II.3.4.2 Actions associées aux étapes

A chaque étape est associée une action ou plusieurs, c'est à dire un ordre vers la partie opérative ou vers d'autres grafquets. Mais on peut rencontrer aussi une même action associée à plusieurs étapes ou une étape vide (sans action).

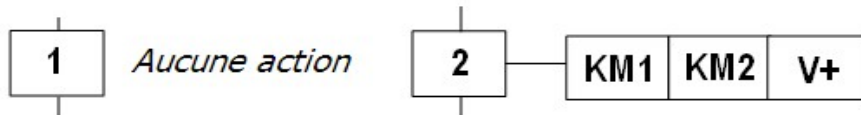


Figure II.8: Représentation normalisée des actions associées aux étapes du grafcet.

II.3.4.3 Transition

Une transition indique la possibilité d'évolution qui existe entre deux étapes et donc la succession de deux activités dans la partie opérative. Lors de son franchissement, elle va permettre l'évolution du système.

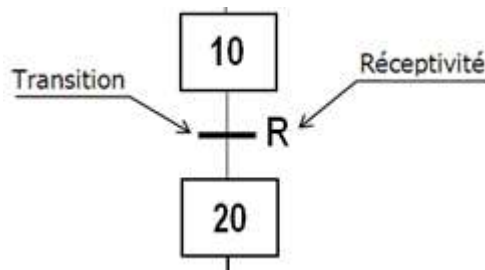


Figure II.9 : Représentation normalisée des Transition.

II.3.4.4 Liaisons orientées

Elles sont de simples traits verticaux qui relient les étapes aux transitions et les transitions aux étapes. Elles sont normalement orientées de haut vers le bas. Une flèche est nécessaire dans le cas contraire.

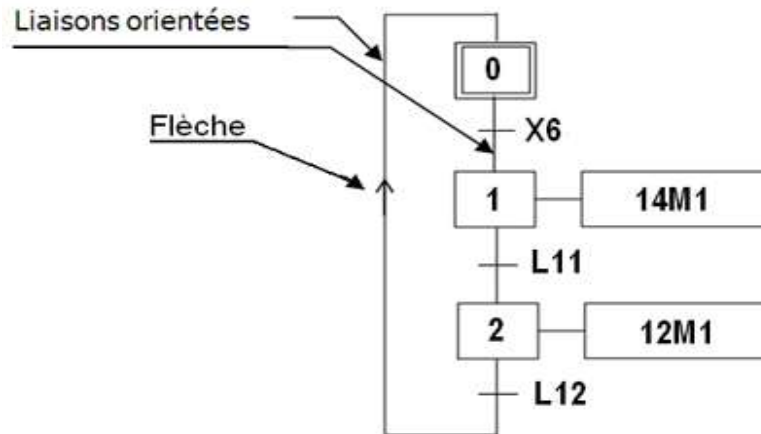


Figure II.10: Représentation des Liaisons orientées.

II.3.5 Les règles de Grafcet

- **Règle 1** : état initial :

L'évolution de l'état nécessite des étapes actives au début de l'opération (au moins une).

- **Règle 2** : Transposition d'une transition :

Une transition est active ou activée que si toutes les étapes à son entrée sont actives (sinon, elle est inactive). Une Transition ne peut être transposée que si elle est active et est vraie la condition associée (fonction de réceptivité).

- **Règle 3** : évolution des étapes actives :

La transposition d'une Transition conduit à la désactivation de toutes les étapes sur ses entrées et l'activation de toutes les étapes sur ses sorties.

- **Règle 4** : Transposition simultanée des transitions :

Toutes les transitions actives sont transposées simultanément.

- **Règle 5** : Activation et désactivation simultanées d'un Step :

Dans ce cas, l'activation est prioritaire. [14]

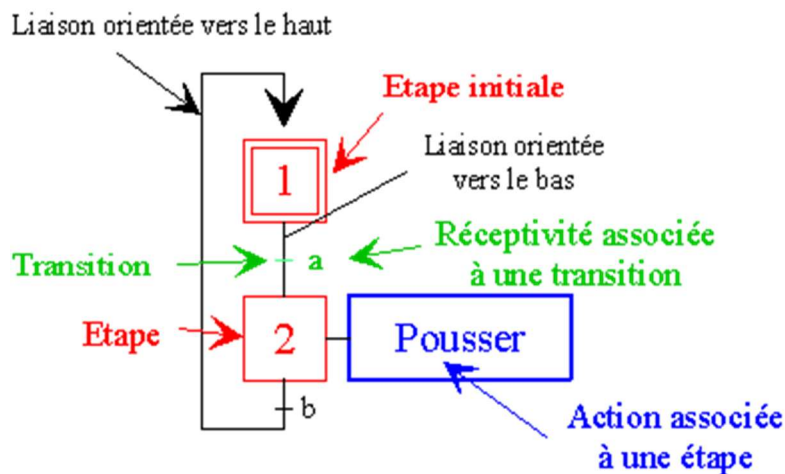


Figure II.11:Schéma regroupant les règles et le fonctionnement de Grafcet.

II.3.6 Description de Grafcet

La description du comportement attendu d'un automatisme peut être représentée par un GRAFCET d'un certain « niveau ». La caractérisation de ce dernier nécessite de prendre en compte trois dimensions :

La 1^{ère} : Le point de vue, caractérisant le point de vue selon lequel un observateur est impliqué dans le fonctionnement du système pour donner une description. Il existe trois points de vue :

- ✓ Un point de vue système,
- ✓ Un point de vue de la partie opérative,
- ✓ Un point de vue Order Party.

La 2^{ème} : Le cahier des charges, caractérisant la nature des spécifications techniques auxquelles le client doit répondre. On note trois groupes de spécifications :

- ✓ Spécifications fonctionnelles,
- ✓ Spécifications technologiques,
- ✓ Spécifications opérationnelles.

La 3^{ème} : La finesse, caractérisant le niveau de détail dans la description de l'opération, du niveau global au niveau de détail complet. [15]

II.4 Conclusion

Ce Chapitre traite la compréhension de la logique Ladder et de la programmation qui y est associée. Ainsi dû le Langage Grafcet et sa relation avec les PLC et les LD. Enfin, il discute sur la logique de relais et sur l'évolution que la logique de l'échelle en a faite.

CHAPITRE III

ARDUINO

III.1 Introduction

Aujourd'hui, l'électronique est de plus en plus remplacée par de l'électronique programmée. On parle de système embarqué ou d'informatique embarquée. Son but est de simplifier les schémas électroniques et par conséquent de réduire l'utilisation de composants électroniques, réduisant ainsi le coût de fabrication d'un produit. Il en résulte des systèmes plus complexes et performants pour un espace réduit.

Depuis que l'électronique existe, sa croissance est fulgurante et continue encore aujourd'hui. L'électronique est devenue accessible à toutes personnes ayant l'envie d'y accéder.

Ce que nous allons apprendre dans ce travail est un mélange d'électronique et de programmation. Nous allons en effet parler d'électronique embarquée qui est un sous domaine de l'électronique et qui a l'habileté d'unir la puissance de la programmation à la puissance de l'électronique.

III.2 Définition d'un Arduino

Arduino est un circuit imprimé il englobe des composant, le plus important un microcontrôleur qui est le cerveau de la carte programmé pour tester et produire des signaux électriques, de manière à effectuer plusieurs tâches comme les techniques d'électronique.

La carte Arduino est utilisée pour réaliser des projets électroniques plus développée. Elle est composée d'un circuit physique programmable, est dite microcontrôleurs et de logiciel utilisé pour créer et télécharger le code de l'ordinateur à la carte. [16]

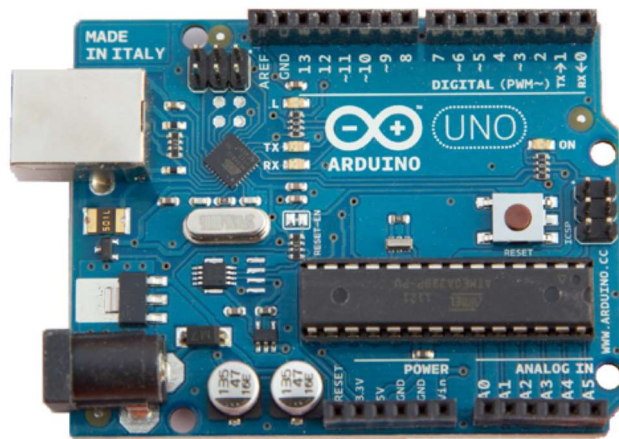


Figure III.1 : Description de la carte Arduino.

III.3 Les gammes de la carte Arduino

Actuellement, il existe plus de 20 versions de module Arduino, nous citons quelques-unes afin d'éclaircir l'évaluation de ce produit scientifique et académique :

III.3.1 La carte Arduino UNO

C'est la carte idéale pour découvrir l'environnement ARDUINO. Elle permet à tout débutant de se lancer dans tous ses premiers petits projets. Comme c'est la carte la plus utilisée, il est très facile de se référer aux tutoriels très nombreux sur le net et ainsi de ne pas rester seul dans son exploration. Sa simplicité devient par contre un handicap lorsqu'il s'agit de multiplier les périphériques, de manipuler des algorithmes lourds ou d'interagir avec les OS Android pour lesquels d'autres cartes Arduino sont plus adaptées. [17]

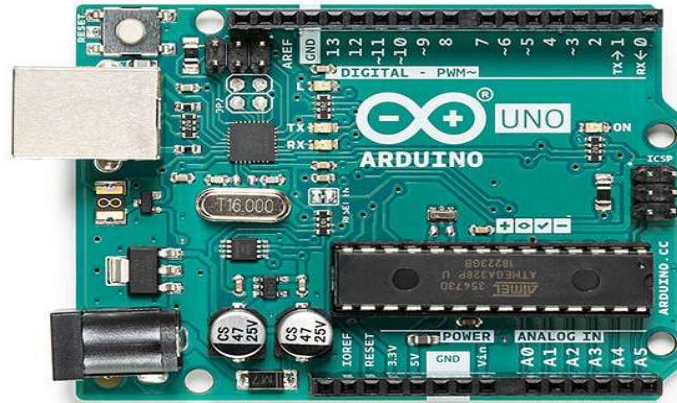


Figure III.2 : La carte Arduino UNO.

III.3.2 La carte Arduino Leonardo

C'est la carte qui est prévue pour succéder à la carte Arduino Uno en présentant des caractéristiques équivalentes mais une ergonomie revue et une stabilité plus éprouvée. [17]



Figure III.3:carte Arduino Lenardo.

III.3.3 La carte Arduino Méga

La carte Arduino Méga est la carte la plus diffusée après la carte Arduino Uno. Elle offre un nombre d'entrées/sorties beaucoup plus important, un processeur plus puissant doté d'une mémoire plus vaste qui permet d'exploiter des algorithmes plus complexes. [17]



Figure III.4: Carte Arduino Méga.

III.3.4 La carte Arduino Méga ADK

La carte Arduino méga ADK offre les mêmes caractéristiques techniques que la carte Arduino méga mais son port USB permet de la connecter avec un environnement Android ouvrant de nouvelles perspectives d'interaction avec le monde des smartphones et des capteurs dont ils sont dotés. Sa mise en œuvre nécessite par contre de solides connaissances en Java et la capacité à développer ses propres applications. [17]



Figure III.5: carte Arduino Méga ADK.

III.3.5 La carte Arduino Due

Elle permet de manipuler rapidement des algorithmes lourds, particulièrement utiles dans le monde de la robotique par exemple. [17]



Figure III.6: Carte Arduino Due.

III.3.6 La carte Arduino Nano

La carte Arduino nano n'est ni plus ni moins qu'une carte Arduino uno miniaturisée. Sa taille et son poids réduits la destinent à une utilisation dans des espaces réduits (en textile par exemple) ou dans des applications de robotique [17].

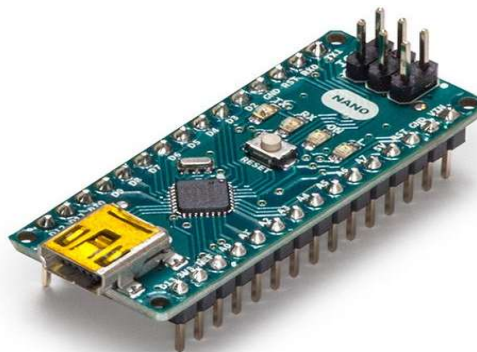


Figure III.7: Carte Arduino Nano

III.3.7 La carte Arduino Mini Pro

La carte Arduino Mini Pro est une carte Arduino Uno simplifiée à l'extrême permettant néanmoins de piloter de petits projets ou certains éléments d'un projet. Attention, cette carte n'intègre pas de port USB ce qui rends sa connectivité délicate. [17].

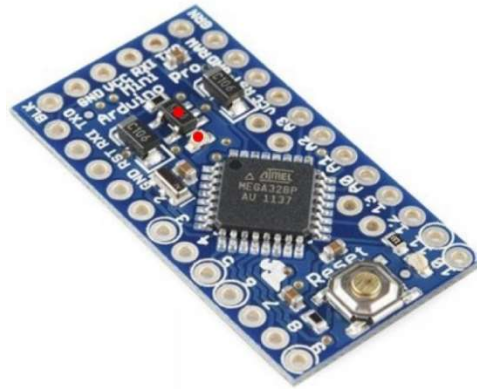


Figure III.8 : carte Arduino Mini Pro.

III.3.8 La carte Arduino Yun

La carte Arduino Yun, récemment proposée par Arduino, est conçue pour contrer les avantages de la carte Raspberry. Elle est un dérivé de la carte Leonardo et a pour objectif de combiner la puissance de Linux avec la facilité d'utilisation d'une carte Arduino. Elle est également la première carte Arduino à être dotée nativement d'un wifi intégré. [17].



Figure III.9:carte Arduino Yun.

III.4Eléments des cartes Arduino

Les éléments d'une carte Arduino peuvent être divisés en deux catégories :

- Catégories Matériel
- Catégories Logiciel

III.4.1 Catégories Matériel

La carte de développement Arduino se compose de nombreux composants qui, ensemble, le font fonctionner.

Ici sont quelques-uns de ces principaux blocs de composants qui aident à son fonctionnement :

- **Microcontrôleur** : c'est le cœur de la carte de développement, qui fonctionne comme un mini-ordinateur et peut recevoir et envoyer des informations ou des commandes aux périphériques connectés à lui.
- **Alimentation externe** : cette alimentation est utilisée pour alimenter la carte de développement Arduino avec une tension régulée allant de 9 à 12 volts.
- **Prise USB** : Cette prise est un port très important de cette carte. Il est utilisé pour télécharger (graver) un programme au microcontrôleur à l'aide d'un câble USB. Il a également une puissance régulée de 5V qui alimente également la carte Arduino en cas d'absence d'alimentation externe.
- **Programmeur interne** : le code logiciel développé peut être téléchargé sur le microcontrôleur via port USB, sans programmeur externe.
- **Bouton de réinitialisation** : ce bouton est présent sur la carte et peut être utilisé pour réinitialiser l'Arduino microcontrôleur.
- **Broches analogiques** : il existe des broches d'entrée analogiques allant de A0 à A7 (typique). Ces broches sont utilisées pour l'entrée / sortie analogique. Le nom des broches analogiques varie également d'une carte à l'autre.

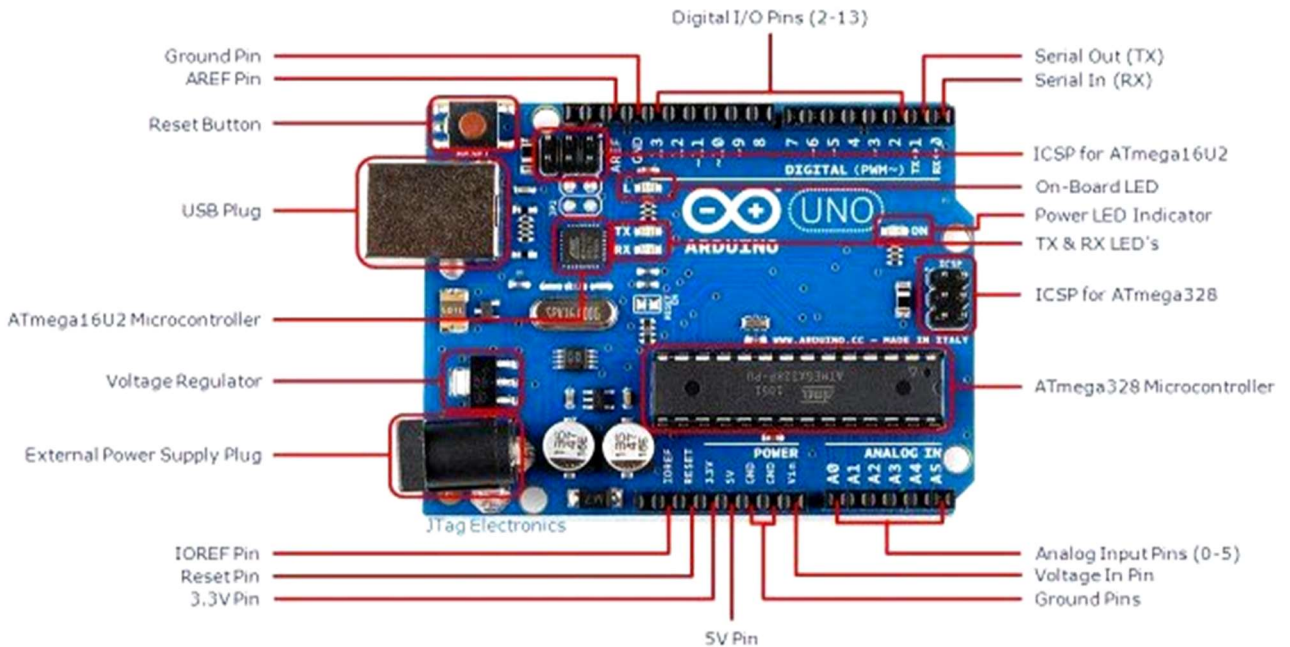


Figure III.10 : Représentation des éléments d'une carte Arduino.

- **Broches d'E/S numériques** : certaines broches d'entrée numériques sont également comprises entre 2 et 16. Celles-ci les broches sont utilisées pour l'entrée / sortie numérique.
- **Broches d'alimentation et GND** : il y a des broches sur la carte de développement qui fournissent 3,3 ; 5 volts et terre à travers eux.

III.4.2 Catégories logiciel

Le code de programme écrit pour Arduino est connu sous le nom de croquis. Le logiciel utilisé pour développer de telles esquisses pour un-Arduino est communément connu sous le nom d'IDE Arduino. Cet IDE contient les éléments suivants parties de celui-ci :

- **Éditeur de texte** : c'est ici que le code simplifié peut être écrit à l'aide d'une version simplifiée de C ++ langage de programmation.
- **Zone de message** : elle affiche une erreur et donne également un retour sur l'enregistrement et l'exportation du code.
- **Texte** : la console affiche la sortie de texte par l'environnement Arduino, y compris l'erreur complète messages et autres informations.
- **Barre d'outils de la console** : cette barre d'outils contient divers boutons tels que Vérifier, Télécharger, Nouveau, Ouvrir, Enregistrer et Serial Monitor. Dans le coin inférieur droit de la fenêtre, le Carte de développement et port série utilisé. **[18]**

III.5 Applications

Le système Arduino nous permet de réaliser beaucoup de choses, qui ont des applications dans divers domaines. Le champ d'utilisation d'Arduino est énorme. On peut mentionner et donner quelques exemples qui sont :

- Contrôler la vitesse d'un moteur.
- Contrôler les appareils domestiques.
- Fabriquer votre propre robot.
- Faire un jeu de lumière.
- Communiquer avec l'ordinateur.
- Télécommander un appareil mobile (modélisme)...

III.6 Logiciel de programmation l'IDE d'Arduino

Arduino IDE (Integrated Development Environment). Le logiciel est gratuit et open source dont la simplicité d'utilisation est remarquable. Ce logiciel va nous permettre de programmer les implémentations de la commande directement sur la carte, et son interface se compose d'un ensemble d'outils développés qui rend les cartes Arduino facilement programmables.

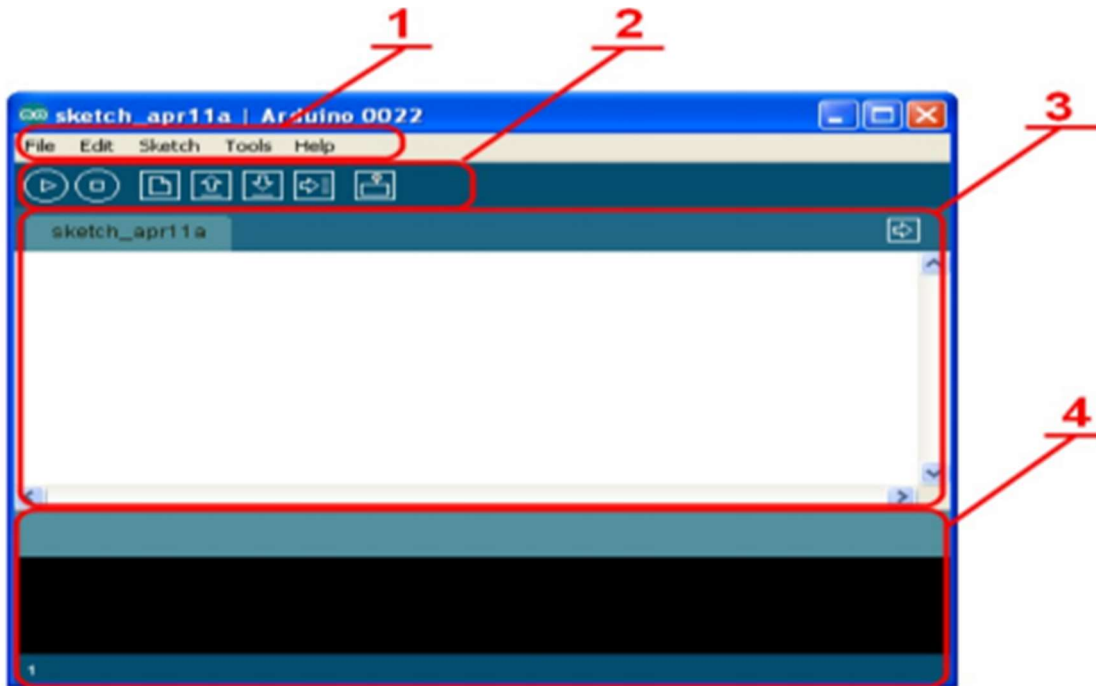


Figure III.11 : Présentation IDE Arduino

Le cadre numéro 1 : Ce sont les options de configuration du logiciel.

Le cadre numéro 2 : Il contient les boutons nécessaires pour la programmation des cartes.

Le cadre numéro 3 : C'est le bloc qui contiendra le programme créé.

Le cadre numéro 4 : Il aide à corriger en signalant les erreurs commises dans le programme. C'est le débogueur.

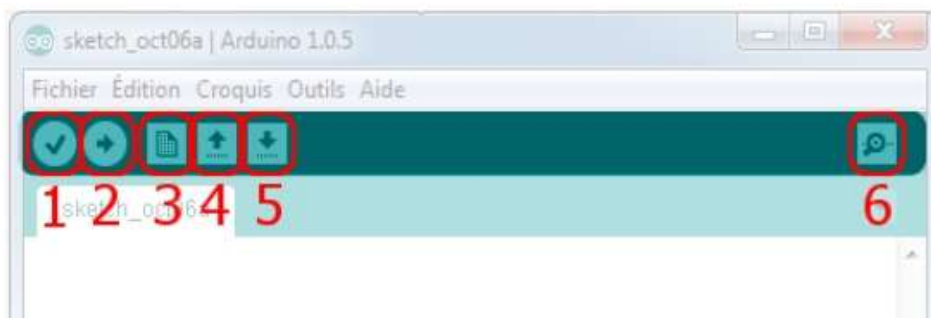


Figure III.12: La barre d'outils de l'IDE d'Arduino

La barre d'outils de l'IDE d'Arduino comporte six boutons, encadrés en rouge et numérotés dans la figure précédente.

Bouton 1 : Ce bouton permet de vérifier le programme, il actionne un module qui cherche les erreurs dans le programme écrit.

Bouton 2 : Charge (téléverse) le programme dans la carte Arduino sélectionné dans (outils).

Bouton 3 : Il permet de créer un nouveau fichier.

Bouton 4 : Il permet d'ouvrir un fichier déjà enregistré, ou présent dans la liste des exemples fournie par Arduino.

Bouton 5 : C'est le bouton de sauvegarde, il permet d'enregistrer le fichier ou d'enregistrer des modifications apportées à ce dernier.

Bouton 6 : Ce bouton ouvre le moniteur série, qui permet de communiquer avec la carte Arduino à travers la liaison série.

III.7 La structure d'un programme

Un programme Arduino comporte trois parties illustrées dans la figure Suivants :

```

programmerArduinoExemple | Arduino 0022
File Edit Sketch Tools Help
programmerArduinoExemple

1 int brocheCapteur = A0; // selection de la broche sur laquelle est connectée le capteur
  int brocheLED = 13; // selection de la broche sur laquelle est connectée la LED
  int valeurCapteur = 0; // variable stockant la valeur du signal reçu du capteur

2 void setup() {
  // broche de la LED configurée en sortie
  pinMode(ledPin, OUTPUT);
}

3 void loop() {
  // lecture du signal du capteur
  valeurCapteur = analogRead(brocheCapteur);
  // allume la LED
  digitalWrite(brocheLED, HIGH);
  // delai de "valeurCapteur" millisecondes
  delay(valeurCapteur);
  // éteint la LED
  digitalWrite(brocheLED, LOW);
  // delai de "valeurCapteur" millisecondes
  delay(valeurCapteur);
}
    
```

Figure III.13: Structure d'un programme

1. La partie déclaration des variables (optionnelle)
2. La partie initialisation et configuration des entrées/sorties : la fonction setup ()
3. La partie principale qui s'exécute en boucle : la fonction Loop ()

Dans chaque partie d'un programme sont utilisées différentes instructions issues de la syntaxe du langage Arduino.

III.8 Le langage d'Arduino

Le programme est lié à une série d'instruction élémentaires sous forme de texte, donc la carte lit après exécute les instructions suivant un ordre un après l'autre.

- Un ordinateur.
- Une carte Arduino.
- Programme lié à l'Arduino.

Syntaxe du langage : C et C++ qui est le suivant [19] :

Code minimal : son rôle de divisé le programme en deux parties La fonction :

Setup () : est considéré comme fonction d'initialisation on l'appelle une seul fois au début du Programme.

Loop () : c'est pour écrire le cœur du programme. Elle est appelée en permanence : en boucle infinie.

Les instructions : sont des lignes contenant des codes, exemple : « Fait ceci » « Fait cela »

Les points-virgules (;) : pour finir les instructions

Les accolades [] : sont utilisées pour les fonctions ; les boucles. Elles sont obligatoires.

Les commentaires : // cette ligne a un commentaire. /* pour plusieurs lignes

Les variables : Les variables booléennes peuvent prendre deux valeurs soit vraie ou faux donc, si une variable vaut (0) on la considère comme variable booléennes fausse et si une variable prend n'importe quelle valeur différente de zéro on la considère comme variable booléenne vraie.

- **char** (variable 'caractère')
- **Int** (variable 'nombre entier')
- **long** (variable 'nombre entier de très grande taille')
- **string** (variable 'chaine de caractères')
- **Array** (tableau de variables).

Parmi les types d'Arduino, nous allons choisir pour notre travail et notre réalisation une carte Arduino Méga. L'intérêt principal de cette carte est de faciliter la mise en œuvre d'une commande qui sera détaillée par la suite.

III.9 Présentation générale de l'Arduino Méga 2560

III.9.1 Caractéristiques de l'Arduino Méga 2560

Cette carte dispose de [20] :

- 54 broches numériques d'entrées/sorties (dont 14 peuvent être utilisées en sorties PWM (largeur d'impulsion modulée) ;
- 16 entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques).
- 4 UART (port série matériel) ;
- Un quartz 16Mhz ;
- Une connexion USB ;
- Un connecteur d'alimentation jack ;
- Un connecteur ICSP (programmation "in-circuit") ;
- Un bouton de réinitialisation (reset) ;

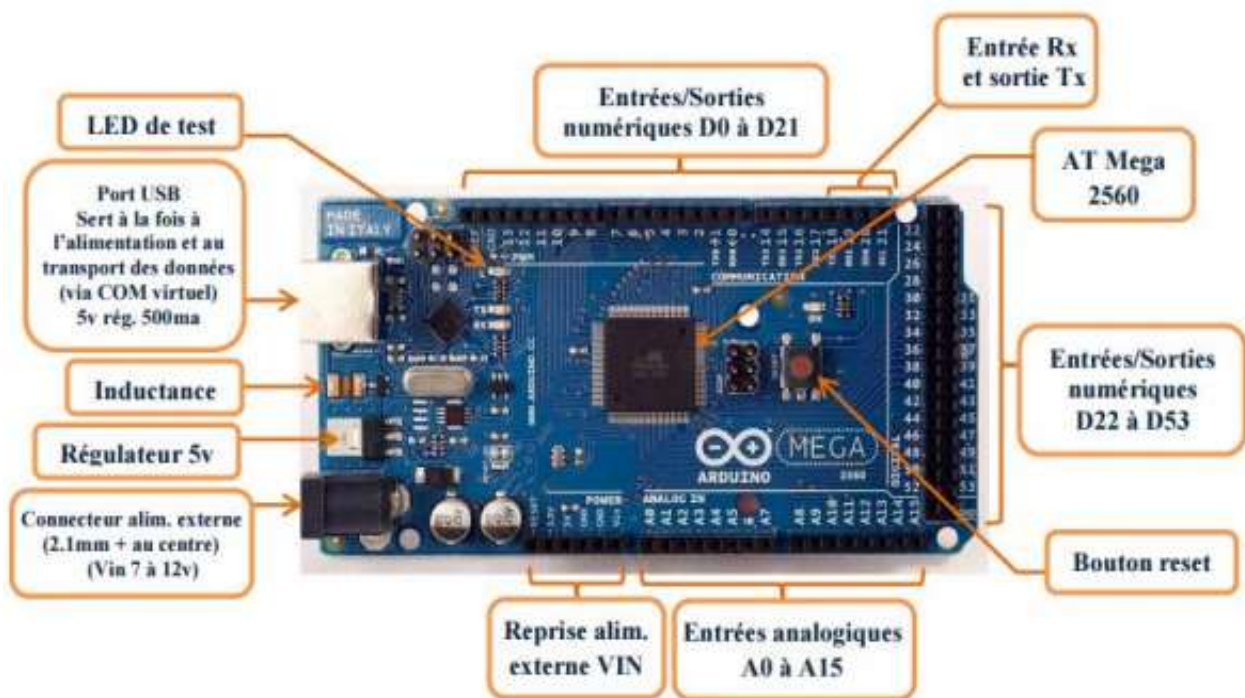


Figure III.14: Description de la carte Arduino MEGA 2560.

Elle contient tout ce qui est nécessaire pour le fonctionnement du microcontrôleur ; pour pouvoir l'utiliser et se lancer, il suffit simplement de la connecter à un ordinateur à l'aide d'un câble USB (ou de l'alimenter avec un adaptateur secteur ou une pile, mais ceci n'est pas indispensable, l'alimentation étant fournie par le port USB).

III.9.2 Caractéristique technique de la carte Arduino Méga 2560

Un module Arduino est généralement construit autour d'un microcontrôleur ATMEL AVR, et de composants complémentaires qui facilitent la programmation et l'interfaçage avec d'autres circuits. Chaque module possède au moins un régulateur linéaire 5V et un oscillateur à quartz 16 MHz (ou un résonateur céramique dans certains modèles). Le microcontrôleur est préprogrammé avec un bootloader de façon à ce qu'un programmeur dédié ne soit pas nécessaire.

III.9.3 Synthèse des caractéristiques

Généralement tout module électronique qui possède une interface de programmation est basé toujours dans sa construction sur un circuit programmable ou plus [20].

Tableau III.1: Constitution de la carte Arduino Méga 2560.

Microcontrôleur	ATMEGA2560
Tension de fonctionnement	5V
Tension d'alimentation	7 à 12V
Broches E/S numérique	54 (dont 14 disposent de sortie PWM)
Broches d'entrées analogiques	16
Vitesse d'horloge	16 MHz
Mémoire programme Flash	256KB dont 8 KB utilisés en bootloader
Mémoire SRAM	8 KB
Mémoire EEPROM	4 KB

III.9.3.1 Alimentation de la carte

La carte Arduino Méga 2560 peut être alimentée soit via la connexion USB (qui fournit 5V jusqu'à 500mA) ou à l'aide d'une alimentation externe. La source d'alimentation est sélectionnée

automatiquement par la carte. L'alimentation externe (non-USB) peut être soit un adaptateur secteur (pouvant fournir typiquement de 3V à 12V sous 500mA) ou des piles (ou des accus).

L'adaptateur secteur peut être connecté en branchant une prise 2.1mm positif au centre dans le connecteur jack de la carte. Les fils en provenance d'un bloc de piles ou d'accus peuvent être insérés dans les connecteurs des broches de la carte appelées GND (masse ou 0V) et Vin (Tension positive en entrée) du connecteur d'alimentation. La carte peut fonctionner avec une alimentation externe de 6 à 20 volts.

Cependant, si la carte est alimentée avec moins de 7V, la broche 5V pourrait fournir moins de 5V et la carte pourrait être instable. Si on utilise plus de 12V, le régulateur de tension de la carte pourrait chauffer et endommager la carte.

Les broches d'alimentation sont les suivantes :

- **VIN** : la tension d'entrée positive lorsque la carte Arduino est utilisée avec une source de tension externe (à distinguer du 5V de la connexion USB ou autre source 5V régulée).
- **5V** : la tension régulée utilisée pour faire fonctionner le microcontrôleur et les autres composants de la carte. Le 5V régulé fourni par cette broche peut donc provenir soit de la tension d'alimentation VIN via le régulateur de la carte, ou bien de la connexion USB (qui fournit du 5V régulé) ou de toute autre source d'alimentation régulée.
- **3,3V** : Une alimentation de 3.3V fournie par le circuit intégré FTDI (circuit intégré faisant l'adaptation du signal entre le port USB de votre ordinateur et le port série de l'ATmega) de la carte. L'intensité maximale disponible sur cette broche est de 50mA
- **GND** : broche de masse (0V). [20]

III.9.3.2 Les mémoires

L'ATmega 2560 à 256Ko de mémoire FLASH pour stocker le programme (dont 8Ko également utilisés par le bootloader), également 8 ko de mémoire SRAM (volatile) et 4Ko d'EEPROM (non volatile - mémoire qui peut être lue à l'aide de la librairie EEPROM) [20].

III.9.3.3 Entrées et sorties numériques

Chacune des 54 broches numériques de la carte Méga peut être utilisée soit comme une entrée numérique, soit comme une sortie numérique. Ces broches fonctionnent en 5V. Chaque broche peut fournir ou recevoir un maximum de 40mA d'intensité et dispose d'une résistance interne de "rappel au plus" (pull-up) (déconnectée par défaut) de 20-50 KOhms. Cette résistance interne s'active sur une broche en entrée à l'aide de l'instruction *digitalWrite* (broche, HIGH).

De plus, certaines broches ont des fonctions spécialisées :

- **Communication Série** : Port Série Serial : 0 (RX) et 1 (TX) ; Port Série Serial 1 : 19 (RX) and 18 (TX) ; Port Série Serial 2 : 17 (RX) and 16 (TX) ; Port Série Serial 3 : 15 (RX) and 14 (TX).
- **Interruptions Externes** : Broches 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), et 21 (interrupt 2). Ces broches peuvent être configurées pour déclencher une interruption sur une valeur basse, sur un front montant ou descendant, ou sur un changement de valeur.
- **Impulsion PWM** (largeur d'impulsion modulée) : Broches 0 à 13. Fournissent une impulsion PWM 8-bits à l'aide de l'instruction *analogWrite* ().
- **SPI** (Interface Série Périphérique) : Broches 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Ces broches supportent la communication SPI (Interface Série Périphérique).
- **I2C** : Broches 20 (SDA) et 21 (SCL). Supportent les communications de protocole I2C.
- **LED** : Broche 13. Il y a une LED incluse dans la carte connectée à la broche 13. [20]

III.9.3.4 Broches analogiques

La carte Mega2560 dispose de 16 entrées analogiques, chacune pouvant fournir une mesure d'une résolution de 10 bits (1024 niveaux soit de 0 à 1023) à l'aide de la très utile fonction *analogRead* () du langage Arduino. Par défaut, ces broches mesurent entre le 0V (valeur 0) et le 5V (valeur 1023). Les broches analogiques peuvent être utilisées en tant que broches numériques. [20]

III.9.3.5 Autres broches

Il y a deux autres broches disponibles sur la carte [20] :

- **AREF** : Tension de référence pour les entrées analogiques (si différent du 5V). Utilisée avec l'instruction *analogReference* ().
- **Reset** : Mettre cette broche au niveau BAS entraîne la réinitialisation du microcontrôleur. Comme un port de communication virtuel pour le logiciel sur l'ordinateur, La connexion série de l'Arduino est très pratique pour communiquer avec un PC.

III.10 Conclusion

Dans ce chapitre dédié à la présentation d'Arduino, Nous avons présenté les différents composants de la carte, son utilité spécialement pour notre projet ensuite on a vu comment utiliser les broches pour contrôler les composants électroniques. En conclue que c'est de l'électronique embarquée qui nous facilite la réalisation de beaucoup de projets

CHAPITRE IV

Simulation Et Conception

IV.1 Introduction

Depuis la plus haute antiquité, les hommes ont cherché un moyen de favoriser le déplacement vertical des charges. Qui vont dans la future définit par l'ascenseur.

L'ascenseur devient plus intelligent pour anticiper et mieux gérer le trafic, plus confortable et plus communicant pour limiter le stress et permettre l'assistance en toute circonstance. Il est aussi plus respectueux de l'environnement et se trouve aux cours du concept d'accessibilité pour tous.

Nous avons choisi pour notre travail la réalisation d'un automate programmable d'un ascenseur. Ce chapitre portera sur deux parties

➤ **Partie présentation**

Une approche des différents outils utilisés pour la réalisation de notre prototype en justifiant le choix des composants, ainsi le logiciel utilisé « ARDUINO ».

➤ **Partie réalisation**

Une réalisation d'une miniature maquette faite pour contrôler notre système.

IV.2 Exemple

Dans le but d'élaboration d'un système d'ascenseur, il est nécessaire de connaître les bases pour créer un système d'automate programmable fiable. Pour ce faire nous avons effectué trois exemplaires afin de mieux maîtriser nos logiciels.

Chaque exemplaire est réalisé avec le grafcet, ensuite converti en 1er lieu en langage Ladder et en second lieu en Arduino. Ce dernier langage est simulé au final en utilisant Proteus 8

On note que nous avons réalisé les grafkets à l'aide de logiciel automgen

IV.2.1 Exemple 1 : Exemple Allumage des LED Avec la Divergence

IV.2.1.1 Grafcet en Divergence appeler aussi Séquences multiples exclusives

C'est Lorsque, à partir d'une étape, on peut effectuer un choix entre plusieurs séquences possibles conditionnées par plusieurs réceptivités exclusives, les différentes séquences possibles sont installées sous un trait horizontal qui représente l'élargissement de la sortie de l'étape, et se retrouvent par un trait analogue représentant l'entrée de l'étape à nouveau commune. D'une

façon analogue au double trait vu ci-dessus, on parle de « divergence » et de « convergence en OU » pour cette représentation.

Dans notre travail nous avons rédigé ce grafcet pour un système qui alimente les 4 LED avec (OU)

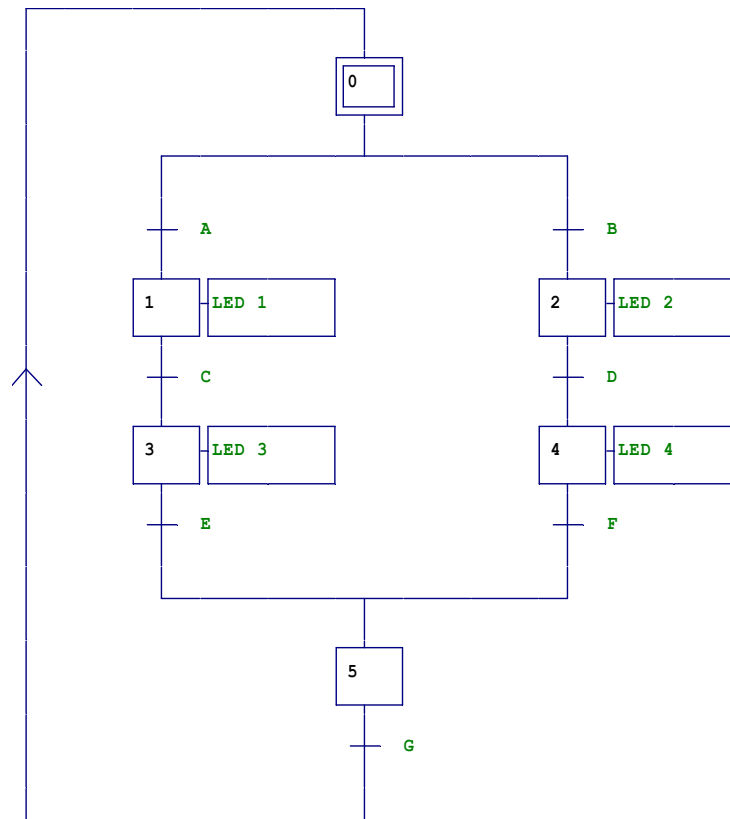


Figure III.15: illustration d'un grafcet d'allumage des LED Avec la Divergence

Les Equations de ce système Sont Comme Suit :

$$\begin{aligned}
 y_0^+ &= G * y_5 + y_0 * \bar{y}_1 * \bar{y}_2 \\
 y_1^+ &= A * y_0 + y_1 * y_1 * \bar{y}_3 \\
 y_2^+ &= B * y_0 + y_2 * y_2 * \bar{y}_4 \\
 y_3^+ &= C * y_1 + y_3 * y_2 * \bar{y}_5 \\
 y_4^+ &= D * y_2 + y_4 * y_2 * \bar{y}_5 \\
 y_5^+ &= E * y_3 + F * y_4 + y_5 * \bar{y}_0
 \end{aligned}$$

IV.2.1.2 Ladder

Après avoir extraire nos équations, nous avons les Convertis en langage Ladder :

Le (RINIT) est un relais interne.

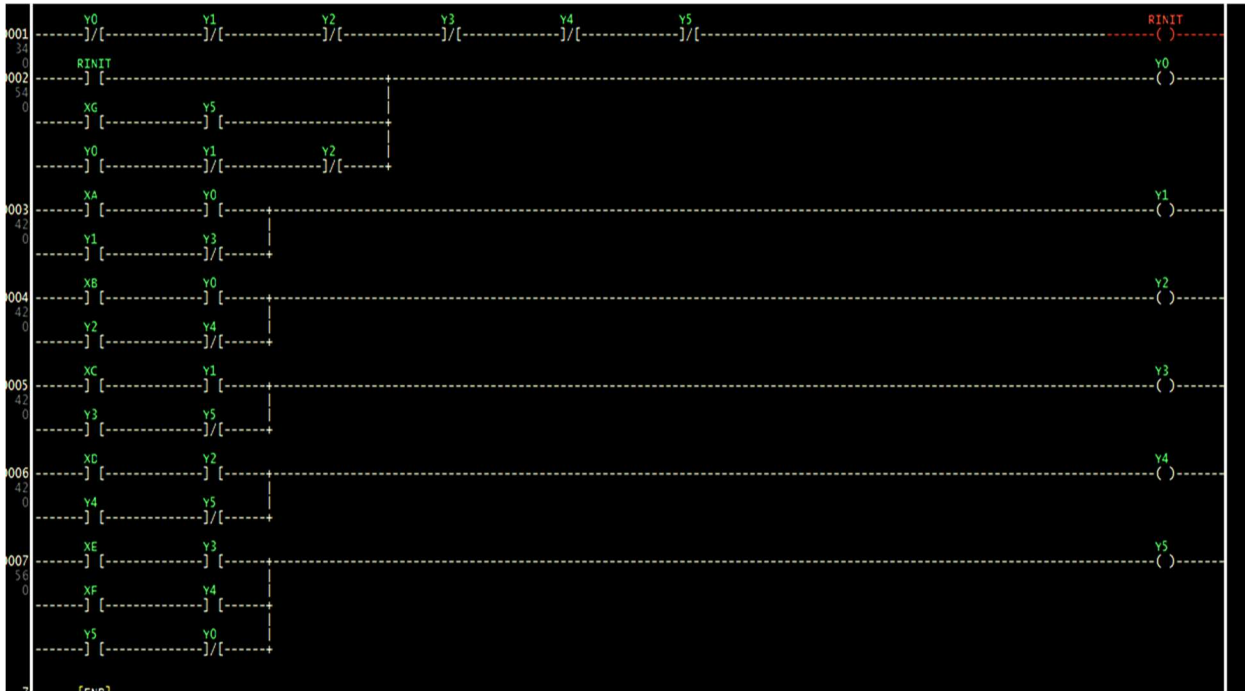


Figure III.16 : Ladder d'allumage Des LED Avec la Divergence

IV.2.1.3 Arduino

Après avoir lancé la simulation et vérifier les résultats non erronés qui assurent la fiabilité de notre raisonnement nous avons procéder notre système en code Arduino.

```

Fichier Edition Croquis Outils Aide
sketch_may06a

// constants won't change. They're used here to set pin numbers:
const int Ain = 7; // the number of the pushbutton pin
const int Bin = 6;
const int Cin = 5;
const int Din = 4;
const int Ein = 3;
const int Fin = 2;
const int Gin = 1;

const int Y0out = 13;
const int Y1out = 12;
const int Y2out = 11;
const int Y3out = 10;
const int Y4out = 9;
const int Y5out = 8;

const int ledFin13 = 13; // the number of the LED pin
const int ledFin12 = 12; // the number of the LED pin
const int ledFin11 = 11; // the number of the LED pin
const int ledFin10 = 10; // the number of the LED pin
const int ledFin9 = 9; // the number of the LED pin
const int ledFin8 = 8; // the number of the LED pin

int i=0;

Compilation terminée.

Le croquis utilise 1570 octets (4%) de l'espace de stockage de programmes. Le maximum est de 32256 octets.
Les variables globales utilisent 37 octets (1%) de mémoire dynamique, ce qui laisse 2011 octets pour les variables locales. Le maximum est de 2048 octets.
    
```

Figure III.17 : Interface De Logiciel Arduino avec le Code

Après que le programme est exécuté, nous avons le vérifier en cliquant sur **Compiler**, dans le cas où aucun problème n'est survenu et afficher, nous avons exporté les fichiers binaires (qui vont créer des fichiers supplémentaires dans notre dossier), puis nous allons passer à la dernière étape suivante.

IV.2.1.4 Proteus 8

La dernière étape est de simuler notre programme et examiner et confirmer par la suite notre raisonnement pour le visualiser.

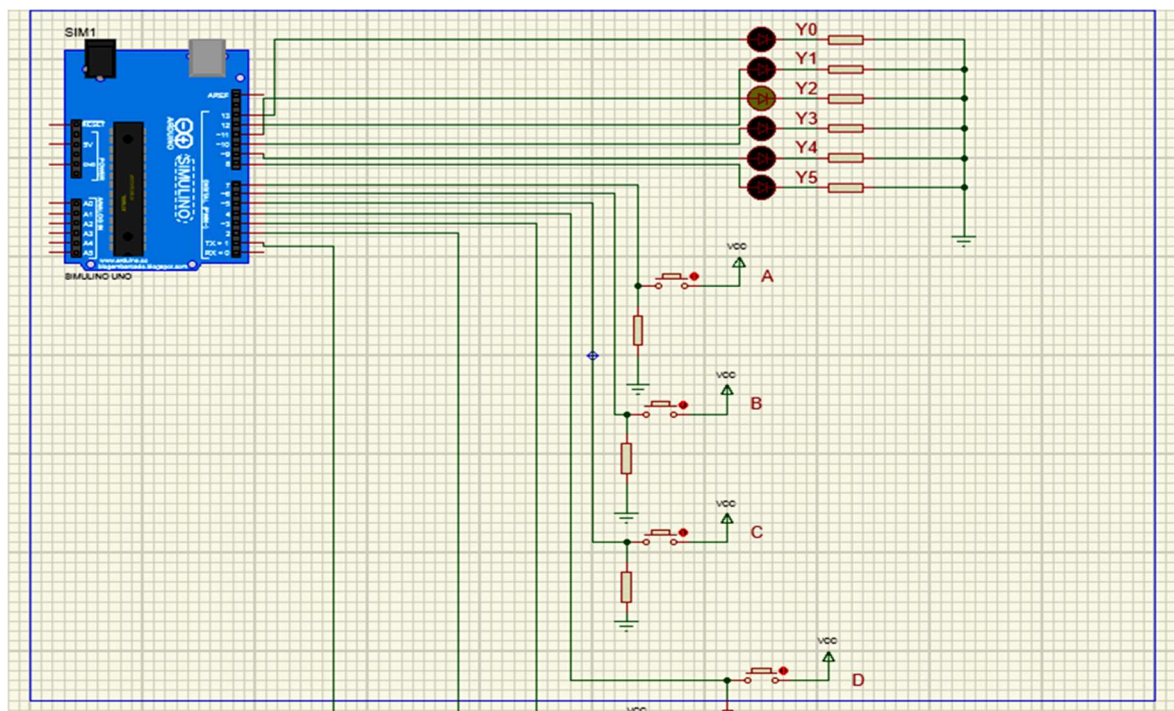


Figure III.18: Schéma Proteus 8 d'allumage Des LED Avec la Divergence

IV.2.2 Exemple Allumage des LED Avec la Convergence

IV.2.2.1 Grafset avec convergence appelé Séquences multiples simultanées :

C'est lorsque le franchissement d'une transition conduit à activer plusieurs étapes, les séquences issues de ces étapes sont dites « séquences simultanées ». Les séquences simultanées débutent toujours sur une *réceptivité unique* et se terminent toujours sur une *réceptivité unique*. En effet, les différentes séquences démarrent en même temps puis évoluent ensuite indépendamment les unes des autres. Ce n'est donc que lorsque toutes les étapes finales de ces séquences sont actives simultanément que l'évolution peut se poursuivre par le franchissement simultané d'une même transition.

Le début et la fin des séquences simultanées sont représentés par deux traits qui ne constituent pas des entités spécifiques du grafcet, mais qui doivent être compris comme l'élargissement de l'entrée ou de la sortie de la transition. (On parle parfois de « divergence » et de « convergence en ET » pour cette représentation.)

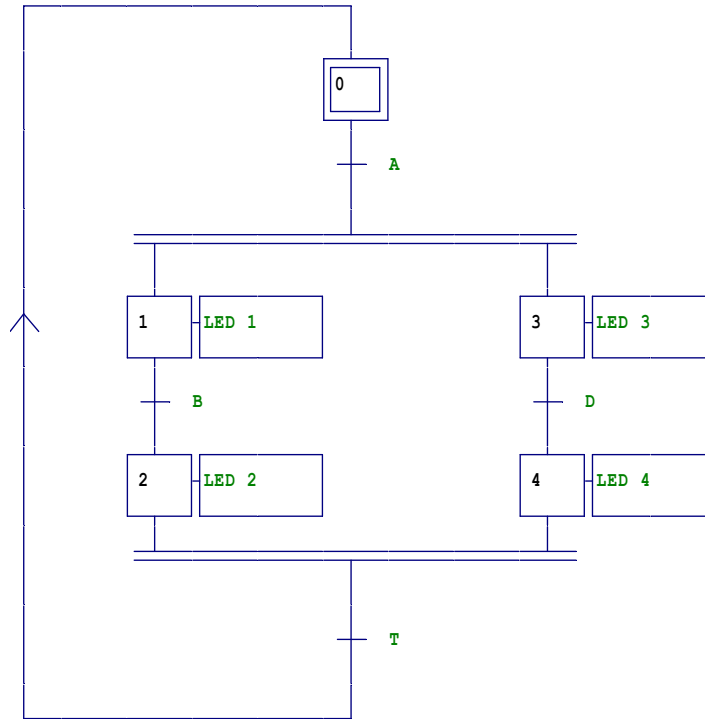


Figure III.19: illustration d'un grafcet d'allumage des LED Avec la Convergence

En analysant les données que nous avons et identifians les sorties et les transitions, les équations obtenues à partir de ce grafcet sont comme suit :

$$y_0^+ = t (y_2 * y_4) + y_0 * \bar{y}_1 * \bar{y}_3$$

$$y_1^+ = A * y_0 + y_1 * \bar{y}_2$$

$$y_2^+ = B * y_1 + y_2 * \bar{y}_0$$

$$y_3^+ = A * y_0 + y_3 * \bar{y}_4$$

$$y_4^+ = D * y_3 + y_4 * \bar{y}_0$$

IV.2.2.2 Ladder

En suivant le même enchainement pour l'exemple qui précède 1, nous avons fait le schéma suivant. Nous avons constaté par la suite que notre simulation a marché, ce qui confirme que nos résultats à partir des équations que nous avons testé et appliqué dans Ladder sont justes.

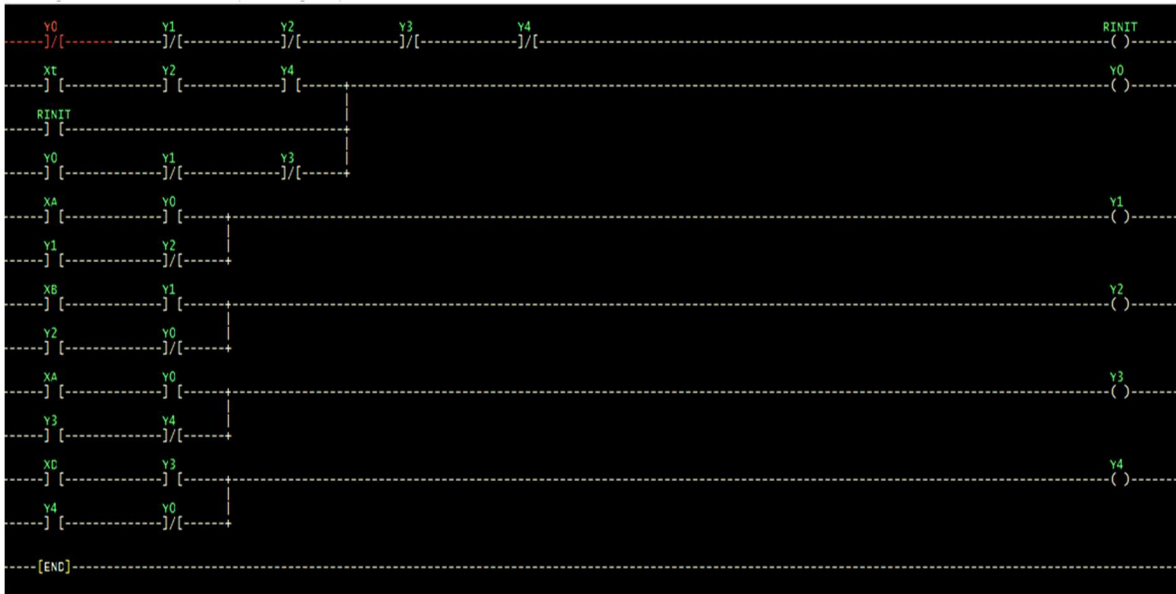


Figure III.20: Ladder d'allumage Des LED Avec la Convergence

Après avoir appliqué notre code dans l'Arduino, nous avons aperçu que la compilation est achevée sans erreur, aucune faille a été détecté. Les fichiers binaires sont exportés.

IV.2.2.3 Proteus 8

En transportant et incluant tous nos données au Proteus 8 (Résistances, LED, Arduino...), notre simulation est faite avec succès. Ce qui approuve notre raisonnement.

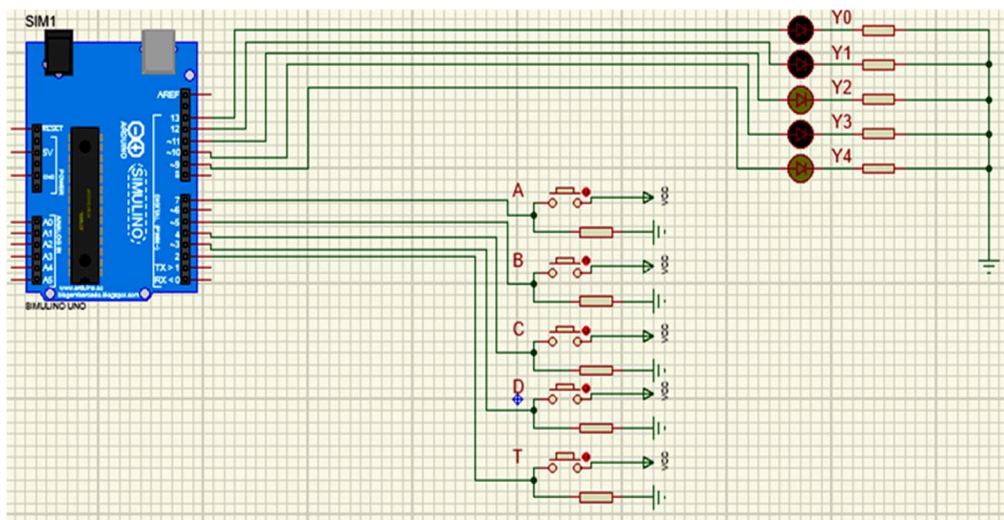


Figure III.21 : Schéma Proteus 8 d'allumage Des LED Avec Convergence

IV.2.3 Exemple Allumage des LED avec Le Timer

Est un relais utilisé pour des fonctions de temporisations et retardements.

IV.2.3.1 Grafcet

Pour cet exemple nous avons proposé cette illustration de grafcet suivante

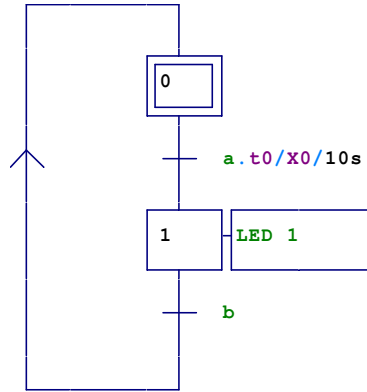


Figure III.22: Illustration Grafcet d’allumage Des LED Avec un Timer

Les équations acquis pour ce système et apportés sont comme suit :

$$y_0^+ = a * t * y_1 + y_0 * \overline{y_1}$$

$$y_1^+ = b * y_0 + y_1 * \overline{y_0}$$

IV.2.3.2 Ladder

Après avoir rapporté ces équations dans Ladder, nous avons réalisé ce schéma. Nous avons remarqué après 1 seconde que le programme Ladder a fonctionné correctement.

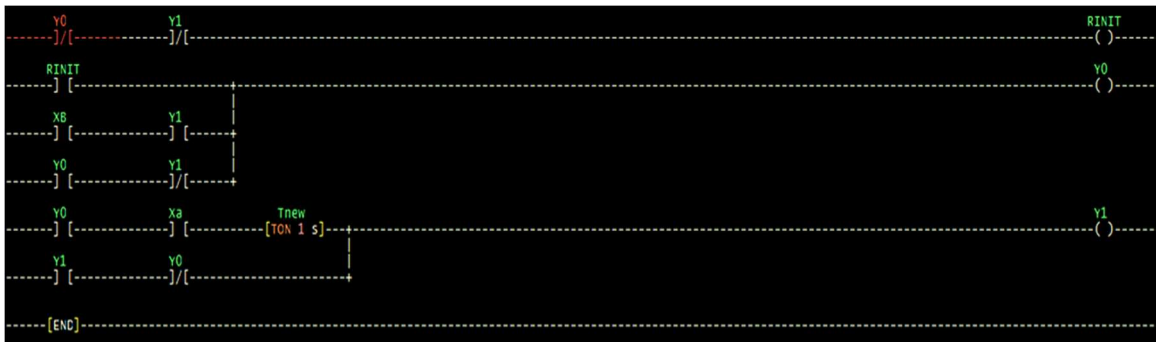


Figure III.23: Ladder d’allumage Des LED avec un Timer

IV.2.3.3 Arduino

Dans le code de ce système nous avons utilisé la fonction millis() au lieu la fonction delay ().

En conclusion la fonction millis() est généralement meilleure et fortement recommandé de l'utiliser. Cela nous permettra de programmer en utilisant des différents threads simultanément et elle est plus précise. La fonction delay() est recommandé pour être utilisé dans des programmes simples quand une action de blocage est nécessaire.

IV.2.3.4 Proteus 8

Voici une illustration de notre simulation après la vérification qui demeure juste.

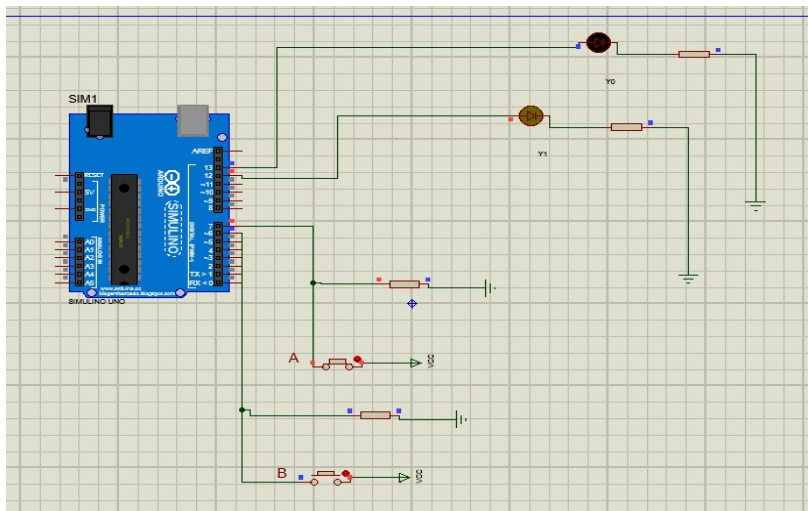


Figure III.24: Schéma Proteus 8 d'allumage Des LED Avec Le Timer

IV.3 Etude théorique :

IV.3.1 Partie Simulation

IV.3.1.1 Cahier des charges

Etant donné l'ascenseur initialement au repos, à n'importe quel étage, le système va s'initialiser au niveau "0" à chaque démarrage.

Un appel émanant d'un étage quelconque se réalise comme suit :

Si l'appel vient de l'étage où se trouve l'ascenseur, l'afficheur indique la présence de la cabine à l'étage demandé et la porte s'ouvre une fois en plus.

S'il vient d'un autre étage, alors l'ascenseur monte ou descend vers l'étage demandé, l'afficheur indique le niveau attend au même temps.

A chaque arrivé à l'étage désiré, la porte s'ouvre et la cabine poursuit sa course après un petit retard (delay)

S'il y a plusieurs appels, c'est le premier qui va être pris en considération.

Si l'ascenseur doit monter ou descendre de plus d'un étage, alors à l'arrivée de chaque étage intermédiaire, un test de demande d'appel est effectué. La cabine s'arrête pour laisser monter ou descendre les personnes et continuera de répondre à la première demande. Les priorités sont telles que le sens de la cabine est celui du premier appel. La demande d'utilisation de la cabine peut se faire au niveau 0 ou à chaque niveau après l'appui sur l'un des boutons poussoirs.

Remarque : pour notre projet les boutons d'appels et de demandes sont les mêmes.

IV.3.1.2 Les organigrammes

IV.3.1.2.1 Organigramme général de fonctionnement de l'ascenseur

Après avoir introduit les différents mécanismes de travail dans le système, ainsi que les fonctions partielles du système, nous atteignons un niveau l'ensemble des organigrammes mécanisme de fonctionnement du système. L'organigramme décrivant le fonctionnement de notre ascenseur est le suivant :

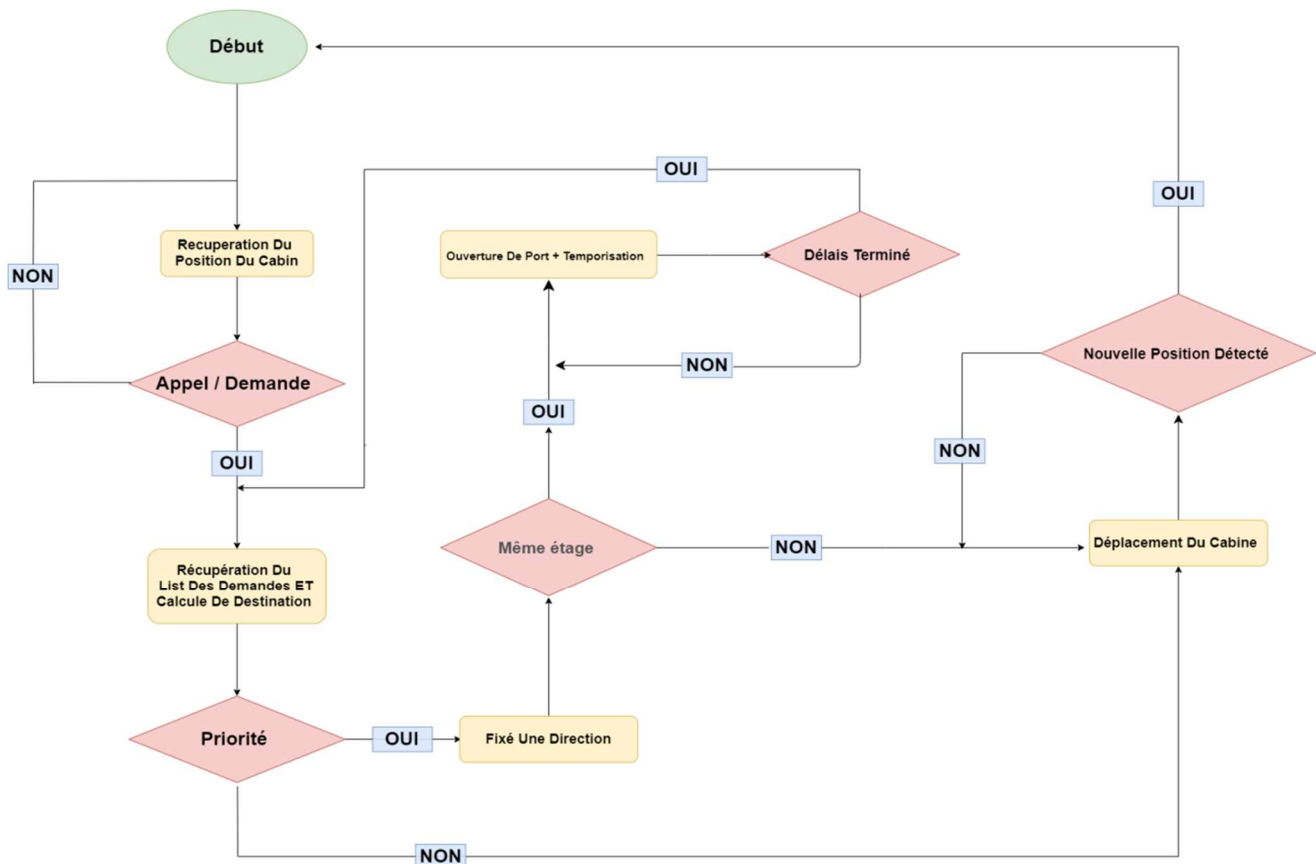


Figure III.25: Organigramme général de fonctionnement de l'ascenseur.

IV.3.1.2.1 Organigrammes de détection de la position et de contrôle des portes

La présence de l'ascenseur enfonce le contact du détecteur qui génère un niveau logique haut vers la carte de l'Arduino. Pour détecter l'arrivée de l'ascenseur à l'étage nous proposons des détecteurs de présence. L'ouverture et la fermeture de la porte sont temporisées. Une plus grande attention doit être portée à la sécurité des personnes entrante ou sortantes de l'ascenseur.

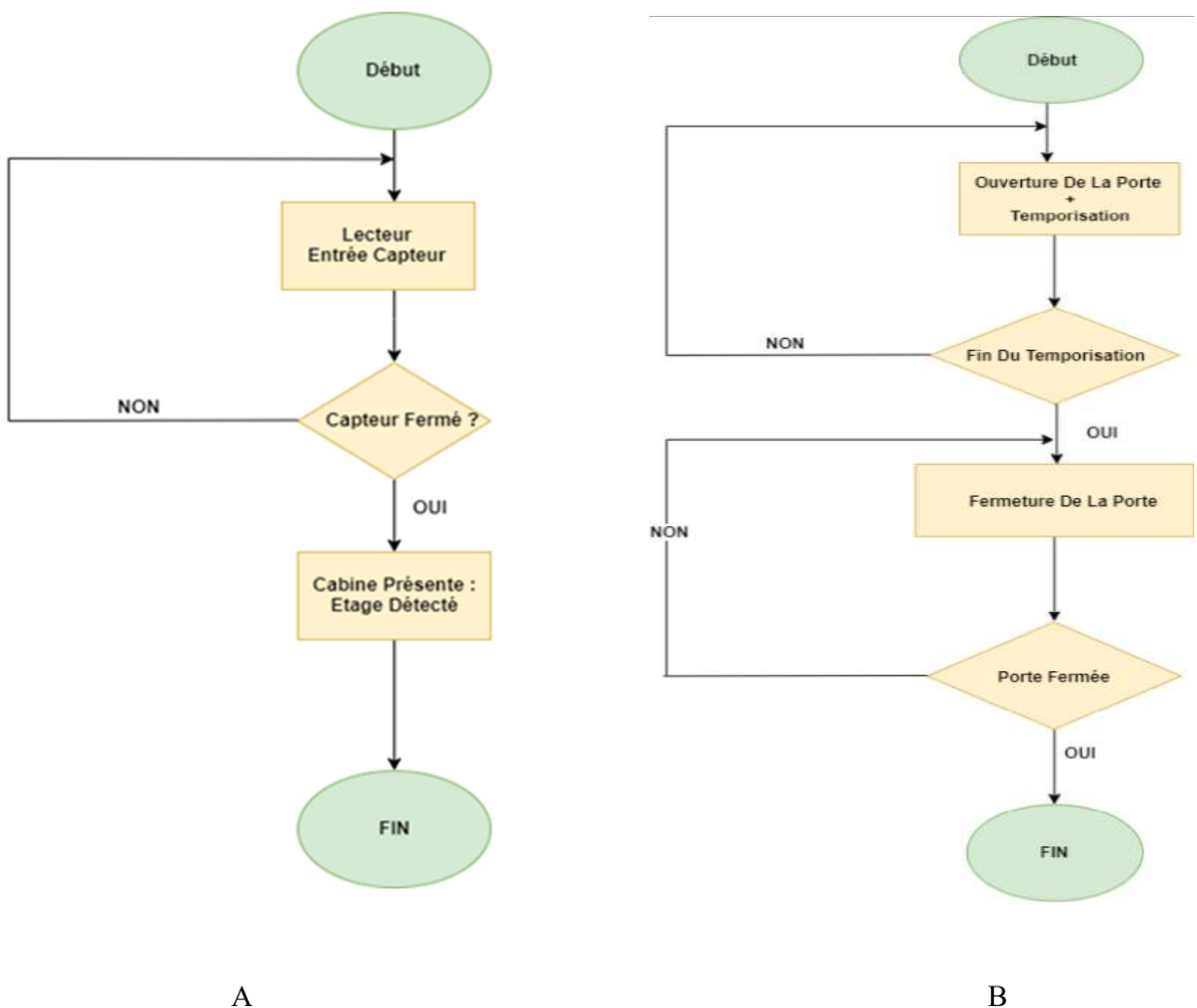


Figure III.26: Organigramme de détection de la position(A) et de contrôle des portes(B).

IV.3.1.2.2 Organigramme de mouvement de la cabine

Cet organigramme présente la partie " scanne " de l'ascenseur où notre programme examinera les différents demandes et appels subit par l'utilisateur.

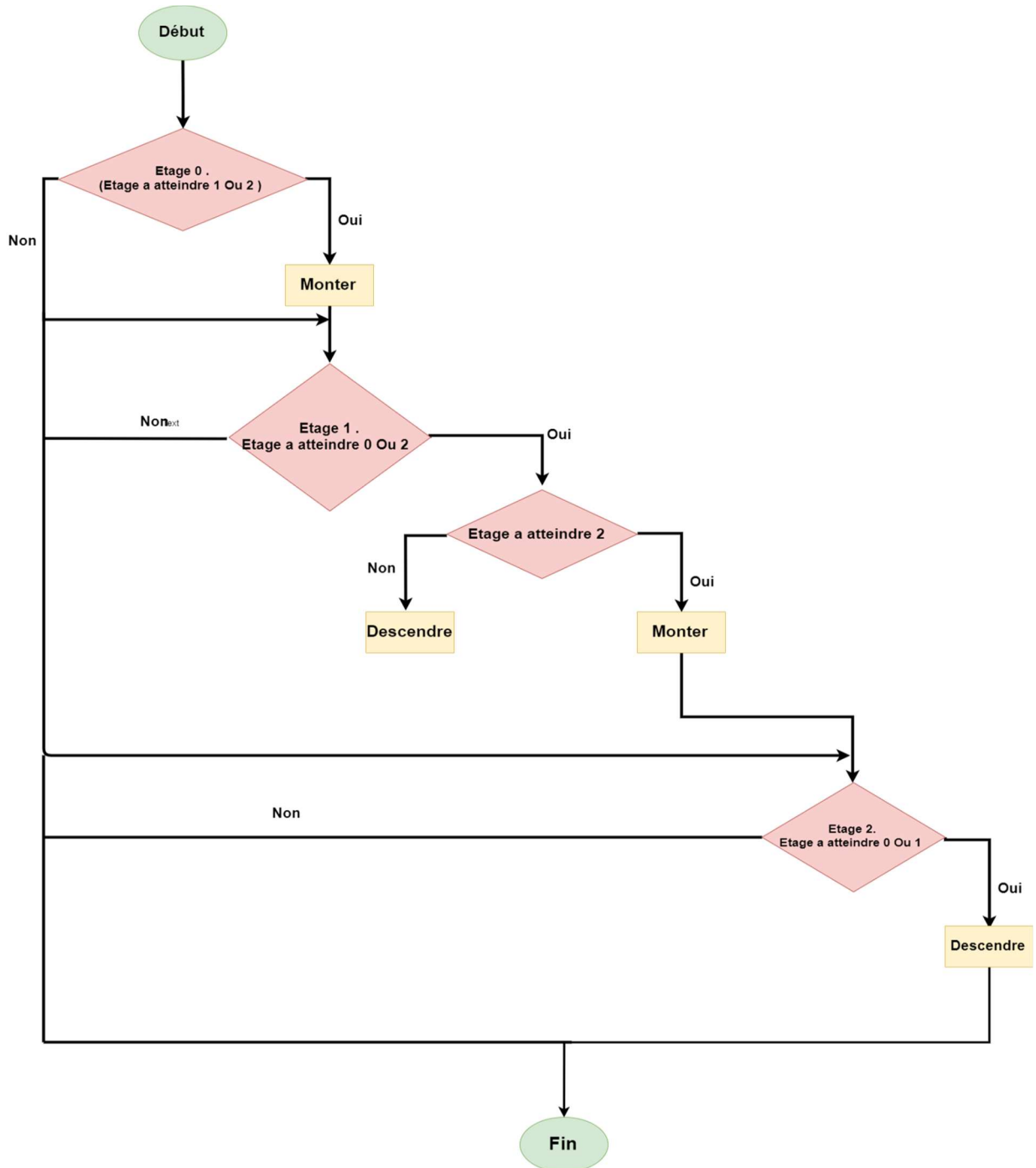


Figure III.27: Organigramme de mouvement de la cabine

IV.3.1.3 Grafcet

IV.3.1.3.1 Grafcet d'appel de la cabine

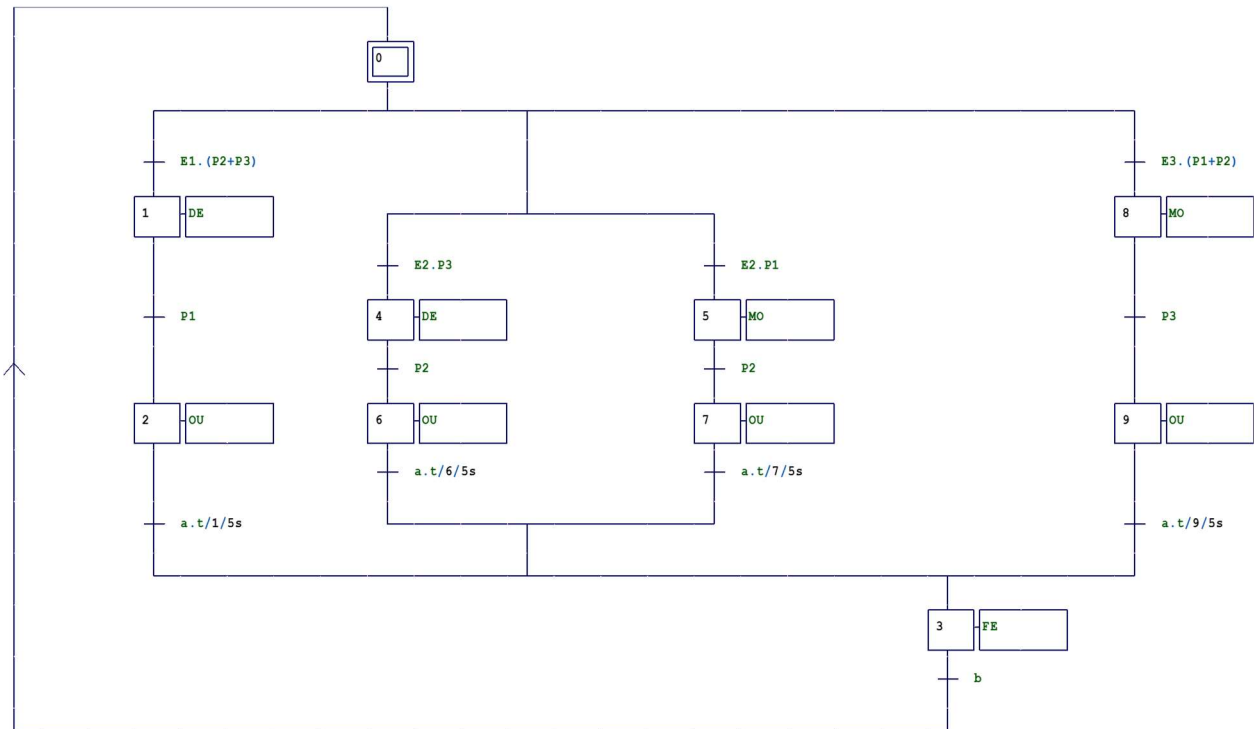


Figure III.28 : illustration d'un grafcet d'appel de la cabine.

IV.3.1.3.2 Grafcet Demande de l'étage

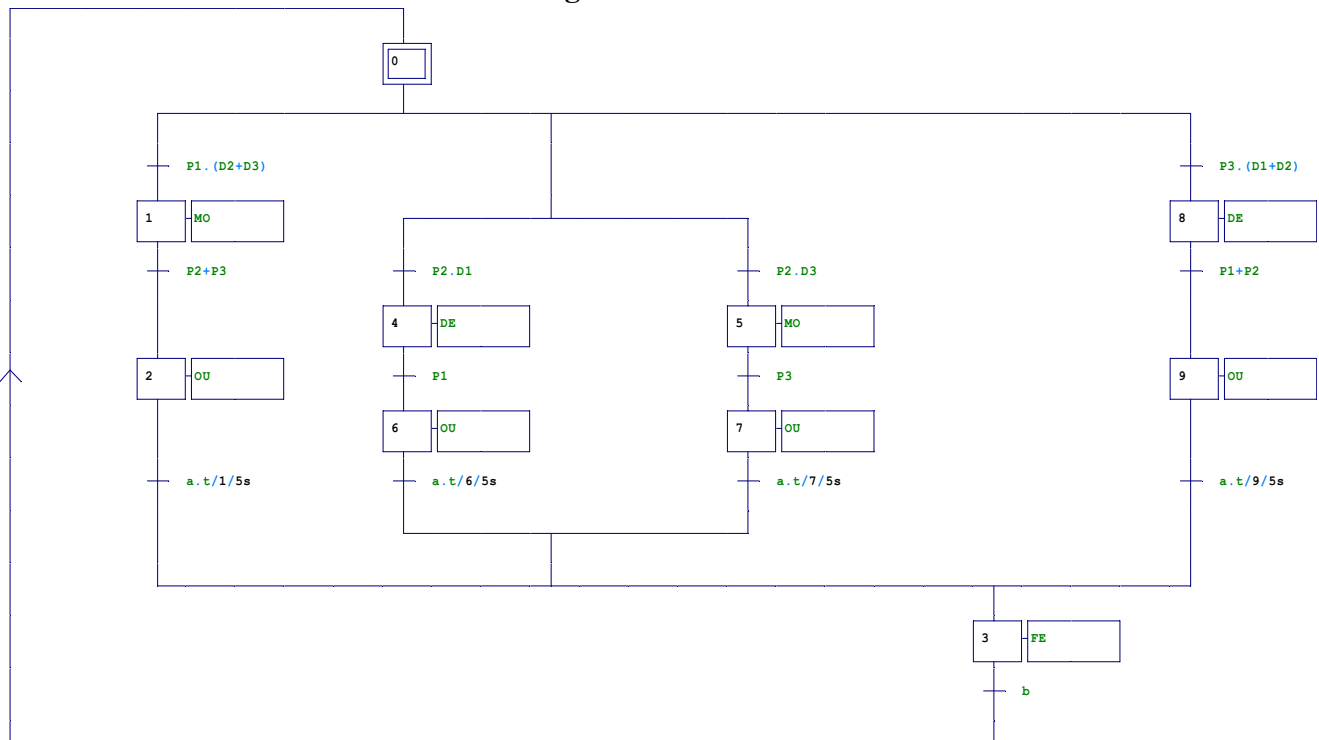


Figure III.29: illustration d'un grafcet demande de l'étage.

Les entrées et sortie de ces automates sont récapitulé dans le tableau suivant :

Nom	Discreption
E1	Bouton Poussoir Appel 1 ^{er} étage
E2	Bouton Poussoir Appel 2 ^{em} étage
E3	Bouton Poussoir Appel 3 ^{em} étage
D1	Bouton Poussoir Demande 1 ^{er} étage
D2	Bouton Poussoir Demande 2 ^{em} étage
D3	Bouton Poussoir Demande 3 ^{em} étage
P1	Position De La Cabine 1 ^{er} étage
P2	Position De La Cabine 2 ^{em} étage
P3	Position De La Cabine 3 ^{em} étage
Mo	Montée cabine
De	Descente cabine
Ou	Ouverture porte
Fe	Fermeture porte
A	Capteur Porte Ouverte
B	Capteur Porte Fermée

Tableau III-1: récapitulation des entrées et sortie des automates.

A partir des deux grafkets précédents, nous avons établi nos équations :

IV.3.1.4 Equations des appels

$$y_0^+ = y_3 * b + y_0 * \bar{y}_1 * \bar{y}_4 * \bar{y}_5 * \bar{y}_8$$

$$y_1^+ = y_0 * E1 * (P2 + P3) + y_1 * \bar{y}_2$$

$$y_2^+ = y_1 * P1 + y_2 * \bar{y}_3$$

$$y_3^+ = (y_2 * a * t) + (y_6 * a * t) + (y_7 * a * t) + (y_9 * a * t) + y_3 * \bar{y}_0$$

$$y_4^+ = y_0 * E2 * (P3) + y_4 * \bar{y}_6$$

$$y_5^+ = y_0 * b (P1) + y_5 * \bar{y}_7$$

$$y_6^+ = y_4 * P2 + y_6 * \bar{y}_3$$

$$y_7^+ = y_5 * P2 + y_7 * \overline{y_3}$$

$$y_8^+ = y_0 * E3(P1 + P2) + y_8 * \overline{y_9}$$

$$y_9^+ = y_8 * P3 + y_9$$

IV.3.1.5 Equations des Demandes :

$$y_0^+ = y_3 * b + y_0 * \overline{y_1} * \overline{y_4} * \overline{y_5} * \overline{y_8}$$

$$y_1^+ = y_0 * P1 * (D2 + D3) + y_1 * \overline{y_2}$$

$$y_2^+ = y_1 * (P1 + P2) + y_2 * \overline{y_3}$$

$$y_3^+ = (y_2 * a * t) + (y_6 * a * t) + (y_7 * a * t) + (y_9 * a * t) + y_3 * \overline{y_0}$$

$$y_4^+ = y_0 * P2 * (D1) + y_4 * \overline{y_6}$$

$$y_5^+ = y_0 * P2 * (D3) + y_5 * \overline{y_7}$$

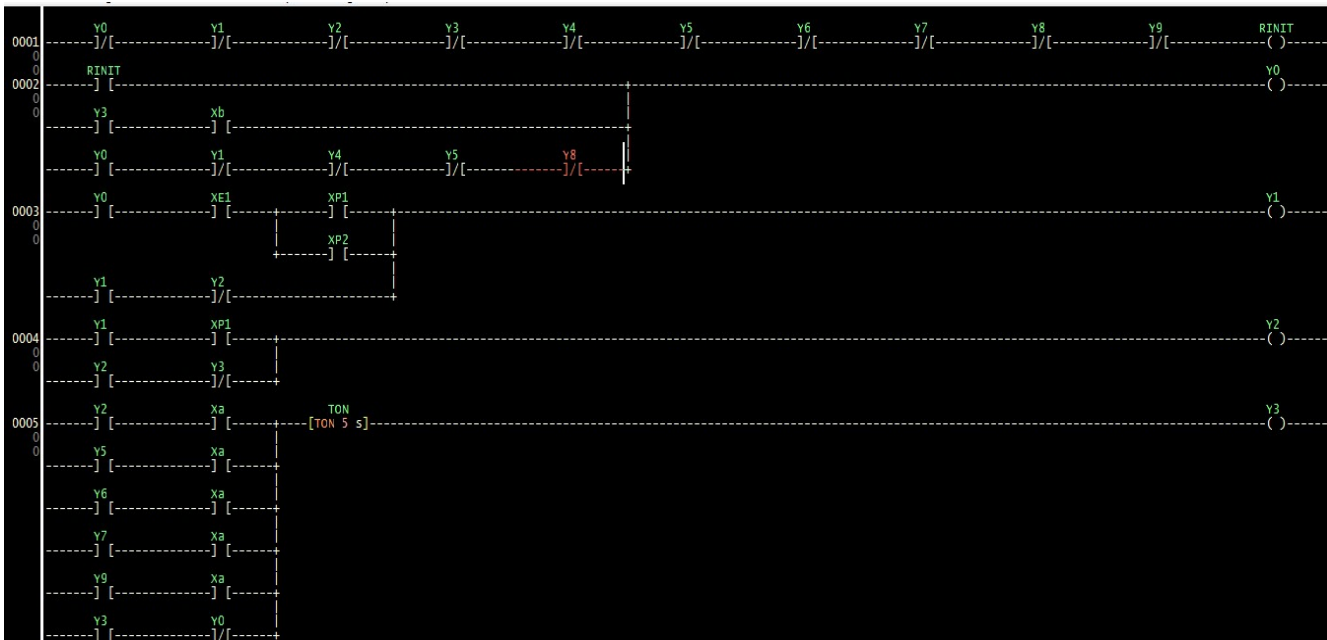
$$y_6^+ = y_4 * P1 + y_6 * \overline{y_3}$$

$$y_7^+ = y_5 * P3 + y_7 * \overline{y_3}$$

$$y_8^+ = y_0 * P3(D1 + D2) + y_8 * \overline{y_9}$$

$$y_9^+ = y_8 * (P1 + P2) + y_9 * \overline{y_3}$$

IV.3.1.6 Ladder d'appel



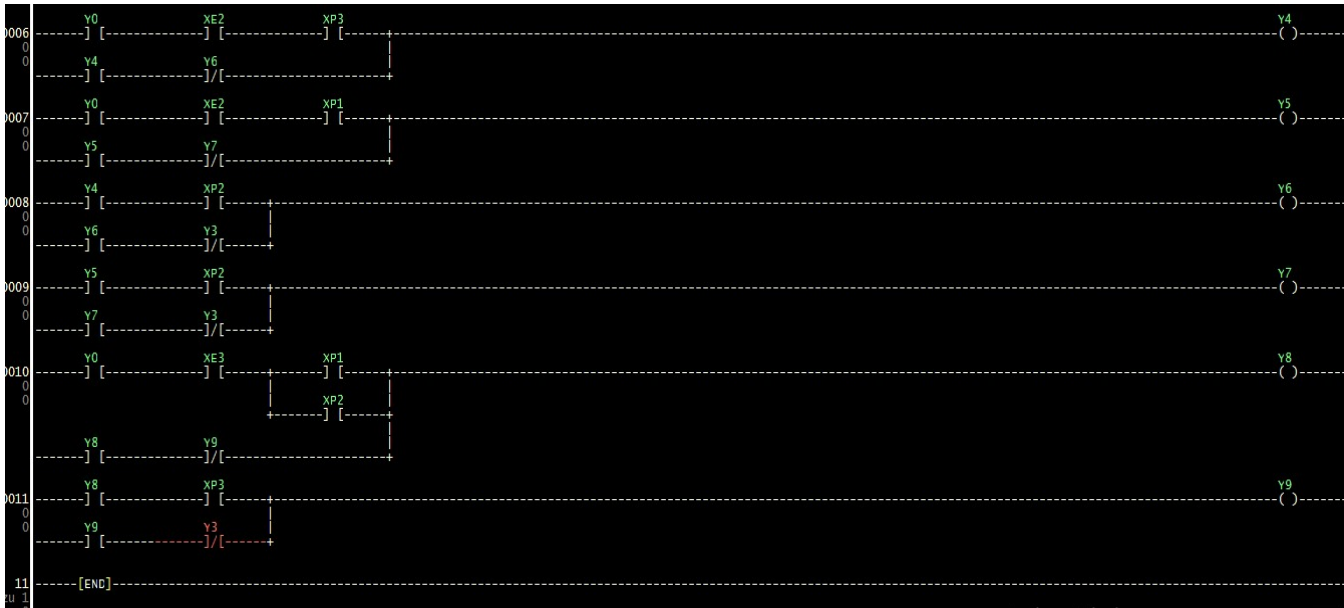
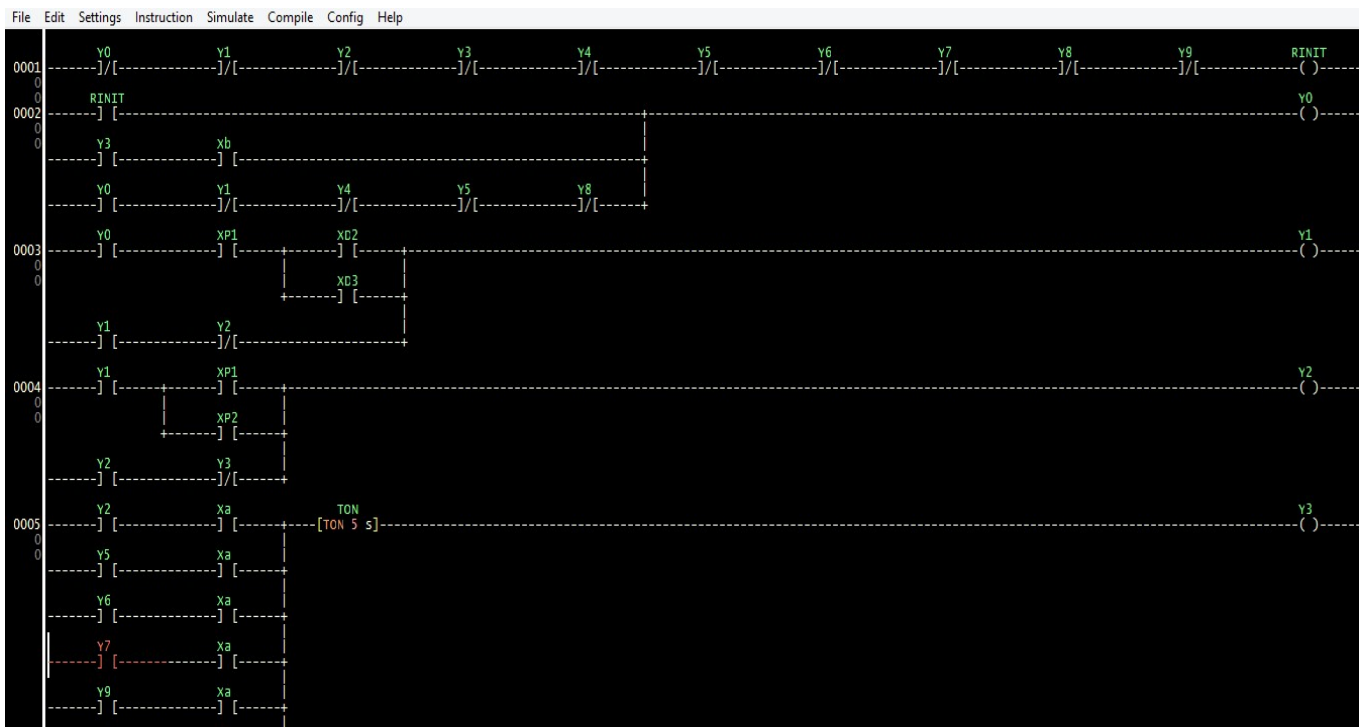


Figure III.30: Ladder appelle de l'ascenseur

IV.3.1.7 Ladder Demande



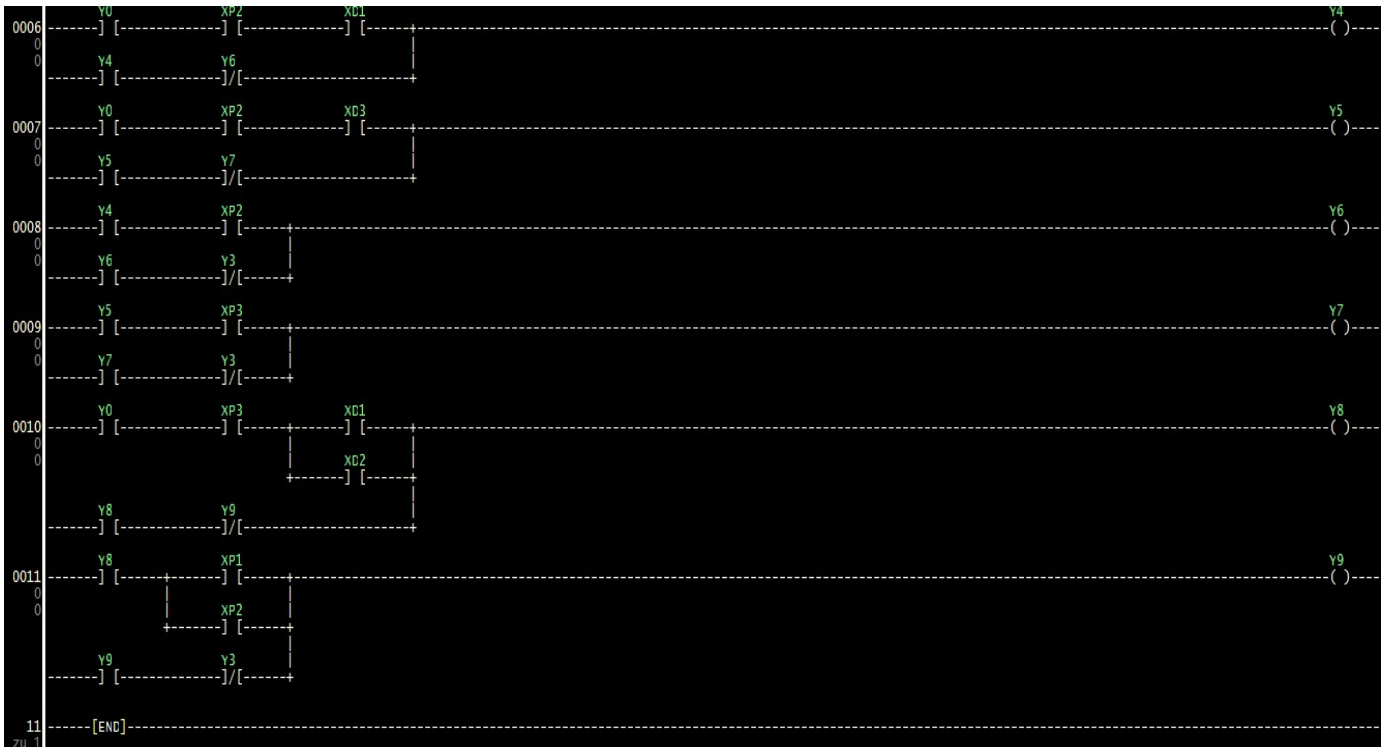


Figure III.31 : Ladder demande de l’ascenseur

Notre simulation a été effectuée avec succès, comme s’est affiché dans les figures au-dessus, nous pouvons passer à notre étape suivante.

IV.3.1.8 Arduino

L’environnement de programmation Arduino (IDE en anglais) est une application écrite en Java inspirée du langage Processing. La syntaxe des commandes d’Arduino est donnée dans la table des matières suivante. Chaque instruction est suivie de sa traduction entre-parenthèses Afin de compléter mon travail, dans cette étape code (voir annexe 1), nous avons suivi autre démarche que la méthode établie dans les exemples précédents dans la partie Arduino.

Nous avons recommandé cette méthode qui est plus explicite et claire dans ses notions et nomination des équations parce qu’elle nous a permis d’avoir plus de liberté d’expressions, et plus d’instructions.

C’est la méthode la plus adapté et facile dans cet exemple.

Commandes De Structure Du Programme	
Structure générale	<ul style="list-style-type: none"> •Voidsetup() (configuration, préparation) •Voidloop() (exécution)
Opérations de comparaison	<ul style="list-style-type: none"> •==(équivalent à) •!=(différent de) •>(supérieur à) •<(inférieur à) •<=(inférieur ou égal à) •>=(supérieur ou égal à)
Opérations booléennes	<ul style="list-style-type: none"> •&&(et) • (ou) •!(et pas)
Contrôle et conditions	<ul style="list-style-type: none"> •if (si..) •if...else(si...sinon...) •for (pour...) •switch case (dans le cas où...) •while(pendant que...)
Autres commandes	<ul style="list-style-type: none"> •//(commentaires simple ligne) •/**/(commentaires multi-lignes) •#define(donner une valeur à un nom)

Tableau III-2: Les commandes du programme Arduino.

Fonctions	
Entrées-sorties numériques	<ul style="list-style-type: none"> • <code>pinMode(broche, état)</code> (configuration des broches). • <code>digitalWrite(broche, état)</code>(écrire un état sur une broche numérique). • <code>digitalRead(broche)</code> (lire un état sur une broche numérique) • <code>digitalPinToInterrupt(broche,état)</code> (traduire la broche numérique en numéro d'interruption spécifique
Entrées analogique	<ul style="list-style-type: none"> • <code>intanalogRead(broche)</code>(lire la valeur d'une broche analogique). • <code>analogWrite(broche, valeur)</code>(écrire une valeur analogique sur les broches 9, 10 ou 11)
Interruptions Externs	<ul style="list-style-type: none"> • <code>attachInterrupt()</code>(déclencher l'interruption chaque fois que la broche change)
Gestion du Temps	<ul style="list-style-type: none"> • <code>millis()</code> (temps de fonctionnement du programme) • <code>delay(ms)</code>(attente, en millisecondes)
Communication	<ul style="list-style-type: none"> • <code>Serial.print()</code> (envoyer des informations d'Arduino Vers l'ordinateur) • <code>Serial.println()</code>(Imprime les données sur le port série sous forme de texte)
Autre	<ul style="list-style-type: none"> • <code>static</code>(créer des variables qui sont visibles que pour une seule fonction) • <code>#include</code>(utilisé pour inclure des bibliothèques extérieures dans votre croquis)

Tableau III-3: Les fonctions du programme Arduino.

Variables	
Variables	<ul style="list-style-type: none"> • <code>int</code>(variable 'nombre entier') • <code>long</code> (variable 'nombre entier de très grande taille') • <code>uint8_t</code> (entier non signé de longueur 8 bits) • <code>volatile</code> (modifier la manière dont le compilateur et le programme traitent la variable)
Niveaux logiques des connecteurs numériques	<ul style="list-style-type: none"> • <code>HIGH</code>(état 1) • <code>LOW</code> (état 0) • <code>INPUT</code> (configuré en entrée) • <code>OUTPUT</code> (configuré en sortie)

Tableau III-4: Les différentes variables du programme Arduino.

I.1.1.1 Proteus 8

Avant la réalisation pratique de nos cartes, nous avons réalisé la simulation de ces dernières en utilisant le logiciel Proteus 8. , Proteus est un outil qui permet de dessiner des schémas électroniques, de les simuler et de réaliser le circuit imprimé correspondant.

Cette suite logicielle est très connue dans le domaine de l'électronique. Elle est utilisée dans de nombreuses entreprises et organismes de formation. Outre la popularité de l'outil,

Proteus possède d'autres avantages :

- ✓ Pack contenant des logiciels faciles et rapides à comprendre et à utiliser ;
- ✓ Support technique performant ;
- ✓ Outil de création de prototype virtuel permettant de réduire les coûts matériel et logiciel lors de la conception d'un projet

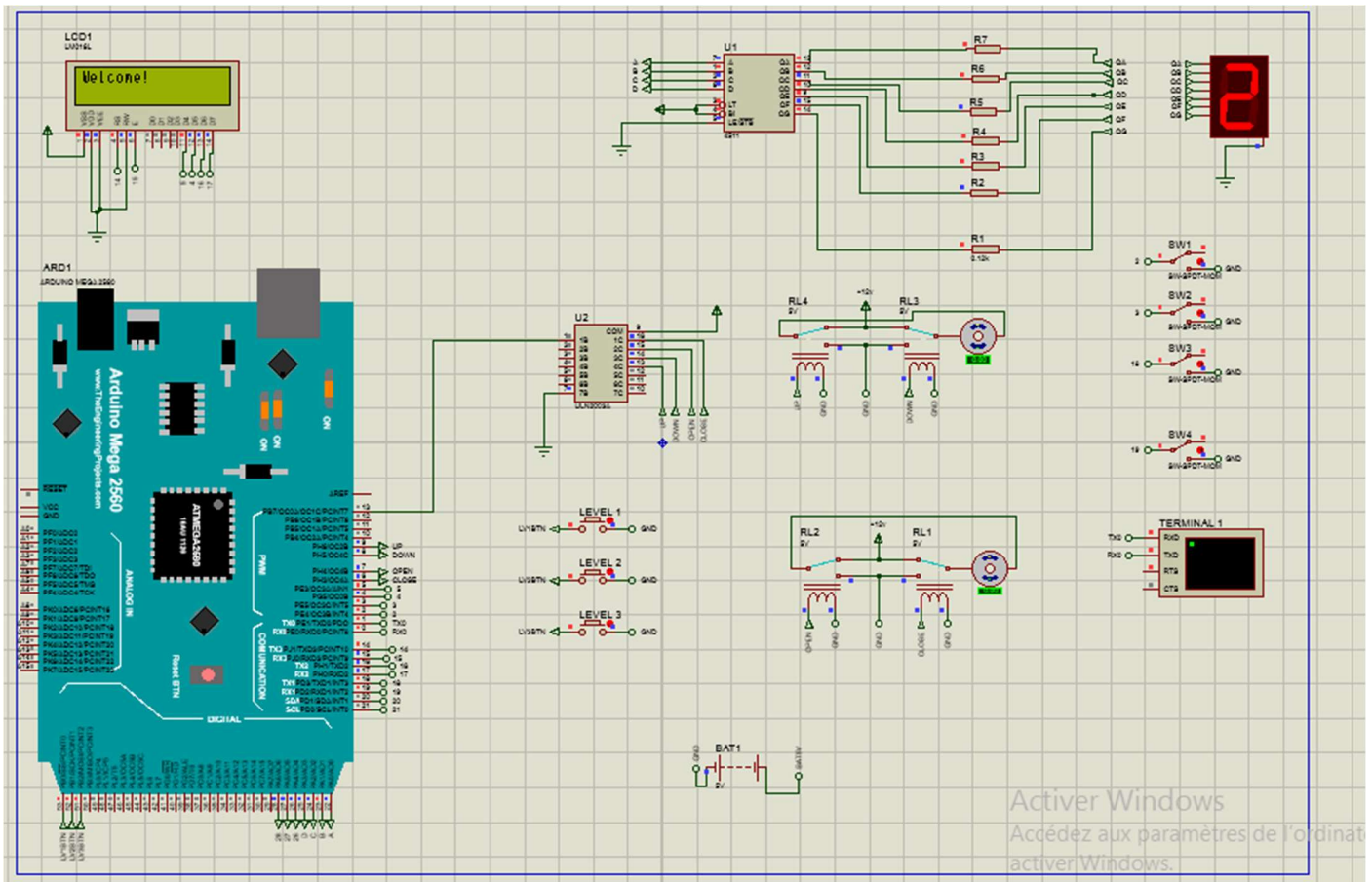


Figure III.32 : Simulation d'un automate programmable d'un ascenseur a 3 étages avec Proteus 8

IV.3.2 Partie électronique

IV.3.2.1 Principe de Fonctionnement

Dans notre projet, l'ascenseur est contrôlé par PLC. Cet ascenseur est exploité par utilisation d'un moteur à courant continu.

L'API donne la commande au relais. Le relais fait fonctionner le moteur à courant continu dans sens avant et sens arrière. Sur la base de cette opération de relais l'ascenseur monte et descend.

Le circuit BCD à sept segments est utilisé pour afficher la position actuelle d'ascenseur.

Le circuit de contrôle, contrôle également la fermeture et l'ouverture de la porte de l'ascenseur. C'est une commande par le relais électro-aimant lorsque les deux relais sont sous tension, l'ascenseur se déplace vers le ou vers le bas.

Si le relais est excité, le portail est ouvert et si l'autre relais est excité, la porte de l'ascenseur ferme.

IV.3.2.2 Les composants

IV.3.2.2.1 ULN2803

L'ULN2803 est un réseau de transistors Darlington haute tension et courant élevé. Il est principalement utilisé comme pilote de relais avec une capacité à gérer 8 relais à la fois. Il est livré avec une tension collecteur-émetteur d'environ 50V et une tension d'entrée résidant à 30V.

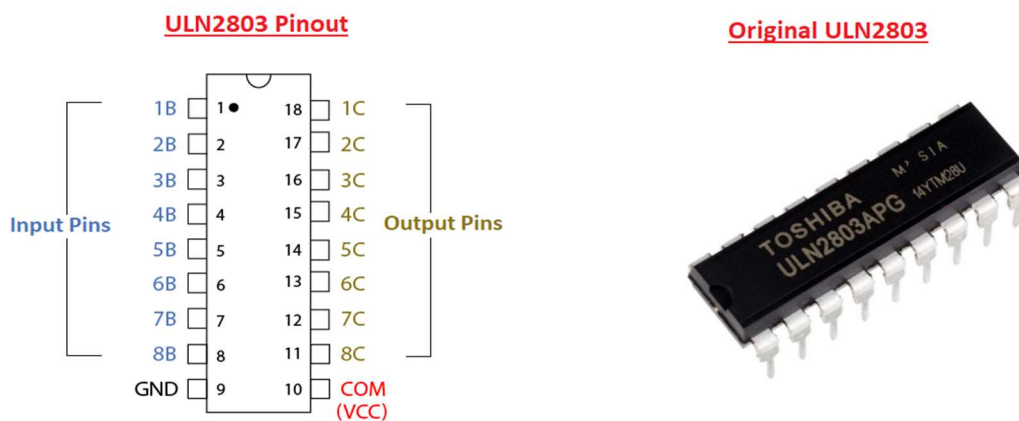


Figure III.33 : composé d'ULN2803.

Il est dédié à la commande des moteurs, lampes, relais... etc., en agissant comme un interrupteur reliant la masse de l'organe à commander avec l'une des sorties de L'ULN qui est un collecteur d'un des transistors qui sera à son tour relié à la masse quand la base de ce dernier est excitée par une tension de commande donc on aura un circuit fermer.

IV.3.2.2.2 Les Relais

Un relais est un interrupteur à commande électrique. Il utilise généralement un électro-aimant (bobine) pour faire fonctionner leur mécanisme de commutation mécanique interne (contacte). Lorsqu'un contact de relais est ouvert, cela mettra sous tension un circuit lorsque la bobine est activée. Il est doté d'un bobinage en guise d'organe de commande, la tension appliquée à ce bobinage va créer un contact, ce courant produisant un champ électromagnétique a l'entremet de la bobine, ce champ magnétique va être capable de faire déplacer un élément mécanique monté sur un axe mobile, qui va déplacée des contacts électroniques.

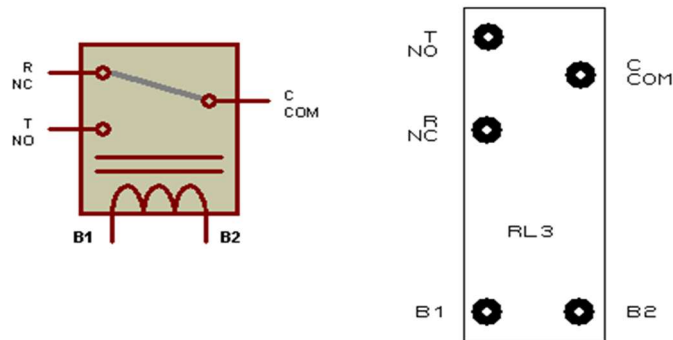


Figure III.34 : Schéma interne d'un relais.

IV.3.2.2.3 Branchement d'un bouton poussoir (entrée)

Un bouton (ou bouton poussoir) est un interrupteur simple qui permet de contrôler les capacités d'un processus. Le poussoir est de plus en plus utilisé, notamment pour les circuits va-et-vient qui nécessitent plus de deux points de commande. R : $10\text{ K}\Omega \pm 5\%$ - 1/8 W



Figure III.35 : Bouton poussoir (Droite : schéma avec câblage, Gauche : Pièce Original)

IV.3.2.2.4 Afficheur 7 segments

C'est un composant qui permet d'afficher les chiffres de 0 à 9. Ils sont encore utilisés actuellement malgré l'importante présence du LCD sur le marché de l'affichage électronique. Comme son nom l'indique, Il possède 7 segments, Ces afficheurs sont adaptés lorsqu'un large affichage numérique est requis comme pour les écrans dans les gares, les montres, calculatrices, ascenseurs ...etc.

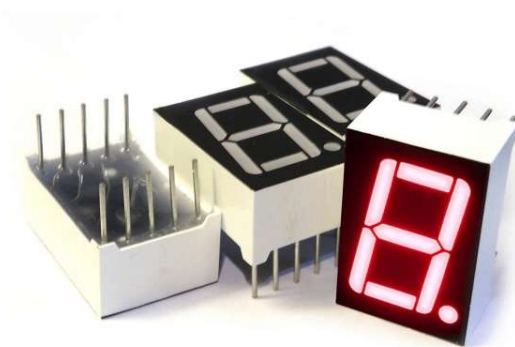


Figure III.36: image d'un afficheur à 7 segments.

Fonctionnement

Un segment est en fait une LED plate, et les 7 sont arrangées en forme de huit. Pour disposer les chiffres il faut allumer certains segments, et en laisser d'autres éteints. Ces segments pour être identifiés facilement sont associés à des lettres.

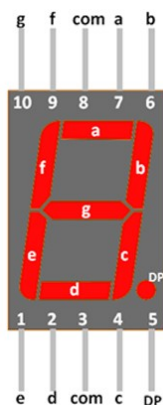


Figure III.37: Chiffrement d'un afficher 7 segments

Voici la signification des différentes broches et l'exemple de branchement d'un afficheur a anode commune :

1. LED de la cathode E ;
2. LED de la cathode D ;
3. Anode commune des LED ;
4. LED de la cathode C ;
5. Le point décimal ;
6. LED de la cathode B ;
7. LED de la cathode A ;
8. Anodes commune des LED ;
9. LED de la cathode F ;
10. LED de la cathode G

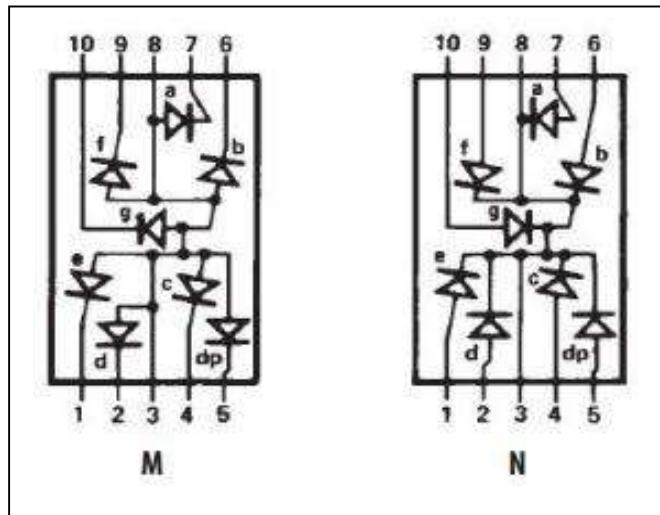


Figure III.38: Brochage standard d'un afficheur 7 segments CC et CA.

Si on met côte à côté le schéma d'agencement des segments et l'état des segments pour chaque chiffre, on obtient le tableau suivant :

Digit	G	F	E	D	C	B	A
0	OFF	ON	ON	ON	ON	ON	ON
1	OFF	OFF	OFF	OFF	ON	ON	OFF
2	ON	OFF	ON	ON	OFF	ON	ON
3	ON	OFF	OFF	ON	ON	ON	ON
4	ON	ON	OFF	OFF	ON	ON	OFF
5	ON	ON	OFF	ON	ON	OFF	ON
6	ON	ON	ON	ON	ON	OFF	ON
7	OFF	OFF	OFF	OFF	ON	ON	ON
8	ON	ON	ON	ON	ON	ON	ON
9	ON	ON	OFF	ON	ON	ON	ON

Tableau III-5 : Table de vérité d'un affichage à 7 segments

IV.3.2.2.5 CD4511

Le CD4511 est un circuit intégré de commande de décodeur de verrouillage BCD à 7 segments formé d'une logique CMOS et de dispositifs de sortie à transistor bipolaire NPN sur une structure fixe. Ce circuit intégré est utilisé lorsque nous devons piloter des affichages à cathode commune comme un affichage à 7 segments, un affichage fluorescent basse tension et un affichage à incandescence.

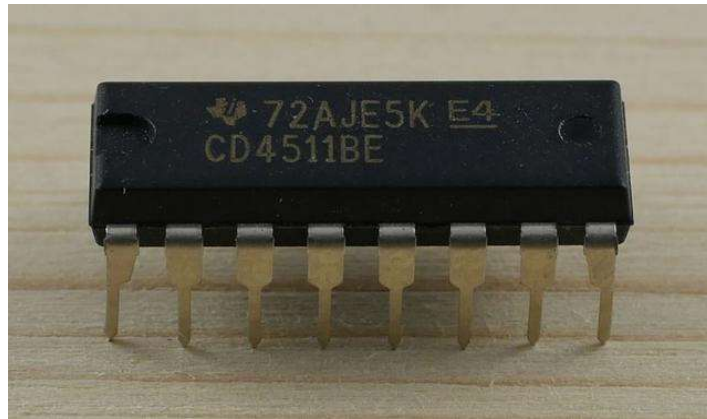


Figure III.39: Le CD4511.

IV.3.2.2.6 Le microcontrôleur (carte Arduino Méga 2560)

La MEGA 2560 est conçu pour des projets plus complexes. Avec 54 broches E /S dont 14 PWM, 16 entrées analogiques et d'un plus grand espace pour les croquis. Cela donne à notre projet beaucoup d'espace et d'opportunités. En effet la carte Arduino Méga 2560 offre toutes les fonctionnalités de la carte UNO, mais avec des fonctionnalités supplémentaires. [20]

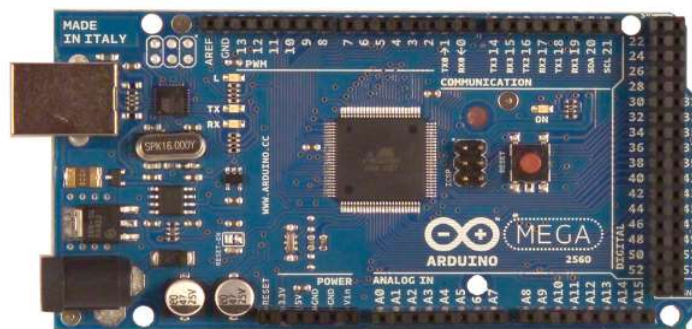


Figure III.40: carte Arduino Méga 2560.

Les caractéristiques de la Méga 2560	
✚ Version : Rev.3 ;	✚ Alimentation : via port USB ou 7 à12 V
✚ Microprocesseur : ATmega2560 ;	✚ Mémoire flash : 256 KB ;
✚ Mémoire SRAM : 8 KB ;	✚ Mémoire EEPROM : 4 KB ;
✚ 54 broches d'E/S dont 14 PWM ;	✚ 16 entrées analogiques 10 bits ;
✚ Intensité par E/S : 40 mA ;	✚ Cadencement : 16 MHz ;
✚ 3 ports séries ;	✚ Bus 12C et SPI ;
✚ Gestion des interruptions ;	✚ Fiche USB B ;
✚ Dimensions : 107 x 53 x15 mm	

Tableau III-6: tableau regroupant les caractéristiques de la Mega 2560.

IV.3.2.2.7 Afficheur LCD :

LCD, ou Liquid Crystal Display est un type d'affichage qui utilise des cristaux liquides pour afficher des images, il peut se présenter sous la forme d'un moniteur spécifique.

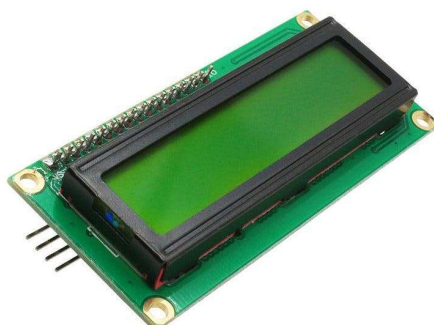


Figure III.41: Liquid Crystal Display 16*2

Les écrans LCD fonctionnent en utilisant une combinaison de rétro éclairage, de polariseurs, de cristaux liquides et de filtres de couleur. En ajustant la quantité d'électricité qui va aux cristaux liquides, l'écran LCD peut contrôler la couleur de chaque photo.

IV.3.2.2.8 Interrupteur de position

Les interrupteurs de positions mécaniques peuvent aussi être appelés « Détecteur de position » et « Interrupteur de fin de course ». Ils coupent ou établissent un circuit lorsqu'ils sont actionnés par un mobile.

La détection s'effectue par contact d'un objet extérieur sur le levier ou un galet. Ce capteur peut prendre alors deux états :

- Enfoncé (en logique positive l'interrupteur est fermé).
- Relâché pour la logique de tous

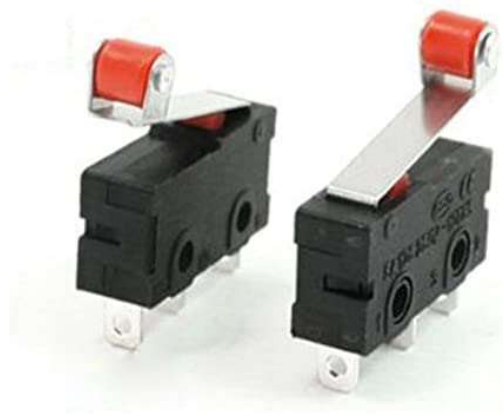


Figure III.42: Interrupteur de position (Capteur fin de course mécanique)

IV.3.2.2.9 DC Moteur

Les moteurs à courant continu (Direct Courant) sont actuellement les plus couramment utilisés car ils sont faciles à miniaturiser et offrent un bon contrôle de la rotation ainsi qu'un rendement élevé. Ils bénéficient aussi d'une longue durée de vie, d'une facilité d'entretien et d'un faible bruit car ils suppriment les balais et les collecteurs, qui sont les inconvénients des moteurs à courant continu à balais.

- Couple nominal 1,6 Ncm
- Puissance utile de 4,9 à 5,3 W
- Tension nominale 12 ou 24 Vdc
- Vitesse nominale de 3150 à 2900 tr/mn

- Courant nominal de 0,45 à 0,95 A
- Constante de vitesse de 1281 à 1375

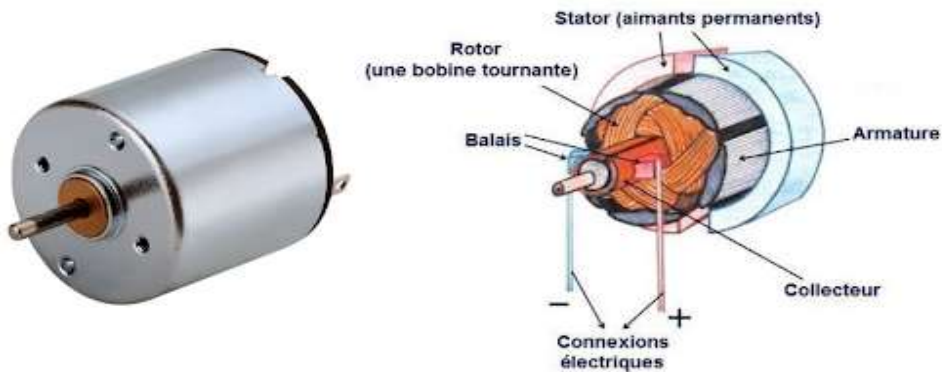


Figure III.43: Structure interne d'un moteur à courant continu

IV.3.2.2.10 Carte d'alimentation

La conception de cette carte est faite selon nos besoins d'alimentation. Le moteur doit être alimenté par 12V ; la carte Arduino par 5V

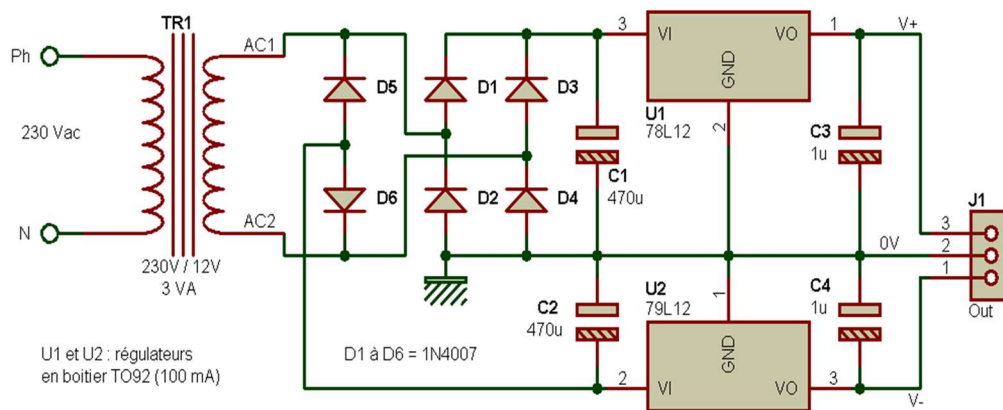


Figure III.44: Schéma interne d'une carte d'alimentation

IV.3.2.2.11 Terminal

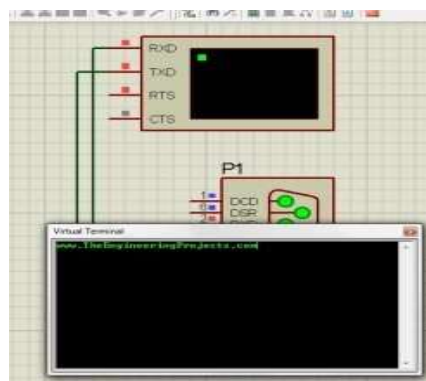


Figure III.45: Image d'un Terminal Virtual

Virtual Terminal est un outil de Proteus, qui est utilisé pour afficher les données provenant du port série (DB9) et également utilisé pour envoyer les données au port série.

IV.3.2.3 Schéma synoptique de la carte de commande :

Cette partie se base sur la conception des cartes électroniques : deux pour l'alimentation, une carte de puissance pour la commande du moteur, 3 (trois) Capteurs (un pour chaque niveau) et un afficheur 7 segments ainsi que des boutons poussoir d'appel.

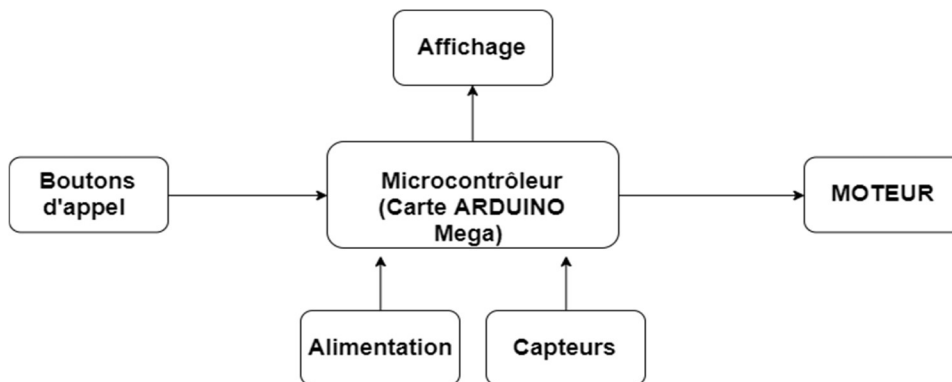


Figure III.46: Organigramme de la carte de commande

IV.3.2.4 Calcul de puissance pour le moteur

Le moteur utilisé dans ce travail est un moteur à courant continu. Voici quelque exemple de calcule qui sont appliqué pour réaliser un ascenseur avec des paramètre réel.

Exemple :

La puissance, le travail et la force sont calculés comme suit :

$$\text{Puissance} = \text{Travail} / \text{Temps}$$

$$\text{Travail} = \text{Force} \times \text{Distance}$$

$$\text{Force, } F = mg$$

$$\text{Ce qui implique : Puissance} = \text{Force} \times \text{distance} / \text{Temps}$$

$$\text{Puissance} = \text{Force} \times \text{Vitesse}$$

En va prendre les coordonnées suivantes pour avoir une meilleure idée,

Si le poids de la cabine vide = 100 kg donc Le Contrepoids = 100 kg

Et en supposent que 10 personnes de 65 kg chacune = 650 kg

Pour un fonctionnement à vitesse constante de 1 m/s ou 60 m/min

$$\text{Force} = 650 \times 9,8 = 6370 \text{ N}$$

Puissance = $6370\text{N} \times 1\text{m/s} = 6370\text{W}$ (Avec $1\text{cv} = 745\text{ W}$)

Donc 6370 W sera $6370 / 745 = 8.55\text{cv}$

Cela peut être approximatif à 9hp ou 6705 W

Vitesse de rotation = 1500 tr/min

Le couple du moteur peut être calculé en utilisant l'expression suivante :

Couple du moteur (Nm) = puissance en *Watts* / $2\pi\omega = 6705 / 2\pi \times 1500 = 31,5\text{ Nm}$

IV.3.2.5 Calcul Force de la corde

La tension de la corde et son choix de la sélection peut être déterminée avec un calcul simple. La force exercée sur le peut être trouvé en utilisant la formule de Newton.

$F = \text{masse} \times \text{accélérations} = mg$

Le poids total prévu de l'ascenseur lorsqu'il est complètement chargé comme l'exemple suivant sera :

10 personnes de 65 kg chacune = 650 kg

Poids de la cabine = 100 kg

Contrepoids = 100 kg

$F = (650+100+100) \times 9,8 = 8330\text{ N} = 8,33\text{ kN}$

La force agissant sur la corde est de $8,33\text{ kN}$. Donc une chaîne avec la capacité de supporter au moins 9kN doit être choisi sur une base de la force de corde standard.



Figure III.47: Corde D'un ascenseur

IV.3.2.6 Calcul d'engrenage

Pour atteindre la vitesse souhaitée de 1 m/s à partir de la puissance du moteur de 1500 tr/min, le diamètre et la circonférence des engrenages doivent être calculés ainsi :

- Poids de la cabine = 100kg
- Contre poids = 100kg
- 10 personnes avec 65kg = 650kg
- La vitesse de l'ascenseur est de 1 m/s = 60m/min = 6000 cm/min
- Circonférence de l'engrenage 2, $c_2 = 30\text{cm}$

Par conséquent, diamètre de l'engrenage 2, $d_2 = 30 / 3.14 = 9,55\text{ cm}$

Pour trouver le régime de l'engrenage :

$\text{RPM (tr/min)} = \text{vitesse} / \text{circonférence}$

Vitesse de l'engrenage 2 = $6000 / 30 = 200\text{ tr/min}$

IV.3.2.7 Teste De La maquette :

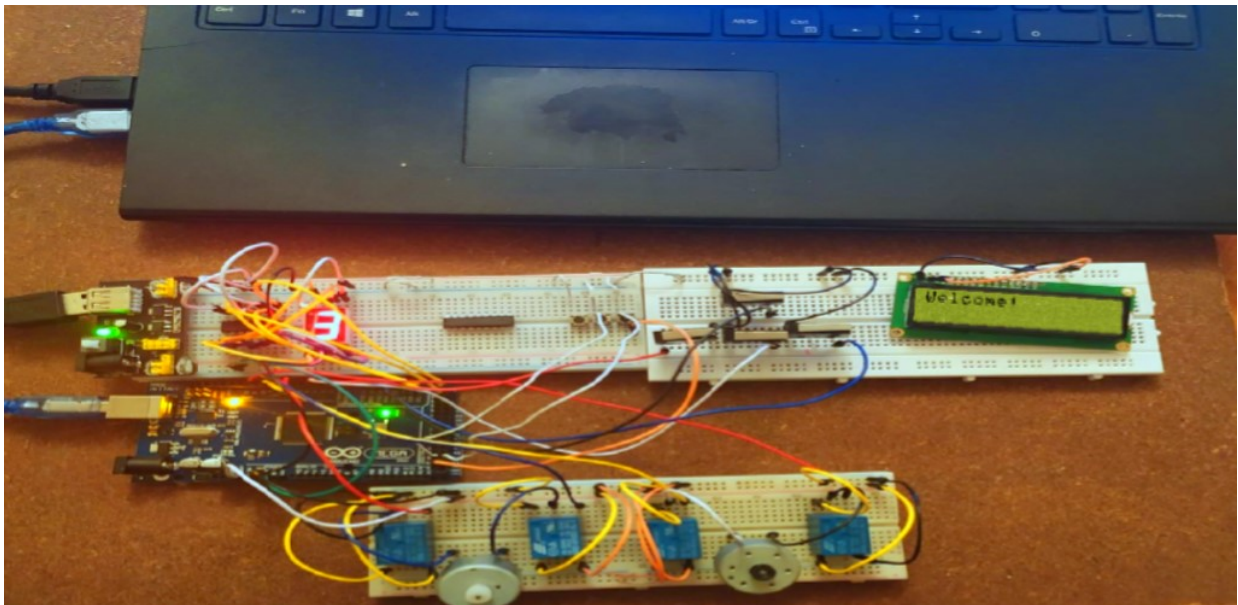


Figure III.48 automate programmable d'un ascenseur a base d'un microcontrôleur ARDUINO

IV.4 Conclusion

Dans ce dernier chapitre, nous avons abordé dans une première approche les différentes phases d'élaboration de la carte de commande. Ensuite nous avons effectué un test afin de vérifier le fonctionnement et les performances de notre maquette.

L'ascenseur est un système automatisé intéressant et que sa réalisation fait appel à plusieurs domaines technologiques, de plus, c'est un moyen de déplacement très utilisé et de plus en plus répandu, ce qui met en œuvre le rôle important de la maintenance dans l'optimisation de la sécurité des usagers et la disponibilité de l'appareil. Avoir eu la possibilité d'étudier une maquette assimilant le fonctionnement d'un ascenseur a été très bénéfique, abstraction faite des difficultés rencontrées en ce qui concerne la procuration des composantes nécessaires à la réalisation que nous aurions souhaitée complète.

Conclusion Générale

Au cours de ce projet de fin d'études, nous avons pu concrétiser notre idée qui consiste à réaliser un automate programmable industriel à base d'une carte Arduino Méga 2560.

De nos jours, l'utilisation des API est généralisée en industrie, en domotique, en automatisme et dans les systèmes embarqués. Les exigences de production, ont fait des automates des éléments essentiels des chaînes automatisées, surtout en milieu industriel.

L'automate que nous avons conçu et réalisé nous a permis de comprendre plus finement les concepts de l'automatisation.

On a commencé nos travaux par des essais et des analyses graphiques en utilisant Grafcet et le langage Ladder, puis on a implémenté notre raisonnement dans le Logiciel Arduino, finalement La conception des schémas électroniques a été fait et simulé sur le logiciel Proteus 8 avant la mise en œuvre.

En perspective, ce projet constitue un domaine d'études et de développement en master aussi bien en génie industrielle, ainsi que l'instrumentation, Malgré les difficultés rencontrées lors de la réalisation de ce modeste projet, le résultat final est un succès étant donné que l'objectif visé est atteint.

Ce projet de conception peut éventuellement être élargi en termes d'application, et ce, en ajoutant quelques options pour la maquette. A titre d'exemple, nous proposons les améliorations suivantes :

-Arrêt d'urgence.

-Charge maximal.

-Bouton poussoir pour annuler les appels et les demande et maintenir la position actuelle.

ANNEXE

ANNEXE OU

```

constintAin = 7;
constintBin = 6;
constintCin = 5;
constintDin = 4;
constintEin = 3;
constintFin = 2;
constintGin = 1;
constintY0out = 13;
constintY1out = 12;
constintY2out = 11;
constintY3out = 10;
constintY4out = 9;
constintY5out = 8;
constintledPin13 = 13;
constintledPin12 = 12;
constintledPin11 = 11;
constintledPin10 = 10;
constintledPin9 = 9;
constintledPin8 = 8;
inti=0;

intbuttonState = 0;
int R=0 ,Y0 = 0 ,Y1 = 0 , Y2 = 0 , Y3 = 0 ,
Y4 = 0 , Y5 = 0 ,A,B,C,D,E,F,G;

void setup() {
pinMode (Y0out, OUTPUT);
pinMode (Y1out, OUTPUT);
pinMode (Y2out, OUTPUT);
pinMode (Y3out, OUTPUT);
pinMode (Y4out, OUTPUT);
pinMode (Y4out, OUTPUT);
pinMode (Ain, INPUT);
pinMode (Bin, INPUT);
pinMode (Cin, INPUT);
pinMode(Din, INPUT);
pinMode (Ein, INPUT);
pinMode (Fin, INPUT);
pinMode(Gin, INPUT);

```

```

Y0 = digitalRead (Y0out);
Y1 = digitalRead (Y1out);
Y2 = digitalRead (Y2out);
Y3 = digitalRead (Y3out);
Y4 = digitalRead (Y4out);
Y5= digitalRead (Y5out);}

voidloop() {

A = digitalRead (Ain);
B = digitalRead (Bin);
C = digitalRead (Cin);
D = digitalRead (Din);
E = digitalRead (Ein);
F = digitalRead (Fin);
G = digitalRead (Gin);

Y0 = digitalRead (Y0out);
Y1 = digitalRead (Y1out);
Y2 = digitalRead (Y2out);
Y3 = digitalRead (Y3out);
Y4 = digitalRead (Y4out);
Y5 = digitalRead(Y5out);

digitalWrite(Y0out , Y0);
Y1 = (A & Y0 ) | ( Y1 & ~Y3 );
digitalWrite (Y1out , Y1 );
Y2 = (B & Y0 ) | ( Y2 & ~Y4 );
digitalWrite (Y2out , Y2);
Y3 = (C & Y1 ) | ( Y3 & ~Y5 );
digitalWrite (Y3out , Y3);
Y4 = (D & Y2 ) | ( Y4 & ~Y5 );
digitalWrite (Y4out , Y4);
Y5 = (E & Y3 ) | ( F & Y4 ) | ( Y5 & ~Y0 );
digitalWrite (Y5out , Y5);

}

```

ANNEXE ET

```
const int Ain = 7;
const int Bin = 6;
const int Cin = 5;
const int Din = 4;
const int tin = 3;
const int Y0out = 13;
const int Y1out = 12;
const int Y2out = 11;
const int Y3out = 10;
const int Y4out = 9;
int i=0;
int R=0 ,Y0 = 1 ,Y1 = 0 ,Y2 = 0 ,Y3 = 0 ,Y4 =
0 , A , B ,C, D ,t ;

void setup() {
pinMode (Y0out, OUTPUT);
pinMode (Y1out, OUTPUT);
pinMode (Y2out, OUTPUT);
pinMode (Y3out, OUTPUT);
pinMode (Y4out, OUTPUT);

pinMode (Ain, INPUT);
pinMode (Bin, INPUT);
pinMode (Cin, INPUT);
```

```
pinMode (Din, INPUT);
pinMode (tin, INPUT);
Y0=1;
}

void loop() {
do{
A= digitalRead (Ain);
B = digitalRead (Bin);
C = digitalRead (Cin);
D = digitalRead (Din);
t = digitalRead (tin);
Y0=(t & Y2 & Y4)|( Y0 & ~Y1 & ~Y3);
digitalWrite (Y0out , Y0);
Y1 = (Y0 & A) | ( Y1 & ~Y2 ) ;
digitalWrite (Y1out , Y1 );
Y2 = (Y1 & B) | ( Y2 & ~Y0 ) ;
digitalWrite (Y2out , Y2 );
Y3 = (Y0 & A) | ( Y3 & ~Y4 ) ;
digitalWrite (Y3out , Y3 );
Y4 = (Y3 & C) | ( Y4 & ~Y0 ) ;
digitalWrite (Y4out , Y4 );
}
while(1);
}
```

ANNEXE TIMMER

```

#include <elapsedMillis.h>
const int Ain = 7;
const int Bin = 6;
const int Y0out = 13;
const int Y1out = 12;
const int ledPin13 = 13;
int i=0;
int R=0 ,Y0 = 0 ,Y1 = 0 , A , B ;

void setup() {
  pinMode(Y0out, OUTPUT);
  pinMode(Y1out, OUTPUT);
  pinMode(Ain, INPUT);
  pinMode(Bin, INPUT);
  Y0 = 1 ;
  Y1 = 0;
}

class TONclass
{ private:
  int state_ ;
  bool TON0_ ;
  int Tinit_ ,Tfin_ ;
  int Tinterval_ ;
  elapsedMillis TONmil_ ;
public:
  TONclass(intTONinter=0)
  {TON0_=0;state_=0; Tinterval_ = TONinter;};

  int delays (int x, int TONinter=0);

  ~TONclass(){state_=0;Tinterval_=0;}; };
int TONclass::delays(int x,int TONinter)
{
  if (TONinter!=0)Tinterval_=TONinter;
  if (x)
  { if (state_==0 )
    { Tinit_ = TONmil_ state_=1; }
    Tfin_=TONmil_ ;
    if
    ( Tfin_ - Tinit_ >= Tinterval_ )
    { TON0_ = 1 ; state_=0 ; } }else
    { state_ = 0 ; if (TON0_==1)
    { TON0_=0;return TON0_ ;} }
  return TON0_ ;
}

TONclass Ton1(2000),Ton2;
void loop() {
  A = digitalRead(Ain);
  B = digitalRead(Bin);
  Y0 = (Ton2.delays(B,1000) & Y1 ) | ( Y0 & ~Y1 ) ;
  digitalWrite(Y0out , Y0 );
  Y1 = (Ton1.delays(A & Y0,2000) ) | ( Y1 & ~Y0 ) ;
  digitalWrite(Y1out , Y1 );
}

```

ANNEXE ASCENSEUR : VERSION 1

```

#include <elapsedMillis.h>
const int Ain = 11;
const int Bin = 10;
const int E1in = 9;
const int E2in = 8;
const int E3in = 7;
const int P1in = 6;
const int P2in = 5;
const int P3in = 3;

const int Y0out = 53;
const int Y1out = 20;
const int Y2out = 19;
const int Y3out = 18;
const int Y4out = 17;
const int Y5out = 16;
const int Y6out = 15;
const int Y7out = 14;
const int Y8out = 13;
const int Y9out = 12;

const int ledPin21 = 53;
const int ledPin20 = 20;
const int ledPin19 = 19;
const int ledPin18 = 18;
const int ledPin17 = 17;
const int ledPin16 = 16;
const int ledPin15 = 15;
const int edPin14 = 14;
const int ledPin13 = 13;
const int ledPin12 = 12;

elapsedMillis TONmil;

int i=0;
int Tinit , Tfin;
int buttonState = 0;
int Y0 = 0 , Y1 = 0 , Y2 = 0 , Y3 = 0 ,
Y4 = 0 , Y5 = 0 , Y6 = 0 , Y7 = 0 , Y8 = 0 , Y9 =
0 , TON = 0 , E1,E2,E3,P1,P2,P3,A,B;
int state =0 ;

void setup() {
  pinMode(Y0out, OUTPUT);
  pinMode(Y1out, OUTPUT);
  pinMode(Y2out, OUTPUT);
  pinMode(Y3out, OUTPUT);
  pinMode(Y4out, OUTPUT);
  pinMode(Y5out, OUTPUT);
  pinMode(Y6out, OUTPUT);
  pinMode(Y7out, OUTPUT);
  pinMode(Y8out, OUTPUT);
  pinMode(Y9out, OUTPUT);
  pinMode(E1in, INPUT);
  pinMode(E2in, INPUT);
  pinMode(E3in, INPUT);
  pinMode(P1in, INPUT);
  pinMode(P2in, INPUT);
  pinMode(P3in, INPUT);
  pinMode(Ain, INPUT);
  pinMode(Bin, INPUT);
  Y0 = digitalRead (Y0out);
  Y1 = digitalRead (Y1out);
  Y2 = digitalRead (Y2out);
  Y3 = digitalRead (Y3out);
  Y4 = digitalRead (Y4out);
  Y5 = digitalRead (Y5out);
  Y6 = digitalRead (Y6out);
  Y7 = digitalRead (Y7out);
  Y8 = digitalRead (Y8out);
  Y9 = digitalRead (Y9out);
}

void loop() {
  A = digitalRead (Ain);
  B = digitalRead (Bin);
  E1 = digitalRead (E1in);
  E2 = digitalRead (E2in);
  E3 = digitalRead (E3in);
  P1 = digitalRead (P1in);
  P2 = digitalRead (P2in);
  P3 = digitalRead (P3in);
  Y0 = digitalRead(Y0out);
  Y1 = digitalRead (Y1out);
  Y2 = digitalRead (Y2out);
  Y3 = digitalRead (Y3out);
  Y4 = digitalRead (Y4out);
  Y5 = digitalRead (Y5out);
}

```

```
Y6 = digitalRead (Y6out);
Y7 = digitalRead (Y7out);
Y8 = digitalRead (Y8out);
Y9 = digitalRead (Y9out);
```

```
Y0 = (B & Y3 ) | ( Y0 & ~Y1 & ~Y4 & ~Y5 & ~Y8 );
digitalWrite (Y0out , Y0);
```

```
Y1 = ((E1 & Y0)&( P2|P3 )) | ( Y1 & ~Y2 );
digitalWrite (Y1out , Y1 );
```

```
Y2 = (Y1 & P1 ) | ( Y2 & ~Y3 );
digitalWrite (Y2out , Y2);
```

```
int TONDelay = 1000 ;
```

```
if (A & Y2)
{ if (state ==0 ) { Tinit = TONmil ; state =1; }
Tfin=TONmil ;
```

```
if ( Tfin- Tinit >= TONDelay )
{
TON = 1 ; state=0 ;
}
}
else
{ state = 0 ; }
```

```
Y3 = (Y2 & A & TON ) | ( Y6 & A & TON ) | ( Y7 & A & TON ) | ( Y9 & A & TON ) | ( Y3 & ~Y0 );
digitalWrite (Y3out , Y3);
TON = 0 ;
```

```
Y4 = (E2 & P3 & Y2 ) | ( Y4 & ~Y6 );
digitalWrite (Y4out , Y4);
```

```
Y5 = (B & P1 & Y0 ) | ( Y5 & ~Y7 );
digitalWrite (Y5out , Y5);
```

```
Y6 = (P2 & Y4 ) | ( Y6 & ~Y3 );
digitalWrite (Y6out , Y6);
```

```
Y7 = (P2 & Y5 ) | ( Y7 & ~Y3 );
digitalWrite(Y7out , Y7);
Y8 =((E3 & Y0) &( P1|P2 )) | ( Y8 & ~Y9 );
digitalWrite (Y8out , Y8);
```

```
Y9 = (P3 & Y8 ) | ( Y9 & ~Y0 );
digitalWrite (Y9out , Y9);
```

```
}
```

ANNEXE ASCENSEUR VERSION 2

```

#include "PinChangeInterrupt.h"
#include <LiquidCrystal.h>

volatile int level = 0;
volatile int levelsToGo[3] = {0, 0, 0, };
volatile int doorOpen = false;
volatile int doorClosing = false;
volatile int comingFrom = 0;
volatile int doorClosed = true;
volatile int moving = false;

int doorDelay = 5000;
const int rs = 14, en = 15, d4 = 5, d5 = 4, d6 =
16, d7 = 17;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup()
{
displayDigitInit();
limitSwitchersInit();
upDownInit();
buttonsInit();
closeOpenInit();
Serial.begin(9600);
Serial.println("starting ....");
lcd.begin(0, 0)
lcd.print("Welcome!");
levelInfo();
}

void loop()
{
if (doorOpen && !doorClosing)
{
delay(doorDelay);
closeDoor();
delay(5000);
lcd.setCursor(0, 1);
lcd.print("Welcome! ");
}
}

void levelInfo()
{
Serial.print ( "on level : " );

```

```

Serial.println(level + 1);
displayDigit();
Serial.print ("levels to go : ");
Serial.print (levelsToGo[0]);
Serial.print (" ");
Serial.print(levelsToGo[1]);
Serial.print(" ");
Serial.print(levelsToGo[2]);
Serial.print(" ");
}

/** 7 segments **/

uint8_t a = 22;
uint8_t b = 23;
uint8_t c = 24;
uint8_t d = 25;
uint8_t e = 26;
uint8_t f = 27;
uint8_t g = 28;

void displayDigitInit()
{
pinMode (a, OUTPUT); //A
pinMode (b, OUTPUT); //B
pinMode (c, OUTPUT); //C
pinMode (d, OUTPUT); //D
pinMode (e, OUTPUT); //E
pinMode (f, OUTPUT); //F
pinMode (g, OUTPUT); //G}

void displayDigit()
{
int digit = level + 1;
displayDigitOff();
PORTA = digit ;
}

void displayDigitOff()
{
digitalWrite(a, LOW);
digitalWrite (b, LOW);
digitalWrite (c, LOW);
digitalWrite (d, LOW);
digitalWrite (e, LOW);

```



```

digitalWrite (f, LOW);
digitalWrite (g, LOW);
}
/** 7 segments **/

/** limit switches pins **/

uint8_t LV1Switich = 2;
uint8_t LV2Switich = 3;
uint8_t LV3Switich = 18;
uint8_t DoorSwitich = 19;

void limitSwitchersInit()
{
pinMode(LV1Switich, INPUT_PULLUP);
pinMode(LV2Switich, INPUT_PULLUP);
pinMode(LV3Switich, INPUT_PULLUP);
pinMode(DoorSwitich, INPUT_PULLUP);

attachInterrupt(digitalPinToInterrupt
(LV1Switich), floor1reach, RISING);

attachInterrupt(digitalPinToInterrupt
(LV2Switich), floor2reach, RISING);

attachInterrupt(digitalPinToInterrupt
(LV3Switich), floor3reach, RISING);

attachInterrupt(digitalPinToInterrupt
(DoorSwitich), stopDoor, RISING);
}

void floorReach(int lv)
{
level = lv;
if (levelsToGo[lv])
{
stopMoving();
levelsToGo[lv] = 0;
}
levelInfo();
}

void floor1reach()
{ floorReach(0); }

void floor2reach()
{ floorReach(1); }

```

```

void floor3reach()
{ floorReach(2); }

/** limit switches pins **/

/** buttons **/

volatile uint8_t upPin = 9;
volatile uint8_t downPin = 8;
void upDownInit()
{
pinMode(upPin, OUTPUT);
pinMode(downPin, OUTPUT);
}
void goUp()
{
Serial.println( "going up ...." );
digitalWrite(downPin, LOW);
digitalWrite(upPin, HIGH); lcd.setCursor(0, 1);
lcd.print( "Going Up ...." );
moving = true;
}
void goDown()
{ Serial.println( "going down ..." );
digitalWrite(upPin, LOW);
digitalWrite(downPin, HIGH); lcd.setCursor(0,
1);
lcd.print("Going Down ....");
moving = true;
}
void stopMoving()
{
Serial.println( "stopping ..." );
digitalWrite(upPin, LOW);
digitalWrite(downPin, LOW); lcd.setCursor(0,
1); lcd.print("Stopping....");
moving = false;
openDoor();
}

uint8_t LV1btn = 53;
uint8_t LV2btn = 52;
uint8_t LV3btn = 51;

```

```

void buttonsInit()
{
pinMode(LV1btn, INPUT_PULLUP);
pinMode(LV2btn, INPUT_PULLUP);
pinMode(LV3btn, INPUT_PULLUP);

attachPCINT(digitalPinToPCINT
(LV1btn), LV1btnClick, CHANGE);

attachPCINT (digitalPinToPCINT
(LV2btn), LV2btnClick, CHANGE);

attachPCINT (digitalPinToPCINT (LV3btn),
LV3btnClick, CHANGE);
}
void LV1btnClick(void)
{
if (level == 0 & !moving)
openDoor();
else
{
levelsToGo[0] = 1;
levelInfo();
move();
}
}

void LV2btnClick(void)
{
if (level == 1 & !moving)
openDoor();
else
{
levelsToGo[1] = 1;
levelInfo();
move();
}
}

void LV3btnClick(void)
{
if (level == 2 & !moving)
openDoor();
else
{
levelsToGo[2] = 1;
levelInfo();
move();
}
}

/** buttons */

/**Movement */
void move()
{ if (doorClosed)
{
if
(level == 0 && (levelsToGo[1] || levelsToGo[2] ))
goUp();
if (level == 1 && (levelsToGo[2] ||
levelsToGo[0]))
if (level == 1 && levelsToGo[2]) goUp();
else
goDown();
if (level == 2 && (levelsToGo[0] ||
levelsToGo[1]))
goDown();
comingFrom = level;
}
}

/**Movement */

/** open and close door */

volatile uint8_t openPin = 7;
volatile uint8_t closePin = 6;
void closeOpenInit()
{
pinMode(openPin, OUTPUT);
pinMode(closePin, OUTPUT);}

void openDoor()
{
doorClosing = false;
doorClosed = false;

Serial.println("opening door ....");
digitalWrite(closePin, LOW);
digitalWrite(openPin, HIGH);
}

```

```
lcd.setCursor(0, 1);
lcd.print("Opening Door....");

}

void closeDoor()
{
doorClosing = true;
Serial.println("closing door ...");
digitalWrite(openPin, LOW);
digitalWrite(closePin, HIGH);
lcd.setCursor(0, 1);
lcd.print("Closing Door...." );

}

void stopDoor()
{
Serial.println("stopping door ...");
digitalWrite(openPin, LOW);
digitalWrite(closePin, LOW);
lcd.setCursor(0, 1);
lcd.print( "Stopping Door ..." );
if (doorClosing)
{
doorOpen = false;
doorClosing = false;
doorClosed = true;
move();
}
else
{
doorOpen = true;
}

}
```

October 1987

Revised January 1999

CD4511BC

BCD-to-7 Segment Latch/Decoder/Driver

General Description

The CD4511BC BCD-to-seven segment latch/decoder/driver is constructed with complementary MOS (CMOS)

enhancement mode devices and NPN bipolar output drivers

in a single monolithic structure. The circuit provides the

functions of a 4-bit storage latch, an 8421 BCD-to-seven

segment decoder, and an output drive capability. Lamp test (25 mA)(LT), blanking (BI), and latch enable (LE) inputs are used to

test the display, to turn-off or pulse modulate the brightness

of the display, and to store a BCD code, respectively. It can

be used with seven-segment light emitting diodes (LED),

incandescent, fluorescent, gas discharge, or liquid crystal combinations readouts either directly or indirectly.

Applications include instrument (e.g., counter, DVM, etc.) display driver, computer/calculator display driver, cockpit

display driver, and various clock, watch, and timer

Features

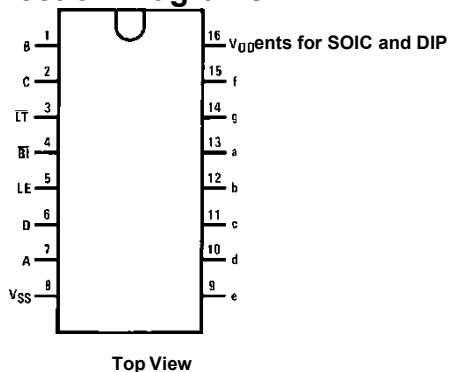
- Low logic circuit power dissipation
- High current sourcing outputs (up to
- Latch storage of code
- Blanking input
- Lamp test provision
- Readout blanking on all illegal input
- Lamp intensity modulation capability
- Time share (multiplexing) facility
- Equivalent to Motorola MC14511

Ordering Code:

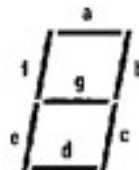
Order Number	Package Number	Package Description
CD4511BCWM	M16B	16-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300" Wide
CD4511BCN	N16E	16-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide

Devices also available in Tape and Reel. Specify by appending suffix letter "X" to the ordering code.

Connection Diagrams



Segment



Top View



Octal High Voltage, High Current Darlington Transistor Arrays

The eight NPN Darlington connected transistors in this family of arrays are ideally suited for interfacing between low logic level digital circuitry (such as TTL, CMOS or PMOS/NMOS) and the higher current/voltage requirements of lamps, relays, printer hammers or other similar loads for a broad range of computer, industrial, and consumer applications. All devices feature open-collector outputs and freewheeling clamp diodes for transient suppression.

The ULN2803 is designed to be compatible with standard TTL families while the ULN2804 is optimized for 6 to 15 volt high level CMOS or PMOS.

MAXIMUM RATINGS ($T_A = 25^\circ\text{C}$ and rating apply to any one device in the package, unless otherwise noted.)

Rating	Symbol	Value	Unit
Output Voltage	V_O	50	V
Input Voltage (Except ULN2801)	V_I	30	V
Collector Current - Continuous	I_C	500	mA
Base Current - Continuous	I_B	25	mA
Operating Ambient Temperature Range	T_A	0 to +70	$^\circ\text{C}$
Storage Temperature Range	T_{stg}	-55 to +150	$^\circ\text{C}$
Junction Temperature	T_J	125	$^\circ\text{C}$

$$R_{\theta JA} = 55^\circ\text{C/W}$$

Do not exceed maximum current limit per driver.

ORDERING INFORMATION

Device	Characteristics		
	Input Compatibility	$V_{CE}(\text{Max})/I_C(\text{Max})$	Operating Temperature Range
ULN2803A	TTL, 5.0 V CMOS	50 V/500 mA	$T_A = 0$ to $+70^\circ\text{C}$
ULN2804A	6 to 15 V CMOS, PMOS		

Order this document by ULN2803/D

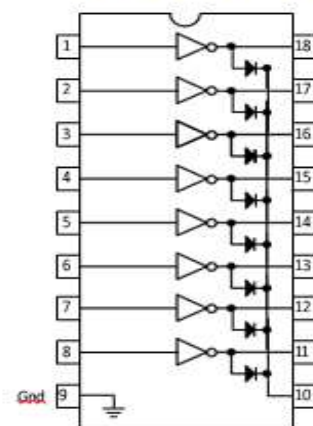
ULN2803

ULN2804

OCTAL PERIPHERAL DRIVER ARRAYS SEMICONDUCTOR TECHNICAL DATA



PIN CONNECTIONS



REFERENCES

- [1] Segovia, V. R., & Theorin, A. (2012). History of Control History of PLC and DCS. *University of Lund*.
- [2] Programmable Logic Controller '<https://ibrahim6060.weebly.com/introduction.html>'
- [3] Programmable Logic Controllers (PLCs): Basics, Types & Applications '<https://www.electrical4u.com/programmable-logic-controllers/>'
- [4] Ait Brahim Oualid,: '*Solutions pour le problème de télégestion et les problèmes Electriques-Mécanique durant le démarrage et l'arrêt des groupes de station de traitement d'eau*' Mémoire 2015.
- [5] Les Automates Programmables Industriels 'http://www.gecif.net/articles/genie_electrique/' Auteur . Jean-Christophe Michel
- [6]. Mme EL HAMMOUMI 'Cours Automatisme Logiques & Industriels GE1' cours en ligne usmba - Ecole Supérieure de Technologie Fès
- [7]. ANDRIANIRINA Mamy Harifetra Eddy '*Etudes de liaison dans un système d'équipements interconnectés intégrant en particulier un API (Automate Programmable Industriel) dans un Réseau Local Industriel (RLI)*' Ecole Supérieure polytechnique université d'Antananarivo- mémoire 2017.
- [8]. Hadjaissa Boubakeur '*Automates Programmables Industriels Description et programmation*', Université Amar Telidji de Laghouat Faculté de Technologie (cours) 2019.
- [9] Cherchour, H., Chahboune, M. L., & Mendil, B.. '*Commande et supervision d'un processus de margarine via un automate programmable chaine pilote*' Doctoral dissertation, Université abderrahmane mira béjaia 2015.
- [10] Sean Balogh '<https://www.encompass-inc.com/industrial-automation/>' Encompass Solutions 2019.
- [11] Zimmerman, G. P. '*Programmable logic controllers and Ladder logic*'. Rapid City: Dr. Alfred R. Boysen, Department of Humanities, South Dakota School of Mines and Technology 2008.
- [12] Base de la logique Ladder '<https://Ladderlogicworld.com/Ladder-logic-basics/>' Ladderlogicworld.com 2021.
- [13] L. Bergougnoux '*Automates Programmables Industriels*' '<https://www.technologuepro.com/cours-automate-programmable-industriel/Cours-Grafcet-notions-de-base.htm>' POLYTECH' Marseille 2004.
- [14] Paulo Jorge Oliveira & Revision with José Gaspar '*Industrial Automation Grafcet*' (2010) 'http://users.isr.ist.utl.pt/~jag/courses/api14/docs/API_I_C4.pdf'
- [15] GRAFCET: THE BASICS (2020) '<https://lab4sys.com/en/grafcet-the-basics/>'
- [16] RIAHI, I. '*Etude, simulation et réalisation De mini-générateurs BF et d'un mini-voltmètre AC-DC piloté par une carte Arduino Uno R3*', Doctoral dissertation 2016 .
- [17] Les différents type d'Arduino (2019) '<https://www.academia.edu/39339945>'
- [18] Louis, L. (2016). '*working principle of Arduino and using it*'. International Journal of Control, Automation, Communication and Systems (*IJCACS*), 1(2), 21-29.
- [19] Par Astalaseven , Eskimon et olyte Arduino '*Pour bien commencer en électronique et en programmation*' '<https://wiki.mdl29.net/lib/exe/fetch.php?media=elec:arduino-pour-bien-commencer-en-electronique-et-en-programmation.pdf>'
- [20] Référence Arduino français 'http://www.mon-club_elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.MaterielMega2560'
- [21] Arduino Mega 2 '<http://marcusjenkins.com/wp-content/uploads/2014/06/mega2.pdf>'