Table des matières

Introduction Génerale **1**

1 Présentation générale du projet **3**

Introduction 3

1.1 Présentation de la societé 3

1.1.1 Présentation de l'organisme d'accueil 3

1.1.2 Domaines d'activités 4

1.1.3 Les services de la société 5

1.2 Cadre du projet 6

1.2.1 Contexte de projet 6

1.2.2 Problématique 8

1.2.3 Solution proposée 9

1.2.4 Objectif du projet 10

1.2.5 Méthodologie de travail 11

1.2.6 Modélisation SYSML 11

1.3 Cahier des charges 12

1.3.1 Spécification des besoins 12

1.3.2 Diagramme d'exigences 14

1.3.3 Diagramme de cas d'utilisation 16

Conclusion 20

2 ÉTUDE DE LA SOLUTION TECHNIQUE DU PROJET 21

Introduction 21

2.1 Architecture générale du système 21

2.1.1 Présentation générale du système 22

2.1.2 Description fonctionnelle globale 22

2.2 Étude matérielle 23

2.2.1 Carte IHM du servomoteur 23

2.2.2 Module esp32 wroom 32 24

2.2.3 Microcontrôleur STM32G474VET6 26

2.2.4 Mémoire flash W25Q32JV 27

2.2.5 Carte de développement ESP32 28

2.2.6 PC 29

2.2.7 Convertisseur USB vers TTL 29

2.2.8 STLINK 30

2.3 Étude protocolaire 30

2.3.1 Bluetooth Low Energy 31

2.3.2 Protocole AT 34

2.3.3 Protocole de communication UART 35

2.3.4 Protocole de communication QSPI 36

2.4 Étude logicielle 37

2.4.1 Github 37

2.4.2 Docklight 38

2.4.3 Python 38

2.4.4 STM32CubeMX 39

2.4.5 IAR Embedded Workbench 39

2.4.6 STM32CubeProgrammer 40

2.4.7 HxD 41

Conclusion 41

3 Conception et réalisation du projet **42**

Introduction 42

3.1 Conception générale de la solution proposée 42

3.1.1 Conception électrique de la solution proposée 43

3.1.2 Diagramme de séquence 44

3.1.3 Structure des données BLE 45

3.1.4 Conception logicielle de la solution proposée 46

3.2 Configuration du systéme 50

3.2.1 Configuration Cube MX 50

3.2.2 Configuration Cube Programmer 52

3.3 Test du système 53

3.3.1 Test de la transmission complète du firmware 54

3.3.2 Test du flashage du firmware dans la mémoire externe 59

3.4 Résultat 61

Conclusion 62

Conclusion générale **63**

Table des figures

1.1 Logo de l' I@T 4

1.2 Servomoteur 7

1.3 Interface de l'application mobile 8

1.4 Transfert du firmware de mise à jour à travers l'application mobile 9

1.5 Cycle en V 11

1.6 Diagramme d'exigences 15

1.7 Diagramme de cas d'utilisation 16

2.1 Architecture générale 23

2.2 Carte IHM 24

2.3 ESP32-WROOM-32 broches 25

2.4 STM32G474VET6 broches 26

2.5 W25Q32JV broches 28

2.6 Carte de développement ESP32 29

2.7 Convertisseur USB vers TTL 30

2.8 ST-LINK 30

2.9 Clients et serveur 32

2.10 Mécanismes d'annonce et de découverte 33

2.11 Les principaux éléments de l'architecture GATT 34

2.12 Fonctionnement UART 36

2.13 Trame QSPI 36
2.14 Logo de GitHub 38
2.15 Logo de Docklight 38
2.16 Logo de Python 39
2.17 Logo de STM32CubeMX 39
2.18 Logo de IAR Embedded Workbench 40
2.19 Logo de STM32CubeProgrammer 40
2.20 Logo de HXD 41

3.1 Diagramme de séquence 45
3.2 Organigramme du fonctionnement du code de transfert des données via BLE 48
3.3 Organigramme du code de flashage des données vers la mémoire externe . . 49
3.4 Configuration des broches de STM32G474 dans STM32CubeMX 50
3.5 Configuration CubeMX USART1 51
3.6 Configuration CubeMX QSPI1 52
3.7 L'intégration du fichier external loader 53
3.8 Installation du firmware AT 53
3.9 Lancement du programme Python 54
3.10 Confirmation de connexion BLE 55
3.11 Changement de la taille du MTU 55
3.12 Réception des deux mots de passe 56
3.13 Réception de la taille du fichier 56
3.14 Réception des données par le microcontrôleur STM32G4 56
3.15 Réception des deux premières trames du fichier de mise à jour 57
3.16 Visualisation des deux premières paquets en hexadécimal 58
3.17 Chargement du buffer avec 256 octets 58
3.18 Réception du dernier paquet du fichier de mise à jour 59
3.19 Visualisation du dernier paquet en hexadécimal 59

3.20 Terminaison du transfert du fichier 59
3.21 Affichage des données initialement présentes dans la mémoire flash externe 60
3.22 Comparaison des trames du fichier d'origine et du fichier flashé 61
3.23 Version mise à jour 4.2.3 62
3.24 Version mise à jour 4.2.4 62
3.25 Intervalle de temps du transfert du fichier de mise à jour 62

Liste des tableaux

2.1 Caractéristiques principales du microcontrôleur STM32G474VET6 27
2.2 Description des broches du circuit W25Q32JV 28
2.3 Caractéristiques techniques du PC utilisé 29
2.4 Commandes AT+BLE 35
2.5 Structure de la trame QSPI 37

3.1 Configuration des broches pour la communication série ESP32–USB-TTL . 43
3.2 Configuration des broches pour la communication série ESP-WROOM-32 et STM32G4 43
3.3 Configuration des broches pour la communication entre STM32G474VET6 et W25Q32JV 44
3.4 Caractéristiques et descripteurs GATT du service BLE n°4 46

Liste des acronymes

BLE : Bluetooth Low Energy

UART : Universal Asynchronous Receiver Transmitter

SYSML : Systems Modeling Language

GAP : Generic Access Profile

GATT : Generic ATTribute Profile

UUID : Universally Unique Identifier

IHM : Interface Homme-Machine

I@T : Innov-Alliance-Tech

QSPI : Quad Serial Peripheral Interface

AT : Authenticated Transfer

MTU : Maximum Transmission Unit

CRC : Cyclic Redundancy Check

Introduction Génerale

Les systèmes embarqués jouent un rôle central dans l'industrie moderne. Ces dispositifs électroniques intégrés permettent de contrôler et d'automatiser des processus variés, améliorant ainsi la précision, la productivité et la flexibilité des machines industrielles. Parmi les composants clés utilisés dans ces systèmes, les servomoteurs occupent une place majeure. Ils assurent des mouvements précis dans les robots, les chaînes de production ou encore les équipements automatisés.

Avec l'évolution rapide des technologies, les servomoteurs sont devenus plus intelligents et connectés. La technologie Bluetooth Low Energy (BLE) permet désormais de les rendre plus innovants, offrant la possibilité de les contrôler et de les surveiller à distance. Cette évolution facilite notamment la mise à jour de leur logiciel interne évitant ainsi les interventions physiques.

Cependant, la mise à jour des servomoteurs via BLE reste un processus complexe, pouvant entraîner des interruptions dans la production. Dans ce contexte, notre projet au sein de la société I@T vise à développer un système embarqué permettant de réaliser cette mise à jour via BLE de façon rapide, simple et sécurisée, grâce à une communication sans fil optimisée.

Le rapport est structuré en trois chapitres :

Le premier chapitre présente le cadre du projet, en détaillant l'organisme d'accueil, le contexte industriel, la problématique à résoudre ainsi que la démarche méthodologique adoptée.

Le deuxième chapitre est consacré à l'étude technique de la solution. Il explore l'architecture du système, décrit les composants matériels et logiciels utilisés, et expose les protocoles de communication mis en œuvre.

Le troisième chapitre est dédié à la conception et à la mise en œuvre pratique du projet. Il présente les étapes de développement, les configurations réalisées, les tests effectués ainsi que les résultats obtenus.

Chapitre 1

Présentation générale du projet

Introduction

Ce chapitre présente une vue d'ensemble du projet. Il débute par une description de la société d'accueil, de ses services et de ses domaines d'activités. Ensuite, le cadre du projet est expliqué, incluant la problématique, la solution proposée, les objectifs, la méthodologie de travail ainsi que la présentation de la modélisation . Enfin, les besoins du système sont définis à travers un cahier des charges détaillé par des diagrammes SYSML.

1.1 Présentation de la societé

INNOV-ALLIANCE-TECH propose une gamme de services adaptés aux besoins spécifiques de ses clients, couvrant le conseil, l'ingénierie et les composants prêts à l'emploi. Chaque domaine permet de répondre efficacement aux exigences des projets, en offrant des solutions innovantes et adaptées.

1.1.1 Présentation de l'organisme d'accueil

INNOV-ALLIANCE-TECH est une entreprise tunisienne spécialisée dans le conseil en ingénierie et en technologies. Elle fait partie du groupe IPS, un acteur majeur du secteur

industriel en Afrique du Nord.

L'entreprise accompagne ses partenaires et clients dans la gestion de projets liés à la stratégie, au développement et à la validation de technologies, qu'il s'agisse de logiciels ou de produits innovants.

Grâce à son expertise et à sa réputation internationale, INNOV-ALLIANCE-TECH joue un rôle clé dans la recherche et le développement de solutions adaptées aux besoins industriels.

L'un de ses principaux objectifs est d'accroître son développement en intégrant de jeunes talents et en misant sur la formation continue. Le choix du nom de l'entreprise reflète son engagement en faveur de l'innovation et de l'excellence technologique dans le domaine de l'ingénierie [1].

Le logo de l'I@T est présenté dans la figure 1.1.

FIGURE 1.1 – Logo de l' I@T

1.1.2 Domaines d'activités

I@T opère dans plusieurs domaines stratégiques, en mettant son expertise au service de solutions technologiques innovantes :

- **Énergie :** I@T possède une expertise reconnue en Smart Metering, un domaine clé pour la gestion intelligente de l'énergie. Elle propose des solutions et des services d'architecture évolutifs et performants, adaptés aux besoins spécifiques de ses clients.
- **Télécommunications et Réseaux :** Grâce à un partenariat stratégique avec

AVAYA, I@T développe des solutions de communication unifiées. Cette collaboration permet d'offrir une large gamme de produits et services répondant aux exigences des entreprises en matière de connectivité.

- **Internet des objets (IoT) :** L'expertise des équipes I@T s'étend également au développement de capteurs intelligents et de logiciels dédiés à la collecte et au traitement des données, facilitant ainsi l'interconnexion des systèmes.
- **Biomédical :** I@T a conçu et développé une gamme variée d'équipements de laboratoire, comprenant des centrifugeuses, des agitateurs et des bains-marie. Grâce à ses solutions technologiques avancées, l'entreprise contribue à l'amélioration des performances et de la précision des dispositifs biomédicaux.
- **Automatisation industrielle :** I@T propose des solutions d'automatisation visant à réduire les coûts et les erreurs humaines, à accroître la productivité et à améliorer la fiabilité des processus. Ses technologies permettent d'optimiser les performances des systèmes de production et de renforcer leur uniformité.

1.1.3 Les services de la société

La société propose une variété de services, soigneusement adaptés pour répondre aux besoins spécifiques de chaque client. Ces services sont définis avec précision en fonction des attentes et des exigences des clients.

Service de conseil

I@T accompagne ses clients de manière personnalisée en analysant et en sélectionnant les solutions disponibles sur le marché qui répondent le mieux à leurs besoins. Ce service inclut une évaluation des technologies existantes, la sélection de la solution la plus appropriée et l'examen des normes applicables aux produits.

Ingénierie

Cette offre comprend plusieurs spécialités :

- **Systèmes embarqués :** Cela inclut les protocoles de communication, le développement de logiciels embarqués, ainsi que la conception de schémas et de cartes PCB.
- **Logiciel et outils sur PC :** Ce domaine couvre des services comme la conception graphique et la communication multimédia.
- **Logiciel Back-end et systèmes d'information :** L'accent est mis sur le stockage des données, la gestion de la mémoire et le développement d'interfaces web.
- **Tests et validation :** Ce service inclut l'étude de produits, l'analyse des normes et standards applicables, ainsi que la rédaction de tests fonctionnels.

Composants prêts à l'emploi

INNOV-ALLIANCE-TECH propose à ses clients et partenaires des piles de communication préconfigurées et adaptées à leurs besoins. Ces solutions incluent des options économiques avantageuses, telles que des stacks de communication DLMS COSEM, Buridis, M-Bus, Wireless M-Bus, et bien d'autres.

1.2 Cadre du projet

Dans cette partie, nous présentons le cadre général du projet. Nous commençons par décrire le contexte, puis nous exposons la problématique à résoudre. Ensuite, une solution technique est proposée pour répondre aux besoins identifiés. Enfin, les objectifs du projet et la méthode de travail adoptée sont expliqués, accompagnés d'une modélisation SysML pour mieux structurer le système.

1.2.1 Contexte de projet

Dans un environnement industriel en constante évolution, de nombreuses entreprises investissent dans l'amélioration de leurs servomoteurs électriques afin de répondre aux exigences croissantes du marché en termes de précision, de fiabilité et d'automatisation.

Un servomoteur est un dispositif combinant un moteur, un système d'engrenages et une électronique de commande. Il permet de contrôler précisément une position, même en présence de contraintes mécaniques. Ces dispositifs sont largement utilisés dans des domaines variés tels que l'automatisation industrielle, la régulation de vannes ou tout autre système nécessitant un positionnement fiable [2].

Comme illustré dans la figure 1.2, le servomoteur est monté sur une vanne permettant ainsi un contrôle précis de l'ouverture et de la fermeture du système en fonction des besoins de régulation.

FIGURE 1.2 – Servomoteur

Dans le cadre de ses activités, l'entreprise I@T a établi un partenariat stratégique avec une société leader mondiale dans la fabrication de servomoteurs, dans le but d'intégrer des solutions matérielles et logicielles performantes. Grâce à cette collaboration, I@T est en mesure de concevoir des systèmes embarqués modernes, adaptés aux environnements industriels.

Les servomoteurs intégrés dans ce projet sont contrôlés via une application mobile reposant sur la technologie Bluetooth Low Energy (BLE). Cette application permet à l'utilisateur de commander localement le servomoteur, de visualiser en temps réel les données de fonctionnement, d'effectuer un diagnostic rapide en cas d'anomalie, ainsi que de mettre à jour le firmware du dispositif de manière simple et sécurisée.

La figure 1.3 présente l'interface de l'application utilisée pour le contrôle des servomoteurs.

FIGURE 1.3 – Interface de l'application mobile

Toutefois, malgré ces fonctionnalités avancées, la solution actuelle présente encore certaines limites, notamment en ce qui concerne la mise à jour du firmware.

1.2.2 Problématique

Dans le domaine des technologies embarquées, les systèmes doivent répondre à des exigences strictes en matière de performance, de rapidité et de fiabilité. Ces critères sont encore plus importants lorsque le système interagit avec une application mobile, comme c'est le cas dans ce projet.

Les servomoteurs intègrent une carte IHM (Interface Homme-Machine) qui assure le contrôle et la communication avec une application Android via Bluetooth Low Energy (BLE). L'application permet à l'utilisateur de commander le servomoteur et de mettre à jour le firmware de cette carte.

Cependant, les retours des utilisateurs soulignent une lenteur importante lors de la mise à jour du firmware de la carte IHM, pouvant durer jusqu'à 30 minutes. Cette lenteur est due à la méthode de transmission utilisée, dans laquelle le fichier est envoyé sous forme de paquets de 20 octets, avec un intervalle d'une seconde entre chaque envoi.

La figure 1.4 illustre l'envoi du fichier de mise à jour.

L'objectif est de concevoir une solution plus efficace, capable de réduire fortement le temps de mise à jour tout en assurant la sécurité, l'intégrité des données et la compatibilité avec les ressources limitées du système embarqué.

FIGURE 1.4 – Transfert du firmware de mise à jour à travers l'application mobile

1.2.3 Solution proposée

Étant donné que nous ne disposons pas d'un accès à l'application mobile officielle utilisée pour contrôler les servomoteurs, nous proposons de développer un système autonome capable de reproduire son comportement lors de la mise à jour du firmware via Bluetooth Low Energy (BLE). Ce système repose sur un programme Python, exécuté en temps réel sur un ordinateur, qui communique avec le servomoteur à travers une carte ESP32.

Ce programme prend en charge l'ensemble du processus de mise à jour, notamment le découpage du fichier en paquets, la transmission des données, ainsi que la gestion des réponses envoyées par la carte IHM intégrée au servomoteur.

Pour optimiser le temps de transfert du fichier, plusieurs améliorations sont mises en œuvre :

- **Augmentation de la taille des paquets,** dans les limites permises par le protocole BLE et les dispositifs électroniques utilisés.
- **Réduction de l'intervalle entre les paquets,** afin d'accélérer le transfert tout en respectant les contraintes de stabilité de la communication.

En parallèle, une amélioration du processus de flash mémoire sera étudiée. Il s'agira d'examiner des méthodes plus rapides et plus efficaces pour écrire les données reçues, réduisant ainsi les temps d'attente liés aux opérations d'effacement et de programmation.

1.2.4 Objectif du projet

L'objectif principal de ce projet est de concevoir et de développer un système autonome de mise à jour du firmware d'un servomoteur via une communication Bluetooth Low Energy (BLE), en assurant une amélioration en termes de rapidité, de fiabilité et de sécurité du processus.

Pour atteindre cet objectif, plusieurs sous-objectifs techniques sont à définir :

- **Conception d'une architecture matérielle et logicielle adaptée :** permettre l'envoi, la réception, le stockage et la gestion efficace des fichiers de mise à jour via BLE, en s'appuyant sur une carte ESP32.
- **Mise en place d'une communication BLE optimisée :** garantir un transfert rapide et stable entre le système embarqué à développer et la carte IHM du servomoteur.
- **Implémentation d'un mécanisme sécurisé de mise à jour :** assurer l'intégrité du firmware face aux erreurs pendant le processus de mise à jour.
- **Optimisation des performances :** réduire la durée totale du transfert et du flashage du firmware, afin d'accélérer le processus.
- **Automatisation du processus :** développer un programme permettant de piloter la mise à jour de manière entièrement autonome et supervisée depuis un ordinateur.

- **Validation et tests :** vérifier la fiabilité et la performance du système dans des conditions réelles d'utilisation.

1.2.5 Méthodologie de travail

Pour encadrer le développement de ce projet, une méthodologie rigoureuse a été mise en place, conformément aux recommandations des encadrants de l'I@T. Le cycle de vie choisi est celui en V comme présenté dans la figure 1.5, reconnu pour sa clarté, son organisation méthodique, ainsi que pour sa capacité à assurer une bonne traçabilité entre les phases de conception et de tests [3].

Ce modèle suit une logique descendante lors de la phase de conception, puis ascendante lors de la phase de validation, associant à chaque étape de développement une phase de vérification correspondante.

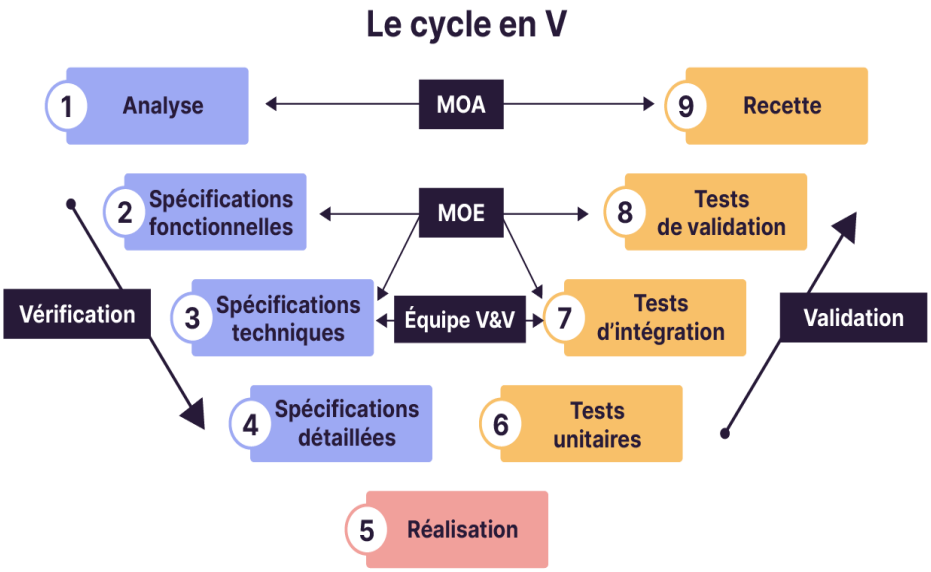

FIGURE 1.5 – Cycle en V

1.2.6 Modélisation SYSML

La modélisation Systems Modeling Language (SYSML), est un langage graphique utilisé pour représenter de manière claire et organisée les systèmes complexes. Il aide à

décrire la structure, le comportement, ainsi que les exigences d'un système, ce qui facilite la compréhension et la collaboration entre les membres de l'équipe.

Dans ce projet, le langage SysML a été employé pour modéliser les différents aspects du système de manière claire et structurée. Trois types de diagrammes ont été particulièrement utilisés :

Un diagramme d'exigences : permet de définir de façon précise les besoins fonctionnels et non fonctionnels du système.

Un diagramme de cas d'utilisation : illustre les interactions entre le système et les différents acteurs externes.

Un diagramme de séquence : représente l'enchaînement des messages échangés entre les composants du système au cours d'un scénario donné.

1.3 Cahier des charges

Cette section vise à définir clairement ce que le système doit accomplir et dans quelles conditions il doit fonctionner. Pour y parvenir, nous allons identifier les exigences fonctionnelles, c'est-à-dire les actions que le système doit être capable de réaliser, ainsi que les exigences non fonctionnelles, qui couvrent des aspects tels que la performance, la fiabilité ou encore la facilité d'utilisation. Afin de mieux structurer et représenter ces besoins, nous allons également présenter un diagramme d'exigences qui permet de visualiser les différentes contraintes et attentes liées au projet. De plus, un diagramme de cas d'utilisation sera fourni pour illustrer les principales interactions entre les utilisateurs et le système envisagé.

1.3.1 Spécification des besoins

Dans le but de préciser les besoins des utilisateurs de notre système, nous présentons les besoins fonctionnels ainsi que les besoins non fonctionnels.

Spécification des besoins non fonctionnels

Pour assurer le bon fonctionnement du processus de mise à jour via Bluetooth Low Energy, tout en garantissant une expérience utilisateur satisfaisante, le système embarqué doit respecter plusieurs exigences non fonctionnelles. Ces exigences visent à renforcer la fiabilité et la performance globale du système, assurant une mise à jour fluide, sécurisée et adaptée aux contraintes industrielles.

- ☞ **Temps de mise à jour réduit**

  Le système doit être conçu pour effectuer le transfert du fichier de mise à jour dans un délai réduit, permettant une amélioration d'au moins 30 à 50% par rapport aux méthodes de mise à jour précédentes.

- ☞ **Fiabilité du système**

  Le système doit garantir un taux de réussite élevé lors du transfert des mises à jour, avec un objectif de fiabilité supérieur ou égal à 90%.

- ☞ **Simplicité d'intégration**

  Le module de mise à jour doit pouvoir être intégré au système existant par une simple modification logicielle, sans nécessiter de changements matériels côté servomoteur. Cela permet une mise en œuvre rapide et garantit une bonne compatibilité avec l'architecture existante.

Spécification des besoins fonctionnels

Le système embarqué doit satisfaire plusieurs fonctions essentielles pour assurer la mise à jour du firmware du servomoteur via BLE. Ces fonctions sont décrites ci-dessous.

- ☞ **Connexion Bluetooth Low Energy**

  La carte ESP32 doit établir une connexion sans fil stable et sécurisée avec la carte IHM grâce à la technologie Bluetooth Low Energy. Cette connexion constitue la base de toute la procédure de mise à jour puisqu'elle permet un échange d'informations bidirectionnel entre les deux cartes. Le fichier de mise à jour est transmis à la carte

IHM, tandis que des messages d'état sont renvoyés à la carte ESP32. La stabilité et la sécurité de cette connexion sont essentielles pour éviter toute interruption du processus.

- ☞ **Transmission du fichier de mise à jour**

  Le transfert du fichier de mise à jour s'effectue par l'envoi successif de paquets via la connexion BLE établie entre les deux cartes. Ce transfert doit être réalisé de manière fiable et ordonnée afin d'éviter toute perte de données. En cas d'erreur détectée sur un paquet, le transfert est interrompu.

- ☞ **Réception du fichier de mise à jour**

  La carte IHM doit être capable de recevoir le fichier contenant le nouveau firmware à installer. Ce fichier peut varier en taille ou en format selon les versions logicielles du servomoteur. Le transfert doit donc être suffisamment flexible pour gérer ces différentes versions, tout en garantissant que l'intégralité du fichier est correctement reçue.

- ☞ **Gestion et vérification de la mise à jour du firmware**

  Après la phase de réception, la carte IHM doit assurer un traitement efficace du fichier. Une fois le transfert terminé, le firmware doit être correctement flashé dans la mémoire de la carte. Le système doit ensuite vérifier l'intégrité de l'opération et garantir le bon fonctionnement du firmware.

1.3.2 Diagramme d'exigences

Après avoir étudié attentivement les besoins de notre système, nous avons effectué sa modélisation à travers le diagramme d'exigence permettant de décrire graphiquement les capacités à réaliser et les contraintes qui doivent être dépassées.

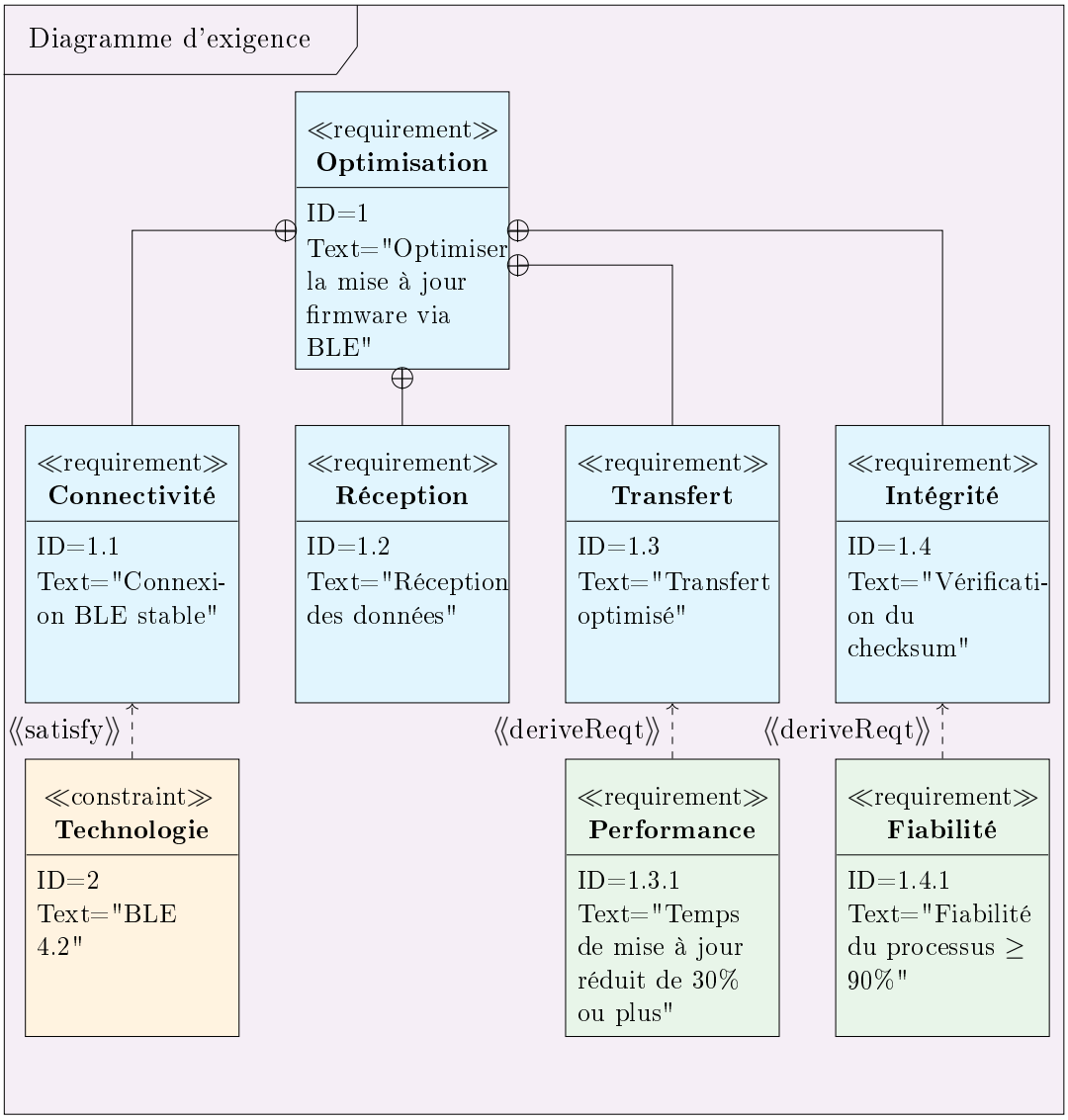

FIGURE 1.6 – Diagramme d'exigences

Comme illustré dans le diagramme de la figure 1.6, plusieurs exigences ont été définies à partir de cette exigence principale, telles que la connectivité stable ID=1.1, la réception des données ID=1.2, le transfert optimisé ID=1.3 et la vérification de l'intégrité des données via un checksum ID=1.4. Des exigences dérivées ont également été ajoutées afin de quantifier les objectifs attendus, comme la réduction du temps de mise à jour d'au moins 30 % ID=1.3.1 et une fiabilité du processus supérieure ou égale à 90 % ID=1.4.1. Par ailleurs, une contrainte technologique ID=2, spécifiant l'utilisation du BLE 4.2, a été définie pour satisfaire l'exigence de connectivité. Ce diagramme permet ainsi de poser les bases d'un cahier des charges formel, tout en assurant la

traçabilité et la cohérence des besoins tout le long du développement.

1.3.3 Diagramme de cas d'utilisation

Cette structure est bien adaptée pour décrire de manière claire les fonctions et les objectifs les plus importants d'un système. C'est pour cette raison que l'élaboration d'un diagramme de cas d'utilisation est souvent l'une des premières étapes lors de la conception de système ou de la planification de nouveaux processus métier. Cela permet de visualiser facilement et clairement quels cas d'utilisation doivent être pris en compte dans la conception pour que les acteurs atteignent les objectifs escomptés.

Nous décrivons en tant qu'utilisateurs ce diagramme présenté dans la figure 1.7 qui donne un aperçu global sur le rôle de chaque acteur ainsi de comprendre l'idée générale du projet.

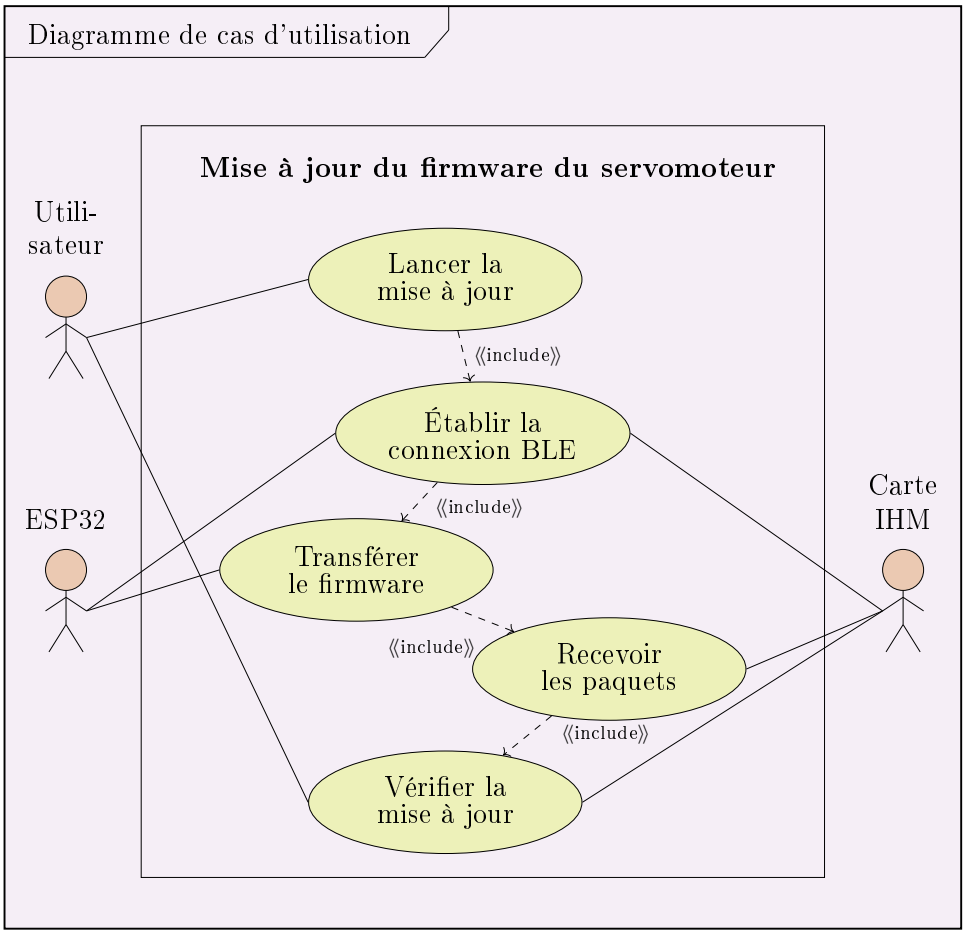

FIGURE 1.7 – Diagramme de cas d'utilisation

Présentation des acteurs

Dans le cadre du système de mise à jour du firmware via BLE, plusieurs acteurs interagissent pour assurer le bon déroulement des différentes fonctionnalités. Ces acteurs sont décrits comme suit :

Utilisateur : Personne chargée de lancer la mise à jour du firmware depuis un ordinateur. Ce rôle consiste à démarrer le processus de transfert du fichier de mise à jour vers le système.

ESP32 : La carte ESP32 établit une connexion BLE avec la carte IHM dès le démarrage du programme. Elle assure ensuite l'envoi du fichier de mise à jour sous forme de paquets.

Carte IHM : La carte IHM reçoit ces paquets via la communication BLE et participe à la vérification finale garantissant la bonne réalisation de la mise à jour du firmware.

Description des cas d'utilisation

Dans cette partie, nous détaillons chaque cas d'utilisation en identifiant les acteurs impliqués et en décrivant leur rôle.

- **Premier cas d'utilisation :** Lancer la mise à jour
  - Acteur :

    Utilisateur
  - Description :

    Démarrer le processus de mise à jour du firmware en lançant le transfert du fichier depuis l'ordinateur.
  - Précondition :

    L'ordinateur est prêt et le fichier de mise à jour est disponible.
  - Scénario principal :

    L'utilisateur active le transfert du fichier de mise à jour.

- **Deuxième cas d'utilisation :** Établir la connexion BLE
  - Acteurs :

    ESP32, Carte IHM
  - Description :

    Établir une connexion BLE entre la carte ESP32 et la carte IHM pour permettre le transfert des données.
  - Précondition :

    Les deux cartes sont sous tension et le BLE est activé.
  - Scénario principal :

    La carte ESP32 initie la connexion BLE.

    La carte IHM accepte la connexion.

    La connexion BLE est établie.

- **Troisième cas d'utilisation :** Transférer le firmware
  - Acteur :

    ESP32
  - Description :

    La carte ESP32 transfère le fichier de mise à jour du firmware divisé en paquets, vers la carte IHM via une connexion BLE.
  - Précondition :

    Une connexion BLE active et stable est établie entre la carte ESP32 et la carte IHM.
  - Scénario principal :

    La carte ESP32 envoie successivement les paquets du fichier de mise à jour à la carte IHM.

    En cas de perte d'un paquet, le transfert est considéré comme échoué.

- **Quatrième cas d'utilisation :** Recevoir les paquets
  - Acteur :

    Carte IHM
  - Description :

    La carte IHM reçoit les paquets du fichier de mise à jour transmis par la carte ESP32 via BLE.
  - Précondition :

    Une connexion BLE active et stable est établie entre la carte ESP32 et la carte IHM.
  - Scénario principal :

    La carte IHM reçoit chaque paquet envoyé par la carte ESP32.

    La carte IHM stocke les paquets reçus dans la mémoire.

- **Cinquième cas d'utilisation :** Valider la mise à jour
  - Acteurs :

    Carte IHM, Utilisateur
  - Description :

    La carte IHM calcule la valeur CRC du fichier de mise à jour reçu afin de vérifier son intégrité.
  - Précondition :

    Le fichier de mise à jour a été entièrement reçu et stocké dans la mémoire de la carte IHM.
  - Scénario principal :

    La carte IHM lit le fichier de mise à jour stocké en mémoire.

    Elle calcule la valeur CRC correspondante pour en vérifier l'intégrité.

    L'utilisateur visualise la version de la mise à jour affichée sur l'interface de l'IHM.

Conclusion

Ce chapitre nous a permis de mieux comprendre le cadre dans lequel s'inscrit notre projet, à travers une présentation de la société et de ses activités. Nous avons également clarifié les attentes et les contraintes du projet en définissant un cahier des charges précis. Cette première étape pose les bases nécessaires pour poursuivre l'étude du système dans le chapitre suivant.

Chapitre 2

ÉTUDE DE LA SOLUTION TECHNIQUE DU PROJET

Introduction

Dans ce chapitre, nous présentons l'architecture générale du système, en détaillant les principaux composants, les protocoles utilisés ainsi que les outils logiciels mis en œuvre pour assurer le bon déroulement du projet. Ces éléments sont essentiels pour garantir une intégration cohérente et un fonctionnement fiable.

2.1 Architecture générale du système

Cette section offre une vue d'ensemble du système en présentant ses principales parties et leur organisation. Elle décrit son fonctionnement global, en mettant l'accent sur la présentation générale et la description fonctionnelle. Ce cadre facilite la compréhension globale avant d'aborder les aspects techniques plus détaillés.

2.1.1 Présentation générale du système

Ce projet vise à développer un système embarqué capable de mettre à jour à distance le firmware d'un servomoteur grâce à une communication Bluetooth Low Energy (BLE). Cette approche évite toute intervention physique directe sur le servomoteur, ce qui est particulièrement avantageux dans des environnements industriels automatisés.

Le système repose sur l'intégration de trois éléments principaux :

- ☞ **Utilisateur** : lancer et superviser le processus de mise à jour.
- ☞ **Carte ESP32** : assurer la communication sans fil et le transfert du firmware.
- ☞ **Carte IHM** : gérer la réception et la validation du firmware.

2.1.2 Description fonctionnelle globale

Le processus de mise à jour s'appuie sur l'interconnexion des modules décrits dans la sous-section 2.1. Une communication série est d'abord établie entre le PC et la carte ESP32, permettant le transfert progressif du firmware sous forme de paquets. Ces données sont ensuite envoyées via Bluetooth Low Energy (BLE) à la carte IHM, qui se charge de leur vérification en appliquant un calcul CRC.

Le programme exécuté sur le PC découpe le fichier binaire en segments de 256 octets, envoyés à intervalles réguliers via la liaison UART. La carte ESP32, configurée comme client BLE, réceptionne ces paquets et les transfère immédiatement à la carte IHM, qui agit comme serveur BLE.

Cette dernière vérifie l'intégrité des données reçues et stocke temporairement les informations. Une fois la transmission complète et validée, elle procède à l'écriture du firmware dans la mémoire cible.

Tout au long de ce processus, l'utilisateur peut superviser l'état de la mise à jour grâce aux messages affichés dans le terminal, facilitant ainsi la détection rapide des erreurs de communication.

La figure 2.1 illustre l'architecture générale du système.

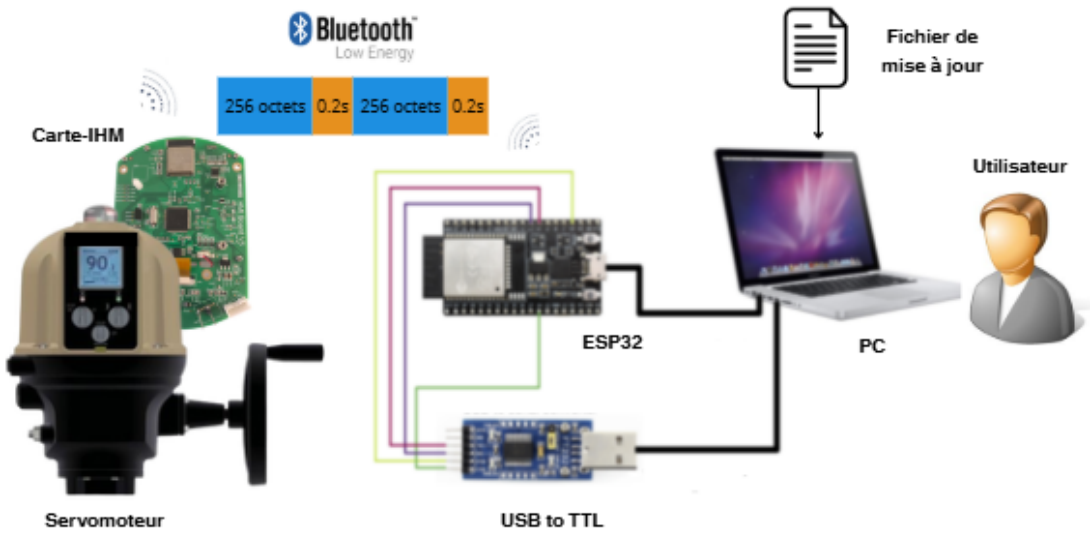

FIGURE 2.1 – Architecture générale

2.2 Étude matérielle

Cette section décrit les principaux composants matériels intégrés dans notre projet, en détaillant leur rôle et leurs caractéristiques techniques.

2.2.1 Carte IHM du servomoteur

La carte Interface Homme-Machine (IHM), présentée dans la figure 2.2, constitue un élément central du système. Conçue et fabriquée par la société productrice des servomoteurs, elle est intégrée directement dans chaque servomoteur et assure de manière autonome les fonctions de contrôle, de surveillance et de communication.

Cette carte regroupe plusieurs composants électroniques. Elle intègre deux microcontrôleurs, un écran LCD pour l'affichage, une mémoire flash externe, des voyants LED indiquant l'état du système, des boutons de commande, des jumpers de configuration, ainsi que des circuits de gestion de l'alimentation. Tous ces éléments sont conçus pour fonctionner de façon robuste dans un environnement industriel.

Dans le cadre du projet, la carte IHM joue un rôle clé dans le processus de mise à jour du firmware. Elle reçoit les données via le module ESP32-WROOM-32, basé sur la

technologie Bluetooth Low Energy. Ce module établit la communication sans fil, reçoit les données, puis les transmet au microcontrôleur principal STM32G4. Ce dernier traite les informations reçues, vérifie leur intégrité et les enregistre dans la mémoire flash externe W25Q32JV. L'ensemble assure une mise à jour fiable et sécurisée.

L'étude se concentre principalement sur trois composants fondamentaux présents sur cette carte : le module ESP32-WROOM-32, le microcontrôleur STM32G4 et la mémoire W25Q32JV. Les sous-sections suivantes détaillent leurs caractéristiques et leur rôle respectif dans l'architecture matérielle du système.

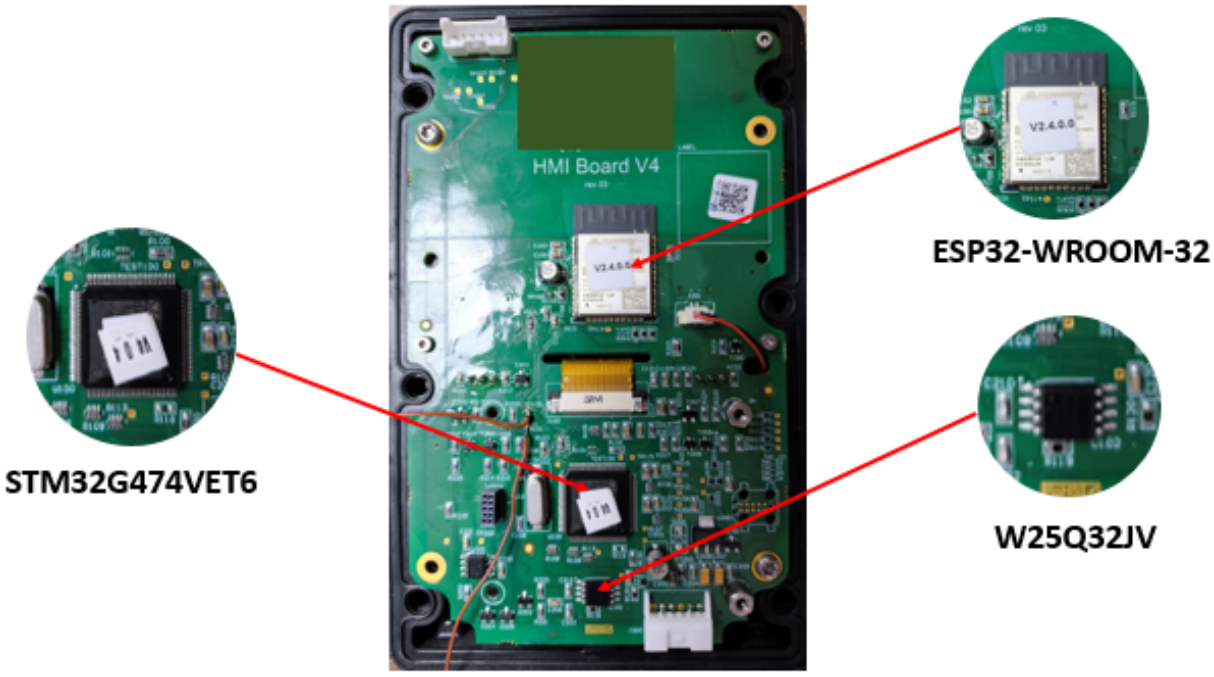

FIGURE 2.2 – Carte IHM

2.2.2 Module esp32 wroom 32

Les premières versions des cartes IHM utilisaient des microcontrôleurs de la société Microchip. Toutefois, en raison de la difficulté d'approvisionnement de ces composants, le module ESP32-WROOM-32 s'est progressivement imposé comme une alternative pertinente. Ce module compact, conçu par Espressif Systems, propose des performances

élevées grâce à son processeur double cœur capable d'atteindre une fréquence de 240 MHz. Il intègre également une connectivité sans fil complète, incluant le Wi-Fi et le Bluetooth Low Energy. Facile à intégrer aussi bien sur le plan matériel que logiciel, l'ESP32-WROOM-32 offre une solution fiable et économique, parfaitement adaptée aux exigences actuelles des interfaces homme-machine.

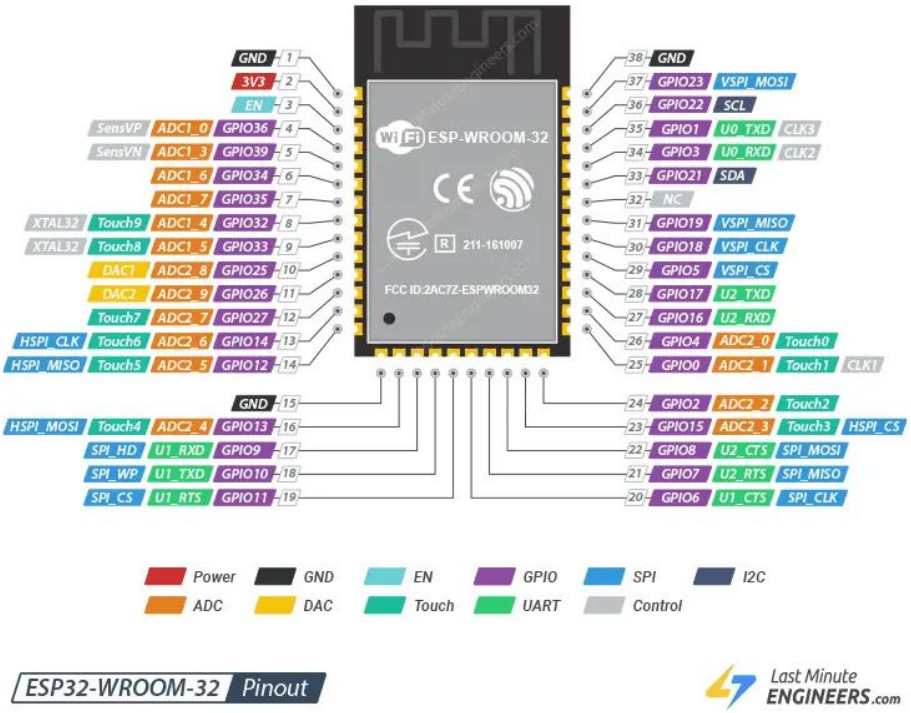

FIGURE 2.3 – ESP32-WROOM-32 broches

Le brochage de l'ESP32-WROOM-32, illustré dans la figure2.3, met en évidence la diversité de ses fonctionnalités. Il comprend des broches dédiées à l'alimentation, aux communications série avec d'autres périphériques (SPI, I²C, UART), ainsi qu'un ensemble de broches d'entrées/sorties numériques (GPIO). Le module propose également des interfaces analogiques, avec des broches pour la conversion analogique-numérique (ADC), la conversion numérique-analogique (DAC), et des entrées tactiles capables de détecter un contact capacitif [4]. Grâce à cette richesse fonctionnelle, l'ESP32-WROOM-32 s'intègre facilement dans les applications embarquées.

2.2.3 Microcontrôleur STM32G474VET6

Le microcontrôleur STM32G474VET6 est un choix puissant et performant pour la gestion de servomoteurs, grâce à son cœur ARM Cortex-M4 cadencé à 170 MHz, ses timers avancés, et sa capacité à générer des signaux PWM haute résolution (HRPWM). Par rapport à d'autres microcontrôleurs, il se distingue par l'intégration de périphériques analogiques comme les amplificateurs opérationnels, les comparateurs, et les convertisseurs ADC/DAC de haute précision, ce qui le rend idéal pour des applications de commande moteur exigeantes. Sa faible consommation, combinée à des performances élevées permettent une régulation précise et efficace du servomoteur, tout en optimisant les ressources du système. Ce microcontrôleur STM32G474VET6 dispose d'un grand nombre de broches aux fonctions multiples. La figure 2.4 met en évidence la répartition fonctionnelle de ses broches [5].

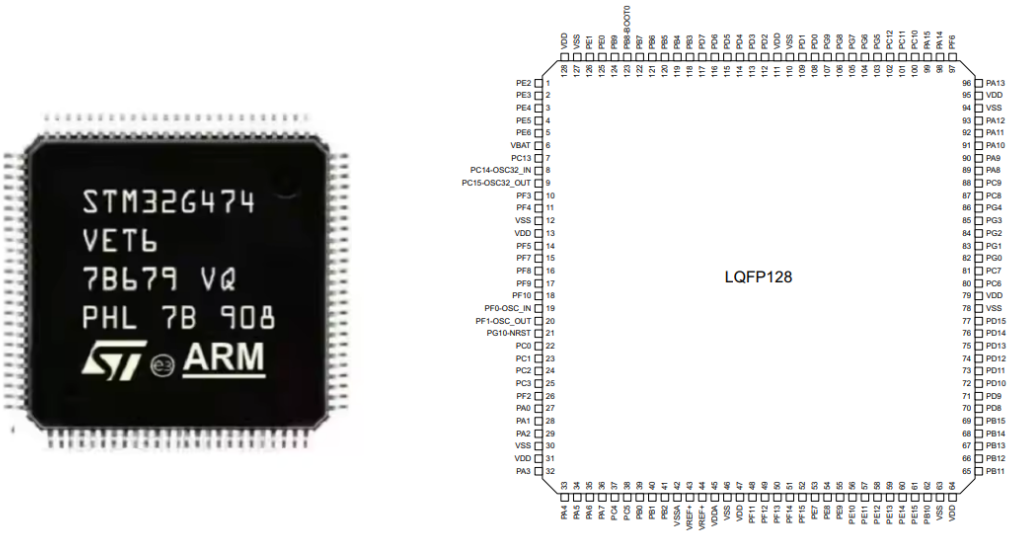

FIGURE 2.4 – STM32G474VET6 broches

Le tableau 2.1 présente les principales caractéristiques techniques du microcontrôleur afin d'en fournir une vue d'ensemble plus complète.

TABLE 2.1 – Caractéristiques principales du microcontrôleur STM32G474VET6

Attribut	**Valeur**
Fabricant	STMicroelectronics
Série	STM32G4
Nom commercial	STM32
Catégorie du produit	Microcontrôleurs ARM
Type de produit	MCU + FPU
Cœur	ARM Cortex-M4 avec FPU
Fréquence maximale	170 MHz
Largeur du bus de données	32 bits
Mémoire programme (Flash)	512 kB
RAM (SRAM)	128 kB
Type de RAM	SRAM
Nombre d'E/S numériques	107
Résolution CAN (ADC)	5 x 12 bits
Nombre de canaux ADC	42
Watchdog intégré	Oui

2.2.4 Mémoire flash W25Q32JV

La mémoire W25Q32JV, intégrée dans la carte IHM offre une bonne capacité de stockage de 4 Mo, adaptée aux besoins de notre application. Sa compatibilité avec le protocole SPI, ainsi que les modes Dual et Quad SPI permettent d'échanger les données rapidement avec le microcontrôleur, ce qui améliore les performances globales du système.

De plus, cette mémoire consomme peu d'énergie, un point important pour la stabilité et l'efficacité de notre dispositif embarqué. Sa fiabilité et sa large utilisation dans l'industrie facilitent également son intégration et garantissent un fonctionnement stable sur le long terme.

La figure 2.5 présente le brochage du circuit.

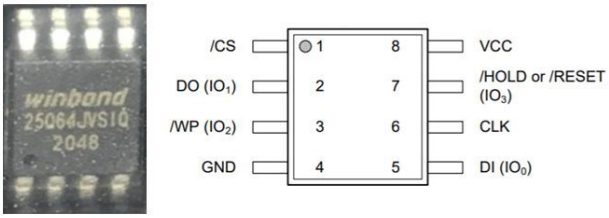

FIGURE 2.5 – W25Q32JV broches

Dans ce qui suit, le tableau 2.2 présente les caractéristiques techniques de ce composant.

TABLE 2.2 – Description des broches du circuit W25Q32JV

Broche N°	**Nom de broche**	**I/O**	**Fonction**
1	CS	E	Sélection de puce
2	DO (IO1)	E/S	Entrée/Sortie de données 1
3	IO2	E/S	Entrée/Sortie de données 2
4	GND		Masse
5	DI (IO0)	E/S	Entrée/Sortie de données 0
6	CLK	E	Horloge série
7	IO3	E/S	Entrée/Sortie de données 3
8	VCC		Alimentation

2.2.5 Carte de développement ESP32

Dans le cadre de l'établissement d'une connexion BLE avec la carte IHM, le choix de la carte de développement ESP32, présentée dans la figure 2.6, s'est porté sur un modèle intégrant le même module ESP32-WROOM-32 que celui utilisé dans la carte IHM. Cette cohérence matérielle garantit une communication fiable et fluide entre les deux cartes. Outre des caractéristiques techniques similaires, la carte de développement comprend également un port USB, un régulateur de tension et un convertisseur USB-TTL, facilitant ainsi les étapes de programmation, de test et de débogage, tout en assurant une intégration simple et directe dans notre solution.

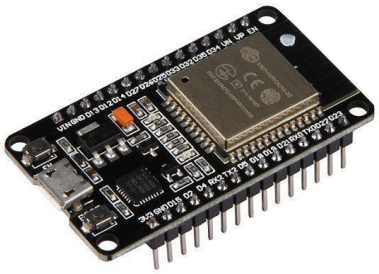

FIGURE 2.6 – Carte de développement ESP32

2.2.6 PC

Le développement et les tests de la solution ont été réalisés sur un ordinateur portable dont les principales caractéristiques techniques sont présentées dans le tableau 2.3.

TABLE 2.3 – Caractéristiques techniques du PC utilisé

Élément	**Caractéristique**
Processeur	Intel Core i3-1005G1
Mémoire RAM	8 Go
Stockage	1 To HDD
Système d'exploitation	Windows 10

2.2.7 Convertisseur USB vers TTL

Pour assurer une communication facile entre le module ESP32 et un ordinateur, nous avons choisi d'utiliser un convertisseur USB vers TTL, présenté dans la figure 2.7. Ce choix permet de convertir les signaux USB utilisés par l'ordinateur, en signaux série TTL compatibles avec le microcontrôleur. Grâce à ce convertisseur, il est possible de programmer le module, d'échanger des données, mais aussi de visualiser en temps réel les informations transmises via l'interface UART à l'aide d'un logiciel de terminal série.

FIGURE 2.7 – Convertisseur USB vers TTL

2.2.8 STLINK

Pour pouvoir programmer et tester le microcontrôleur STM32G4, nous utilisons le module ST-LINK illustré dans la figure 2.8. Cet outil permet de transférer facilement le code vers la carte, de faire du débogage en temps réel, et d'observer le comportement du système pendant son exécution. Il constitue donc un lien essentiel entre l'environnement de développement et le microcontrôleur.

FIGURE 2.8 – ST-LINK

2.3 Étude protocolaire

Nous abordons ici les différents protocoles de communication mis en œuvre dans notre système. Ces protocoles assurent un rôle essentiel dans la transmission des données entre les différents modules matériels et dans la connexion avec des dispositifs externes.

2.3.1 Bluetooth Low Energy

Le Bluetooth est une technologie de communication sans fil qui permet l'échange bidirectionnel de données sur de courtes distances. Il fonctionne grâce à des ondes radio dans la bande de fréquence UHF, plus précisément autour de 2,4 GHz, ce qui le rend adapté à une large gamme d'applications personnelles et industrielles.

Le Bluetooth Low Energy (BLE), également appelé Bluetooth Smart, constitue une évolution du Bluetooth classique. Il a été intégré pour la première fois dans la norme Bluetooth à partir de la version 4.0. Conçu initialement par Nokia sous le nom de code "Wibree", le BLE a ensuite été adopté par le Bluetooth Special Interest Group (SIG). Ce protocole se distingue par sa faible consommation d'énergie, ce qui le rend particulièrement adapté aux dispositifs portables et aux objets connectés. Nous présentons une vue d'ensemble du fonctionnement du BLE, en expliquant la structure des données échangées, le mécanisme d'annonce de présence, ainsi que le processus de création de connexion [6].

Generic Access Profile (GAP)

Le Generic Access Profile (GAP) constitue la base des communications Bluetooth, définissant les principes essentiels de découverte, connexion et sécurité entre appareils. Il établit une architecture à deux rôles complémentaires :

- **Les appareils périphériques :** Ces dispositifs à faible consommation énergétique (capteurs, écouteurs, wearables) adoptent une approche passive pour la connexion. Ils émettent périodiquement des paquets d'annonce contenant leurs informations d'identification et capacités. Optimisés pour une autonomie maximale, ils disposent de ressources matérielles limitées et ne peuvent généralement maintenir qu'une seule connexion à la fois.
- **Les appareils centraux :** Représentés par des appareils plus puissants (smartphones, tablettes, ordinateurs), ils jouent un rôle actif dans l'établissement des connexions. Dotés de capacités de traitement supérieures, ils scannent en

permanence leur environnement pour détecter les annonces des périphériques, puis initient et gèrent les connexions. Un central peut simultanément interagir avec plusieurs périphériques.

La figure 2.9 présente un serveur connecté à quatre clients différents.

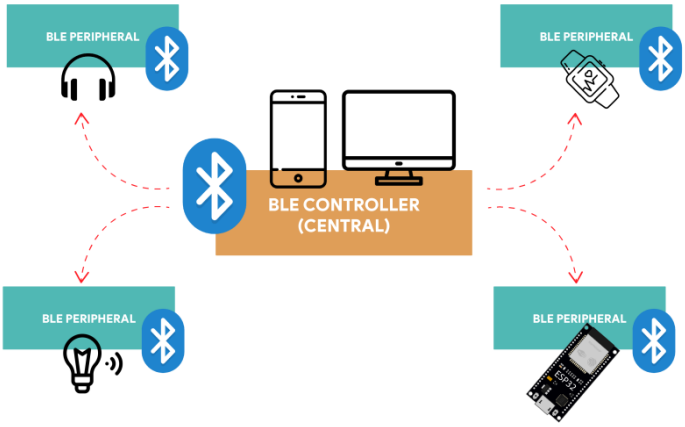

FIGURE 2.9 – Clients et serveur

Ces rôles central et périphérique interagissent principalement à travers deux mécanismes fondamentaux, qui constituent la base du processus d'annonce et de découverte. Ces méthodes sont décrites en détail dans la partie suivante .

Mécanismes d'annonce et de découverte

Dans le cadre du Bluetooth Low Energy, le processus de découverte repose sur deux mécanismes complémentaires : l'advertising émission d'annonces et le scanning écoute de ces annonces. Ces mécanismes, présentés dans la figure 2.10 sont directement liés aux rôles définis par le GAP.

- **Advertising :** Cette méthode est utilisée par les appareils périphériques. Elle consiste à envoyer périodiquement des paquets d'annonce sur des canaux spécifiques généralement trois canaux d'advertising dans la bande des 2,4 GHz. Ces paquets contiennent des informations essentielles telles que l'identifiant du périphérique, son nom, et parfois des données de service. Le but est de rendre l'appareil visible et identifiable par les dispositifs centraux.

- **Scanning :** Les appareils centraux utilisent cette méthode pour détecter la présence de périphériques à proximité. En mode scanning, le central écoute les canaux d'advertising afin de recevoir les paquets transmis. Lorsqu'un paquet pertinent est détecté, le central peut choisir de répondre via une requête de connexion ou simplement de collecter les informations disponibles pour une utilisation ultérieure.

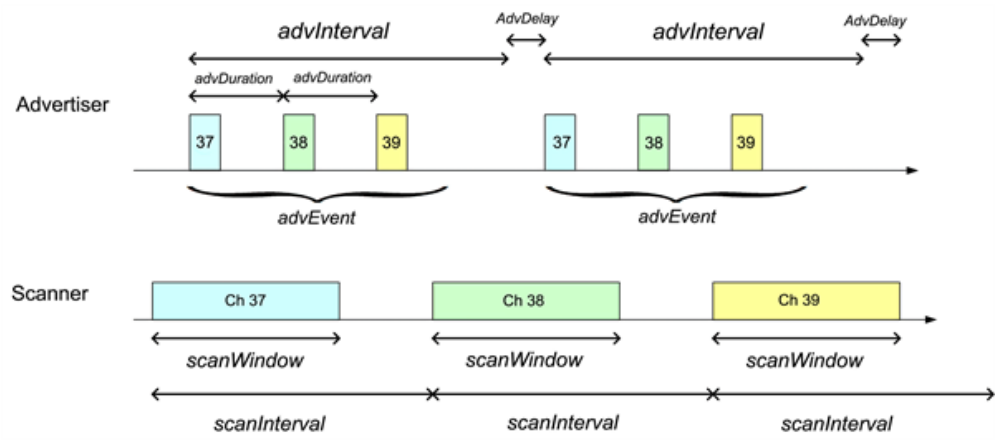

FIGURE 2.10 – Mécanismes d'annonce et de découverte

Generic Attribute Profile (GATT)

Une fois la connexion établie entre un appareil central et un périphérique Bluetooth Low Energy, le modèle de communication utilisé est défini par le GATT. Ce profil organise la manière dont les données sont structurées et échangées, en s'appuyant sur une relation client-serveur.

Dans ce cadre, l'appareil central agit généralement comme client, tandis que le périphérique joue le rôle de serveur. Le serveur contient une base de données d'attributs comprenant des services et des caractéristiques. Chaque service regroupe un ensemble de caractéristiques liées à une fonction particulière du périphérique [7].

Les éléments principaux de cette architecture, illustrés dans la figure 2.11 sont :

- **Services :** Ils définissent une fonctionnalité spécifique et regroupent les caractéristiques associées.

- **Caractéristiques :** Ce sont les unités de données de base. Chaque caractéristique inclut une valeur et peut être associée à des descripteurs pour en préciser les propriétés.
- **Descripteurs :** Ils ajoutent des informations supplémentaires aux caractéristiques, comme la façon dont les données sont présentées ou leur unité de mesure.

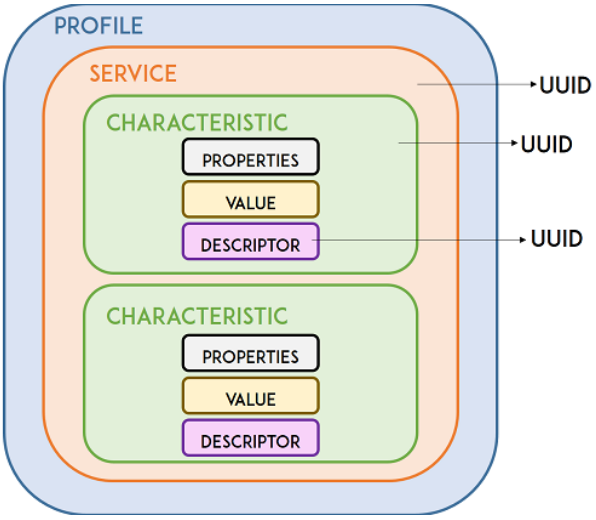

FIGURE 2.11 – Les principaux éléments de l'architecture GATT

Le transfert d'informations repose sur le protocole AT, qui permet au client d'accéder aux données via des opérations standards telles que la lecture `Read`, l'écriture `Write` ou la notification `Notify`.

2.3.2 Protocole AT

Dans notre projet, la communication BLE repose sur l'utilisation du protocole AT pour piloter le module Bluetooth. Les commandes AT permettent d'établir une connexion, d'échanger des données, et de configurer les paramètres de la liaison sans fil. Ce protocole simplifie l'interaction avec le module Bluetooth en utilisant des instructions textuelles standardisées, ce qui facilite le développement et le débogage de la communication entre le système embarqué et la carte IHM. Le tableau suivant illustre quelques commandes AT+BLE [8].

Table 2.4 – Commandes AT+BLE

Commande AT	**Description**
AT+RST	Réinitialisation du module Bluetooth
AT+BLEINIT	Initialisation du Bluetooth Low Energy
AT+BLEGATTSSRVSTART	Démarrer les services GATTS
AT+BLECONN	Établir une connexion BLE
AT+BLECFGMTU	Définir la taille MTU
AT+BLEGATTSNTFY	Notifier une caractéristique
AT+BLEGATTSSETATTR	Définir un attribut GATTS
AT+BLEGATTCPRIMSRV	Découvrir les services principaux GATTC
AT+BLEGATTCCHAR	Découvrir les caractéristiques GATTC
AT+BLEGATTCWR	Écrire une caractéristique GATTC
AT+BLEENC	Démarrer un chiffrement BLE
AT+BLEKEYREPLY	Répondre avec la clé
AT+BLECONFREPLY	Répondre avec la valeur de confirmation

2.3.3 Protocole de communication UART

Le projet comporte deux communications UART. La première s'établit entre le PC et la carte ESP32 à travers convertisseur USB-TTL pour l'exécution du programme de mise à jour, tandis que la deuxième s'effectue entre deux composants de la carte IHM, le module ESP32-WROOM-32 et le microcontrôleur STM32G4. Après la réception de chaque paquet, le module transfère les données au microcontrôleur STM32 via l'interface UART. Ce protocole est un choix adapté pour ce type d'échange car il est facile à configurer, ne demande que deux fils TX pour la transmission et RX pour la réception, et fonctionne sans signal d'horloge partagé. Ce protocole transmet les données bit par bit, avec un bit de début pour signaler le début de la trame, suivi des bits de données, éventuellement d'un bit de parité pour vérifier les erreurs, et enfin un ou plusieurs bits de fin. Grâce à cette structure simple, la communication reste rapide et fiable, ce qui est essentiel pour notre projet [9].

La figure met en évidence le fonctionnement du protocole UART.

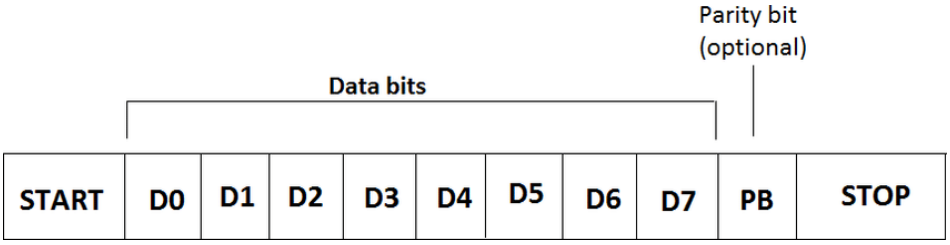

FIGURE 2.12 – Fonctionnement UART

2.3.4 Protocole de communication QSPI

Pour pouvoir flasher rapidement les données de notre firmware dans la mémoire flash externe, nous avons utilisé l'interface QuadSPI, qui offre de meilleures performances que la procédure de mise à jour actuelle par SPI. En effet, contrairement à ce dernier, le QuadSPI utilise quatre lignes de données, permettant ainsi le transfert de plusieurs bits en parallèle. Cette architecture augmente la vitesse d'écriture et de lecture. Grâce à cette interface, le microcontrôleur peut accéder au contenu de la mémoire externe sans ralentissement, ce qui nous a permis d'atteindre notre objectif et d'optimiser le temps de transfert.

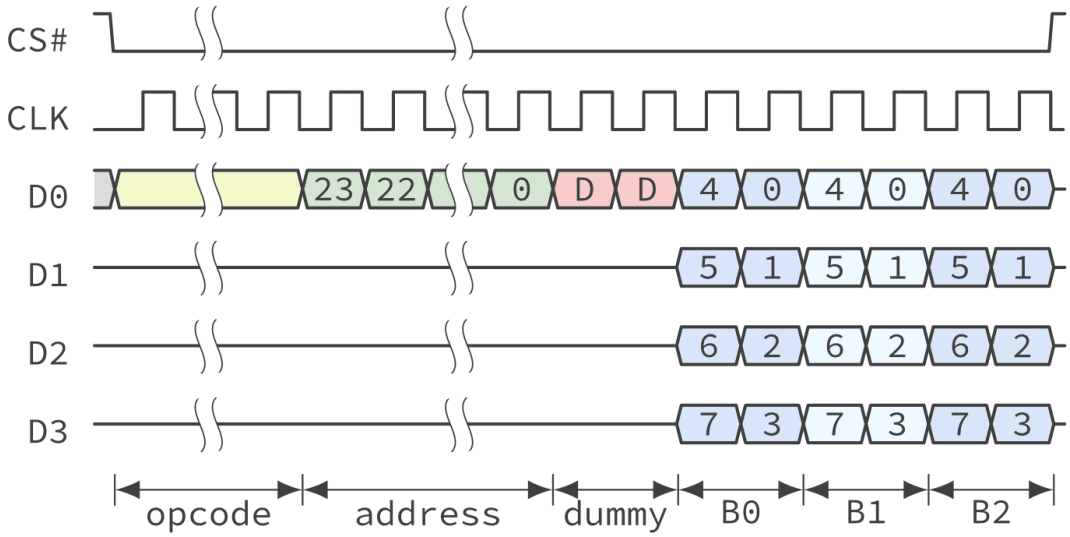

FIGURE 2.13 – Trame QSPI

Cette figure 2.5 illustre le processus de transfert d'une trame en mode QSPI.

TABLE 2.5 – Structure de la trame QSPI

Élément	**Description**	**Rôle**
CS	Chip Select	Active/désactive le périphérique
CLK	Signal d'horloge	Synchronise l'échange de données
D0-D3	Bus de données 4 bits	Transport parallèle des données
Opcode	Code d'opération	Commande (lecture/écriture/effacement)
address	Adresse mémoire	Localisation des données à lire/écrire
dummy	Cycles d'attente	Compensation de latence du périphérique
B0-B2	Bits optionnels	Configuration (mode QSPI/Dual-SPI, etc.)

Le tableau 2.13 illustre une séquence typique de lecture en mode QSPI (Quad SPI), où le signal CS active la mémoire externe et la transmission débute par l'envoi d'un opcode, représentant l'instruction de lecture, suivi de l'adresse mémoire sur plusieurs lignes de données (D0 à D3) en mode parallèle. Ensuite, des dummy cycles sont insérés pour laisser à la mémoire le temps nécessaire à la préparation des données avant leur transmission. Les données sont ensuite lues en plusieurs blocs (B0, B1, B2), ce qui permet d'optimiser le débit en exploitant le transfert sur 4 lignes simultanément.

2.4 Étude logicielle

Dans notre projet, nous avons travaillé sur de nombreux logiciels que nous traitons un par un.

2.4.1 Github

C'est un service web que nous avons utilisé pour la centralisation et la gestion de notre teste du code ainsi que notre rapport puisqu'il nous garantit la sauvegarde des versions que nous avons traité précédemment et le gain en temps à travers la répartition des tâches.

La figure 2.14 présente le logo de GitHub

FIGURE 2.14 – Logo de GitHub

2.4.2 Docklight

Nous avons recours à Docklight pour surveiller les données transférées via la communication UART. Cet outil permet de visualiser en temps réel les messages échangés, de simuler l'envoi de commandes vers le microcontrôleur, et de vérifier les réponses reçues. Il s'avère particulièrement utile pour le débogage de la communication série, en garantissant le bon fonctionnement des échanges de données entre les différents éléments du système.

La figure 2.15 affiche le logo de Docklight.

FIGURE 2.15 – Logo de Docklight

2.4.3 Python

Python a été employé dans ce projet pour développer un programme personnalisé servant à automatiser certaines opérations spécifiques. Grâce à sa simplicité de syntaxe et à la richesse de ses bibliothèques, ce langage a permis de répondre rapidement et efficacement à un besoin bien précis dans le cadre du développement. Il a ainsi facilité l'intégration d'une fonctionnalité complémentaire au fonctionnement global du système.

La figure2.16 illustre le logo de Python.

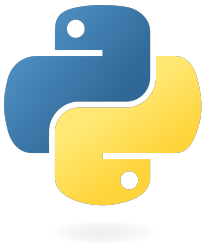

FIGURE 2.16 – Logo de Python

2.4.4 STM32CubeMX

Nous avons choisi STM32CubeMX, un outil logiciel fourni par STMicroelectronics. Il offre de nombreuses fonctionnalités, telles que la configuration graphique du microcontrôleur, l'initialisation automatique des périphériques et la génération du code source correspondant. Ce code peut ensuite être facilement importé dans un environnement de développement. Dans notre projet, nous avons utilisé IAR Embedded Workbench pour intégrer et développer le programme généré par STM32CubeMX.

La figure 1.3 montre le logo de STM32CubeMX.

FIGURE 2.17 – Logo de STM32CubeMX

2.4.5 IAR Embedded Workbench

Nous avons utilisé IAR Embedded Workbench, un environnement de développement intégré IDE reconnu pour sa performance et sa compatibilité avec les microcontrôleurs STM32. Cet outil permet d'éditer le code, de le compiler, de le déboguer et de le

téléverser sur la carte cible. Il est particulièrement adapté aux projets embarqués grâce à son optimisation du code, ses outils de debug avancés et sa prise en charge du langage C/C++. Dans notre projet, IAR a été utilisé pour développer et tester le code généré par STM32CubeMX.

La figure 2.18 présente le logo de l'outil IAR Embedded Workbench.

FIGURE 2.18 – Logo de IAR Embedded Workbench

2.4.6 STM32CubeProgrammer

Dans le but d'accéder et de lire le contenu de la mémoire flash externe connectée au microcontrôleur, nous avons utilisé STM32CubeProgrammer, un outil développé par STMicroelectronics. Il permet d'interagir avec des mémoires internes ou externes via différentes interfaces de communication telles que OCTOSPI ou QUADSPI. Grâce à son interface graphique intuitive, il facilite les opérations de lecture, d'écriture et d'effacement, ce qui s'est révélé particulièrement utile dans le cadre de notre projet.

La figure 2.19 présente le logo de STM32CubeProgrammer.

FIGURE 2.19 – Logo de STM32CubeProgrammer

2.4.7 HxD

Afin d'examiner le contenu binaire des fichiers générés, nous avons utilisé HxD, un éditeur hexadécimal libre et performant. Cet outil permet d'ouvrir et d'analyser des fichiers binaires en affichant leur représentation hexadécimale. Grâce à son interface claire et à ses fonctionnalités avancées, il est particulièrement utile pour vérifier l'intégrité des données du fichier firmware.

La figure 2.20 montre le logo de HXD.

FIGURE 2.20 – Logo de HXD

Conclusion

Ce chapitre a permis d'analyser en détail la solution technique du projet, en abordant ses aspects matériels, logiciels et protocolaires. Cette étude constitue une base solide pour aborder dans le chapitre suivant, la phase de conception et de réalisation concrète du système.

Chapitre 3

Conception et réalisation du projet

Introduction

Dans ce chapitre, nous présentons la conception de la solution mise en place, en décrivant les aspects matériels et logiciels. Ensuite, nous détaillons les tests réalisés pour vérifier le bon fonctionnement du système. Enfin, nous analysons les résultats obtenus afin de mettre en évidence les performances du système et les améliorations apportées par rapport à la situation initiale.

3.1 Conception générale de la solution proposée

Dans cette section, nous présentons la conception électrique en mettant en évidence le câblage entre les différents composants du système. Nous abordons ensuite le diagramme de séquence ainsi que la structure des données échangées via BLE. Enfin, nous décrivons la conception logicielle en détaillant le fonctionnement général à travers des organigrammes illustrant la logique des différents modules développés.

3.1.1 Conception électrique de la solution proposée

La conception électrique repose sur l'intégration de deux interfaces principales, l'UART pour la communication série ainsi que le QSPI pour le transfert de données.

Branchement Uart

Afin d'établir la communication série entre le PC et la carte ESP32 à travers un convertisseur USB vers TTL, nous avons réalisé le câblage conformément aux connexions présentées dans le tableau 3.1.

TABLE 3.1 – Configuration des broches pour la communication série ESP32–USB-TTL

Pin carte ESP32	**Pin USBTTL**
3.3V	3.3V
GPIO16 (RX2)	TX
GPIO17 (TX2)	RX
GND	GND

Lorsque la carte IHM commence à recevoir les paquets du fichier de mise à jour via BLE grâce au module ESP-WROOM-32 intégré, elle transmet chaque paquet directement au microcontrôleur STM32G4 par communication UART.

Le branchement UART est résumé dans le tableau 3.2.

TABLE 3.2 – Configuration des broches pour la communication série ESP-WROOM-32 et STM32G4

Pin ESP-WROOM-32	**Fonction**	**Pin STM32G4**	**Fonction**
GPIO3	RX	PB6	TX
GPIO1	TX	PB7	RX
GND	Masse	GND	Masse

Branchement QSPI

Nous avons utilisé le Bank 2 du QSPI afin de permettre le flashage vers la mémoire flash externe. Cette interface assure une communication rapide entre le microcontrôleur STM32 et la mémoire externe, facilitant un stockage efficace des données. La connexion entre le microcontrôleur et la mémoire W25Q32JV est décrite dans le tableau 3.3.

TABLE 3.3 – Configuration des broches pour la communication entre STM32G474VET6 et W25Q32JV

Pin STM32G474VET6	Signal	Pin W25Q32JV
PD3	CS	1
PB10	CLK	6
PD4	IO1	5
PD5	IO2	2
PD6	IO3	3
PD7	IO4	7

3.1.2 Diagramme de séquence

Le diagramme de la figure 3.1 illustre le processus de transmission d'un fichier. Celui-ci débute par le lancement du programme, qui établit une connexion BLE entre la carte de développement ESP32 et le module ESP32 intégré à la carte IHM. Cette phase est suivie par l'envoi d'informations initiales, telles que les mots de passe et la taille du fichier, chaque transmission étant confirmée par une réponse « OK ». Une fois cette étape terminée, la transmission des données s'effectue sous forme de paquets successifs. Chaque paquet est envoyé via la liaison BLE, accusé de réception par un message « OK », puis transféré vers le microcontrôleur STM32G474VET6 à travers l'interface UART. Le microcontrôleur écrit ensuite les données dans la mémoire flash W25Q32JV. À la fin du processus de flashage, un calcul CRC est effectué par le STM32G474VET6 pour vérifier l'intégrité du fichier reçu.

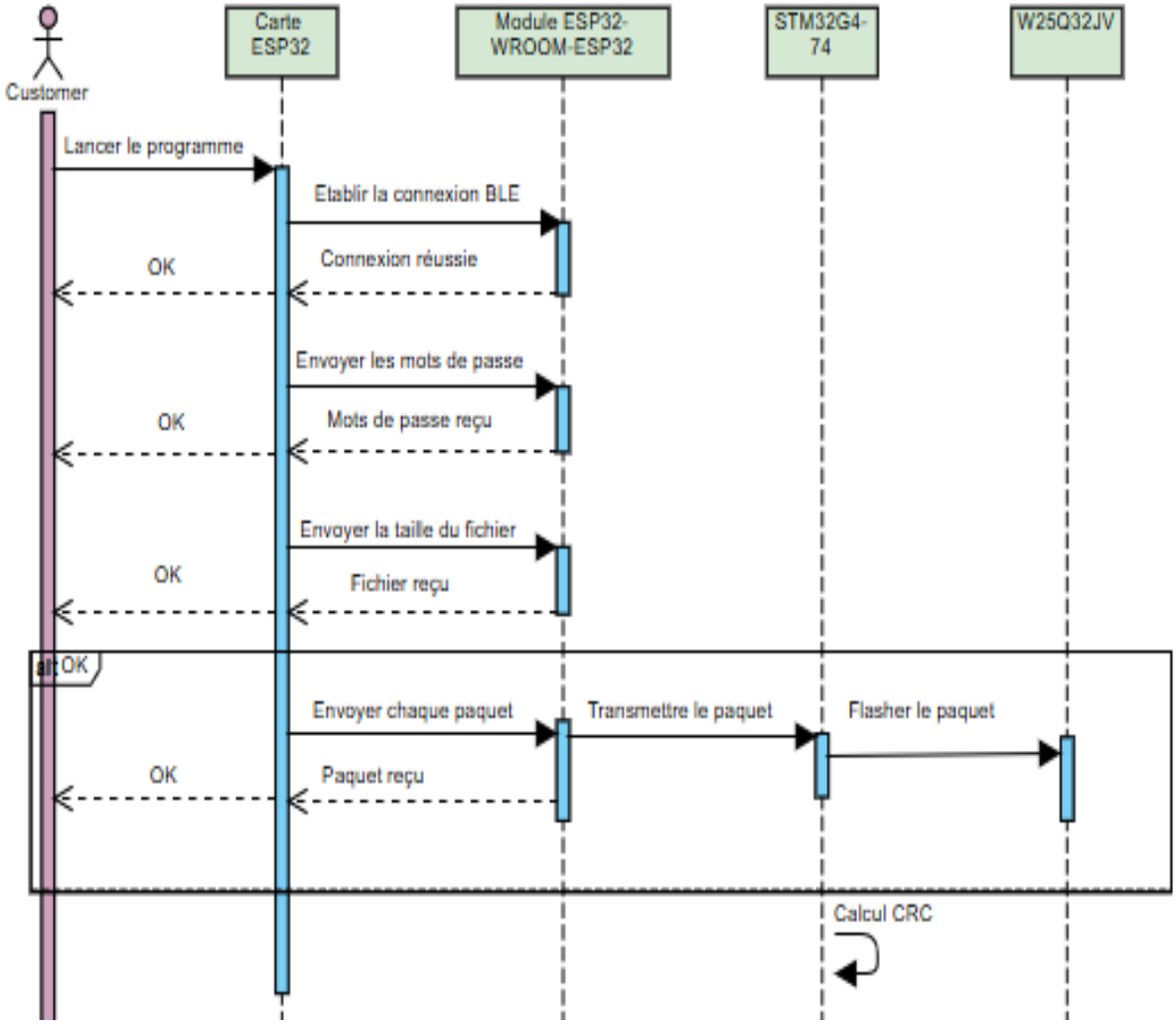

FIGURE 3.1 – Diagramme de séquence

3.1.3 Structure des données BLE

L'utilisation du BLE pour notre projet repose sur l'exploitation du service primaire numéro 4. Ce service, caractérisé par un UUID personnalisé 0x3425652D09F4474EB0AB, est utilisé pour les échanges de données liés à la transmission et à la mise à jour de fichiers.

Ce service intègre plusieurs caractéristiques présentées dans le tableau 3.4, chacune identifiée par un UUID unique, accompagnées avec des descripteurs associés, en précisant leur type, leur position dans le service, ainsi que leur valeur.

TABLE 3.4 – Caractéristiques et descripteurs GATT du service BLE n°4

Type	Service	Char	UUID des caractéristiques	Valeur
char	4	1	0xAE564A8A91AC43B997AECAE452E2AA17C	0x12
desc	4	1	1	0x2902
char	4	2	0x8E2878F2DCA4AE578D5B122D0B440A4C	0x08
char	4	3	0xF45CAF86F74E4A428C9D6CC120E678EE	0x1a
desc	4	3	1	0x2902
char	4	4	0xFE90218BE2F34D0DBAB87A18B404955	0x1a
desc	4	4	1	0x2902
char	4	5	0x0E7C3DCD480B4F729ED9054A0776428F	0x1a
desc	4	5	1	0x2902

Chaque caractéristique remplit un rôle spécifique. La première permet de lire l'état actuel du servomoteur, la deuxième sert à l'envoi de commandes de contrôle, la troisième concerne la lecture de données spécifiques, la quatrième autorise l'écriture vers le périphérique, et la cinquième est utilisée pour le transfert de fichiers. Dans le cadre de notre projet, seules la deuxième et la cinquième caractéristique seront utilisées.

3.1.4 Conception logicielle de la solution proposée

Dans cette approche, nous présentons les organigrammes illustrant le fonctionnement du code de transfert des données via BLE et celui du code de flashage des données vers la mémoire externe.

Organigramme du fonctionnement du code de transfert des données via BLE

L'organigramme présenté dans la figure 3.2 illustre le déroulement du fonctionnement du code de transfert des données, dont l'implémentation est réalisée en Python via BLE, utilisant des commandes AT transmises par un port série. Le processus débute par l'initialisation et la configuration du port série, incluant le choix du port COM et baud rate. Après l'initialisation du port série, le code procède à la configuration du module ESP32 en tant que client, à l'aide de la commande AT+BLEINIT=1. Cette

commande active la pile BLE du module et définit son rôle dans la communication. La connexion avec le servomoteur, agissant comme serveur, est ensuite établie via la commande AT+BLECONN, en précisant l'adresse MAC de ce dernier.

Une fois la connexion établie, le système entame la découverte des services GATT. La commande AT+BLEGATTCPRIMSRV=0 permet d'identifier les services principaux exposés par le serveur. Ensuite, la commande AT+BLEGATTCCHAR=0,4 est utilisée pour découvrir les caractéristiques associées au service numéro 4, sélectionné pour la transmission des données. Pour optimiser les échanges, la commande AT+BLECFGMTU=0,514 est exécutée afin d'augmenter la taille maximale des paquets de données, permettant ainsi une transmission plus efficace.

Le programme passe ensuite à la transmission de deux mots de passe. Étant donné que le processus est identique pour chaque mot de passe, une boucle de répétition est utilisée. À chaque itération, la commande AT+BLEGATTTCWR=0,4,2,,10 est envoyée, suivie de l'attente de la réponse >, indiquant que le module est prêt à recevoir les données. Une fois cette réponse reçue, le mot de passe, codé sur 10 octets, est transmis. La réception du message OK confirme que les données ont été correctement reçues. Un délai de 4 secondes est respecté entre chaque transmission afin d'assurer la stabilité de la communication.

Un fonctionnement similaire est employé pour transmettre la taille du fichier, encodée sur 5 octets et envoyée via la commande AT+BLEGATTTCWR=0,4,5,,5, vers la caractéristique numéro 5 dédiée au transfert de fichiers.

Après cette étape de préparation, le code passe à la transmission du fichier. Le fichier est découpé en paquets de 256 octets. La variable p représente le nombre total de paquets, tandis que n indique le numéro du paquet courant. Chaque paquet est envoyé vers la caractéristique numéro 5 à l'aide de la commande AT+BLEGATTTCWR=0,4,5,,256, suivant le même principe de répétition que pour les mots de passe. Un délai de 0,2 seconde est inséré entre chaque envoi. La variable n est incrémentée à chaque transmission, et lorsque n = p, la transmission est considérée comme terminée, entraînant la fermeture du port série

Afin de garantir le bon déroulement de la transmission, si la réponse > ou OK n'est pas reçue comme prévu, cela est considéré comme un échec de communication. Toutes les étapes sont affichées sur le terminal, ce qui permet un suivi en temps réel du déroulement du programme.

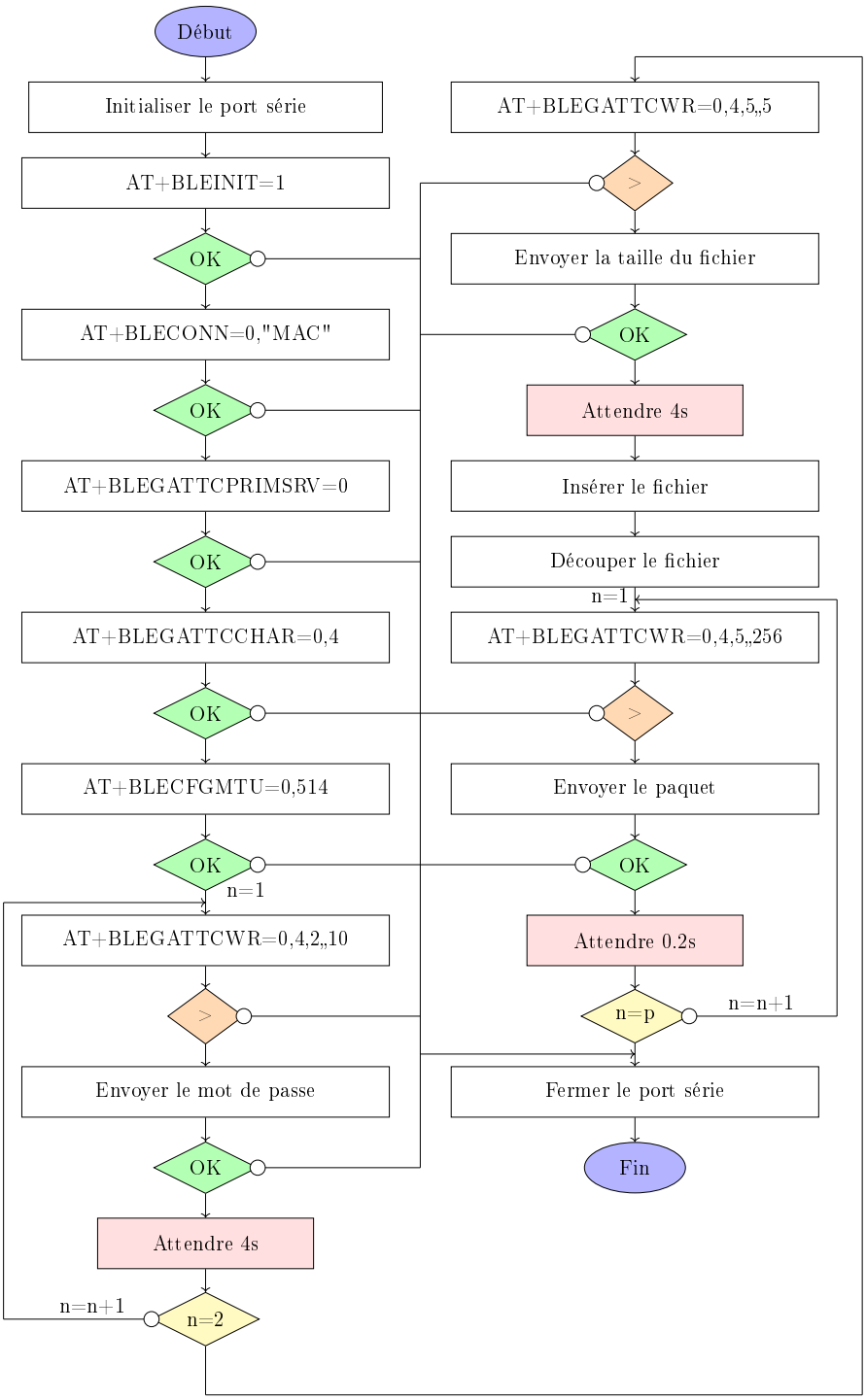

FIGURE 3.2 – Organigramme du fonctionnement du code de transfert des données via BLE

Organigramme du code de flashage des données vers la mémoire externe

L'organigramme de la figure 3.3 met en évidence les étapes du processus de gestion de la mémoire QSPI dans notre projet. Le processus commence par l'initialisation du périphérique, suivie de l'effacement complet de la mémoire pour la préparer à une nouvelle opération d'écriture. Une vérification est ensuite effectuée via l'interface QSPI afin de s'assurer que la mémoire est prête à recevoir des données.

L'activation de la fonction d'écriture permet d'enregistrer une page de données dans la mémoire externe. Une fois l'écriture terminée, un contrôle est réalisé pour confirmer son bon déroulement. Le processus s'achève ainsi, marquant la fin de la séquence de gestion de la mémoire externe.

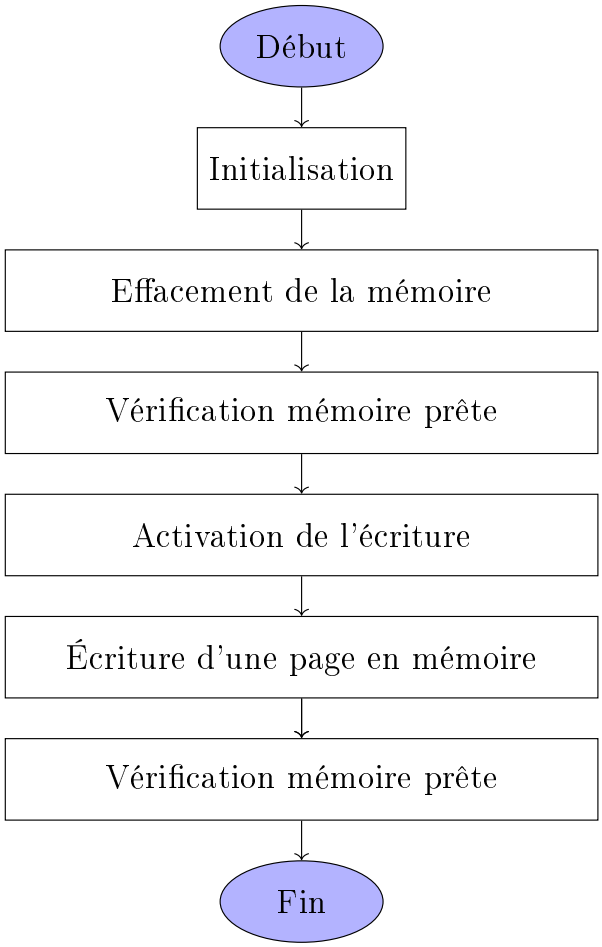

FIGURE 3.3 – Organigramme du code de flashage des données vers la mémoire externe

3.2 Configuration du systéme

Dans cette section, nous expliquons la partie de configuration du système réalisée à l'aide de l'outil STM32CubeMX et CubeProgrammer.

3.2.1 Configuration Cube MX

Dans cette partie, nous présentons la configuration du microcontrôleur STM32G4 à l'aide de l'outil STM32CubeMX. Elle concerne principalement la mise en œuvre des périphériques USART1 et QSPI1.

La figure 3.4 illustre les broches utilisées pour ces deux interfaces.

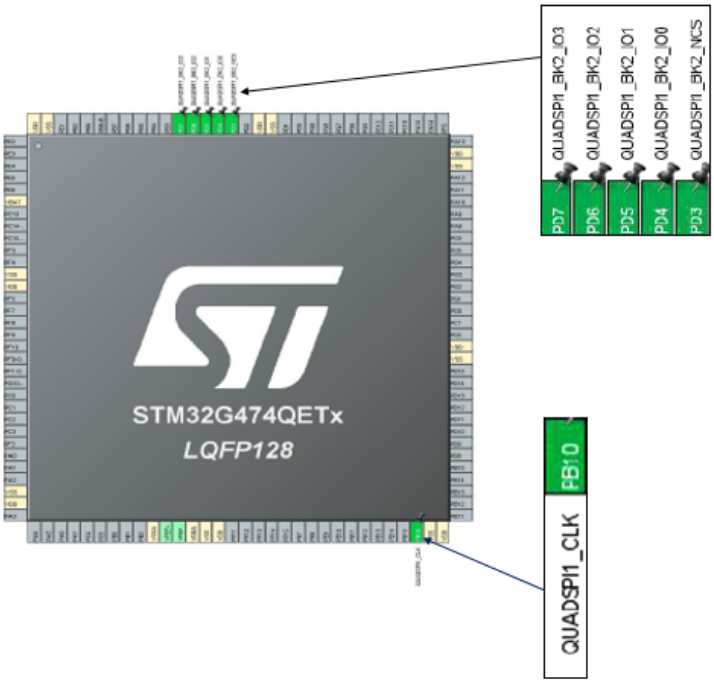

FIGURE 3.4 – Configuration des broches de STM32G474 dans STM32CubeMX

Configuration USART1 sur CubeMX

Cette configuration définit le périphérique USART1 pour une communication série asynchrone avec un débit de 115200 bits/s, standard pour les échanges de données. Les paramètres de base incluent une longueur de mot de 8 bits et 1 bit de stop.

Les paramètres avancés permettent une communication bidirectionnelle, avec transmission et réception à 16 échantillons par bit pour une meilleure stabilité. Le préscaler d'horloge est fixé à 1.

Les inversions de niveau actif sont activées pour les broches TX et RX, permettant une adaptation aux logiques inversées. La gestion de dépassement overrun est activée pour détecter les erreurs. L'ordre de transmission commence par le bit de poids faible.

Enfin, les interruptions NVIC sont configurées pour gérer les événements USART1 de manière asynchrone, et les broches GPIO sont activées pour assurer la communication.

La figure 3.5 montre en détail la configuration du périphérique USART1.

FIGURE 3.5 – Configuration CubeMX USART1

Configuration QSPI1 sur CubeMX

Ce réglage active le périphérique QUADSPI1 pour une communication haute vitesse avec la mémoire flash externe. Les paramètres clés incluent un prescaler d'horloge à 1 pour une fréquence maximale, un seuil FIFO fixé à 4 pour optimiser les transferts, et une taille de flash réglée sur 25 bits. Le temps de désélection est de 4 cycles. La mémoire est identifiée par Flash ID 2 , correspondant au Bank2, avec le mode Dual Flash désactivé.

Le DMA est configuré pour des transferts efficaces depuis le QUADSPI vers la mémoire via DMA1 Channel 1, avec une priorité faible pour ne pas bloquer d'autres processus critiques. Les adresses mémoire sont incrémentées, tandis que celles du périphérique restent fixes, et la largeur des données est définie sur 8 bits. Les interruptions NVIC sont activées pour le DMA et le QUADSPI avec une priorité élevée, assurant une gestion réactive des transferts et des erreurs.

La figure 3.6 présente la configuration détaillée du périphérique QSPI1.

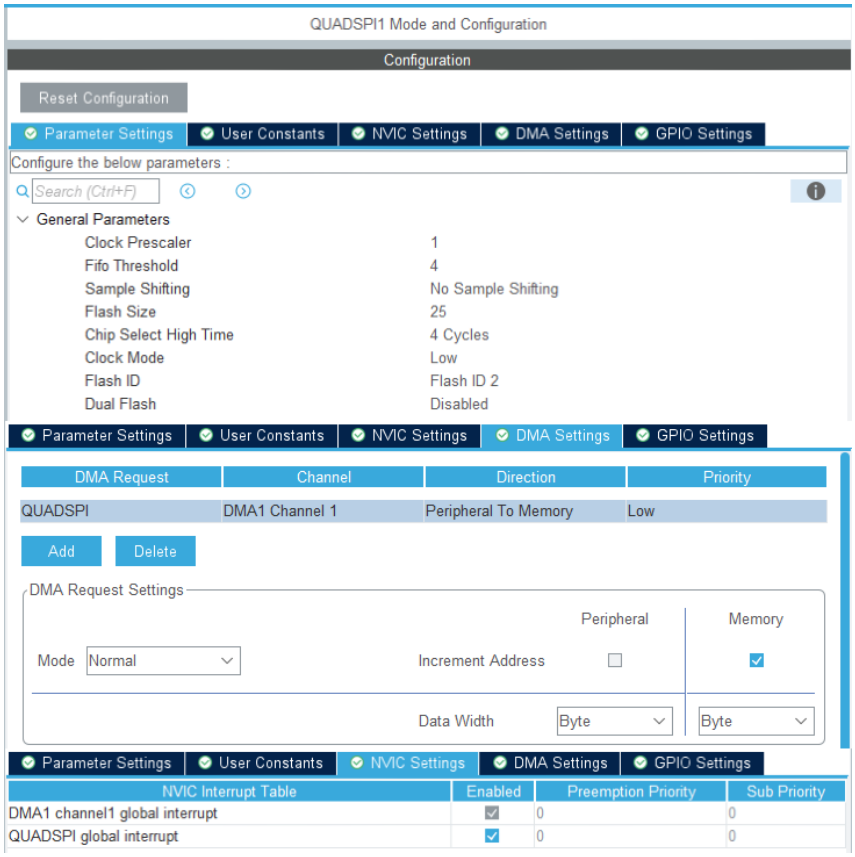

FIGURE 3.6 – Configuration CubeMX QSPI1

3.2.2 Configuration Cube Programmer

La configuration de CubeProgrammer repose sur l'inclusion d'un fichier external loader, lequel permet la prise en charge de la mémoire externe pour effectuer des opérations de lecture, d'écriture ou d'effacement directement depuis l'outil.

La figure 3.7 illustre l'intégration du nouveau fichier correspondant à la carte IHM.

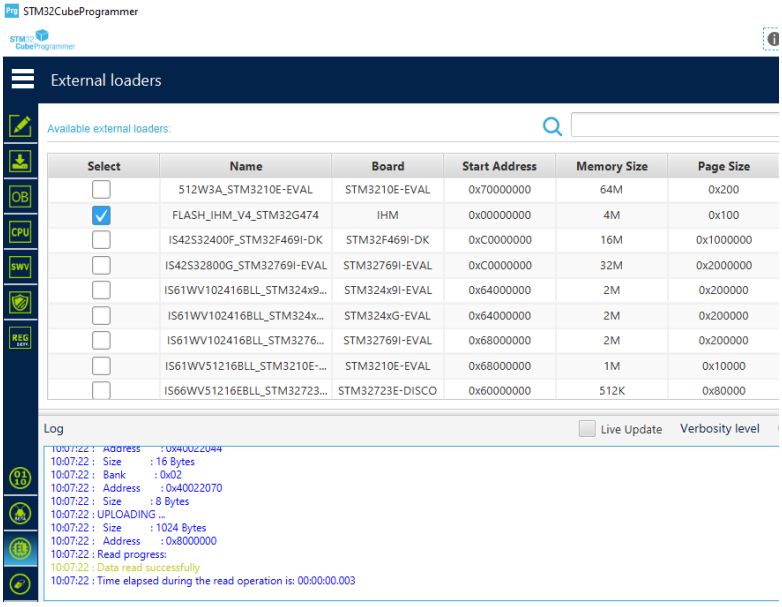

FIGURE 3.7 – L'intégration du fichier external loader

3.3 Test du système

Dans cette section, nous présentons les tests portant sur la transmission du firmware et sur le processus de flashage. Comme expliqué précédemment, la communication via Bluetooth Low Energy repose sur l'envoi de commandes AT. Pour activer cette fonctionnalité, il est nécessaire de flasher la carte ESP32 avec un firmware AT.

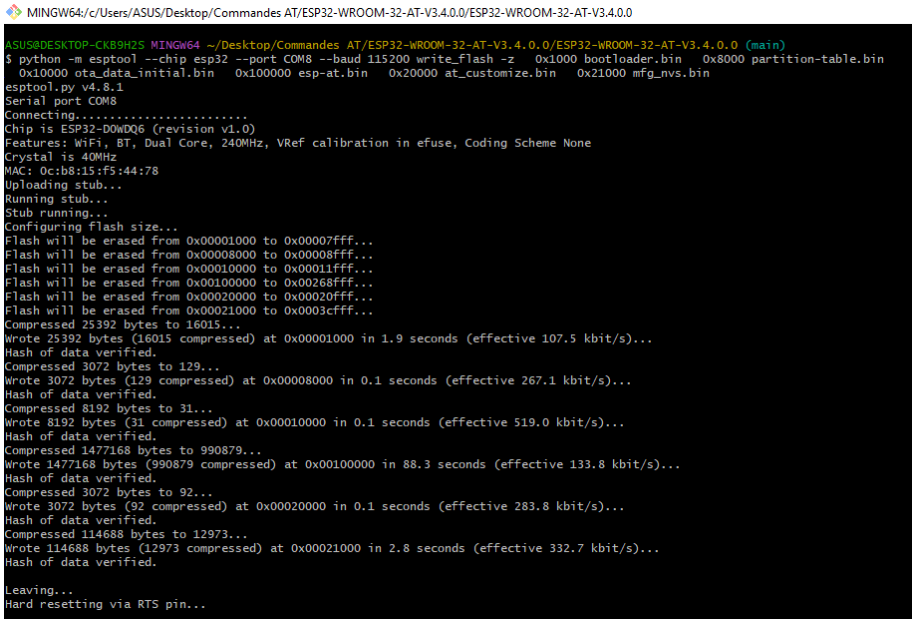

FIGURE 3.8 – Installation du firmware AT

La programmation du firmware AT sur l'ESP32 a été réalisée avec succès via l'outil esptool.py, permettant d'activer le support des commandes AT essentielles pour les communications BLE. Le processus a consisté à transférer six fichiers binaires bootloader, table de partition, firmware principal et trois fichiers de configuration. Une vérification du flashage complet des données a été effectuée, comme résumé dans la figure 3.8.

3.3.1 Test de la transmission complète du firmware

La première étape du test de transmission de fichier consiste à exécuter le code à l'aide de la commande **"python firmware-servo.py"** dans le terminal. Une fois lancé, le programme suit les étapes décrites dans l'organigramme de la figure 3.2.

```
Commande envoyée : AT+BLEINIT=1
>> AT+BLEINIT=1
>> OK

Commande envoyée : AT+BLECONN=0,"08:3A:8D:02:12:3E"
>> AT+BLECONN=0,"08:3A:8D:02:12:3E"
>> +BLESCANDONE
>> +BLECONN:0,"08:3a:8d:02:12:3e"
>> OK

Commande envoyée : AT+BLEGATTCPRIMSRV=0
>> AT+BLEGATTCPRIMSRV=0
>> +BLEGATTCPRIMSRV:0,1,0x1801,1
>> +BLEGATTCPRIMSRV:0,2,0x1800,1
>> +BLEGATTCPRIMSRV:0,3,0x180A,1
>> +BLEGATTCPRIMSRV:0,4,0x3425652D09F4474EB0AB1DFEA1BB2F01,1
>> OK

Commande envoyée : AT+BLEGATTCCHAR=0,4
>> AT+BLEGATTCCHAR=0,4
>> +BLEGATTCCHAR:"char",0,4,1,0xAE564A8A91AC43B997AECAE45E2AA17C,0x12
>> +BLEGATTCCHAR:"desc",0,4,1,1,0x2902
>> +BLEGATTCCHAR:"char",0,4,2,0x8E2878F2DCA44E578D5B122D0B440A4C,0x08
>> +BLEGATTCCHAR:"char",0,4,3,0xF45CAF86F74E4A428C9D6CC120E678EE,0x1a
>> +BLEGATTCCHAR:"desc",0,4,3,1,0x2902
>> +BLEGATTCCHAR:"char",0,4,4,0xFE90218BE2F34D0DBA8B7A18B4044955,0x1a
>> +BLEGATTCCHAR:"desc",0,4,4,1,0x2902
>> +BLEGATTCCHAR:"char",0,4,5,0x0E7C3DCD480B4F729ED9054A0776428F,0x1a
>> +BLEGATTCCHAR:"desc",0,4,5,1,0x2902
>> OK

Commande envoyée : AT+BLECFGMTU=0,514
>> AT+BLECFGMTU=0,514
>> OK
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

FIGURE 3.9 – Lancement du programme Python

Comme illustré dans la figure 3.9, les différentes commandes exécutées ainsi que les numéros des paquets envoyés sont affichés en temps réel dans le terminal, ce qui permet de suivre précisément le déroulement de la transmission.

Dès que le programme commence à s'exécuter, nous connectons un convertisseur USB-TTL au module ESP32-WROOM-32 intégré dans la carte IHM afin de visualiser les commandes ainsi que les trames reçues par ce module. La première trame affichée, illustrée dans la figure 3.10, correspond à la validation de la connexion avec la carte ESP32, indiquée par l'affichage de son adresse MAC.

```
13/06/2025 09:33:55.939 [RX] - +BLECONN:0,"48:e7:29:c4:21:de"<CR><LF>

13/06/2025 09:33:55.978 [RX] - ><CR><LF>
OK<CR><LF>
```

FIGURE 3.10 – Confirmation de connexion BLE

Une fois la connexion établie, une indication est reçue concernant la modification de la valeur du MTU, comme indiqué dans la figure 3.11. Cette valeur est configurée à une taille de 514 octets, ce qui est conforme à la limite maximale de 517 octets supportée par le module ESP32.

Cette configuration augmente l'efficacité du transfert en réduisant le nombre de paquets nécessaires à l'envoi du fichier de mise à jour.

```
13/06/2025 09:33:57.237 [RX] - +BLECFGMTU:0,514<CR><LF>

13/06/2025 09:33:57.316 [RX] - ><CR><LF>
OK<CR><LF>
><CR><LF>
OK<CR><LF>
```

FIGURE 3.11 – Changement de la taille du MTU

L'étape suivante consiste à recevoir les deux mots de passe, comme présentée dans la figure 3.12. Deux indications « +WRITE :0,3,2,10 » sont reçues. La première correspond à la réception de la trame contenant le premier mot de passe destiné au servomoteur, tandis que la seconde indique la réception de la trame contenant le deuxième mot de passe.

```
13/06/2025 09:33:57.357 [RX] - +WRITE:0,3,2,,10,<NUL>▯ <NUL>▯ j<STX> <NUL>
▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <CR><LF>

13/06/2025 09:33:57.651 [RX] - ><CR><LF>
OK<CR><LF>
><CR><LF>
OK<CR><LF>
13/06/2025 09:34:01.437 [RX] - +WRITE:0,3,2,,10,<NUL>▯ <NUL>▯ (#<NUL>▯
<NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <CR><LF>
><CR><LF>
OK<CR><LF>
>
13/06/2025 09:34:01.461 [RX] - <CR><LF>
OK<CR><LF>
```

FIGURE 3.12 – Réception des deux mots de passe

Ensuite, une nouvelle trame contenant la taille du fichier est reçue, identifiée par +WRITE :0,3,5,5. La communication cible alors la caractéristique numéro 5, dédiée au transfert de fichier, comme le montre la figure 3.13.

```
13/06/2025 09:34:05.532 [RX] - +WRITE:0,3,5,,5,<STX> 8-<EOT>▯ <NUL>▯ <CR>
<LF>

13/06/2025 09:34:07.893 [RX] - ><CR><LF>
OK<CR><LF>
><CR><LF>
OK<CR><LF>
```

FIGURE 3.13 – Réception de la taille du fichier

La figure 3.14 présente les données reçues par le microcontrôleur STM32G4 via le protocole UART, incluant l'acceptation des deux mots de passe et l'affichage de la taille du fichier, d'une taille de 300 600 octets.

Password1OK	'.' (0x1A)
Password2OK	'.' (0x1A)
TailleFichier	300'600

FIGURE 3.14 – Réception des données par le microcontrôleur STM32G4

La phase suivante est la réception des données du fichier, transmises sous forme de paquets de 256 octets.

La figure 3.15 illustre les deux premières trames reçues.

```
13/06/2025 09:34:09.932 [RX] - +WRITE:0,3,5,,256,ƒÝ<EOT>▯ <BS>▯ ‡Ý<EOT>▯
<BS>▯ ‹Ý<EOT>▯ <BS>▯ ▯ Ý<EOT>▯ <BS>▯ "Ý<EOT>▯ <BS>▯ –Ý<EOT>▯ <BS>▯ ›Ý<EOT>▯ <BS>▯ ŸÝ
<EOT>▯ <BS>▯ £Ý<EOT>▯ <BS>▯ §Ý<EOT>▯ <BS>▯ «Ý<EOT>▯ <BS>▯ ¯Ý<EOT>▯ <BS>▯ ³Ý<EOT>▯
<BS>▯ ·Ý<EOT>▯ <BS>▯ »Ý<EOT>▯ <BS>▯ ¿Ý<EOT>▯ <BS>▯ ÃÝ<EOT>▯ <BS>▯ ÇÝ<EOT>▯ <BS>▯ ËÝ
<EOT>▯ <BS>▯ ÍÝ<EOT>▯ <BS>▯ ÓÝ<EOT>▯ <BS>▯ xÝ<EOT>▯ <BS>▯ ÛÝ<EOT>▯ <BS>▯ ßÝ<EOT>▯
<BS>▯ ãÝ<EOT>▯ <BS>▯ çÝ<EOT>▯ <BS>▯ ëÝ<EOT>▯ <BS>▯ íÝ<EOT>▯ <BS>▯ óÝ<EOT>▯ <BS>▯ ÷Ý
<EOT>▯ <BS>▯ ûÝ<EOT>▯ <BS>▯ ÿÝ<EOT>▯ <BS>▯ <ETX>▯ Þ<EOT>▯ <BS>▯ <BEL>▯ Þ<EOT>▯ <BS>
▯ <VT>
Þ<EOT>▯ <BS>▯ <SI>▯ Þ<EOT>▯ <BS>▯ <DC3>▯ Þ<EOT>▯ <BS>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯
<ETB>▯ Þ<EOT>▯ <BS>▯ <ESC>▯ Þ<EOT>▯ <BS>▯ <US> Þ<EOT>▯ <BS>▯ #Þ<EOT>▯ <BS>▯ 'Þ
<EOT>▯ <BS>▯ +Þ<EOT>▯ <BS>▯ /Þ<EOT>▯ <BS>▯ 3Þ<EOT>▯ <BS>▯ óÛ<EOT>▯ <BS>▯ 7Þ<EOT>▯
<BS>▯ ;Þ<EOT>▯ <BS>▯ ?Þ<EOT>▯ <BS>▯ CÞ<EOT>▯ <BS>▯ GÞ<EOT>▯ <BS>▯ KÞ<EOT>▯ <BS>▯ OÞ
<EOT>▯ <BS>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ $@<NUL>▯ <NUL>▯ <NUL>▯ <NUL>
▯ <NUL>▯ <NUL>▯ Y@<NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ ˆÃ@<NUL>▯ <NUL>▯ <NUL>▯ <NUL>
▯ „x–A<NUL>▯ €à7yÃAC<CR><LF>

13/06/2025 09:34:10.015 [RX] - ><CR><LF>
OK<CR><LF>
><CR><LF>
OK<CR><LF>

13/06/2025 09:34:10.202 [RX] - +WRITE:0,3,5,,256,<ETB>▯ n<ENQ>▯ µµ."Fõù?é
<ETX>▯ O8M2<GS> 0ùHw,Z<¿s▯ ÝO<NAK>▯ uE:\IAT                          GIT_Local
\HMI\Src\A1.c<NUL>▯ <NUL>▯ %02d%02d%02d<NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>
▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ ú<NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯
<NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ <NUL>▯ ÌŒ<NUL>▯  x
<NUL>▯  ( <NUL>▯   Ž<NUL>▯  w<SOH>▯ „<STX> <NUL>▯ <NUL>▯ x<SOH>▯ …<STX> <SOH>▯
<NUL>▯ Œ<SOH>▯ –<STX> <STX> <NUL>▯ u<STX> œ<STX> <ETX>▯ <NUL>▯ Š<SOH>▯ •<STX>
<EOT>▯ <NUL>▯ |<SOH>▯ ¾<STX> <ENQ>▯ <NUL>▯ ¾<SOH>▯ "<STX> <ACK>▯ <NUL>▯ ˆ<SOH>▯
"<STX> <BEL>▯ <NUL>▯ ‡<SOH>▯ '<STX> <BS>▯ <NUL>▯ †<SOH>▯ '<STX> <HT>
       <NUL>▯ ~<SOH>▯ Š<STX> <LF>
<NUL>▯ v<STX> ▯ <STX> <VT>
<NUL>▯ …<SOH>▯ ▯ <STX> <CR>
<NUL>▯ „<SOH>▯ ▯ <STX> <SO>▯ <NUL>▯  <SOH>▯ ¯<STX> <SI>▯ <NUL>▯ v<SOH>▯ ƒ<STX>
<DLE>▯ <NUL>▯ ƒ<SOH>▯ Ž<STX> <SYN>▯ <NUL>▯  <SOH>▯ ™<STX> <ETB>▯ <NUL>▯ ‹<SOH>▯ –
<STX> <CAN>▯ <NUL>▯ ,<SOH>▯ <CR><LF>

13/06/2025 09:34:10.243 [RX] - ><CR><LF>
OK<CR><LF>
><CR><LF>
OK<CR><LF>
```

FIGURE 3.15 – Réception des deux premières trames du fichier de mise à jour

À l'aide de l'outil DOCKLIGHT, il est possible de visualiser les données reçues au format hexadécimal. La figure 3.16 illustre la représentation des deux premières trames. Les premières données de chaque trame, encadrées en bleu, correspondent à l'instruction +WRITE, comprenant les indices 0, 3, 5 et 256. Les données entourées en noir représentent le contenu transmis dans les paquets du fichier. Enfin, à la fin de chaque trame, les deux octets restants 0A et 0D correspondent respectivement aux caractères de contrôle Line Feed (LF) et Carriage Return (CR), mis en évidence en vert.

```
13/06/2025 09:34:09.647 [RX] - 2B 57 52 49 54 45 3A 30 2C 33 2C 35 2C 2C
32 35 36 2C 00 9B 00 20 91 DB 04 08 B9 DB 04 08 BB DB 04 08 BD DB 04 08
BF DB 04 08 C1 DB 04 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C3 DB 04 08 C5 DB 04 08 00 00 00 00 C7 DB 04 08 C9 DB 04 08 D7 DC 04 08
DB DC 04 08 DF DC 04 08 E3 DC 04 08 E7 DC 04 08 EB DC 04 08 EF DC 04 08
F3 DC 04 08 F7 DC 04 08 FB DC 04 08 FF DC 04 08 03 DD 04 08 07 DD 04 08
0B DD 04 08 E9 DB 04 08 0F DD 04 08 13 DD 04 08 17 DD 04 08 1B DD 04 08
1F DD 04 08 23 DD 04 08 F5 DB 04 08 27 DD 04 08 2B DD 04 08 FF DB 04 08
2F DD 04 08 33 DD 04 08 37 DD 04 08 3B DD 04 08 3F DD 04 08 43 DD 04 08
47 DD 04 08 4B DD 04 08 04 02 04 04 4F DD 04 08 53 DD 04 08 57 DD 04 08
0D DC 04 08 5B DD 04 08 5F DD 04 08 63 DD 04 08 67 DD 04 08 6B DD 04 08
6F DD 04 08 73 DD 04 08 77 DD 04 08 7B DD 04 08 7F DD 04 08 0D 0A
13/06/2025 09:34:09.932 [RX] - 2B 57 52 49 54 45 3A 30 2C 33 2C 35 2C 2C
32 35 36 2C 83 DD 04 08 87 DD 04 08 8B DD 04 08 8F DD 04 08 93 DD 04 08
97 DD 04 08 9B DD 04 08 9F DD 04 08 A3 DD 04 08 A7 DD 04 08 AB DD 04 08
AF DD 04 08 B3 DD 04 08 B7 DD 04 08 BB DD 04 08 BF DD 04 08 C3 DD 04 08
C7 DD 04 08 CB DD 04 08 CF DD 04 08 D3 DD 04 08 D7 DD 04 08 DB DD 04 08
DF DD 04 08 E3 DD 04 08 E7 DD 04 08 EB DD 04 08 EF DD 04 08 F3 DD 04 08
F7 DD 04 08 FB DD 04 08 FF DD 04 08 03 DE 04 08 07 DE 04 08 0B DE 04 08
0F DE 04 08 13 DE 04 08 00 00 00 00 17 DE 04 08 1B DE 04 08 1F DE 04 08
23 DE 04 08 27 DE 04 08 2B DE 04 08 2F DE 04 08 33 DE 04 08 F3 DB 04 08
37 DE 04 08 3B DE 04 08 3F DE 04 08 43 DE 04 08 47 DE 04 08 4B DE 04 08
4F DE 04 08 00 00 00 00 00 00 24 40 00 00 00 00 00 00 59 40 00 00 00 00
00 88 C3 40 00 00 00 00 84 D7 97 41 00 80 E0 37 79 C3 41 43 0D 0A
```

Paquet 2

Paquet 2

FIGURE 3.16 – Visualisation des deux premières paquets en hexadécimal

Pour chaque réception de paquet, les données sont transmises au microcontrôleur STM32G4 via l'interface UART, puis enregistrées dans un tampon nommé Buff, d'une taille de 256 octets.

La Figure 3.17 illustre le succès du chargement de ce buffer.

Upload_ctx	<struct>	
Buff	<array>" "	
size_buff	256	

FIGURE 3.17 – Chargement du buffer avec 256 octets

Le programme permet d'incrémenter l'envoi des paquets jusqu'à la transmission complète du fichier.

La figure 3.18 illustre la dernière trame reçue du fichier transféré, affichée en modes ASCII et hexadécimal, comme le détaille la figure 3.19.

Le dernier paquet contient 56 octets, car la taille totale du fichier (300 600 octets) n'est pas un multiple exact de 256.

```
13/06/2025 09:39:43.004 [RX] - +WRITE:0,3,5,,56,<STX> €<STX> ,üð<CR>
<BS>▯ ö<SOH>▯ ;<STX> <GS> <NUL>▯ ±<STX> ±<STX> úúú<SOH>▯ ú<SOH>▯ "<SUB>▯ <ETX>
▯ £<SUB>▯ <SUB>▯ !<GS> ø<EOT>▯ <ETX>▯ úú<NUL>▯ <NUL>▯ <NUL>▯ ²»Ì<NUL>▯ Ý<NUL>▯ îÿ
<HT>   "j°C`¡ß<CR><LF>

13/06/2025 09:39:43.056 [RX] - ><CR><LF>
OK<CR><LF>
><CR><LF>
OK<CR><LF>
```

FIGURE 3.18 – Réception du dernier paquet du fichier de mise à jour

```
13/06/2025 09:39:43.004 [RX] - 2B 57 52 49 54 45 3A 30 2C 33 2C 35 2C 2C
35 36 2C 02 80 02 82 FC F0 0D 08 F6 01 3B 02 1D 00 B1 02 B1 02 FA FA FA
01 FA 01 22 1A 03 A3 1A 1A 21 1D F8 04 03 FA FA 00 00 00 AA BB CC 00 DD
00 EE FF 09 22 6A B0 43 60 A1 DF 0D 0A
```

FIGURE 3.19 – Visualisation du dernier paquet en hexadécimal

3.3.2 Test du flashage du firmware dans la mémoire externe

Dans cette phase, le test consiste à écrire chaque paquet dans la mémoire externe. L'accès à la mémoire ne s'effectue qu'une fois la réception de tous les paquets confirmée par le microcontrôleur, comme illustré dans la figure 3.20.

TailleFichier	300'600
TailleRecu	300'600

FIGURE 3.20 – Terminaison du transfert du fichier

Pour vérifier ce processus, nous avons visualisé les données enregistrées à partir de l'adresse 0xD0000, sur une plage de 0x4FFF. Chaque case mémoire correspond à 32 bits. Cette observation a été réalisée à l'aide de l'outil STM32CubeProgrammer.

La figure 3.21 présente les premières données stockées dans la mémoire flash externe.

Address 0xD0000 Size 0x4ffff Data width 32-bit Find Data 0x

Address	0	4	8	C
0x000D0000	20009B00	0804DB91	0804DBB9	0804DBBB
0x000D0010	0804DBBD	0804DBBF	0804DBC1	00000000
0x000D0020	00000000	00000000	00000000	0804DBC3
0x000D0030	0804DBC5	00000000	0804DBC7	0804DBC9
0x000D0040	0804DCD7	0804DCDB	0804DCDF	0804DCE3
0x000D0050	0804DCE7	0804DCEB	0804DCEF	0804DCF3
0x000D0060	0804DCF7	0804DCFB	0804DCFF	0804DD03
0x000D0070	0804DD07	0804DD0B	0804DBE9	0804DD0F
0x000D0080	0804DD13	0804DD17	0804DD1B	0804DD1F
0x000D0090	0804DD23	0804DBF5	0804DD27	0804DD2B
0x000D00A0	0804DBFF	0804DD2F	0804DD33	0804DD37
0x000D00B0	0804DD3B	0804DD3F	0804DD43	0804DD47
0x000D00C0	0804DD4B	04040204	0804DD4F	0804DD53
0x000D00D0	0804DD57	0804DC0D	0804DD5B	0804DD5F
0x000D00E0	0804DD63	0804DD67	0804DD6B	0804DD6F
0x000D00F0	0804DD73	0804DD77	0804DD7B	0804DD7F
0x000D0100	0804DD83	0804DD87	0804DD8B	0804DD8F
0x000D0110	0804DD93	0804DD97	0804DD9B	0804DD9F
0x000D0120	0804DDA3	0804DDA7	0804DDAB	0804DDAF
0x000D0130	0804DDB3	0804DDB7	0804DDBB	0804DDBF
0x000D0140	0804DDC3	0804DDC7	0804DDCB	0804DDCF
0x000D0150	0804DDD3	0804DDD7	0804DDDB	0804DDDF
0x000D0160	0804DDE3	0804DDE7	0804DDEB	0804DDEF
0x000D0170	0804DDF3	0804DDF7	0804DDFB	0804DDFF

FIGURE 3.21 – Affichage des données initialement présentes dans la mémoire flash externe

Une fois le flashage effectué, nous avons enregistré les données flashées à l'aide de l'outil STM32CubeProgrammer, dans un fichier nommé **External_flash**. Ce fichier, de type binaire, a la même taille que le fichier original de mise à jour, intitulé **HMI_V4_For_CI2802_v4.2.4_nBOOTLOADER.bin**.

L'étape finale consiste à vérifier l'intégrité du transfert de fichier en comparant les deux fichiers obtenus. Ces derniers sont ouverts à l'aide de l'outil HxD pour une visualisation en mode hexadécimal.

La figure 3.22 présente les premières et les dernières séquences de données, qui correspondent parfaitement aux mêmes adresses mémoire dans les deux fichiers,

confirmant ainsi la fidélité du transfert. Dans cette figure, les derniers octets encadrés en bleu indiquent la fin des données, tandis que ceux encadrés en vert représentent le résultat du calcul du CRC.

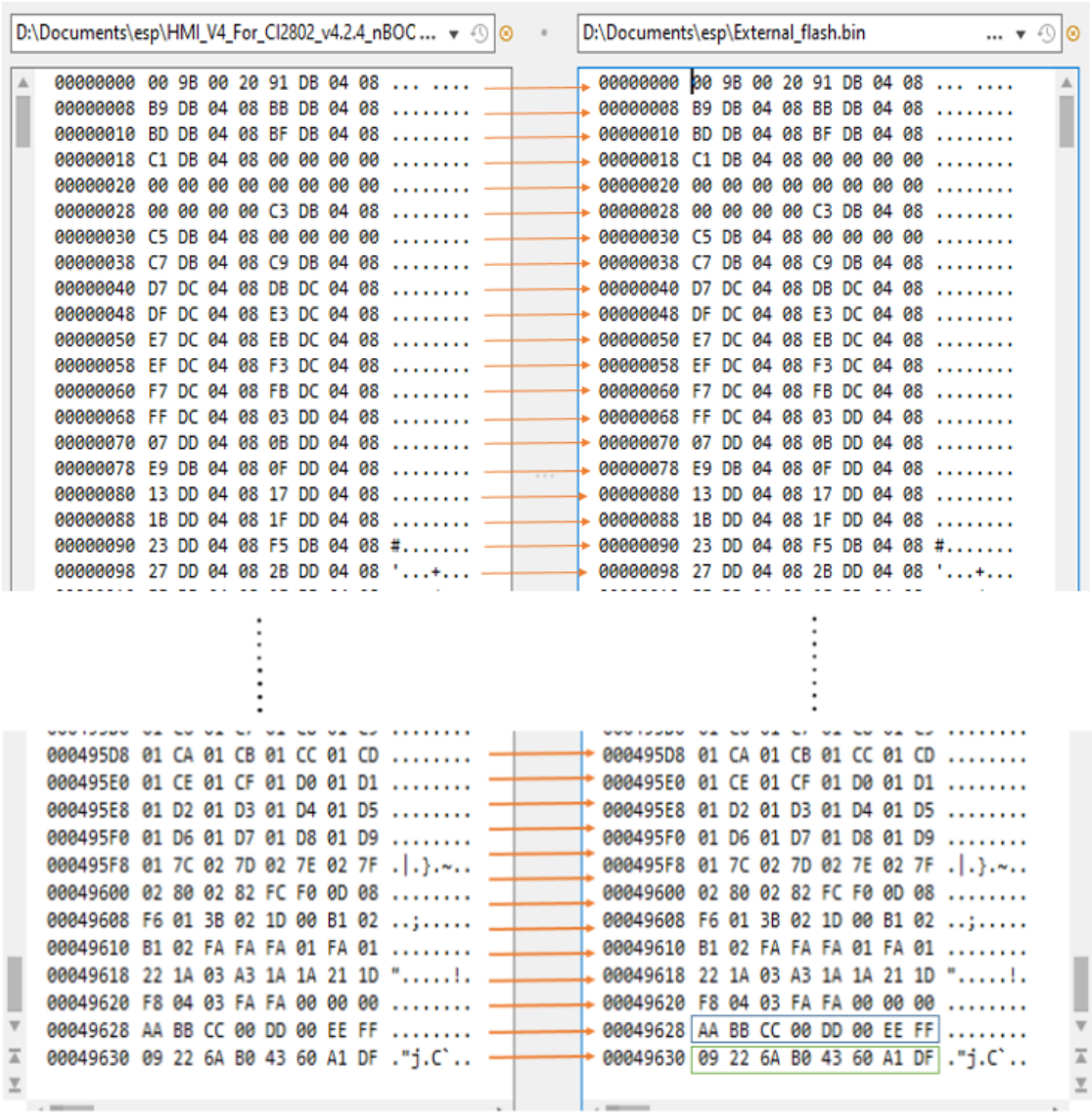

FIGURE 3.22 – Comparaison des trames du fichier d'origine et du fichier flashé

3.4 Résultat

Le résultat principal de notre projet est que la nouvelle mise à jour fonctionne correctement avec le servomoteur, tout en apportant une optimisation du temps d'exécution.

La figure 3.23 présente l'ancienne version de la mise à jour version 4.2.3 avant le transfert du fichier, tandis que la figure 3.24 montre le processus de sauvegarde et le bon fonctionnement de la mise à jour version 4.2.4.

FIGURE 3.23 – Version mise à jour 4.2.3

FIGURE 3.24 – Version mise à jour 4.2.4

Avant l'intégration de notre solution, le temps de mise à jour était d'environ 30 minutes. Grâce aux optimisations apportées, ce délai a été réduit à 6 minutes.

La figure 3.25 illustre l'intervalle de temps entre la réception de la première commande de connexion (figure 3.10) et celle de la dernière trame du fichier (figure 3.19).

FIGURE 3.25 – Intervalle de temps du transfert du fichier de mise à jour

Conclusion

Ce chapitre a présenté la phase de conception de notre système, une étape essentielle pour garantir la réussite de sa mise en œuvre. Elle nous a permis de structurer notre travail et de poser les bases nécessaires à la réussite des tests effectués par la suite.

Conclusion Générale

Au terme de ce projet, nous avons réussi à concevoir et développer un système embarqué performant permettant la mise à jour à distance des servomoteurs via la technologie Bluetooth Low Energy (BLE). Cette solution répond efficacement à la contrainte majeure liée à l'absence d'accès à une application mobile dédiée, en proposant une alternative fiable et sécurisée.

L'optimisation du processus a permis de réduire significativement le temps de mise à jour, passant de 30 minutes à 6 minutes, ce qui constitue un gain opérationnel substantiel dans un contexte industriel où la minimisation des interruptions est primordiale. Le recours aux technologies récentes a assuré une architecture moderne et évolutive, adaptée aux exigences techniques et fonctionnelles du projet.

Le projet a été mené selon une démarche méthodique, en s'appuyant sur l'analyse des besoins fonctionnels et techniques, puis sur la conception et le développement du système embarqué. Les phases de tests et de validation ont permis d'assurer la fiabilité et la performance de la solution, en conformité avec les objectifs fixés.

En gisse de perspective, l'intégration de la solution complète au sein de l'application mobile existante permettra de maximiser son impact. De plus, l'adoption des nouvelles technologies et l'optimisation continue du système contribueront à renforcer l'efficacité opérationnelle et la flexibilité du dispositif dans un environnement industrielss.

Bibliographie

[1] site officiel de innov-alliance-tech https://ia-tech.net/. Accès le 24/02/2025.

[2] Wikipidea,https://fr.wikipedia.org/wiki/Servomoteur. Accès le 7/05/2025.

[3] Le cycle en v,https://openclassrooms.com/fr/courses/8186936-gerez-un-projet-avec\hline--la-methodologie-cycle-en-v/8301123-identifiez-les-specificites-du-cycle-en-v. Accès le 1/03/2025.

[4] Datasheet esp32-wroom-32 ,https://lastminuteengineers.com/esp32-wroom-32-pinout-reference/. Accès le 16/03/2025.

[5] Datasheet st, https://www.st.com/en/microcontrollers-microprocessors/stm32g474re.html. Accès le 19/04/2025.

[6] Ble, https://www.bluetooth.com/learn-about-bluetooth/tech-overview/. Accès le 03/03/2025.

[7] Gap & gatt, https://punchthrough.com/how-gap-and-gatt-work/l. Accès le 05/03/2025.

[8] Commande-at+ble, https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT_Command_Set/BLE_AT_Commands.html. Accès le 15/03/2025.

[9] Uart, https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware\hline-communication-protocol.html. Accès le 10/04/2025.

Résumé

Ce projet de fin d'études, réalisé au sein de l'entreprise I@T, a porté sur le développement d'un système embarqué permettant d'optimiser le temps de transfert de fichiers de mise à jour destinés à un servomoteur, via la technologie Bluetooth Low Energy (BLE).

L'objectif principal était de garantir une transmission rapide, fiable et sécurisée, en étudiant les performances de la communication BLE et en implémentant des solutions efficaces pour minimiser le temps de transfert. Ce travail m'a permis de consolider mes compétences en systèmes embarqués, en communication sans fil, ainsi qu'en optimisation logicielle, tout en me familiarisant avec les contraintes techniques du monde industriel.

Abstract

This final-year project, conducted at the company I@T, focused on the development of an embedded system aimed at optimizing the transfer time of update files for a servomotor via Bluetooth Low Energy (BLE) communication.

The main objective was to ensure fast, reliable, and secure transmission by analyzing the performance of BLE and implementing appropriate techniques to reduce transfer time. This project allowed me to strengthen my skills in embedded systems, wireless communication, and software optimization, while gaining hands-on experience with the technical constraints of industrial environments.