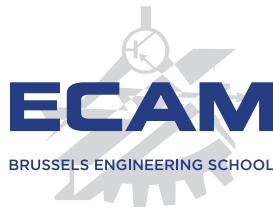


ECAM Institut Supérieur Industriel



and



Flanders Make

Conception et Implémentation d'un Circuit DC de Précharge et Machine à États pour Applications Industrielles

27 mai 2024

Rapport de stage

Xavier Allaud

Supervisé par :

Thomas Vandehove, Flanders Make

François Defrance, ECAM

Table des matières

1 Résumé	4
2 Introduction	5
2.1 Flanders Make	5
2.2 Objectifs et Méthodologie	6
2.3 Structure	6
3 Contexte	7
3.1 Retour du Courant Continu	7
3.2 Concept de Circuit de Pré-Charge	7
3.3 Résultats Attendus	9
3.4 Protections Électriques	10
3.5 Outils	11
4 Développement	13
4.1 <i>Hardware</i> du Prototype	13
4.1.1 Circuit de Pré-charge	13
4.1.2 Mesures	16
4.1.3 Contrôle et Acquisition	16
4.1.4 Alimentation	18
4.1.5 Sécurité	19
4.1.5.1 Bouton <i>E-STOP</i>	20
4.1.5.2 Bouton <i>RESET</i>	21
4.1.6 Disjoncteur de Protection	21
4.2 <i>Software</i> du Prototype	22
4.2.1 Programmation Orientée Objet (POO) en ST	22
4.2.2 Choix de la Méthode de Programmation	23
4.2.3 Machine à État - Diagramme de Classes	24
4.2.4 Machine à État - Diagramme d'États-Transitions	26
4.2.5 Machine à État - <i>MAIN</i>	27
4.2.6 Sécurité	27
4.2.7 Récupération des Données	28
4.3 Résultats	29
5 Conclusion et Perspectives	32
5.1 Projet	32
5.2 Expérience Personnelle	33

Annexes	36
.1 Schéma Électrique du Système	36
.2 HMI	36
.3 Photo du Système	37
.4 Diagramme de Classe UML	38
.5 Diagramme d'États	39
.6 Diagramme de Séquence	39
.7 Diagramme d'États - FB_FileWriter	40
.8 Circuits Simulation LTSpice	40

Chapitre 1

Résumé

Introduction

Ce rapport de stage, réalisé chez **Flanders Make**, détaille le développement d'une machine à état dans un environnement PLC en utilisant la programmation orientée objet (POO). L'objectif principal est de concevoir un prototype comprenant à la fois le matériel (hardware) et le logiciel (software) pour un circuit de pré-charge destiné à des applications industrielles.

Les objectifs de ce stage incluent :

1. **Développer une machine à état** en utilisant la POO dans un environnement PLC.
2. **Concevoir un circuit de test** de la source jusqu'à la charge.
3. **Développer une interface homme-machine (HMI)** pour les opérateurs.
4. **Démontrer la fonctionnalité et la robustesse** de la solution choisie.

Contexte et Développement

Le processus de conception détaillé dans ce rapport couvre la création d'un prototype (*hardware* et *software*), suivi de la comparaison des résultats obtenus avec les attentes théoriques.

Conclusions et Perspectives

Résultats obtenus :

1. **Machine à état** : La machine à état a été développée avec succès en utilisant le langage ST dans le logiciel TwinCAT 3. La méthode de POO utilisée pour sa conception est le *State Pattern*.
2. **Circuit de test** : Bien que je n'ai pas pu entièrement concevoir le circuit de test, j'ai pu au moins assurer le câblage et la finalisation de la construction du système. Le circuit de test est pleinement opérationnel et a été testé avec succès.
3. **Interface HMI** : L'interface HMI a été développée dans un fichier *Visualization* au sein du projet PLC. Cela a facilité sa création et son intégration avec le reste du projet.
4. **Fonctionnalité et robustesse** : La robustesse de la solution a été démontrée par la méthode utilisée pour sa conception. La fonctionnalité de la solution a été démontrée lors de la présentation du projet à mon superviseur de stage et à mon maître de stage.

Chapitre 2

Introduction

2.1 Flanders Make

Flanders Make, situé en Flandre, est un centre de recherche stratégique pour l'industrie manufacturière. Il encourage l'innovation ouverte et la collaboration sur des projets tels que la conception de produits, l'intégration de technologies innovantes, et l'optimisation des processus de production. Le centre dispose aussi d'infrastructures complètes pour tester et valider les produits.



(a) Site de Leuven (lieu du stage)



(b) Laboratoire utilisé pour le projet

Flanders Make vise à développer des technologies pour les véhicules, machines et usines de demain, renforçant ainsi la valeur ajoutée de l'industrie manufacturière. Il joue un rôle crucial dans la transformation numérique vers l'Industrie 4.0, augmentant la compétitivité globale et attirant des investissements étrangers tout en soutenant les entreprises locales.

Avec plus de 850 chercheurs, le centre dirige des projets dans trois domaines clés pour la transformation numérique : conception de bout en bout, production avancée, et produits en mouvement. Ces domaines comprennent la modélisation, la virtualisation et des méthodes de conception basées sur des modèles pour améliorer les processus de design. Flanders Make facilite également la digitalisation des environnements de production, promouvant la collaboration entre humains et robots pour améliorer la productivité.

Flanders Make est un pilier pour l'industrie manufacturière flamande, mettant l'accent sur la coopération internationale et la participation à des projets de recherche européens.

2.2 Objectifs et Méthodologie

Les objectifs de ce stage sont :

- **Développer une machine à état** dans un environnement PLC en utilisant la POO (Programmation Orientée Objet).
- **Concevoir un circuit de test** de la source jusqu'à la charge.
- **Développer une HMI (*Human Machine Interface*)** pour les opérateurs.
- **Démontrer la fonctionnalité et la robustesse** de la solution choisie

Ce rapport détaillera le processus de conception d'un prototype (*hardware* et *software*) visant à atteindre ces objectifs. Ensuite, nous comparerons les résultats obtenus avec les attentes théoriques.

2.3 Structure

Le rapport commence par une **Introduction**, présentant l'entreprise Flanders Make et les objectifs et méthodologie du stage. Ensuite, le **Contexte** décrit l'évolution du courant continu, le concept de circuit de pré-charge et les protections électriques nécessaires. La partie **Développement** détaille la conception du prototype, tant au niveau du hardware que du software, y compris la programmation orientée objet et les mesures de sécurité. Les **Résultats** obtenus ont été analysés pour vérifier la fonctionnalité et la robustesse du prototype. Enfin, la **Conclusion et Perspectives** récapitule les objectifs atteints, propose des améliorations futures et présente l'expérience personnelle du stagiaire. Les **Annexes** fournissent des schémas, photos et diagrammes supplémentaires pour illustrer et approfondir les aspects techniques abordés dans le rapport .

Chapitre 3

Contexte

3.1 Retour du Courant Continu

Au cours de ces dernières années, un changement significatif s'est opéré dans la production, la distribution et la consommation de l'énergie [11] :

- **Production** : Augmentation de la production énergétique renouvelable. À la fin de 2023, la capacité mondiale d'énergie renouvelable s'élevait à 3870GW , dont 1419GW (37%) issus de la **production solaire**, comparés à 500GW en 2018 [12].
- **Consommation** : On observe une augmentation de l'utilisation d'appareils consommant du courant continu, allant des LEDs utilisées dans l'éclairage domestique aux *smartphones* et ordinateurs, jusqu'aux moyens de transport électriques. La démocratisation des *superchargeurs* pour les véhicules électriques contribue également à cette hausse de la consommation de DC.
- **Distribution** :
 - Le développement et l'installation des *microgrids* [14] encouragent le retour de l'utilisation du courant continu. À l'échelle d'une maison, ces systèmes pourraient permettre de recharger une voiture électrique avec l'excédent de production des panneaux solaires. Ensuite, la batterie de la voiture pourrait servir de tampon lorsque la production solaire diminue, réduisant ainsi la consommation d'électricité du réseau principal.
 - L'extension rapide des réseaux de distribution électrique, favorisée par la libéralisation des marchés, a conduit à une augmentation des lignes HVDC au détriment des lignes HVAC. Les lignes **HVDC** offrent une **meilleure capacité de transfert de puissance** sur de longues distances et ont un **coût inférieur** par rapport aux lignes HVAC [7].

Ces tendances montrent clairement une évolution vers une plus grande utilisation du courant continu dans divers secteurs énergétiques.

3.2 Concept de Circuit de Pré-Charge

Un circuit de pré-charge permet de limiter les courants d'appels lors de la fermeture du circuit entre la source et la charge capacitive. Sans circuit de pré-charge, les courants d'appels pourraient endommager la source d'alimentation (particulièrement si c'est une batterie), souder les contacteurs ou faire fondre les câbles du circuit.

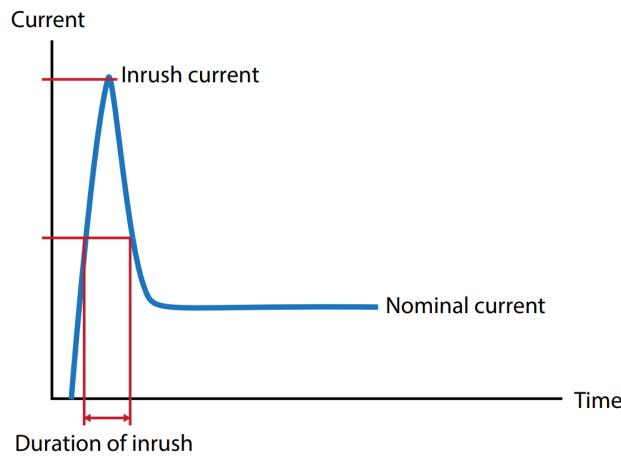
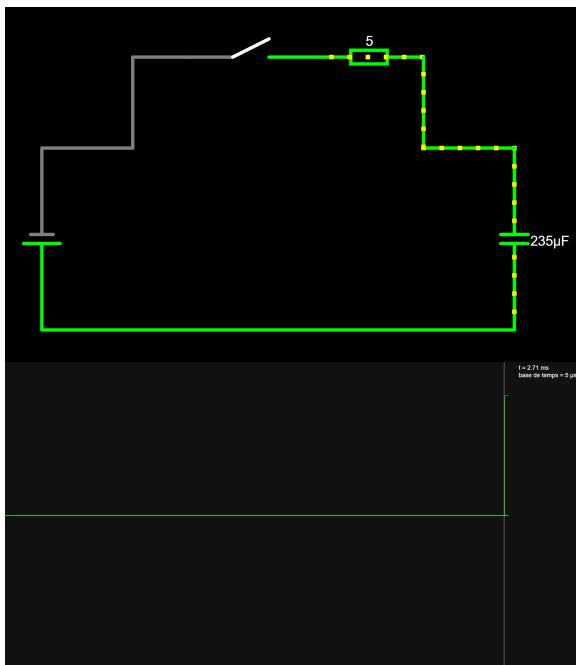


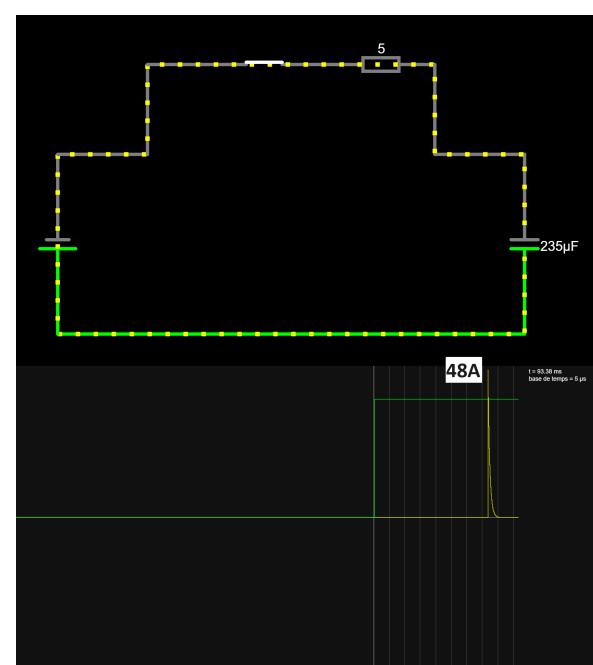
FIGURE 3.1 – Graphique de $i(t)$ pour une charge capacitive sans pré-charge [10].

Afin d'illustrer mon propos, voici ci-dessous (Figure 3.2) une simulation d'un circuit sans pré-charge. Il se compose d'une alimentation de 250V, d'un contacteur, d'une résistance de 5Ω en série qui permet de modéliser celle des câbles et d'un condensateur de $235\mu F$ agissant comme la charge du circuit.

Sur la Figure 3.2b qui correspond au circuit une fois le contacteur fermé et le condensateur chargé, un pic de courant apparaît sur l'oscilloscope au moment de la fermeture du contacteur. Ce pic de courant est d'environ **48A**.



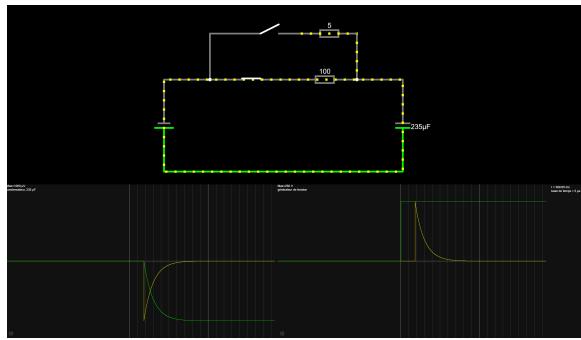
(a) Circuit ouvert.



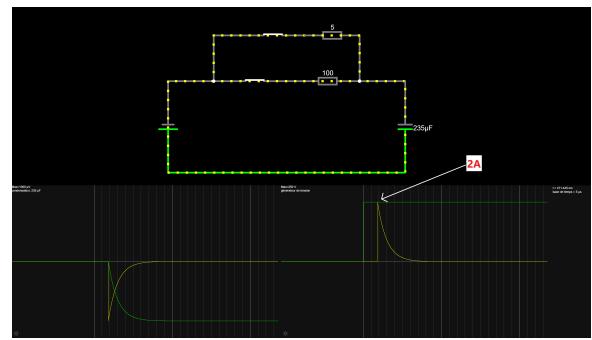
(b) Circuit fermé.

FIGURE 3.2 – Simulation Falstad

Pour limiter les courants d'appels, il suffit d'ajouter une résistance en série avec la capacité durant son temps de charge. Une fois la capacité chargée (Figure 3.3a), la résistance peut être contournée via un second interrupteur durant tout le reste du fonctionnement du circuit jusqu'à sa mise hors tension (Figure 3.3b). Grâce au circuit de pré-charge, le pic de courant n'est plus que de **2A**.



(a) Fermeture du contacteur de pré-charge.



(b) Fermeture du contacteur de "bypass".

FIGURE 3.3 – Simulation Falstad

Pour ce projet, dû à l'alimentation symétrique utilisée, deux systèmes de précharge seront placés de part et d'autre de la charge. Ceci sera détaillé dans la Section 4.1.1.

3.3 Résultats Attendus

Tout d'abord, les 3 équations fondamentales aux circuits RC sont :

$$u_c(t) = 1 - e^{\frac{-t}{RC}} \quad [V] \quad (3.1)$$

$$i_c(t) = \frac{U_s - u_c(t)}{R} \quad [A] \quad (3.2)$$

$$\tau = RC \quad [s] \quad (3.3)$$

Où u_c est la tension aux bornes du condensateur, i_c est l'expression du courant vu par le condensateur, τ est la constante de temps du circuit RC , U_s est la tension de l'alimentation et R est la résistance de pré-charge.

En partant de ces équations, on peut trouver l'équation de la puissance $p(t)$ et de l'énergie consommée $e(t)$:

$$p(t) = u(t) \cdot i(t) = \left(\frac{U_s}{R} e^{\frac{-t}{RC}} \right)^2 \cdot R \quad [W] \quad (3.4)$$

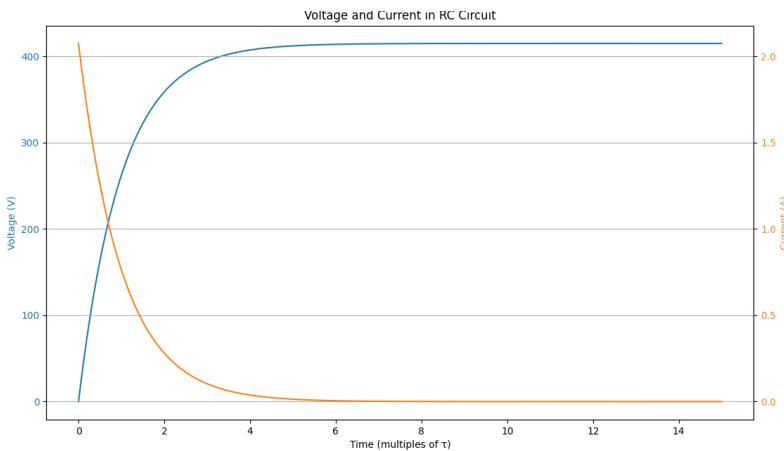
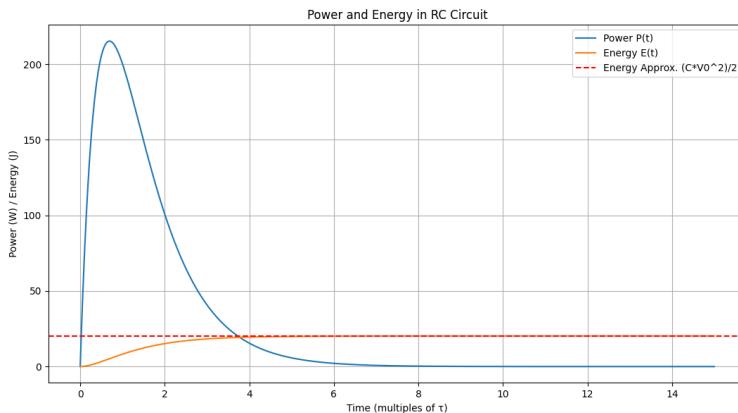
$$e(t) = \int p(t) dt = \frac{U_s^2 \cdot C}{R} e^{\frac{-2t}{RC}} \quad [J] \quad (3.5)$$

Pour $t > 3\tau$, cette dernière équation peut être approximée par (voir Figure 3.5) :

$$E = \frac{CU^2}{2} \quad [J] \quad (3.6)$$

En prenant les valeurs de résistances et de capacité utilisées dans le reste du rapport : $R = 200\Omega$ et $C = 235\mu F$, on obtient $\tau = 47ms$.

Ceci nous permet de tracer les courbes de tension et courant que nous serons sensées mesurer plus tard dans le rapport ainsi que les courbes de puissance et énergie.

FIGURE 3.4 – Graphique Matplotlib - $i(t)$ et $u(t)$.FIGURE 3.5 – Graphique Matplotlib - $p(t)$ et $e(t)$.

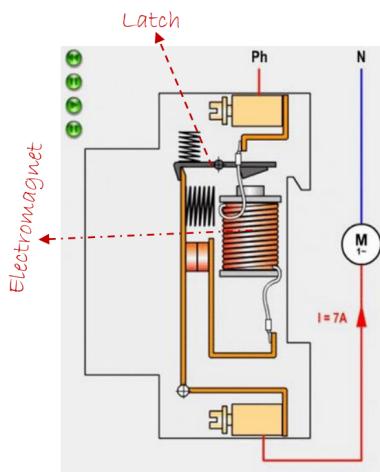
En prenant $\mathbf{U} = 415\text{V}$ comme tension d'alimentation, nous trouvons que la puissance moyenne que la résistance de pré-charge devra dissiper durant la charge du condensateur est :

$$E = \frac{235 \cdot 10^{-6} \cdot 415^2}{2} = 20,236 [\text{J}] \Rightarrow P = \frac{E}{5 \cdot \tau} = 86.11 [\text{W}] \quad (3.7)$$

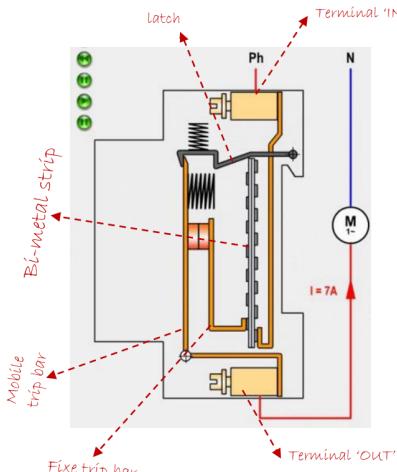
3.4 Protections Électriques

Afin de garantir la sécurité de l'installation électrique utilisée pour ce projet ainsi que la sécurité des opérateurs, deux types de protections sont nécessaires : des disjoncteurs et des interrupteurs différentiels (RCDs).

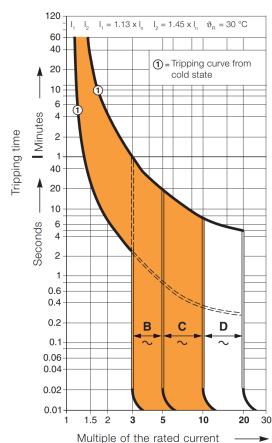
1. **Disjoncteurs** : Ce dispositif de protection inclut deux mécanismes différents. Tout d'abord, le mécanisme **thermique** qui déclenche le disjoncteur lorsque le courant dépasse le courant nominal pendant une certaine durée (allant de quelques secondes à plusieurs minutes). Le mécanisme **magnétique** quant à lui, déclenche le disjoncteur (en quelques ms) lorsqu'un courant de surcharge est détecté en utilisant la force résultant du champ magnétique proportionnel au courant $F = BiL$.



(a) Fonctionnement déclencheur thermique [6].



(b) Fonctionnement déclencheur magnétique [6].



(c) Caractéristiques de déclenchement - Classe C [1].

2. **Interrupteurs Différentiels** : Ils protègent les personnes contre les chocs électriques en détectant, via un transformateur de courant différentiel, un déséquilibre entre le courant entrant et sortant du circuit. En général ces courants varient entre 5mA et 30mA [13].

3.5 Outils

Les outils utilisés pour ce projet sont :

1. **TwinCAT 3 Engineering** : Logiciel de programmation basé sur Visual Studio permettant de programmer les automates Beckhoff. Le langage utilisé est le **Structured Text (ST)** supporté par la norme industrielle **IEC 61131-3** des automates programmables (PLC).

Malgré sa facilité d'apprentissage (comparé à d'autres méthodes de programmation de PLC), le ST présente certaines limites :

- Peu de flexibilité dans la déclaration des variables et de leur type.
- Peu de réusabilité du code (création de librairies, modules réutilisables...).
- Gestion de la complexité causée par la structure rigide du langage.
- Documentation limitée qui rend certaines résolution d'erreurs difficiles.

2. **IPC C6030-0060** : Un PC industriel conçu par la compagnie Beckhoff. Celui-ci permet notamment d'exécuter le code de plusieurs projets PLC simultanément. En effet, une partie des cores du processeur est dédiée à l'exécution de tâches en temps réel tandis que l'autre partie est utilisée par le système d'exploitation.

Dans le cadre de mon projet, cet IPC me permettra donc d'exécuter en temps réel le code de la machine à état qui contrôle le circuit de pré-charge.

3. **EtherCAT Terminals** : Les terminaux EtherCAT sont des modules d'entrées/sorties fabriqués par Beckhoff qui permettent de connecter des capteurs et actionneurs à l'IPC. Ils utilisent le protocole EtherCAT pour communiquer avec l'IPC.

Dans le cadre de ce projet, ils me permettront de mesurer les tensions et courants du circuit de pré-charge, d'actionner les contacteurs et de contrôler les boutons d'arrêt d'urgence et de réinitialisation.

4. **Draw.io et Lucidchart** : **Draw.io** m'a permis de dessiner le schéma électrique du circuit tandis que **Lucidchart** m'a permis de créer facilement de beaux diagrammes UML qui permettront de décrire le fonctionnement de la machine à état.

5. **Python** : Langage de programmation utilisé pour simuler les courbes de tension, courant, puissance et énergie du circuit de précharge. Il est également utilisé pour générer des graphiques à partir des données mesurées par l'IPC.

Chapitre 4

Développement

4.1 Hardware du Prototype

Cette section vise à décrire l'équipement utilisé pour le prototype et ainsi que les raisons pour lesquelles il a été choisi.

4.1.1 Circuit de Pré-charge

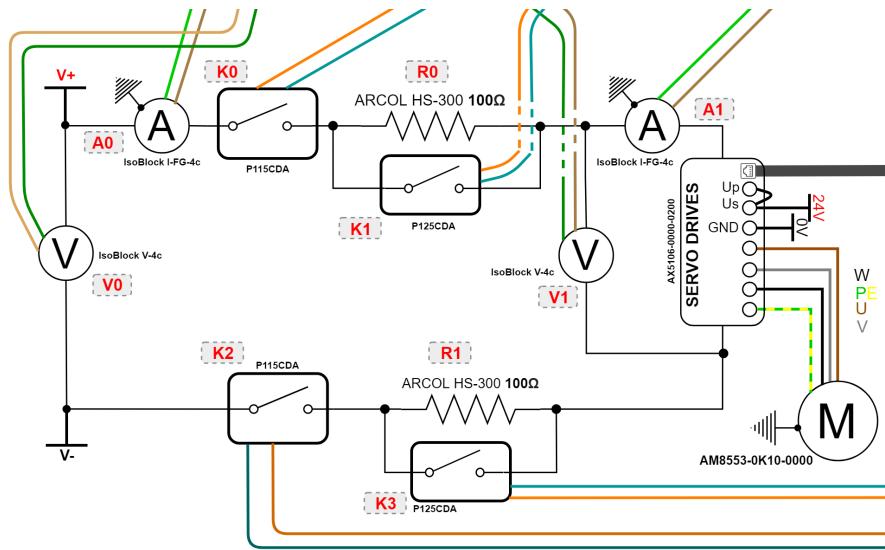


FIGURE 4.1 – Zoom sur le circuit de pré-charge (schéma complet en Annexe .1).

Le circuit de pré-charge est constitué de (voir Figure ci-dessus) :

1. Deux contacteurs **P115CDA** **K0** et **K2** qui sont les contacteurs principaux du circuit. Ceux-ci sont conçus pour supporter des courants (DC) et tensions (DC) allant jusqu'à, respectivement, 50A et 1500V. Leur type de contact est **SPST NO** (Single Pole Single Throw Normally Open). Les contacteurs sont commandés par le **terminal 7 EL2004** (cf. Section 4.1.3).

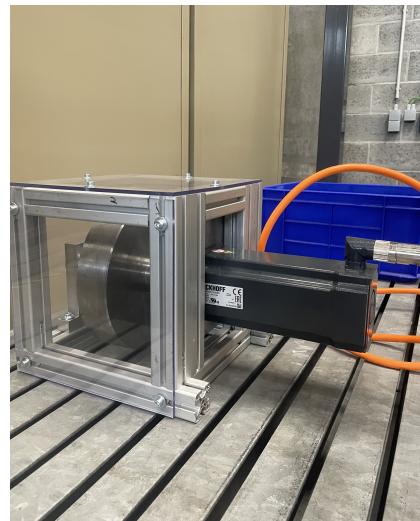


FIGURE 4.2 – Contacteurs **K0, K2, K1, K3** (dans l’ordre, sur la gauche de la photo) et résistances **R0** et **R1** (droite de la photo).

2. Deux contacteurs **P125CDA K1** et **K3** qui sont les contacteurs de *bypass*, ils permettent de contourner la résistance de pré-charge à la fin de la charge du condensateur. Le courant (DC) et la tension (DC) qu’ils peuvent supporter sont, respectivement, 30A et 1200V. Leur type de contact est **SPST NO**. Ils sont commandés par le même terminal que les contacteurs **K0** et **K2**.
3. Deux résistances **ARCOL HS-300 R0** et **R1** de **100Ω**. Elles peuvent dissiper jusqu’à **300W** de puissance, ce qui est inférieur à **86,11W** qui est la puissance moyenne durant la charge du condensateur (cf. calcul Section 3.3).
4. Une charge constituée d’un *driver* **AX5106** et d’un moteur **AM8553-0K10**. Le *driver* est utilisé dans une configuration DC et est contrôlé directement par l’IPC via protocole EtherCAT et en utilisant un câble de type **RJ45**. Sa **capacité** interne de **filtrage** est de **235μF**. Le moteur est un moteur synchrone à aimants permanents dont la tension nominale est **400V**. Celui-ci entraîne une roue d’inertie qui simule la présence d’une charge.



(a) *Driver* du moteur.



(b) Moteur entraînant la roue d’inertie.

FIGURE 4.3 – Photos du *driver* et du moteur.

5. La différence de tension entre les points **V+** et **V-** (cf. Figure 4.1) sera assurée par une alimentation AC-DC symétrique (voir Section 4.1.4).

Comme mentionné dans la Section 3.2, deux systèmes de pré-charge sont utilisés au lieu d’un seul, comme c’est souvent le cas. Cela est nécessaire en raison de la symétrie de l’alimentation AC-DC (4.1.4)

employée. L'intégration d'un second système de pré-charge sur le côté négatif de la charge présente deux avantages principaux :

1. Il empêche la connexion directe du côté négatif de la charge à une tension négative élevée, ce qui pourrait endommager le matériel.
2. Il offre une sécurité supplémentaire en cas de défaillance du premier système.

La figure suivante illustre une simulation du circuit avec les deux systèmes de pré-charge lors de la mise en pré-charge de la charge (cf. circuit utilisé pour la simulation en Annexe .8, Figure 8) :

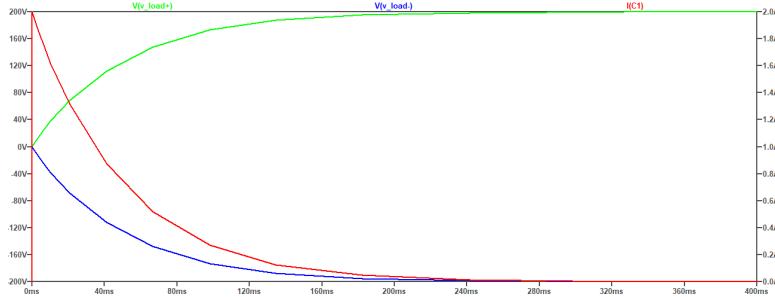


FIGURE 4.4 – Simulation de la tension et du courant lors de la mise en pré-charge.

Sur le graphique ci-dessus, on peut observer que la tension à la borne négative de la charge (en bleu) démarre à 0V et atteint sa valeur nominale de $-200V$ de manière exponentielle en raison de la dynamique RC du circuit. La tension à la borne positive (en vert) suit une évolution similaire, mais avec une tension nominale positive de $200V$. Le courant (en rouge) présente la dynamique attendue : il atteint son pic de $4A$ à $t = 0$ millisecondes puis diminue de manière exponentielle jusqu'à $0A$.

La simulation suivante montre le circuit avec un seul système de pré-charge (cf. circuit utilisé pour la simulation en Annexe .8, Figure 9) :

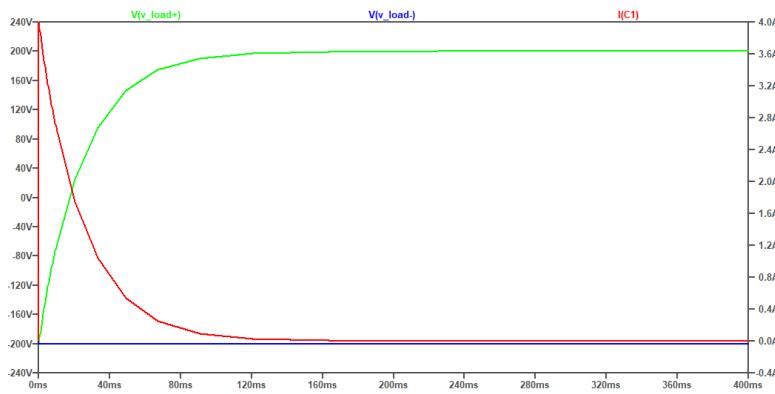


FIGURE 4.5 – Simulation de la tension et du courant lors de la mise en pré-charge avec un seul système.

Dans la figure ci-dessus, on constate que la tension à la borne négative de la charge (en bleu) reste constante à $-200V$. La tension à la borne positive (en vert) démarre à $-200V$ et atteint $200V$ de manière exponentielle. Étant donné que la constante de temps τ reste inchangée, la pente de la tension positive augmente, ce qui fait augmenter la valeur du pic de courant (en rouge), qui passe ainsi à $4A$.

L'utilisation de deux systèmes de pré-charge pour une alimentation AC-DC symétrique est essentielle pour protéger les composants contre des tensions négatives élevées et pour ajouter une sécurité en cas de défaillance. Les simulations confirment que cette configuration assure une mise en charge équilibrée et contrôlée, respectant les dynamiques RC attendues, et garantissant ainsi la fiabilité et la sécurité du système.

4.1.2 Mesures

Afin de mesurer les tensions et courants de la charge et de l'alimentation, le circuit est équipé de deux capteurs de tension et deux capteurs de courant (cf. Figure 4.1).

Les capteurs de tension IsoBlock V-4c V0 et V1 mesurent respectivement la tension d'alimentation et la tension aux bornes de la charge. Ceux-ci sont conçus pour isoler et abaisser la différence de tension entre les points qu'ils mesurent afin qu'un système d'acquisition puisse y être connecté. Le signal de sortie se situe dans une plage de $\pm 10V$. Pour ce modèle, la relation entre la tension d'entrée et la tension de sortie est de **1 : 100** avec une erreur de linéarité de $\pm 0,04\%$ et un décalage en sortie de $\pm 500\mu V$.

Tandis que les capteurs de courant IsoBlock I-FG-4c A0 et A1 mesurent respectivement le courant d'alimentation et le courant de la charge. Ces capteurs sont conçus pour mesurer des courants allant jusqu'à $25A$. Le signal de sortie se trouve dans une plage de $\pm 5V$ et la relation entre le courant d'entrée et le courant de sortie est de **1 : 5** avec une erreur de linéarité de maximum $280ppm/A$, une erreur d'Hysteresis de maximum $\pm 10mV$ et un décalage en maximum $\pm 5mV$.

Les sorties des capteurs de tension sont connectées au **terminal 4 EL3702** tandis que celles des capteurs de courant sont connectées au **terminal 2 EL3702**. Les capteurs sont alimentés par l'alimentation $24V$ AC-DC.

4.1.3 Contrôle et Acquisition

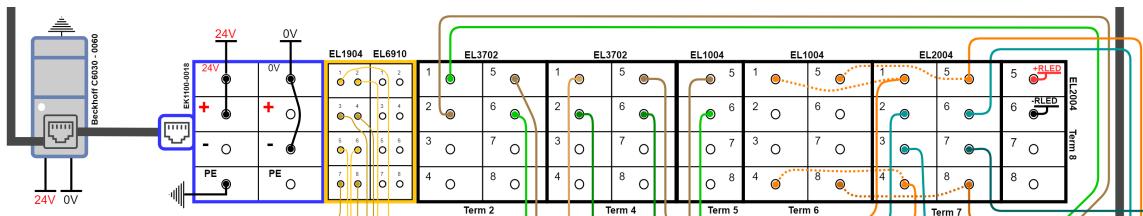


FIGURE 4.6 – Zoom sur le contrôle et l'acquisition (schéma complet en Annexe 1).



FIGURE 4.7 – IPC (gauche) et module EK1100 (droite).

Le module **EK1100** permet de connecter les terminaux *EtherCAT* à l'IPC (via à un câble RJ-45). Les données des différents terminaux sont alors disponibles dans TwinCAT lors de la création d'un projet PLC :

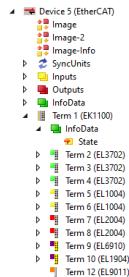


FIGURE 4.8 – Données des terminaux dans TwinCAT.

Grâce à ce système, il est possible de lier les entrées et sorties des capteurs ou actuateurs à des variables dans le programme PLC.

La capture d'écran ci-dessous montre le lien entre la variable `n_V_sensors[0]` et le *channel* 1 du terminal 4 EL3702. Ce *channel* est connecté au capteur de tension **V0** qui mesure la tension d'alimentation.

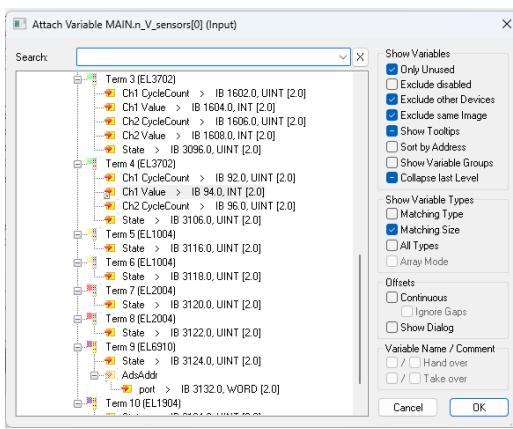


FIGURE 4.9 – Configuration du lien entre la variable et le terminal.

Les types de terminaux utilisés pour ce projet sont les suivants :

- **EL3702** : Terminal avec deux canaux d'**entrées analogiques**. La plage de tension acceptée pour ses entrées est de $\pm 10\text{V}$. Les valeurs de ces dernières sont ensuite digitalisées sur **16 bits** et transmises au contrôleur. Ce terminal peut également être utilisé pour faire de l'*oversampling*, c'est-à-dire de la mesure de tension à une fréquence plus élevée que la fréquence d'échantillonnage du contrôleur. Cette fonctionnalité ne sera pas utilisée pour ce projet.
Ce terminal est utilisé pour **mesurer les tensions** (terminal 4) et **courants** (terminal 2) de la charge et de l'alimentation.
- **EL1004** : Terminal avec 4 canaux d'**entrées digitales**. Le **bit 1** correspond à une tension de **24V** et le **bit 0** à une tension de **0V**. Ce terminal est utilisé pour le **feedback des contacteurs** (terminal 6) et pour l'état du **bouton RESET** (terminal 5).
- **EL2004** : Terminal avec 4 canaux de **sorties digitales**. Le bit 1 correspond à une tension de **24V** et le bit 0 à une tension de **0V**. Ce terminal est utilisé pour **commander les contacteurs du circuit de pré-charge** (terminal 7). Il est aussi utilisé pour **contrôler la LED** du bouton RESET (terminal 8).
- **EL1904** : Terminal spécifique pour la **sécurité**. Il dispose de 4 canaux d'**entrées digitales**. Le bit 1 correspond à une tension de **24V** et le bit 0 à une tension de **0V**, tout comme le terminal EL1004. Il se différencie de ce dernier par le fait qu'il est spécifiquement conçu pour répondre aux normes de sécurité. Il dispose également de mécanismes de redondance et de surveillance pour détecter les défauts tels que les courts-circuits, les coupures de câbles et les dysfonctionnements des capteurs. Dans le cadre de ce projet, couplé au terminal EL6910, ce terminal est utilisé pour détecter un appui

sur le **bouton E-STOP** (terminal 10).

- **EL6910** : Terminal PLC (*Programmable Logic Controller*) dédié à la sécurité. Il est utilisé pour **contrôler les terminaux de sécurité** tels que l'**EL1904**. Il permet d'exécuter des *function blocks* assemblés par l'utilisateur via une interface graphique dans TwinCAT. Dans le cadre de ce projet, il effectue des opérations sur les entrées du terminal EL1904 et pour **déclencher des actions de sécurité** telles que l'arrêt ou la réinitialisation du système (cf. Section 4.1.5).

4.1.4 Alimentation

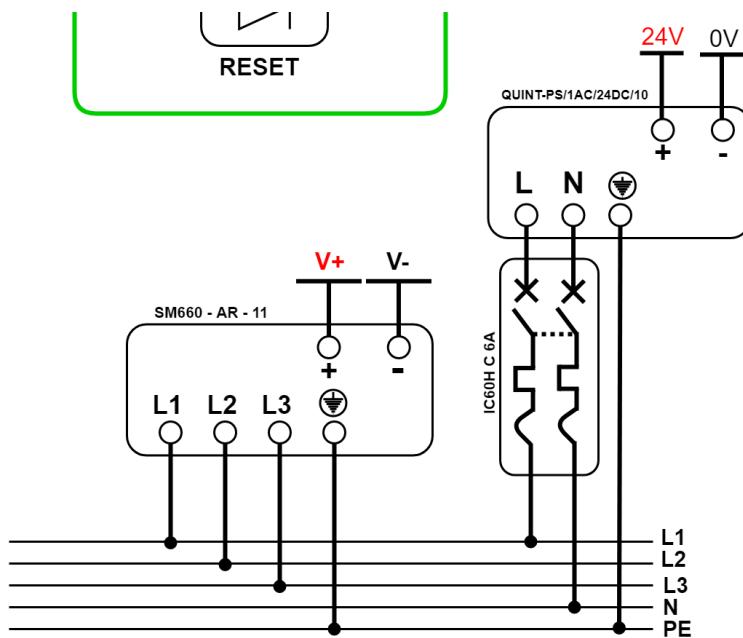


FIGURE 4.10 – Zoom sur les alimentations du système (schéma complet en Annexe .1).

Le système est alimenté par deux alimentations :

- Une alimentation AC-DC haute tension **SM 660-AR-11**. Elle servira à fournir la tension nécessaire au moteur. Cette alimentation est symétrique par rapport à 0V. Pour ce projet, elle est configurée pour être connectée aux 3 phases et à la terre. La puissance d'entrée maximum est de $3.3kW$ pour un courant de maximum $5.6A$ (pour une tension de $400V$ entre chaque phase). La tension de sortie est une plage de **0 – 660V DC** pour un courant de **0 – 11A**. Cette configuration a été choisie parce qu'un câble triphasé était déjà disponible dans le laboratoire et que la puissance d'entrée correspondait à celle supportée par la prise et celle de sortie correspondait aux besoins du moteur.



FIGURE 4.11 – Alimentation AC-DC haute tension.

- Une alimentation AC-DC basse tension **QUINT-PS/1AC/24DC/10**. Elle servira à alimenter le système

et les périphériques du *driver*. La plage de tension nominale en entrée est de 100VAC à 240VAC pour un courant de 1.3A. Le courant d'appel est limité à 15A ce qui est inférieur à la limite imposée par le disjoncteur de la prise (16A). Pour ce projet, elle sera connectée à une phase et au neutre afin d'obtenir une tension d'entrée de 230VAC. Sa tension de sortie est de **24V DC ±1%** pour un courant nominal de **10A**. L'alimentation ne garantit qu'une différence de potentiel 24V entre ses deux bornes mais ne garantit pas que la négative soit à 0V. C'est pourquoi il est nécessaire de forcer la borne négative à 0V en la reliant à la terre (cf. Figure 4.12)

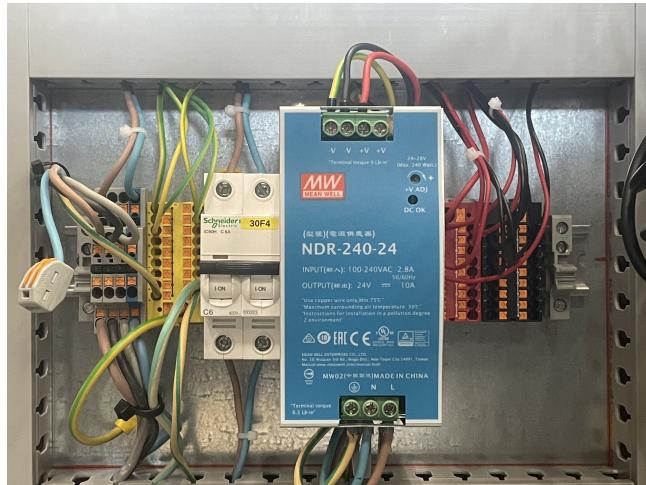


FIGURE 4.12 – Disjoncteur (gauche) et alimentation AC-DC basse tension (droite).

C'est une alimentation à découpage (cf. Figure 4.13). Ce qui permet de conserver la même tension de sortie en diminuant le nombre de tours n du transformateur et en augmentant la fréquence f du courant alternatif grâce à un *chopper* (3^e module). Cela permet de réduire la taille du transformateur et d'augmenter le rendement de l'alimentation.

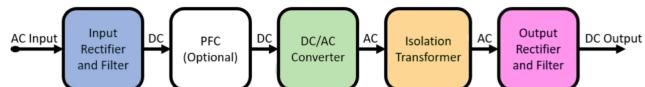


FIGURE 4.13 – Fonctionnement d'une alimentation AC-DC à découpage [9].

4.1.5 Sécurité

La catégorie de configuration utilisée pour ce bouton est la 3^e (cf. Figure 4.14 extraite du guide d'application *TwinSAFE* [3]).

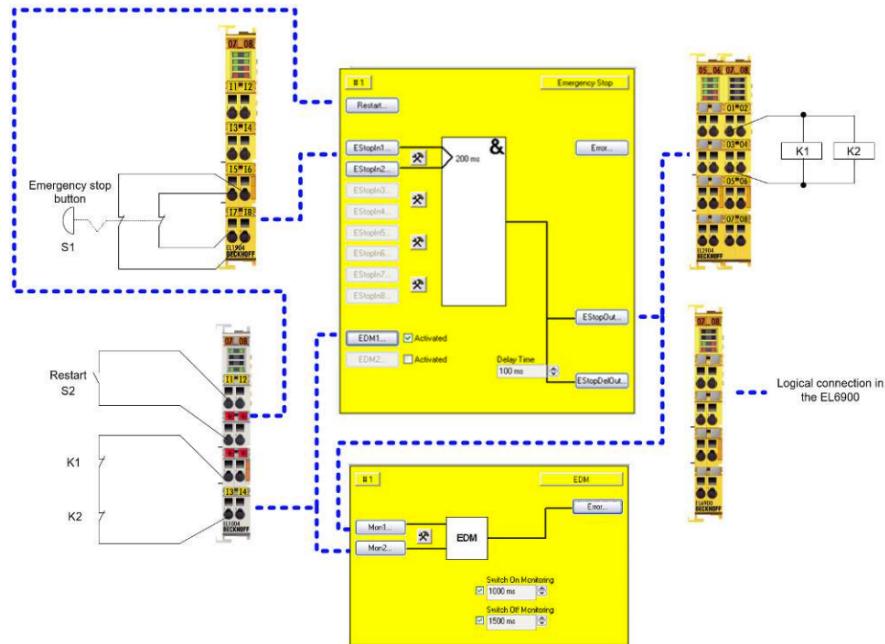


FIGURE 4.14 – Schéma de configuration de sécurité pour le bouton E-STOP [3].

Cette configuration a été légèrement modifiée pour l'adapter à l'installation ; les contacteurs K1 et K2 ont été supprimés puisque ceux utilisés pour arrêter le système seront les contacteurs du circuit de pré-charge (voir Section 3.2 et 4.2.6). La connection du bouton *RESET* a également été modifiée : celui-ci est connecté au **terminal 5 EL1004** (cf. Annexes .1). Le terminal **EL2904** n'a également pas été utilisé pour ce projet. La sortie **EstopOut** est à la place reliée à une variable d'entrée de la machine à état (cf. Section 4.2.6). Cette configuration a été choisie pour sa simplicité et sa fiabilité.

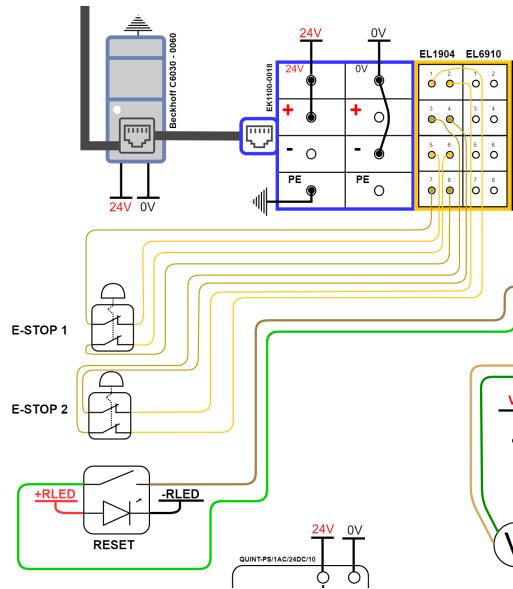


FIGURE 4.15 – Zoom sur la connexion des boutons E-STOP (Section 4.1.5.1) et RESET (Section 4.1.5.2) (voir Annexe .1).

4.1.5.1 Bouton E-STOP

Deux boutons E-STOP ont été installés pour garantir la sécurité :

- Un bouton E-STOP principal, intégré dans le panneau avant de l'installation.

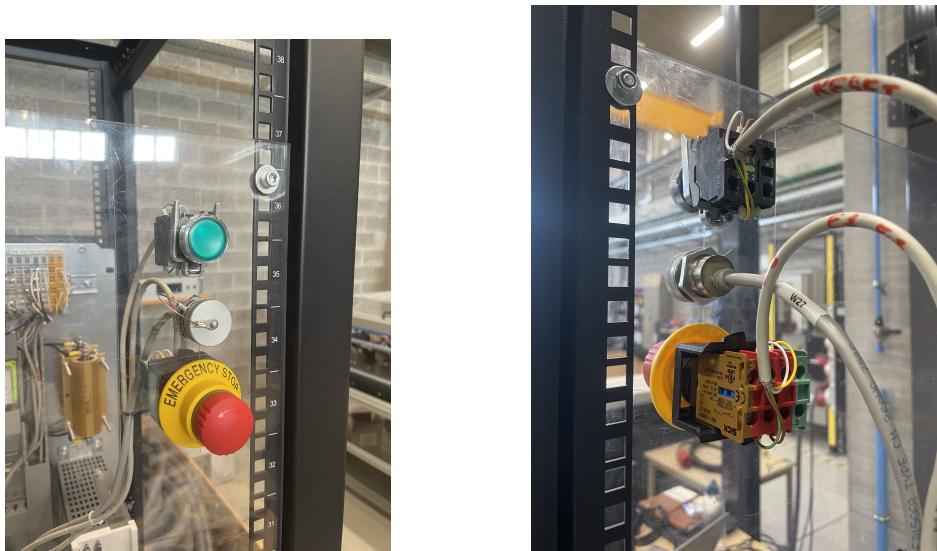


FIGURE 4.16 – Bouton *E-STOP* sur le panneau avant de l'installation.

- Un bouton *E-STOP* secondaire, situé sur le bureau de l'opérateur.



FIGURE 4.17 – Bouton *E-STOP* sur le bureau de l'opérateur.

Le bouton est équipé de deux contacteurs (Figure 4.15) pour garantir une **redondance** du signal, ce qui est crucial en cas de défaillance de l'un d'eux (par exemple, contacts soudés ou bouton endommagé). Ces contacteurs sont de type **NC** (normalement fermé), ce qui assure l'arrêt du système dans des situations telles que la coupure des câbles du bouton *E-STOP*. Cette configuration répond au principe de sécurité **Fail Safe**. Conformément au principe **Two Actions**, le bouton doit être poussé puis tourné sur sa partie rouge pour être désactivé, assurant ainsi que l'action de *STOP* reste active jusqu'à ce que le risque soit totalement éliminé.

Le terminal **EL1904** est conçu spécialement pour la sécurité de l'installation, isolant les fonctions de sécurité dans le logiciel de la machine à état.

4.1.5.2 Bouton *RESET*

Quant au bouton *RESET*, celui-ci permet de redémarrer le processus après avoir actionné le bouton *E-STOP*. Le bouton *RESET* n'étant pas fondamental à la sécurité de l'installation, il est connecté (voir Figure 4.15) aux mêmes terminaux (**EL3702**) que le reste des capteurs du circuit. L'actionnement de la LED est gérée dans le projet de *safety* (cf. Section 4.2.6) et les signaux de sortie au bouton via le **terminal 8 EL2004**.

4.1.6 Disjoncteur de Protection

Le système dispose de deux disjoncteurs pour protéger l'installation des surintensités :

1. Un disjoncteur “miniature” à 3 pôles + neutre DL61C32 est inclus dans la prise qui permet d'alimenter le système en triphasé (+neutre) avec terre. Ce disjoncteur est configuré pour un courant nominal de **32A** et a une courbe de fonctionnement de type C (cf. Section 3.4). Il est conçu pour couper un courant maximum de **10kA**.
 2. Un disjoncteur intégré dans le circuit IC60H C6A (cf. Figure 4.12, photo, et Figure 4.10, schéma électrique). Il est configuré pour un courant nominal de **6A** et a une courbe de fonctionnement de type C. Il est conçu pour couper un courant maximum de **10kA**. Ce disjoncteur protège l'alimentation basse tension du système.

4.2 Software du Prototype

Cette section vise à décrire le logiciel utilisé pour le prototype et les raisons pour lesquelles il a été choisi. Pour ce projet, le logiciel *TwinCAT 3* a été utilisé ainsi que le langage de programmation ST (*Structured Text*) (décris dans la Section [3.5](#)).

Le type de programmation utilisée est la *Programmation Orientée Objet* (POO).

Tout le code est disponible sur le répertoire [GitHub](#) du projet.

4.2.1 Programmation Orientée Objet (POO) en ST

Dans le language ST, la **POO** est différente des langages plus utilisés tels que le **C**, **C#**... Un projet ST est composé de *References*, DUTs (*Data Unit Types*), GVLs (*Global Variable Lists*), POUs (*Program Organization Units*), de VISus (*Visualizations*), de tâches et d'instances. Dans cette section du rapport, nous nous concentrerons sur les POUs.

Les POU sont des unités de codes qui peuvent être de plusieurs types [4] :

- FUNCTION
- FUNCTION_BLOCK
- PROGRAM

Dans ce projet, un seul POU `PROGRAM` sera utilisé, il s'agit du programme principal `MAIN` (cf. Section 4.2.5) qui permettra d'initialiser les machines à état et d'associer les variables contenant leur état courant à la HMI (cf. Annex .2) .

Quant aux FB, ce sont des classes découpées en deux parties :

```

1  FUNCTION FB_FileWriter
2    VAR_INPUT
3      sFilePath           : T_MaxString;
4      sFileName          : T_MaxString;
5      sFileExt            : STRING; // FileId of the target, leave empty for local
6    END_VAR
7
8    VAR CONSTANT
9      nArrBufferSize     : INT := 1000;
10   END_VAR
11
12   VAR
13     fbFileOpen          : FB_FileOpen;
14     fbFilePut          : FB_FilePut;
15     fbFileClose         : FB_FileClose;
16     fbFileHandle        : WORD;
17
18
19     arrBuffer          : ARRAY[0..nArrBufferSize] OF T_MaxString;
20     efileWriteState    : (IDLE, OPEN_FILE, WRITE_TO_FILE, CLOSE_FILE, ERROR);
21   END_VAR
22
23
24 CASE eFileWriteState OF
25   IDLE:
26     // Make sure all the file writing FBs are ready for use (trigger on execute)
27     Init();
28
29
30   // We have pending data to be written
31   IF arrBuffer[0] <> "" THEN // Makes sure that the first element of the array is not an empty string
32     efileWriteState := OPEN_FILE;
33   END_IF
34
35
36   OPEN_FILE:
37     // Opens a file for writing at the end of the file (append).
38     // If the file does not exist, a new file is created.
39
40     fbFileOpen(
41       bAppend           := TRUE,
42       sFileId           := sFileId,
43       sPathName         := CONST(sFilePath, sFileName),
44       nMode             := FORCED_NOCREATEPEND);
45
46 END_CASE

```

FIGURE 4.18 – Structure d'un *Function Block* en ST.

La première partie constitue la définition de la classe ainsi que des variables qui lui sont propres. Dans

le cadre du projet, nous n'utiliserons que des variables d'entrée `VAR_IN` qui sont l'équivalent de variables publiques en C# et des variables constantes `VAR_CONSTANT`, des variables de sortie `VAR_OUTPUT` et des variables internes `VAR` qui dont l'utilisation et l'accès sont restreints à la classe. La **seconde partie** constitue le code à exécuter lorsque le FB est appelé.

À l'intérieur d'un FB, il est possible de déclarer des méthodes (ci-dessous, en bleues et avec un "M") et des propriétés (ci-dessous, en rouge et avec un "P"). Ces dernières ont les mêmes fonctions que les propriétés en C#.

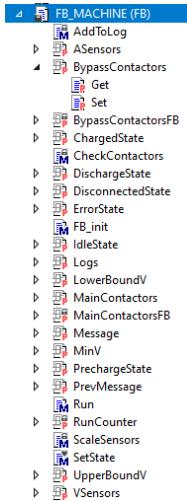


FIGURE 4.19 – Structure d'un FB en ST.

La définition d'une méthode est, comme pour un FB, découpée en deux parties. Il est possible de leur assigner une visibilité.

```

METHOD PRIVATE CheckContactors
VAR
    nIndex : INT := 0;
    bMainContactorsCheck : BOOL := TRUE; // true means that there is no problem
    bBypassContactorsCheck : BOOL := TRUE;
    bConclusion : BOOL := TRUE; // true means that there is no problem
    fbDischargeState : FB_DischargeState(THIS);
END_VAR

FOR nIndex := 0 TO 1 DO
    bMainContactorsCheck := THIS^.bMainContactors[nIndex] = THIS^.bMainContactorsFB[nIndex]; // checks that the feedback signal corresponds to the
    bBypassContactorsCheck := THIS^.bBypassContactors[nIndex] = THIS^.bBypassContactorsFB[nIndex];
    bConclusion := FALSE;
    IF NOT(bMainContactorsCheck) THEN
        THIS^.Message := CONCAT('!CheckContactors) The ', CONCAT(INT_TO_STRING(nIndex), ' Main contactor state does not match the signal sent'));
        bConclusion := FALSE;
    END_IF
    IF NOT(bBypassContactorsCheck) THEN
        THIS^.Message := CONCAT('!CheckContactors) The ', CONCAT(INT_TO_STRING(nIndex), ' Bypass contactor state does not match the signal sent'));
        bConclusion := FALSE;
    END_IF
END_FOR

IF NOT(bConclusion) THEN
    THIS^.Message := CONCAT(THIS^.Message, '[CheckContactors]@0 THE SYSTEM IS NOW BEING DISCHARGED');
    THIS^.SetState(fbDischargeState);
    bConclusion := TRUE;
END_IF

```

FIGURE 4.20 – Structure d'une méthode en ST.

4.2.2 Choix de la Méthode de Programmation

Selon [Stefan Henneken](#), ingénieur en génie logiciel, il y a globalement 3 différentes approches pour réaliser une machine à état [8] :

1. En utilisant une **instruction CASE**, qui permet de définir un état pour chaque cas. Les cas sont déterminés par une variable d'énumération `ENUM`. L'avantage réside dans la **simplicité du code**, car tout peut être écrit dans le programme principal. Cependant, cette méthode est **peu flexible**, difficile à maintenir et devient rapidement **complexe** avec l'augmentation du nombre d'états.
2. En utilisant les **méthodes d'une classe** pour effectuer les transitions entre états. Cette méthode est **plus flexible** que la précédente, mais elle est **plus complexe** à mettre en place, car une transition peut

être commune à plusieurs états. Le code reste contenu dans une seule classe, ce qui le rend **difficile à étendre**.

3. Le **State Pattern**. Cette méthode utilise deux concepts importants de la POO : le principe de **Responsabilité unique** (chaque classe a une seule responsabilité) et le principe d'héritage (les méthodes générales sont partagées entre plusieurs classes). Dans les deux approches précédentes, la classe principale de la machine à état, appelée ici `FB_MACHINE`, devait être modifiée pour être étendue, ce qui va à l'encontre du principe d'ouverture/fermeture. Dans cette approche, chaque état est un FB contenant le comportement complet de l'état correspondant. Il devient donc facile d'ajouter ou de supprimer des états sans modifier le FB principal de la machine à état. C'est donc cette méthode qui a été choisie pour le projet.

4.2.3 Machine à État - Diagramme de Classes

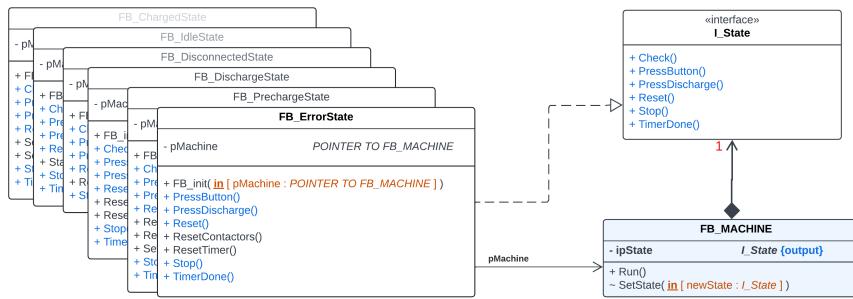


FIGURE 4.21 – Diagramme de classes simplifié de la machine à état (diagramme détaillé en Annexe 4).

Sur le diagramme ci-dessus, on peut voir que `FB_MACHINE` compose l'interface `I_State`. Dans le code, cela se traduit par la création d'une variable de type `I_State` dans `FB_MACHINE` qui sera utilisée pour appeler les méthodes de l'état courant :

```
1 FUNCTION_BLOCK PUBLIC FB_MACHINE
2 //...
3 VAR_OUTPUT
4     ipState : I_State := fbDisconnectedState; // L'état par défaut est l'état déconnecté
5 END_VAR
6 //...
```

Listing 4.1 – Aggrégation de l’interface I_State dans FB_MACHINE.

Au lieu que `FB_MACHINE` doive mettre en œuvre tous les comportements par lui-même, il (l'objet instancié) stocke une référence à l'un des objets d'états (`ipState` dans le cas présent) qui représente son état actuel et délègue toutes les tâches liées à l'état de cet objet. Cette délégation est effectuée en appelant les méthodes de l'objet d'état courant à partir de l'objet `ipState` :

```
1 IF (charge_TON.Q) THEN // Lorsque le timer de pre-charge expire
2     ipState.TimerDone();
3 END_IF
4
5 IF (rtrigButtonDisch.Q) THEN // Lorsque le bouton de decharge est presse dans l'interface
6     ipState.PressDischarge();
7 END_IF
8
9 IF (rtrigButtonAck.Q) THEN // Lorsque le bouton de validation est presse dans l'interface
10    ipState.PressButton();
11 END_IF
```

Listing 4.2 – Exemple de délégation d'appel de méthode issu de la méthode Run de FB_MACHINE.

Chaque objet d'état implémente les méthodes selon le comportement attendu pour cet état. Par exemple, l'état `fbDisconnectedState` implémente la méthode `PressButton()` comme suit :

```

1 // Recuperation de la tension d'entree
2 fVSensors := THIS^.pMachine^.VSensors;
3 fInputVoltage := fVSensors[0];
4
5 IF (fInputVoltage >= THIS^.pMachine^.MinV) THEN
6   THIS^.pMachine^.Message := '[DISCONNECTED] Acknowledge button has been pressed. $0A SYSTEM
    ENTERS IDLE STATE.'; // permet d'afficher un message dans l'interface
7   THIS^.pMachine^.SetState(THIS^.pMachine^.IdleState); // changement d'état
8 END_IF

```

Listing 4.3 – Exemple de méthode `PressButton` de l'état `fbDisconnectedState`.

Tandis que l'état `fbIdleState` implémente la méthode `PressButton()` de la manière suivante :

```

1 // Recuperation de la tension d'entree
2 fVSensors := THIS^.pMachine^.VSensors;
3 fInputVoltage := fVSensors[0];
4
5 IF (fInputVoltage >= THIS^.pMachine^.LowerBoundV AND fInputVoltage <= THIS^.pMachine^.
  UpperBoundV) THEN
6   THIS^.pMachine^.Message := '[IDLE] Acknowledge button has been pressed. $0A SYSTEM ENTERS
    PRECHARGE STATE.'; // permet d'afficher un message dans l'interface
7   THIS^.pMachine^.SetState(THIS^.pMachine^.PrechargeState); // changement d'état
8   THIS^.StartTimer(); // demarrage du timer de pre-charge
9 END_IF

```

Listing 4.4 – Exemple de méthode `PressButton` de l'état `fbIdleState`.

Cette méthode permet donc de déléguer les tâches liées à l'état à l'objet d'état lui-même et d'avoir un comportement différent pour une même action de déclenchement (ici, un appui sur un bouton de l'interface 2). Elle permet également de rendre le code plus lisible et plus facile à maintenir.

Comme illustré dans les exemples de code ci-dessus (cf. Listing 4.4 et Listing 4.3), pour changer d'état, la méthode `SetState()` est appelée via le pointeur `pMachine`, qui est une référence à l'objet `FB_MACHINE`. Ce pointeur permet d'appeler la méthode `SetState()` de l'objet `FB_MACHINE`, qui se charge de changer l'état courant de la machine en l'état passé en paramètre :

```

1 METHOD INTERNAL SetState
2 VAR_INPUT
3   newState : I_State;
4 END_VAR
5
6 THIS^.ipState := newState; // changement d'état

```

Listing 4.5 – Méthode `SetState` de la classe `FB_MACHINE`.

Chaque objet d'état doit donc être instancié dans la classe principale avec un pointeur vers celle-ci :

```

1 FUNCTION_BLOCK PUBLIC FB_MACHINE
2 /**
3 VAR
4   fbDisconnectedState : FB_DisconnectedState(THIS);
5   fbIdleState : FB_IdleState(THIS);
6   fbPrechargeState : FB_PreloadState(THIS);
7   fbChargedState : FB_ChargedState(THIS);
8   fbDischargeState : FB_DischargeState(THIS);
9   fbErrorState : FB_ErrorState(THIS);
10 /**
11 END_VAR
12 /**

```

Listing 4.6 – Instanciation des états dans la classe principale `FB_MACHINE`.

Ces deux derniers extraits de code permettent d'expliquer deux choses sur le diagramme de classes détaillé (cf. Annexe .4) :

- La relation de composition entre les objets d'états et la classe principale `FB_MACHINE` est dûe à leur instantiation dans cette dernière.
- La relation d'association entre les objets d'états et la classe principale `FB_MACHINE` est dûe à la référence à cette dernière dans les objets d'états via le pointer `pMachine`.

4.2.4 Machine à État - Diagramme d'États-Transitions

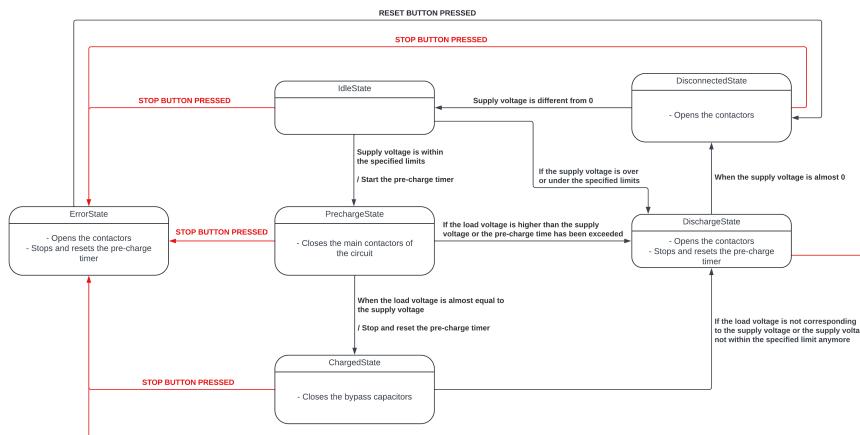


FIGURE 4.22 – Diagramme d'états-transitions de la machine à état (diagramme détaillé en Annexe .5).

Ci-dessus, le diagramme d'états-transitions de la machine à état est présenté. Il est composé de 6 états différents :

- `DisconnectedState` : état initial de la machine, tous les contacteurs sont ouverts et l'alimentation est éteinte.
- `IdleState` : état dans lequel la machine se trouve lorsqu'elle est allumée mais que la tension d'alimentation n'est pas dans la plage de tension attendue.
- `PrechargeState` : état dans lequel la machine se trouve lorsqu'elle est allumée, que la tension d'alimentation est comprises dans les limites spécifiées et que les contacteurs principaux (pas ceux de *bypass*) sont fermés. La machine reste dans cet état pendant un certain temps pour permettre la pré-charge du circuit.
- `ChargedState` : état dans lequel la machine se trouve lorsqu'elle est allumée, que la tension d'entrée est dans la plage de tension attendue et que la pré-charge du circuit est terminée. Les contacteurs sont donc tous fermés et la machine est prête à être utilisée.
- `DischargeState` : état dans lequel la machine se trouve lorsqu'elle est allumée, que la tension d'alimentation est différente de *OV* et que tous les contacteurs sont ouverts. Cet état est transition entre n'importe lequel des états précédents et l'état *Disconnected*.
- `ErrorState` : état dans lequel la machine se trouve lorsqu'une erreur est détectée dans le système. Tous les contacteurs sont ouverts et le *timer* de pré-charge est arrêté et remis à 0. Le seul moyen de sortir de cette état est d'appuyer sur le bouton *RESET*.

Dans ce diagramme simplifié, il n'est pas mentionné que dans tous les états, sauf `ErrorState`, `DisconnectedState` et `DischargeState`, la machine peut passer à l'état `DischargeState` lorsque le bouton de décharge est pressé sur l'interface (cf. Annexe .2). De plus, il n'est pas indiqué que chaque changement d'état (sauf pour `ErrorState`) doit être confirmé par l'utilisateur en appuyant sur le bouton de validation sur l'interface. Pour comprendre

en détail le passage d'un état à un autre, il est nécessaire de se référer au diagramme d'états-transitions détaillé en Annexe .5 et au diagramme de séquence en Annexe .6.

4.2.5 Machine à État - MAIN

Le fichier `MAIN` est le fichier principal du projet. Il permet d'instancier la machine à état, de récupérer les données des capteurs et de les passer en paramètre à l'objet `FB_MACHINE`:

```

1 PROGRAM MAIN
2 VAR
3   (* Inputs *)
4   n_V_sensors      AT %I*      : ARRAY[0..1] OF INT; // defining 2 elements array of INT type
5   as input to the PLC task
6   n_A_sensors       AT %I*      : ARRAY[0..1] OF INT;
7
8   b_reset           AT %I*      : BOOL; // variable for the reset button
9   b_estop          AT %I*      : BOOL; // variable for the E-STOP button
10
11  b_main_contactors_fb    AT %I*      : ARRAY[0..1] OF BOOL; // defining 4 elements array of
12    BOOL type as input of feedback for the contactors to the PLC task
13  b_bypass_contactors_fb AT %I*      : ARRAY[0..1] OF BOOL;
14
15  (* Outputs *)
16  b_main_contactors     AT %Q*      : ARRAY[0..1] OF BOOL; // defining 4 elements array of BOOL
17    type as output to the PLC task
18  b_bypass_contactors  AT %Q*      : ARRAY[0..1] OF BOOL;
19
20  (* FB_MACHINE *)
21  b_user_button         : BOOL := FALSE; // variable for the user button on the
22    visualization file
23  b_discharge           : BOOL := FALSE;
24  n_lower_boundV        : INT := 380;
25  n_upper_boundV        : INT := 460;
26  n_minV                : INT := 5;
27
28  t_max_time            : TIME := T#250MS; //5x the time constant for a RC circuit with R=
29    100 ohms and C=235 uF
30
31 fbMachine              : FB_MACHINE(nLowerBoundV := n_lower_boundV,
32                                     nUpperBoundV := n_upper_boundV,
33                                     nMinV := n_minV,
34                                     tMaxTime := t_max_time); //instanciation of the state machine
35
36 fbMachine.Run(
37   bAck:=b_user_button,
38   bDischarge := b_discharge,
39   nVSensors := n_V_Sensors,
40   nASensors := n_A_Sensors,
41   bBypassContactorsFB := b_bypass_contactors_fb,
42   bMainContactorsFB := b_main_contactors_fb,
43   bReset := b_reset,
44   bESTOP := b_estop,
45   bMainContactors => b_main_contactors,
46   bBypassContactors => b_bypass_contactors,
47 );

```

Listing 4.7 – Exemple d'utilisation de la machine à état dans le fichier `MAIN`.

4.2.6 Sécurité

Le projet contient un projet PLC ainsi qu'un projet *SAFETY* dédiés à la gestion du bouton *E-STOP* et du bouton *RESET*. Les détails relatifs à ce projet ne seront pas abordés dans ce rapport mais peuvent être

trouvés dans le code source du projet [2].

La gestion de la sécurité est effectuée dans la fonction `Run` classe `FB_MACHINE`:

```

1 //E-Stop check
2 IF NOT(bESTOP) THEN
3   ipState.Stop(); // sets state to ErrorState
4   ipState.Check();
5   bMainContactors := THIS^.MainContactors;
6   bBypassContactors := THIS^.BypassContactors;
7   RETURN;
8 END_IF
9 //Reset check
10 IF ipState = ErrorState AND bReset THEN // reset has been pressed
11   ipState.Reset();
12 END_IF
13 // Functionning contactors check
14 IF RunCounter MOD 15 = 0 THEN // Checks contactors every 15 run loop because it takes some
   time for the feedback to adjust
15   CheckContactors(); // Ensures that the contactors signals are the same as the contactors
   states
16 END_IF

```

Listing 4.8 – Gestion de la sécurité dans la méthode `Run` de la classe `FB_MACHINE`.

Cette façon de faire n'est pas optimale, car elle ne permet pas de gérer la sécurité de manière indépendante par rapport à la machine à état. Une amélioration possible serait l'implémentation la partie relative à la sécurité dans un projet *SAFETY*. Ceci permettrait de garantir que la sécurité est toujours active, même si la machine à état est dans un état d'erreur.

4.2.7 Récupération des Données

Pour récupérer les données, une autre machine à état, beaucoup plus simple, a été implémentée. Cette machine permet de collecter les valeurs des capteurs de courant et de tension et de les stocker dans un tableau *buffer*. Les données de ce tableau sont ensuite progressivement enregistrées dans un fichier texte. Le diagramme d'états-transitions de cette machine à état est présenté en Annexe .7.

Une fois ces fichiers .txt sauvés, ils peuvent être analysés et visualisés à l'aide d'un court programme python. Ce programme trace les données des fichiers .txt respectifs dans des graphiques et, dans le cas des fichiers de tension, calcule la constante de temps et en déduit la valeur du condensateur de la charge (ici, du *driver* cf. Section 4.1.1).

4.3 Résultats

Voici les résultats obtenus pour le prototype. Sur le graphique ci-dessus, une seule pré-charge a été réalisée.

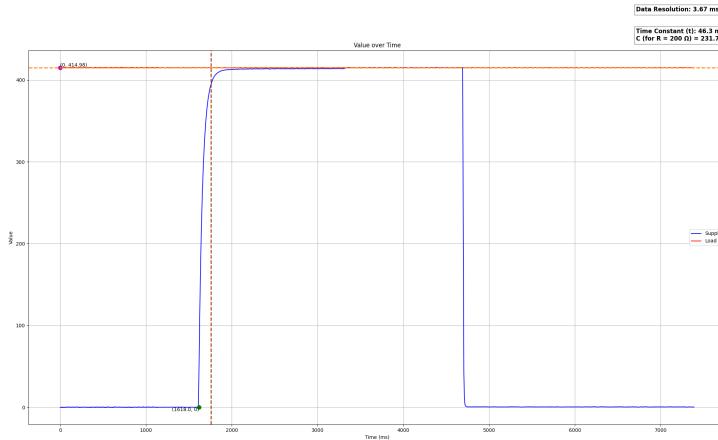


FIGURE 4.23 – Graphique de la tension en fonction du temps.

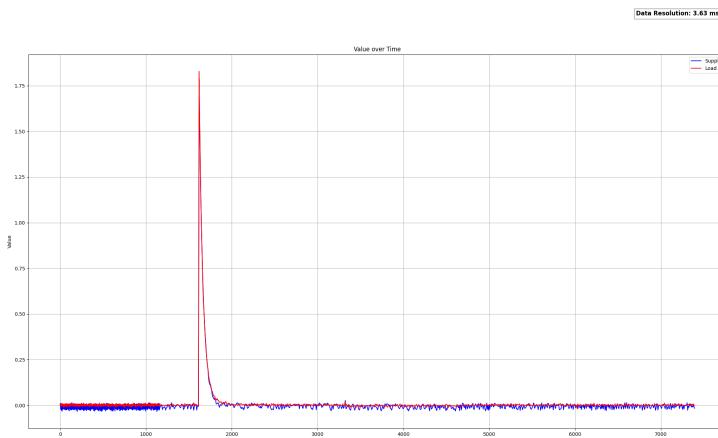


FIGURE 4.24 – Graphique du courant en fonction du temps.

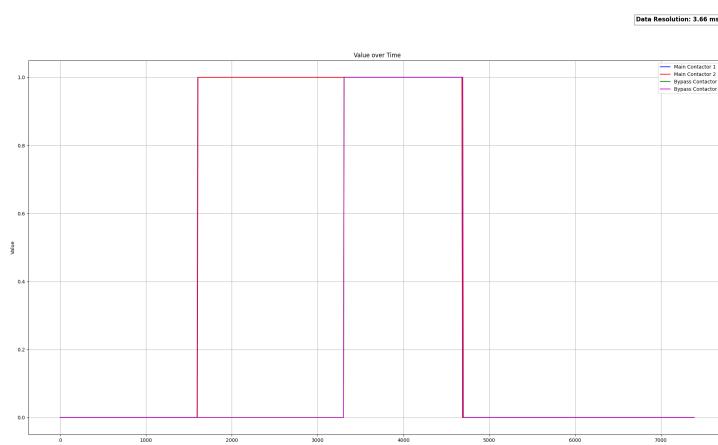
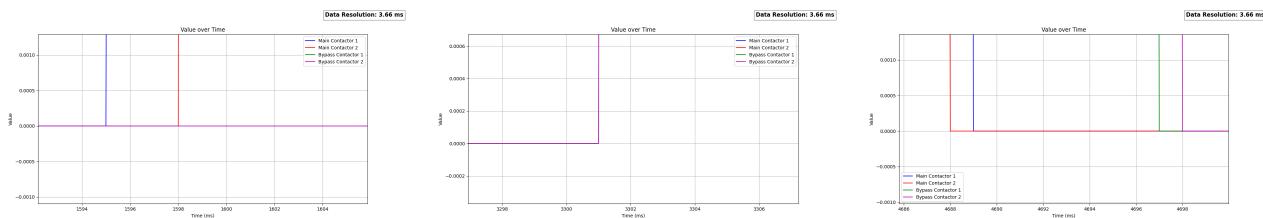


FIGURE 4.25 – Graphique de la tension en fonction du temps pour le calcul de la constante de temps.

Sur le premier graphique (cf. Figure 4.23), la tension suit la courbe attendue (cf. Section 3.3 et 4.1.1). Le script Python permet de calculer la **constante de temps du circuit** en mesurant le temps écoulé entre le moment où la tension de la charge commence à varier et celui où elle atteint **63%** de sa valeur finale. La constante de temps est ensuite déterminée en divisant cette différence de temps par **3**. En connaissant la valeur de la résistance du circuit, il est facile d'estimer celle du condensateur, qui est ici de **231,7 μ F**, ce qui est **proche** de la **valeur réelle** de $235\mu F$ (cf. Section ??). Une fois la charge déconnectée, la tension chute rapidement à 0V grâce à une résistance de décharge (cf. *datasheet driver* [5]). Cette résistance étant environ 4 fois plus petite que celle du circuit RC principal, cela explique la chute rapide de la tension.

Sur le second graphique (cf. Figure 4.24), la dynamique du courant correspond à celle anticipée (cf. Section 3.3). Un **pic de courant** d'environ **1.8A** est observé au moment de la pré-charge. Ceci correspond aux **2A** prévus lors de la simulation pour le courant de pré-charge (cf. Figure 4.4).

Le dernier graphique (cf. Figure 4.25) permet de vérifier le fonctionnement des contacteurs. Voici, ci-dessous, un zoom sur les parties intéressantes du graphique en question :



(a) Zoom sur la fermeture des contacteurs principaux (b) Zoom sur la fermeture des contacteurs de bypass. (c) Zoom sur l'ouverture de tous les contacteurs.

On remarque que la **fermeture** des contacteurs principaux (cf. Figure 4.26b) ainsi que l'**ouverture** de tous les contacteurs (cf. Figure ??) ne sont **pas simultanées**, avec un léger décalage de **1 milliseconde** entre les différents contacteurs. Cependant, il n'y a pas de décalage visible lors de la fermeture des contacteurs de *bypass* (cf. Figure ??).

En réalité, il y a un décalage, mais il n'est pas visible sur le graphique. Il est probable que ces décalages soient dus à la **fréquence d'échantillonnage** des capteurs, qui est d'environ **3 millisecondes**. En effet, chaque capteur est échantillonné à tour de rôle et sa valeur est stockée dans un fichier texte. Les valeurs des deux contacteurs principaux et des deux contacteurs de *bypass* sont donc stockées dans des fichiers différents via différentes instances de `FB_FileWriter`, ce qui induit un certain décalage :

```

1 PROGRAM MAIN
2 VAR
3 /**
4   fileBlocks           : ARRAY[0..8] OF FB_FileWriter := [fbCurrent1, fbCurrent2,
5   fbVoltage1, fbVoltage2];
6   fileBlocksContactors : ARRAY[0..8] OF FB_FileWriter := [fbBypass1, fbBypass2,
7   fbMain1, fbMain2];
8 /**
9 END_VAR

```

Listing 4.9 – Déclaration des différents machine à état de sauvegarde des données `FB_FileWriter` dans le fichier `MAIN`.

La dernière observation est donc la **RÉSOLUTION DES DONNÉES**, dont la moyenne est calculée et affichée sur chaque graphique. Elle permet d'avoir une idée sur la précision des mesures. Elle doit être suffisamment inférieure à la constante de temps des phénomènes étudiés afin de pouvoir les tracer correctement.

Lors de premiers tests, elle était justement trop élevée ce qui faussait les résultats. Pour la réduire, il a fallu diminuer à 1 milliseconde les “tics de cycle” de la tâche associée au projet de pré-charge :

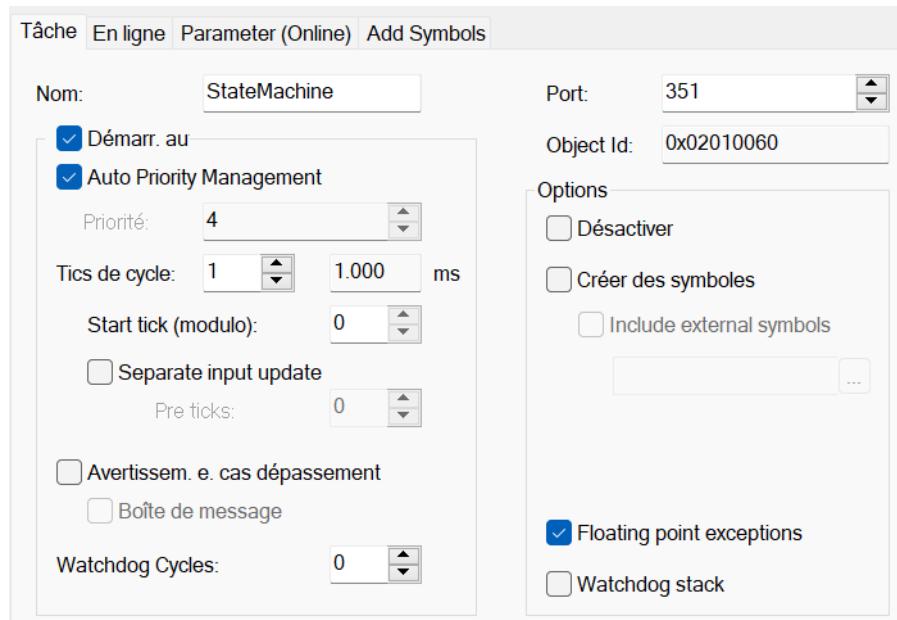


FIGURE 4.27 – Réglage des “tics de cycle” de la tâche associée au projet de pré-charge.

Pour conclure, les résultats obtenus sont satisfaisants et correspondent aux attentes. D'autres tests auraient pu être effectués pour vérifier la robustesse du système, tels que l'évaluation de la puissance et de l'énergie, ou la vérification de la constante de temps sur plusieurs précharges successives, mais cela n'a pas été possible dans le temps imparti pour ce projet.

Chapitre 5

Conclusion et Perspectives

Ce dernier chapitre a pour but de conclure le rapport selon deux perspectives différentes :

1. Le projet en lui-même, en résumant les objectifs atteints et les résultats obtenus ainsi qu'en donnant des pistes d'amélioration pour le futur.
2. L'expérience personnelle du stagiaire, en décrivant les compétences acquises et les leçons apprises durant le stage.

5.1 Projet

Pour débuter cette conclusion, il est important de rappeler les objectifs du projet :

1. *Développer une machine à état dans un environnement PLC en utilisant la POO (Programmation Orientée Objet).*

Cet objectif a été atteint en développant une machine à état via la méthode de *State Pattern* décrite dans la Section 4.2.2 dans le logiciel TwinCAT 3 (cf. 3.5).

2. *Concevoir un circuit de test de la source jusqu'à la charge.*

Étant donné que le stage était initialement prévu pour un étudiant de master et pour une durée de minimum 10 semaines, il n'a pas été possible de réaliser cette partie du projet. C'est donc mon maître de stage qui a préalablement conçu le circuit de test. J'ai néanmoins pu terminer le câblage et la construction du système.

3. *Développer une HMI (Human Machine Interface) pour les opérateurs.*

L'interface HMI a été développée dans un fichier *Visualization* (cf. 2) au sein du projet PLC. Cela a facilité sa création et son intégration avec le reste du projet. Ce compromis, suggéré par mon maître de stage dès le début, a été nécessaire en raison du manque de temps pour développer une HMI.

4. *Démontrer la fonctionnalité et la robustesse de la solution choisie.*

Selon moi, la robustesse de la solution a été démontrée par la méthode utilisée pour sa conception : la machine à état est facilement modifiable et extensible grâce à la POO. La fonctionnalité de la solution a été démontrée lors de la présentation du projet à mon superviseur de stage et à mon maître de stage.

Concernant les améliorations futur du projet :

1. La **délégation** de la partie **sécurité** (cf. Section 4.2.6) à un projet *SAFETY* est nécessaire pour garantir le fonctionnement de celle-ci même en cas de panne du projet principal.
2. L'ajout du **contrôle de l'alimentation** AC-DC moyenne tension (cf. Section 4.1.4) dans la machine à état et dans le projet *SAFETY*. Ceci permettrait d'améliorer la sécurité du système et de faciliter l'utilisation de celui-ci en le rendant plus autonome.

3. L'ajout d'un **disjoncteur DC** dans le circuit de pré-charge pour protéger le système en cas de surcharge ou de court-circuit.
4. L'ajout d'une **classe intermédiaire** entre le fichier `MAIN` et la machine à état afin de **fluidifier la gestion de plusieurs circuits de pré-charge**. Cette classe intermédiaire pourrait prendre en paramètres le nombre de circuits, les capteurs de tension et courant, les contacteurs et les autres paramètres sous forme de variables `STRUCT`. Ensuite, elle instancierait les différentes machines à états pour chaque circuit. Elle serait particulièrement utile pour les circuits complexes où plusieurs circuits de pré-charge sont utilisés simultanément avec des tensions d'alimentation différentes.
5. Le **remplacement** du **contacteur** principal et de *bypass* par un seul contacteur de type **SPTT** (Single Pole Triple Throw) qui a donc 3 positions : circuit ouvert, résistance de pré-charge et *bypass*.
6. L'**amélioration** de la **sauvegarde** des **données** (cf. Section [4.2.7](#)) afin d'accélérer le processus et de le rendre plus évolutif.

5.2 Expérience Personnelle

Ce stage m'a permis de développer mes compétences en programmation PLC et en POO. J'ai appris à programmer des IPC en utilisant le langage ST. J'ai également découvert les méthodes de travail et de conception ainsi que les différents systèmes utilisés dans la recherche industrielle. J'ai appris à collaborer avec des techniciens, par exemple lors du montage du système sur une ancienne armoire de serveur. J'ai aussi appris à évoluer dans une entreprise, à discuter avec des collègues et des supérieurs, et à m'adapter à un nouvel environnement de travail.

Les discussions avec mes collègues ont toujours été très enrichissantes et m'ont permis de mieux comprendre les différents aspects du métier d'ingénieur. Elles m'ont également aidé à saisir les enjeux de la recherche industrielle.

Ce stage a confirmé mon envie de travailler dans le milieu de la recherche et du développement. J'ai pu découvrir les différentes facettes de ce métier et j'ai hâte de continuer à apprendre et à évoluer dans ce domaine.

Bibliographie

- [1] ABB. *Comparison of tripping characteristics for miniature circuit-breakers*. URL : https://library.eabb.com/public/114371fcc8e0456096db42d614bead67/2CDC400002D0201_view.pdf.
- [2] Xavier ALLAUD. *GitHub Repository*. URL : <https://github.com/sstav0/Pre-Charge-State-Machine.git>.
- [3] BECKHOFF. *Application Guide TwinSAFE*. URL : <https://download.beckhoff.com/download/Document/automation/twinsafe/applicationguidetwinsafeen.pdf>.
- [4] BECKHOFF. *Beckhoff Information System*. URL : <https://infosys.beckhoff.com/>.
- [5] BECKHOFF. *System manual AX5000*. URL : https://download.beckhoff.com/download/Document/motion/ax5000_system_manual_hw2_en.pdf.
- [6] J. CHAMASSI. *Residential Electrical Installations*.
- [7] Ahmad EZZEDDINE. *An In-depth Comparison of HVDC and HVAC*. URL : <https://eepower.com/technical-articles/an-in-depth-comparison-of-hvdc-and-hvac/>.
- [8] Stefan HENNEKEN. *IEC 61131-3 : The State Pattern*. URL : <https://stefanhenneken.net/2018/11/17/iec-61131-3-the-state-pattern/>.
- [9] MPS. *Understanding AC/DC Power Supplies*. URL : <https://www.monolithicpower.com/en/ac-dc-power-supply-basics>.
- [10] Brian MURANI et Andrew SCHNEER. *How to design a precharge circuit for hybrid and electric vehicle applications*. URL : <https://www.sensata.com/sites/default/files/a/sensata-how-to-design-precharge-circuits-evs-whitepaper.pdf>.
- [11] Jean G. SAINT MARTIN L. Del Maestro A. *Current wars 2.0 : A new era of direct electric current is set to transform your world*. URL : <https://www.strategyand.pwc.com/gx/en/insights/2018/current-wars-2-0/current-wars-2-0.pdf>.
- [12] *Statistiques de capacité renouvelable 2024*. URL : https://www.irena.org.translate.google.com/Publications/2024/Mar/Renewable-capacity-statistics-2024?_x_tr_sl=en&_x_tr_tl=fr&_x_tr_hl=fr&_x_tr_pto=sc.
- [13] WIKIPEDIA. *Residual-current device*. URL : https://en.wikipedia.org/wiki/Residual-current_device.
- [14] Elisa WOOD. *What is a microgrid?* URL : <https://www.microgridknowledge.com/about-microgrids/article/11429017/what-is-a-microgrid>.

Annexes

.1 Schéma Électrique du Système

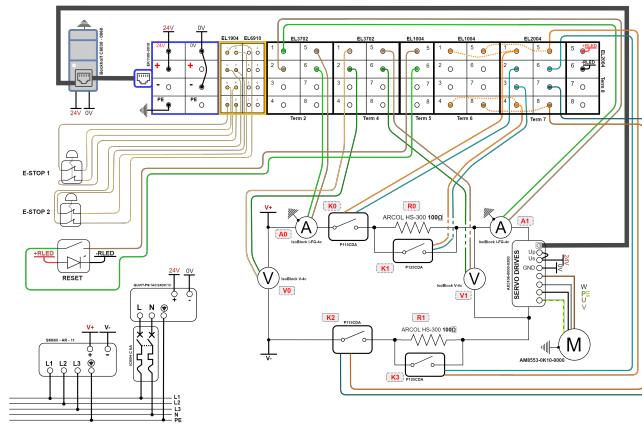


FIGURE 1 – Schéma électrique du système.

Le fichier source de ce schéma est disponible sur le dépôt GitHub du projet [2].

.2 HMI

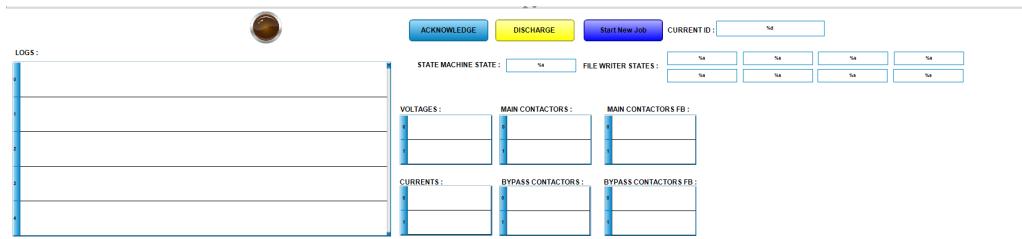


FIGURE 2 – Interface Homme-Machine (HMI) du projet.

.3 Photo du Système

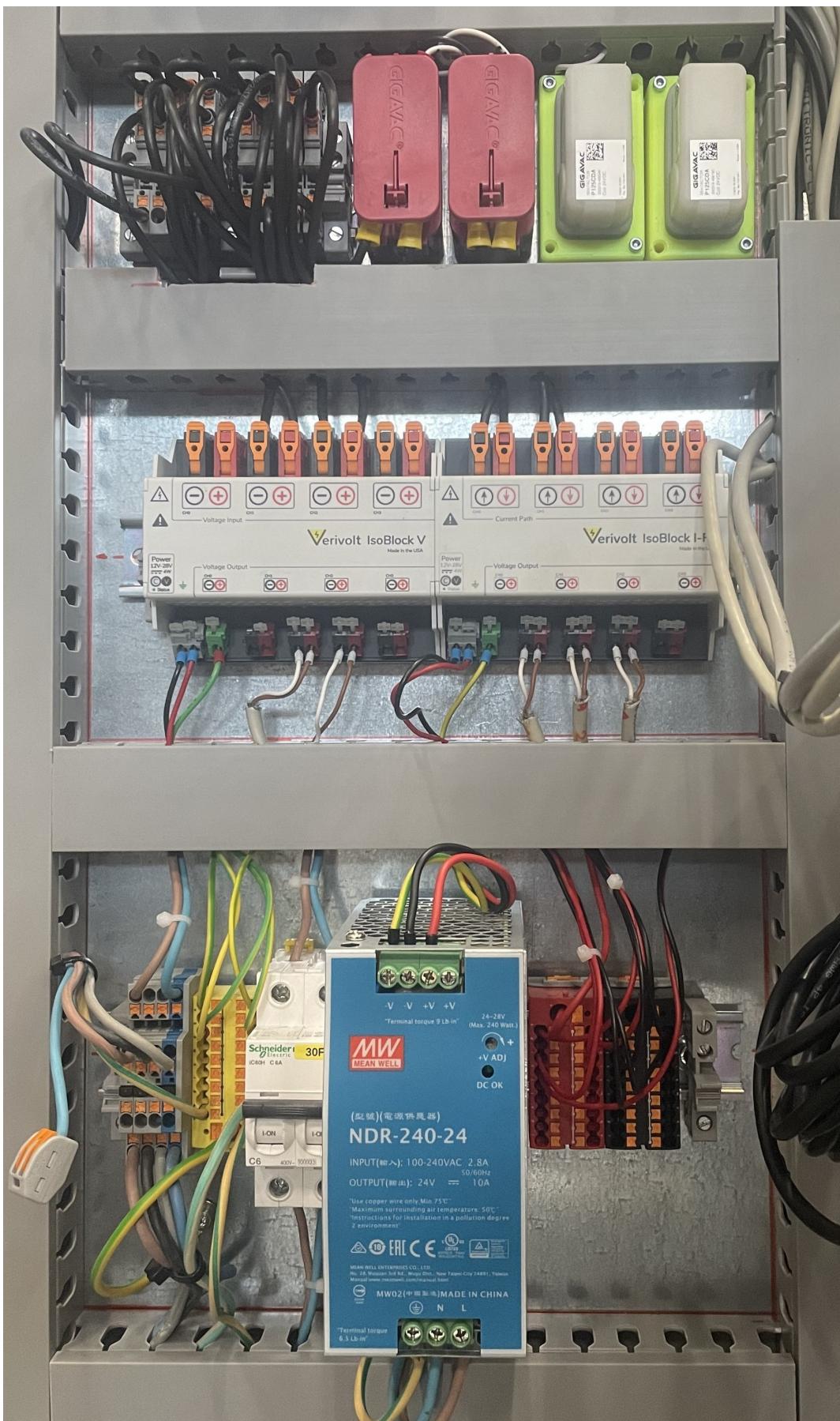


FIGURE 3 – Photo du système.

.4 Diagramme de Classe UML

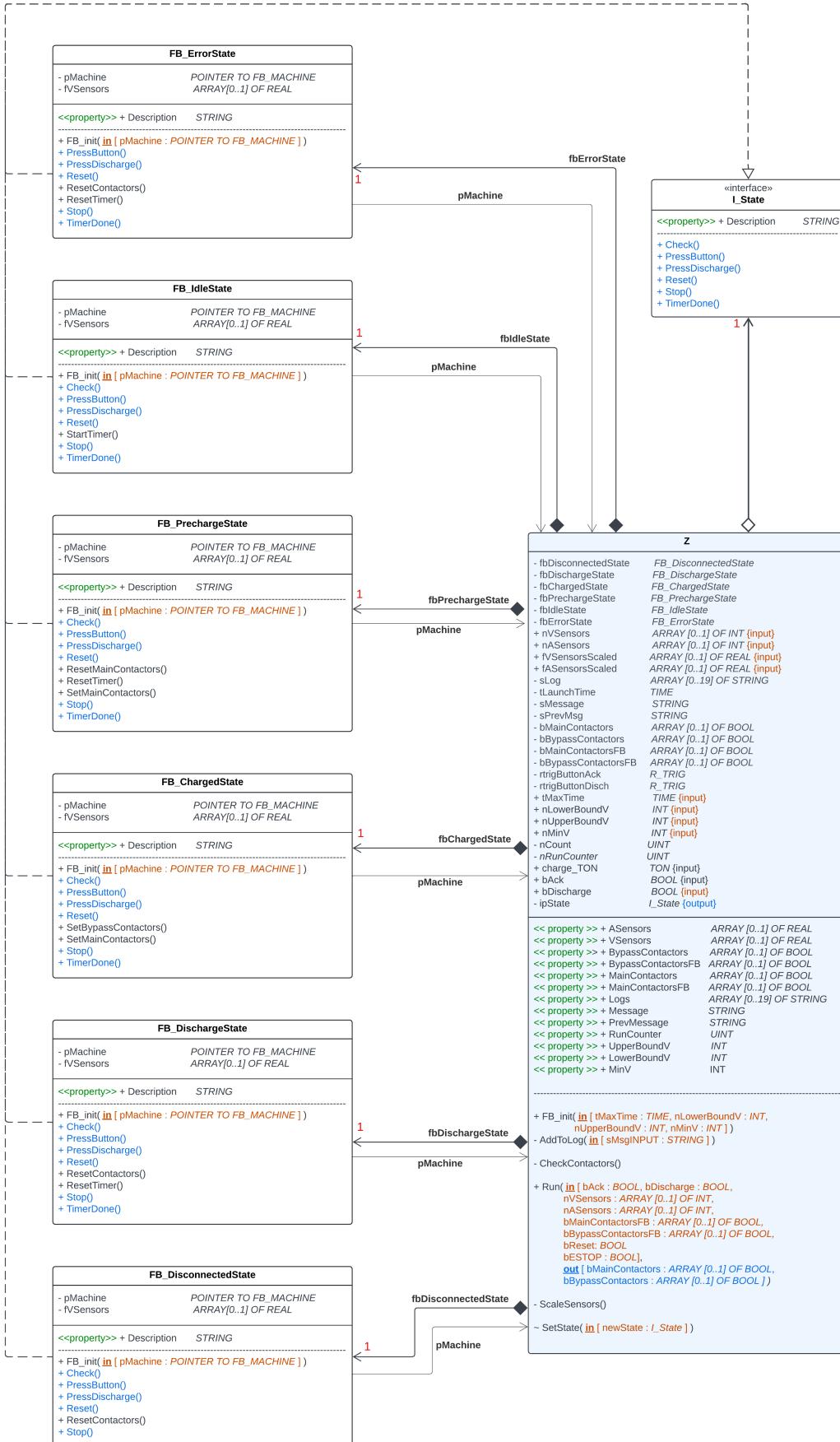


FIGURE 4 – Diagramme de classe UML du projet PLC TwinCAT.

Le fichier source de ce diagramme est disponible ici : [LucidChart](#) et le fichier .png est disponible sur le dépôt GitHub du projet [2].

.5 Diagramme d'États

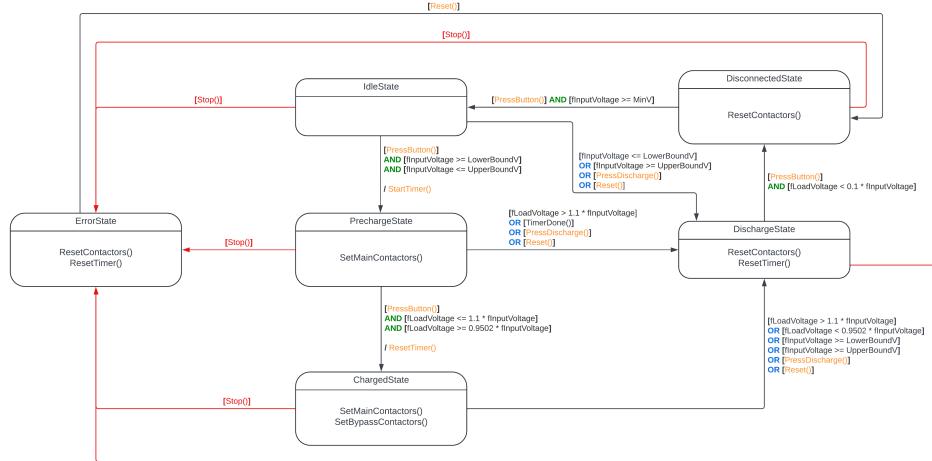


FIGURE 5 – Diagramme d'états de la machine à états.

Le fichier source de ce diagramme est disponible ici : [LucidChart](#). Le fichier .png est disponible sur le dépôt GitHub du projet [2].

.6 Diagramme de Séquence

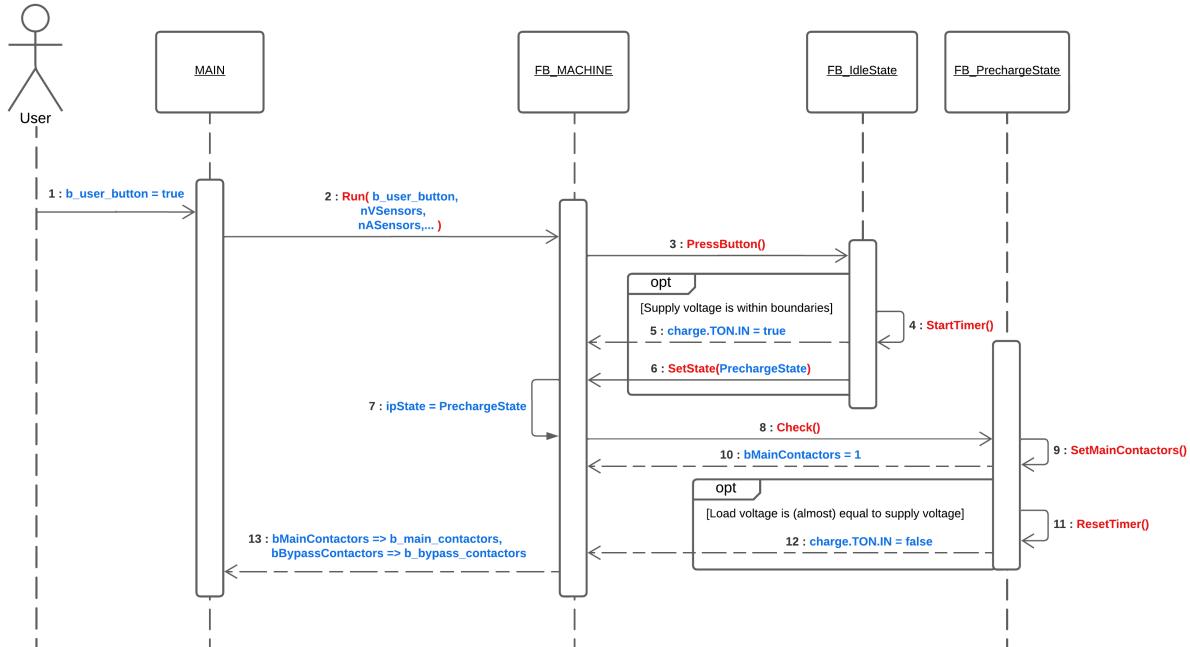


FIGURE 6 – Diagramme de séquence de la machine à états.

Le fichier source de ce diagramme est disponible ici : [LucidChart](#). Le fichier .png est disponible sur le dépôt GitHub du projet [2].

.7 Diagramme d'États - FB_FileWriter

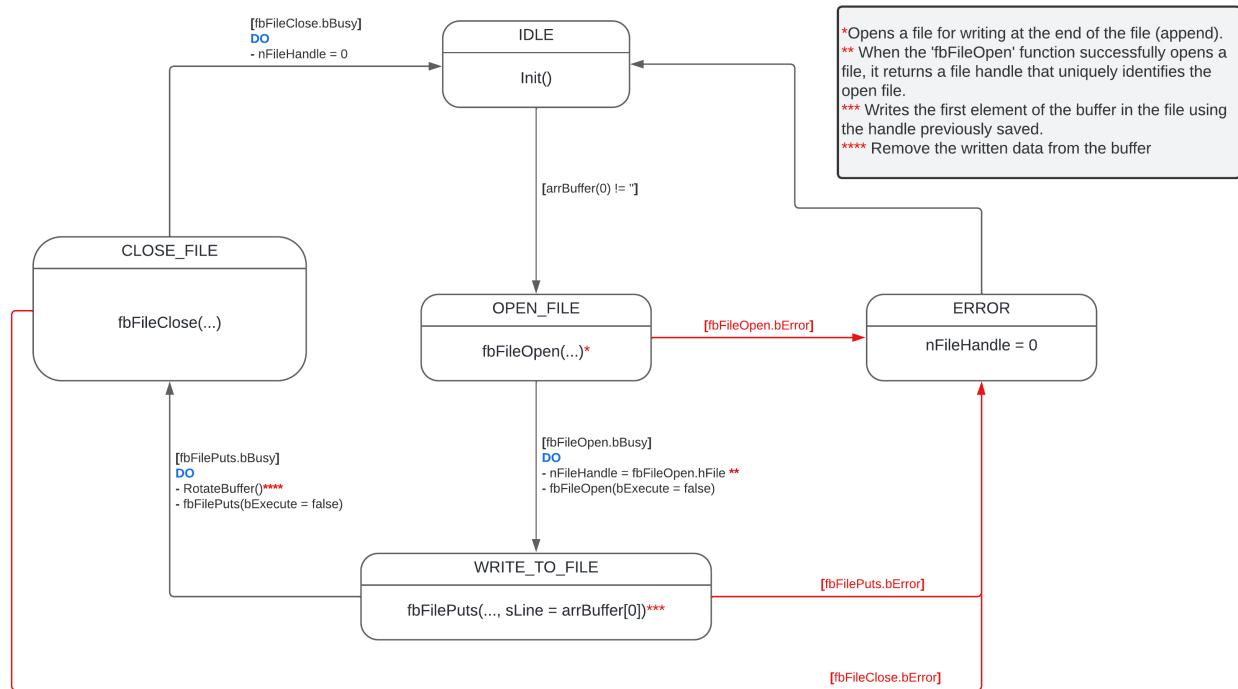


FIGURE 7 – Diagramme d'états de la machine à états codée dans le fichier FileWriter.

Le fichier source de ce diagramme est disponible ici : [LucidChart](#). Le fichier .png est disponible sur le dépôt GitHub du projet [2].

.8 Circuits Simulation LTSpice

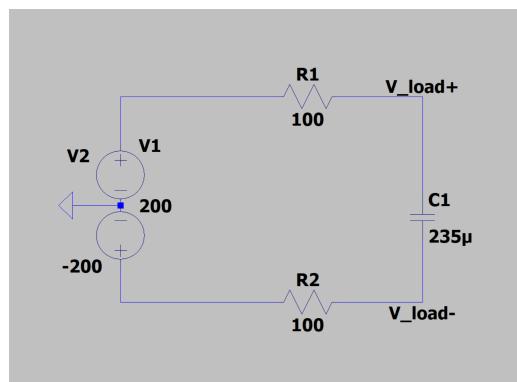


FIGURE 8 – Circuit utilisé pour la simulation LTSpice 4.4.

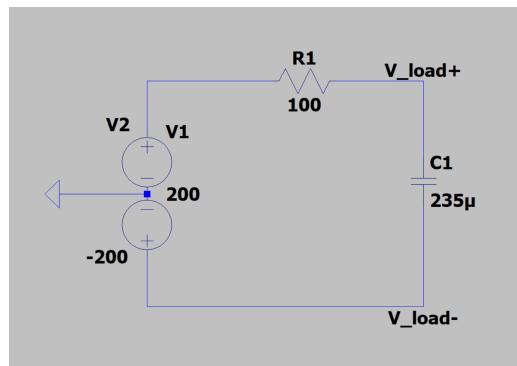


FIGURE 9 – Circuit utilisé pour la simulation LTSpice 4.5.

Les fichiers sources de ces circuits sont disponibles sur le dépôt GitHub du projet [2].