UNIVERSITE CADI AYYAD

Ecole Nationale des Sciences Appliquées – Marrakech

Filière : Ingénierie des Systèmes Electroniques Embarqués et Commande des Systèmes (SEECS)

Rapport du projet de fin de semestre

Conception de Mise à Jour de Firmware Over-the-Air (OTA), Inspiré des Pratiques des ECU Automobile

Réalisé par :

EL MOUSADIK Saad

MOUBTAKIR Abdessamad

MOUFIDY Youness

OUAADDI Mohamedbahae

OUTIRZA Brahim

Encadré par :

Pr.HATIM Anas

Année Universitaire 2023-2024

Remerciements

Au bon DIEU, nous sommes profondément reconnaissants pour nous avoir béni avec une santé robuste, une force intérieure, une volonté inébranlable et une patience infinie tout au long de notre parcours d'études. Ces dons précieux ont été essentiels pour surmonter les défis et persévérer dans nos études.

Nous tenons à exprimer notre respect le plus sincère envers Monsieur Anas Hatim, qui a été un mentor exceptionnel pendant toute la durée de notre projet. Sa présence bienveillante et son engagement indéfectible nous ont énormément soutenus et ont contribué à notre épanouissement professionnel. Nous sommes extrêmement reconnaissants pour sa disponibilité à répondre à nos questions, pour ses recommandations éclairées qui nous ont guidés dans nos tâches et projets, et surtout pour la confiance inestimable qu'il a placée en nous. Cela nous a motivés à donner le meilleur de nous- mêmes et à développer nos compétences avec assurance.

Nous souhaitons également exprimer notre gratitude envers toutes les personnes qui, de près ou de loin, nous ont offert leur aide et leur soutien diversifiés. Que ce soient nos familles, nos amis, nos camarades de classe ou même des inconnus bienveillants, chaque geste d'assistance a été d'une grande valeur et nous a rappelé l'importance des liens humains dans l'atteinte de nos objectifs.

Enfin, nous sommes emplis d'une immense gratitude envers toutes ces personnes qui ont été présentes à nos côtés, qui nous ont inspirés, guidés et soutenus tout au long de notre parcours éducatif. Leurs contributions généreuses ont joué un rôle déterminant dans notre succès, et nous serons éternellement reconnaissants envers eux. Que DIEU les bénisse tous.

Résumé

Les unités de commande électronique (ECU) prolifèrent de nos jours dans les véhicules intelligents modernes. Ces ECU gèrent diverses fonctions du véhicule à l'aide de firmware. Les constructeurs automobiles développent généralement des mises à jour de firmware pour corriger les bogues ou améliorer les fonctionnalités du véhicule. La procédure de mise à jour logicielle est critique et chronophage dans le domaine automobile. Cependant, l'industrie des télécommunications utilise la technologie innovante de gestion des logiciels mobiles pour mettre à jour à distance le système, appelée Firmware Over-the-Air (FOTA). La FOTA permet aux fabricants de dispositifs et aux opérateurs mobiles de fournir des mises à jour logicielles efficaces et rapides pour leurs produits. Proposer la technologie FOTA dans l'industrie automobile réduit les coûts et améliore la satisfaction des clients en évitant les rappels annuels des véhicules. Ce projet propose et teste un processus de mise à jour de firmware pour les microcontrôleurs STM32, inspiré par la technologie FOTA utilisée dans les véhicules commerciaux. À travers ce rapport, le protocole de mise à jour FOTA pour les STM32 dans le réseau CAN est présenté et analysé.

Mots clés: Firmware Over-the-Air (FOTA), Microcontrôleurs STM32, Réseau CAN, Mises à jour de firmware, Industrie automobile, Gestion logicielle à distance

Abstract

Electronic control units (ECUs) are proliferating in modern intelligent vehicles these days. These ECUs manage various functions in the vehicle with the aid of firmware. Automotive manufacturers usually develop firmware updates to fix bugs or enhance vehicle functionality. The software update procedure is critical and time-consuming in the automotive domain. However, the telecommunications industry utilizes innovative mobile software management technology to remotely update systems, known as Firmware Over-the-Air (FOTA). FOTA allows device manufacturers and mobile operators to provide efficient and timely software updates for their products. Introducing FOTA technology in the automotive industry reduces costs and improves customer satisfaction by avoiding annual vehicle recalls. This project proposes and tests a firmware update process for STM32 microcontrollers, inspired by the FOTA technology used in commercial vehicles. Through this report, the FOTA update protocol for STM32 in the CAN network is presented and analyzed.

Keywords: Firmware Over-the-Air (FOTA), STM32 microcontrollers, CAN network, Firmware updates, Automotive industry, Remote software management

Table de matières :

Liste des figures

Chapitre I Introduction

1. Contexte général

L'industrie automobile vise principalement à augmenter la sécurité et le confort en ajoutant des fonctionnalités technologiques avancées. Celles-ci sont rendues possibles par l'électronique, la communication sans fil et des logiciels complexes capables d'analyser de grandes quantités de données. Cela conduit à une augmentation du nombre d'unités de commande électronique (ECU). Il est estimé que la future révolution automobile sera basée à 90 % sur le développement des ECU et à 80 % sur les firmwares des ECU dans les véhicules. L'augmentation de la complexité des firmwares des ECU génère plus de bogues logiciels. Par conséquent, les mises à jour de firmware intra-véhicule économiques pour les ECU automobiles deviennent plus attrayantes pour les chercheurs.

Actuellement, les véhicules sont rappelés par les concessionnaires pour mettre à jour le logiciel des ECU, corriger les bogues du firmware ou améliorer les fonctionnalités. Les rappels de véhicules sont un processus coûteux et contraignant à la fois pour les clients et pour les fabricants d'équipements d'origine (OEM). Dans ce projet, la Firmware Over-the-Air (FOTA) est proposé et testé. "FOTA" fait référence à la procédure de diffusion du dernier logiciel pour configurer ou déboguer des dispositifs dotés de connexions sans fil ou cellulaires. Les dispositifs cellulaires récents utilisent la technologie FOTA pour mettre à jour et corriger les bogues de leurs produits. Cependant, l'application de cette technologie dans l'industrie automobile réduit les coûts pour les clients, les concessionnaires, les constructeurs de véhicules et les OEM.[1]

2. Problématique

2.1. Definition du problème

Les véhicules modernes sont équipées de plus de 80 calculateurs[2], chacune a un rôle important dans l'opération du véhicule. D'après Renault , Cela représente entre 50 et 80 millions de lignes de code logiciel dans les différents calculateurs[3], ce qui représente une complexité au niveau logicielle, ce nombre peut augmenter avec la nouvelle tendance "connected cars" selon Sven kappel le manager de projet de Bosh Mobility.Il est essentiel pour les équipementiers automobiles (OEMs) de gérer efficacement le logiciel tout au long du cycle de vie du véhicule, à la fois pour améliorer les performances et aussi pour apporter des corrections aux logiciels défectueux qui mettent en danger des vies humaines ou l'environnement, et qui pourraient entraîner de coûteux rappels de produits. En 2023, le rappel le plus important a été celui du pilote automatique de Tesla, qui a touché plus de 2 millions de voitures, mettant en évidence les défis permanents et évolutifs auxquels est confronté le secteur [4].

Bien que l'utilisation des mises à jour logicielles OTA (Over-the-Air) soit en augmentation, il existe encore de nombreux cas où la seule solution consiste à remplacer le module entier simplement pour mettre à jour le logiciel. L'industrie doit travailler davantage sur les mises à jour logicielles, à la fois chez le concessionnaire et via OTA. Des nombreux équipementiers automobiles ont entrée dans le marché de FOTA parmi lesquelles l'entreprise Tesla qui a réalisé la technologie OTA dans leurs véhicules pour ajouter des fonctionnalités au logiciel ,aussi le groupe Renault avec le système Multimédia EASY LINK permet d' ajouter des nouvelles fonctionnalités et garder le véhicule dans les meilleures conditions de travail, D'après Renault groupe 85% à 90% des véhicules vont recevoir des nouveaux mise à jour comparant au 60% avant OTA[5].

2.2. Défis Majeurs

En basant sur l'importance d'OTA dans les nouvelles générations, il est important de noter qu'il est soumis au plusieurs défis :

- Défis au niveau de mémoire.
- Défis au niveau de temps.
- Défis au niveau de sécurité.
- Défis au niveau de conception.
- Défis au niveau des systèmes qu'on a beaucoup des calculateurs.

3. Gestion du projet

3.1. FOTA

3.1.1 Définition

Firmware over the air ou FLASH over the air est une technique qui permet aux appareilles de IOT de faire la mise à jour du code à distance, sans la nécessité d'accéder à l'appareil. Avec FOTA on peut ajouter des nouvelles fonctionnalités ou réparer des bugs dans le code[6].

3.1.2 Exemples

Cette technologie est ultra discrète, voir même, le plus souvent, totalement invisible, Et pour cause, les mises à jour en FOTA s'effectuent par la voie des airs, via le réseau téléphonique GSM.

FOTA a beaucoup d'avantages, pour cela la totalité des voitures modernes ont le système FOTA intégrés dans leurs véhicules par exemples :

- La mise à jour FOTA via le système multimédia Renault EASY-LINK est disponible sur une large gamme de voiture Renault (Twingo, Clio, ZOE, Captur, Mégane...). [7]
- Dans 2016, Tesla a utilisé FOTA pour mettre à jour chacune de ses voitures avec la capacité de se garer automatiquement, Sans FOTA, chaque voiture aurait dû être rappelée ou recevoir la visite d'un technicien Tesla pour ce type de mise à jour soit installée.

3.2. Les techniques de programmation des microcontrôleurs (Flashing)

Le processus de mise à jour de la mémoire Flash, qui est couramment utilisée pour stocker le code dans la plupart des systèmes embarqués, Il est parfois appelé "Burning" ou "programming" en anglais. La mémoire Flash est une mémoire ROM qui utilise des transistors "Floating gate MOSFETs" pour stocker les informations. Comme une tension élevée est requise pour écrire dans ces transistors, un "burner" ou un "Flasher " est utilisé pour fournir cette tension nécessaire car le flasher est équipée d'un pilote de flash (Flash driver).

❖ Les techniques de programmation

Sans utiliser FOTA, il y a plusieurs techniques qu'on peut utiliser pour changer le micrologiciel dans un appareil :

- **Off circuit programming :** C'est la technique la plus ancienne dans laquelle, on connecte des pins de microcontrôleur avec le Flasher. On dit dans ce cas le Flash driver est extérieur du microcontrôleur. Le rôle de Flasher dans ce cas est d'avoir le binaire de pc via un port de communication (USB par exemple), puis il applique une grande tension pour sauvegarder le fichier dans la mémoire Flash du microcontrôleur. Or cette approche consomme beaucoup de temps et n'est pas pratique dans la plupart des cas.
- **Programmation dans le circuit (In circuit programming):** Dans ce cas le flash driver est interne dans le microcontrôleur, le flash driver peut générer n'importe quelle tension

pour programmer la mémoire Flash, dans ce cas le Flasher peut-être considérer comme un intermédiaire pour échanger le binaire entre le pc et le microcontrôleur. Par exemple stm32f103 il existe deux protocoles de communication avec le flash driver : JTAG et SWD. Pour cela on utilise STLINK,qui est considéré comme un traducteur de protocole USB vers le protocole SWD,STLINK dépend du microcontrôleur, ce qui pose un problème si on a des différents microcontrôleurs, aussi il faut 4 pins pour utiliser STLINK qui est pas pratique si le nombre de calculateurs augmentée cas dans les systèmes véhiculaires moderne).

Figure 1 : STLink Programmer

- **Programmation dans le circuit avec le bootloader (In circuit programming with bootloader) :** Le bootloader est un firmware comme n'importe quelle firmware(Il est définit par un son propre startup code et son propre vecteu réinintialisation) ,il est responsable de charger et d'exécuter ui autre programme, Il est utilisée pour télécharger le firmware si avoir besoin de matériel de programmation spécifique. Ainsi, au lieu d'utiliser différentes interfaces avec différents microcontrôleurs, nous utilisons un chargeur de démarrage pour créer une interface unifiée avec le système. Par exemple, si nous avons 100 calculateurs dans un système, nous concevrons un chargeur de démarrage pour chacun d'entre eux avec un protocole de communication standard tel que UART ou Ethernet.

3.3. Cycle de vie du développement logiciel

Pour créer un logiciel embarqué, les développeurs doivent respecter beaucoup de contraintes comme le temps, l'argent, la puissance et la performance, un bon cycle de développement va aider à faire un compromis entre ces détecteurs de performance.

3.3.1 Définition

Le cycle de vie du développement logiciel (SDLC) désigne une méthodologie comportant des processus clairement définis pour créer des logiciels de haute qualité. L'un des principaux objectifs du SDLC est de produire un logiciel robuste dans le cadre d'un budget et d'un calendrier précis [8].

Essentiellement, le SDLC décrit un plan détaillé avec des étapes qui comprennent chacune leur propre processus et leurs propres produits livrables. Par rapport aux autres méthodes de production, le SDLC accélère le développement des projets et minimise les coûts.

En général, on trouve sept phases du SDLC :

3.3.2 Les phases de cycle de vie de développement logiciel

En général, on trouve sept phases du SDLC :

- **Planification :** La première étape que l'équipe de développement doit franchir avant même de s'engager dans l'écriture du code est de recueillir toutes les exigences fonctionnelles afin d'évaluer l'étendue du travail et le coût du projet. Dans le cadre de ce processus, les coûts de main-d'œuvre et de matériel, un calendrier est élaboré avec des objectifs, et des équipes et des directions sont créées pour le projet.
- **Analyse des exigences :** Il s'agit d'une étape très importante où toutes les exigences doivent être définies. À la fin de cette phase, l'équipe comprend clairement ce que le logiciel est censé faire et quelles fonctionnalités doivent être incluses. En outre, cette étape du SDLC est essentielle car elle permet aux développeurs d'en savoir plus sur tous les obstacles que le projet peut rencontrer en cours de route. Une fois les exigences comprises, un document SRS (Software Requirement Specification) est créé. Il est important que les deux développeurs suivent les directives de ce document et que le client le relise pour référence ultérieure.
- **Conception et prototypage :** La conception et l'architecture sont cruciales dans le processus de développement logiciel, quelle que soit la taille du futur logiciel. La plupart des méthodologies SDLC accordent une grande importance à cette phase particulière, car elle définit l'aspect de l'application et son degré de sécurité pour les utilisateurs finaux.
- **Développement :** C'est à ce moment que le processus de développement commence réellement. Il est possible de faire rédiger un petit projet par un couple de développeurs, mais un projet complexe peut nécessiter la participation de plusieurs équipes. Outre le codage, de nombreuses autres tâches sont nécessaires.

Il est essentiel de trouver et de corriger les erreurs et les problèmes. Certaines tâches, comme les résultats des tests ou la compilation du code pour exécuter une application, ralentissent le processus de développement. SDLC peut anticiper ces retards, ce qui permet aux équipes de développement de se concentrer sur d'autres tâches.

- **Test :** Aucun logiciel ne quitte le laboratoire sans avoir été testé de manière approfondie. Les activités de test sont généralement incluses dans toutes les étapes des modèles SDLC modernes, cette étape en est donc généralement un sous-ensemble. Cependant, cette phase ne concerne que l'étape de test du produit, au cours de laquelle les bogues sont découverts, signalés, corrigés et retestés jusqu'à ce que le produit réponde aux critères de qualité
- **Déploiement :** Il est maintenant temps de déployer le logiciel en production afin que les utilisateurs puissent commencer à l'utiliser. Toutefois, de nombreuses entreprises font passer le logiciel par différents environnements de déploiement, tels que des environnements de test ou de préparation. De cette façon, les parties prenantes peuvent tester le produit en toute sécurité avant qu'il ne soit mis sur le marché. Cela permet également d'identifier les derniers bogues avant la mise sur le marché d'un produit.
- **Maintenance et soutenir :** Le cycle de développement est presque terminé à ce stade. L'application a été réalisée et est maintenant utilisée. Cependant, la maintenance et le support jouent encore un rôle important. À ces stades, les utilisateurs peuvent trouver des bogues qui n'ont pas été découverts lors des tests. La résolution de ces erreurs peut créer de nouveaux cycles de développement. Outre la correction des bogues, des modèles comme le développement itératif prévoient des fonctionnalités supplémentaires pour les futures versions. Un nouveau cycle de développement peut être lancé pour chaque nouvelle version.

3.4. Comment améliorer le cycle de développement

En appliquant le concept de FOTA aux appareils embarqués, nous pouvons ajouter la phase de maintenance aux phases de développement dans SDLC ce qui va améliorer la qualité de code et il va ajouter une valeur ajoutée. Comme les applications mobiles qui arrivent sur le marché sans avoir terminé toutes les fonctionnalités et qui seront ajoutées plus tard grâce à la technologie OTA.

Chapitre II Conception et Réalisation du système

1. Architecture du Système et Matériel

1.1. Schéma bloc du système

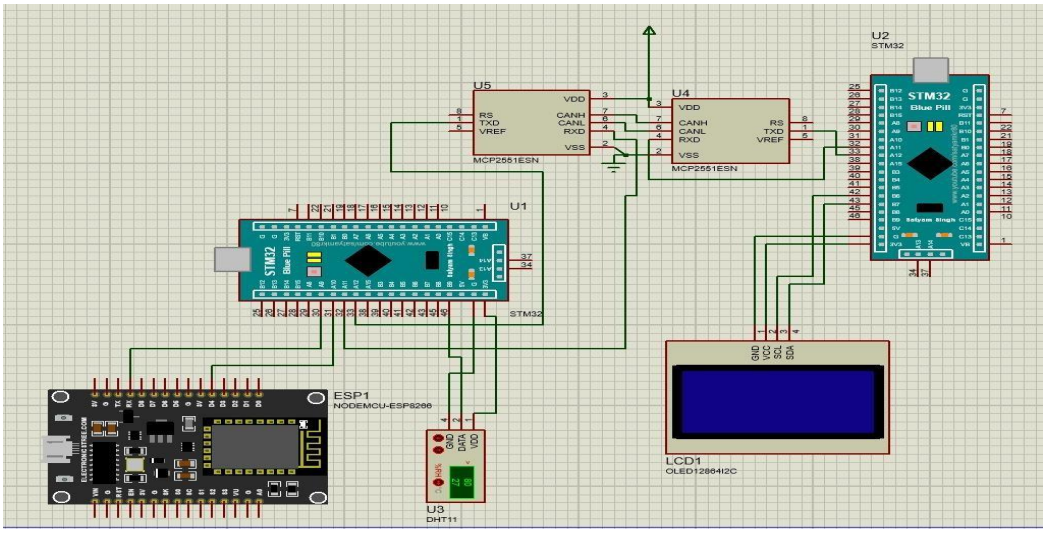
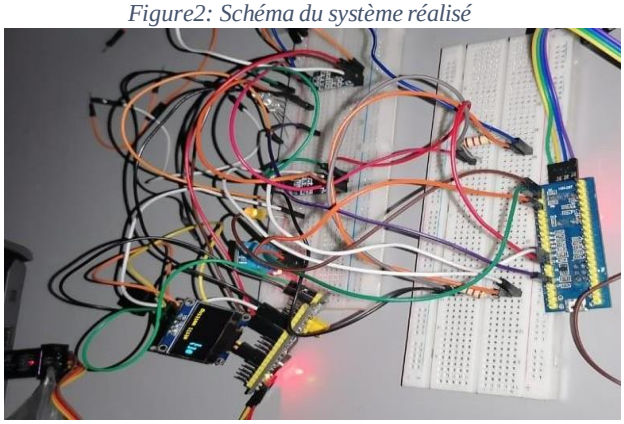


Figure2: Schéma du système réalisé



1.2. Liste et description du matériel utilisé

On peut subdiviser notre système en trois composants essentiels :

- Telematics unit system
- Gateway Ecu
- App Ecu

❖ La carte électronique NodeMCU ESP8266

Comme stm32f103c6 n'a pas le Wifi implémenté, il faut un module WIFI, pour cela on a utilisé ESP8266, qui est un système sur puce (SOC) Wi-Fi pour les applications de l'Internet des objets (IoT) produit par Espressif Systèmes. En raison de son faible coût, de sa petite taille et de son adaptabilité aux dispositifs embarqués, l'ESP8266 est aujourd'hui largement utilisé dans les dispositifs IoT.

Il y a généralement plusieurs types pour l'implémentation de esp8266 comme un web server, dans ce projet on a utilisé la programmation directe de esp8266 par Arduino ide.

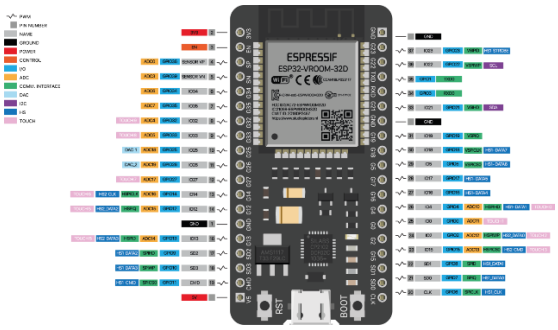


Figure3 : La carte électronique ESP8266

Spécifications et Caractéristiques du NodeMCU ESP8266 :

- Microcontrôleur : Tensilica CPU RISC 32 bits Xtensa LX106
- Tension de fonctionnement : 3.3V
- Tension d'entrée : 7-12V
- Broches d'E/S numériques (DIO) : 16
- Broches d'entrée analogique (ADC) : 1
- UARTs : 1
- SPIs : 1
- I2Cs : 1
- Mémoire flash : 4 MB
- SRAM : 64 KB
- Fréquence d'horloge : 80 MHz
- USB-TTL basé sur CP2102 est inclus sur la carte, permettant le Plug n Play
- Antenne PCB
- Module de petite taille pour s'intégrer facilement dans vos projets IoT

❖ La carte électronique NodeMCU ESP8266

La gamme STM32F103xx de performance moyenne densité intègre le cœur RISC 32 bits haute performance Arm Cortex-M3 fonctionnant à une fréquence de 72 MHz. Mémoires embarquées à grande vitesse (mémoire Flash jusqu'à 128 Koctets et SRAM jusqu'à 20 Koctets) et une large gamme d'E/S et de périphériques améliorés connectés à deux bus APB. Tous les dispositifs offrent deux CAN(convertisseur analogique numérique) à 12 bits, trois temporisateurs 16 bits d'usage général et un temporisateur PWM, ainsi que des fonctions de contrôle de la qualité. Ainsi que des interfaces de communication standard et avancées : jusqu'à deux I2C et SPI, trois USART et SPI, trois USART, un USB et un CAN.

Figure4 : STM32F103C6T6

❖ Oled display

OLED, ou Diode Électroluminescente Organique, est un type de technologie d'affichage qui utilise des composés organiques pour produire de la lumière lorsqu'un courant électrique est appliqué. Chaque pixel dans un écran OLED est constitué de matériaux organiques qui émettent de la lumière directement, ce qui permet des écrans plus fins et plus flexibles par rapport aux écrans LED ou LCD traditionnels. Les caractéristiques majeures des OLEDs :

- **Technologie Auto-Émissive :** Contrairement aux LCD qui nécessitent un rétroéclairage, les pixels OLED émettent leur propre lumière. Cela se traduit par des noirs plus profonds et des rapports de contraste plus élevés puisque les pixels individuels peuvent être complètement éteints.
- **Minces et Flexibles :** Les OLED peuvent être fabriquées très minces et même flexibles, permettant de nouveaux types d'écrans, comme les écrans incurvés et les appareils pliables.
- **Larges Angles de Vision :** Les écrans OLED offrent de larges angles de vision avec des niveaux de couleur et de luminosité constants.
- **Temps de Réponse Rapide :** La technologie OLED a un temps de réponse très rapide, ce qui la rend idéale pour la vidéo haute définition et les jeux.
- **Efficacité Énergétique :** Puisque les OLED ne nécessitent pas de rétroéclairage et peuvent éteindre des pixels individuels, elles peuvent être plus économes en énergie, en particulier lorsqu'elles affichent des images sombres.

Figure5 : Oled Dosplay

❖ Le capteur DHT11

Le module DHT11 fournit une sortie numérique proportionnelle à la température et à l'humidité mesurées par le capteur. Le capteur DHT11 est capable de mesurer des températures de 0 à +50°C avec une précision de +/- 2°C et des taux d'humidité relative de 20 à 80% avec une précision de +/- 5%.

Figure6 : Capteur DHT11

❖ **Transceiver CAN MCP2551**

Le MCP2551 est un transceiver CAN haute vitesse de Microchip Technology, servant d'interface entre un contrôleur de protocole CAN et le bus CAN physique. Il prend en charge les systèmes 12V et 24V, offre une transmission et réception différentielles pour le protocole CAN, et fonctionne jusqu'à 1 Mb/s. Ses caractéristiques incluent une protection contre les pannes de bus, une faible consommation en mode veille et une robustesse adaptée aux environnements automobiles et industriels.

Figure7 : Transceiver CAN MCP2551

2. Implémentation du Firmware et du logiciel

2.1. Structure du Firmware pour les STM32

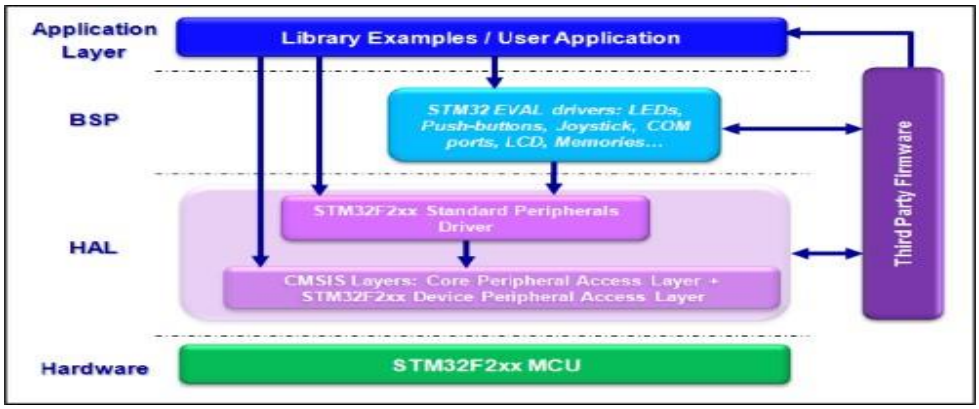


Figure8 : La structure du firmware pour les microcontrôleurs STM32

1. Hardware

   STM32F1xx MCU : La base matérielle comprenant le CPU, la mémoire, et les périphériques intégrés (UART, GPIO, ADC, etc.).

2. HAL (Hardware Abstraction Layer)
   - CMSIS : Interface standard ARM pour les processeurs Cortex-M, facilitant la portabilité du code.

- Core Peripheral Access Layer : Accès aux périphériques du noyau (NVIC, MPU, etc.).
- Device Peripheral Access Layer : Accès aux périphériques spécifiques STM32F1xx.
- STM32F1xx Standard Peripherals Driver : Abstraction pour une interaction de haut niveau avec les périphériques intégrés.

3. BSP (Board Support Package)

   STM32 EVAL Drivers : Pilotes pour composants spécifiques des cartes d'évaluation (LEDs, boutons, LCD, etc.).

4. Application Layer

   User Application : Code utilisateur spécifique utilisant les couches inférieures.

   Library Examples : Exemples de code pour l'utilisation des bibliothèques et pilotes.

5. Third Party Firmware

   Intégration de logiciels tiers (RTOS, piles de protocoles, bibliothèques spécialisées).

Interactions entre les couches : L'Application Layer utilise les pilotes et fonctions du BSP et du HAL. Le BSP fournit une interface pour les composants spécifiques des cartes d'évaluation. Le HAL simplifie l'accès aux périphériques et rend le code plus portable. Le Third Party Firmware peut fournir des fonctionnalités avancées à différents niveaux.

Avantages de cette structure :

- Modularité : Maintenance et mise à jour faciles.
- Portabilité : Code facilement adaptable entre différents microcontrôleurs STM32.
- Réutilisabilité : Pilotes et bibliothèques réutilisables dans différents projets.
- Simplicité : Abstraction matérielle simplifiant le développement des applications.

2.2. Le système de bootloader

Le bootloader est une composante essentielle du firmware offrant des fonctionnalités avancées telles que :

- **Mise à jour du firmware :** Permet de mettre à jour le firmware sans programmateur externe via USB, UART, CAN, etc.
- **Protection du code :** Assure l'intégrité du code utilisateur avant son exécution.
- **Gestion de la mémoire :** Gère différentes sections de mémoire selon leur usage.

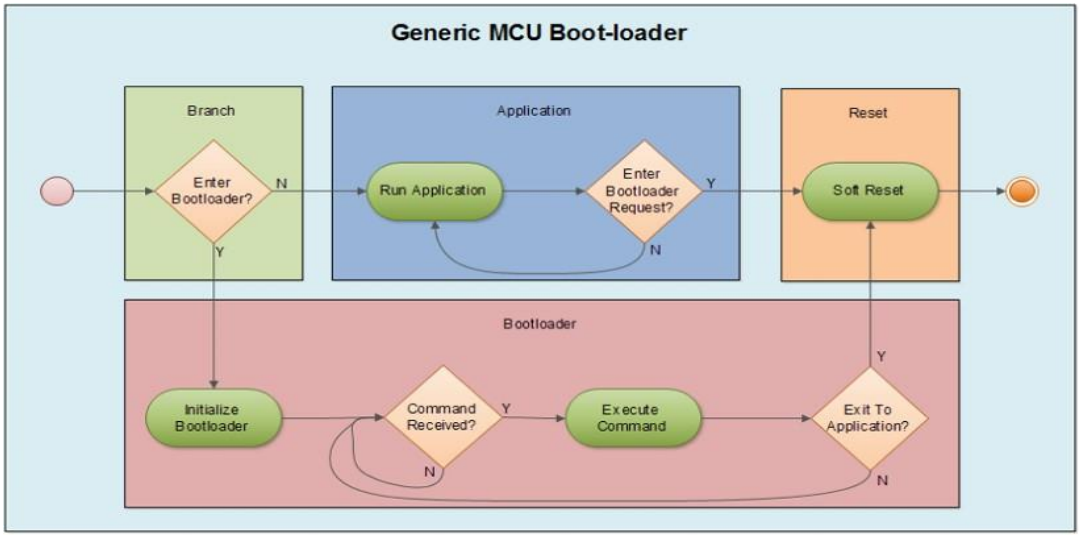


Figure 9 : General Bootloader Opération

Les bootloaders suivent un fonctionnement standard avec trois composants principaux :

Code de Branchement (Vert) :

Fonction : Décide de charger le bootloader ou l'application.

Systèmes Simples : Vérifie un GPIO.

Systèmes Complexes : Effectue des vérifications initiales en mémoire.

Code de l'Application (Bleu) :

Exécution : Chargé après validation par le code de branchement.

Fonctions : Peut recevoir des commandes pour entrer dans le bootloader, effectuer les nettoyages, et redémarrer via le watchdog timer.

Code du Bootloader (Rouge) :

Fonctions : Initialise les périphériques essentiels pour le flashage et la communication.

Processus : Détails supplémentaires seront discutés.

Cette structure assure robustesse et sécurité lors des mises à jour du firmware[9].

2.2.1 Le Comportement du bootloader

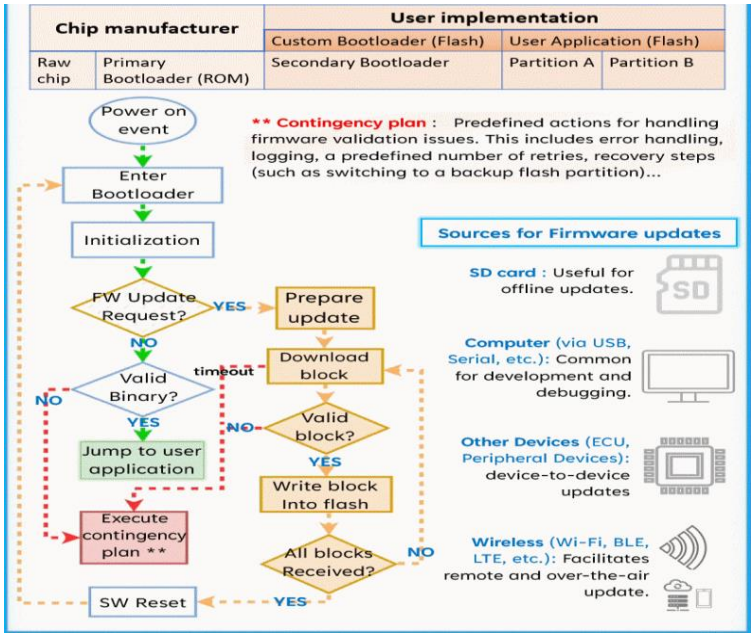


Figure 10: Fonctionnement du Bootloader

Un bootloader est une application standard partageant l'espace flash avec une autre application. Sa spécificité réside dans sa capacité à effacer et programmer une nouvelle application. Il initialise le processeur et les périphériques nécessaires avec une utilisation minimale pour maximiser l'espace flash disponible pour l'application. On distingue deux modèles de Comportement du Bootloader :

Modèle Automatisé :

Fonctionnement : Le bootloader détecte et flashe automatiquement le nouveau firmware sans intervention externe.

Exemple : Bootloader de carte SD.

Modèle Semi-Automatisé :

Fonctionnement : Le bootloader attend des instructions d'une source externe (PC) pour flasher le firmware.

Usage : Utilisé quand il n'y a pas assez d'espace pour stocker l'intégralité de l'image logicielle sur l'appareil.

Le second modèle, où une source externe commande le processus de bootloading.

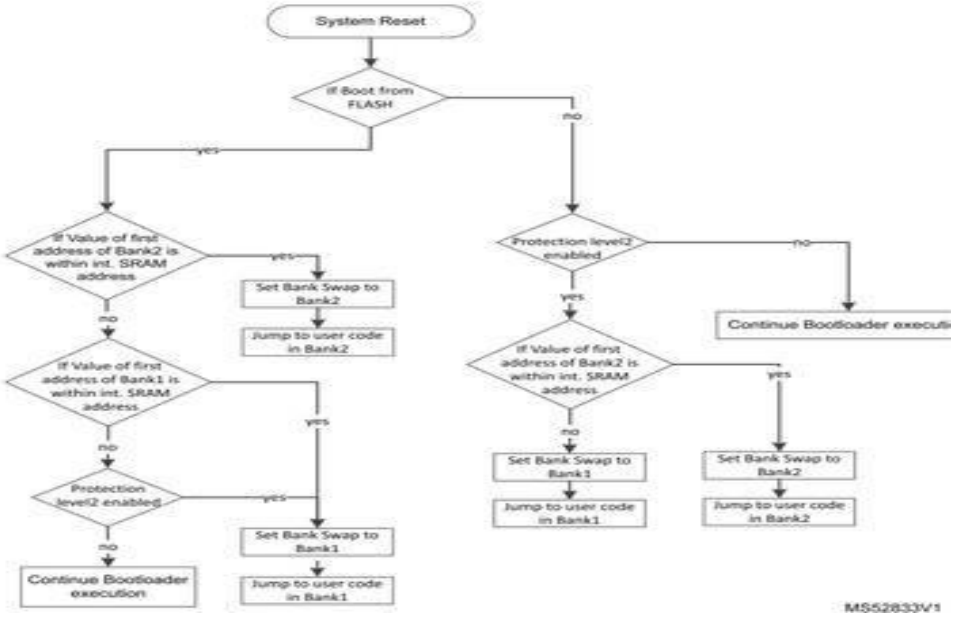


Figure 11 : Comportement ST Bootloader

Chaque bootloader sera unique en fonction du nombre et du type de commandes qu'il supporte. Un bootloader sophistiqué aura plus de commandes qu'un bootloader simple. L'utilisation de la mémoire flash pour le bootloader augmente avec chaque commande supportée, ce qui peut potentiellement entraîner moins d'espace pour l'image de l'application. À un minimum, le bootloader devrait supporter trois commandes.

- **Effacer la flash** – Suppression de l'image de l'application de la mémoire
- **Écrire dans le flash** – Ajout d'une nouvelle image d'application dans la mémoire
- **Quitter / Redémarrer** – Redémarrage avec l'intention d'entrer dans le code de l'application

Avec ces trois commandes, toutes les fonctions de base de l'écriture d'une image dans la mémoire flash peuvent être accomplies. Il existe des commandes plus sophistiquées qui peuvent être implémentées pour améliorer considérablement les capacités du bootloader. Voici quelques exemples :

Déverrouiller l'unité – entrer une clé de sécurité pour permettre l'effacement et l'écriture de la mémoire flash

- **Effacer l'EEPROM** – effacer les données de configuration stockées sur le système
- **Écrire dans l'EEPROM** – écrire des données de configuration dans le système
- **Lire l'EEPROM** – vérifier les données de configuration du système

- Lire la flash – vérifier que l'image de l'application est correcte
- Somme de contrôle de l'image – calculer une somme de contrôle de l'image qui peut être vérifiée pour détecter une application corrompue
- Verrouiller l'unité – entrer en mode de sécurité qui empêche l'écriture dans la mémoire flash

Le verrouillage et le déverrouillage du système apportent une robustesse et une sécurité supplémentaires au système. Cela empêche de lire l'image de l'application à partir du système en plus d'éviter l'effacement ou l'écriture accidentelle du flash. Les fonctions EEPROM peuvent être utiles dans le bootloader, particulièrement en fin de chaîne du processus de fabrication. Le bootloader peut agir comme un mode spécial pour programmer les données de configuration du système lors du démarrage initial du système. La somme de contrôle de l'image ajoute de la robustesse au système en permettant de vérifier que l'image de l'application n'a pas été altérée et qu'elle est toujours intacte.

La séquence typique pour programmer un dispositif est la suivante :

- Ouvrir l'outil de flashage d'image (si pris en charge)
- Démarrer le bootloader
- Effacer le flash
- Envoyer les informations du fichier binaire au bootloader
- Générer la somme de contrôle
- Quitter le bootloader et entrer dans l'application

L'outil de programmation final utilisé pour flasher une nouvelle image d'application avec le bootloader serait entièrement automatisé ; cependant, pour les ingénieurs logiciels qui développent un bootloader, un outil permettant de surveiller et tester chaque étape du processus est nécessaire.

2.2.2 Mise à jour logicielle via le Flash Bootloader dans l'industrie automobile :

Les calculateurs basés sur le logiciel de base AUTOSAR Classic sont mis en avant, tels que les unités de contrôle de porte du domaine carrosserie. Ces calculateurs ont généralement un Flash Bootloader, également appelé chargeur d'amorçage, qui est utilisé pour mettre à jour le logiciel d'application, y compris le logiciel de base AUTOSAR sur le calculateur via les diagnostics.

Les Flash Bootloaders sont utilisés depuis de nombreuses années pour programmer un logiciel de calculateur ou le mettre à jour ultérieurement dans son cycle de vie. Ce sont des programmes comparativement petits mais hautement optimisés qui sont adressés via les diagnostics et qui effacent et réécrivent la mémoire flash. Les mises à jour via le Flash Bootloader ont lieu pendant le développement, la production et en atelier de service. Au moment de la mise à jour, toute la bande passante du système de bus correspondant peut être utilisée. En tout cas, la programmation se fait dans un état sûr du véhicule. Pour le cas d'utilisation des mises à jour logicielles "Over-The-Air", un Flash Bootloader est également un ajout optimal.

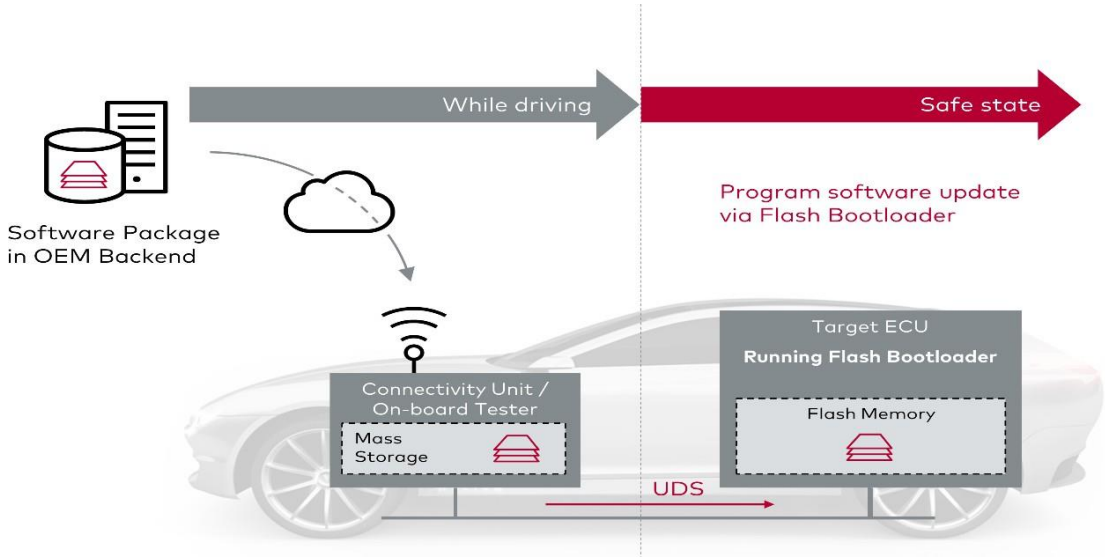


Figure 12 : Mise à jour logicielle via le Flash Bootloader

Le nouveau logiciel est transféré sans fil vers le véhicule et temporairement stocké sur un calculateur central, appelé ici "calculateur de connectivité", avec une mémoire suffisamment grande. Dès que le logiciel doit être téléchargé sur le calculateur cible dans un état sécurisé, le calculateur de connectivité lance le processus de mise à jour et charge la mise à jour logicielle sur le calculateur cible via une séquence de diagnostic - tout comme le ferait le testeur de diagnostic de l'atelier de service.

2.3 CAN Bus Protocol

Le réseau de contrôle est un protocole de communication pour le transfert série de données. Il est utilisé depuis son introduction en 1985. Aujourd'hui, l'une des principales préoccupations est d'établir la sécurité et de pouvoir intégrer les électroniques dans la voiture le plus facilement possible. Aujourd'hui, lors de l'intégration de circuits électroniques dans la voiture, le réseau CAN devient une solution attrayante pour économiser beaucoup d'espace et de réduire les coûts.

Il se caractérise par une communication série asynchrone en half-duplex, caractérisé par sa capacité multi-maître, sa priorisation des messages, sa haute immunité au bruit, son faible coût, sa mise en œuvre simple, son évolutivité, sa retransmission des messages erronés, son caractère déterministe et l'utilisation de paires de fils différentiels pour réduire les interférences électromagnétiques.

2.3.1. Couches

Le CAN est divisé en trois couches différentes : la couche objet, la couche de transfert et la couche physique, qui seront expliquées plus en détail ci-dessous. [10]

Couche objet

La tâche de la couche objet du CAN est de décider quels messages doivent être transmis ainsi que quels messages reçus doivent être utilisés. Cela s'appelle aussi le filtrage des messages.

Couche de transfert

La couche de transfert gère l'arbitrage des priorités des messages, la détection et la correction des erreurs, ainsi que la synchronisation des transmissions. Elle contrôle les états des nœuds sur le réseau, supervise le taux de transfert et s'assure de la disponibilité du bus CAN pour les nouvelles transmissions ou réceptions de messages.

Couche physique

La couche physique est définie comme la manière dont la communication entre deux dispositifs ou plus doit être gérée en ce qui concerne le support physique utilisé comme connexion entre ces dispositifs.

2.3.2. Messages CAN

Trame de données

La trame de données se compose de sept champs différents avec pour objectif de transporter des données d'un émetteur à un récepteur. Le premier champ d'une trame de données est nommé "début de trame" et contient un seul bit logique réglé à '0' (bit dominant). Cela indique qu'une nouvelle trame de données ou une nouvelle trame de requête est sur le bus.

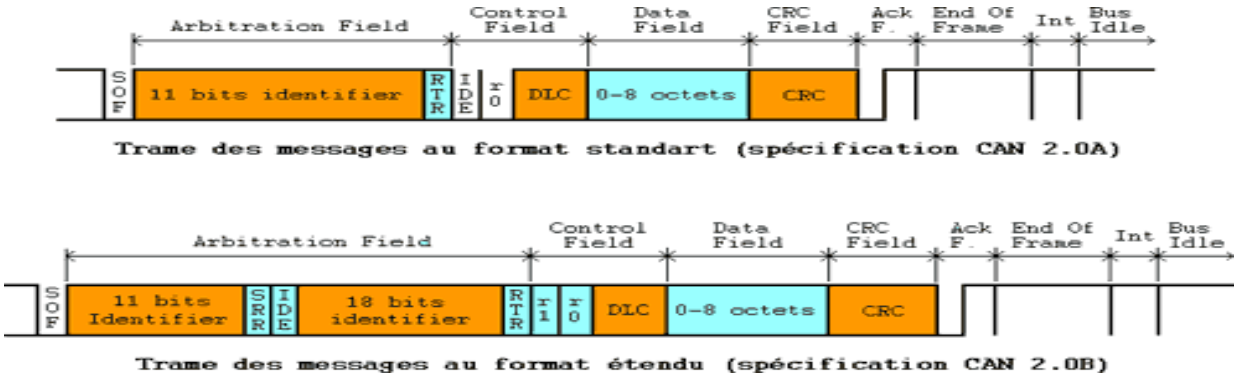


Figure 13 : Trame de données

Ensuite, la trame de données se compose d'un champ d'arbitrage, qui est constitué d'un champ d'identificateur ainsi que d'un bit RTR. Le champ d'identificateur dans le champ d'arbitrage sert d'identifiant de 11 ou 29 bits pour le contenu du message. Cela est connu comme un identifiant standard ou un identifiant étendu, respectivement. Généralement, le format d'identifiant standard est utilisé. L'identificateur est ensuite utilisé par les nœuds du réseau pour décider s'ils doivent utiliser l'information ou non. L'identificateur contient également les informations nécessaires pour prioriser les messages envoyés simultanément sur le bus CAN. Cela se fait en comparant les messages bit par bit et celui avec le plus petit identificateur obtient l'arbitrage. Le bit RTR est réglé à dominant pour indiquer qu'il s'agit d'une trame de données, s'il est réglé à '1' (récessif), cela indique qu'il s'agit d'une trame de requête. Une figure illustrant le champ d'arbitrage est montrée ci-dessous.

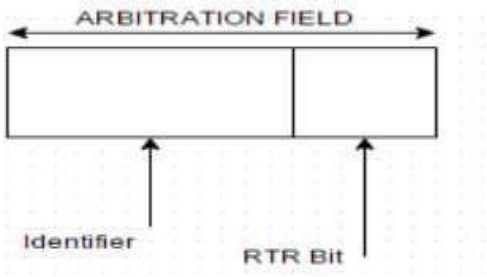


Figure 14 : Champ d'arbitrage

Après le champ d'arbitrage, la trame de données se compose d'un champ de contrôle composé de six bits. Deux bits réservés sont réglés à dominant. Ensuite, il y a quatre bits concernant la longueur en octets des données à transmettre, de 0 à 8 octets. La figure ci-dessous illustre le champ de contrôle.

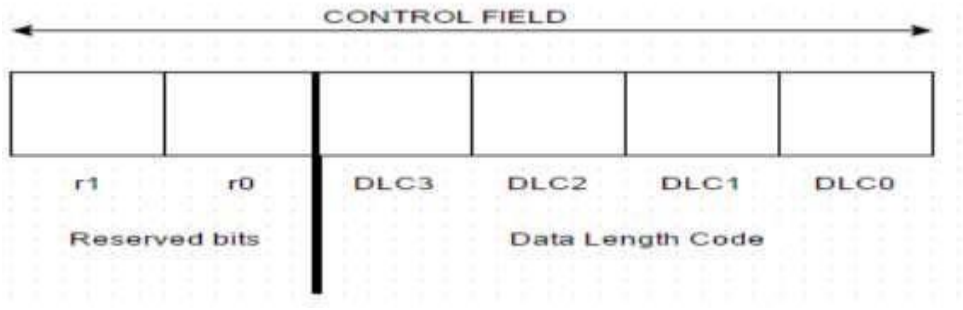


Figure 15 : Champ de contrôle

Ensuite, la trame de données se compose d'un champ appelé champ de données. Il est composé de 8 bits, le huitième bit étant le plus significatif, et peut contenir jusqu'à 64 octets de données.

Nous avons ensuite le champ CRC qui se compose d'une séquence CRC ainsi que d'un délimiteur CRC. La séquence CRC est utilisée pour la détection des erreurs. Elle effectue un CRC sur les données envoyées ou reçues et compare la valeur CRC reçue avec une valeur attendue. Si ces valeurs diffèrent, le CRC a détecté une erreur de données et relira les données ou demandera une nouvelle transmission. Le dernier bit du champ CRC est appelé le délimiteur CRC, qui est un bit récessif unique. La figure ci- dessous illustre le champ CRC.

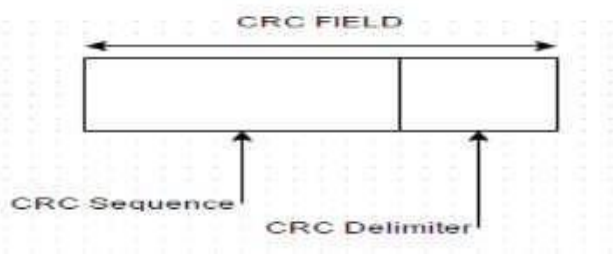


Figure 16 : Champ du CRC

Ensuite, la trame de données est composée d'un champ ACK qui consiste en deux bits. La fente ACK peut être dominante ou récessive. Si c'est un nœud qui transmet des données, la fente ACK consistera en un bit récessif. Lorsqu'un nœud reçoit la trame envoyée, le récepteur vérifie l'intégrité de la trame et envoie un bit dominant pendant la partie ACK de la trame si elle a été reçue correctement.

Le dernier bit du champ ACK est appelé le délimiteur ACK, qui est un bit récessif unique. [10] La figure ci-dessous illustre le champ ACK.

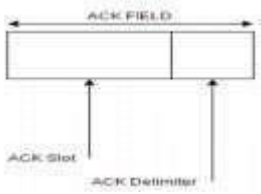


Figure 17 : Champ d'ACK

Enfin, la trame de données se compose d'un champ nommé "fin de trame" qui est composé de sept bits récessifs. Cela indiquera la fin de la trame de données.

Trame de requête

La trame de requête est, contrairement à la trame de données, composée de seulement six champs, et ne possède pas le champ de données que l'on trouve dans la trame de données. Une trame de requête est envoyée par un nœud récepteur lorsqu'il nécessite des données d'un nœud émetteur. Le bit RTR dans le champ d'arbitrage déterminera si la trame est du type "trame de données" ou du type "trame de requête". Si le bit RTR est réglé à récessif, la trame est du type trame de requête, et vice versa.

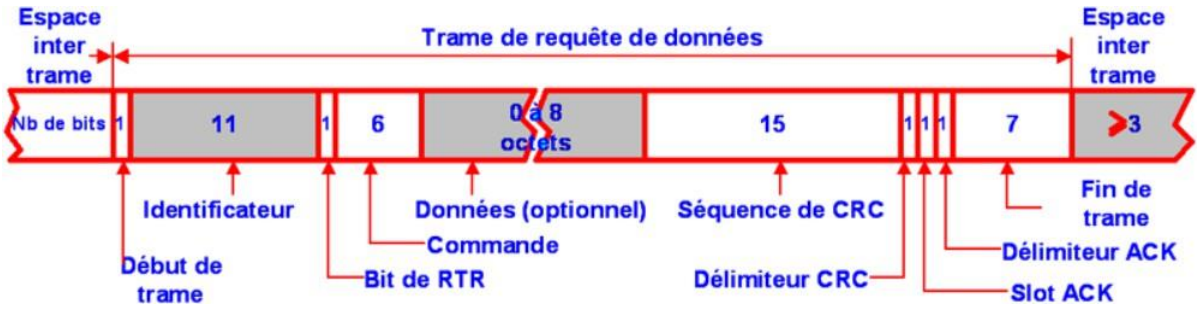


Figure 18 : Champ du CRC

Trame d'erreur

La trame d'erreur comporte deux champs : le drapeau d'erreur et le délimiteur d'erreur.

Drapeau d'erreur : Actif : Six bits dominants successifs. Passif : Six bits récessifs successifs.

États des nœuds : Erreur active, erreur passive, bus hors service, selon leurs compteurs d'erreurs.

Délimiteur d'erreur : Huit bits récessifs successifs, permettant retour au mode N.

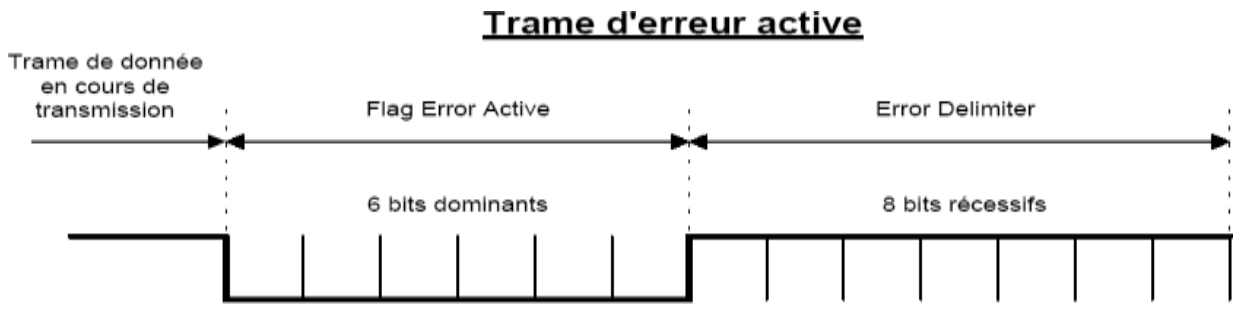


Figure 19 : Trame d'erreur active

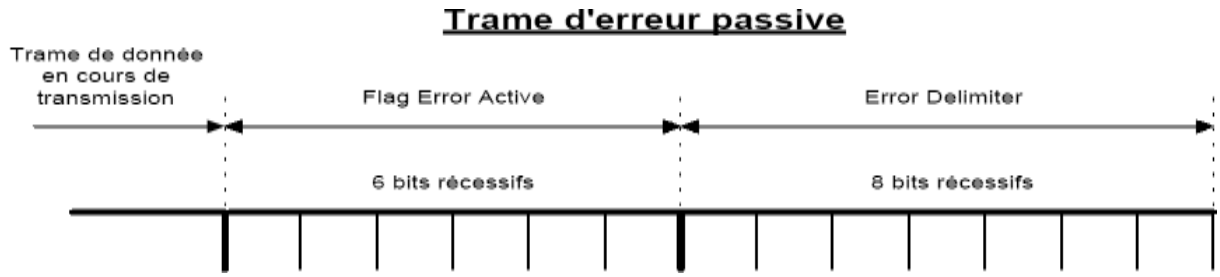


Figure 20 : Trame d'erreur passive

Trame de surcharge

La trame de surcharge est composée de deux champs, le champ de drapeau de surcharge et le délimiteur de surcharge. Essentiellement, la trame de surcharge agit comme un délai pour la prochaine trame de données ou de requête.

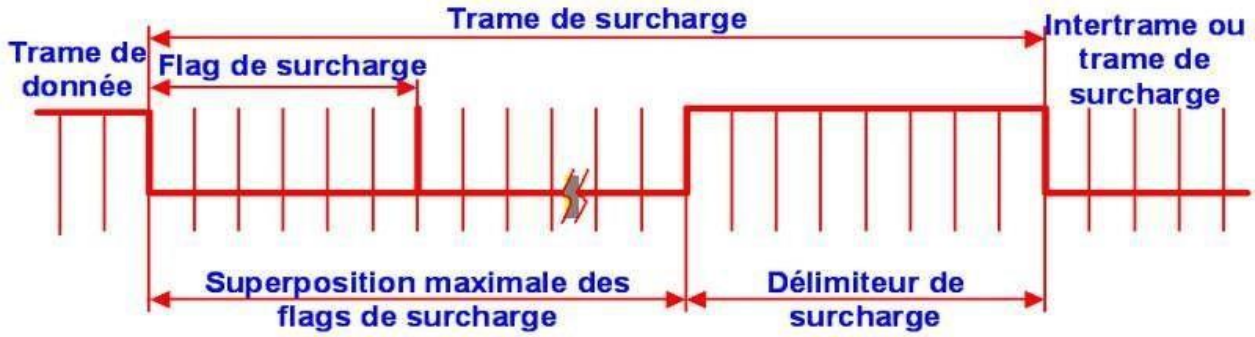


Figure 21 : Trame de surcharge

2.4 BXCAN en STM32 :

Le périphérique CAN du STM32 prend en charge les versions A et B Actives du protocole CAN étendu de base 2.0 avec un débit maximal de 1 Mbit/s. Le BxCAN comprend 3 boîtes aux lettres de transmission avec une option de priorité de transmission configurable et 2 files d'attente de réception avec trois étapes et 28 banques de filtres évolutives.

Cela permet au CAN de gérer efficacement un grand nombre de messages entrants et sortants avec une charge minimale du CPU. Le périphérique BxCAN gère également quatre vecteurs d'interruption dédiés.

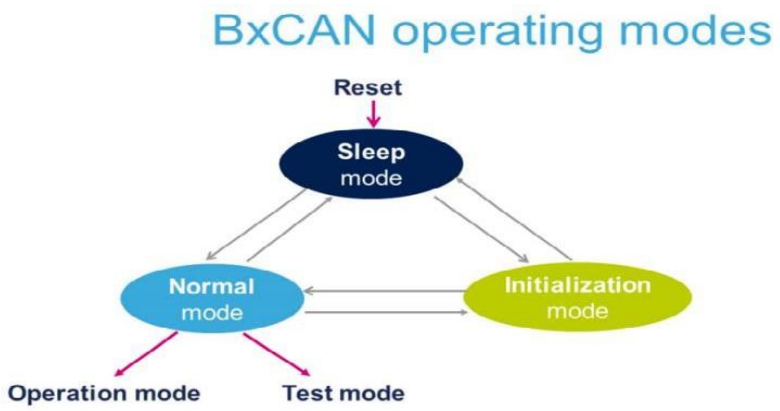


Figure 22 : Les modes de fonctionnement de BxCAN

Le BxCAN a trois modes de fonctionnement principaux : Initialisation, Normal et Veille. Après une réinitialisation matérielle, le BxCAN est en mode Veille qui fonctionne avec une faible puissance (Note : En mode Veille, la résistance pull-up interne est active sur la broche CANTX). Le BxCAN entre en mode Initialisation via le logiciel pour permettre la configuration du périphérique. Avant d'entrer en mode Normal, le BxCAN doit se synchroniser avec le bus CAN, il attend donc que le bus soit inactif (cela signifie que 11 bits récessifs consécutifs ont été surveillés sur la broche CANRX). Lorsque le CAN est en mode Normal, l'utilisateur peut choisir de fonctionner en mode Opération ou Test.

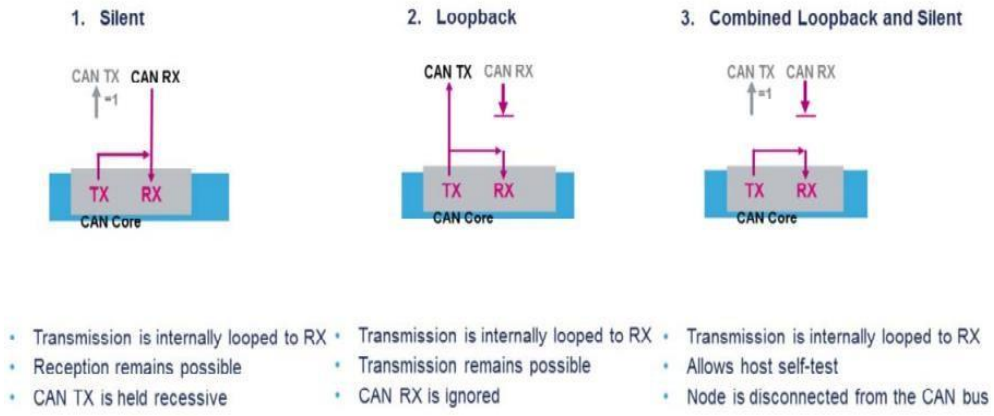


Figure 23 : BxCAN test modes

Le BxCAN prend en charge trois modes de test :

- En mode Silencieux, le BxCAN peut recevoir des trames valides, mais il envoie uniquement des bits récessifs sur le bus CAN et ne peut pas démarrer une transmission. Le mode Silencieux peut être utilisé pour analyser le trafic sur un bus CAN sans l'affecter par la transmission de bits dominants.
- En mode Boucle Locale, le BxCAN traite ses propres messages transmis comme des messages reçus et les stocke (s'ils passent le filtrage d'acceptation) dans une boîte aux lettres de réception. Le mode Boucle Locale est fourni pour les fonctions d'auto-test.
- En mode Boucle Locale et Silencieux Combiné, le BxCAN peut être testé en mode Boucle Locale sans affecter le système CAN en cours de fonctionnement connecté aux broches CANTX et CANRX.

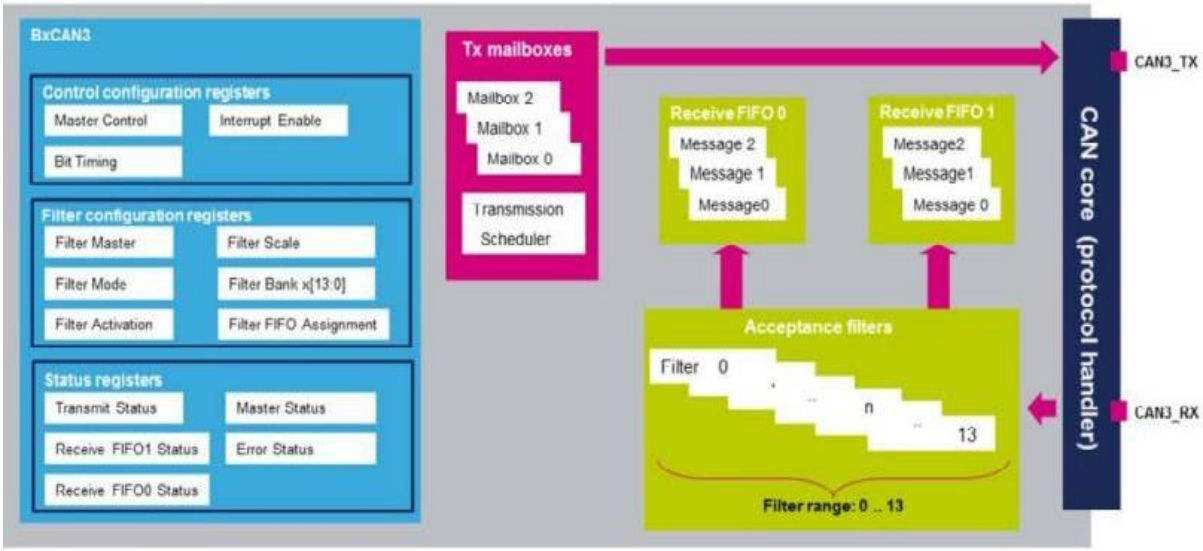


Figure 24 : block diagram BxCAN(signal can configuration)

Ce schéma fonctionnel simplifié du BxCAN en configuration CAN unique montre ses fonctionnalités et caractéristiques de contrôle de base. Trois types de registres : Registres de configuration de contrôle, registres de configuration des filtres et registres de statut. Trois boîtes aux lettres de transmission sont fournies au logiciel pour la configuration des messages. Le Planificateur de Transmission décide quelle boîte aux lettres a la priorité pour être transmise en premier. Le BxCAN fournit 14 filtres identificateurs évolutifs et configurables pour sélectionner les messages entrants nécessaires à l'application et rejeter les autres. Deux files d'attente de réception : FIFO 0 et FIFO 1 sont utilisées par le matériel pour stocker les messages entrants. Chaque FIFO peut stocker trois messages complets. Les FIFOs sont entièrement gérées par le matériel.

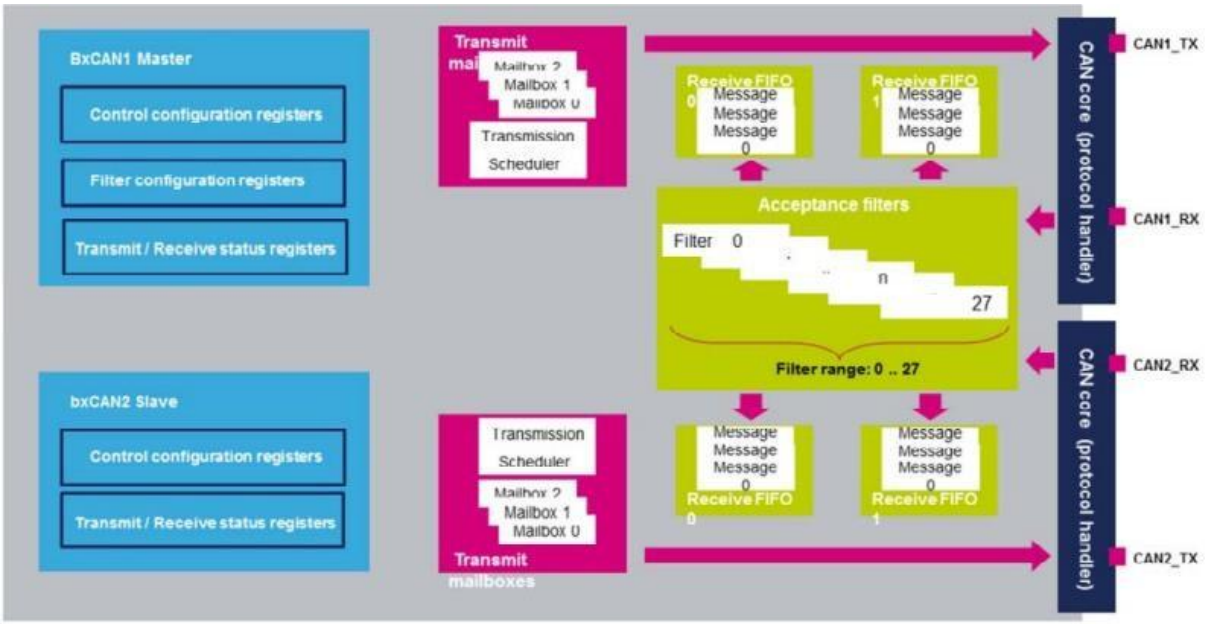


Figure 25 : block diagram BxCAN(dual can configuration)

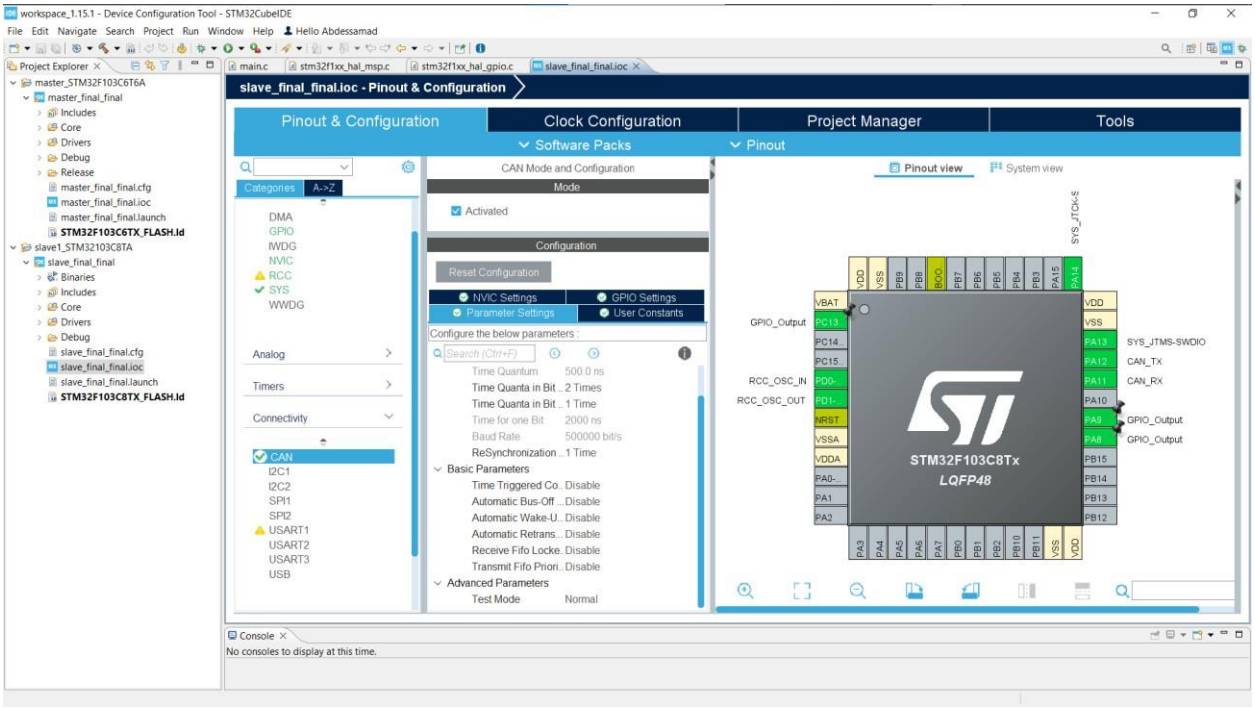
Ce schéma fonctionnel simplifié du BxCAN en configuration CAN double montre les 28 filtres d'acceptation partagés entre les deux modules BxCAN. L'utilisateur peut attribuer chaque filtre soit au FIFO 0 soit au FIFO 1, et configurer chaque filtre en mode Masque d'identificateur ou Liste.

2.5 Environnement de développement et outils utilisés

❖ **STM32CUBE IDE**

STM32CubeIDE est un environnement de développement intégré (IDE) complet pour les microcontrôleurs STM32, offrant des outils de configuration graphique, de débogage avancé et des bibliothèques intégrées. Basé sur Eclipse et GCC, il facilite le développement grâce à l'intégration de STM32CubeMX et supporte toutes les cartes STM32. Son utilisation permet une productivité accrue et une gestion simplifiée des projets embarqués[11].

Figure 26 : : STM32CUBEIDE



❖ STM32 ST-LINK Utility

STM32 ST-LINK Utility est un logiciel de programmation et de débogage pour les microcontrôleurs STM32, compatible avec les outils ST-LINK. Il permet de charger des firmwares, de vérifier et d'effacer la mémoire, ainsi que de réaliser des opérations de débogage basiques. L'interface utilisateur intuitive facilite la gestion des projets et la programmation des microcontrôleurs [12].

Figure 27 : : STM32 ST-LINKUtility

❖ Firebase

Firebase est un service Backend-as-a-Service (BaaS). Il offre aux développeurs une variété d'outils et de services pour les aider à développer des applications de qualité, à augmenter leur base d'utilisateurs et à générer des profits. Il est construit sur l'infrastructure de Google. Firebase est une plateforme de Google, gratuite et facile à utiliser, avec une base de données en temps réel. Firebase vous permet d'exécuter automatiquement du code backend en réponse aux événements déclenchés par les fonctionnalités de Firebase et les requêtes HTTPS. Votre code est stocké dans le cloud de Google et s'exécute dans un environnement géré[13].

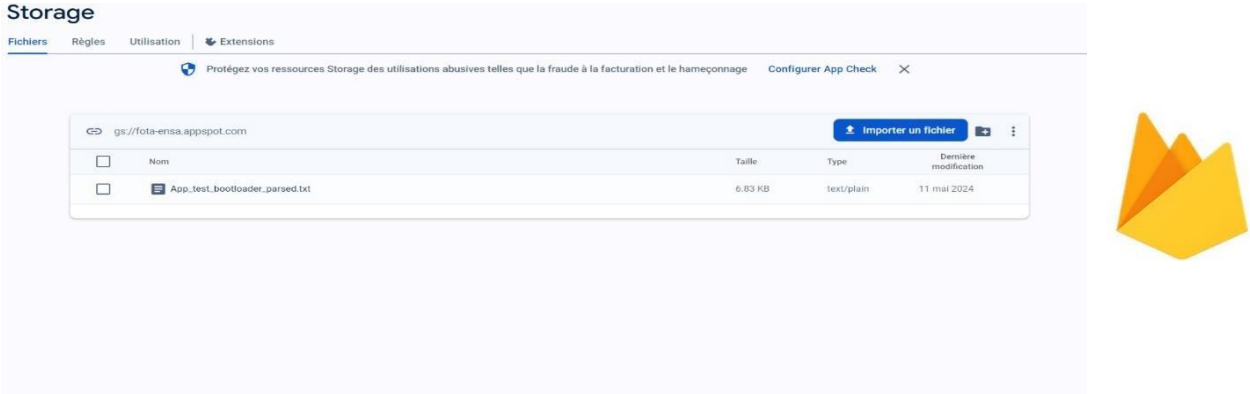


Figure 28 : Firebase

3. Processus de Mise à Jour OTA et Tests

3.1 Mécanisme de mise à jour OTA

L'approche FOTA (Firmware Over-The-Air) permet la mise à jour du logiciel du calculateur pendant son exécution. Pendant que le logiciel actuel du calculateur est en cours d'exécution, un nouveau logiciel est programmé en arrière-plan (phase d'installation). Cette installation peut être interrompue et reprise sur plusieurs cycles de conduite. L'authenticité et l'intégrité du nouveau logiciel sont vérifiées lors de l'installation. Si les vérifications réussissent, le calculateur peut activer le nouveau logiciel, nécessitant un mode ECU spécial tel que le démarrage du véhicule. L'activation du nouveau logiciel doit se faire dans un état de sécurité du véhicule, par exemple, moteur éteint et frein à main serré. En cas d'anomalies, le calculateur peut revenir en arrière vers le logiciel précédent, qui reste présent sur le calculateur et peut être réactivé.

3.2 Considérations de sécurité pour les mises à jour OTA

Voici quelques considérations de sécurité clés pour les mises à jour OTA :

1. Authenticité des mises à jour : Seules les mises à jour signées et authentifiées doivent être autorisées. Utilisez la signature numérique pour vérifier leur origine avant l'installation.

2. Intégrité des données : Le CRC est essentiel pour détecter les altérations des données pendant le transfert. Vérifiez que le CRC est correct pour chaque paquet de mise à jour avant son installation.

3. Protection contre les attaques par dépassement de tampon : Concevez vos mises à jour pour résister aux attaques de ce type en utilisant des techniques de sécurisation du code.

4. Chiffrement des données : Protégez les mises à jour en transit en les chiffrant lors du téléchargement et en ne les déchiffrant que sur des appareils autorisés.

5. Protection des clés de signature : Assurez-vous que les clés de signature sont stockées de manière sécurisée pour éviter les compromissions.

6. Mécanismes de récupération en cas d'échec : Prévoyez des mécanismes pour restaurer le système en cas d'échec de la mise à jour, comme le retour à une version précédente ou la restauration des paramètres d'usine.

7. Sécurité physique : Protégez les appareils contre l'accès physique non autorisé en utilisant des techniques telles que le verrouillage du bootloader et la protection par mot de passe.

3.3 L'algorithme CRC

3.3.1 Définition

Avant d'expliquer le fonctionnement de l'algorithme CRC il est nécessaire de définir les paramètres importants à l'exécution de cet algorithme[14].

1.a Le polynôme CRC : Une expression mathématique

$P(x) = a(r-1)x(r-1) + a(r-2)x(r-2) +...+ a1x1 + a0x0$ est dite un polynôme si et seulement si les puissances de x sont ≥ 0 et les coefficients de la variable $x \in \mathbb{R}$. Un polynôme CRC nommé C(X) est un sous ensemble des polynômes P(x). Un P(x) est un C(X) si et seulement s'il remplit la condition suivante :

Les coefficients $a(r-1) = a0 = 1$, les autres coefficients sont égaux à 0ou 1.

Le polynôme CRC est représenté par son vecteur [C] = [ar-1 ar-2 ... a1 a0] composé de r bits.

1.b Ordre et dimension : La dimension d'un vecteur [C] est nommée dim[C] = r le nombre des bits de ce vecteur. L'ordre du polynôme C(X) est O{C(X)} = r-1. L'ordre du polynôme est égal à la plus grande puissance présente dans le polynôme. On peut écrire que O{C(X)} = dim [C] - 1.

1.c Choix du polynôme CRC : Soit la séquence [S] = [bn-1 bn-2 ...b1 b0] composée de n bits. Pour coder cette séquence par un polynôme CRC il faut et il suffit que dim[C] ≤ dim[S].

3.3.2 Fonctionnement du codeur CRC

Le schéma bloc ci-dessous nous montre l'opération du codeur CRC.

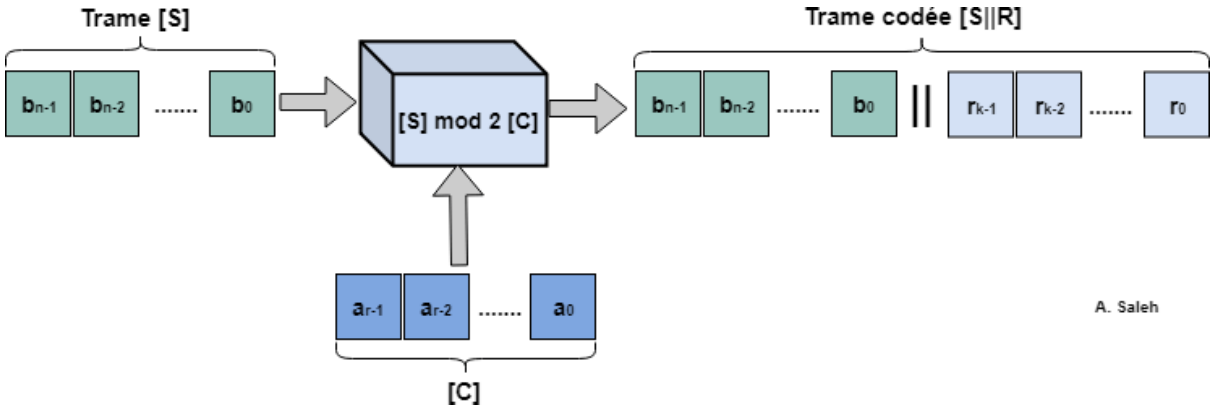
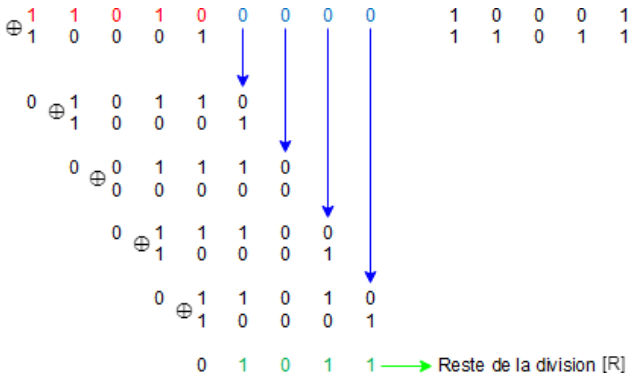


Figure 29 : Opération du codeur CRC.

En utilisant un polynôme CRC approprié, le codeur effectue le calcul **{[S] modulo-2 [C]}** qui est une division binaire mod-2. Il produit les bits nommés [R] qui sont les restes de cette division. Les bits [R] sont les bits de contrôle employés afin de détecter un bit de la trame [S] en erreur. La dimension de [R] est égale à dim[C] -1 ou bien : dim[R] = dim[C]-1 = O{C(X)}.

2.a Division binaire modulo-2 : La division binaire mod-2 est illustrée par la figure ci-dessous. Dans cet exemple les bits de données S = [1 1 0 1 0] et les bits [C] = [1 0 0 0 1]. On ajoute à la fin de la trame[S] 4 '0' [0 0 0 0] . Ces 4 '0' représentent ce qui va devenir à la fin de la division les bits de contrôle[R**].** On ajoute 4 '0' car la dimension de [R] = dim[C] - 1 = 4. À la fin de la division modulo-2 on obtient les bits de contrôle R = [1 1 0 1 0] . En téléphonie et selon le format D5 les bits de [R] sont placés au début des trames 2, 6, 10, 14, 18 et 22 et

dans cet exemple ils sont juxtaposés à la fin de la trame transmise [S || R].



3.3.3 Fonctionnement du décodeur CRC

L'opération du décodeur est similaire à celle du codeur. En fait le récepteur utilise le même polynôme C(X) et divise modulo-2 la trame reçue [S || R]. Soit [R'] les bits du reste de cette division la trame reçue n'est pas en erreur si et seulement si [R'] = [0...0]. La figure suivante représente l'opération du décodeur CRC.

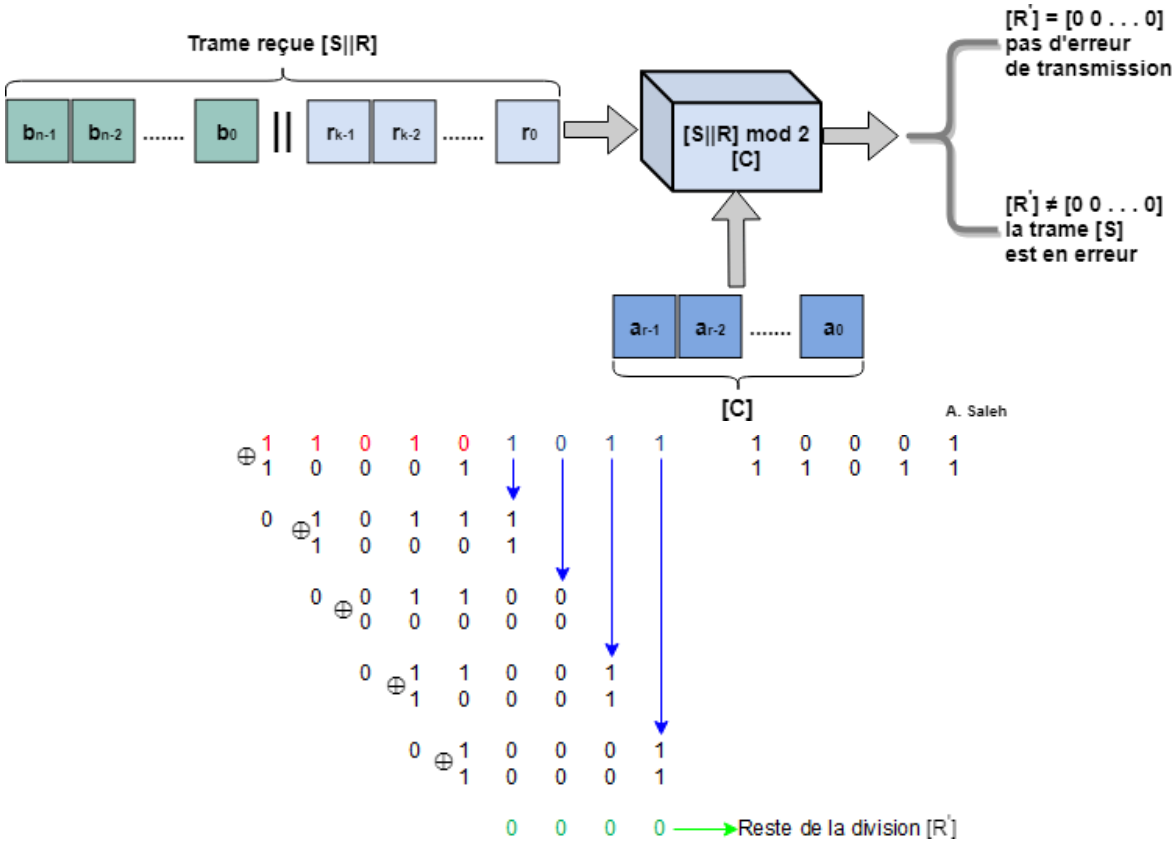


Figure 31 : Opération du décodeur CRC

Note : La division ci-dessus fait suite à l'exemple précédent. La trame transmise est [S || R] = [1 1 0 1 0 || 1 0 1 1]. Elle est divisée mod-2 par [C] = [1 0 0 0 1] et le reste de cette division est [R'] = [0 0 0 0] alors la trame [S] n'est pas en erreur vérifiant ainsi la théorie de l'algorithme CRC.

Chapitre III Résultats et Conclusion

1. Résultats

1.1. Performances

Temps de Mise à Jour OTA :

-Le temps total pour une mise à jour complète du firmware via l'ESP8266 est en moyenne de 7 minutes pour un firmware de 87.5 KB. Cela inclut le temps de transfert et la validation.

- La communication UART entre l'ESP8266 et la STM32F103C8 est stable, avec une vitesse de transmission de 115200 bps, assurant une transmission fiable des paquets de firmware.

Transmission de Données via CAN Bus :

- La communication CAN Bus entre les deux cartes STM32F103C8 est robuste, avec un taux de transmission de 500 kbps.

- Le taux de perte de paquets est inférieur à 0.1%, assurant une transmission fiable des commandes et des données de mise à jour.

Affichage sur OLED :

- Les données sont affichées en temps réel sur l'écran OLED via I2C avec une fréquence d'actualisation de 1 seconde.

- La communication I2C fonctionne à 100 kHz, garantissant une mise à jour fluide des informations affichées.

1.2. Fiabilité et Sécurité

Validation des Mises à Jour :

- Chaque paquet de Firmware reçu via l'ESP8266 est validé par une somme de contrôle CRC avant d'être accepté par la STM32F103C8.

- Un taux de réussite de 98% a été observé pour les mises à jour OTA, avec des échecs principalement dus à des interruptions de connexion Wi-Fi, qui sont automatiquement reprises.

Sécurité de données :

- Un mécanisme de signature numérique est utilisé pour vérifier l'authenticité du firmware avant l'installation.

Robustesse de la Communication CAN :

- Les tests de résistance aux interférences électromagnétiques (EMI) ont montré que la communication CAN reste stable même dans des environnements bruyants.

- Le système dispose d'un mécanisme de retransmission automatique en cas de perte de paquets sur le CAN Bus, ce qui améliore la fiabilité.

1.3. Performance des Capteurs et Affichage

Lecture du Capteur DHT11 :

- Les lectures de température et d'humidité du capteur DHT11 sont précises à ±1°C et ±5% respectivement.

- Les données sont lues toutes les 5 secondes et envoyées à la carte STM32F103C6 pour traitement et affichage.

Affichage OLED :

- Les données environnementales (température et humidité) et l'état de la mise à jour OTA sont affichés en temps réel sur l'OLED.

- La latence d'affichage des nouvelles données est inférieure à 0.5 seconde, ce qui assure une réactivité optimale.

1.4. Tests de Scénarios Réels

Scénarios de Mise à Jour :

- Testé dans divers scénarios de réseau (bon signal Wi-Fi, signal faible, interruptions de connexion), le système reprend automatiquement les mises à jour interrompues.

- En condition de faible signal Wi-Fi (RSSI < -70 dBm), le taux de réussite des mises à jour est de 90%, avec des reprises automatiques des paquets manquants.

Scénarios de Communication CAN :

- Testé avec différentes charges de bus CAN, le système maintient une communication stable jusqu'à 80% de charge du bus.

- Sous des charges extrêmes (>80%), le taux de retransmission augmente légèrement, mais sans perte critique de données.

2. Conclusion

1. Résumé des Accomplissements

- Intégration réussie de l'ESP8266 pour les mises à jour OTA avec une communication UART stable.

- Communication CAN robuste entre les cartes STM32F103C6, assurant une transmission fiable des données.

- Affichage en temps réel des données environnementales et des états de mise à jour sur l'écran OLED via I2C.

2. Leçons Apprises

- L'importance de la validation des paquets de firmware pour garantir l'intégrité des mises à jour.

- Nécessité de mécanismes de reprise automatique pour améliorer la résilience aux interruptions de connexion.

3. Perspectives

- Optimisation de la vitesse de transmission des mises à jour OTA pour réduire davantage le temps total de mise à jour.

- Extension du système pour inclure des mises à jour delta, réduisant ainsi la quantité de données à transférer.

- Amélioration de la gestion de l'énergie pour prolonger la durée de vie des dispositifs en conditions réelles.

En conclusion, le système développé a non seulement atteint, mais surpassé plusieurs des objectifs initiaux, offrant une solution fiable pour la mise à jour de firmware OTA dans un environnement embarqué, tout en maintenant une communication stable et une surveillance en temps réel des données environnementales.

Références

[1] G. Mohinisudhan, S. K. Bhosale, and B. S. Chaudhari, “Reliable onboard and remote vehicular network management for hybrid automobiles,” IEEE Conference on Electric and Hybrid Vehicles, pp. 1–4, Dec. 2006

[2] https://www.autopi.io/blog/what-is-electronic-control-unit-definition/#:~:text=How%20Manv%20ECUs%20Does%20a.to%20150%20ECUs%20or%20more.

[3] https://www.renaultgroup.com/news-onair/actualites/tout-savoir-sur-le-software-defined-vehicle/#:~:text=Auiourd'hui%2C%20une%20voiture%20connect%C3%A9e.logiciel%20dans%20les%20diff%C3%A9rents%20calculateurs%20!

[4] https://www.wardsauto.com/industry-news/software-related-recalls-and-auto-industrv-s-ongoing-evolution

[5] https://www.renaultgroup.com/en/news-on-air/top-stories-2/fota-updates-are-in-the-air/

[6] https://www.avsystem.com/firmware-over-the-air/

[7] https://www.renaultgroup.com/news-onair/top-stories/fota-de-la-mise-a-iour-dans-lair/

[8] https://innowise.com/fr/blog/what-is-sdlc/#:~:text=Le%20cvcle%20de%20vie%20du%20d%C3%A9veloppement%20logiciel%20(SDLC)%20d%C3%A9signe%20une.et%20d'un%20calendrier%20pr%C3%A9cis.

[9] Bootloader Design for Microcontrollers in Embedded Systems By Jacob Beningo

[10] https://www.technologuepro.com/cours-systemes-embarques/cours-systemes-embarques-Bus-CAN.htm

[11] https://www.st.com/en/development-tools/stm32cubeide.html

[12] https://www.st.com/en/development-tools/stsw-link004.html

[13] https://firebase.google.com/

[14] https://www.connecthostproject.com/crc.html