Université de Technologie de
Belfort-Montbéliard

Master Ingénierie des Systèmes Complexes
Spécialisation Mécatronique

Rapport Technique Détaillé

Contrôle d'un Système d'Ascenseur Didactique

Auteur :
Yawo Emmanuel YOVO

Encadrant :
M. Bruno AVISSE

Automne 2025

Table des matières

Chapitre 1

Introduction

Ce rapport technique documente la conception et l'implémentation du système de contrôle-commande pour une maquette d'ascenseur à quatre niveaux (modèle ASC89). Ce projet, réalisé dans le cadre du Master Ingénierie des Systèmes Complexes-Mécatronique, répond à une problématique classique de l'automatisme : la gestion d'événements asynchrones (appels aléatoires) et le pilotage précis d'actionneurs dans un environnement temps réel.

L'objectif de ce document est de fournir l'ensemble des informations techniques nécessaires à la reproduction du système. Il détaille le choix et le rôle des composants matériels, l'architecture informatique distribuée (FPGA/RT) sur la cible NI myRIO, ainsi que les algorithmes de décision implémentés sous LabVIEW.

Chapitre 2

Analyse et Description du Matériel

La réussite d'un projet mécatronique repose d'abord sur une compréhension fine des composants en interaction. Le système se divise en deux parties distinctes : la partie opérative (l'ascenseur physique) et la partie commande (l'intelligence embarquée).

2.1 La Partie Opérative : Maquette ASC89

La maquette ASC89 simule le comportement physique d'un ascenseur réel. Elle est constituée d'une structure verticale desservant quatre étages (RDC, 1, 2, 3) et d'une cabine mobile comme le montre la figure 2.1.

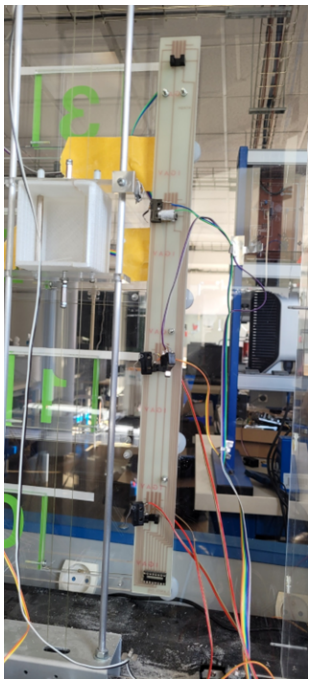

FIGURE 2.1 – Ascenseur ASC89

2.1.1 Système d'Actionnement (Motorisation)

Le mouvement est assuré par des moteurs à courant continu (DC Motors). Leur rôle est de convertir l'énergie électrique en mouvement mécanique rotatif. Dans notre projet, cinq motoréducteurs (2.2 sont utilisés pour deux fonctions distinctes :

1. **Treuil de la cabine (1 moteur) :** Ce moteur principal assure la translation verticale de la cabine. Il nécessite un couple élevé pour soulever la charge.
2. **Mécanisme des portes (4 moteurs) :** Chaque étage dispose de son propre moteur dédié à l'ouverture et la fermeture des portes palières.

Ces actionneurs sont pilotés en tension. Une inversion de la polarité permet de changer le sens de rotation (montée/descente ou ouverture/fermeture). Cependant, le myRIO ne pouvant fournir la puissance nécessaire (courant trop faible et 5v maximum en sortie), une interface de puissance est indispensable.

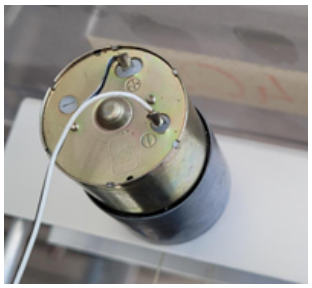

FIGURE 2.2 – moteur

2.1.2 Système d'Acquisition (Capteurs)

Pour asservir le système, il est impératif de connaître l'état physique de l'ascenseur à tout instant. Il a été utilisé pour connaître l'état physique, des :

— **Capteurs de position cabine :** Placés à chaque étage, ils changent d'état logique lorsque la cabine atteint le niveau désiré. Ils servent à l'arrêt précis de l'ascenseur.
— **Capteurs de fin de course portes :** Chaque porte est équipée de deux capteurs (un pour l'état "fermé", un pour l'état "ouvert"). Ils garantissent la sécurité en empêchant le démarrage de la cabine si une porte est ouverte.

Nous utilisons pour cela des capteurs Tout-Ou-Rien (TOR), de type SS-01GL137 (2.3). Ils envoient un signal quand il y a l'activation du bouton poussoir dont ils disposent. Le passage de la cabine à chaque étage permet d'assurer le contact avec le bouton poussoir comme le montre la figure 2.4.

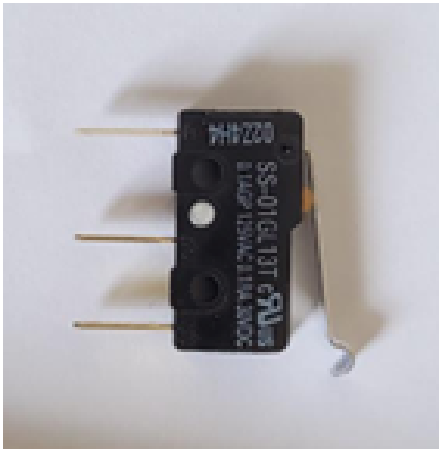

FIGURE 2.3 – Capteur TOR

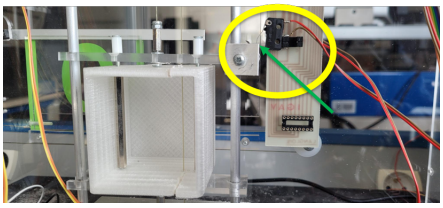

FIGURE 2.4 – Signal de présence cabine

2.2 L'Interface de Puissance : Drivers L298N

Les sorties numériques du contrôleur travaillent en logique 5V/3.3V avec des courants faibles. Les moteurs, eux, nécessitent une tension de 12V et un courant important. Le module **L298N** est un double pont en H (H-Bridge) conçu pour combler ce fossé. Son rôle est double :

- **Amplification de puissance :** Il puise l'énergie d'une alimentation externe pour la délivrer aux moteurs selon les commandes logiques reçues.
- **Contrôle bidirectionnel :** Grâce à sa structure en H, il permet d'inverser le sens du courant dans le moteur, et donc son sens de rotation, simplement en commutant deux entrées logiques.

Implémentation : Trois modules L298N ont été intégrés pour piloter les 5 moteurs indépendamment.

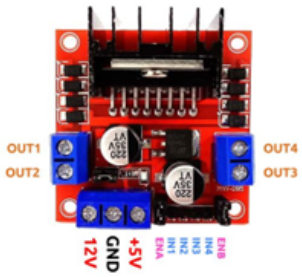

FIGURE 2.5 – image driver L298N

2.3 Le Cerveau du Système : NI myRIO-1900

Le choix du contrôleur s'est porté sur le NI myRIO pour son architecture hétérogène, combinant un processeur temps réel et un FPGA (Field-Programmable Gate Array). Cette dualité est au cœur de la performance du projet.

FIGURE 2.6 – myRIO

2.3.1 Interfaces d'Entrées/Sorties : Les Ports MXP

Pour s'interfacer avec l'environnement extérieur, le myRIO dispose de deux connecteurs d'extension identiques nommés MXP A et MXP B (myRIO Expansion Ports). Ces connecteurs haute densité permettent d'accéder directement aux lignes d'entrées/sorties du FPGA.

Chaque port MXP fournit :

— 16 lignes d'Entrées/Sorties Numériques (DIO).

— Des Entrées Analogiques (AI) et Sorties Analogiques (AO).

— Des sorties d'alimentation (+5V, +3.3V) pour les capteurs externes.

2.3.2 Rôle du FPGA (Bas niveau)

Le FPGA est un circuit logique reconfigurable matériellement. Contrairement à un processeur qui exécute des instructions séquentiellement, le FPGA traite les signaux en parallèle à la vitesse de l'horloge matérielle (40 MHz). Il est dédié à la gestion critique des Entrées/Sorties (I/O). Il assure la lecture de manière rapide des capteurs et la génération des signaux PWM pour les moteurs, sans latence, garantissant une réactivité de l'ordre de la microseconde. Voici ses spécifications :

— **Type :** Xilinx Z-7010

— **Horloge :** 40 MHz (Cycle de 25ns).

2.3.3 Rôle du Processeur Real-Time (Haut niveau)

Le processeur ARM Cortex-A9 exécute un OS (Operating System) temps réel (Linux RT). Il héberge l'intelligence du système, les machines à états et les algorithmes décisionnels. Il traite les données communique avec le FPGA pour prendre des informations et commander des actions. Voici ses spécifications :

— **Type :** ARM Cortex-A9 Dual-Core.

— **Fréquence :** 667 MHz (Cycle 14,99ms).

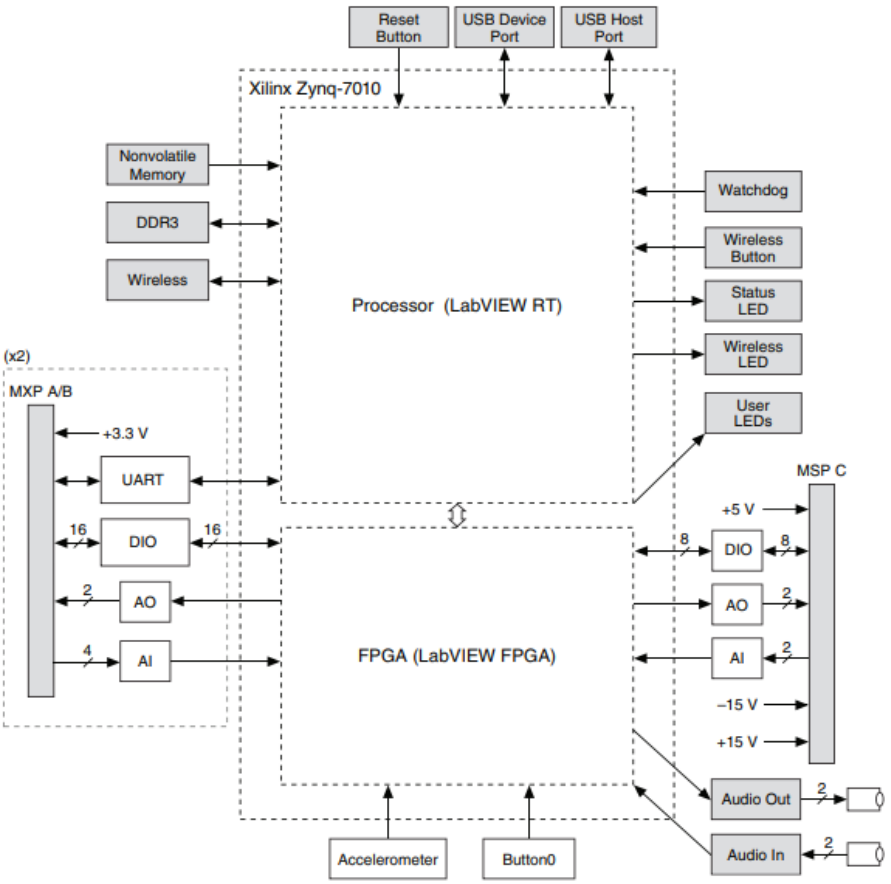

FIGURE 2.7 – composants myRIO

Chapitre 3

Architecture Système et Câblage

3.1 Architecture physique

La mise en œuvre physique nécessite une rigueur dans le raccordement pour éviter les courts-circuits et assurer la fiabilité des signaux.

Stratégie de Câblage adoptée : Afin de clarifier le câblage et d'éviter les interférences, nous avons opté pour une séparation physique des flux de données (voir schéma électrique en Annexe C.1) :

1. **Port MXP A (Acquisition) :** Il est dédié à la lecture des capteurs. Il centralise les signaux provenant des 16 capteurs TOR (Positions cabine et états des portes). Le FPGA scanne ce port pour mettre à disposition de la couche RT l'état des portes et la position de la cabine.
2. **Port MXP B (Commande) :** Il est dédié au pilotage des actionneurs. Il transmet les signaux de commande. Une masse commune (GND) a été établie entre l'alimentation de puissance des moteurs, les drivers L298N et le myRIO pour assurer une référence de potentiel stable.

Cette architecture permet une maintenance simplifiée : en cas de problème sur un moteur, seul le connecteur B est concerné, sans risque de déconnecter un capteur sur le port A.

L'architecture logicielle suit le modèle "Maître/Esclave", permettant de découpler la gestion du matériel de la logique décisionnelle.

3.2 Collaboration FPGA / Real-Time

C'est le point clé de l'architecture. Le code n'est pas monolithique mais distribué :

1. **Le VI FPGA :** Ce programme est implémenté dans le silicium du FPGA. Il lit en permanence l'état des capteurs physiques et écrit sur exposent . Il expose ces

états au processeur RT à travers des variables. En parallèle, il expose des variables de commande à la couche RT pour le pilotage des moteurs.

2. **Le VI Real-Time :** Il s'exécute sur le processeur. Il vient lire les registres du FPGA pour connaître l'état des capteurs, exécute l'algorithme, et écrit dans les registres du FPGA pour commander les moteurs.

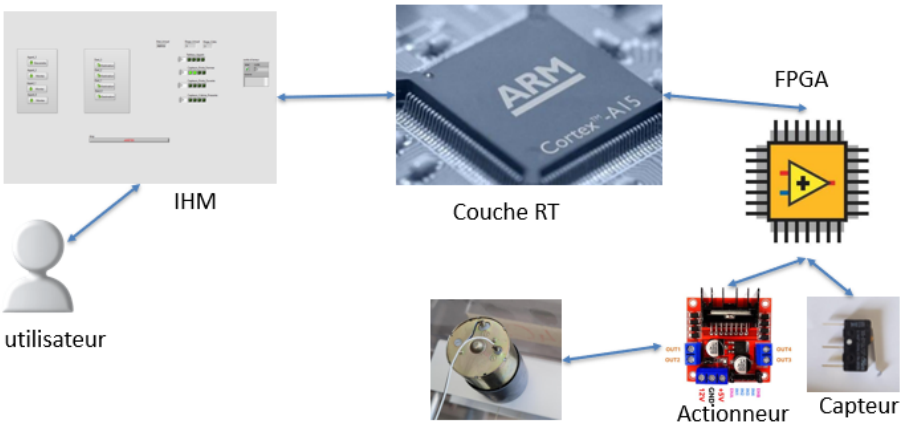

FIGURE 3.1 – architecture

Chapitre 4

Implémentation Logicielle sous LabVIEW

4.1 Architecture de la Machine à États (State Machine)

Le programme principal est structuré autour d'une machine à états standard, garantissant que l'ascenseur ne peut être que dans une seule configuration à la fois. Les états principaux sont :

- **INIT :** l'objectif de cet etat est d'assurer une position connue à la cabine.
- **REPOS :** L'ascenseur est à l'arrêt, portes fermées, en attente d'un appel pour passer à l'état DEPLACEMENT.
- **DEPLACEMENT :** Activation du moteur cabine pour atteindre une destination (étage cible). L'algorithme vérifie en continu si l'étage actuel correspond à l'étage cible et passe a l'état OUVERTURE PORTE.
- **OUVERTURE PORTE :** L'ascenseur a atteint sa destination. Elle ouvre la porte de l'étage. Cet etat gère le moteur de porte avec vérification du capteur de fin de course associée à la porte ouverte. L'ascenseur passe à l'état ATTENTE TEMPO quand le capteur de porte ouverte est actif.
- **ATTENTE TEMPO :** Dans cet etat, on simule le temps d'entrée (ou de sortie) des personnes de l'ascenseur. L'ascenseur passe à l'état FERMETURE PORTE après une temporisation de 5 secondes.
- **FERMETURE PORTE :** Les porsonnes sont entrée. La porte de l'étage se ferme. Cet etat gère le moteur de porte avec vérification du capteur de fin de course associée à la porte fermée. L'ascenseur passe à l'état REPOS quand le capteur porte fermée est activé.

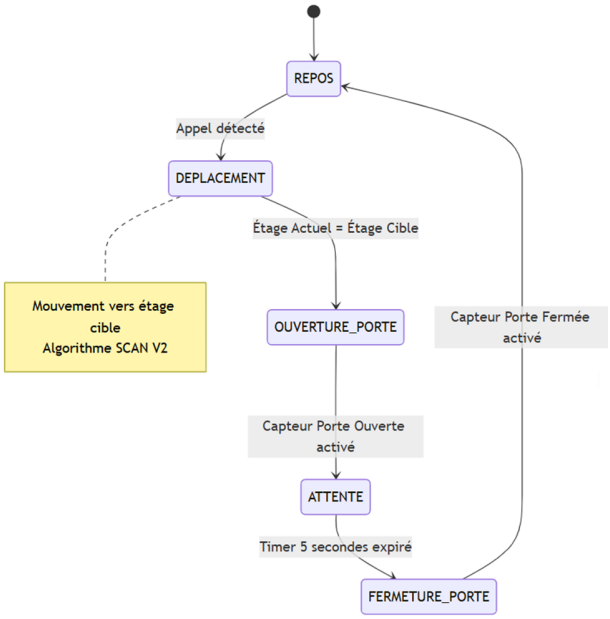

FIGURE 4.1 – diagramme d'état

4.2 L'Algorithme de Gestion SCAN V2

Pour optimiser les déplacements, un algorithme simple de type FIFO (Premier arrivé, premier servi) serait inefficace. Nous avons implémenté l'algorithme SCAN (ou algorithme de l'ascenseur) :

- L'ascenseur continue dans sa direction actuelle tant qu'il y a des appels dans cette direction.
- Une fois la demande la plus éloignée satisfaite, il change de direction.
- Cela évite les oscillations inutiles et réduit le temps d'attente moyen.

4.3 Implémentation de la couche FPGA

Comme présentée au chapitre précédent, la couche FPGA est chargée de communiquer avec les composants physiques.

4.3.1 Lecture des capteurs TOR

Pour communiquer avec les capteurs sur la couche FPGA, il faut spécifier les entréés (FPGA I/O) qui correspondent aux références des broches DIO (Digital Input/Output). Le schéma électrique C a été utilisé pour associer la bonne entrée I/O au bon capteur. Ensuite le VI (virtual instrument) **Noeud d'Entrée/Sortie FPGA** a été utilisé pour effectué la lecture. Ce VI prend en entrée la référence I/O FPGA (la broche) et fourni en sortie l'état de l'entrée. Pour lire l'état de tous les capteurs, il faut donc utiliser autant

de VI **Noeud d'Entrée/Sortie FPGA** que de capteurs.
Les capteurs ont été regroupés en trois (03) catégories :

- Portes ouvertes : ce sont tous les capteurs de porte ouverte,
- Portes fermées : ce sont tous les capteurs de porte fermée,
- Présence cabine : ce sont tous les capteurs de présence cabine qui sont situées à chaque étage.

Un tableau a été utilisé regroupé l'état des capteurs qui est exposé à la couche RT. Il y a donc au total trois (03) tableaux pour les capteurs :

- Portes_ouvertes : tableau de booléens ayant 4 cases. L'état de chaque case i indique si la porte de l'étage $i+1$ est ouverte ou non.
- Portes_fermees : tableau de booléens ayant 4 cases. L'état de chaque case i indique si la porte de l'étage $i+1$ est fermée ou non.
- Présence_cabine : tableau de booléens ayant 4 cases. L'état de chaque case i indique si la cabine est présente à l'étage $i+1$ ou non.

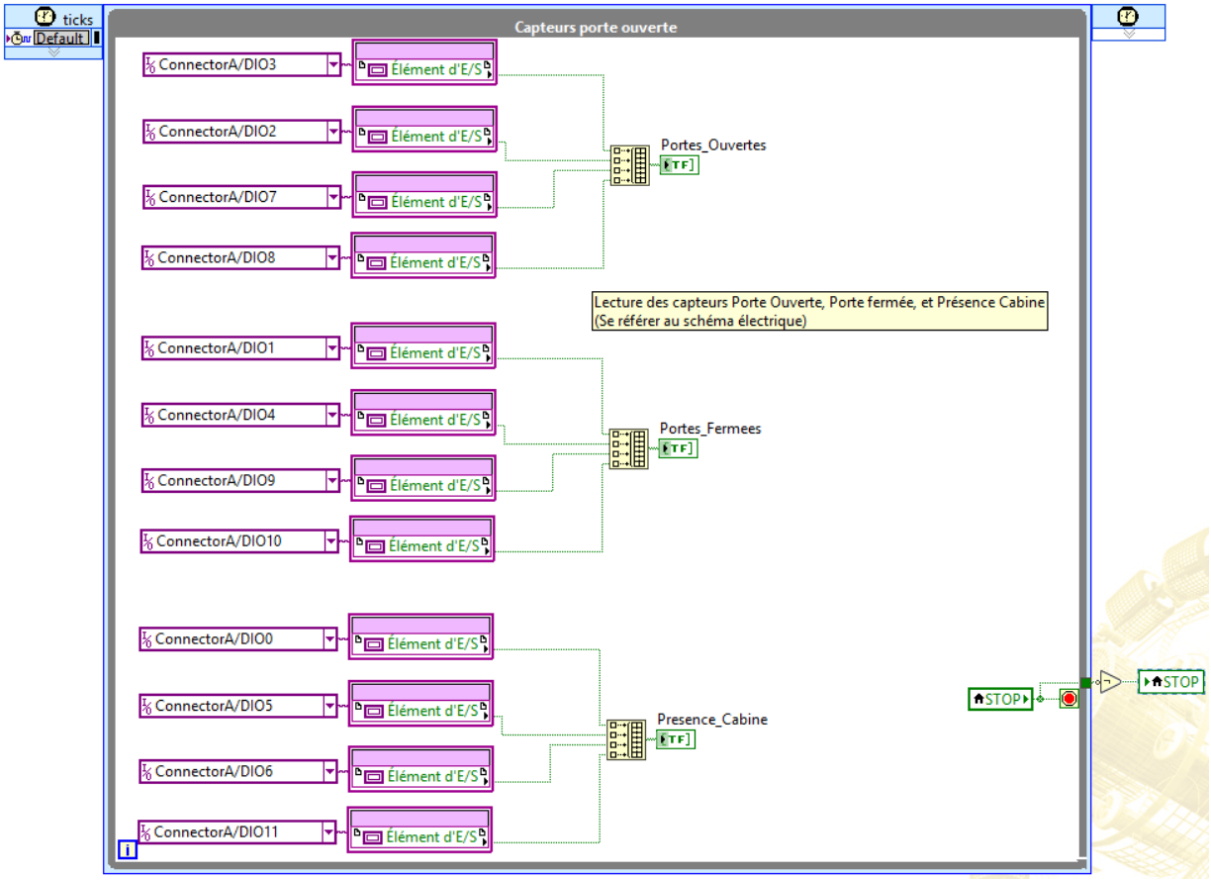

FIGURE 4.2 – Lecture des capteurs sur FPGA

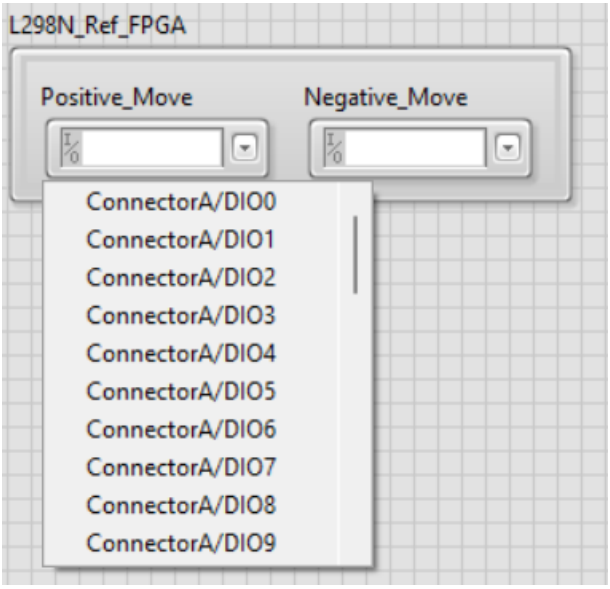

FIGURE 4.3 – TypeDef L298N

4.3.2 Commande des moteurs

Les moteurs sont commandés par le FPGA au travers du pont L298N comme spécifié plus haut dans le rapport. Un driver L298N a été créé sur la couche FPGA. Dans cette partie, les détails de l'implémentation seront présentées. Cependant, la table de vérité et la configuration du pont L298N sont présentées en annexe B.

TypeDef L298N$_{Ref}$

Un type de donnée personnalisé a été créé pour faciliter l'implémentation. Pour un moteur, le pont L298N expose deux broches IN1 et IN2 (Input 1 et Input 2), chacun pilotant respectivement le deplacement dans le sens positif et le sens négatif (confère annexe B). Donc le TypeDef personnalisé contient deux I/O (entrée sortie) FPGA : une pour le Input 1 et l'autre pour le Input 2.

Le driver (VI L298N_Driver_FPGA)

Il prend en entrée un cluster de configuration `L298N_Ref` et une consigne d'action. L'action est définie par une énumération stricte (*Typedef*) contenant les états : **Arrêt (Frein)**, **Montée**, **Descente**, qui correspondent aux combinaisons logiques appliquées aux ponts en H. Voici la table de vérité qui est exécutée :

Input A	Input B	Comportement du Moteur
0	0	**Arrêt (Freinage)**
0	1	**Rotation Sens 1*** (ex : Monter / Ouverture)
1	0	**Rotation Sens 2*** (ex : Descendre / Fermeture)
1	1	**Arrêt (Freinage)**

TABLE 4.1 – Table de vérité L298N (Configuration avec Jumpers Enable)

*Le sens de rotation physique réel (Horaire/Anti-horaire) dépend de la polarité de connexion des câbles moteurs sur le bornier de sortie du driver. Lors du cablâge, une erreur d'inversion de polarité a été faite pour les moteurs.

Une structure condition a été utilisée pour affecter les valeurs des différentes entrées du pont H. Ensuite le VI Noeud E/S FPGA a été utilisé pour écrire sur les bonnes entrées du pont.

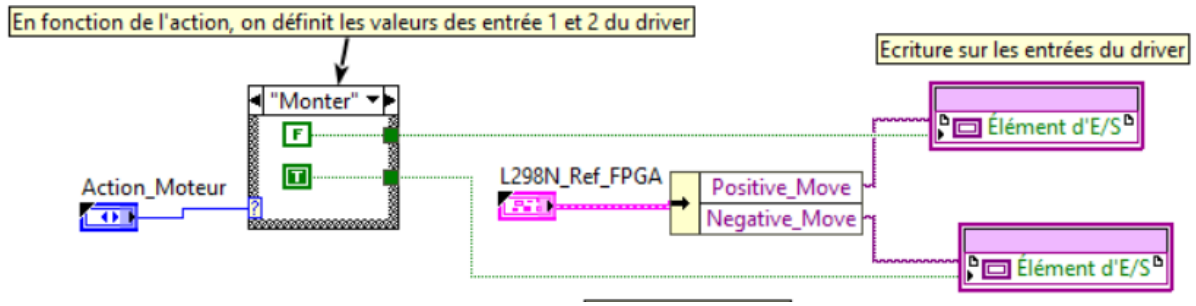

FIGURE 4.4 – Diagramme du driver L298N

Le main du FPGA)

Dans la face avant du VI principal du FPGA, il y a les différentes variable d'intéraction : **les tableaux** et **les commandes moteurs**. Ainsi, la couche FPGA viendra lire les tableaux et écrire sur les commandes moteurs (typedef enum). Comme l'ascenseur comporte 5 moteurs, il y a donc 5 commandes moteurs exposées par le FPGA : les moteurs **cabine**, **porte RDC**, **porte étage 1**, **porte étage 2**, **porte étage 2**.

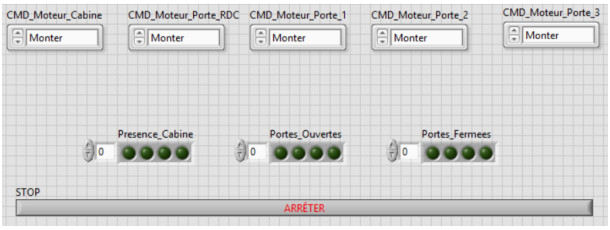

FIGURE 4.5 – Face avant du VI Main FPGA

Dans le diagramme du VI, une boucle cadencée est dédiée à chaque moteur. Avant l'entrée de chacune des boucles, les références L298N_Ref_FPGA sont initialisées. Dans la boucle, on appelle le VI L298N_Driver_FPGA auquel on passe la référence L298N et la commande (qui est modifiable par la couche RT) pour écrire sur les sorties du pont H.

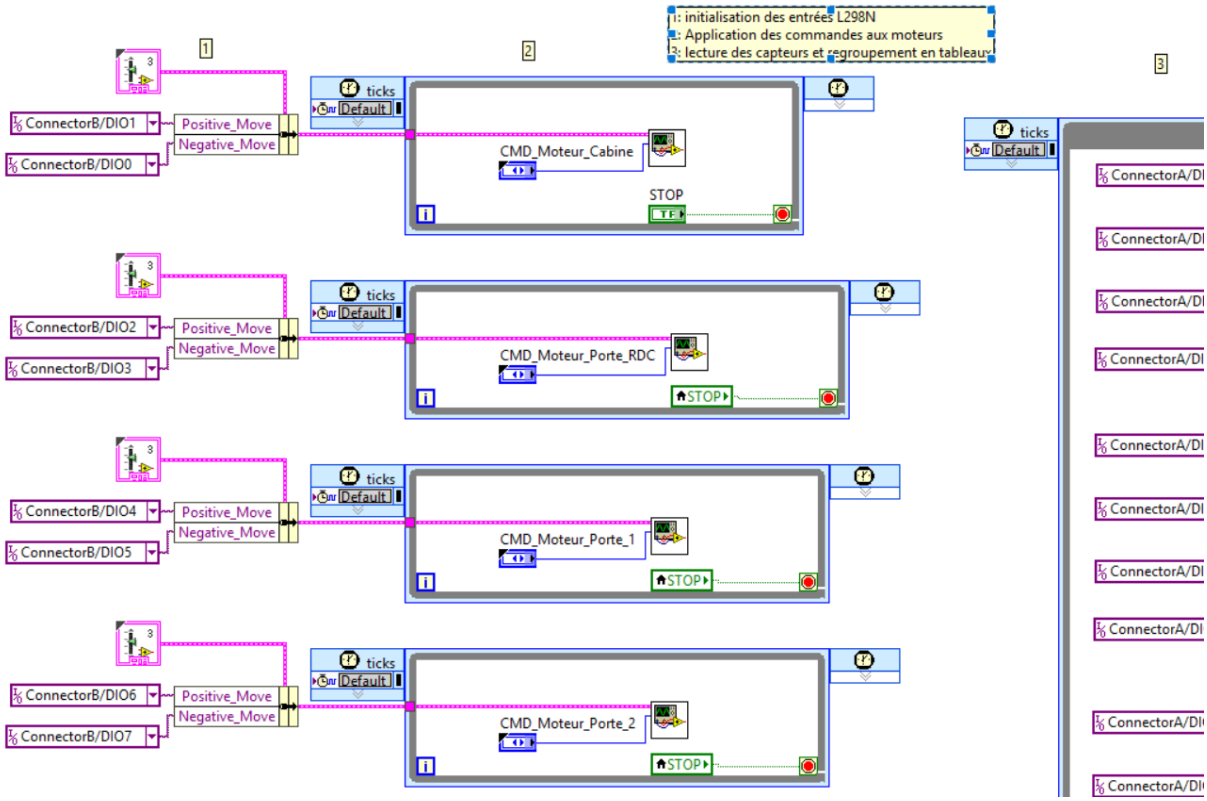

FIGURE 4.6 – Bloc diagramme du VI Main FPGA

4.4 Implémentation de la couche RT

La couche Real-Time, hébergée sur le processeur ARM, orchestre le fonctionnement de l'ascenseur. Son implémentation repose sur une architecture de Machine à États Finis (State Machine).

Pour garantir la modularité et la lisibilité du code LabVIEW, l'ensemble des données nécessaires au cycle de décision a été encapsulé dans une structure de données unique.

4.4.1 Gestion des Données : Le Cluster d'État

Contrairement à une approche par variables globales, nous avons opté pour un flux de données unique circulant via un registre à décalage (Shift Register). Ce flux est matérialisé par un *Cluster* défini en tant que Typedef Strict, ce qui permet de modifier la structure de données en un seul point et de propager les changements à tout le projet.

Ce cluster regroupe trois typologies de variables distinctes :

a. Les Variables d'État (Mémoire du Système)

Elles définissent la situation logique de l'ascenseur et permettent l'exécution de l'algorithme séquentiel. Ce sont les seules véritables variables d'état au sens de l'automatisme.

Variable	**Type**	**Rôle dans l'algorithme**
`Etat_Actuel`	Énumération	Définit l'état actif de la machine (ex : *REPOS, DEPLACEMENT, etc.*). C'est le cœur du séquentiel.
`Etage_Cible`	Entier	Mémoire de la destination en cours. L'ascenseur compare son étage actuel à cette valeur pour décider de s'arrêter.
`Direction_Cabine`	Énumération	Mémoire du sens de déplacement (Montée/Descente). Indispensable pour l'algorithme SCAN.
`Tableau_Appels`	Tableau de Bool	File d'attente des requêtes. Chaque index correspond à un étage, la valeur Vrai indique qu'un arrêt y est demandé.

TABLE 4.2 – Variables internes de mémorisation

b. Les Entrées Capteurs

Ces variables sont mises à jour à chaque itération par lecture du FPGA. Elles servent de conditions pour les transitions d'état.

- **Etage_Actuel :** Résultat du traitement des capteurs de position. Indique où se trouve physiquement la cabine.
- **Capteurs_Porte (Ouv./Ferm.) :** Image binaire de l'état des portes.
- **Capteurs_Présence_Cabine :** Image brute des détecteurs de fin de course.

c. Ressources Système

Enfin, le cluster transporte la référence `Référence_VI_FPGA`. C'est un pointeur logiciel permettant aux sous-VI (Handlers) d'accéder aux moteurs et aux capteurs sans devoir rouvrir une session FPGA coûteuse en temps CPU.

4.4.2 Architecture Générale du Programme

L'application Real-Time est structurée selon le modèle de conception standard **Initialisation - Boucle Principale - Fermeture** de LabVIEW. Ce modèle garantit que

les ressources matérielles sont correctement allouées au démarrage et libérées proprement à l'arrêt, évitant ainsi les fuites de mémoire ou les états indéfinis.

L'exécution séquentielle se décompose en quatre phases distinctes :

a. Initialisation (Sous-VI Open Refs)

Avant d'entrer dans la boucle de contrôle, le programme prépare l'environnement (voir VI d'initialisation).

- **Ouverture de Session FPGA :** Le RT initie la communication avec le circuit FPGA via la fonction `Open FPGA VI Reference`. Cela charge le bitfile (fichiers FPGA compilés) et établit le pont de communication.
- **Instanciation du Cluster :** Le cluster de données global est créé avec des valeurs par défaut sécuritaires (État initial = `INIT`, Moteurs à l'arrêt, Tableaux initiés à false). C'est ici que la référence FPGA est injectée dans le cluster pour être transmise aux sous-VI.

b. Acquisition des Données (Sous-VI Read Inputs)

Au début de chaque itération de la boucle principale, une phase de lecture est exécutée pour mettre à jour l'état du système. Le sous-VI `Read_Inputs` centralise toutes les sources d'information :

- **Lecture Hardware :** Il interroge les indicateurs du FPGA (via la méthode *Read/Write Control*) pour récupérer l'état physique des capteurs (Portes, Présence Cabine).
- **Lecture IHM :** Il récupère les interactions utilisateur depuis la Face Avant (Boutons d'appel, Boutons de destination).

Ces données brutes sont ensuite regroupées dans le cluster principal, garantissant que la machine à états qui suit travaille sur une les données mises à jour du système.

c. Le Cœur du Système : Machine à États (Main)

La logique décisionnelle réside dans une boucle *While* qui assure la persistance des données via un registre à décalage. L'architecture interne repose sur une Structure Conditionnelle (*Case Structure*) pilotée par la variable `Etat_Actuel`.

Concept des Handlers (Gestionnaires d'état) : Pour maintenir le code lisible et modulaire, le code à l'intérieur de chaque état a été encapsulé dans des sous-VI spécifiques appelés Handlers (ex : `Handler_Repos`, `Handler_Deplacement`).

1. Le Handler reçoit le Cluster d'état complet.
2. Il exécute la logique propre à cet état (ex : vérifier si un bouton est appuyé).
3. Il met à jour les variables (ex : changer l'état vers `DEPLACEMENT`).

4. Il renvoie le Cluster modifié vers le registre à décalage pour l'itération suivante.

Pour faciliter la lecture du code, voici le tableau des sous-VI et leurs rôles :

Nom du Sous-VI	**Fonctionnalité et Rôle**
`Init_Handler.vi`	Initialise les variables du cluster (tableaux vides, moteurs à l'arrêt) au démarrage.
`Repos_Handler.vi`	État d'attente active. Surveille le tableau d'appels pour détecter une nouvelle requête et déclencher le mouvement.
`Calculateur_Destination.vi`	Cerveau de l'algorithme SCAN. Analyse les appels en cours pour déterminer le prochain étage logique à desservir.
`Deplacement_Handler.vi`	Pilote le moteur de la cabine vers l'étage cible et compare la position actuelle avec la destination.
`Ouverture_Porte_Handler.vi`	Active le moteur d'ouverture jusqu'à détection du capteur Porte Ouverte.
`Attente_Tempo_Handler.vi`	Simule la temporisation permettant aux passagers d'entrer/sortir avant la fermeture.
`Fermeture_Porte_Handler.vi`	Active le moteur de fermeture jusqu'à détection du capteur Porte Fermée.
`Arret_Urgence_Handler.vi`	Coupe l'alimentation de tous les moteurs et force le système dans un état de sécurité.

TABLE 4.3 – Liste des modules fonctionnels (Handlers)

d. Arrêt et Libération (Close Refs)

Lorsque l'état `STOP` est atteint ou qu'une erreur critique survient :

— La boucle principale s'arrête.

— Une commande d'arrêt d'urgence est envoyée aux moteurs (Mise à zéro des commandes FPGA).

— La session FPGA est fermée proprement via `Close FPGA VI Reference`, libérant ainsi l'accès matériel pour d'autres applications potentielles.

Chapitre 5

Bilan et Perspectives

5.1 Difficultés Techniques Rencontrées

Lors de la phase de validation, un défaut mécanique a été détecté : le contact entre la cabine et les capteurs TOR peut être aléatoire lors de la phase de montée, provoquant des ratés de détection d'étage. Une camme a été conçue en 3D sous le logiciel tinkercad. De plus, la connectique actuelle sur les breadboard (plaque d'essai) reste complexe.

5.2 Améliorations Futures

Pour rendre le système robuste et pérenne, il est recommandé de :

1. Concevoir un circuit imprimé (PCB) dédié pour remplacer le câblage filaire et intégrer les composants proprement.
2. Faire une nouvelle conception pour fixer les capteurs TOR à l'ascenseur.
3. Valider ou améliorer la conception de la camme de contact entre la cabine et les capteurs TOR.

Chapitre 6

Conclusion

Ce projet a permis de valider une chaîne de commande complète d'un ascenseur didactique. Il a couverts les aspect matériel et logiciel. L'utilisation conjointe du FPGA du myRIO pour la rapidité d'exécution et du processeur RT pour la logique séquentielle démontre la puissance des architectures dans les systèmes embarqués modernes.

Annexe A

Présentation du myRIO

Pour plus de détails, veuillez consulter le produit détaillé sur le site officiel de National Intruments

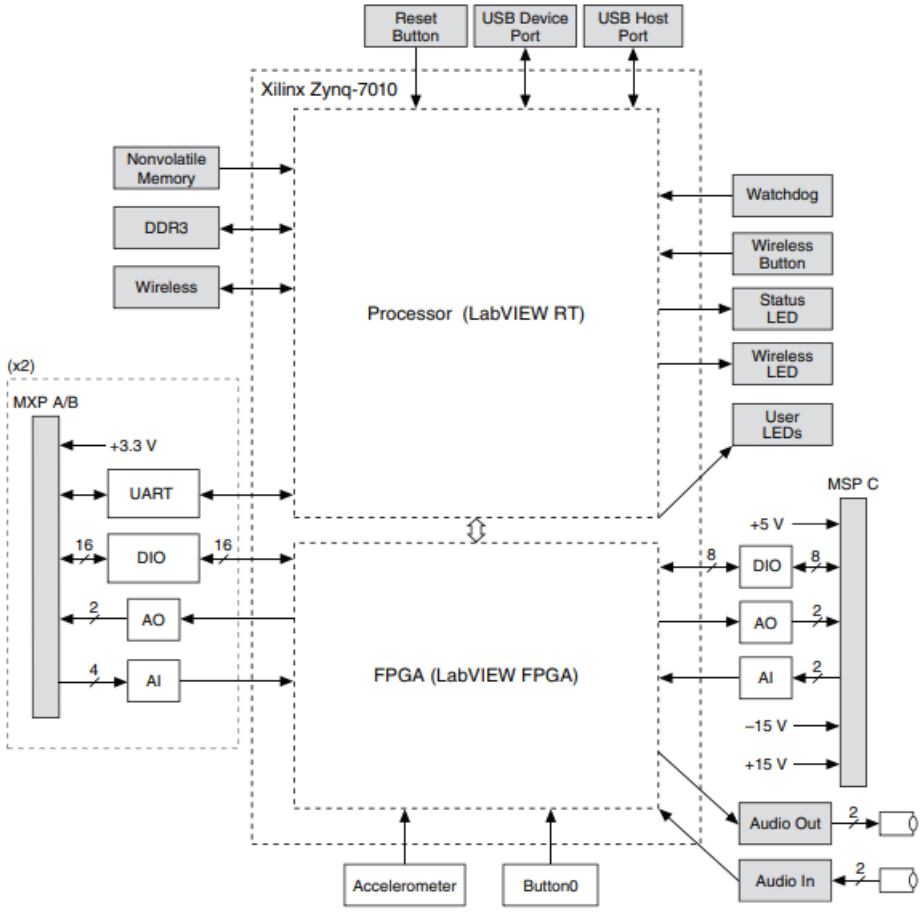

FIGURE A.1 – composants myRIO

Annexe B

Configuration du Driver L298N

Dans le cadre de ce projet, les cavaliers de shunt (jumpers) ont été maintenus sur les broches d'activation (*Enable*). Par conséquent, les moteurs sont pilotés à vitesse nominale constante.

Le tableau ci-dessous présente la logique de commande simplifiée utilisée pour les sorties moteurs (Câblage sur connecteur MXP B).

Input A	**Input B**	**Comportement du Moteur**
0	0	**Arrêt (Freinage)**
1	0	**Rotation Sens 1** (ex : Montée / Ouverture)
0	1	**Rotation Sens 2** (ex : Descente / Fermeture)
1	1	**Arrêt (Freinage)**

TABLE B.1 – Table de vérité L298N (Configuration avec Jumpers Enable)

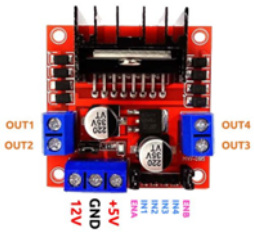

FIGURE B.1 – L298N annexe

La broche *Enable* étant forcée à 5V par le cavalier, le PWM (Pulse With Modulation) est désactivé. Le moteur reçoit la tension maximale disponible en entrée du driver. Pour plus de détails, consulter la documentation STM32Electronics

Annexe C

Schémas de Connexion

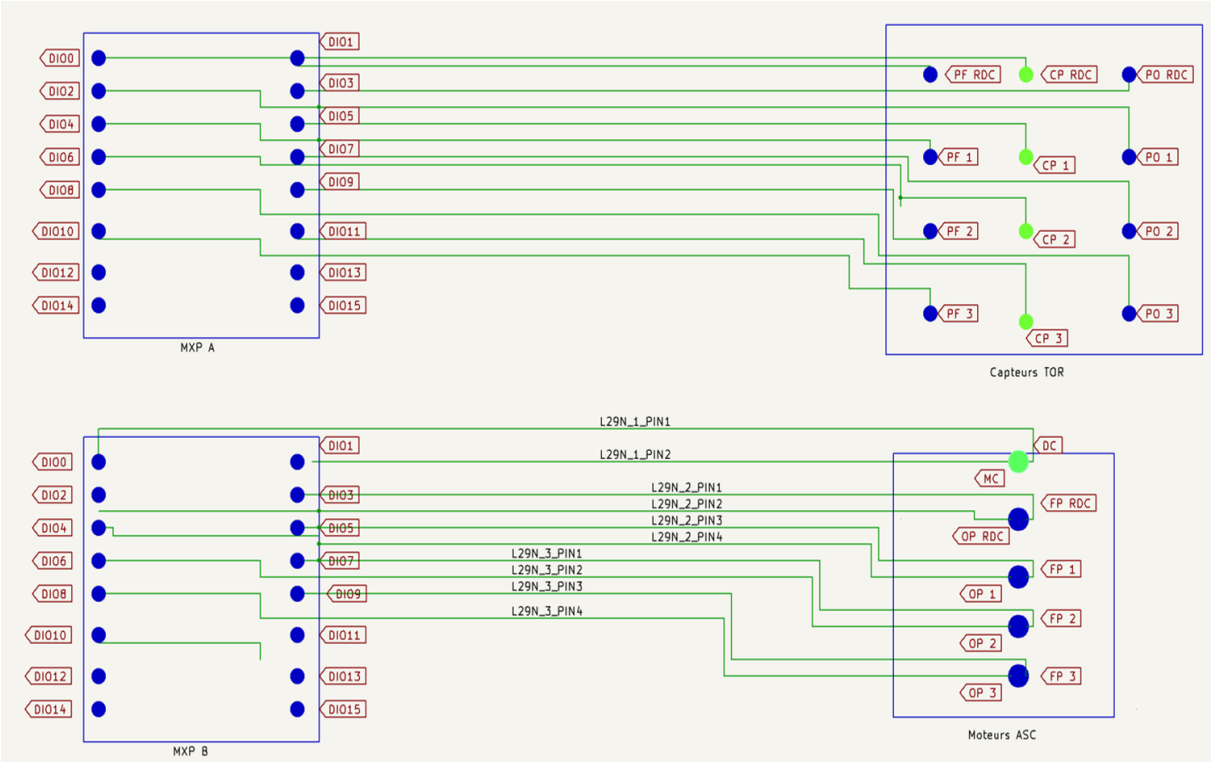

FIGURE C.1 – Schéma électrique détaillé du système

Annexe D

Diagramme d'état

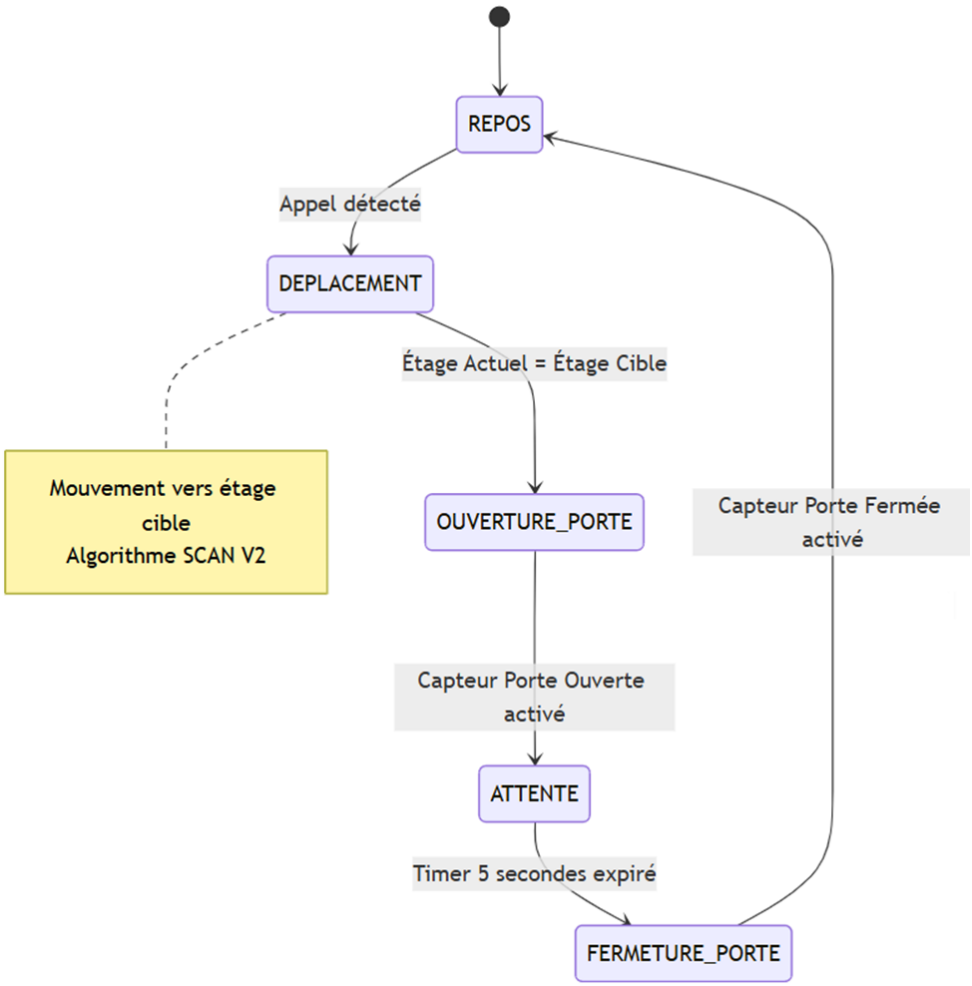

FIGURE D.1 – diagramme d'état