République Algérienne Démocratique et Populaire
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Ecole Nationale Polytechnique

Département Automatique

Laboratoire de Commande des processus

Mémoire de projet de fin d'études pour l'obtention du diplôme d'ingénieur d'état en Automatique

Thème

Conception et réalisation d'un capteur de niveau Application à la supervision d'une station de pompage didactique

Réalisé par : **Ahmed ALLALI**

Lyes Heythem BETTACHE

Sous la direction de M. El Madjid BERKOUK Professeur à l'ENP Alger

Présenté et soutenu publiquement le 19/06/2017

Composition du Jury :

Président :	L.ABDELOUL	Chargé de Cours à l'ENP
Rapporteur:	E.M.BERKOUK	Professeur à l'ENP
Examinateur:	D.BOUKHETALA	Professeur à l'ENP

ENP 2017

République Algérienne Démocratique et Populaire
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Ecole Nationale Polytechnique

Département Automatique

Laboratoire de Commande des processus

Mémoire de projet de fin d'études pour l'obtention du diplôme d'ingénieur d'état en Automatique

Thème

Conception et réalisation d'un capteur de niveau Application à la supervision d'une station de pompage didactique

Réalisé par : **Ahmed ALLALI**

Lyes Heythem BETTACHE

Sous la direction de M. El Madjid BERKOUK Professeur à l'ENP Alger

Présenté et soutenu publiquement le 19/06/2017

Composition du Jury :

Président :	L.ABDELOUL	Chargé de Cours à l'ENP
Rapporteur:	E.M.BERKOUK	Professeur à l'ENP
Examinateur:	D.BOUKHETALA	Professeur à l'ENP

ENP 2017

ملخص

العنوان انجاز محطة ضخ تعليمية باستعمال المسيرالصناعي المبرمج "سيمنس" S7 314 IFM و الأداة أردوينو
العمل المنجز في المذكرة يتمحور حول انجاز وتسيير محطة ضخ تعليمية بواسطة المبرمج S7 314 IFM, برنامجي ستابسات و وينيسيسي و الأداة أردوينو
المحطة المنجزة تحتوي على عدة أدوات تسمح باختيار نمط التسيير:

- كلي (TOR) او نسبي (ANALOGIQUE)
- مراقبة مباشرة لعملية الضخ (بواسطة وينسيسي)
- مراقبة و تسيير عن قرب أو عن بعد (باستعمال وينسيسي و الشبكة المحلية)
- الإحتفاظ و تخزين معطيات العملية

الكلمات المفتاحية

مسيرصناعي مبرمج"سيمنس ,برنامج "ستابسا ت" , برنامج" وينسيسي فلكسيبل ", أردوينو, محطة ضخ المياه.

Abstract :

The present work is mainly about the realization and management of dicdactic pumping station using the programmable logic controller S7 314 IFM, Step7 and WinCC softwares and arduino.

The realized station has many tools that allow us to choose the management type :

- logic or analogic
- Real time visualization of the process (using WinCC)
- Close or Remote visualization and control (using WinCC & local network)
- Archiving and storing of process's data.

Keywords:
Siemens PLC S7-314 IFM, STEP7 software, WINCC software, didactic pumping station, Arduino.

Résumé :

Ce mémoire présente essentiellement sur la réalisation d'une station de pompage didactique par le biais des automates programmables SIEMENS, Arduino et des logiciels STEP 7 et WinCC.

La station rélisée a des outils qui permettent de choisir le type de fonctionnement :

- Diagital or analogic
- Visualisation du procédé en temps réel
- Control et visualisation sur place ou à distance
- Archivage des données du procédé

Mots clés : Automate Siemens S7-300, Wincc, Step7, station de pompage didactique, Arduino.

Remerciements

Nous tenons à remercier dieu de nous avoir donné la force morale, physique et l'aide pour accomplir ce modeste travail.

Nous tenons à remercier notre promoteur Pr. BERKOUK pour nous avoir acceptées encadrées et dirigées durant l'élaboration de ce travail ainsi que pour son assistance et tous ses conseils.

Nous remercions également nos professeurs H.ACHOUR, R.ILLOUL et O.STIHI, pour leurs conseils et leurs aides.

Nous remercions chaleureusement les membres du jury pour l'honneur qu'ils nous ont fait en acceptent d'évaluer notre projet.

Nous souhaitons aussi remercier tous les enseignants de l'Ecole Nationale Polytechnique d'Alger, et en particulier, Nos professeurs d'Automatique qui nous ont encadrées auparavant et tous nos enseignants pour les connaissances qu'ils nous ont transmis, leur disponibilité et leurs efforts.

Que tous ceux qui ont contribué de près ou de loin à la réalisation de ce modeste travail trouvent ici l'expression de notre sincère gratitude.

Dédicaces

Je dédie ce travail tout d'abord à mes parents qui m'ont tout appris, tant donné sans rien demandé en retour.

A ma chère sœur et mes chers frères qui ont su m'épauler dans les moments difficiles.

A mon ami et binôme Heythem avec qui j'ai beaucoup appris.

A mes amis et camarades avec qui j'ai passé de bons moments.

A tous ceux qui ont contribué de loin ou de près à notre travail.

ALLALI Ahmed

Je dédie ce modeste travail tout d'abord à mon père qui m'a donné tous sans rien demandé en retour.

A ma chère mère sans elle je ne serais pas l'homme que je suis.

A mes sœurs

A mon petit frère

A mon ami et binôme Ahmed sur qui j'ai toujours pu compter.

A toute ma famille

A mes amis et camarades avec qui j'ai passé de bons moments.

A tous ceux qui ont contribué de loin ou de prêt à notre travail.

BETTACHE Lyes Heythem

Table des matières

Liste des tableaux

Liste des figures

Liste des abréviations

AA sortie analogique

AC Alternative Current

AE entrée analogique

AI Analog Input

AO Analog Output

API Automate Programmable Industriel

ASCII American Standard Code for Information Interchange

CHAR caractère

COUNT Schéma à Contact

CP Communication Processor

CPU Central Processing Unit

DA Sortie Digitale

DB Data Bloc

DC Direct Current (courant continu)

DE entrée digitale

DI Digital Input

DO Digital Output

DP Decentralized Periphery

E/S Entrée/Sortie

FB Fonctionnel Bloc

FC Fonction

FM Function Module

HW Config HardWare Configuration

IHM Interface Homme-Machine

HMI Human-Machine Interface

IM Coupleurs d'extension

INT INTeger

LED Light Emitting Diode

LIST Le langage de liste d'instructions

LOG LOGigramme

MPI Multi-Point Interface

OB Organisationnel Bloc

OP Operator Pupiter

PC Personnel Computer

PCMCIA Personal Computer Memory Card International Association

PG la console de programmation sur le terrain

PLC Programmable Logic Controller

PROFIBUS Process Field Bus

PROM Programmable Read Only Memory

EEPROM Electrically Ersable Programmable Read Only Memory

PS Gamme des alimentations stabilisees de Siemens

RAM Random Access Memory

ROM Read Only Memory

SFB System Fonctionnel Bloc

SFC System Fonction

SIMATIC Siemens Automatic

SM Signal Module

S7 Step 7

TOR Tout Ou Rien

PWM Pulse Width Modulation (Modulation à Largeur d'impulsion)

Introduction générale :

De nos jours, l'automatisme occupe une place importante dans les processus industriels, cette technologie revêt désormais d'une importance primordiale. Tant dans la production quotidienne que dans l'enseignement.

A cela s'ajoute que les nouvelles solutions industrielles, telles la décentralisation et la visualisation, exigent de nouveaux systèmes didactiques. En outre, les automates programmables ne sont plus programmés que selon des règles uniformes. Compte-tenu de ces exigences auxquelles sont confrontés les spécialistes de l'automation, il est indispensable aujourd'hui de proposer des systèmes d'entraînement orientés vers la pratique, permettant de transmettre l'état actuel de la technique et la compétence requise dans son maniement.

Dans le but de former ses élèves ingénieurs automaticiens à la maitrise des automates programmables, le Département d'Automatique de l'ENP a proposé le projet d'une station de pompage didactique, où la mise en marche et le développement de cette dernière permet aux élèves ingénieurs de se familiariser avec les systèmes industrielles, comme les API, et d'appliquer les connaissances acquises dans l'informatique industrielle, identification et la conception des circuits de conditionnement du signal.

Le présent rapport est organisé en quatre grands chapitres décrivant les volets principaux de notre projet:
Le premier chapitre sera consacré à la description des automates programmable : domaine d'application, architecture, …etc. et on finira par la description des réseaux de terrain MPI.

Dans le deuxième chapitre, nous décrirons le logiciel step7 et WinCC qui nous permettent de programmer et de contrôler facilement le procédé et de configurer l'interface Homme/Machine pour Visualiser et maîtriser le processus à tout instant.

Le troisième chapitre sera dédié à la conception et la réalisation d'un capteur de niveau à base d'Arduino et d'un circuit de conditionnement du signal.

Le quatrième chapitre sera consacré aux différentes réalisations effectuées tout au long du projet.

Chapitre 1

Les automates programmables industriels et le logiciel STEP7

Introduction

Un Automate Programmable Industriel (API, en anglais PLC pour Programmable Logic Controller) est une machine électronique programmable par un personnel non informaticien et destiné à piloter en environnement industriel et en temps réel des procédés industriels. Particulièrement, les automates Siemens S7-300 sont programmables par le biais du logiciel Step7. Il permet la programmation et la simulation de procédures automatisées avec différents langages normalisés.
Dans ce chapitre, on va présenter les principales fonctionnalités d'un automate programmable, son architecture ainsi que le logiciel de programmation Step7.

4.4.Les automates programmbles industriels

4.4.1. Historique

Les Automates Programmables Industriels sont apparus aux Etats-Unis vers 1969, à la demande de l'industrie automobile américaine. Le fabricant américain de voitures GM a décidé de remplacer les systèmes de commande à base de logique câblée (relais électrique) par une logique programmée.

Le développement de l'industrie a entraîné une augmentation constante des fonctions électroniques présentes dans un automatisme. C'est pour cette raison que l'API a remplacé les armoires grâce à sa souplesse dans la mise en œuvre, mais aussi parce que les coûts de câblage et de maintenance devenaient trop élevés.

4.4.2. Pourquoi l'automatisation ?

L'automatisation permet d'apporter des éléments supplémentaires à la valeur ajoutée par le système. Ces éléments sont exprimables en termes d'objectifs par :

- Accroître la productivité (rentabilité, compétitivité) du système
- Améliorer la flexibilité de production ;
- Améliorer la qualité du produit
- Adapter à des contextes particuliers tel que les environnements hostiles pour l'homme (milieu toxique, dangereux.. nucléaire...), adaptation à des tâches physiques ou intellectuelles pénibles pour l'homme (manipulation de lourdes charges, tâches répétitives parallélisées...),
- Augmenter la sécurité, etc...

4.4.3. L'architecture générale des API : [1]

Les caractéristiques principales d'un automate programmable industriel (API) sont :

- Compact ou modulaire, comme on peut le voir dans fig 1.1 et fig 1.2
- Tension d'alimentation (un réseau 220V en courant alternatif, ou d'une source 24V en courant continu)
- Taille mémoire
- Sauvegarde (EPROM, EEPROM, pile, …)
- Nombre d'entrées / sorties

- Modules complémentaires (analogique, communication,..)

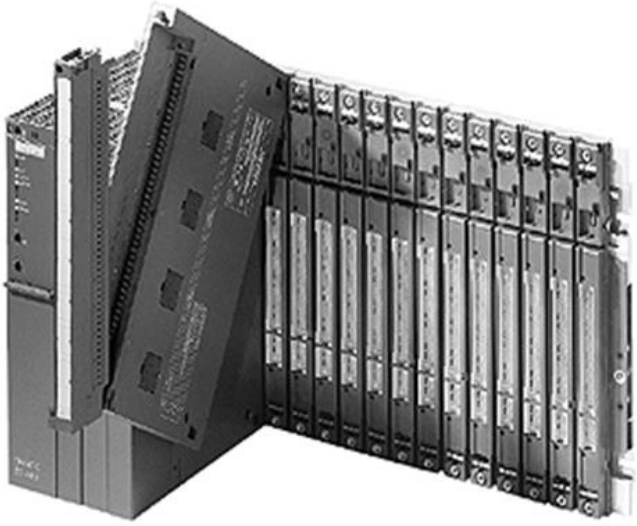


Figure 1.1 : Automate Modulaire [1]

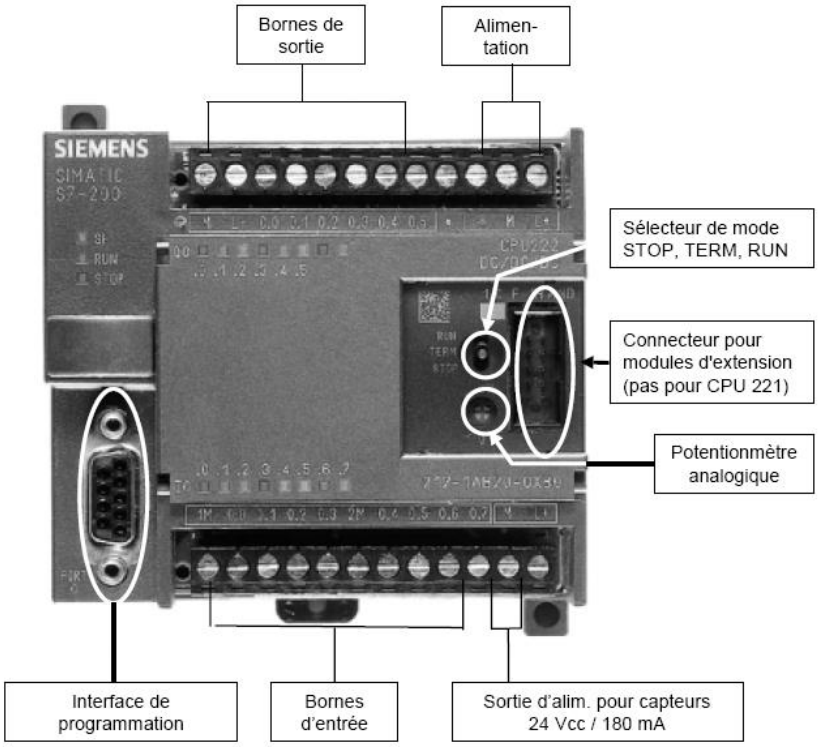


Figure 1.2 : Aspect extérieur d'un automate compact S7-200 CPU222 [1]

Des API en boîtier étanche sont utilisées pour les ambiances difficiles (température, poussière, risque de projection ...) supportant ainsi une large gamme de température, humidité ...etc. L'environnement industriel se présente sous trois formes :

- environnement physique et mécanique (poussières, température, humidité, vibrations);
- pollution chimique ;
- perturbation électrique (parasites électromagnétiques).

4.4.4. **Structure interne d'un automate programmable industriel (API)**

Les API comportent quatre principales parties (Figure 1.3) :

- Une unité centrale de traitement ou CPU (processeur et mémoires);
- Des modules d'entrées-sorties ;
- Des interfaces d'entrées-sorties ;
- Une alimentation 230 V, 50/60 Hz (AC) - 24 V (DC).

La structure interne d'un automate programmable industriel (API) est assez voisine de celle d'un système informatique simple, L'unité centrale est le regroupement du processeur et de la mémoire centrale. Elle commande l'interprétation et l'exécution des instructions programme. Les instructions sont exécutées les unes après les autres, séquencées par une horloge.
Deux types de mémoire cohabitent :
- **La mémoire Programme** où sont stockés les langages utilisés. Elle est en général figée, c'est à dire en lecture seulement. (ROM : mémoire morte)
- **La mémoire de données** utilisable en lecture-écriture pendant le fonctionnement c'est la RAM (mémoire vive). Elle fait partie du système entrées-sorties. Elle fige les valeurs (0 ou 1) présentes sur les lignes d'entrées, à chaque prise en compte cyclique de celle-ci, elle mémorise les valeurs calculées à placer sur les sorties.

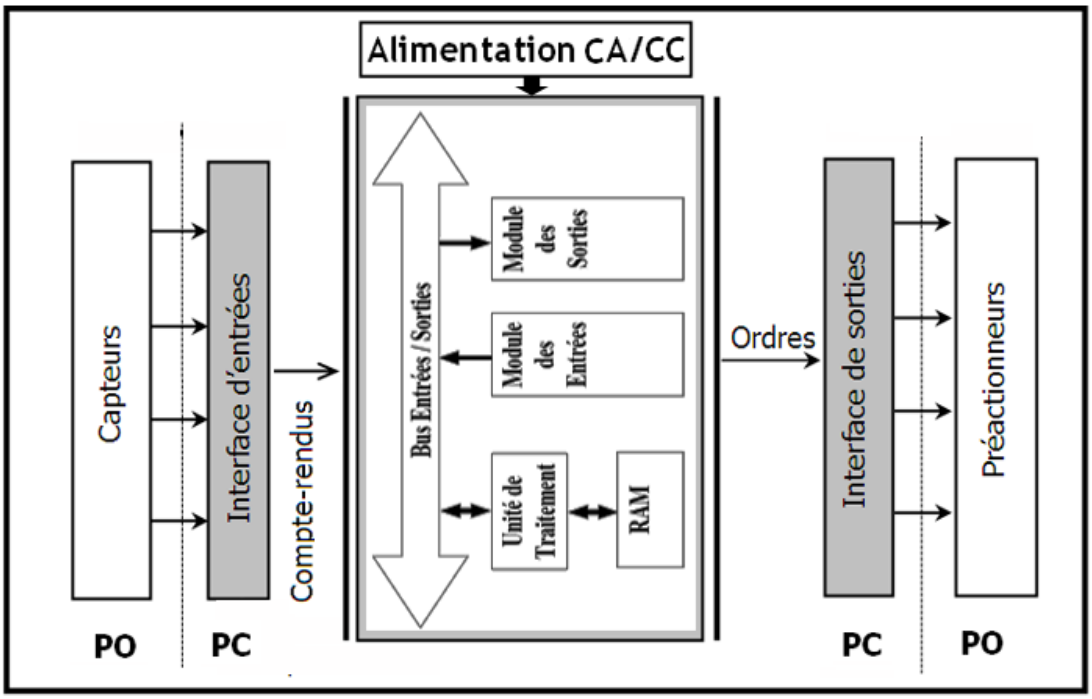


Figure 1.3 : Structure interne d'un automate programmable industriel (API) [1]

4.4.5. **Fonctionnement [1]**

L'automate programmable reçoit les informations relatives à l'état du système et puis commande les pré-actionneurs suivant le programme inscrit dans sa mémoire. Généralement les automates programmables industriels ont un fonctionnement cyclique (Figure 1.4). Le microprocesseur réalise toutes les fonctions logiques ET, OU, les fonctions de temporisation, de comptage, de calcul ...etc. Il est connecté aux autres éléments (mémoire et interface E/S) par des liaisons parallèles appelées ' BUS ' qui véhiculent les informations sous forme binaire.

Lorsque le fonctionnement est dit synchrone par rapport aux entrées et aux sorties, le cycle de traitement commence par la prise en compte des entrées qui sont figées en mémoire pour tout le cycle.

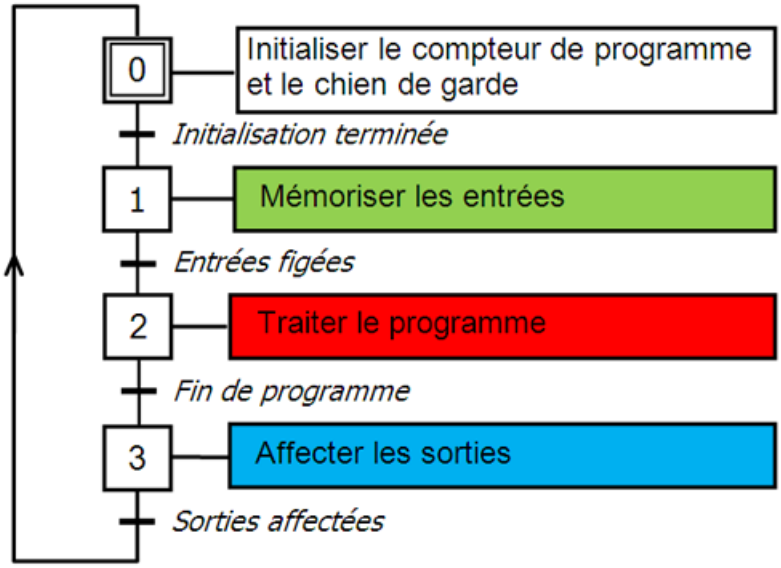


Figure 1.4 : Fonctionnement cyclique d'un API [1]

Le processeur exécute alors le programme instruction par instruction en rangeant à chaque fois les résultats en mémoire. En fin de cycle les sorties sont affectées d'un état binaire, par mise en communication avec les mémoires correspondantes. Dans ce cas, le temps de réponse à une variation d'état d'une entrée peut être compris entre un ou deux temps de cycle (durée moyenne d'un temps de cycle est de 5 à 15 ms Figure 1.6).

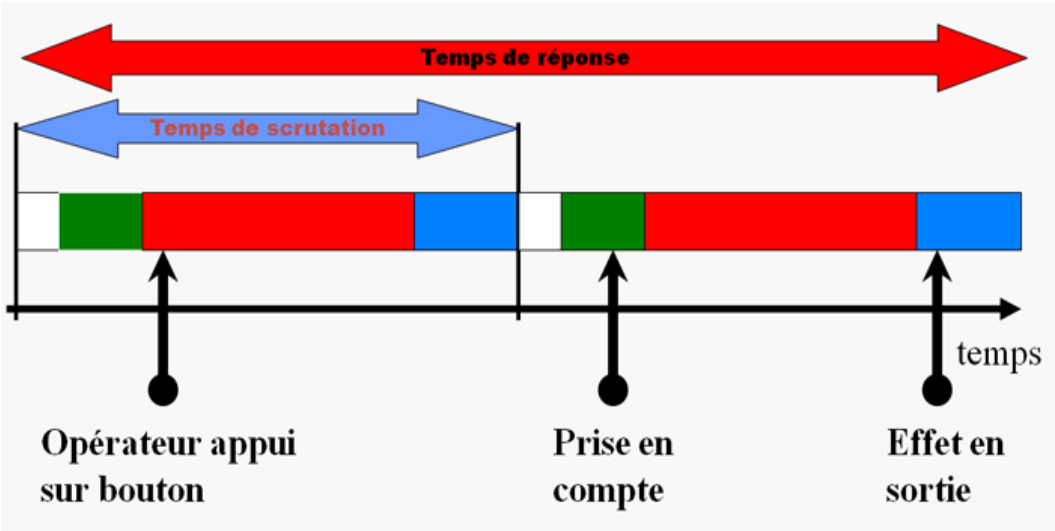


Figure 1.5 : Temps de scrutation vs Temps de réponse [1]

Il existe d'autres modes de fonctionnement, moins courants :

- synchrone par rapport aux entrées seulement ;
- asynchrone.

4.4.6. **Description de quelques éléments d'un API [1]**

4.4.6.1.**Les interfaces et les cartes d'Entrées / Sorties:**

L'interface d'entrée comporte des adresses d'entrée. Chaque capteur est relié à une de ces adresses. L'interface de sortie comporte de la même façon des adresses de sortie. Chaque pré actionneur est relié à une de ces adresses. Le nombre de ces entrées est sorties varie suivant le type d'automate. Les cartes d'E/S ont une modularité de 8, 16 ou 32 voies. Les tensions disponibles sont normalisées (24, 48, 110 ou 230V continu ou alternatif ...).

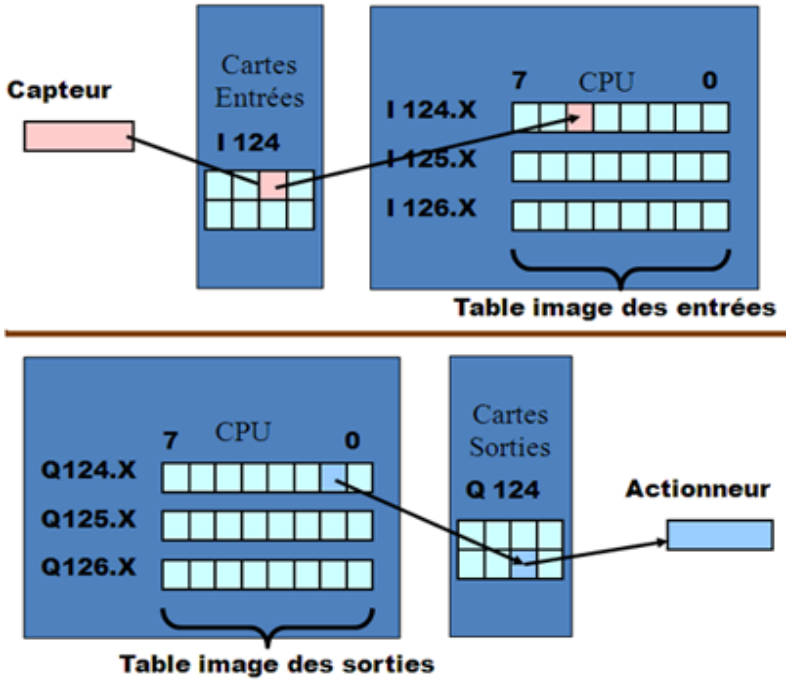


Figure 1.6 : Les interfaces d'entrées/sorties [1]

4.4.6.1.1. Cartes d'entrées :

Elles sont destinées à recevoir l'information en provenance des capteurs et adapter le signal en le mettant en forme, en éliminant les parasites et en isolant électriquement l'unité de commande de la partie opérative. La figure 1.7 montre une carte d'entrée typique

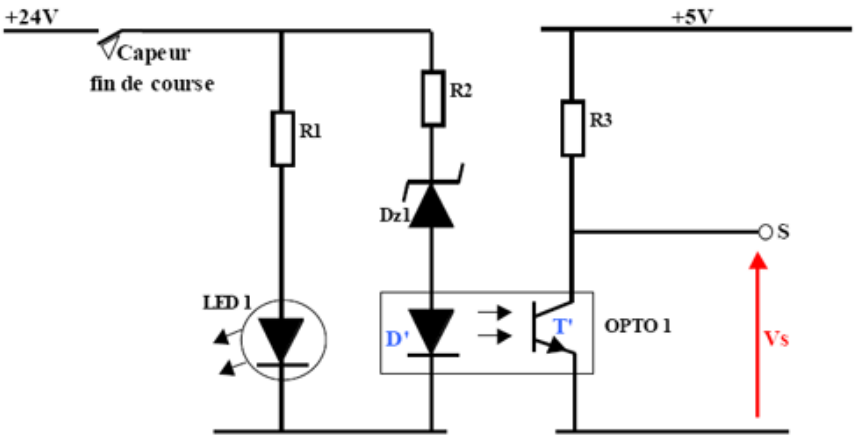


Figure 1.7: Exemple d'une carte d'entrées typique d'un API [1]

4.4.6.1.2. Cartes de sorties :

Elles sont destinées à commander les pré-actionneurs et éléments des signalisations du système et adapter les niveaux de tensions de l'unité de commande à celle de la partie opérative du système en garantissant une isolation galvanique entre ces dernières. La figure 1.8 montre une carte de sortie typique

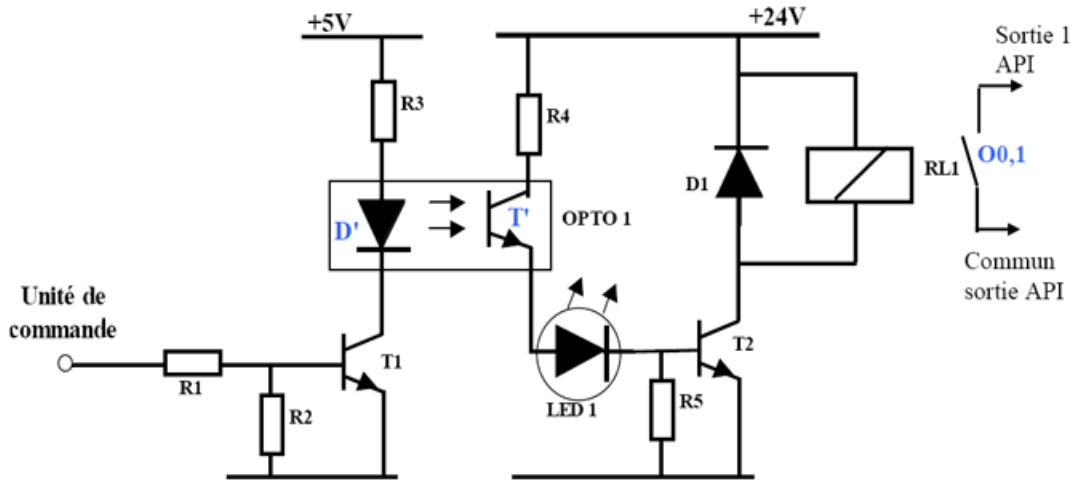


Figure 1.8: Exemple d'une carte de sortie typique d'un API [1]

4.4.6.1.3. Exemple de cartes :

- **Cartes de comptage rapide** : elles permettent d'acquérir des informations de fréquences élevées incompatibles avec le temps de traitement de l'automate. (Signal issu d'un codeur de position)

- **Cartes de commande d'axe** : Elles permettent d'assurer le positionnement avec précision d'élément mécanique selon un ou plusieurs axes. La carte permet par exemple de piloter un servomoteur et de recevoir les informations de positionnement par un codeur. L'asservissement de position pouvant être réalisé en boucle fermée.
- **Cartes d'entrées / sorties analogiques** : Elles permettent de réaliser l'acquisition d'un signal analogique et sa conversion numérique (CAN) indispensable pour assurer un traitement par le microprocesseur. La fonction inverse (sortie analogique) est également réalisée. Les grandeurs analogiques sont normalisées : 0-10V ou 4-20mA.

4.4.6.2. L'alimentation électrique :

Tous les automates actuels sont équipés d'une alimentation 240 V 50/60 Hz, 24 V DC. Les entrées sont en 24 V DC et une mise à la terre doit également être prévue.

4.4.7. Sécurité :

Les systèmes automatisés sont, par nature, source de nombreux dangers (tensions utilisées, déplacements mécaniques, jets de matière sous pression ...).
Placé au cœur du système automatisé, l'automate se doit d'être un élément fiable car un dysfonctionnement de celui-ci pourrait avoir de graves répercussions sur la sécurité des personnes, de plus les coûts de réparation et un arrêt de la production peuvent avoir de lourdes conséquences sur le plan financier.

Aussi, l'automate fait l'objet de nombreuses dispositions pour assurer la sécurité :
- **Contraintes extérieures** : l'automate est conçu pour supporter les différentes contraintes du monde industriel et à fait l'objet de nombreux tests normalisés.

- **Coupures d'alimentation** : l'automate est conçu pour supporter les coupures d'alimentation et permet, par programme, d'assurer un fonctionnement correct lors de la réalimentation (reprises à froid ou à chaud)

- **Mode RUN/STOP** : Seul un technicien peut mettre en marche ou arrêter un automate et la remise en marche se fait par une procédure d'initialisation (programmée)

- **Contrôles cycliques** :

- Procédures d'autocontrôle des mémoires, de l'horloges, de la batterie, de la tension d'alimentation et des entrées / sorties
- Vérification du temps de scrutation à chaque cycle appelée *Watchdog (chien de garde)*, et enclenchement d'une procédure d'alarme en cas de dépassement de celui-ci (réglé par l'utilisateur)

- **Visualisation** : Les automates offrent un écran de visualisation où l'on peut voir l'évolution des entrées / sorties

Les normes interdisent la gestion des arrêts d'urgence par l'automate ; celle-ci doit être réalisée en technologie câblée.

4.4.8. **Réseaux d'automates : [1]**

4.4.8.1.**Principe**

Avec le développement des systèmes automatisés et de l'électronique, la recherche de la baisse des coûts et la nécessité actuelle de pouvoir gérer au mieux la production et a partir du moment où tous les équipements sont de type informatique, il devient intéressant de les interconnecter à un mini-ordinateur ou à un automate de supervision.

L'interconnexion entre deux automates peut être réalisée très simplement en reliant une ou plusieurs sorties d'un automate à des entrées de l'autre et vice-versa (Figure 1.9).

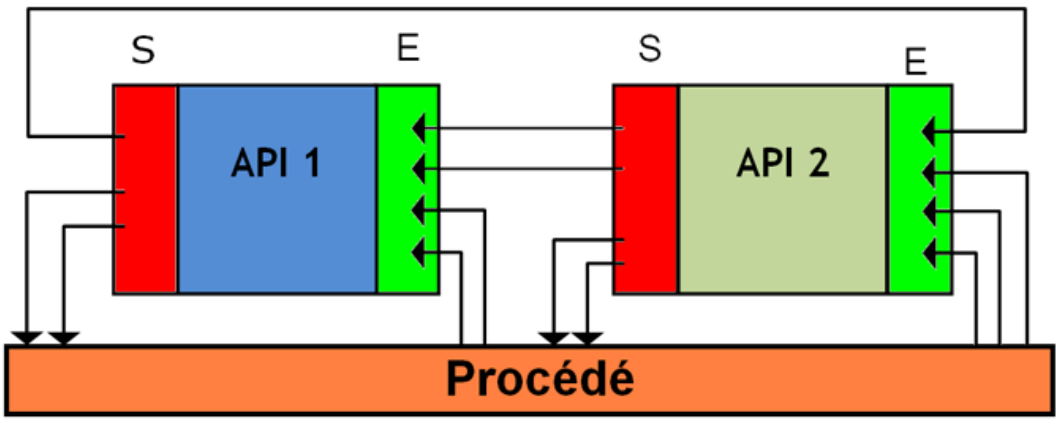


Figure 1.9: Interconnexion simple (Entrées/Sorties) entre deux automates (API) [1]

Cette méthode ne permet pas de transférer directement des variables internes d'un automate sur l'autre, de sorte que celles-ci doivent être converties par programme en variables de sortie avant leur transfert. Elle devient coûteuse en nombre d'entrées/sorties mobilisé pour cet usage et lourde du point de vue du câblage, lorsque le nombre de variables qui doivent être échangées devient important.

4.4.8.2.**Bus de terrain**

Pour diminuer les coûts de câblage des entrées / sorties des automates, sont apparus les bus de terrains. L'utilisation de blocs d'entrées / sorties déportés à permis tout d'abord de répondre à cette exigence.

Les interfaces d'entrées/sorties sont déportées au plus près des capteurs. Avec le développement technologique, les capteurs, détecteurs ... sont devenus « intelligents » et permettent de se connecter directement à un bus.

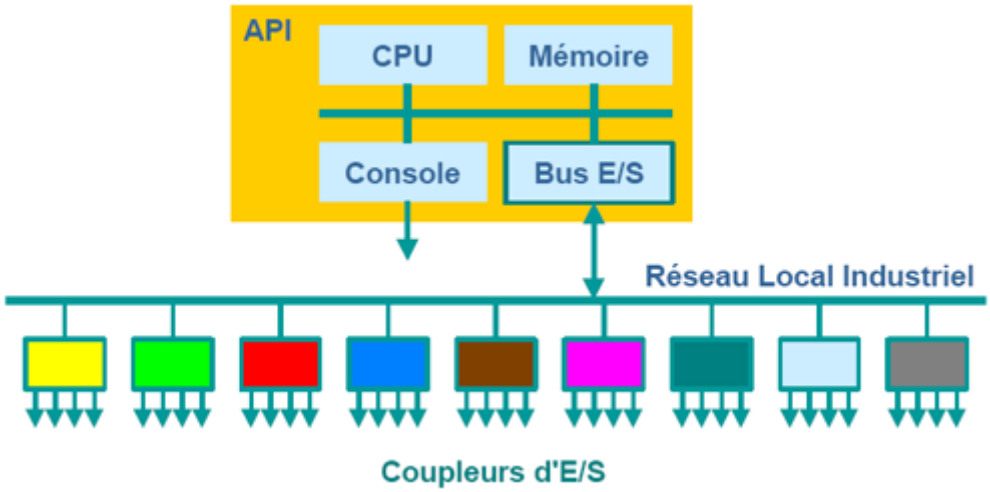


Figure 1.10: Interconnexion par entrées/sorties déportées [1]

Plusieurs protocoles de communication et des standards sont apparus pour assurer le "multiplexage" de toutes les informations en provenance des capteurs / préactionneurs, par exemple le bus ASi (Actuators Sensors interface) est un bus de capteurs/actionneurs de type Maître / Esclave qui permet de raccorder 31 esclaves (capteurs ou pré actionneurs) sur un câble spécifique (deux fils) transportant les données et la puissance.
Ce bus est totalement standardisé et permet d'utiliser des technologies de plusieurs constructeurs

Avantages des bus de terrain :

- Réduction des coûts de câblage et possibilité de réutiliser le matériel existant
- Réduction des coûts de maintenance

Inconvénients des bus de terrain :

- Taille du réseau limitée
- Latence dans les applications à temps critique
- Coût global

4.4.8.3.**Topologie des réseaux d'automates :**

4.4.8.3.1. Réseau en étoile :

Un centre de traitement commun échange avec chacune des autres stations. Deux stations ne peuvent pas échanger directement entre elles (Figure 1.11). Exemple le réseau de terrain BITBUS de la société INTEL

Les avantages :

- Grande vitesse d'échange.
- Différent types de supports de transmission.
- Pas de gestion d'accès au support.

Les inconvénients :

- Coût global élevé.
- Evolutions limitées.
- Tout repose sur la station centrale.

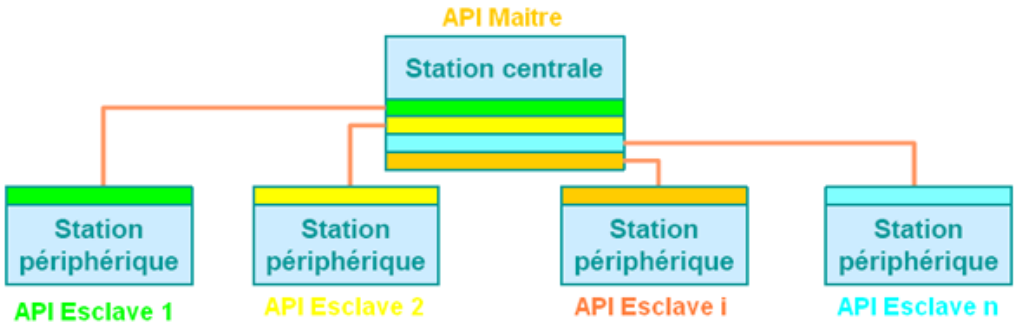


Figure 1.11: Interconnexion par entrées/sorties déportées [1]

4.4.8.3.2. Réseau en anneau :

Chaque station peut communiquer avec sa voisine. Cette solution est intéressante lorsqu'une station doit recevoir des informations de la station précédente ou en transmettre vers la suivante (Figure 1.12).

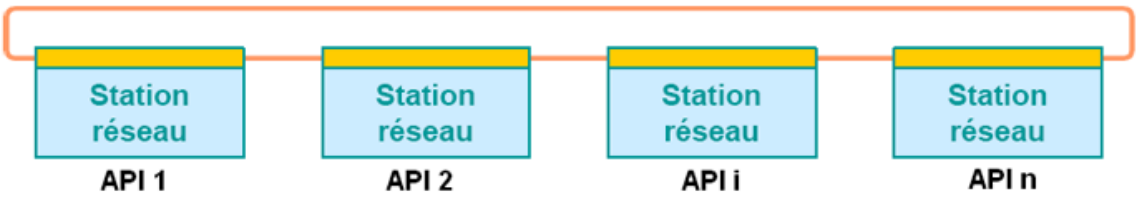


Figure 1.12: Topologie Anneau [1]

Avantages :

- Signal régénéré donc fiable.
- Contrôle facile des échanges (le message revient à l'émetteur).

Inconvénients :

- Chaque station est bloquante.
- Une extension interrompe momentanément le réseau.

4.4.8.3.3. Réseau hiérarchisé :

C'est la forme de réseaux la plus performante. Elle offre une grande souplesse d'utilisation, les informations pouvant circuler entre-stations d'un même niveau ou circuler de la station la plus évoluée (en général un calculateur) vers la plus simple, et réciproquement (Figure 1.13).

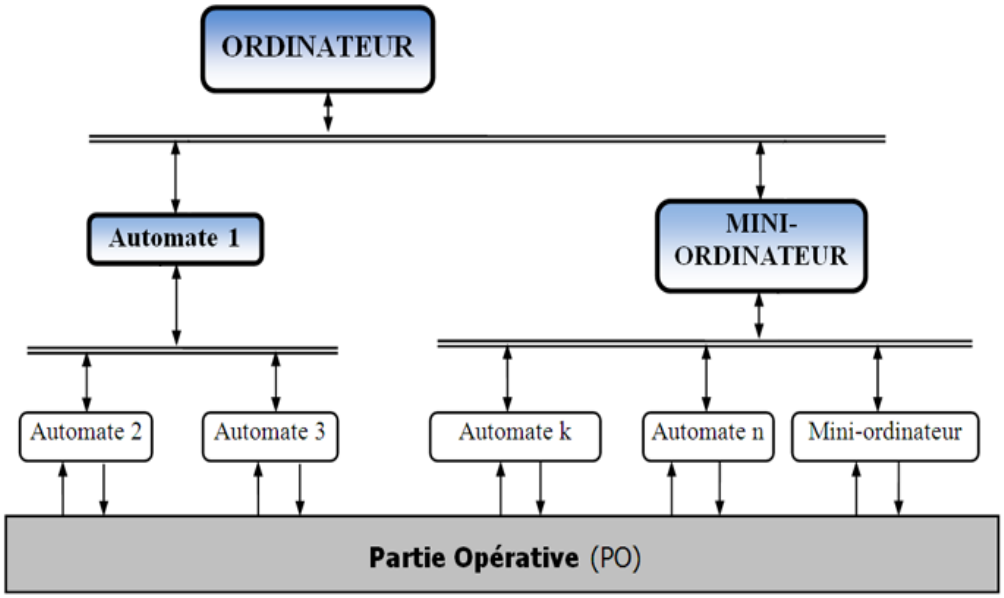


Figure 1.13 : Réseau hiérarchisé

4.4.9. Critères de choix d'un automate

Le choix d'un automate programmable est généralement basé sur :

- Nombre et type d'entrées / sorties : le nombre et le type de cartes peut avoir une incidence sur le nombre de racks dès que le nombre d'entrées / sorties nécessaires devient élevé.

- Type de processeur : la taille mémoire, la vitesse de traitement et les fonctions spéciales offertes par le processeur permettront le choix dans la gamme souvent très étendue.

- Fonctions ou modules spéciaux : certaines cartes (commande d'axe, pesage ...) permettront de "soulager" le processeur et devront offrir les caractéristiques souhaitées (résolution, ...).

- Fonctions de communication : l'automate doit pouvoir communiquer avec les autres systèmes de commande (API, supervision ...) et offrir des possibilités de communication avec des standards normalisés (Profibus ...).

4.5.STEP 7, logiciel de programmation des API Siemens

4.5.1. PRESENTATION

La programmation STEP 7 est une programmation structurée dans des blocs qui sont les blocs d'organisation, les fonctions, les blocs fonctionnels, les blocs de données et blocs d'alarmes cycliques et erreurs.

4.5.2. Parties essentielles du STEP7 :

4.5.2.1.Gestionnaire de projets SIMATIC

Le gestionnaire de projets SIMATIC gère toutes les données relatives à un projet d'automatisation S7 sur lequel elles ont été créées. Il démarre automatiquement les applications requises pour le traitement des données sélectionnées

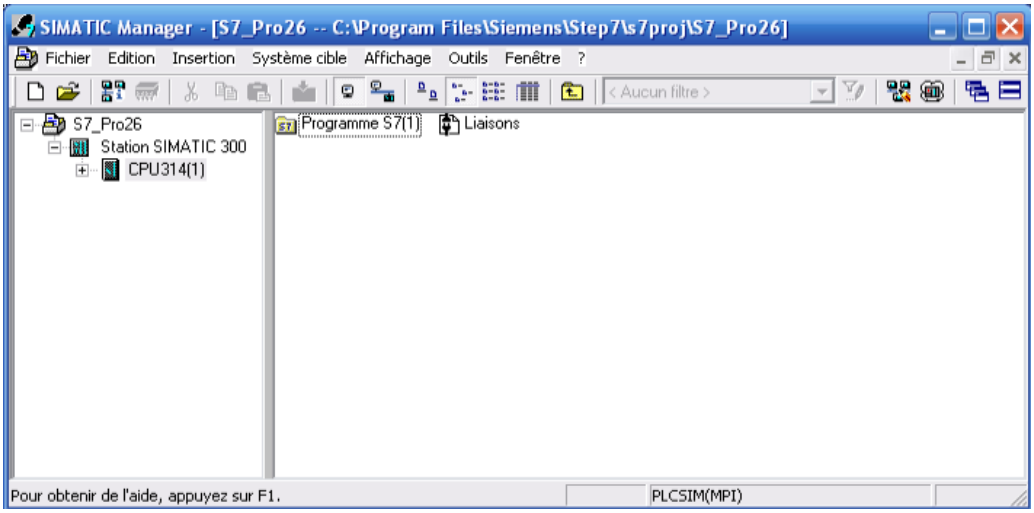


Figure 1.14 : Le gestionnaire de projet STEP 7

4.5.2.2.Edition des mnémoniques :

Les mnémoniques sont des noms que l'on peut donner aux variables globales afin de faciliter la programmation en affectant des noms parlant plus faciles à retenir. Ils améliorent la lisibilité du programme et servent également de documentation. On peut voir un exemple dans la figure 1.15

4.5.2.3.Langages de programmation :

Les langages de programmation CONT, LIST, SCL, S7-GRAPGH et LOG pour S7-300/400 font partie intégrante du logiciel de base.

4.5.2.4.Configuration matérielle :

Dans une table de configuration, nous définissons les modules que nous allons mettre en œuvre dans notre solution d'automatisation ainsi que les adresses permettant d'y accéder depuis le programme utilisateur. Nous pouvons en outre y paramétrer les caractéristiques des modules (les manipulations de base pour la configuration matérielle).

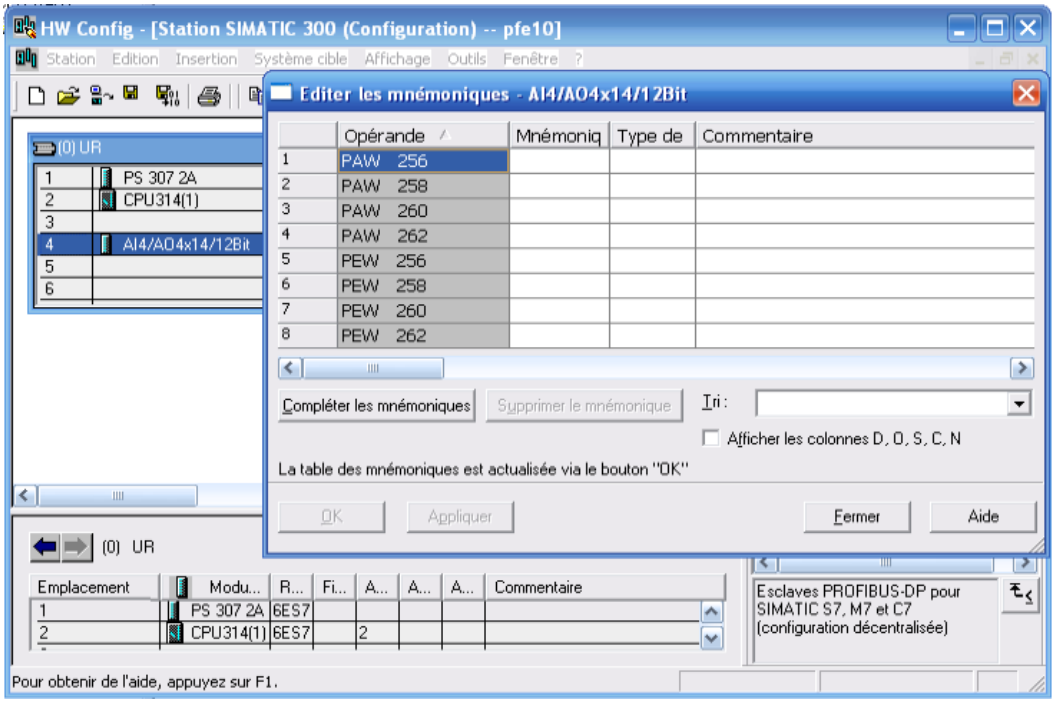


Figure 1.15 : Edition des Mnémoniques.

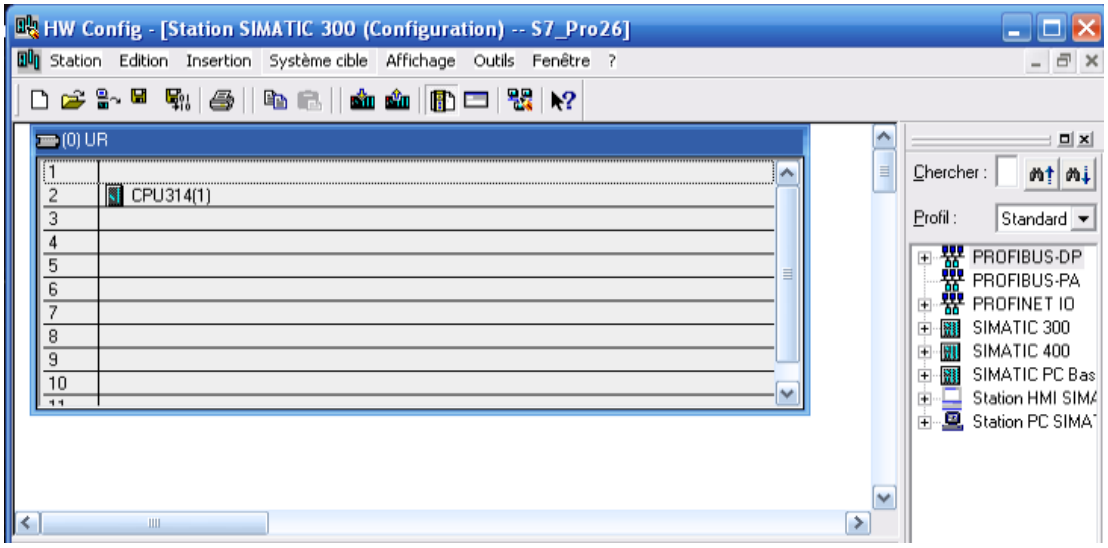


Figure 1.16 : Configuration matérielle pour la station S300

4.5.2.5.**Test de programmes**

Pour effectuer un test ou une simulation, nous avons la possibilité d'afficher les valeurs de variables depuis notre programme utilisateur ou depuis une CPU, d'affecter des valeurs à ces variables et de créer une table des variables que nous souhaitons afficher ou forcer. On peut voir un exemple de simulation dans la figure 1.17.

4.5.2.6.**Surveillance du fonctionnement, diagnostic du matériel [3]**

On détermine les causes du défaut d'un module en affichant des informations en ligne relatives à ce module. On détermine les causes d'un défaut dans le déroulement d'un programme utilisateur à l'aide de la mémoire tampon de diagnostic et du contenu des piles. On peut en outre vérifier si un programme utilisateur est exécutable sur une CPU donnée.

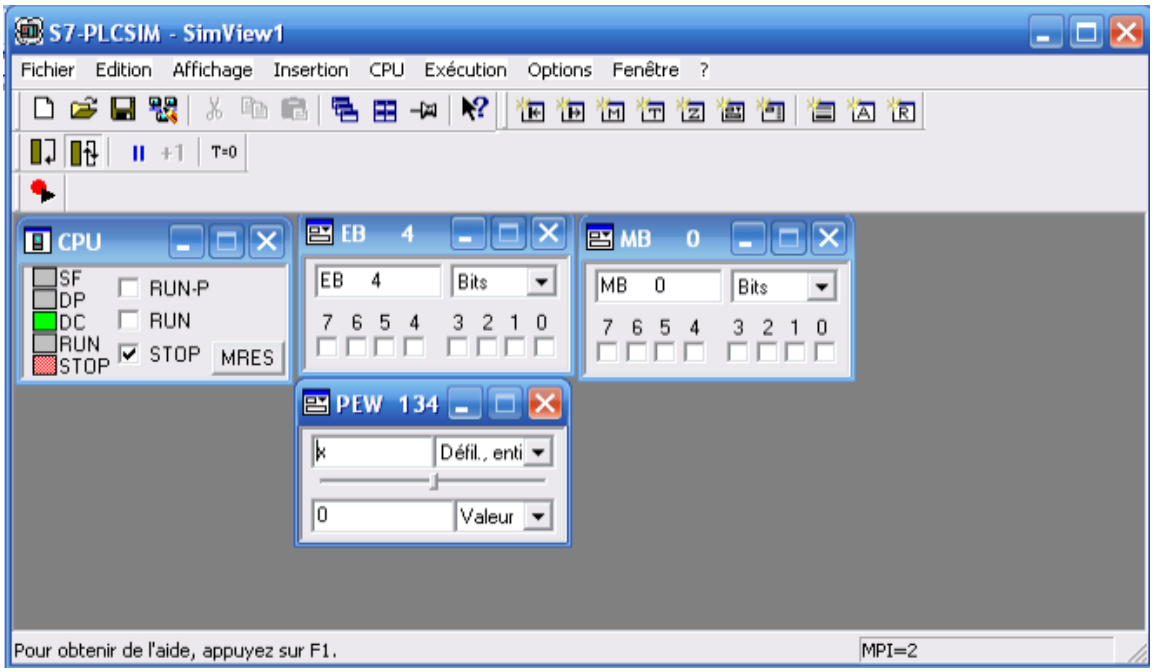


Figure 1.17 : Simulation d'un programme avec le logiciel PLC-SIM

4.5.2.7.**Configuration de réseaux et de liaisons de communication : [3]**

La condition requise à l'établissement d'une communication est l'existence d'un réseau préalablement configuré. Nous devons à cet effet créer les réseaux auxiliaires nécessaires à notre projet, définir leurs propriétés et pour les stations mises en réseau, les caractéristiques de connexion au réseau ainsi que, le cas échéant, les liaisons de communication requises.

Conclusion :

De part la mobilité, la flexibilité de son architecture, la facilité de sa programmation, de sa connexion et de son adaptation dans les milieux industriels, l'automate programmable est devenu un produit incontournable dans les systèmes automatisés de production.
Néanmoins la diversité des possibilités qu'il offre dans sa mise en œuvre et son coût ne constituent pas des conditions suffisantes lors de l'élaboration d'une solution d'automatisme.

Munis d'un logiciel très performant, les automates programmables S7-300 sont exploitables par le biais du logiciel Step7. On a pu voir les différentes possibilités offertes : simulation des systèmes séquentiels et continus, calcul numérique.
Dans ce qui va suivre, on montrera que la programmation des automates S7-300 avec Step7 n'est pas suffisante pour le contrôle des procédés industriels. Pour y arriver, on devra complémenter le programme par une interface graphique sous WinC

Chapitre 2

Logiciel de supervision WinCC

Introduction :

Les API ont pour fonctions de remplir des tâches de commande en élaborant des actions en suivant une algorithmique appropriée à partir d'informations données par des capteurs.

Dans ce qui va suivre nous allons présenter le logiciel d'ingénierie et de supervision SIMATIC WinCC.

L'utilisateur de WinCC dispose de moyens de visualisation des procédés (synoptiques, des graphes, bargraphes…) qui lui permettent de contrôler facilement et avec clarté toutes les opérations d'automatisation.

1.3. Présentation de WinCC

1.3.2. Description générale :

Lorsque la complexité des processus augmente et que les machines et installations doivent répondre à des spécifications de fonctionnalité toujours plus sévères, l'opérateur a besoin d'un maximum de transparence. Cette transparence s'obtient au moyen de l'Interface Homme-Machine (IHM). [4]

Un système IHM constitue l'interface entre l'homme (opérateur) et le processus (machine /Installation). Le contrôle proprement dit du processus est assuré par le système d'automatisation. Il existe par conséquent une interface entre l'opérateur et WinCC flexible (sur le pupitre opérateur) et une interface entre WinCC flexible et le système d'automatisation. Un système IHM se charge des tâches suivantes :

- Représentation du processus :

  Le processus est représenté sur le pupitre opérateur. Lorsqu'un état du processus évolue p. ex., l'affichage du pupitre opérateur est mis à jour.

- Commande du processus :

  L'opérateur peut commander le processus via l'interface utilisateur graphique. Il peut p. ex. définir une valeur de consigne pour un automate ou démarrer un moteur.

- Vue des alarmes :

  Lorsque surviennent des états critiques dans le processus, une alarme est immédiatement déclenchée, p. ex. lorsqu'une valeur limite est franchie.

- Archivage de valeurs processus et d'alarmes :

  Les alarmes et valeurs processus peuvent être archivées par le système IHM. On peut ainsi documenter la marche du processus et accéder ultérieurement aux données de la production écoulée.

- Documentation de valeurs processus et d'alarmes :

  Les alarmes et valeurs processus peuvent être éditées par le système IHM sous forme de journal. On peut ainsi consulter les données de production à la fin d'une équipe p. ex.

- Gestion des paramètres de processus et de machine :

Les paramètres du processus et des machines peuvent être enregistrés au sein du système IHM dans des recettes. Ces paramètres sont alors transférables en une seule opération sur l'automate pour démarrer la production d'une variante du produit p. ex.

1.3.3. **Fonctionnement du WinCC : [2]**

WinCC est un système modulaire. Ses éléments de base sont le logiciel de configuration (CS) et le logiciel Runtime (RT) :

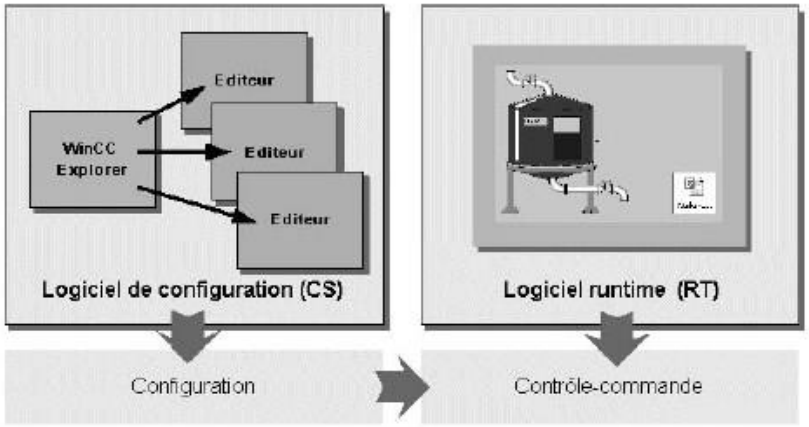


Figure 2.1 : Structure interne du WinCC

a. Logiciel de configuration (CS) :

Après démarrage de WinCC, l'écran affiche l'explorateur WinCC Explorer. WinCC Explorer est le noyau du logiciel de configuration. WinCC Explorer affiche la structure complète du projet et permet de gérer le projet. La configuration s'effectue à l'aide d'éditeurs spécifiques que nous pouvons ouvrir à partir de WinCC Explorer. Chaque éditeur permet de configurer un sous-système particulier de WinCC.

Les principaux sous-systèmes de WinCC sont :

- Le système graphique (Graphics Designer)
- Le système de signalisation (Alarm Logging)
- Le système d'archivage (Tag Logging)
- Le système de journalisation (Report Designer)
- Le gestionnaire des utilisateurs (User Administrator)
- La communication – elle se configure directement sous WinCC Explorer
- Toutes les données de configuration sont enregistrées dans la base de données CS.

b. Logiciel Runtime :

Le logiciel runtime permet à l'opérateur d'assurer la conduite du process. Les tâches incombant au logiciel runtime sont les suivantes :

- Lecture des données enregistrées dans la base de données CS
- Affichage des vues à l'écran

- Communication avec les automates programmables
- Archivage des données actuelles de Runtime, par exemple des valeurs de processus et événements de signalisation

Conduite du processus, par exemple spécification de consignes, mise en marche/arrêt.

1.3.4. Eléments de l'interface utilisateur de WinCC flexible :

L'environnement de travail de WinCC flexible se compose de plusieurs éléments. Certains de ces éléments sont liés à des éditeurs particuliers et uniquement visibles lorsque cet éditeur est activé.

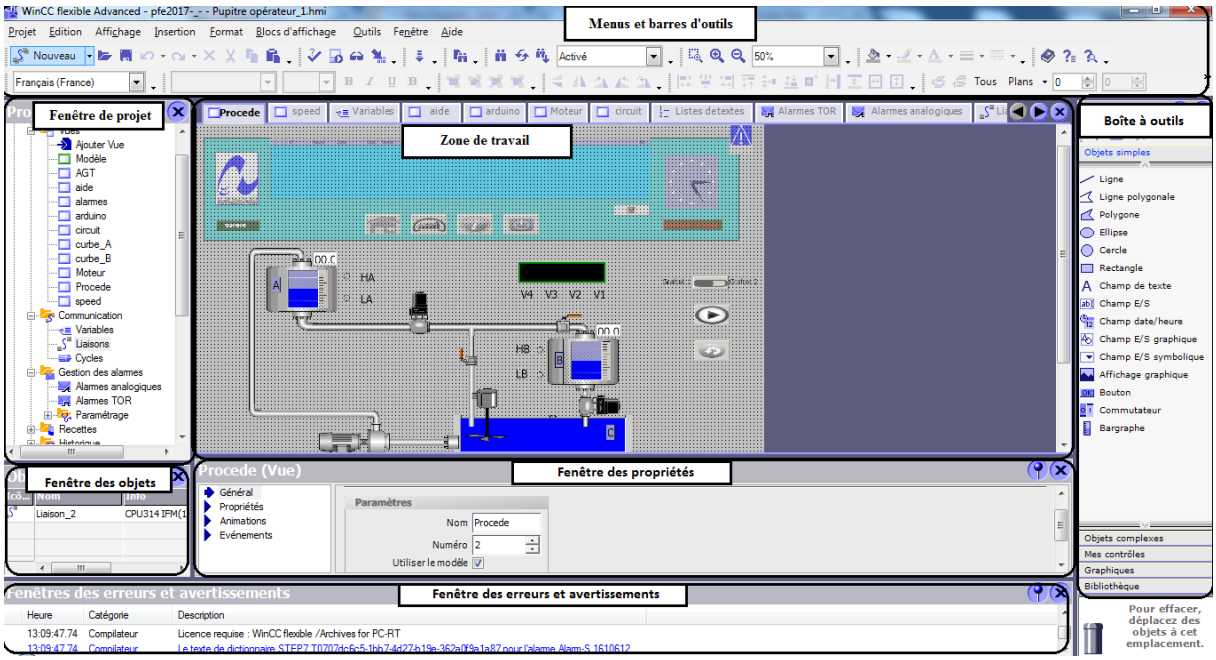


Figure 2.2 : Eléments de l'interface WinCC

WinCC flexible se compose des éléments suivants : [5]

a- Menus et barres d'outils

Nous trouverons dans les menus et barres d'outils toutes les fonctions dont nous avons besoin pour la configuration de notre pupitre opérateur. Lorsqu'un éditeur est actif, les commandes de menu ou barres d'outils correspondantes sont visibles.
Lorsque nous positionnons le pointeur de la souris sur une commande, nous obtenons un info-bulles correspondant à chaque fonction.

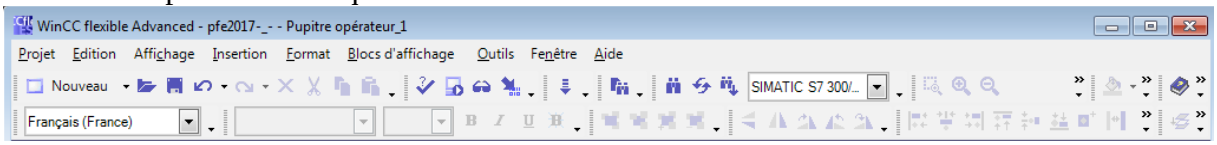


Figure 2.3 : Barre d'outils du logiciel WinCC flexible Advanced

Les menus suivants sont disponibles sous WinCC flexible :

Menu	Descriptif technique
"Projet"	Contient des commandes de gestion de projets.
"Edition"	Contient des commandes servant à utiliser le presse-papiers ainsi que des fonctions de recherche.
"Affichage"	Contient des commandes permettant d'ouvrir et de fermer des éléments ainsi que des paramètres des fonctions zoom et plans. Un élément fermé peut être rouvert via le menu "Affichage".
"Insertion"	Contient des commandes pour l'insertion de nouveaux objets.
"Format"	Contient des commandes servant à disposer et à formater des objets de vue.
"Blocs d'affichage"	Contient des commandes servant à créer et éditer des blocs d'affichage.
"Outils"	Contient, entre autres, des commandes servant à changer de langue d'interface et à configurer les paramètres de base de WinCC flexible.
"Script"	Contient des commandes permettant de synchroniser et de vérifier la syntaxe de scripts.
"Fenêtre"	Contient des commandes de gestion de plusieurs vues de la zone de travail, permettant p. ex. de changer de vue.
"Aide"	Contient des commandes d'accès aux fonctions d'aide.

Figure 2.4 : Description des éléments de Menu Sous WinCC flexible

b- Zone de travail

Dans la zone de travail, nous éditons les données de projet soit sous forme de tableau, ce qui est le cas des variables p. ex., soit sous forme graphique, ce qui est le cas des vues de processus p. ex. Chaque éditeur ouvert est représenté dans la zone de travail dans un onglet particulier.

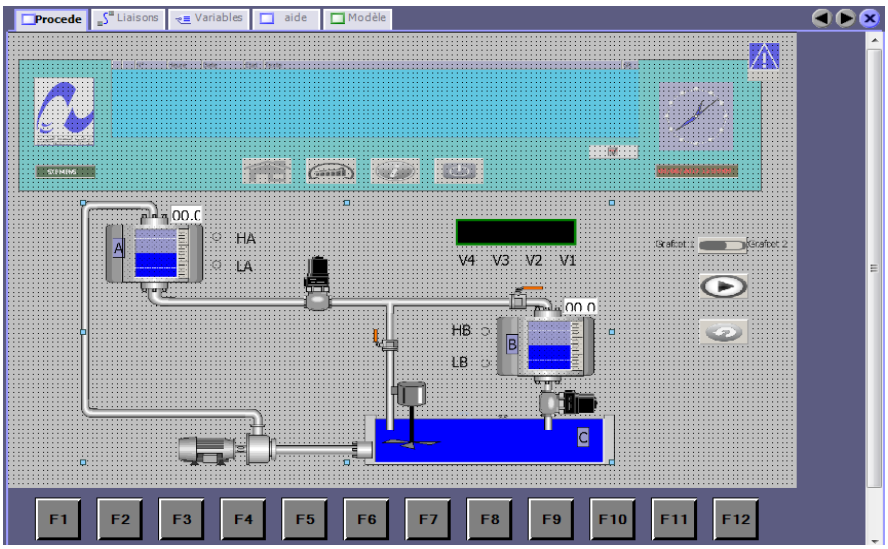


Figure 2.5 : La forme graphique d'une Vue.

c- Fenêtre de projet

La fenêtre du projet est le poste central de traitement du projet. Tous les éléments et tous les éditeurs disponibles d'un projet sont affichés sous forme d'arborescence dans la fenêtre du projet. (Figure 2.6)

La fenêtre de projet sert à créer des objets et à les ouvrir pour les éditer. Nous pouvons créer des dossiers pour structurer les objets de notre projet. Nous pouvons ouvrir pour chaque objet un menu contextuel qui regroupe les principales commandes et nous pouvons aussi accéder aux paramètres du pupitre, à la localisation et à la gestion de versions.

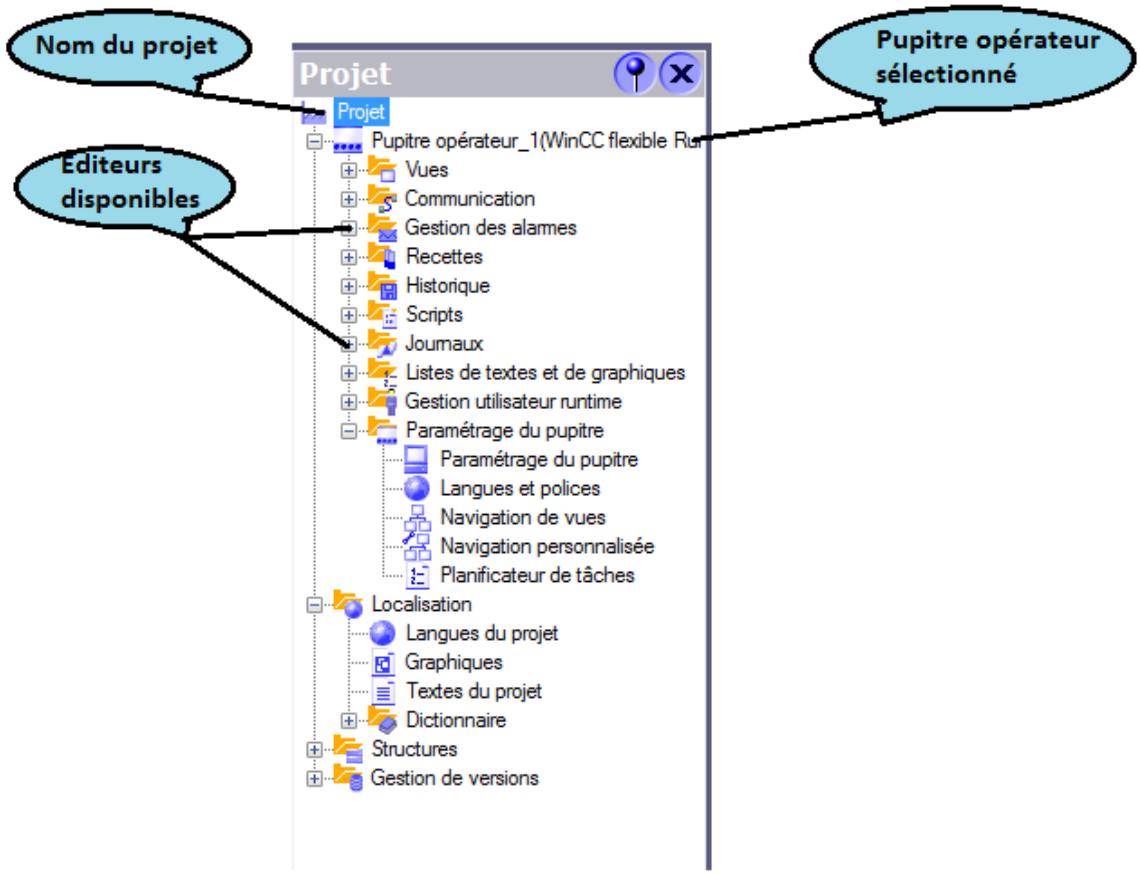


Figure 2.6 : Fenêtre de projet

d- Fenêtre des propriétés

La fenêtre des propriétés permet de modifier les propriétés d'un objet sélectionné dans la zone de travail. Le contenu de la fenêtre des propriétés dépend de l'objet sélectionné.

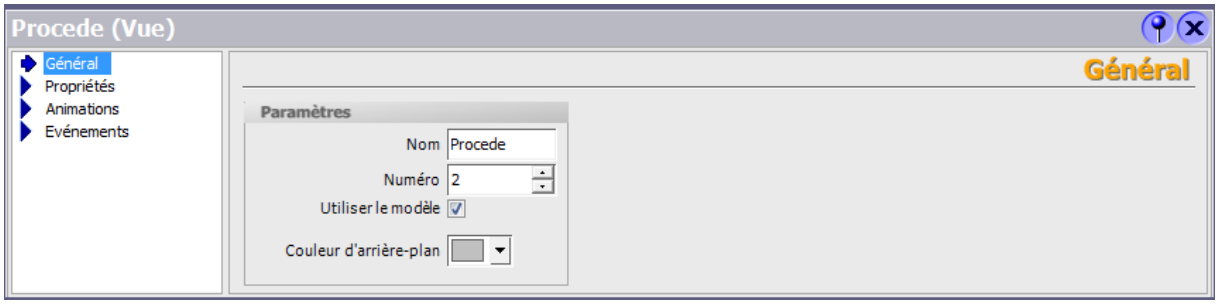


Figure 2.7 : Fenêtre des propriétés

La fenêtre des propriétés affiche les propriétés de l'objet sélectionné classées par catégories. Aussitôt que nous quittons une zone de saisie, les modifications de valeurs effectuées sont actives.

e- Bibliothèque

La bibliothèque fait partie de la fenêtre d'outils, elle est un lieu central d'enregistrement des objets fréquemment utilisés. Sous WinCC flexible, on distingue la bibliothèque globale de la bibliothèque spécifique au projet :

- Bibliothèque globale

  La bibliothèque globale n'est pas enregistrée avec le projet dans la base de données mais sous forme de fichier. Le fichier enregistré est stocké par défaut dans le répertoire d'installation de WinCC flexible. La bibliothèque globale est disponible pour tous les projets.

- Bibliothèque spécifique projet

  La bibliothèque de projet qui est enregistrée avec les données de projet dans la base de données, est uniquement disponible dans le projet où elle a été créée.

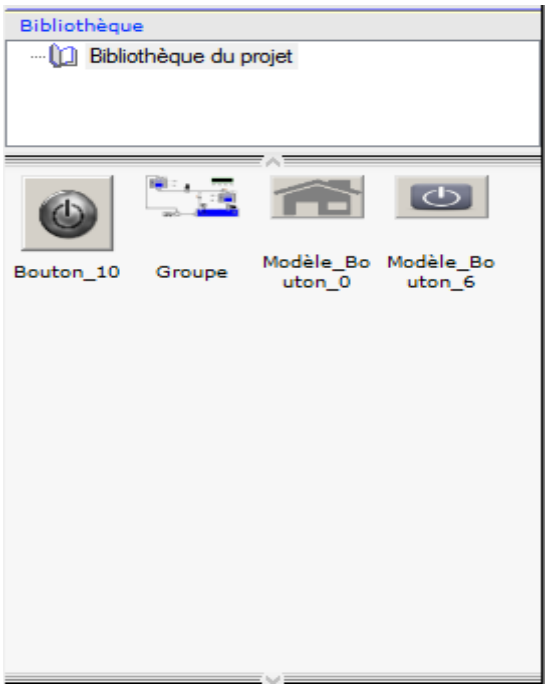


Figure 2.8 : Eléments de la bibliothèque

f- Fenêtre des erreurs et avertissements

La fenêtre des erreurs et avertissements affiche les événements système générés p. ex. lors du test d'un projet.

Dans la fenêtre des erreurs et avertissements, les événements système sont affichés par défaut dans leur ordre d'apparition. Les catégories désignent respectivement le module WinCC flexible qui a généré un événement système. Les alarmes système de la catégorie "Compilateur" sont p. ex. générées durant le contrôle de cohérence.

La fenêtre des erreurs et avertissements affiche tous les événements système se rapportant à la dernière action. A chaque nouvelle action, tous les événements système précédents sont écrasés.

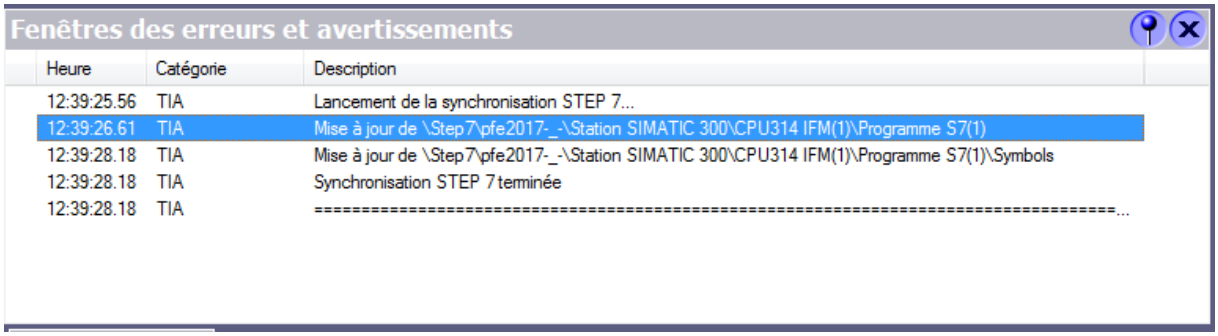


Figure 2.9 : Fenêtre des erreurs et avertissements.

g- Fenêtre des objets

Lorsque nous sélectionnons des dossiers ou des éditeurs dans la fenêtre de projet, leur contenu s'affiche dans la fenêtre des objets comme le montre la figure 2.10.

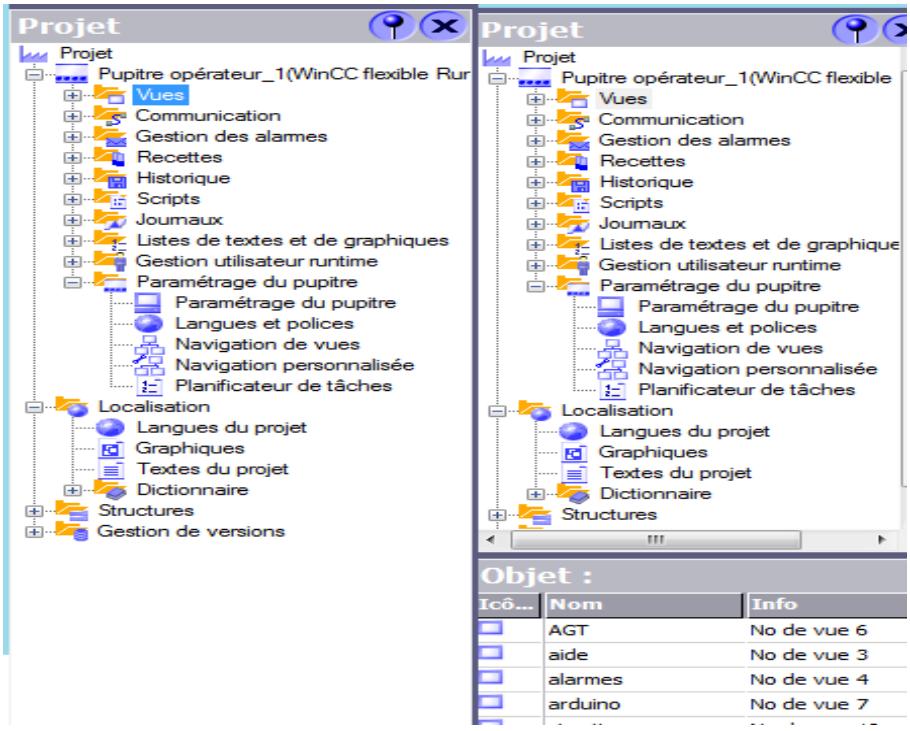


Figure 2.10 : Fenêtre des objets

1.4. Etapes de conception d'un projet sous WinCC :

Nous allons aborder dans cette partie les différentes étapes pour aboutir à un projet et ceci en expliquant chaque étape.

1.4.2. Création d'un projet :

Après avoir lancé le SIMATIC WinCC flexible, on clique sur : « Créer un projet vide » ; une fenêtre de dialogue affiche alors les différents types de pupitre à choisir :

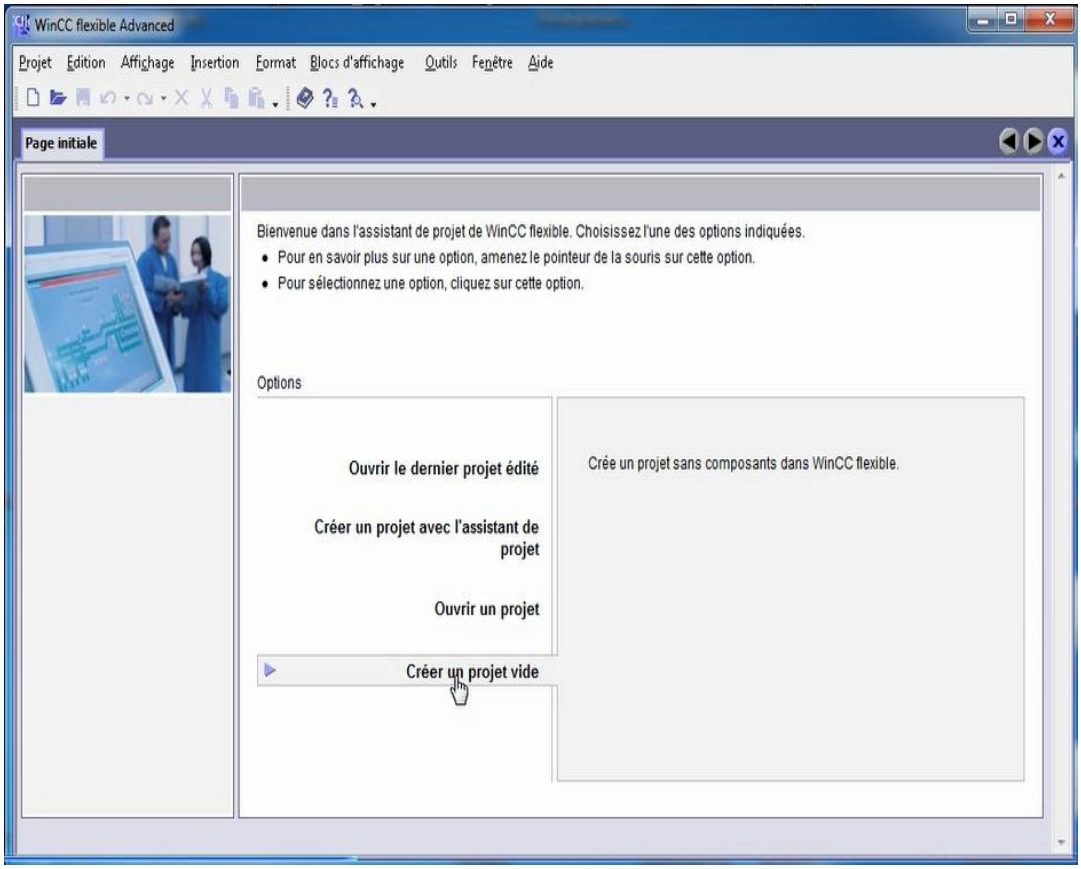


Figure 2.11 : Créer un nouveau WinCC flexible Advanced

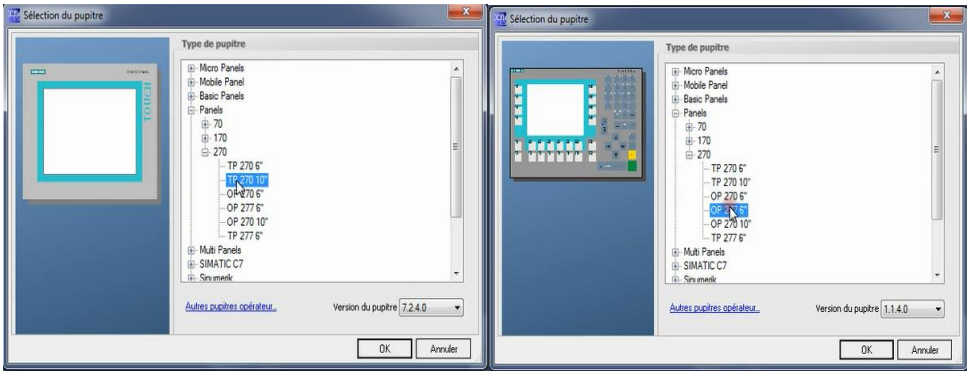


Figure 2.12 : Sélection du pupitre

On choisira pour notre projet ‘WinCC flexible Runtime’

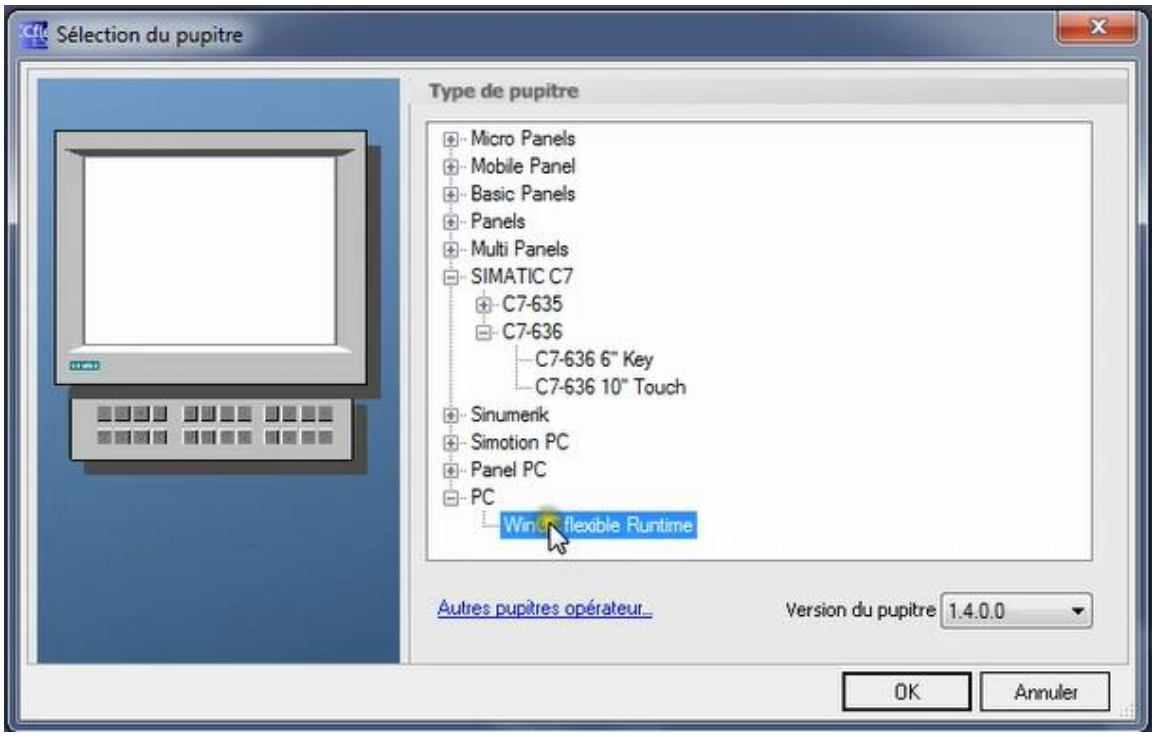


Figure 2.13 : configurer le PC comme pupitre

Avant de commencer il faut d'abord configurer la résolution de l'écran pour assurer une bonne supervision.
Dans la fenêtre de projet, on clique sur paramètre de pupitre et on coche sur 'mode plein écran' et on choisit une résolution d'écran correspondante à notre résolution d'écran du système.

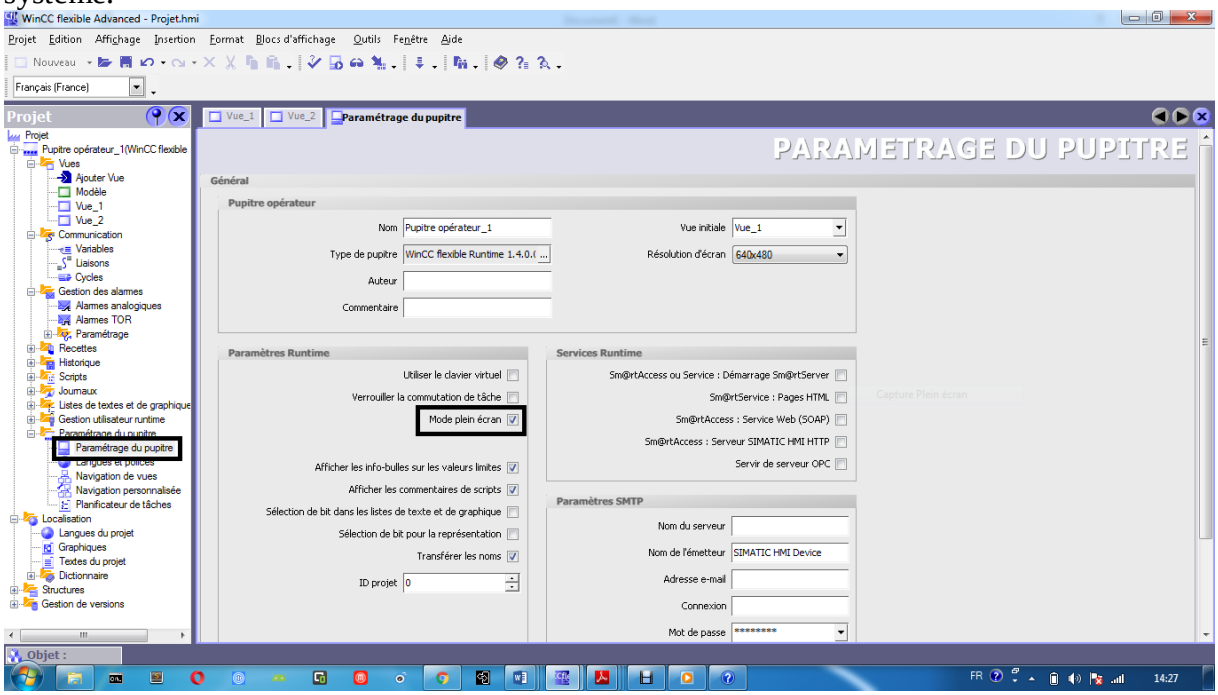


Figure 2.14 : Paramétrage du pupitre

- **WinCC flexible Runtime :**

WinCC flexible Runtime est un logiciel performant et facile à utiliser pour la visualisation du processus des projets créés avec le logiciel de configuration WinCC flexible Advanced.
WinCC flexible Runtime est conçu pour la visualisation et l'utilisation de machines et de

petites installations. Le logiciel Runtime se distingue par son interface utilisateur entièrement graphique basée sur la technique des fenêtres. Il permet grâce à des temps de réaction rapides une conduite de processus sûre, ainsi qu'une collecte sûre des données.

- **Intégrer dans le projet STEP 7**

Nous allons maintenant faire la communication entre WinCC et le projet STEP 7 qui on a déjà crée

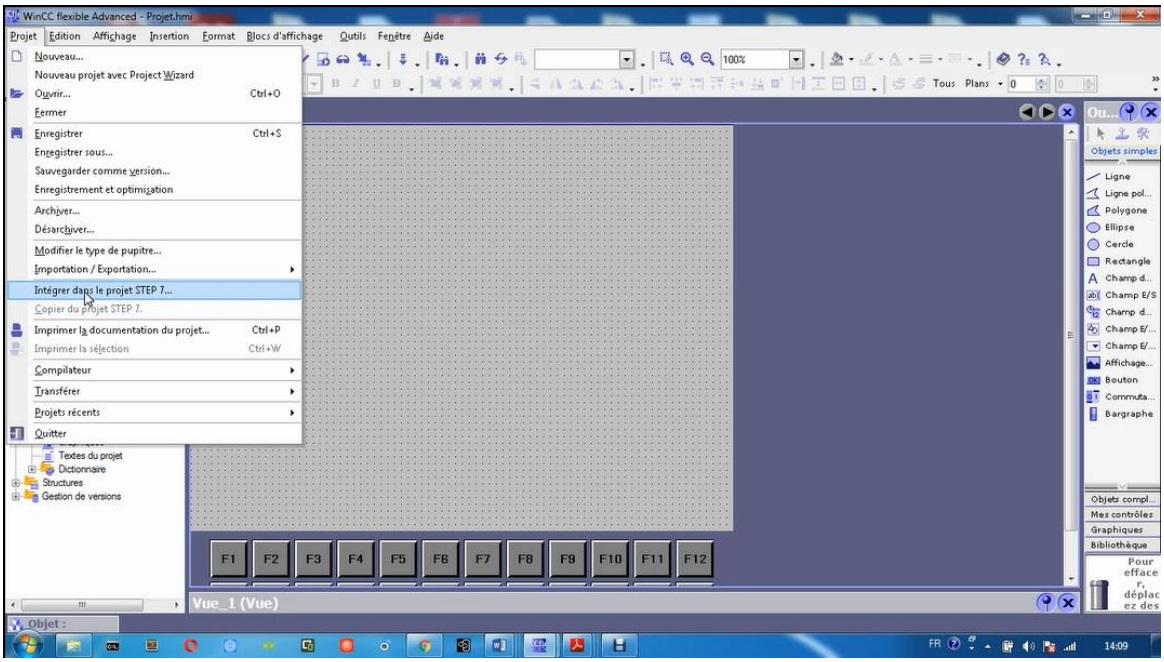


Figure 2.15 : intégration au projet STEP7

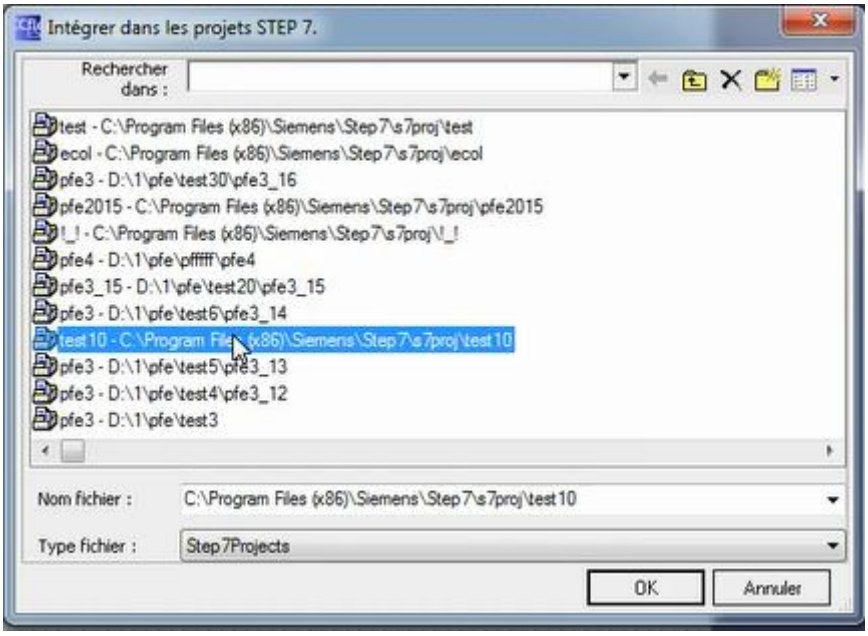


Figure 2.16 : intégration au projet STEP7

On crée ensuite une liaison de type ‘MPI’ pour établir la communication entre l’interface et l'automate, et pour le faire on suit les étapes qui sont cités dans le vidéo (disponible sur notre chaine Youtube [6]).

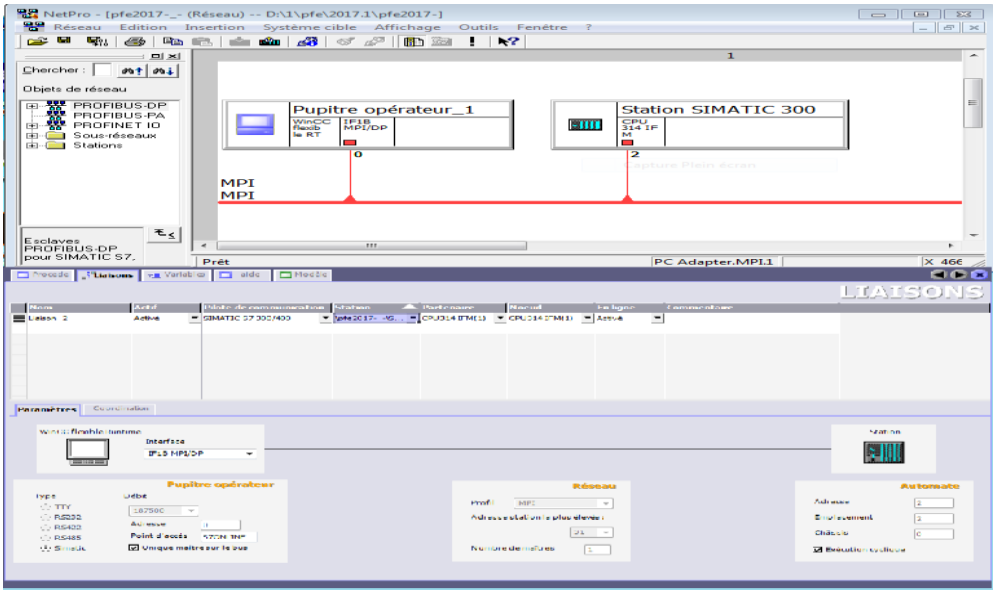


Figure 2.17 : Etablir une liaison MPI entre le pupitre opérateur et la station SIMATIC

1.4.3. Création d'une vue de process :

Dans WinCC flexible, on crée des vues pour le contrôle-commande des machines et d'installations. Pour créer des vues, nous disposons d'objets prédéfinis permettant de représenter notre installation, d'afficher des procédures et de définir des valeurs de processus. (Dans la fenêtre de projets : 'vues' ➔ 'ajouter un vue')

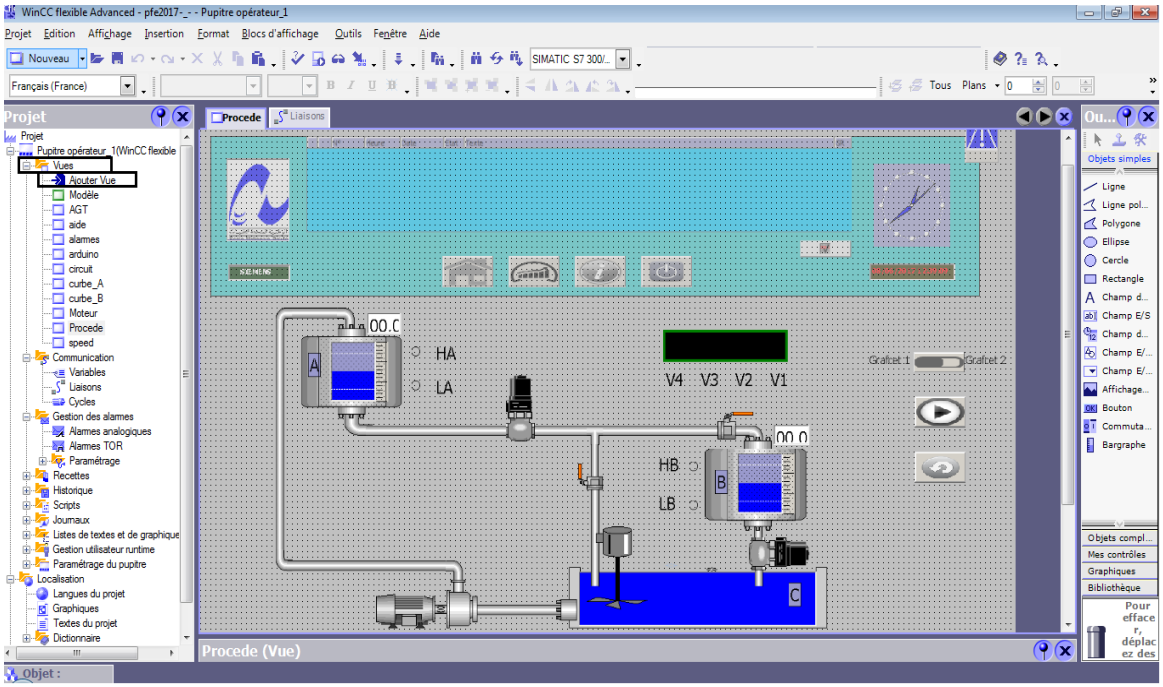


Figure 2.18 : Vue principale du procédé

Une vue peut être composée d'éléments statiques et d'éléments dynamiques.

- ✓ Les éléments statiques : ne changent pas au Runtime, p. ex. le texte et le graphique.

- ✓ Les éléments dynamiques varient en fonction de la procédure, soit à partir de la mémoire de l'automate programmable soit à du pupitre opérateur, sous forme

d'affichages alphanumériques, de courbes et de bargraphes. Même aussi les champs de saisie du pupitre opérateur font également partie des objets dynamiques.

1.4.4. Définition des variables :

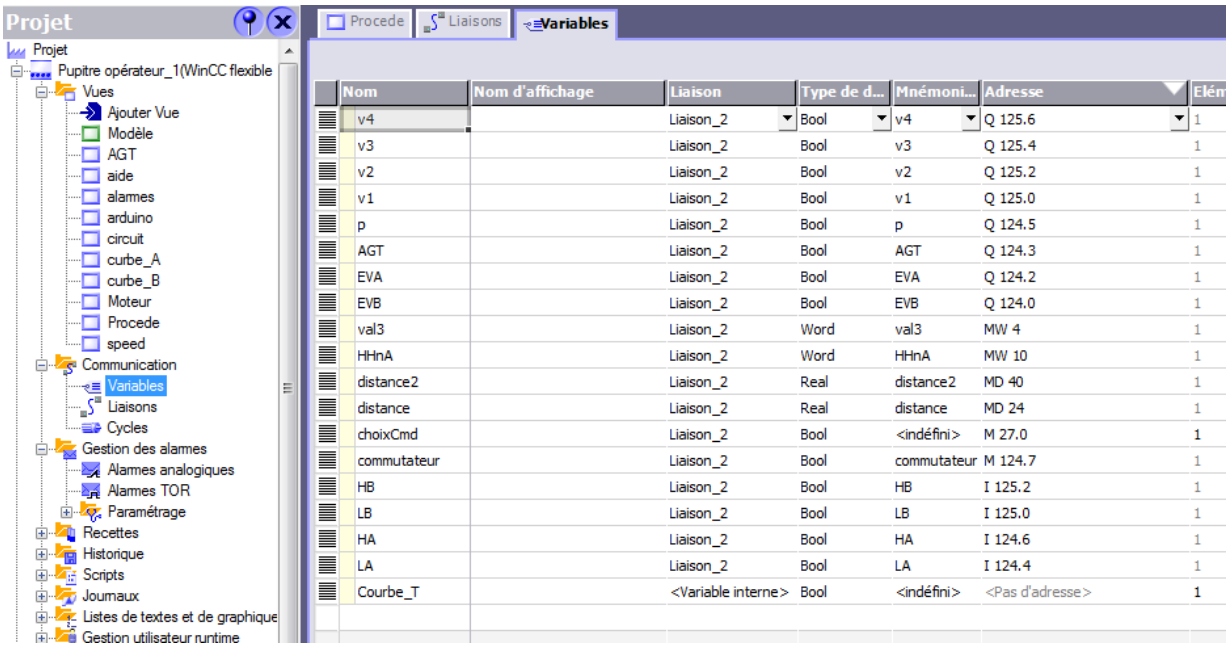
Les variables utilisées dans WinCC représentent soit des valeurs réelles, comme par exemple le taux de remplissage d'un réservoir d'eau, soit des valeurs internes, qui sont calculées où bien simulées en interne par WinCC :

- **Variables externes :**

Les variables externes sont des emplacements en mémoire d'un API ou d'un matériel semblable.
Ainsi le niveau de remplissage du réservoir d'eau est relevé par un capteur de niveau et enregistré dans l'API. Le taux de remplissage est communiqué à WinCC par le canal de communication.

- **Variables internes :**

Les variables internes sont des emplacements en mémoire de WinCC qui assurent les mêmes fonctionnalités qu'un API. Elles peuvent être calculées et modifiées en interne par WinCC.
Les groupes de variables servent à structurer les variables. Toutes les variables peuvent, pour plus de clarté, être rangées dans des groupes de variables.
Pour définir notre variable, on clique dans la fenêtre de projet ➔ communication ➔ variable et on choisit la liaison si le variable est externe sinon on le laisse un variable interne.



Nom	Nom d'affichage	Liaison	Type de d...	Mnémoni...	Adresse	Elén
v4		Liaison_2	Bool	v4	Q 125.6	1
v3		Liaison_2	Bool	v3	Q 125.4	1
v2		Liaison_2	Bool	v2	Q 125.2	1
v1		Liaison_2	Bool	v1	Q 125.0	1
p		Liaison_2	Bool	p	Q 124.5	1
AGT		Liaison_2	Bool	AGT	Q 124.3	1
EVA		Liaison_2	Bool	EVA	Q 124.2	1
EVB		Liaison_2	Bool	EVB	Q 124.0	1
val3		Liaison_2	Word	val3	MW 4	1
HHnA		Liaison_2	Word	HHnA	MW 10	1
distance2		Liaison_2	Real	distance2	MD 40	1
distance		Liaison_2	Real	distance	MD 24	1
choixCmd		Liaison_2	Bool	<indéfini>	M 27.0	1
commutateur		Liaison_2	Bool	commutateur	M 124.7	1
HB		Liaison_2	Bool	HB	I 125.2	1
LB		Liaison_2	Bool	LB	I 125.0	1
HA		Liaison_2	Bool	HA	I 124.6	1
LA		Liaison_2	Bool	LA	I 124.4	1
Courbe_T		<Variable interne>	Bool	<indéfini>	<Pas d'adresse>	1

Figure 2.19 : Fenêtre des variables

1.4.5. Simulation de la vue de process :

Le logiciel de configuration WinCC flexible est fourni avec un simulateur qui permet de tester le projet sans automate. Le simulateur est une application en propre. Il nous permet de contrôler le bon fonctionnement des vues, objets de vue et messages configurés.
Pour simuler le projet terminé, plusieurs possibilités s'offrent :

a- Simulation avec raccordement à l'automate

Nous pouvons simuler notre projet en l'exécutant directement dans runtime. Mais dans ce cas, les variables et les pointeurs de zone ne peuvent fonctionner que si notre PC de configuration

est raccordé à un automate approprié. Si notre ordinateur est raccordé à un automate, Runtime nous permet de réaliser une simulation authentique du pupitre opérateur configuré.

b- Simulation sans raccordement à l'automate

A l'aide du programme de simulation installé en même temps que WinCC flexible Runtime nous pouvons simuler le projet, y compris les variables et les pointeurs de zone sans connexion à un automate. Nous saisissons les paramètres des pointeurs de zone et des variables dans un tableau de simulation qui est lu par WinCC flexible Runtime pendant la simulation.

c- Simulation en fonctionnement intégré

Si nous configurons en mode intégré dans STEP 7, nous pouvons simuler un raccordement à un automate via PLCSIM.
Pour effectuer la simulation avec WinCC flexible Runtime, nous procédons comme suit:
1. on compile notre projet, un fichier génèrent automatiquement portant l'extension *. Fwx sous le même répertoire du projet.
2. on sélectionne dans le menu ‘Projet’ la commande ‘Compilateur’ > Démarrer runtime. Sinon, on clique dans la barre d'outils sur l’icône "Générateur".

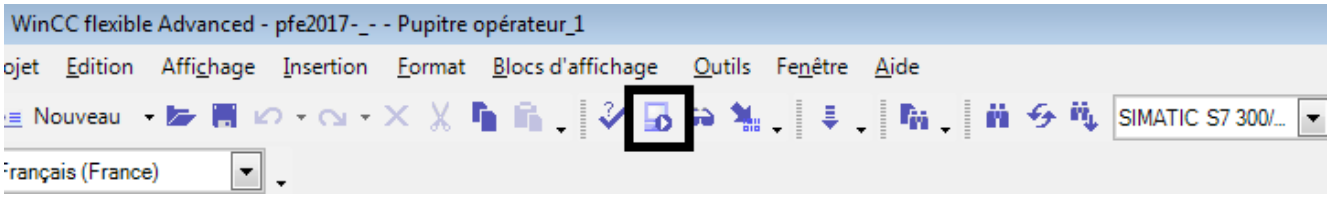


Figure 2.20 : Icône "Générateur"

1.5. Configuration des alarmes : [7]

Dans cette partie nous nous intéressons à la procédure de création d’un système d’alarmes pour le Runtime. Le système d’alarmes informe l'opérateur des états de fonctionnement ou pannes du process et il est créé à l’aide de l’éditeur Alarm Loggin.

Les alarmes configurées dans Alarm Logging sont sorties dans Runtime lorsque l'événement correspondant se produit, par exemple un défaut ou un dépassement de seuil.
WinCC flexible offre les options ci-dessous pour afficher des messages sur un pupitre opérateur :

- **Vue des Alarmes :**
  Une vue des alarmes est configurée pour une vue déterminée. Selon la taille configurée, il peut afficher plusieurs alarmes simultanément. Nous pouvons configurer plusieurs affichages d'alarmes pour des classes d'alarmes différentes et dans des vues différentes.
- **Ligne des Alarmes:**
  La vue des alarmes peut être configuré de manière à ne comporter qu'une seule ligne au Runtime et c'est la dernière alarme qui sera affiché ("Propriétés > Représentation > Mode > Comme ligne d'alarme").
- **Fenêtre des Alarmes:**
  Une vue des alarmes est configurée pour une vue déterminée. Selon la taille configurée, il

peut afficher plusieurs alarmes simultanément. Nous pouvons configurer plusieurs affichages d'alarmes pour des classes d'alarmes différentes et dans des vues différentes.

- **Type des alarmes :**

  Différents types d'alarmes peuvent être exploitées et utilisées : logique (TOR), analogiques, alarmes systèmes …

- **Indicateur de présence d'Alarmes :**
  L'indicateur d'alarme est un symbole graphique configurable qui s'affiche à l'écran quand une alarme apparaît et qui doivent être acquittées selon le paramétrage réalisé .L'indicateur de message d'alarmes ne peut qu'être configuré qu'en tant qu'icône graphique.

Nous trouverons ci-après des astuces pour la création et l'édition des Vues/Fenêtres d'alarmes :

Déplacement des colonnes de la Vue/Fenêtre d'alarme :
Sélectionner la Vue/Fenêtre d'alarme puis faites un Clic droit .Dans le menu déroulant, choisir "Editer". Nous pouvons alors avec le bouton gauche de la souris déplacer et arranger individuellement les titres des colonnes. (Figure 2.21)

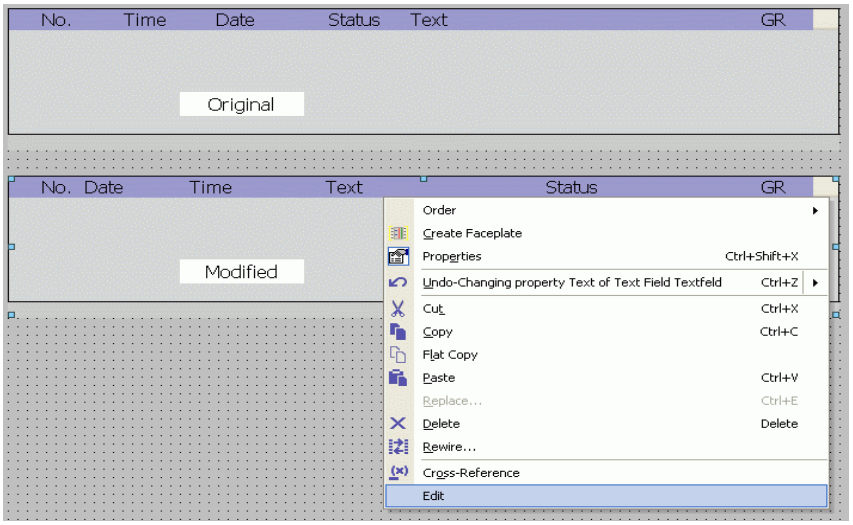


Figure 2.21 : Fenêtre d'alarme

Création d'une classe d'alarme:
Nous pouvons ajouter de nouvelles classes d'alarmes à celles des classes déjà existantes (Erreurs, Avertissements, Système, Evénements de diagnostic). Il est ainsi possible d'affecter une classe d'alarme séparée pour chacun de nos sous-systèmes. Nous pouvons alors affecter la classe d'alarme en conséquence dans notre Vue/Fenêtre d'alarme.
Il n'y aura alors que les alarmes de notre sous-système sélectionné qui seront affichées.

La "Classe d'alarme" se trouve dans l'arborescence du projet "Gestion des alarmes > Paramétrage > Classe d'alarmes". L'image ci-après nous montre la modification d'une classe d'alarme dans la vue des Alarmes.

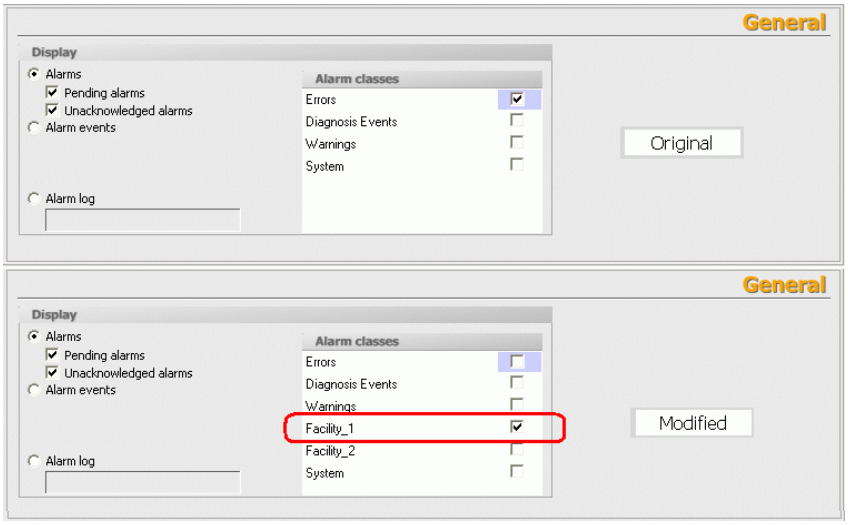


Figure 2.22 : Fenêtre classe d'alarme

Boutons d'édition:
Avec le menu "Propriétés > Paramétrage " de la Vue/Fenêtre des Alarmes, nous pouvons affecter les boutons pour l'affichage (Texte d'Info, Acquittement, Editer). Nous ne pouvons pas changer les icônes de ces boutons. Si nous souhaitons utiliser nos propres Icônes/Textes, nous devons créer vos propres Boutons sur lesquels nous les Symboles / Textes de notre choix. Les fonctions sont disponibles dans le menu des propriétés des boutons ("Propriétés > Evénements > Appuyer > Clavier pour objets de vues > Vue alarme...").

Configuration de la couleur des classes d'alarmes :
Afin de mieux différentier les états des alarmes (arrivé, parti, acquitté) nous pouvons configurer la couleur des classes d'alarmes, nous devons valider "Utiliser Couleur de classes d'alarmes" dans la fenêtre de Projet sous l'arborescence "Gestion des alarmes > Paramétrage > Paramétrage des alarmes > Général".

Variables configurées dans les textes des messages d'alarmes:
Si nous avons configuré une Variable dans un texte de messages pour par exemple afficher la valeur en cours au moment de l'erreur, cette valeur est dite "statique". Cela signifie que la valeur de la variable est affichée avec la valeur que la variable avait au moment où le message est apparu .Cette valeur n'est alors plus mise à jour dans la Vue/Fenêtre des Alarmes.

1.6. Création de fonctions et d'actions (Global Script) :

Le Global Script est l'éditeur dans lequel on peut créer des fonctions et des actions dans le but de dynamiser l'environnement WinCC Runtime. [8]

VBScript (VBS) nous permet d'accéder en Runtime aux variables et aux objets du système graphique Runtime, mais aussi d'exécuter des actions ne dépendant pas des vues :

- Variables: On peut lire et écrire des valeurs variables pour par exemple proposer des valeurs variables pour l'automate, et ce par simple clic de souris sur un bouton.

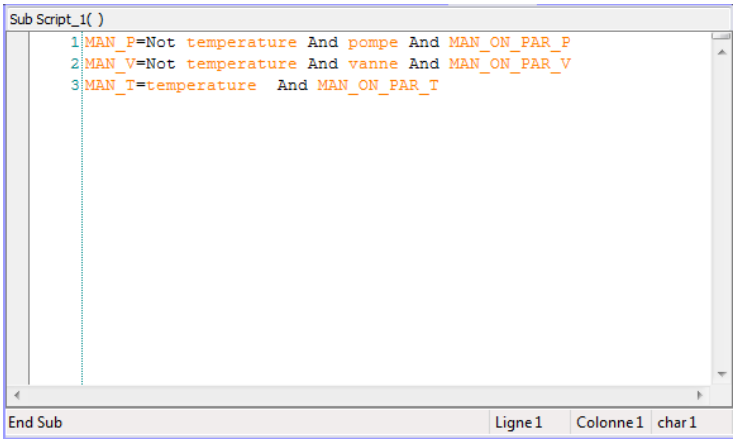


Figure 2.23 : Global Script.

- Objets: On peut dynamiser les propriétés de l'objet avec des actions et déclencher des actions via les événements pour objets.

- Actions ne dépendant pas de la vue: On peut déclencher des actions indépendamment des vues, de façon cyclique ou par le biais de valeurs variables, par exemple la transmission quotidienne de valeurs dans un tableau Excel.

VBS dans WinCC nous permet d'utiliser des procédures, des modules et des actions permettant de dynamiser l'environnement Runtime:

- Procédures: Une procédure correspond à une fonction en C. On inscrit dans les procédures le code qu'on veut utiliser en plusieurs endroits de la configuration.

- Modules: Les modules nous permettent de regrouper les procédures en unités significatives. On crée par exemple des modules pour des procédures qui sont utilisées dans une vue déterminée ou qui font partie d'une rubrique particulière, par exemple les fonctions auxiliaires mathématiques ou les fonctions d'accès à la banque de données.

- Actions: Les actions sont toujours déclenchées par un Trigger. On configure les actions pour des propriétés d'objets graphiques, pour des événements qui interviennent pour un objet graphique ou bien globalement dans le projet. Dans les actions, on peut appeler sous forme de procédures un code utilisé plusieurs fois.

Conclusion :

Nous avons pu voir aussi à travers ce chapitre que WinCC est un logiciel performant qui permet la supervision des processus ainsi que l'exploitation des données acquises à partir de ces derniers.

En outre, il nous offre la possibilité d'intégrer nos propres programmes dans le but de piloter les processus.
Par la suite, ce logiciel va nous servir à visualiser le procédé en temps réel et identifier les systèmes grâce à l'archivage des données.

Chapitre 3

Conception d’un capteur de niveau à base d’Arduino et d’un circuit de conditionnement

Introduction :

Les capteurs sont des éléments sensibles à des grandeurs physiques qu'ils transforment en grandeur électrique (en général une tension). Ils sont souvent intégrés à une chaîne d'acquisition (Figure 3.1) permettant à la grandeur mesurée d'être conditionnée afin que la mesure donne une estimation optimisée du mesurande.

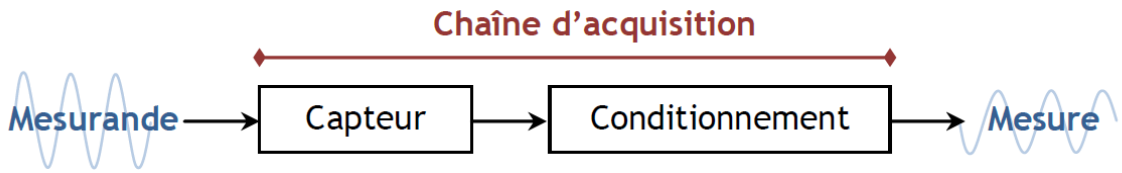


Figure 3.1: Schéma de la chaîne d'acquisition d'une mesure [12]

- **Le mesurande, grandeur physique à mesurer**

Une mesure est une représentation quantifiée d'une grandeur physique (niveau, débit …). On définit la terminologie suivante :

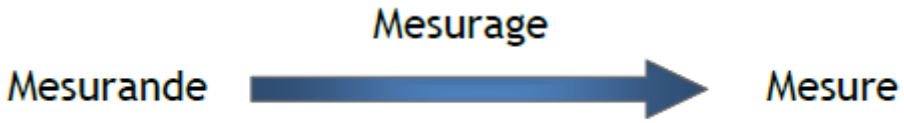


Figure 3.2 : Différents éléments de la mesure [12]

- ✓ Mesurande : grandeur physique soumise à un mesurage (pression, température, ...)
- ✓ Mesurage : toutes les opérations permettant l'obtention de la valeur d'une grandeur physique (mesurande)
- ✓ Mesure : valeur numérique représentant le mesurande

En l'absence d'un capteur de niveau industriel, nous avons nous-mêmes réalisé un capteur de niveau en utilisant une carte Arduino, un capteur aux ultrasons HC-SR04 et un circuit convertisseur. Ce dernier est indispensable pour transmettre l'information de l'Arduino à l'API. En effet, la carte Arduino fournit un signal PWM et l'API en reçoit de l'analogique.

Nous avons fait appel à ces composants en raison de leurs disponibilités et mises en œuvre faciles ainsi que leurs performances satisfaisantes dans un contexte didactique.

Dans ce chapitre, on va détailler le fonctionnement du capteur HC-SR04 et du circuit d'interfaçage.

3.1. Le capteur de niveau à base d'Arduino

Description du capteur HC-SR04 : [11]

Le capteur HC-SR04 utilise les ultrasons pour déterminer la distance d'un objet. Il offre une excellente plage de détection sans contact, avec des mesures de haute précision et stables. Son fonctionnement n'est pas influencé par la lumière du soleil ou des matériaux sombres, bien que des matériaux comme les vêtements puissent être difficiles à détecter.

Figure 3.3 : capteur aux ultrasons HC-SR04 [11]

Relativement à un capteur industriel, le capteur HC-SR04 est intéressant, pour son coût très bas et ses résultats de précision. L'écart est d'environ 3 cm avec un objet placé à 2 m, ce qui représente **une erreur inférieure à 2 %.**

Caractéristiques

Dimensions : 45 mm x 20 mm x 15 mm
Plage de mesure : 2 cm à 400 cm
Angle de mesure efficace : 15 °
Largeur d'impulsion sur l'entrée de déclenchement : 10 µs (Trigger Input Pulse width)

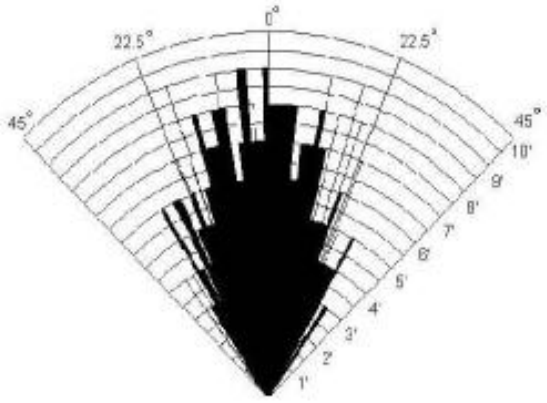


Figure 3.4 : Performances en fonction de l'emplacement de l'obstacle [11]

Broches de connexion

- Vcc = Alimentation +5 V DC
- Trig = Entrée de déclenchement de la mesure (Trigger input)
- Echo = Sortie de mesure donnée en écho (Echo output)
- GND = Masse de l'alimentation

Spécifications et limites

Paramètres	Min	Type	Max	Unité
Tension d'alimentation	4.5	5.0	5.5	V
Courant de repos	1.5	2.0	2.5	mA
Courant de fonctionnement	10	15	20	mA
Fréquence des ultrasons	-	40	-	kHz

Tableau 3.1 : Caractéristiques techniques du HC-SR04

Fonctionnement

Pour déclencher une mesure, il faut présenter une impulsion "high" (5 V) d'au moins 10 µs sur l'entrée "Trig". Le capteur émet alors une série de 8 impulsions ultrasoniques à 40 kHz, puis il attend le signal réfléchi. Lorsque celui-ci est détecté, il envoie un signal "high" sur la sortie "Echo", dont la durée est proportionnelle à la distance mesurée.

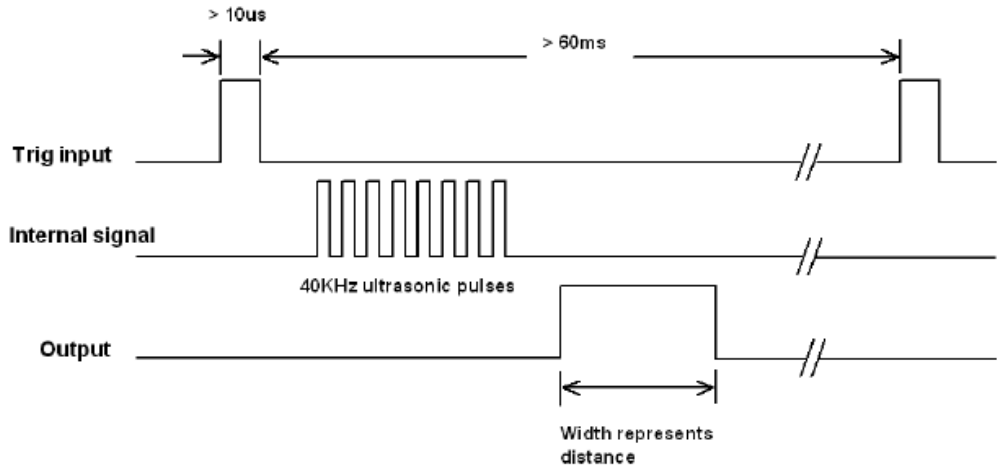


Figure 3.7 : Chronogramme du fonctionnement de la meure [11]

L'entrée « Trig » et la sortie « Echo » devront être manipulées, comme indiqué là-dessus, dans le programme Arduino pour que le capteur HC-SR04 fonctionne. Par la suite, la carte Arduino va générer des tensions PWM dont le rapport cyclique est proportionnel à la distance mesurée et l'amplitude est entre 0 et 5V.

3.2. Réalisation d'un circuit de conversion PWM/analogique

3.2.1. Composants du circuit :

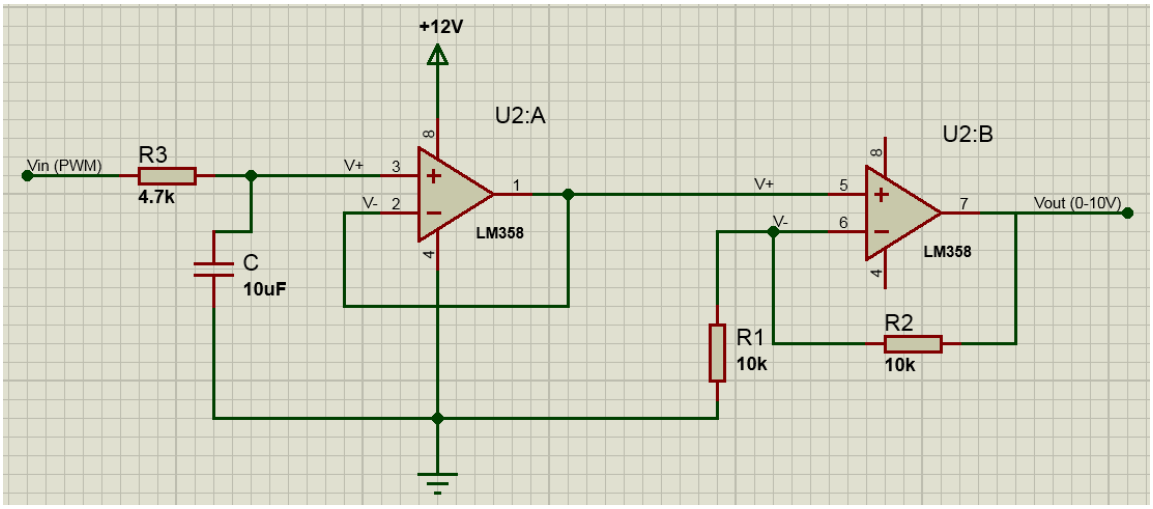


Figure 3.8 : Diagramme du circuit de de conversion PWM/analogique

C'est un circuit qui convertit un signal PWM dont l'amplitude est de 5V, la fréquence est de 490 Hz en un signal analogique 0-10V. Ce convertisseur se décompose en trois parties :

- Un filtre passe-bas:

Le filtre passe-bas est un dispositif qui démontre une réponse en fréquence relativement constante (gain fixe) aux basses fréquences et un gain décroissant aux fréquences supérieures

à la fréquence de coupure « f_c ». La décroissance plus ou moins rapide dépend de l'ordre du filtre.

Idéalement, le filtre passe-bas aurait un gain unitaire aux basses fréquences et un gain nul aux fréquences supérieures à f_c :

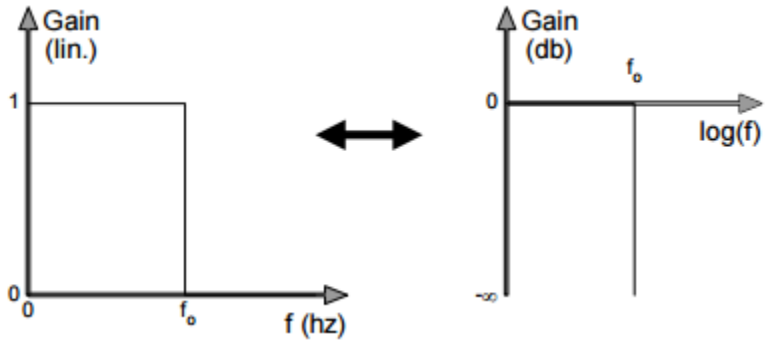


Figure 3.9 : Diagramme de bode d'un filtre passe-bas idéal [15]

On utilise les filtres passe-bas pour réduire l'amplitude des composantes de fréquences supérieures à la fréquence de coupure.

- <u>Un suiveur:</u>

Lorsque l'on charge un montage par un autre, l'interaction des impédances des montages amont et aval altère la tension E prélevée. Alors Vc devient différent de E

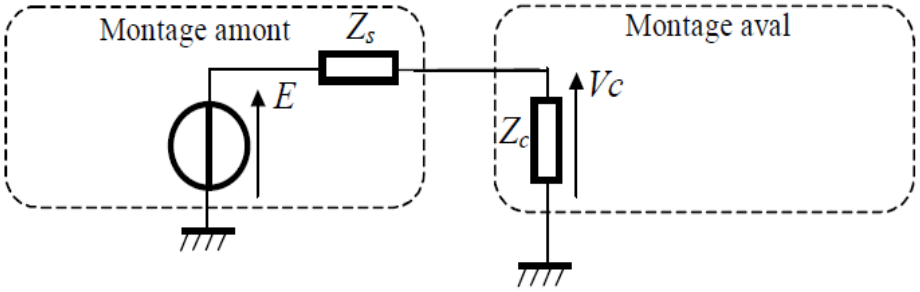


Figure 3.10 : Schéma illustrant le besoin d'un montage suiveur [15]

Pour éviter cet inconvénient, il faut transmettre la tension avec un courant extrait nul. C'est ce que réalise le montage suiveur. Le suiveur de tension permet de prélever une tension sans la perturber.

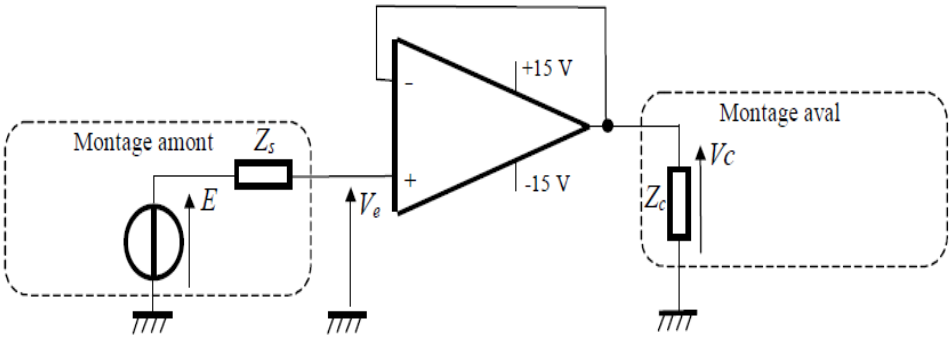


Figure 3.11 : Schéma avec un montage suiveur [15]

- Un amplificateur:

Un amplificateur électronique (ou amplificateur) est un système électronique augmentant la tension et/ou l'intensité d'un signal électrique. On l'utilisera pour avoir une tension (0-10V) à partir d'une tension (0-5V).

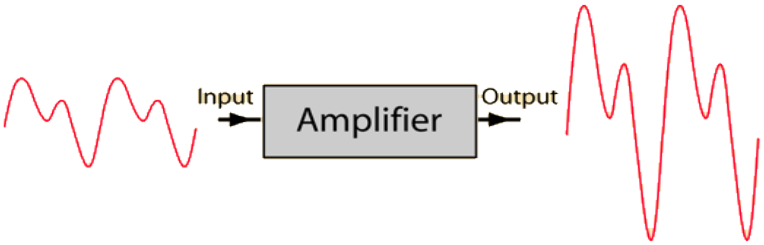


Figure 3.12 : Principe de l'amplification [15]

3.2.2. Fonctionnement détaillé du circuit convertisseur :

✓ **Le filtre :**

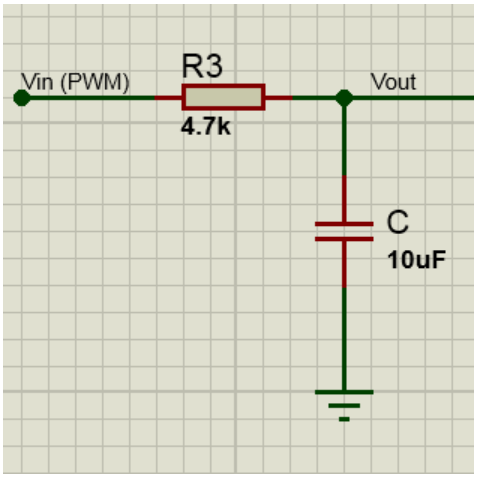


Figure 3.13 : Filtre passe-bas utilisé

Dans le domaine de Laplace, on a :

$$\text{Vout} = \frac{\mathbf{Vin}}{\mathbf{1+RC\,S}} = \frac{\mathbf{Vin}}{\mathbf{1+0.047\,S}}$$

Le diagramme de Bode en amplitude (figure 3.14) du filtre donne sa fréquence de coupure qui sera :

$$f_c = \frac{\mathbf{1}}{\mathbf{2*\pi*RC}} = 3.38 \text{ Hz.}$$

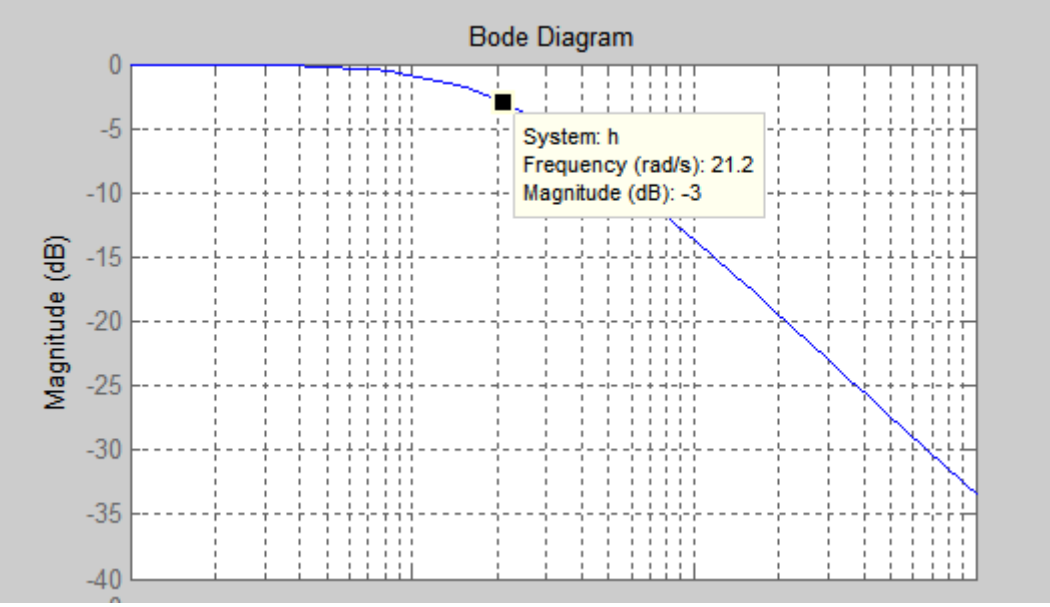


Figure 3.14 : Diagramme de Bode du filtre

Ce filtre fournit en sortie la composante continue du signal PWM, ou autrement dit la valeur moyenne de la tension d'entrée sur une période T.

On peut expliquer cela par les séries de Fourier, une fonction f(x) périodique continue de période T peut être décomposée comme suit:

$$f(x) = \sum_{n=-\infty}^{+\infty} c_n(f) e^{i2\pi \frac{n}{T} x}$$

Avec n ∈ Z

La fonction f(x) représente le signal d'entrée PWM. Les coefficients $C_n(f)$ sont donnés par l'expression suivante :

$$c_n(f) = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-i2\pi \frac{n}{T} t} \mathrm{d}t.$$

Le terme C_0 (le fondamental) représente la valeur moyenne de f, donc la valeur moyenne du signal PWM. Ce terme sera présent en sortie et ne sera pas éliminé par le filtre car c'est un signal continue.

La première harmonique C1 sera éliminée par le filtre car elle présente une fréquence égale à celle du signal PWM, elle est égale à 490 Hz.

Toutes les harmoniques C_2, C_3, C_4,…, C_n seront éliminées car ils auront une fréquence égale à n fois la fréquence du signal d'entrée.

Donc, les valeurs de la résistance R et la capacité C devront être choisies telle que la fréquence de coupure du filtre soit suffisamment petite pour éliminer les harmoniques du

signal PWM.

✓ **Le suiveur :**

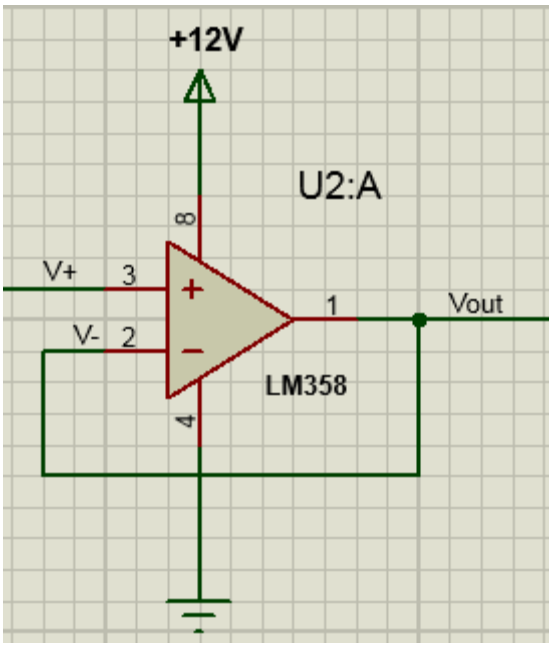


Figure 3.15 : Montage suiveur utilisé

On a : $V^- = V_{out}$ et $V_{out} = A_{OL} * (V^+ - V^-)$

Ou : A_{OL} est le gain en boucle ouverte, typiquement aux environs de 100dB

Donc : $\frac{Vout}{AOL} + V_{out} = V^+$, et comme le gain A_{OL} est très grand, On peut faire l'approximation suivante :

$$V_{out} = V^+$$

✓ **L'amplificateur de tension:**

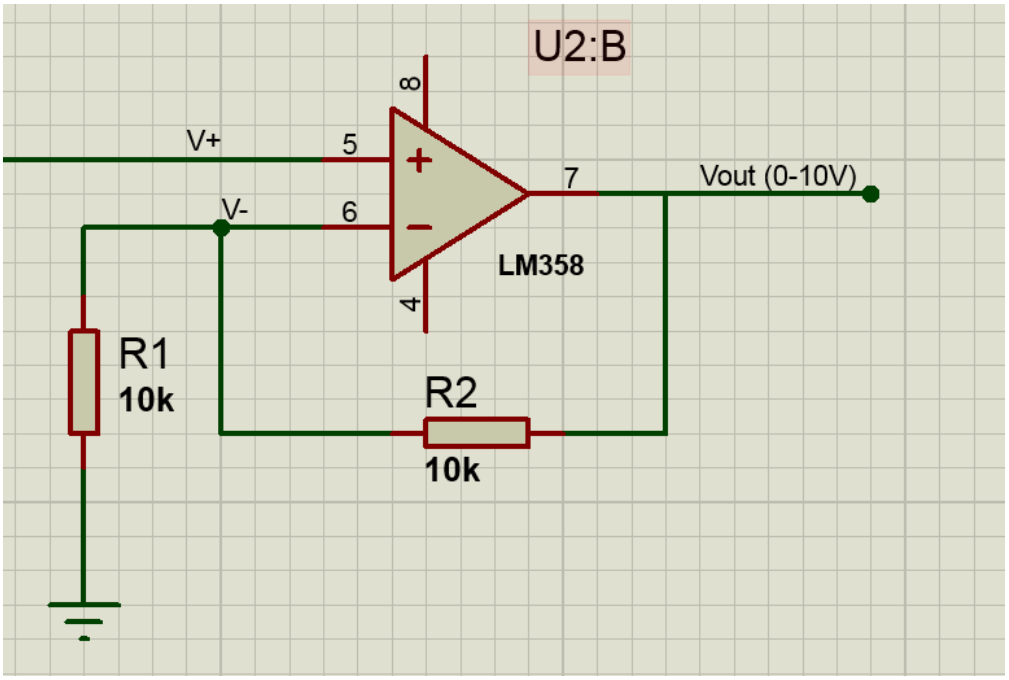


Figure 3.16 : Amplificateur utilisé

On a : $V^- = \left(\frac{R2}{R1+R2}\right)V_{out}$ et $V_{out} = A_{OL} * (V^+ - V^-)$

Ensuite: $\frac{Vout}{AOL} = V^+ - \frac{R2}{R1+R2} V_{out}$ et comme le gain A_{OL} est très grand, On peut faire l'approximation suivante :

$$V^+ = \frac{R2}{R1+R2} V_{out}$$

On déduit que: $V_{out} = \frac{R1+R2}{R2} V^+ = \left(\frac{R1}{R2}+1\right) V^+$

On a besoin d'une tension de sortie qui varie entre 0 et 10V pour que l'automate programmable puisse l'interpréter correctement.

Donc le gain en tension de l'amplificateur doit être égal à 2, vu que le signal V^+ varie entre 0 et 5V. Par conséquent, les résistances R1 et R2 doivent être identiques et pas trop grandes (~kΩ) pour qu'elles n'atteignent pas la résistance d'entrée de l'amplificateur opérationnel.

3.3. Précision du capteur de niveau réalisé :

Comme mentionné précédemment, l'erreur du capteur HC-SR04, seul, est inférieure à 2%. Après avoir effectué des tests expérimentaux, l'erreur du capteur de niveau réalisé (HC-SR04 avec le circuit) est égale à 1% , soit une erreur de ± 0.3 cm.

On a remarqué aussi qu'on peut atteindre ±0.05 cm en avec une alimentation du circuit convertisseur égale à 12V bien précise.

Ce n'est pas un résultat digne d'un capteur industriel où une grande précision est requise mais c'est un très bon résultat en considérant plusieurs facteurs, parmi lesquels :

- Contexte de l'application (elle rentre dans un cadre académique et pédagogique : Projet de Fin d'Etudes, Travaux Pratiques ….).
- Cout : le coût du capteur de niveau réalisé est de 4000 DA alors qu'un capteur de niveau industriel coûte 42.000 DA.

Conclusion:

Les cartes Arduino sont un puissant outil de prototypage pour les cartes électroniques. Aussi, elles permettent un accès facile et intuitif à l'informatique embarqué.

En effet, en utilisant cette carte et un capteur HC-SR04, on a réalisé des capteurs pour la mesure de niveau pour deux cuves présentes sur la station. Une conversion du signal est fondamentale afin que l'API puisse exploiter la mesure fournie par l'Arduino.

La mesure de niveau va nous permettre, par la suite, d'accomplir des tâches importantes comme : l'identification des différents systèmes et la régulation de niveau.

Chapitre 4

Réalisation et travaux effectués sur la

Station de pompage

Introduction

La station de pompage didactique à été réalisée en 2015 et nous l'avons remise en marche dans le but de réaliser des applications et manipulations plus avancées.

Tout au long de ce chapitre, nous allons présenter les réalisations effectuées sur cette station.

Nous décrirons la procédure d'identification, les cahiers des charges des manipulations digitale et analogique et les grafcets. Nous finirons par décrire les programmes sur STEP7 ainsi que la supervision de ces derniers sur WinCC flexible.

L'organigramme de la figure 4.2 montre les étapes suivies dans la réalisation de notre projet.

4.1. Description de la station de pompage et identification du système pompe-niveau

4.1.1. Description de la station de pompage :

La station de pompage est un dispositif éducatif destiné à l'étude et le contrôle de plusieurs systèmes. Le contrôle de ces systèmes se fait par le biais des automates Siemens S7 314 IFM.

Notre prototype a été basé sur le système contrôle de niveau. Les éléments et composants utilisés pour la construction de notre prototype, ont été choisis en fonction des paramètres, disponibilité et coût.

La figure 4.1 donne une vue générale de la maquette, qui permet de réaliser de différentes manipulations de même principe que les stations de pompage hydraulique :

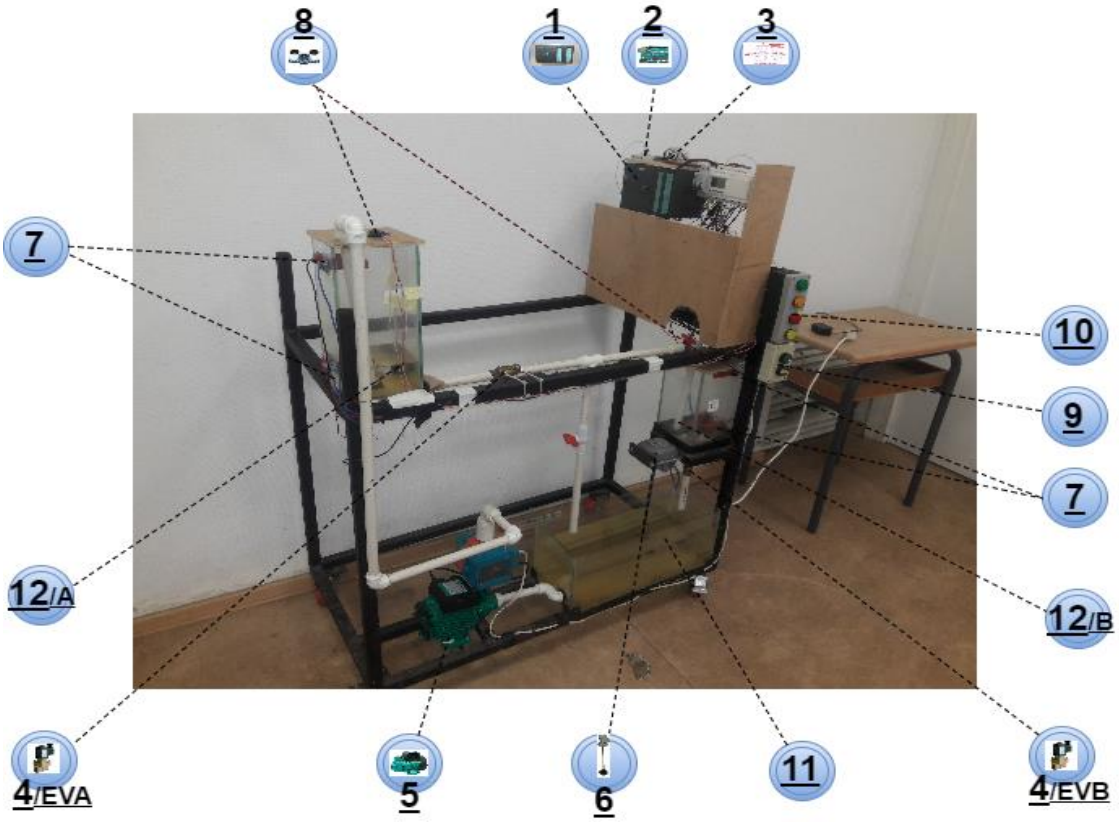


Figure 4.1 : Station de pompage réalisée

1-Automate (API/PLC)	7-Capteur de niveau logique (TOR)
2-Arduino	8-Capteur de niveau (Ultrason hc-05)
3-Circuit convertisseur PMW➔Analogique	9- Boutons
4-Électrovannes (EVA, EVB)	10-Voyants
5-Pompe	11-Réservoir (C)
6-Agitateur	12- Cuves A et B

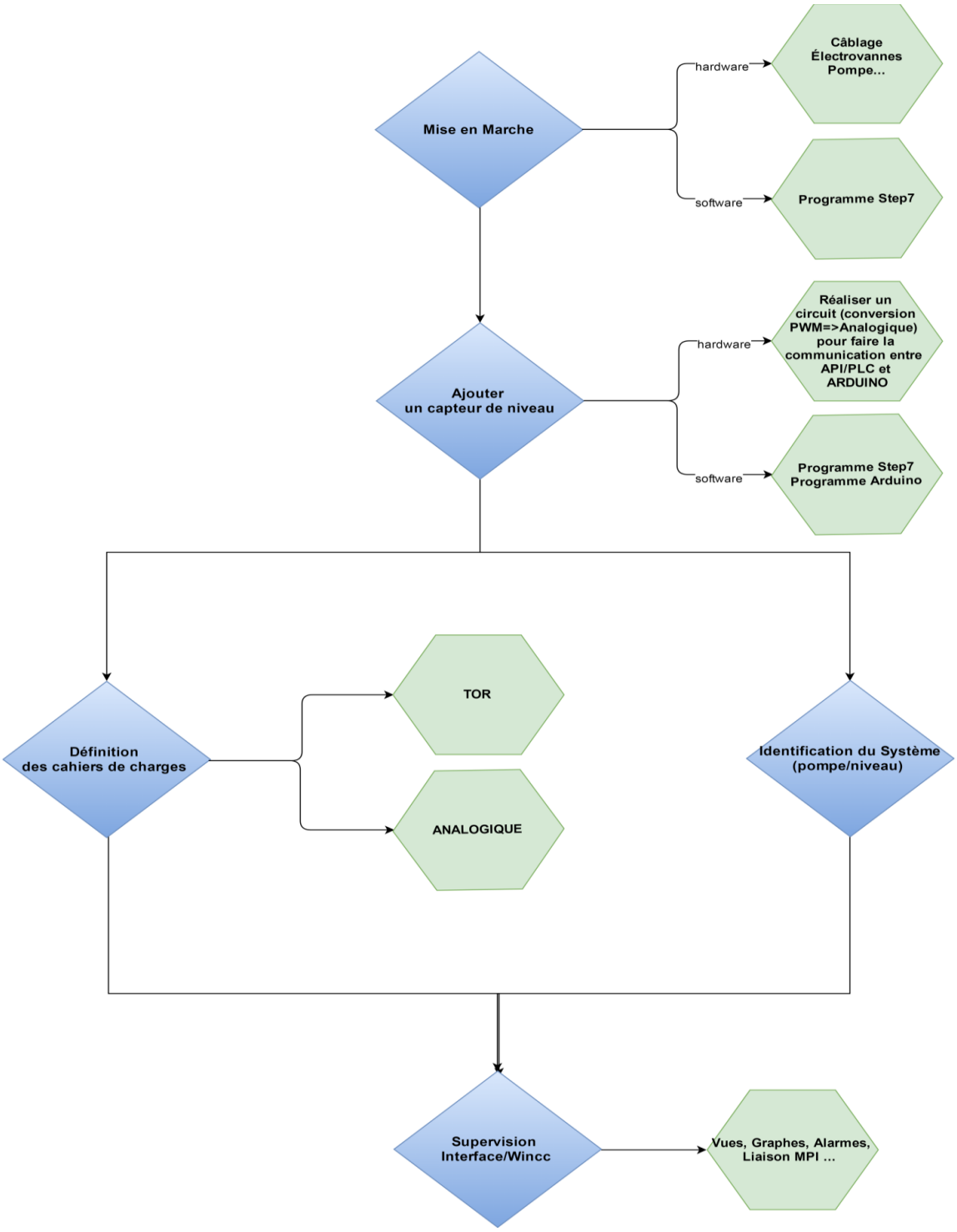


Figure 4.2 : Organigramme des étapes du projet

L’ensemble des éléments et les caractéristiques du prototype réalisé :

Objet	Description
support métallique	Le prototype repose sur un support métallique 1.50 m sur 1m et d’une hauteur de 1.50m qui comporte toutes les organes de la station.
Réservoir	Un réservoir (C) de 0.50m sur 0.20m sur une hauteur de 0.20m
Cuves	Deux Cuves (A et B) de 0.15m sur 0.15m sur une hauteur de 0.30m/0.25m.
Pompe	Une pompe centrifuge QB50, couplé à un moteur asynchrone monophasé alimenté à 220 V La hauteur manométrique maximale de cette pompe est de 20 m, protégé par un cerveau pour éviter la marche à sec de la pompe.
Agitateur	Un mélangeur(Agitateur) alimenté en 220 v pour agiter le liquide.
Démarreur	Un démarreur progressif ATS01N03FT qui améliore les performances de démarrage du mélangeur en permettant un démarrage progressif et contrôlé.
Relais	Quatre relais de niveau JYB-714B alimenté en 220V et muni chacune de 2 bougies de sonde pour détecter les niveaux haut et les niveaux bas des réservoirs A et B. Deux relais électromagnétiques 8 broches servant comme interface de sortie entre l’automate et les actionneurs (pompe et l’agitateur).
robinets d’arrêt à opercule	Les deux robinets d’arrêt à opercule, utilisés pour permettre la flexibilité du

	prototype afin qu'on peut l'utiliser dans deux manipulations différentes (ou plus).
Alimentation	Une alimentation stabilisée (Harvest electric HAS-120-24) alimentée en monophasé 220 V et qui délivre du 24V.
Electrovannes	Deux électrovannes DN40 TOR alimentées en 24V DC.
Disjoncteur	Un disjoncteur pour la sécurité de toute l'installation.
Répartiteur	Deux répartiteurs de tension pour le 24 V et 220 V
Arduino	Une carte de développement qui réalise des calculs pour mesurer le niveau
Automate (API/PLC)	Un automate Siemens de la gamme SIMATIC S7-300/ CPU 314 IFM qui exécute les cahiers de charges et qui contrôle la station
Ultrason	Deux capteurs ultrason hc-04 utilisés pour la mesure du niveau
Circuit convertisseur	Deux circuits qui reçoivent un signal PWM et délivre un signal analogique
Variateur de vitesse	un variateur de vitesse afin de faire fonctionner la pompe de manière analogique pour contrôle le niveau
PC	Une station PC pour la supervision.

Tableau 3.2 : Eléments et composants de la station

La figure 4.3 donne une vision générale sur notre projet :

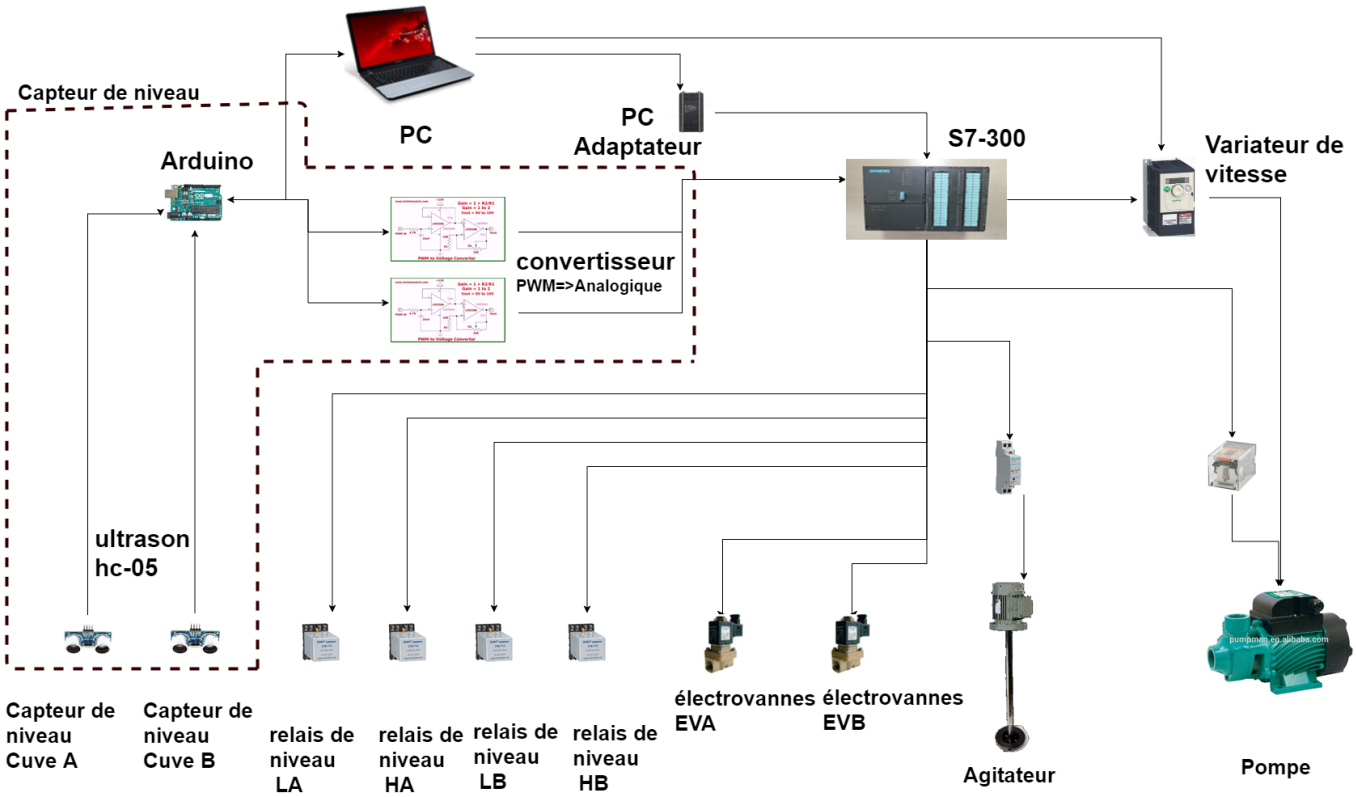


Figure 4.3 : Synoptique du prototype.

4.1.2. Identification du système pompe-niveau :

4.1.2.1. Introduction :

L'automatique consiste en l'étude des systèmes réels des différentes disciplines scientifiques (Electronique, mécanique, thermique, chimie, écologie, biologie, économie…), en vue de l'analyse, de la prédiction, de la surveillance, de la commande, et / ou de l'optimisation des systèmes. Généralement, la connaissance du modèle du système réel (on réalise une modélisation) est nécessaire dans l'étude.

Identifier un processus (système), c'est chercher un modèle (dynamique) mathématique, appartenant à une classe de modèles connue, et qui, soumis à des signaux tests (en entrée), donne une réponse (dynamique et statique en sortie), la plus proche possible du système réel .

Pour élaborer un modèle, deux approches sont souvent considérées :

- Modélisation boîte blanche, Elle se fonde sur les lois physiques, chimiques, mécaniques, biochimiques.
- Modélisation boite noire. La modélisation s'attache à établir « à partir de données expérimentales, une relation entre les variables des entrées du processus et les variables de ses sorties et ne nécessite pas a priori la connaissance des lois physiques.

4.1.2.2. Identification d'un système naturellement instable en boucle ouverte: [14]

Quelle que soit la méthode employée, les paramètres du modèle du procédé à identifier sont ceux d'un intégrateur pur avec retard : k et τ.

La fonction de transfert de ce modèle est la suivante : $H(s) = \frac{Ke^{-Ts}}{s}$

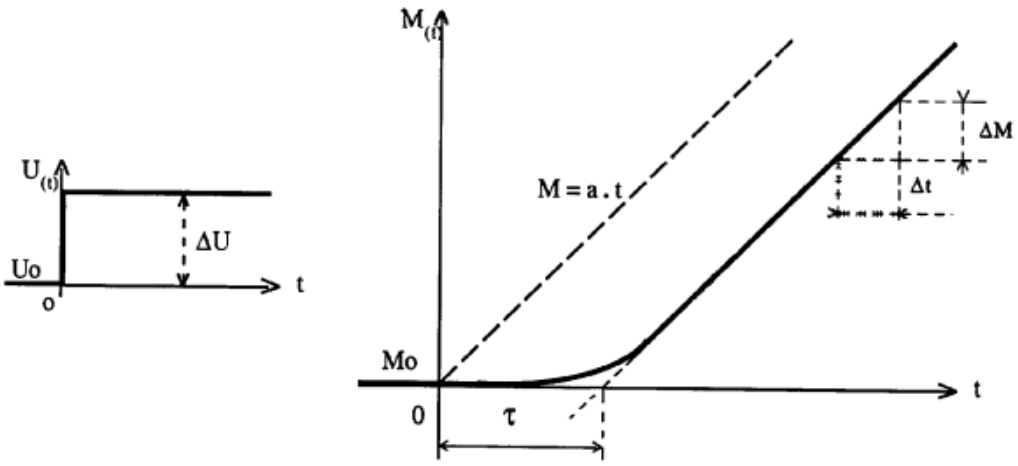


Figure 4.4 : Méthode d'identification d'un système instable [14]

Le temps mort du modèle est déterminé graphiquement

Coefficient d'intégration du procédé : $K = \frac{a}{\Delta U}$

Exemple d'identification de niveau d'un réservoir :

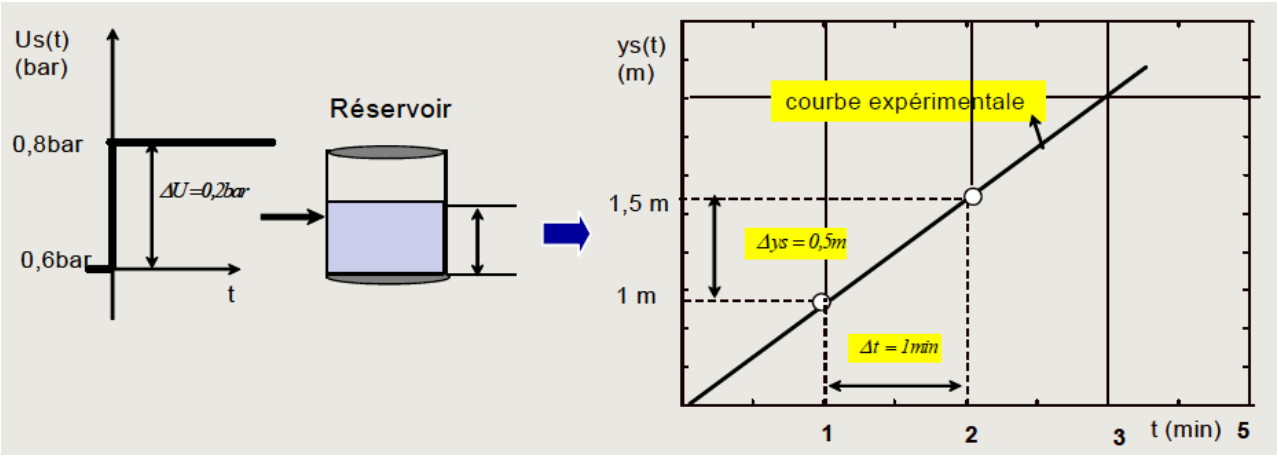


Figure 4.5 : exemple d'identification d'un système instable [14]

$K = \frac{\Delta ys}{\Delta u} = \frac{(1.5-1)\ m/min}{(0.8-0.6)\ bar} = 2.5 \quad [\frac{m}{min.bar}]$

4.1.2.3. Procédure d'identification :

Pour régler le niveau de la cuve A, nous allons utiliser que la partie gauche de notre station c'est-à-dire le circuit de fluide procédé. Dans ce circuit la pompe P déplace de l'eau stockée

dans le réservoir vers la cuve à travers un système de canalisation qui contient une vanne proportionnelle qui contrôle le débit comme le montre la figure 4.6.

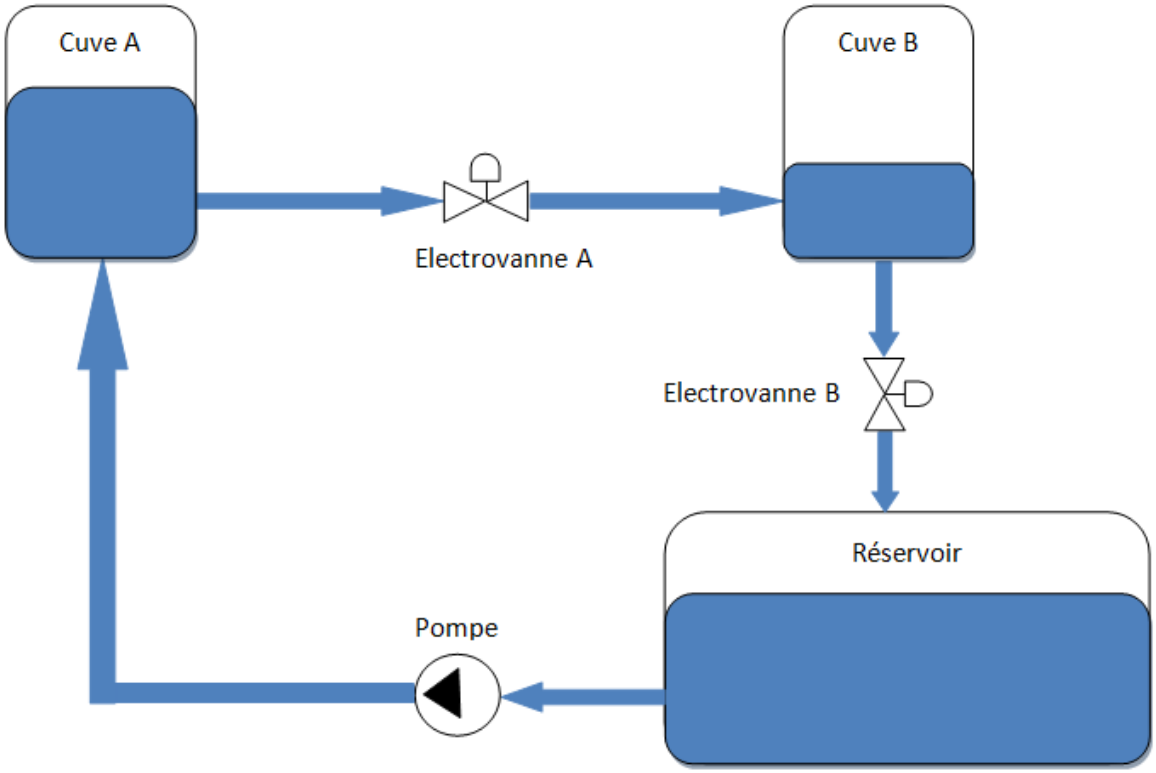


Figure 4.6 : représentation Synoptique de la station

La cuve A étant vide, on garde l'électrovanne « A » fermée et on fait varier le niveau de cette cuve par le biais de la pompe.

Pour l'identification de ce système, nous nous sommes basés sur l'étude de la réponse indicielle en boucle ouverte. On envoie un échelon à la pompe d'amplitude 100% et à l'aide du logiciel WinCC, on relève la réponse qui la tension image du niveau.

En utilisant la méthode d'identification d'un système naturellement instable, la fonction de transfert du modèle est : $H(s)= \frac{Ke^{-\tau s}}{s}$

$K = \frac{a}{\Delta U} = \frac{1.023}{220} = 0.0046$ cm/s.V

Après avoir calculé les différentes constantes du modèle, on donne la fonction de transfert du système pompe-niveau par l'expression: $H(s)= \frac{0.0046\, e^{-0.95\, s}}{s}$

La figure 4.7 illustre les réponses indicielles du modèle et du système réel :

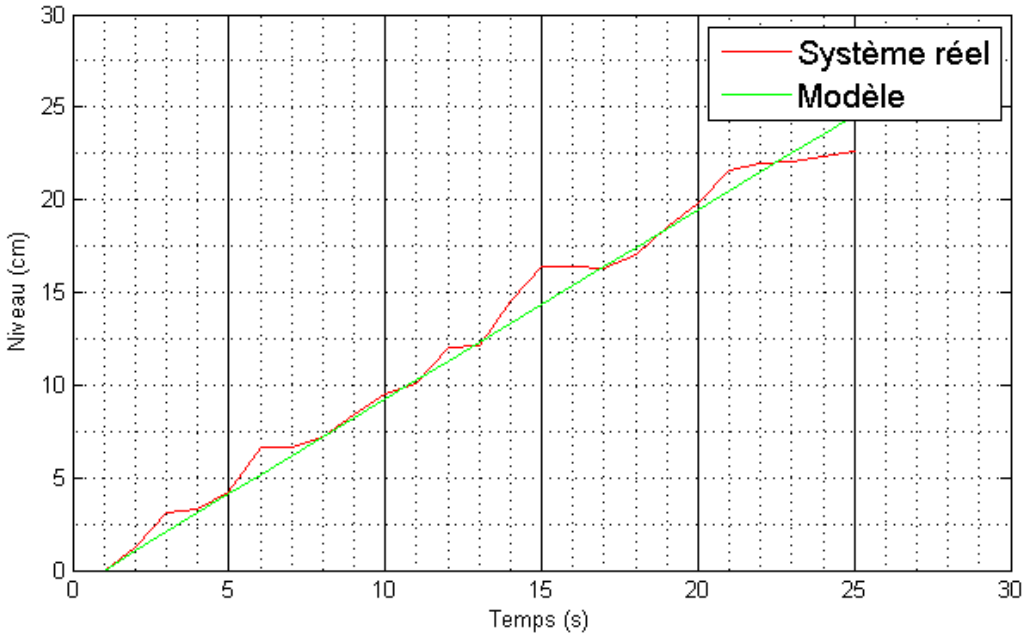


Figure 4.7 : réponses indicielles du modèle et du système réel

Si on met à part les instants t= 3s, t=6s et t=15s, On remarque que la courbe du système réel suit bien la réponse du modèle.

Néanmoins, aux instantes cités, la courbe du système réel présente un écart par rapport à la réponse du modèle, cela est dû à la précision du capteur utilisé qui est de ± 0.3cm.

4.2. Cahiers de charges et Grafcets

Dans cette partie, on présente les grafcets et les manipulations réalisées sur notre station de pompage.

4.2.1. Grafcet et définition du cahier de charges de la manipulation TOR (logique) :

La station étant à l'arrêt et en appuyant sur le bouton « marche », trois cas se présentent comme le montre la figure 4.8 :

- Le niveau haut de la cuve A n'est pas atteint :
  La pompe démarre pour ramener de l'eau du réservoir à la cuve A et le voyant V3 s'allume jusqu'à ce que le niveau haut de cette cuve soit atteint.
- Le niveau haut de la cuve B n'est pas atteint :
  L'électrovanne EVA s'ouvre pour ramener de l'eau de la cuve A à la cuve B et le voyant V1 s'allume jusqu'à ce que le niveau haut de B soit atteint.
- Les niveaux hauts des cuves A et B sont atteints :
  L'électrovanne EVA s'ouvre et l'eau émanant de la cuve A est versé dans le réservoir par le biais du robinet, une fois cette cuve est vidée, la même procédure est exécutée avec l'électrovanne EVB et la cuve B. une fois les deux cuves sont vides, il y a une temporisation de 5 secondes suivie du lancement de l'agitateur pour la même durée.

Les voyants V1 et V2 indiquent, respectivement, que les électrovannes EVA et EVB sont ouvertes tandis que le voyant V4 indique l'activation de l'agitateur.

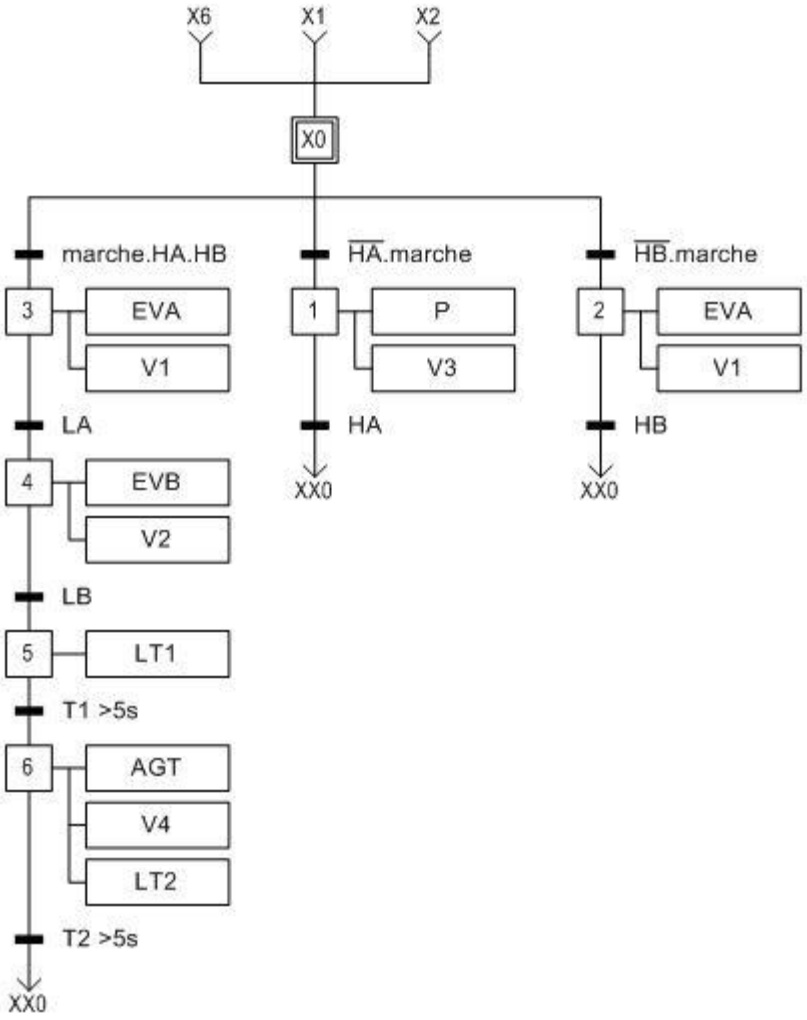


Figure 4.8 : Grafcet représentant la manipulation logique

4.2.2. Grafcet et définition du cahier de charges de la manipulation analogique :

Les symboles d1 et d2 représentent, respectivement, le niveau d'eau dans les cuve A et B.

La station étant à l'arrêt et en appuyant sur le bouton « marche », trois cas se présentent comme le montre la figure 4.9 :

- Si d1 est inférieur à 19 cm :
  La pompe démarre pour ramener de l'eau du réservoir à la cuve A et le voyant V3 s'allume jusqu'à ce que d1 atteigne 20 cm.
- Si d2 est inférieur à 17 cm :
  L'électrovanne EVA s'ouvre pour ramener de l'eau de la cuve A à la cuve B et le voyant V1 s'allume jusqu'à ce que d2 atteigne 18 cm.
- Si d1 est supérieur à 20 cm et d2 est supérieur à 18 cm :

L'électrovanne EVA s'ouvre et l'eau émanant de la cuve A est versé dans le réservoir par le biais du robinet, une fois d1 atteint 7 cm, la même procédure est exécutée avec l'électrovanne EVB et la cuve B jusqu'à ce que d2 atteigne 5 cm. Après, il y a une temporisation de 5 secondes suivie du lancement de l'agitateur pour la même durée. Les voyants V1 et V2

indiquent, respectivement, que les électrovannes EVA et EVB sont ouvertes tandis que le voyant V4 indique l'activation de l'agitateur.

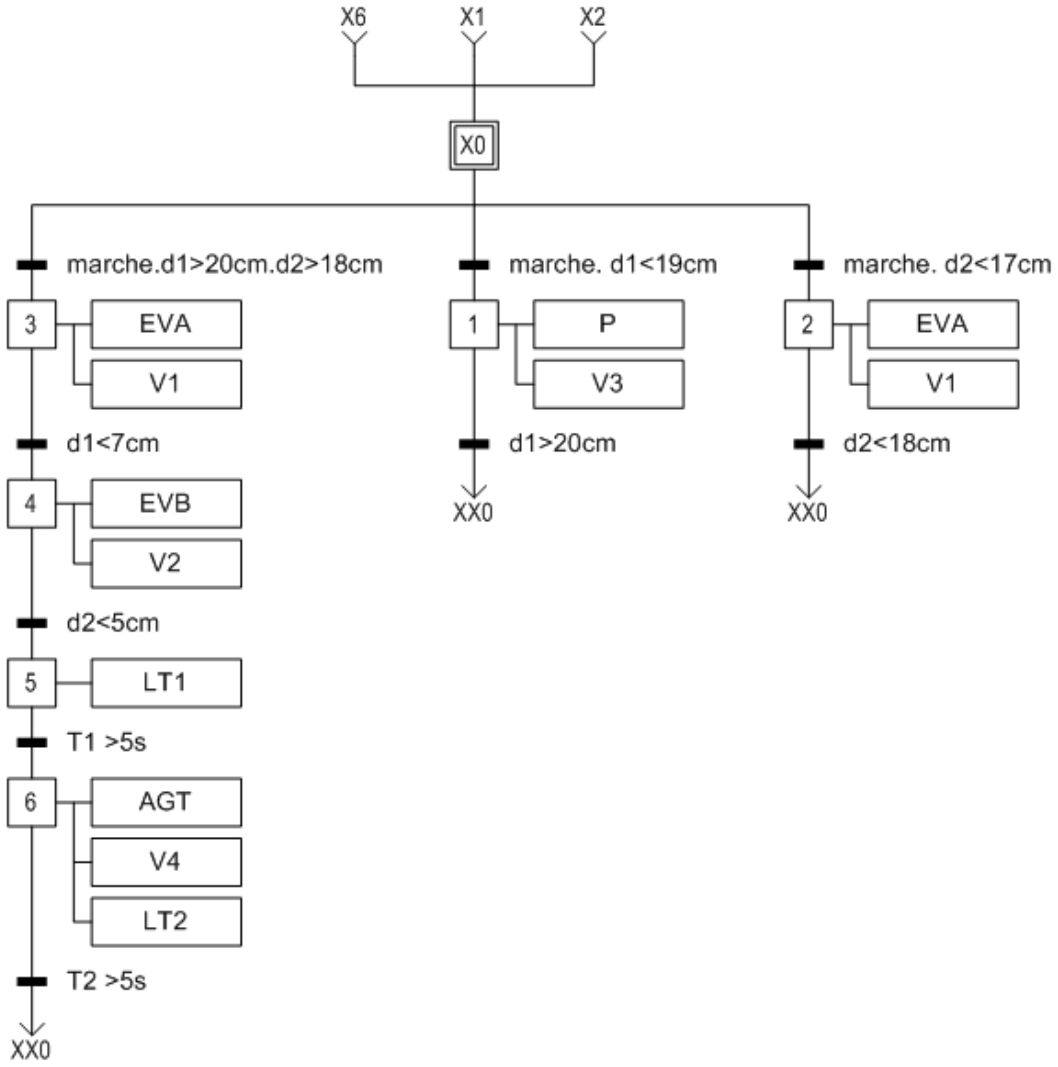


Figure 4.9 : Grafcet représentant la manipulation analogique

4.3. Implémentation sous step7 et WinCC :

4.3.1. Implémentation sous step7 :

STEP 7 permet de répartir le programme utilisateur en différents blocs de programme.
Le bloc d'organisation OB1 est présent par défaut.
Ce bloc constitue l'interface avec le système d'exploitation de la CPU, il est automatiquement appelé par celui-ci et est exécuté de façon cyclique.
Le programme peut être subdivisé en blocs de programme (FB ou FC) plus petits, formant ainsi de petites parties autonomes, et assurant entre autre, une meilleure lisibilité.
La figure 4.10 montre la structure d'un tel programme :

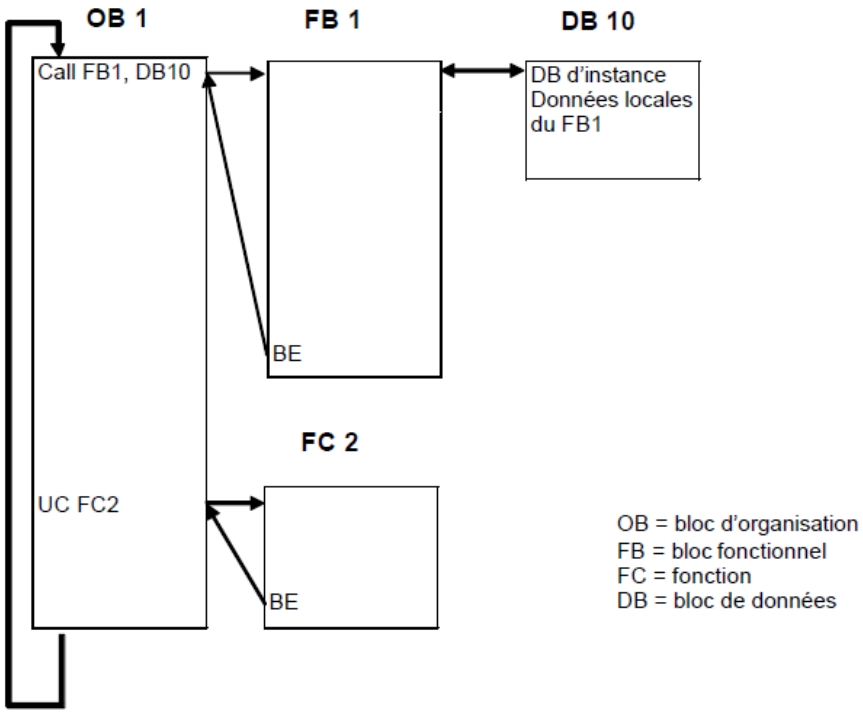


Figure 4.10 : Structure d'un programme Step7

Nous présentons dans cette partie les principales fonctions et blocs utilisés :

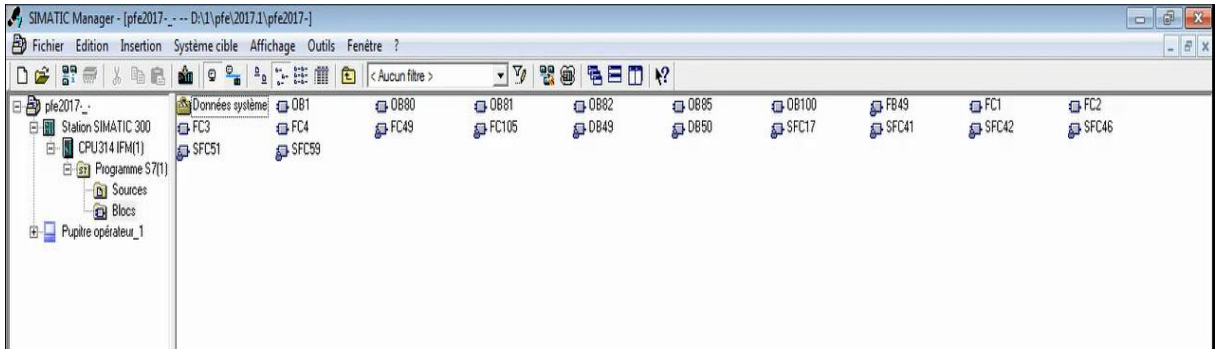


Figure 4.11 : L'ensemble des blocs et fonctions du programme

- **<u>Mise à l'échelle « Scale » (FC105) :</u>**

On fait appel à la fonction lorsqu'on désire traduire une valeur analogique entre 0 et 10V présente sur une entrée de l'automate en une information ou valeur réelle. D'abord, il faut un adressage des variables:

Adresse	Type de variable	information représentée
PEW134	INT	tension image de la distance entre le capteur et le niveau d'eau
MD20	Real	Distance entre le capteur et le niveau d'eau

Tableau 3.3 : variables utilisées par la fonction scale

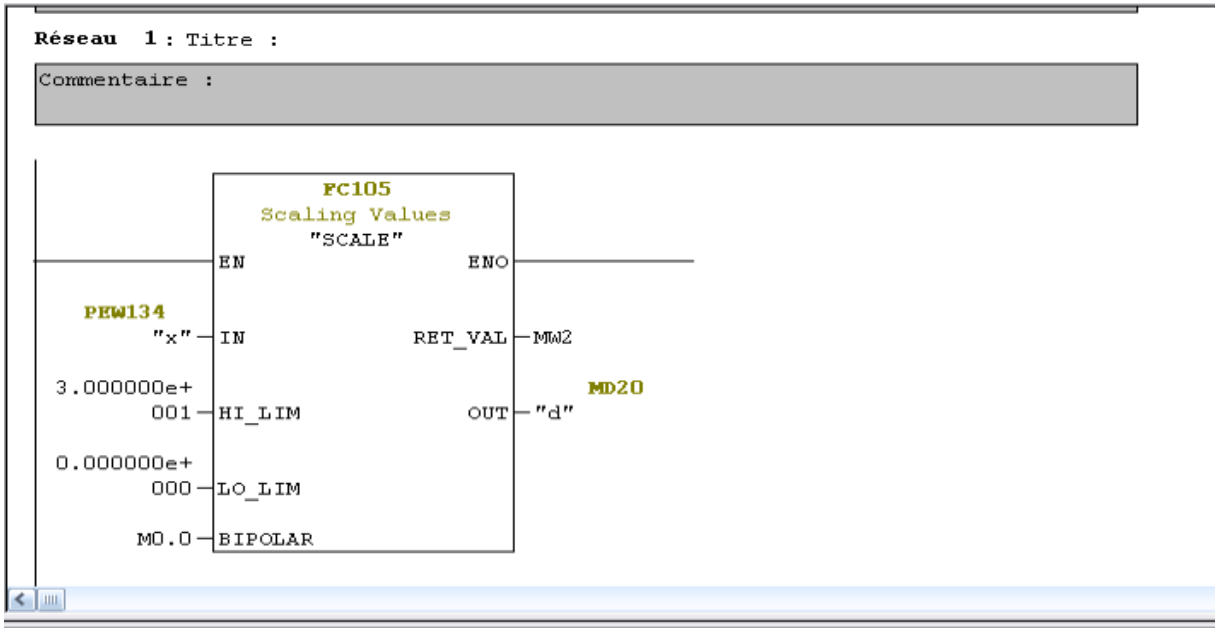


Figure 4.12 : Fonction scale appelée dans le bloc OB 1

La fonction « SCALE » prend une valeur entière (IN) et la convertit selon l'équation ci-après en une valeur réelle exprimée en unités physiques, comprises entre une limite inférieure (LO_LIM=0.0) et une limite supérieure (HI_LIM=30.0) :

OUT = [((FLOAT (IN) –K1)/ (K2–K1)) * (HI_LIM – LO_LIM)] + LO_LIM

Le résultat est écrit dans OUT.

Les constantes K1 et K2 sont définies selon que la valeur d'entrée est bipolaire ou unipolaire.

• Bipolaire : La valeur entière d'entrée est supposée être comprise entre -27648 et 27648, donc :

K1 = -27648.0 et K2 = +27648.0

• Unipolaire : La valeur entière d'entrée est supposée être comprise entre 0 et 27648, donc :

K1 = 0.0 et K2 = +27648.0

Si la valeur entière d'entrée est supérieure à K2, la sortie (OUT) est saturée à la valeur la plus proche de la limite supérieure (HI_LIM) et une erreur est signalée. Si la valeur entière d'entrée est inférieure à K1, la sortie est saturée à la valeur la plus proche de la limite inférieure (LO_LIM) et une erreur est signalée.

Une mise à l'échelle inversée peut être obtenue en programmant une limite inférieure supérieure à la limite supérieure (LO_LIM>HI_LIM). Dans ce cas, la valeur de la sortie diminue quand la valeur de l'entrée augmente.

Si la valeur entière d'entrée est supérieure à K2, la sortie (OUT) est saturée à la valeur la plus proche de la limite supérieure (HI_LIM) et une erreur est signalée. Si la valeur entière d'entrée est inférieure à K1, la sortie est saturée à la valeur la plus proche de la limite inférieure (LO_LIM) et une erreur est signalée. L'état de signal de ENO est mis à « 0 » et RET_VAL prend la valeur W#16#0008.

- **La mise à l'échelle de la sortie « UNSCALE » (FC106) :**

On a fait appel à la fonction « unscale » lorsqu'on voulait transformer une commande réelle calculée suivant une loi donnée en une commande analogique entre 0 et 10V. D'abord, il faut un adressage d'entrées sorties :

Adresse	Type de variable	information représentée
MD24	Real	Commande réelle fournie par le régulateur
PAW128	INT	commande analogique entre 0 et 10V fournie à l'automate

Tableau 4.1 : variables utilisées par la fonction unscale

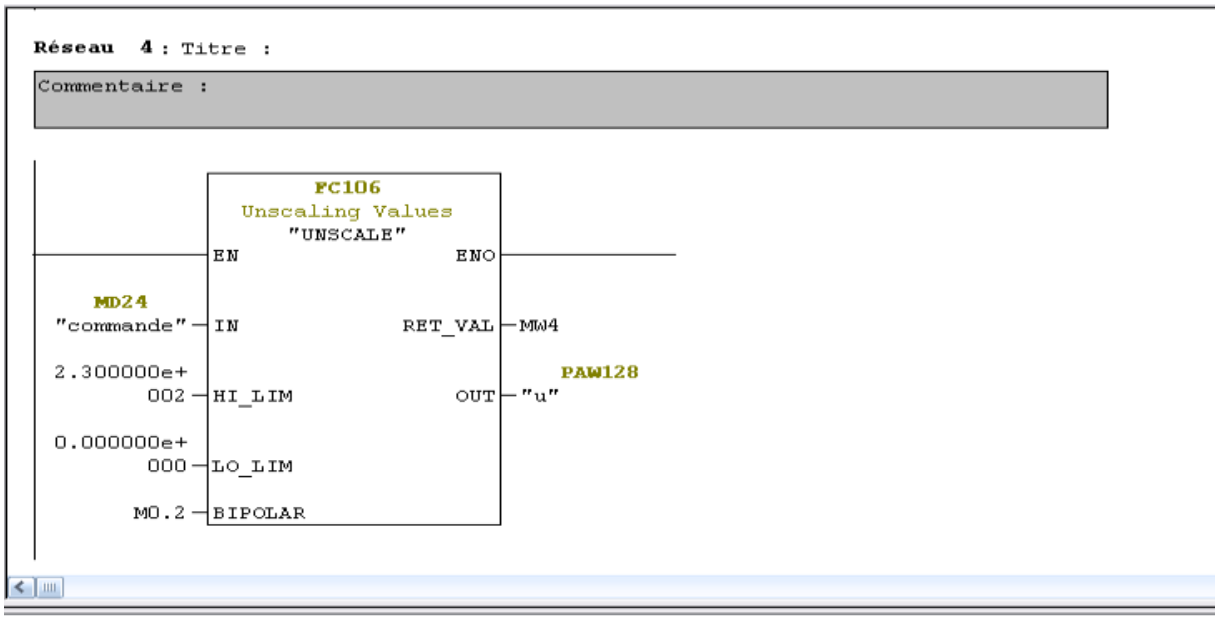


Figure 4.13 : Fonction unscale appelée dans le bloc OB 1

La fonction « Annuler la mise à l'échelle » (UNSCALE) prend une valeur d'entrée réelle (IN) exprimée en unités physiques comprises entre une limite inférieure (LO_LIM) et une limite supérieure (HI_LIM) et la convertit selon l'équation ci-après en une valeur entière :

OUT = [((IN–LO_LIM)/(HI_LIM–LO_LIM)) * (K2–K1)] + K1

Le résultat est écrit dans OUT.

Les constantes K1 et K2 sont définies selon que la valeur d'entrée est bipolaire ou unipolaire.

• Bipolaire : La valeur entière de sortie est supposée être comprise entre -27648 et 27648, donc :

K1 = -27648.0 et K2 = +27648.0

• Unipolaire : La valeur entière de sortie est supposée être comprise entre 0 et 27648, donc :

K1 = 0.0 et K2 = +27648.0

Si la valeur entière d'entrée se situe en dehors de la plage définie par les limites inférieure (LI_LIM) et supérieure (HI_LIM), la sortie (OUT) est saturée à la valeur la plus proche de la limite inférieure ou supérieure de la plage indiquée pour son type (bipolaire ou unipolaire) et une erreur est signalée.

Si la valeur réelle d'entrée se situe en dehors de la plage définie par les limites inférieure (LO_LIM) et supérieure (HI_LIM), la sortie (OUT) est saturée à la valeur la plus proche de la limite inférieure ou supérieure de la plage indiquée pour son type (bipolaire ou unipolaire) et une erreur est signalée. L'état de signal de ENO est mis à « 0 » et RET_VAL prend la valeur W#16#0008.

- **FC (fonction) :**

On a réalisé plusieurs FC pour programmer nos cahiers de charges :
FC1 : dédiée au grafcet de la manipulation logique TOR
FC2 : Calcul de niveau pour les cuves A et B.
FC3 : Sous programme dédié aux alarmes logiques (TOR)
FC4 : dédiée au grafcet de la manipulation analogique

On a fait appel à ces fonctions au bloc d'organisation OB1.

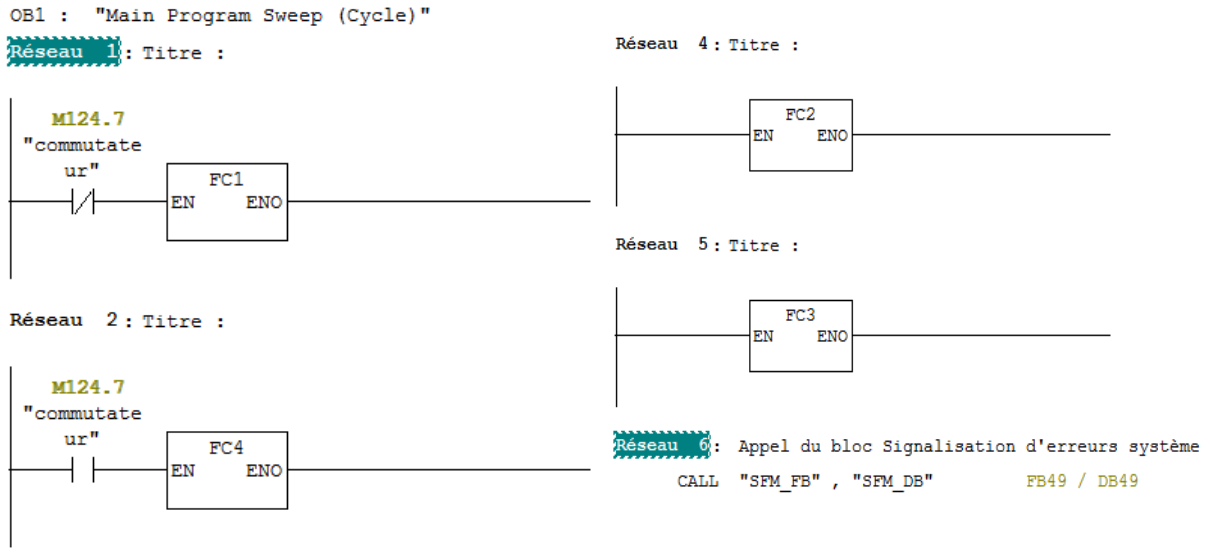


Figure 4.14 : Appel des FC dans le bloc OB1

Les blocs OB pour les erreurs :
On a fait appel à plusieurs blocs pour le traitement d'erreur : OB80 pour l'erreur de temps, OB81 pour les erreurs d'alimentation, OB82 pour l'alarme de diagnostic, OB85 pour les erreurs d'exécution du programme

4.3.2. Implémentation sous WinCC :

Pour la simulation et la visualisation des programmes STEP7 et pour contrôler facilement les opérations d'automatisations, nous avons confectionné une interface graphique interactive à l'aide du logiciel WINCC. Cette interface est composée de trois vues : une vue principale qui contient le procédé, une vue pour la régulation de niveau et une vue pour les informations sur notre matériel.

Dans la vue principale de l'interface graphique, on trouve :

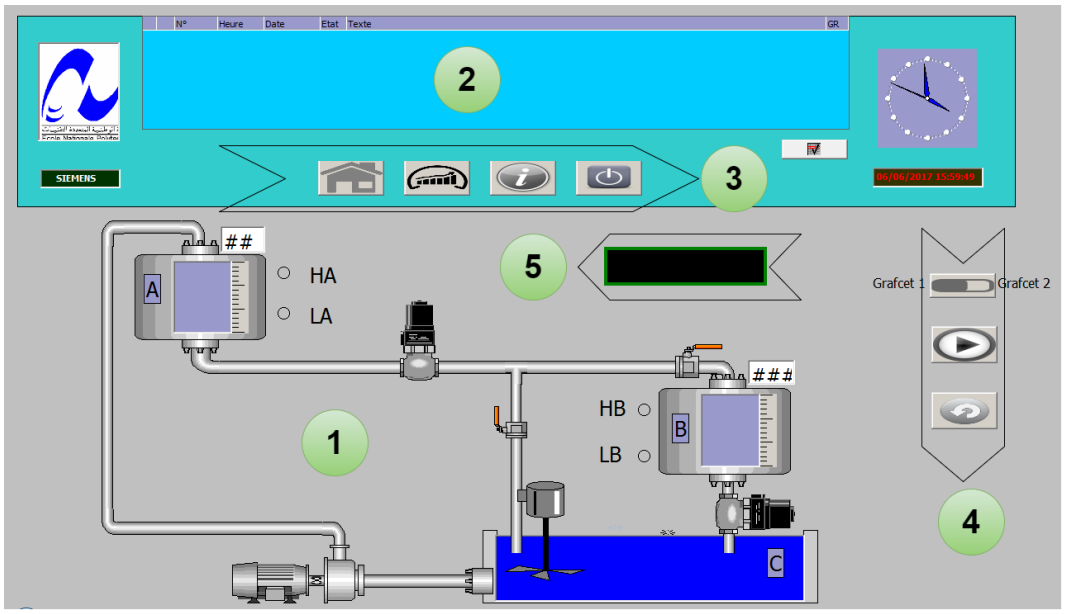


Figure 4.15 : Vue principale

1- un schéma représentatif de la station de pompage
2- Une fenêtre d'alarme pour l'affichage des alarmes et des avertissements
3- 3 boutons pour la permutation entre les Vues et un pour quitter le Wincc Runtime
4- Trois boutons, Un bouton qui permet le choix entre le Grafcet TOR et le Grafcet Analogique, et les deux autres c'est un bouton Marche et Reset
5- Fenêtre pour les voyons

Dans la vue présente sur la figure 4.15, on peut exécuter plusieurs tâches par exemple visualiser le niveau de la cuve A ou cuve B.
Dans le bargraphe ou le champ E, on peut aussi suivre l'évolution du niveau d'une cuve par le biais des graphes. Pour ce faire, il faut juste cliquer sur la cuve correspondante.

Exemple : visualisation du niveau de la cuve B

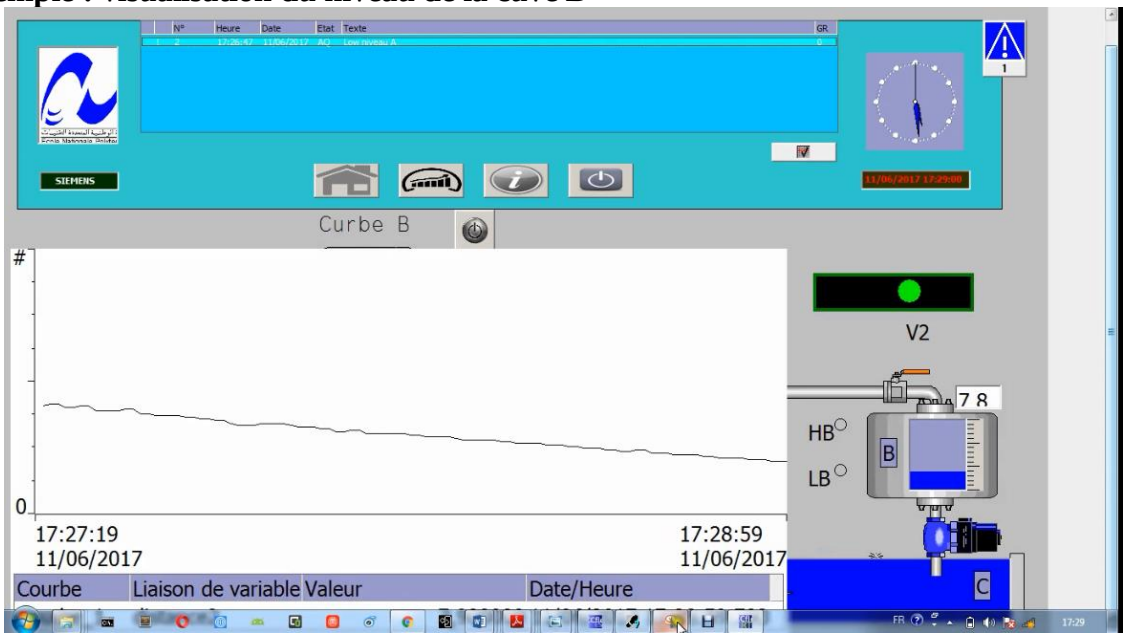


Figure 4.16 : Vue de l'évolution de la cuve B

Dans la vue présente sur la figure 4.17 (régulation de niveau), on trouve :

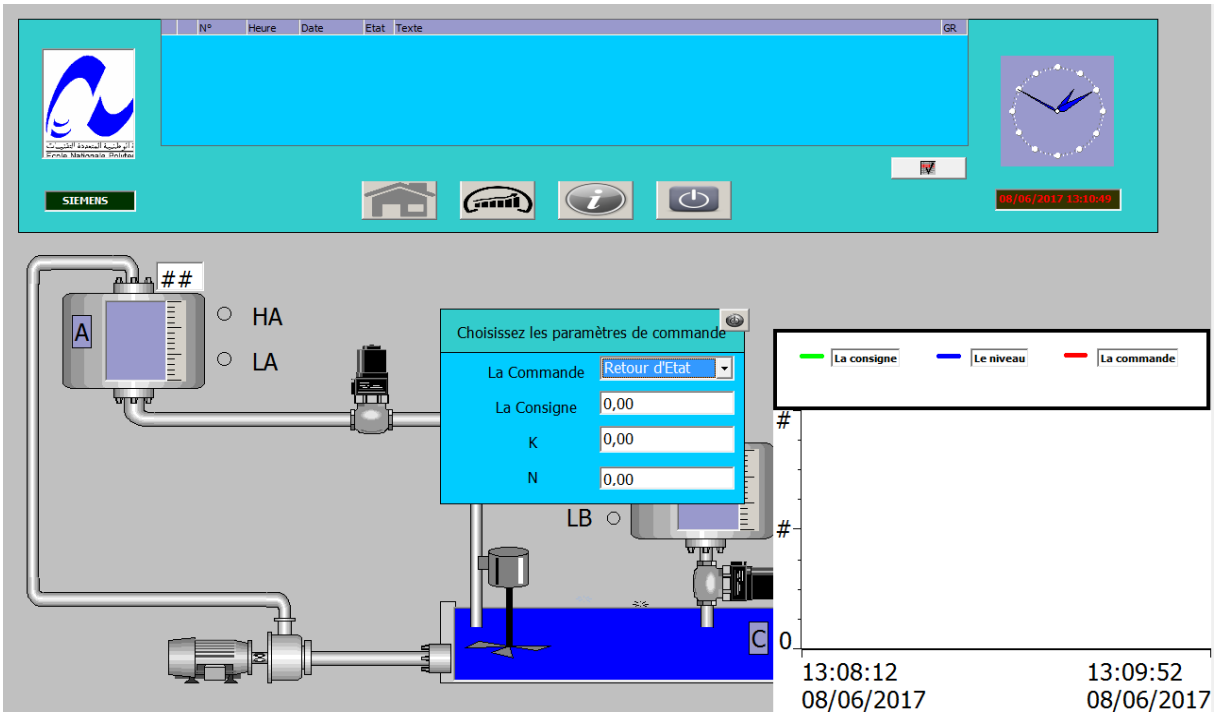


Figure 4.17 : Vue des choix de loi de commande pour la régulation de niveau

Dans la vue présente sur la figure 4.18 (informations des matériels), on trouve :

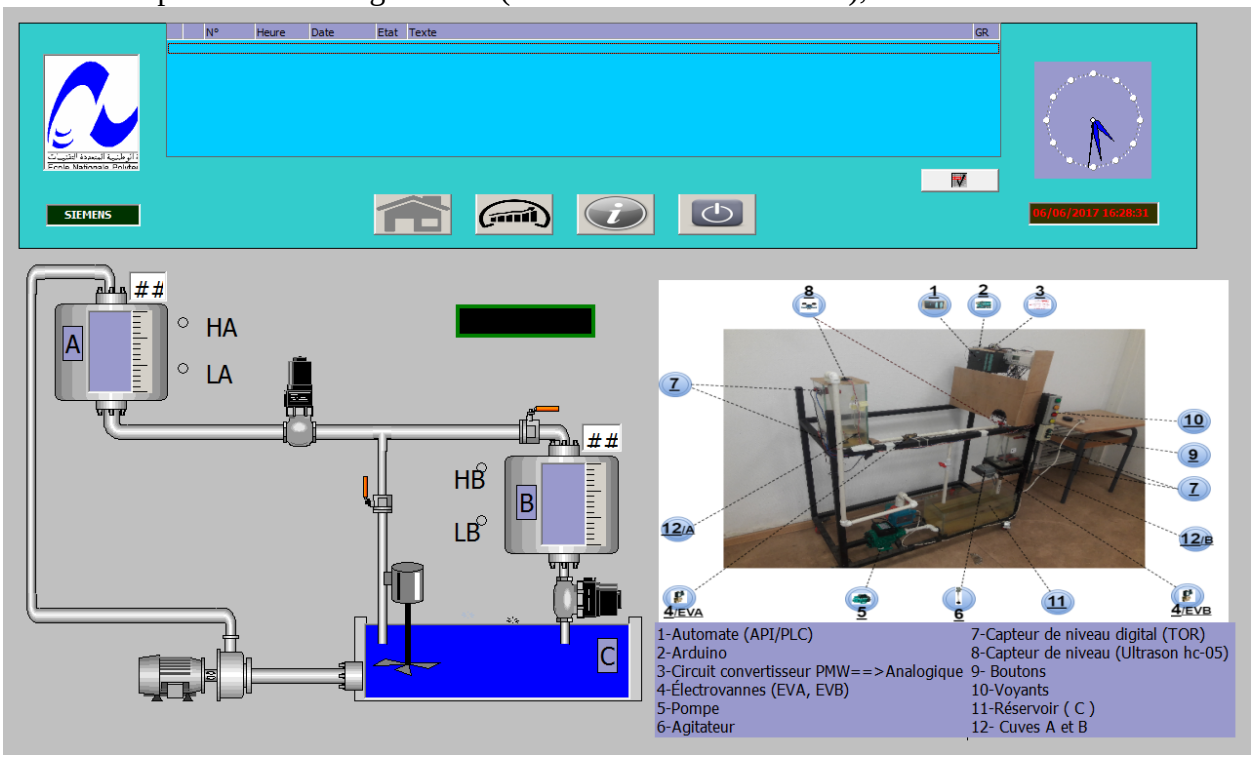


Figure 4.18 : Une vue qui présente les informations des matériels

Dans cette vue, on peut voir les caractéristiques des différents matériels utilisés, pour cela il suffit juste d'appuyer sur le matériel (photo ou nom) dans l'interface graphique.

Exemple : appuyer sur la pompe ou sur son nom

Figure 4.19 : affichage des informations de la pompe

Conclusion :

Nous avons pu voir à travers ce chapitre l'application de concepts théoriques d'identification des systèmes. En effet, l'identification du système pompe-niveau nous a permis de mieux connaître ce système. Cette procédure sera une étape cruciale en vue de la régulation du système pompe-niveau.

Nous avons aussi procédé à la définition des cahiers de charges et leurs implémentations à l'aide des outils de programmation offerts par STEP7 et les outils d'interfaçage graphique réalisé à l'aide de WinCC.

Conclusion générale et perspectives:

Un élève ingénieur ne doit pas se satisfaire d'acquérir des connaissances théoriques, il est plutôt primordiale qu'il complète sa formation par une bonne expérience pratique pour mettre en œuvre, justement, ce qu'il a appris en théorie. Il doit aussi gérer le budget et les moyens disponibles et ne doit pas lâcher devant les problèmes rencontrés.

C'est ce qu'on a essayé de faire tout au long de notre projet, on a rencontré un problème de communication entre l'automate S7-300 et le capteur à base d'Arduino. En l'absence d'un capteur industriel, on a conçu un circuit de conversion PWM/analogique (interfaçage) pour transmettre l'information du capteur à l'automate.

Globalement, on a ajouté plusieurs fonctionnalités à cette station de pompage réalisée en 2015 par des étudiants polytechniciens [4] : le fonctionnement analogique, gestion et visualisation du procédé en temps réel, archivage des variables, alarmes et grandeurs du projet ….
Ce projet est une occasion formidable pour nous d'appliquer nos connaissances acquises le long de notre formation sur des systèmes réels qui ressemblent à ceux qui sont disponibles dans l'industrie, et d'affronter des problèmes d'ingénierie.

Enfin, nous espérons que notre travail sera utile aux étudiants en automatique, pour appliquer ce qu'ils apprennent aux cours de l'informatique industrielle et d'Automatique avancé et pourquoi pas d'identification, et qu'il participe à forger des ingénieurs automaticiens avec beaucoup d'expérience qui leur permet de transiter vers une nouvelle phase où ils seront obligés à manipuler des systèmes réels.

Perspectives

La maquette réalisée en mode logique et analogique, Il serait intéressant de réguler des grandeurs analogiques (niveau, débit, pression, …) et d'implémenter différents lois de commandes pour comprendre la différence entre ces dernières et assimiler les avantages de chacune.

On peut ajouter par exemple:

- Un variateur de vitesse monophasé pour réguler le niveau
- Un débitmètre pour mesurer le débit et une vanne proportionnelle pour le réguler.
- Capteur de pression

Bibliographie :

[1] Technologue Pro, Les Automates Programmables Industriels (API). [En ligne], [consulté le 20 mai 2017]. Disponible sur :

<http://www.technologuepro.com/cours-automate-programmable-industriel/Les-automates-programmables-industriels-API.htm>

[2] ZERARKA Houssem Eddine, KHELIFI Moufek. Gestion de la station CE117 par l'automate S7 314 IFM. 169 pages. Projet de Fin d'Etude : Automatique : Alger, Ecole Nationale Polytechnique : 2016.

[3] SIEMENS, « Programmer avec Step7 », SIMATIC, 2010.

[4] A.A.M.FOUKA, B.FERHAOUI. Réalisation d'une maquette de station de pompage à base d'automate programmable SIEMENS. 105 pages. Projet de Fin d'Etude : Automatique : Alger, Ecole Nationale Polytechnique : 2015.

[5] WinCC flexible 2008 SP2, WinCC flexible Information System.

[6] Youtube, liaison MPI [en ligne], [consulté le 23 juin 2017]. Disponible suer <https://www.youtube.com/watch?v=kN6uYAnNvMo&feature=youtu.be>

[7] SIEMENS, Home, [en ligne], [consulté le 23 juin 2017]. Disponible sur

<www.siemens.com>

[8] SIEMENS AG, «SIMATIC HMI WinCC V6.0, Documentation de base », 2003, 354 pages

[9] LECHALUPE, Julien. Cours d'initiation à Arduino [en ligne]. Mai 2014 [consulté le 20 mai 2017] Disponible sur :
<https://fablab.univ-tlse3.fr/wiki/images/9/92/Cours_arduino_v0.2.pdf >

[10] HACKADAY, Arduino Hacks, [en ligne], [consulté le 23 juin 2017]. Disponible sur :
< http://www.hackaday.com/category/arduino-hacks/>

[11] BACHELARD, Lucien. HC-SR04 - Module de détection aux ultrasons [en ligne], [consulté le 23 juin 2017]. Disponible sur :

<https://www.gotronic.fr/pj2-hc-sr04-utilisation-avec-picaxe-1343.pdf >

[12] HORSIN MOLINARO Hélène- BARBOT Jean-Pierre (ENS Cachan). Capteurs et chaîne d'acquisition [en ligne]. 18 juin 2015. [Consulté le 23 juin 2017]. Disponible sur :
<http://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/pedagogiques/6151/6151-capteur-et-chaine-dacquisition-ens.pdf>

[13] TEXAS INSTRUMENTS. LMx58-N Low-Power, Dual-Operational Amplifiers [en ligne]. Janvier 2000. [Consulté le 23 juin 2017]. Disponible sur :

<http://www.ti.com/lit/ds/symlink/lm158-n.pdf>

[14] BONNET, Pierre. Modélisation et Identification des Processus [en ligne]. 2010 [Consulté le 23 juin 2017]. Disponible sur :
<http://www-lagis.univ-lille1.fr/~bonnet/identif/Cours_Ident.pdf>

[15] Electronix. DIFFÉRENTS MONTAGES AVEC AMPLI OP. [En ligne]. [Consulté le 20 mai 2017]. Disponible sur :
<http://electoonix.free.fr/cours/amplimontage.htm>

LISTE DES ANNEXES

ANNEXES

ANNEXE 1

Fonctionnalités du logiciel STEP7

1. Langages de programmation :

Les langages de programmation CONT, LIST et LOG pour S7-300/400 font partie intégrante du logiciel de base.

- **Le schéma à contacts (CONT) :**

C'est un langage de programmation graphique. La syntaxe des instructions fait penser aux schémas de circuits. CONT permet de suivre facilement le trajet du courant entre les barres d'alimentation en passant par les contacts, les éléments complexes et les bobines.

- **La liste d'instructions (LIST) :**

C'est un langage de programmation textuel proche de la machine. Dans un programme LIST, les différentes instructions correspondent, dans une large mesure, aux étapes par lesquelles la CPU traite le programme. Pour faciliter la programmation, LIST a été complété par quelques structures de langage évolué (comme, par exemple, des paramètres de blocs et accès structurés aux données).

- **Le logigramme (LOG) :**

C'est un langage de programmation graphique qui utilise les boîtes de l'algèbre de Boole pour représenter les opérations logiques. Les fonctions complexes comme par exemple les fonctions mathématiques, peuvent être représentées directement combinées avec les boîtes logiques. Nous disposons des logiciels de langage optionnels (langage évolué) suivants pour la programmation des automates programmables SIMATIC S7-300/400.

- **GRAPH :**

C'est un langage de programmation permettant la description aisée de commandes séquentielles (programmation de graphes séquentiels). Le déroulement du processus y est subdivisé en étapes.

Celles-ci contiennent en particulier des actions pour la commande des sorties. Le passage d'une étape à la suivante est soumis à des conditions de transition.

- **Hi-Graph :**

C'est un langage de programmation permettant la description aisée de processus asynchrones non séquentiels sous forme de graphes d'état. A cet effet, l'installation est subdivisée en unités fonctionnelles pouvant prendre différents états. Ces unités fonctionnelles peuvent se synchroniser par l'échange de messages.

- **SCL :**

C’est un langage évolué textuel conforme à la norme DIN EN 61131-3. Il comporte des éléments de langage que l'on trouve également sous une forme similaire dans les langages de programmation Pascal et C. SCL convient donc particulièrement aux utilisateurs déjà habitués à se servir d'un langage de programmation évolué. Nous pouvons, par exemple, faire appel à SCL pour programmer des fonctions très complexes ou se répétant souvent.

- **CFC :**

Pour S7 et M7 est un langage de programmation permettant l'interconnexion graphique de fonctions existantes. Ces fonctions couvrent un large éventail allant de combinaisons logiques simples à des régulations et commandes complexes. Un grand nombre de ces fonctions est disponible sous la forme de blocs dans une bibliothèque. La programmation se fait en copiant des blocs sur un diagramme et en reliant les connecteurs de blocs par des lignes.

2. Configuration matérielle :

Dans une table de configuration, nous définissons les modules que nous allons mettre en œuvre dans notre solution d'automatisation ainsi que les adresses permettant d'y accéder depuis le programme utilisateur. Nous pouvons en outre y paramétrer les caractéristiques des modules (les manipulations de base pour la configuration matérielle).

3. Test de programmes

Pour effectuer un test, nous avez la possibilité d'afficher les valeurs de variables depuis notre programme utilisateur ou depuis une CPU, d'affecter des valeurs à ces variables et de créer une table des variables que nous souhaitons afficher ou forcer.

4. Surveillance du fonctionnement, diagnostic du matériel

On détermine les causes du défaut d'un module en affichant des informations en ligne relatives à ce module. On détermine les causes d'un défaut dans le déroulement d'un programme utilisateur à l'aide de la mémoire tampon de diagnostic et du contenu des piles. On peut en outre vérifier si un programme utilisateur est exécutable sur une CPU donnée.

5. Configuration de réseaux et de liaisons de communication : [3]

La condition requise à l'établissement d'une communication est l'existence d'un réseau préalablement configuré. Nous devons à cet effet créer les réseaux auxiliaires nécessaires à notre projet, définir leurs propriétés et pour les stations mises en réseau, les caractéristiques de connexion au réseau ainsi que, le cas échéant, les liaisons de communication requises.

6. Création d’une structure programme complète

La mise en place d'une solution d'automatisation avec STEP 7 nécessite la réalisation des tâches fondamentales suivantes :

6.1 Création du projet SIMATIC Step7

Un projet comprend deux données essentielles, les programmes et la configuration du matériel, on peut commencer par définir l’une ou l’autre, mais tout d’abord il faut démarrer le programme SIMATIC Manager. Ce programme est l’interface graphique qui permet la manipulation du projet et l’accès aux autres programmes de STEP7.

Pour en créer un nouveau, il suffit de cliquer sur le bouton « Nouveau projet», attribuer un nom et valider. Ensuite il faut choisir une station de travail. Une station SIMATIC représente une configuration matérielle S7 comportant un ou plusieurs modules programmables. Il existe différents types:

**SIMATIC 400

**SIMATIC 300

** SIMATIC S5

** SIMATIC H : Automate insensible aux défaillances, il se compose de 2 CPU du même type, en cas de problème elle commute de l'une vers l'autre sans perte de données.

** SIMATIC PC : ou Station PC, représente un PC contenant des composants

SIMATIC : des applications (WinCC, par ex.), ou une carte CPU enfichée dans le PC.

** PG/PC : Outils de programmation pour contrôleurs SIMATIC, c'est une console de programmation compatible avec le milieu industriel.

** Autres stations : ce sont soit des appareils d'autres fabricants ou bien des stations de SIMATIC S7 contenus dans un autre projet.

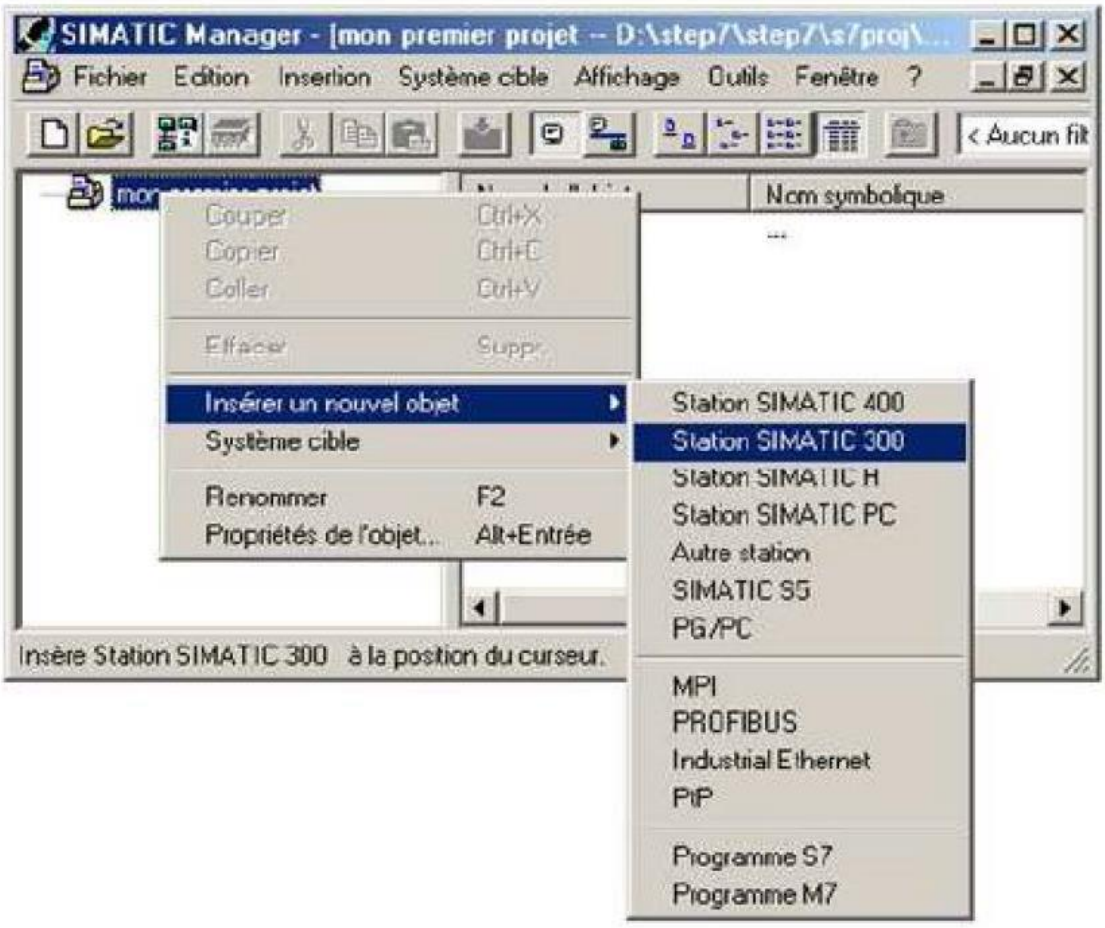


Figure : Choix de la station SIMATIC 300

Pour commencer, le plus simple est de configurer le matériel, d'éditer les programmes puis les charger dans la CPU. Avec un double clic sur « Matériel », on démarre l'application HW Config.

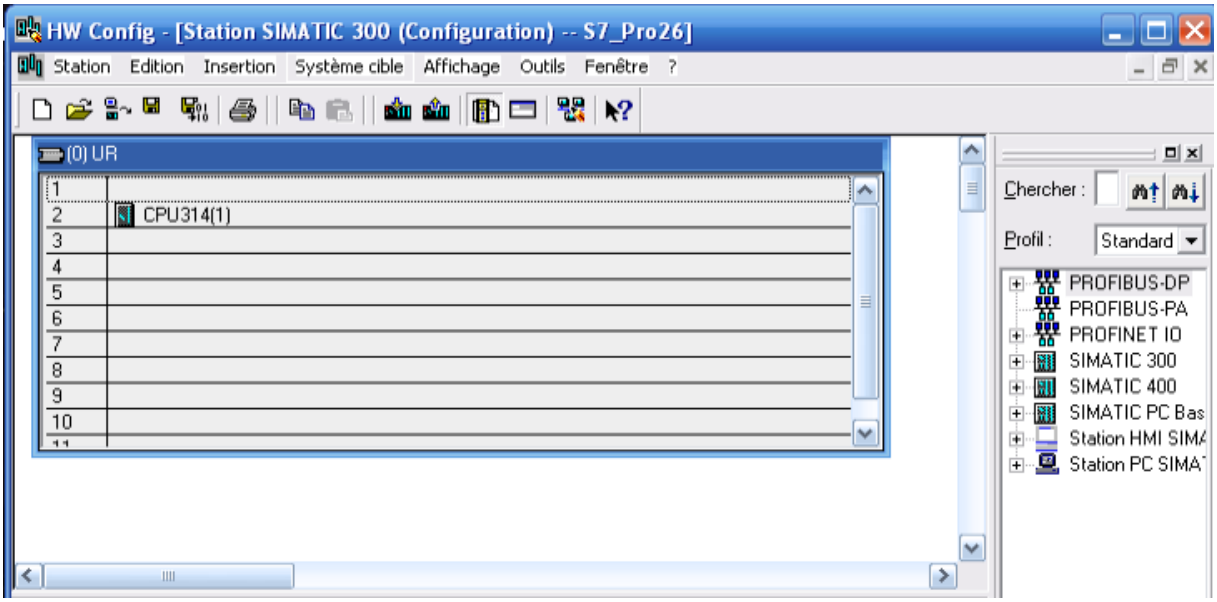


Figure : Configuration matériels pour la station S300

6.2 Configuration du matériel

Pour configurer le matériel, il suffit de faire glisser des éléments du catalogue dans l'emplacement approprié, on choisit le Rack, l'alimentation, la CPU et les E/S...
On distingue les modules qu'on peut affecter à chaque type de station :

- C7 : Système intégré compact qui regroupe automate programmable et interface homme machine (pupitre opérateur) pour la réalisation de commandes de machines sous encombrement réduit.
- CP : Communication Processor, module de communication (PROFIBUS, Industriel Ethernet,….).
- FM : (Function Module), il regroupe les modules de fonctions (régulation, comptage…).
- IM : Coupleurs d'extension, il permet l'ajout d'autres modules.
- M7 : Modules d'extension et cartouches interface pour SIMATIC M7.
- PS : Module d'alimentation.
- Rack : Support mécanique.
- Routeur : Relie Industriel Ethernet à PROFIBUS.
- SM : Signal Module, c'est le module d'E/S, il contient le AI module d'entrées analogiques, le AO module de sorties analogiques, le DI module d'entrées TOR et le DO module de sorties TOR.
- CPU : L'Unité Centrale, noté CPU xxx a b.
    - ✓ xxx est la famille de la CPU
    - ✓ a, b sont les propriétés de la CPU (éléments additionnels, port de communication…).

Par exemple :
C : compact, la CPU intègre des modules E/S ainsi que des fonctions spécialisées.
PtP : Peer to Peer, la CPU intègre un port de communication Point to Point.
H : Fault-tolerant, des unités de traitements insensibles aux défaillances
DP : Decentralized Periphery, la CPU intègre un port de communication

PROFIBUS.

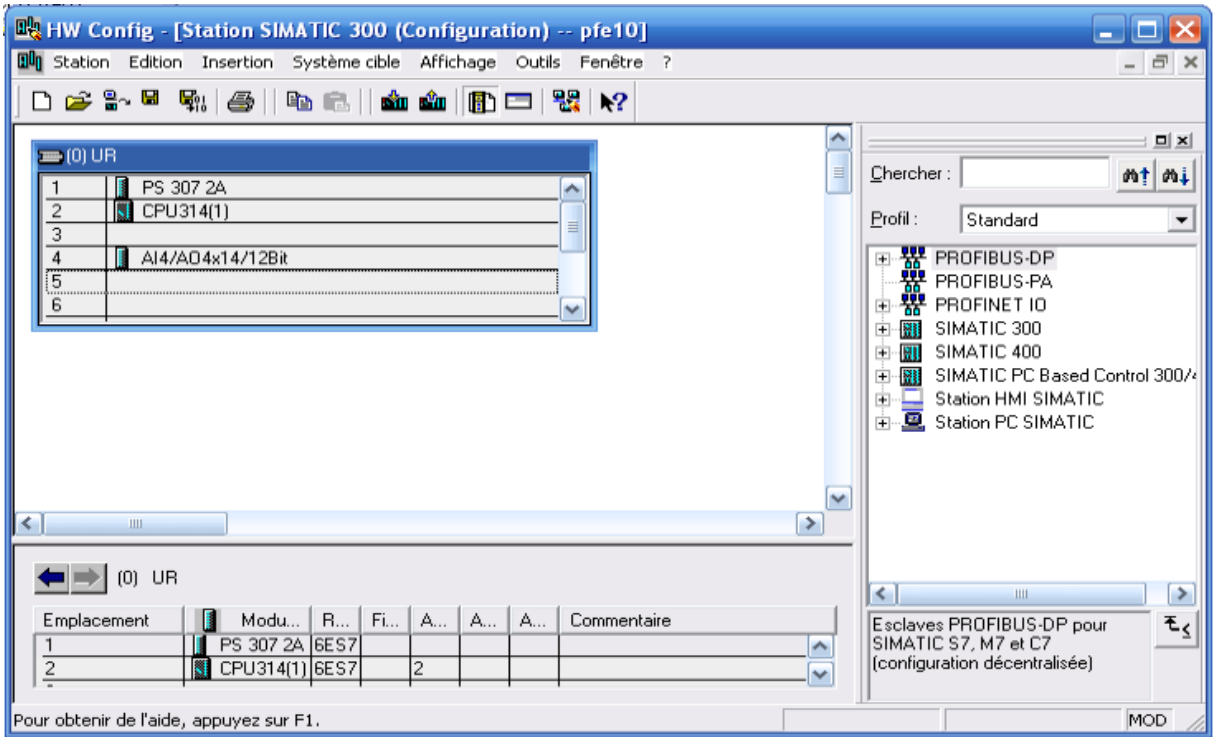


Figure : Sélection des modules

6.3 Définition des mnémoniques

De retour dans le SIMATIC Manager, on trouve de nouveaux éléments. On commence par créer les mnémoniques dans la section programmes.

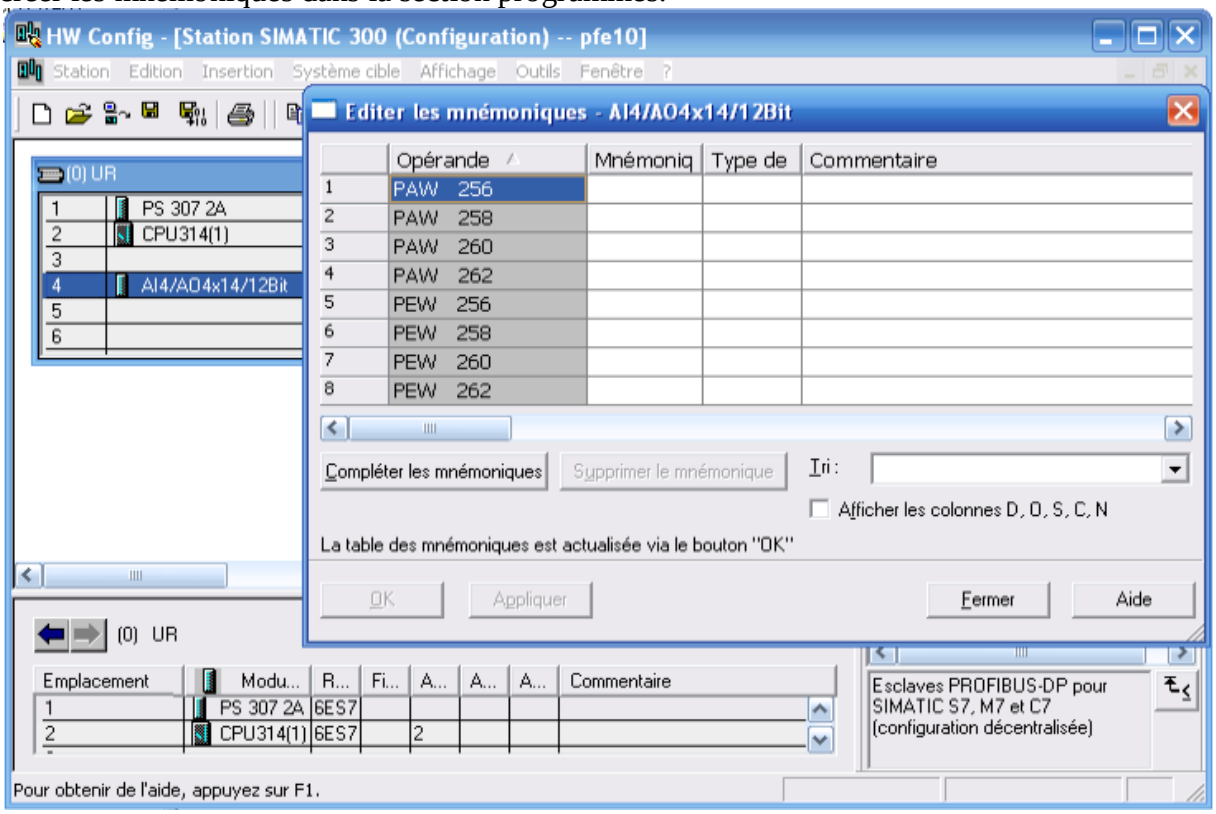


Figure : Edition des Mnémoniques.

En affectant des noms symboliques aux adresses absolues, les programmes deviennent plus lisible, faciles à corriger et à mettre à jour.

Il y a quatre différents types d'opérande : le bit, l'octet, le mot et le double mot. Ces types définissent l'accès à une zone mémoire. Pour chaque opérande un certain type de données est permis :

- Pour le bit : BOOL : variable booléenne (True ou False, 1 ou 0).
- Pour l'octet : deux types de données sont possibles :
- BYTE: nombre hexadécimal de B#16#0 à B#16#FF.
- CHAR : Caractère ASCII, 'A', 'B'…
- Pour le mot : quatre types de données sont possible :
- WORD : nombre hexadécimal de W#16#0 à W#16#FFFF.
- INT : nombre entier de -32768 à 32767.
- S5TIME : Durée S7 en pas de 10 ms (valeur par défaut), de S5T#0H_0M_0S_10MS à S5T#2H_46M_30S_0MS.
- DATE : Date en incréments de 1 jour, de D#1990-1-1 à D#2168-12-31.

Pour le double mot : cinq types de données :

- DWORD: nombre hexadécimal de DW#16#0000_0000 à DW#16#FFFF_FFFF.
- DINT : nombre entier de L#-2147483648 à L#2147483647.
- REAL : nombre à virgule flottante, Limite supérieure : 3.402823e+38 Limite inférieure : 1.175 495e-38.
- TIME : Durée en incréments de 1 ms, de -T#24D_20H_31M_23S_648MS àT#24D_20H_31M_23S_647MS.
- TIME_OF_DAY : Heure en pas de 1ms, de TOD#0:0:0.0 à TOD#23:59:59.999.

6.4 Edition des programmes

Dans la section 'bloc' du SIMATIC Manager, on trouve par défaut le bloc d'organisation 1 'OB1'qui représente le programme cyclique. On peut rajouter d'autres blocs à tout moment par un clic droit dans la section Bloc de SIMATIC Manager.

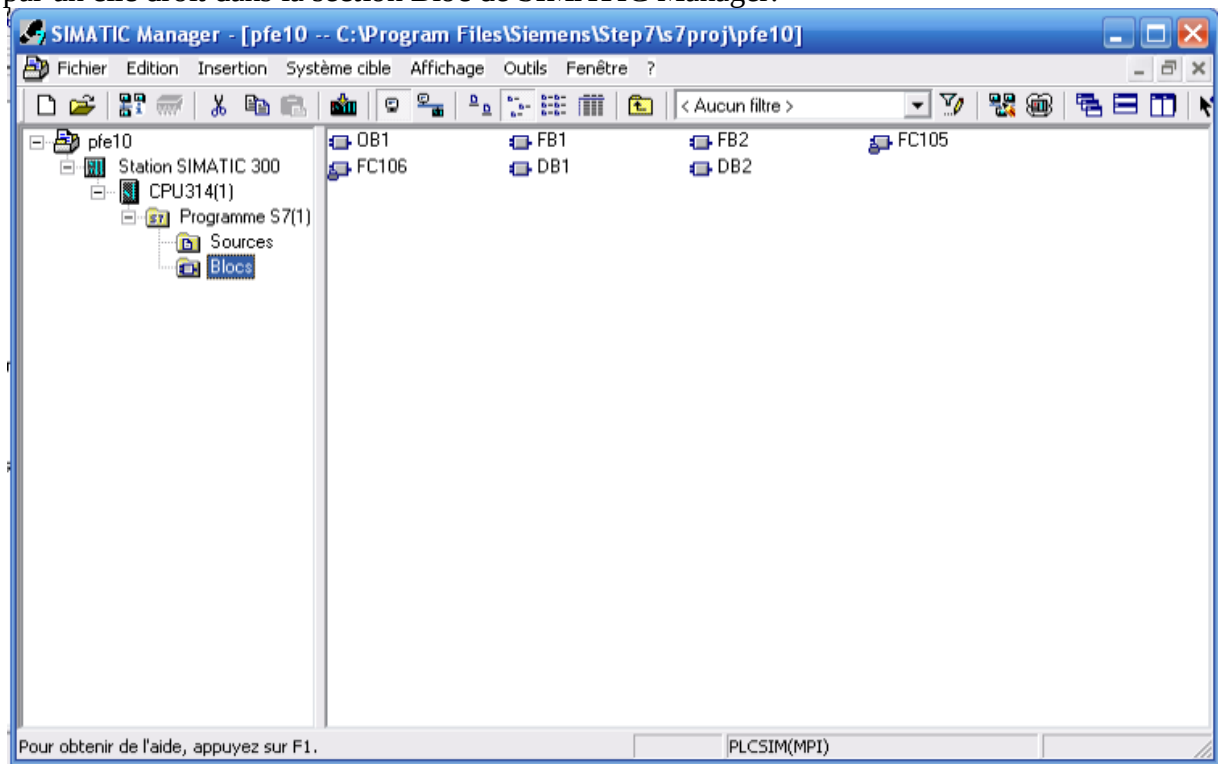


Figure : Edition des programmes

Deux programmes différents s'exécutent dans la CPU: le système d'exploitation et le programme utilisateur.
Le système d'exploitation, organise toutes les fonctions et procédures dans la CPU qui ne sont pas liées à une tâche d'automatisation spécifique. Il gère le déroulement du démarrage à chaud et du redémarrage, l'actualisation de la mémoire image des entrées et l'émission de la mémoire image des sorties, l'appel du programme utilisateur, la gestion des zones de mémoire l'enregistrement des alarmes et l'appel des OB d'alarme…
Le programme utilisateur contient toutes les fonctions nécessaires au traitement des tâches d'automatisation spécifique. Ce programme doit être créé et chargé dans la CPU par l'utilisateur. Il détermine les conditions pour le démarrage à chaud et le redémarrage de la CPU (par exemple, initialiser des signaux), il traite les données du processus (par exemple, combiner des signaux binaires, lire et exploiter des valeurs analogiques), il doit réagir aux alarmes et traiter les perturbations dans le déroulement normal du programme.

STEP7 permet de structurer le programme utilisateur en le subdivisant en différentes parties autonomes ou dépendantes. Ceci permet d'écrire des programmes importants mais clairs, simples à tester et à modifier.

- **Blocs d'organisation**

Les blocs d'organisation (OB) constituent l'interface entre le système d'exploitation et le programme utilisateur. Ils sont appelés par le système d'exploitation et gèrent le traitement du programme cyclique et des programmes déclenchés par alarmes, ainsi que le comportement à la mise en route de l'automate programmable et le traitement des erreurs.
Les blocs d'organisation définissent l'ordre (événements de déclenchement) dans lequel les différentes parties du programme sont traitées. L'exécution d'un OB peut être interrompue par l'appel d'un autre OB (les OB de priorité plus élevée interrompent les OB de priorité plus faible). Les blocs d'organisations les plus prioritaires sont ceux de la mise en route (OB100, OB101 et OB102) et le moins prioritaire est le cycle en arrière-plan (OB90).
On appelle alarmes les événements qui déclenchent l'appel d'un OB donné. Le type d'alarme définit la classe de priorité de celle-ci.

- **Fonctions et blocs fonctionnels**

On peut programmer chaque bloc d'organisation en tant que programme structuré en créant des fonctions (FC) et des blocs fonctionnels (FB).

- Les blocs fonctionnels (FB) sont des blocs de code associés à des blocs de données d'instance, dans lesquels sont sauvegardés les paramètres effectifs et les données statiques des blocs fonctionnels.
- Les fonctions (FC) sont des blocs de code sans rémanence, c'est-à-dire qu'ils ne sont pas associées à des blocs de données, les paramètres effectifs ne sont pas sauvegardés automatiquement.

De plus il existe les blocs fonctionnels système (SFB) et les fonctions système (SFC), qui sont des fonctions préprogrammées. Ils peuvent être appelés à partir du programme utilisateur. On trouve des fonctions système pour la copie de blocs de données, le contrôle du programme utilisateur, la gestion des alarmes horaires et temporisées…

- **Bloc de données**

Les blocs de données (DB) servent à l'enregistrement de données utilisateur. Les blocs de données globales servent à l'enregistrement de données qui peuvent être utilisées par tous les autres blocs. Les blocs de données d'instances sont affectés à des blocs fonctionnels. Les différents blocs cités ci-dessus peuvent être édités avec l'application « CONT LIST LOG ».

La programmation des blocs de codes peut se faire à l'aide de trois applications :

1. CONT LIST LOG : elle permet de programmer des blocs d'organisations OB, des blocs fonctionnels FB et des fonctions FC.

2. GRAPH : elle permet de programmer des blocs fonctionnels FB.

- SCL : elle permet de créer des sources de code. Une source de code est un fichier texte, qui contient une suite d'instructions, une fois compilé il peut être transféré dans la CPU. On peut trouver dans un même fichier source tout le programme utilisateur, c'est-à-dire les blocs d'organisations, les blocs fonctionnels et les fonctions.

6.5 Simulation de modules Avec le logiciel PLCSIM :

S7-PLCSIM dispose d'une interface simple permettant de visualiser et de forcer les différents paramètres utilisés par le programme (comme, par exemple, d'activer ou de désactiver des entrées).
En outre, S7-PLCSIM possède les fonctions suivantes :

- On peut créer des « fenêtres » dans lesquelles on a la possibilité d'accéder aux zones de mémoire d'entrée et de sortie, aux accumulateurs ainsi qu'aux registres de la CPU de simulation. On peut également accéder à la mémoire par adressage symbolique (il faut juste charger la table des mnémoniques dans 'options', puis sur 'outils' 'insérer mnémoniques').

- On peut sélectionner l'exécution automatique des temporisations ou encore les définir et les réinitialiser manuellement.

- On a la possibilité de changer l'état de fonctionnement de la CPU (STOP, RUN et RUNP) [4[comme pour une CPU réelle. De plus, on dispose d'une fonction de pause qui permet d'interrompre momentanément la CPU, sans affecter l'état du programme.

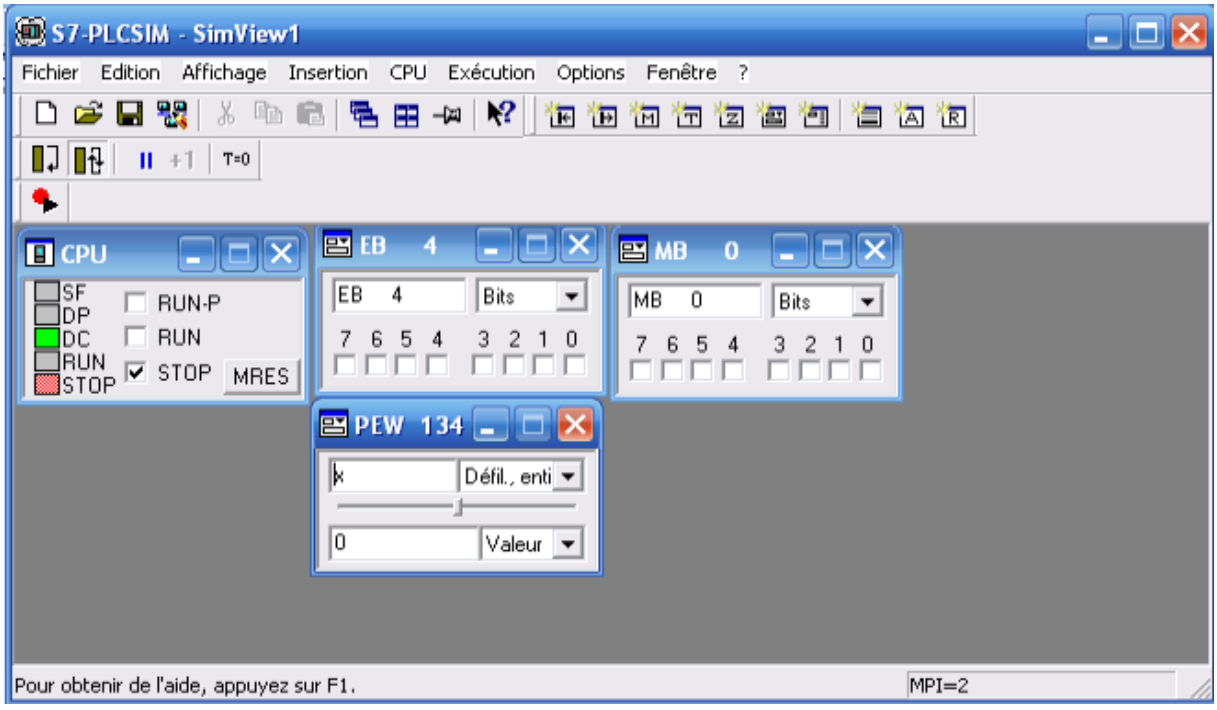


Figure : Logiciel de simulation PLC-SIM

6.6 Chargement du programme dans la CPU :

Une fois la configuration, le paramétrage et la création du programme terminés, on peut transférer le programme utilisateur dans le système cible. La CPU contient déjà le système d'exploitation.

6.7 Surveillance du fonctionnement et diagnostic du matériel

La détermination les causes d'un défaut dans le déroulement d'un programme utilisateur se fait à l'aide de la « Mémoire tampon de diagnostic », accessible depuis le SIMATIC Manager.

ANNEXE 2

ARDUINO

2.1 Présentation de la carte

2.1.1. Caractéristiques techniques de l'Arduino UNO

Un des modèles les plus répandu de carte Arduino est l'Arduino UNO (voir Fig.3.2). C'est la première version stable de carte Arduino. Elle possède toutes les fonctionnalités d'un microcontrôleur classique en plus de sa simplicité d'utilisation.

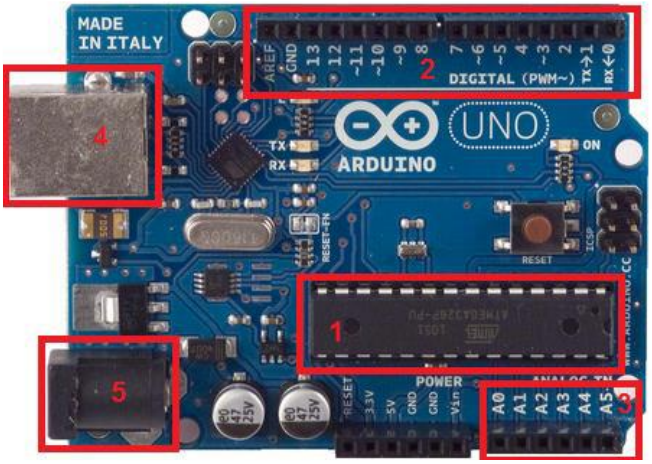


Figure - Arduino UNO

(1) : Microcontrôleur ATmega328P (cadence : **16Mhz**, **SRAM : 2ko, mémoire flash : 32ko**, **EEPROM : 1 ko)**

(2) : 14 Broches d'entrée/sortie numérique dont **6** pouvant générer des **PWM**

(3) : **6 entrées analogiques.**

(4) : Port USB pour la communication/alimentation.

(5) : Connecteur *Power Jack pour l'alimentation*

2.1.2. Un atout : les shields

Pour la plupart des projets, il est souvent nécessaire d'ajouter des **fonctionnalités** aux cartes Arduino. Plutôt que d'ajouter soit même des composants extérieurs (sur une platine d'essai, circuit imprimé, etc.), il est possible d'ajouter des ***shields***. Un shield est une carte que l'on connecte directement sur la carte Arduino qui a pour but d'ajouter des composants sur la carte. Ces *shields* viennent généralement avec une **librairie** permettant de les contrôler. On retrouve par exemple, des *shields* Ethernet, de contrôle de moteur, lecteur de carte SD, etc.

Le principal avantage de ces *shields* est leurs simplicités d'utilisation. Il suffit des les emboiter sur la carte Arduino pour les connecter, les circuits électronique et les logiciels sont déjà faits et on peut en empiler plusieurs. C'est un atout majeur pour ces cartes pour pouvoir tester facilement de nouvelles fonctionnalités.

2.2. Présentation du logiciel

2.2.1 IDE Arduino

IDE Arduino est un environnement de développement libre, gratuit et disponible sur l'adresse http://arduino.cc/en/main/software.

L'interface de l'IDE Arduino est plutôt simple (voir Fig.3.3), il offre une interface minimale et épurée pour développer un programme sur les cartes Arduino. Il est doté d'un éditeur de code avec **coloration syntaxique** et d'une **barre d'outils rapide**. Ce sont les deux éléments les plus importants de l'interface, c'est ceux que l'on utilise le plus souvent. On retrouve aussi une barre de menus plus classique qui est utilisé pour accéder aux fonctions avancées de l'IDE. Enfin, une **console** affichant les résultats de la compilation du code source, des opérations sur la carte, etc.

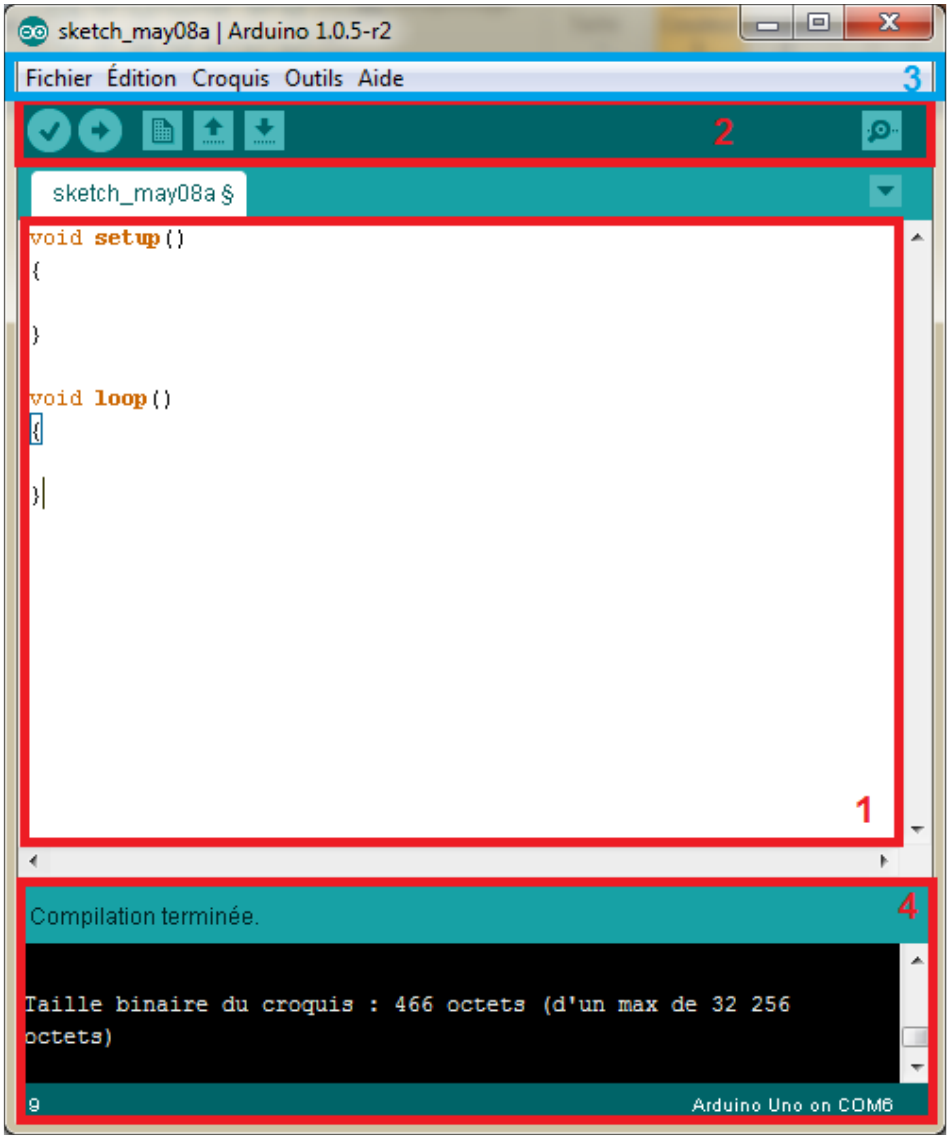


Figure - IDE Arduino

1 : un éditeur de code avec **coloration syntaxique**

2 : une **barre d'outils rapide**

3 : barre de menus

3 : une **console**

2.2.2 Langage Arduino

Le langage Arduino est inspiré de plusieurs langages. On retrouve notamment des similarités avec le C, le C++, le Java et le Processing. Ce langage impose une structure particulière typique de l'informatique embarquée. La fonction ***setup*** contiendra toutes les opérations nécessaires à la configuration de la carte (directions des entrées sorties, débits de communications série, etc.). La fonction ***loop*** est exécutée en boucle après l'exécution de la fonction *setup*, tant que la carte n'est pas mise hors tension ou redémarrée (par le bouton ***reset***). Cette boucle est absolument nécessaire sur les microcontrôleurs étant donné qu'ils n'ont pas de système d'exploitation. En effet, si l'on omettait cette boucle, à la fin du code produit, il sera impossible de reprendre la main sur la carte Arduino qui exécuterait alors du code aléatoire.

Au niveau de la **syntaxe**, on retrouve des similarités avec les langages précédemment cités. La déclaration des variables se fait généralement dans l'espace global (de façon à partager les variables les plus importantes entre les deux fonctions principales). On retrouve les types de base suivant :

Nom	Contenu	Taille (en octet)	Plage de valeurs
(unsigned) char	Entier ou caractère	1	(0->255) -128 -> 127
(unsigned) int	Entier	2	(0->65 535) -32 768 -> 32 767
(unsigned) long	Entier	4	(0 -> 4 294 967 295) -2 147 483 648 -> 2 147 483 647
float/double	Nombre à virgule flottante	4	-3,4028235E+38 -> 3,4028235E+38
String	Chaine de caractères (Objet)	variable	Aucune
boolean	Booléen	1	True / False

Tableau : Types de variables

La déclaration des variables suit cette syntaxe:

(const) <type> <nom>([<longueur du tableau>]) (= valeur);

Exemples :

```
const int constante = 12 ;

float univers = 42.0 ;

char lettre = 'b' ;

String chaine = "Hello World " ;

long tableau[12] ;

boolean vrai = true ;
```

On retrouve les opérateurs les plus courants pour les types de bases. Parmi eux : = (affectation), == (comparaison), != (différence), <, >, <=, >=, && (et logique), || (ou logique), ! (non logique). On retrouve aussi les opérateurs mathématiques (+, -, *, /, %)

et les opérateurs logiques bit à bit (^ (XOR), & (et), | (ou), ~ (non), << (décalage logique à gauche), >> (décalage logique à droite)).

Les **structures de contrôle** sont elles aussi similaires aux langages de références. On y retrouve toutes les structures de contrôle standard, conditions, boucle, switch, fonctions, etc. On peut aussi écrire des structures et des classes. Chaque structure de contrôle est suivie d'un bloc d'instructions délimitées par des accolades. Voici une liste des structures de contrôles les plus utilisées :

Nom	Utilité	Syntaxe
If-else	Condition logique	*If (<valeur booléenne>) { <instruction> } else { <instruction> }*
If-else if - else	Condition logique multiples	*If (<valeur booléenne>) { <instruction> } else if (<valeur booléenne>) { <instruction> } else { <instruction> }*
Switch	Sélecteur	*Switch (<variable>) { case <valeur> : <instruction> break ; default : <instruction> }*
While	Boucle	*While (<valeur booléenne>) { <instruction> }*
For	Boucle itérative	*For (<initialisation> ; <valeur booléenne> ; <évolution>) { <instruction> }*

Tableau : structures de contrôle

Voici quelques exemples d'utilisation de structure de contrôle.

```
If ( variable < 2 ){
    switch (variable) {
        case 1 :
            doSomething(10) ;
        break ;
        case 0 :
            doSomething(1) ;
        default :
doSomething(0) ;
            }
                } else if ( variable > 4 ) {
                        while ( variable != 10 ) {
```

```
                                                    variable =
doSomethingElse(variable-1) ;
                                                        }
                                                }
        else {
            for ( int i = 0 ; i < 10 ; i++ ) {
                variable = variable *2 ; }
        }
```

Les **fonctions**, les **structures** et les **classes** se déclarent de la même façon qu'en **C++**. Elles se déclarent sous cette forme :

Structure :
struct <nom> { <type> <nom du champ> ; } ;

Fonction :

<type de retour> <nom>(<paramètre>) { <instruction> }

Classe :

class <nom> {

public : <attributs et champ publics>

private : <attributs et champ privés>

protected : <attribut et champ privés>

} ;

2.3 Fonctionnalité de base :

2.3.1 Les entrées/sorties :

Le langage Arduino vient avec un nombre important de **fonctions de base** permettant d'interagir avec son **environnement**. Les fonctions les plus utilisée sont les fonctions d'entrée/sorties. Ce sont elles qui permettent **d'envoyer** ou de **mesurer** une tension sur une des broches de la carte.

Dans un premier temps, avant d'effectuer une mesure ou d'envoyer une commande. Il est nécessaire de définir la **direction** des broches utilisées. Pour cela on fait appel à la fonction *pinMode* en lui donnant d'une part, la broche concernée, et d'autre part, la direction :

```
void setup() {
pinMode(1,OUTPUT) ; // Broche 1 en sortie
pinMode(2,INPUT) ; // Broche 2 en entrée
}
```

Une fois cette configuration faite, on peut procéder à l'utilisation des broches. Toutes les broches sont capables d'écrire et de lire des données numériques (c'est-à-dire des 0 (0V) ou des 1 (5V)). Mais, certaines disposent de fonctionnalité supplémentaire.

Tout d'abord, toutes les cartes Arduino possèdent des entrées analogiques. Ce sont les broches A0-A1-A2 etc. Elles permettent de lire des tensions analogiques (comprise entre 0 et 5V) et de le convertir en entier (compris entre 0 et 1023) proportionnellement à la tension mesurée. Certaines cartes Arduino possède des sorties analogique faisant l'opération inverse (met une tension sur la broche proportionnellement à l'entier donné), mais ce n'est pas le cas pour l'Arduino UNO.

Pour pouvoir tout de même contrôler des composants autrement qu'en « *tout ou rien* » il est possible d'utiliser des broches PWM. Ce sont les broches annotés par un tilde ~ sur la carte. Les PWM (Pulse Width Modulation) sont utilisées pour synthétiser des signaux analogiques en modulant le temps passé à l'état 1 (5V). Le signal obtenu est représenté dans la figure 3.4. En utilisant une fréquence relativement élevée, les PWM permettent de commander certains composants comme si il recevait une tension analogique. Cela provient du fait que les composants utilisés dans l'électronique analogique, ne changes pas d'états instantanément. Par exemple, une ampoule à incandescence reste chaude et éclaire un court instant après avoir été éteinte. Ce phénomène est généralement invisible à l'oeil nu. Grâce à elles, on pourra par exemple faire varier l'intensité d'une LED. La plupart des cartes Arduino utilisent des PWM cadencées à **490Hz** environ.

Toutes ces fonctionnalités sur les broches d'entrées sorties sont utilisables par le biais de quatre fonctions :

digitalRead(pin) : mesure une donnée numérique sur une des broches, la broche en question doit être réglée en entrée.

digitalWrite(pin, value) : écrit une donnée numérique sur une des broches, la broche concernée doit être réglée en sortie. Le paramètre *value* doit être égal à *HIGH* (état 1 soit 5V) ou *LOW* (état 0 soit 0V).

analogRead(pin) : mesure une donnée analogique sur une des broches (compatible seulement), la broche doit être réglée sur entrée.

analogWrite(pin, value) : écrit une donnée sous forme de PWM sur une des broches (compatible uniquement), la broche doit être réglée en sortie. Le paramètre value doit être compris dans l'intervalle *[0;255]*

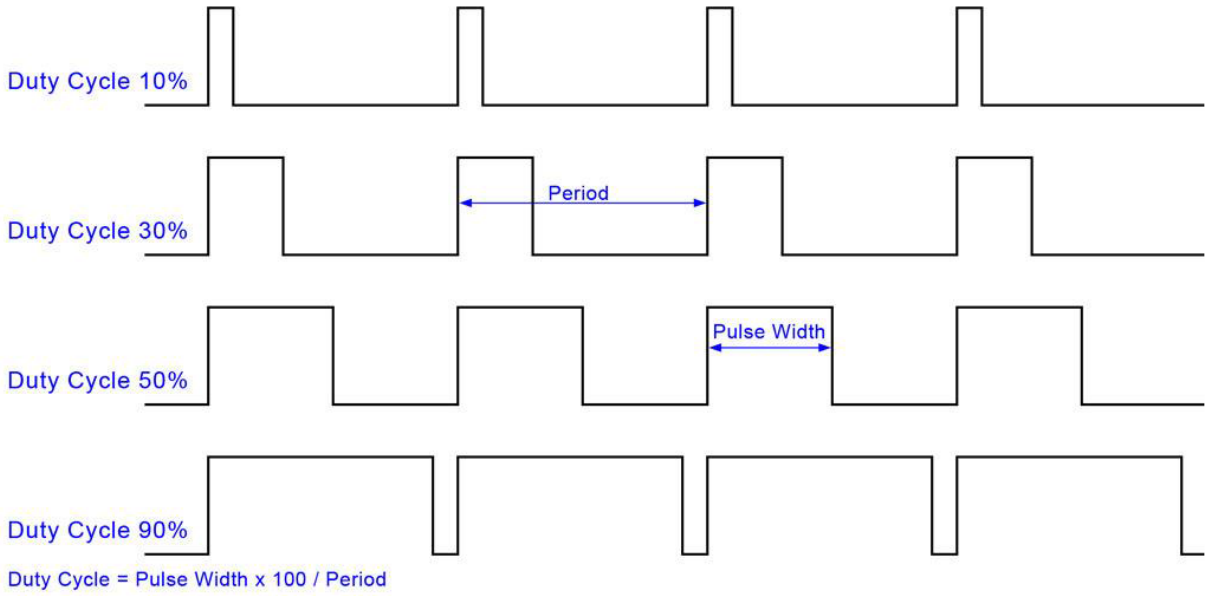


Figure : Signal PWM

2.3.2 La gestion du temps :

Pour la plupart des applications de domotique, il est nécessaire de faire intervenir des intervalles de temps. Par exemple, pour gérer le temps d'appui sur un bouton ou pour faire une sonnerie qui se répète un certains nombre de fois. Le langage Arduino fournis quelques fonctions permettant de gérer le temps.

Il est possible d'insérer une pause dans son programme pendant un instant. Pour cela, on utilise les fonctions *delay* et *delayMicroseconds* qui insère une pause suivant le paramètre passé (en milliseconde pour l'un, en microseconde pour l'autre). Cependant ces fonctions bloquent le microcontrôleur, on ne peut alors plus effectuer aucune action.

En plus d'insérer une pause, il est possible de mesurer le temps. De la même manière que les fonctions de délai, on utilise les fonctions *millis* et *micros* qui donnent le nombre de milliseconde (respectivement microseconde) depuis le lancement de la carte. Attention, ces fonctions incrémente une variable (interne). Ces variables se remettent à zéro une fois le maximum atteint (*overflow*). La variable utilisée pour les millisecondes atteint sont maximum au bout de 49 jours et 17 heures et la variable utilisée pour les microsecondes au bout de 71 minutes et 34 secondes environ. Il faut donc faire attention lors de l'utilisation de ces fonctions pour des utilisations longues durées.

2.3.3 Les Interruptions

Il est parfois nécessaire en informatique embarquée, d'attendre un événement externe (appui sur un bouton, données d'un capteur, etc.) pour effectuer une action. Pour ce type de problème, on utilise les interruptions. Les interruptions sont des portions de code (fonctions) appelés lorsque qu'un événement (interne ou externe) survient et à besoin d'être traité sur le champ. Il faut cependant faire attention, ce mécanisme interrompt le code exécuté, il est prioritaire par rapport au reste du code. Vu qu'il est possible de mesurer les événements ponctuellement (via les fonctions d'entrées/sorties) on utilise généralement les interruptions pour du code critique (arrêt d'urgence par exemple) ou des événements non-ponctuels (transmissions de données depuis un ordinateur par exemple).

Aussi, le nombre d'interruption externe est limité sur à 2 sur la plupart des cartes Arduino. Les interruptions sont utilisables sur les broches compatibles seulement (broches 2 et 3 sur l'Arduino UNO). Pour choisir la fonction et la broche utilisée pour l'interruption, on utilise la fonction *attachInterrupt*. On peut utiliser *detachInterrupt* pour supprimer l'interruption. Il est possible de partir en interruptions sur 4 types d'événements :

LOW : Lorsque la broche est à l'état 0 (0V)

RISING : Lorsque la broche passe de l'état 0 (0V) à l'état 1 (5V) (front montant).

FALLING : Lorsque la broche passe de l'état 1 (5V) à l'état 0 (0V) (front descendant).

CHANGE : Lorsque la broche change d'état (front montant et front descendant).

Voici un exemple d'utilisation :

```
volatile boolean etat = false ;

void appuiBouton() {

etat = !etat ; // Changement d'état }

 void setup() {

      pinMode(2,INPUT) ; // Broche 2 en entrée

       attachInterrupt(0,appuiBouton,RISING) ; // On attache à l'interruption 0 (broche 2) la fonction  appuiBouton sur un front montant

}
```

On remarque l'apparition du mot clef *volatile* avant la déclaration de la variable *etat*. Ce mot clef est nécessaire pour toutes les variables qui sont modifiée dans une interruption. Cela à une incidence sur la manière dont le compilateur traite l'accès à la variable.

Il est parfois nécessaire de désactiver temporairement les interruptions par exemple lorsque l'on exécute du code critique (activation d'un moteur, etc.). Deux fonctions permettent de changer l'activation des interruptions *interrupts* et *noInetrrupts* pour activer (respectivement désactiver) les interruptions.

2.3.4 Quelques librairies :

En plus de la simplicité du langage et des nombreuses fonctionnalités qu'offre. L'IDE vient avec un nombre important de librairies évitant ainsi d'implémenter des fonctions courantes dans l'informatique embarquée.

Une des librairies les plus utilisée est celle implémentant la communication série. La majorité des cartes Arduino possède un émulateur de connexion série pour communiquer au travers de l'USB. Ainsi, on peut communiquer avec l'ordinateur sur lequel la carte Arduino est connectée. Cela permet, par exemple, de déboguer un programme en affichant la valeur des variables ou simplement afficher la valeur des capteurs. Cette librairie à été directement implémentée dans le langage Arduino. On peut accéder à la communication série (au travers l'USB) grâce à l'objet *Serial*. Une autre librairie existe pour communiquer par liaison série via une des broches de la carte.

Il est parfois nécessaire de stocker des informations même après l'arrêt de la carte. Il est possible de stocker une petite quantité d'information sur la mémoire EEPROM (*Electricaly Erasable Programmable Read Only Memory*) intégrée. Une librairie est aussi fournie pour dialoguer avec cette mémoire. Il est possible de lire et d'écrire sur cette mémoire sans rien ajouter sur la carte. Cette mémoire est accessible via l'objet *EEPROM* et en ajoutant la librairie du même nom. A noter que la mémoire EEPROM a une durée de vie limitée (environ 100 000 cycles d'écritures).
Il faut donc veiller à ne pas écrire répétitivement les données. D'autres alternatives existent pour ajouter de plus grandes quantités de mémoires mortes à l'Arduino (carte SD notamment) mais cela demande l'ajout de composants externes.

Pour terminer, il existe aussi des librairies permettant de contrôler des composants externes. Parmi ces librairies, une des plus populaires est celle contrôlant des servomoteurs. Un servomoteur est un composant électronique composé d'un moteur et d'une carte d'asservissement. On peut le contrôler en position (c'est-à-dire choisir l'angle du moteur) grâce aux PWM. Une librairie Arduino est implémentée pour contrôler simplement ces moteurs : la librairie *Servo*. Il faut toutefois éviter d'alimenter les servomoteurs directement sur une des broches d'alimentation de la carte. En effet, un servomoteur consomme une quantité important de courant (suivant le couple exercé par le moteur). Il est donc fortement conseillé de relier les servomoteurs sur des alimentations externes (batteries, piles de 9V, etc.).