Département EEA - Faculté Sciences et Ingénierie

Master STS - EEAS

Électronique Énergie Automatique Signal

Parcours ISTR

Année 2023-2024

Rapport de stage de fin d'études
Développement d'un Flash Bootloader Logger

Rédigé par :
Mondher Abderaouf KHEMISSI

Référent pédagogique :
Alexis GENERES

Référent entreprise :
Mohamed EL KATI

11 juillet 2025

Remerciements

Je tiens à exprimer ma profonde gratitude à l'ensemble des personnes qui ont contribué à la réussite de ce stage.

Tout d'abord, je remercie sincèrement mon maître de stage, Mohamed El KATI, pour son encadrement, sa disponibilité et ses précieux conseils tout au long de ce projet. Son expertise et son soutien ont grandement facilité mon intégration et l'accomplissement de mes missions.

Je souhaite également remercier chaleureusement Aurélien BRABAN, l'architecte de l'équipe, pour son accompagnement et ses orientations techniques, qui ont été déterminantes pour la réussite du projet.

Un grand merci à Hocine KENOUNI et Aymen MOUGAIDA, dont l'aide précieuse et les échanges constructifs ont été d'une grande importance pour le bon déroulement de mon stage.

Enfin, je souhaite exprimer ma reconnaissance envers mes enseignants et professeurs à l'Université Paul Sabatier pour m'avoir permis de développer une curiosité intellectuelle ainsi que pour m'avoir offert les outils et les bases nécessaires pour mener à bien ce projet. Je suis également reconnaissant envers ma famille et mes amis pour leur soutien et leurs encouragements constants tout au long de cette aventure académique.

À toute l'équipe ex-ROGUE ONE, merci pour l'accueil chaleureux, la coopération, et l'ambiance de travail stimulante qui ont rendu cette expérience à la fois enrichissante et mémorable.

Table des matières

Acronymes

ABS Anti-lock Braking System
API Application Programming Interface
BLE Bluetooth Low Energy
BRS Bite Rate Switch
CAN Controller Area Network
CFG Control Flow Graph
CPU Central Processing Unit
CRC Cyclic Redundancy Check
DLC Data Length Code
DMA Direct Memory Access
DWT Data Watchpoint and Trace
ECU Electronic Control Unit
EEPROM Electrically Erasable Programmable Read Only Memory
FBL Flash BootLoader
FEE Flash Emulated EEPROM
GPIO General Purpose Input/Output
HSM Hardware Security Module
ID IDentifier
MC/DC Modified Condition/Decision Coverage
MCU MicroController Unit
MISRA Motor Industry Software Reliability Association
NVM Non Volatile Memory
RAM Random Access Memory
SDD Software Design Document
SoC System on Chip
SPI Serial Peripheral Interface
SWA SoftWare Architecture
UDS Unified Diagnostic Services
UML Unified Modeling Language
UWB Ultra WideBand

Glossaire

ISO-TP : norme internationale pour l'envoi de paquets de données sur le Bus CAN. 24

NXP : entreprise spécialisée dans la conception et la fabrication de semi-conducteurs. 8

TI : entreprise multinationale spécialisée dans la conception et la fabrication de semi-conducteurs et de technologies électroniques. 8

Chapitre 1

Introduction

Ce rapport présente le travail réalisé au cours de la période d'avril 2024 à août 2024, dans le cadre du projet de fin d'étude pour l'obtention du diplôme de Master en Ingénierie des Systèmes Temps Réel de la faculté Paul Sabatier à Toulouse. L'objectif du projet a été de développer une solution (Logger et outil de post-traitement) qui servira à instrumenter le Flash Bootloader dont le rôle est de charger et mettre à jour le logiciel embarqué du véhicule. Dans cet objectif, les taches suivantes ont été réalisées :

— Définition des exigences, de l'architecture et de la conception logicielle.
— Développement en respectant les règles de codage MISRA pour garantir la fiabilité.
— Réalisation des tests unitaires et des tests fonctionnels.
— Développement d'un outil de communication avec le microcontrôleur pour le post-traitement et la visualisation des données.

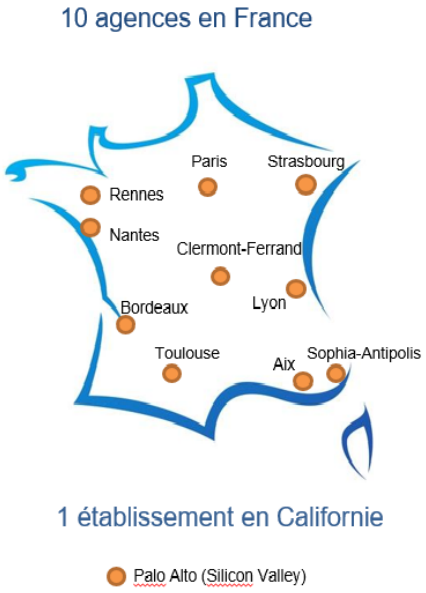
Le stage s'est déroulé au sein de la société CELAD, spécialisée dans le conseil et l'ingénierie, basée à Toulouse. Le projet a été conçu pour le compte de leur client CONTINENTAL, et le travail a été effectué au sein de l'équipe AutoBoot. Ce projet a été divisé en quatre livrables, avec une preuve de concept.

Le rapport commence par une brève présentation de CELAD, de CONTINENTAL et de l'équipe dans laquelle le stage s'est déroulé. Il se poursuit par la présentation du cahier des charges et des objectifs à atteindre, suivie des livrables attendus, de la méthodologie adoptée et du plan de travail. Ensuite, le rapport décrit l'environnement technique utilisé tout au long du stage, ce qui permet d'introduire le travail réalisé ainsi que les résultats obtenus. Enfin, le rapport se termine par un bilan global, les difficultés rencontrées, les solutions élaborées et une discussion sur les perspectives d'améliorations futures.

1.1 Celad

CELAD est une société de conseil et d'ingénierie créée en 1990 à Toulouse et spécialisée initialement en systèmes d'informations et en systèmes industriels depuis 2002. Aujourd'hui, elle est présente sur 10 grandes villes en France (figure ci-dessous) et possède également une agence dans la Silicon Valley. Au cours de l'année 2023 elle avait réalisé un chiffre d'affaires de 152 M€ et comptait un effectif de 1700 ingénieurs/collaborateurs. Avec 30 ans d'existence, CELAD a aujourd'hui un portefeuille de plus de 250 clients jouit d'une croissance régulière et contrôlée (plus de 20% en moyenne sur les 5 dernières années) et, en termes de santé financière, elle a obtenu une cotation Banque de France C3++ (note d'excellence).

(a) Croissance



(b) Répartition géographique

(c) Logo

FIGURE 1.1 – CELAD

1.2 Continental

Continental, groupe allemand spécialisé dans la fabrication de pneumatiques et de solutions technologiques pour l'automobile, est un acteur majeur de l'industrie automobile mondiale. La filiale de Toulouse, située au cœur d'un pôle technologique, se concentre principalement sur le développement de solutions électroniques et logiciels pour véhicules connectés et autonomes. Continental emploie plus de 190 000 personnes à travers le monde, dont plusieurs centaines à Toulouse. Le groupe a généré un chiffre d'affaires de 39,4 milliards d'euros en 2022. Parmi ses clients figurent des constructeurs automobiles de renom tels que BMW, Volkswagen et Daimler. La filiale toulousaine joue un rôle clé dans l'innovation en matière de mobilité.

FIGURE 1.2 – Logo de Continental

1.3 AutoBoot

Le stage a été effectué au sein de l'équipe Boot, nommée AutoBoot. Cette équipe est chargée de développer divers modules logiciels, tels que la cryptographie pour la sécurité, les drivers flash pour effacer et écrire en mémoire flash, et les drivers de communication pour transmettre les données via différents protocoles comme le Controller Area Network (CAN), le Serial Peripheral Interface (SPI), et le FlexRay.

L'ensemble de ces modules forme le Bootloader, d'où le nom de l'équipe. Le Flash BootLoader (FBL) est un outil logiciel qui occupe la mémoire non-volatile de l'unité de contrôle électronique (ECU, pour Electronic Control Unit), il est exécuté lors du démarrage du système. Il est essentiel pour effectuer la reprogrammation (mise à jour) du logiciel intégré (application) qui contrôle l'ECU dans un contexte automobile.

Un exemple d'ECU est l'ECU du système de freinage antiblocage (ABS), qui régule la pression des freins pour éviter le blocage des roues pendant le freinage.

Pour plus d'informations sur le FBL et l'ECU,voir l'Annexe.

Le développement de ces modules se fait en suivant la méthode agile : Scrum. De manière générale, les méthodes agiles privilégient le produit en étant à l'écoute des besoins du client, qui sont en perpétuelle évolution. En revanche, les méthodes classiques, comme le cycle en V par exemple, définissent l'intégralité du produit final dès les premières étapes et laissent peu de place à l'adaptation dans la suite du cycle, en privilégiant ainsi le processus.

Aux phases de développement s'ajoutent l'intégration, la vérification et la validation du produit.

1.4 Contexte

Du développement à la maintenance et au support, l'équipe fait face à certaines complications, impactant fortement le produit final :

— Durant la phase de développement et selon les projets, des soft resets (réinitialisation matérielle) peuvent survenir lors d'une séquence de reprogrammation (comme lors d'une mise à jour du logiciel FBL ou du module matériel de sécurité (HSM, pour Hardware Security Module) par exemple). Le problème est qu'après un reset (réinitialisation), la connexion au débogueur (un outil logiciel et matériel permettant le contrôle de l'exécution du code) est perdue, et les fonctionnalités de points d'arrêt, d'exécution ligne par ligne et l'inspection de la mémoire du débogueur ne peuvent donc plus être utilisées, ce qui complique les tâches d'investigation et de débogage.

— Durant la phase de maintenance et de support, il est parfois impossible de reproduire les problèmes ni de connaître leurs causes.

De ces problématiques est née la nécessité d'un module facilitant le débogage et la résolution des problèmes post-livraison (situations fréquentes). Ce module stocke des informations utiles dans un emplacement mémoire réservé, que le développeur peut consulter après l'exécution d'un scénario de test. Le FBL Logger permettra de renforcer les phases de développement et de tests, et surtout d'améliorer la phase de maintenance et de diagnostic sur le terrain, dans des conditions réelles.

1.5 Cahier de charges

Le FBL Logger devra remplir les exigences suivantes :

1. **Opération Non-Intrusive :** Il doit avoir un impact minimal dans le fonctionnement normal du système, capturant les événements à long terme pour l'analyse sans impacter significativement la performance.
   Structure d'un événement :
   — Temps de capture
   — Identifiant du module (CAN, Flash, Watchdog,...)
   — Identifiant de l'événement (Transmission, Réception, Écriture,...)
   — Données
2. **Stockage persistant :** Il doit stocker des événements dans une mémoire non-volatile (NVM, pour Non Volatile Memory) comme la Flash, assurant que les logs sont préservés à travers les cycles d'alimentation et les réinitialisations.
3. **Diffusion vers l'extérieur :** Il doit pouvoir transmettre les événements via un périphérique de communication vers l'extérieur.
4. **Maintenance et diagnostic :** Il doit pouvoir être intégrés sur le terrain dans le système pour fonctionner de manière continue et autonome, capturant des données pendant l'utilisation réelle, ainsi recueillant des informations sur les problèmes rares ou intermittents qui pourraient ne pas être facilement reproduits dans un environnement contrôlé.

5. **Moniteurs de performances :** Il doit fournir des éléments pour l'analyse des performances, y compris le profilage de l'exécution du code, afin d'optimiser la réactivité et l'efficacité globale.
6. **Générique :** Il doit pouvoir être utilisé sur n'importe quelle carte et n'importe quel projet, indépendamment du microcontrôleur et des périphériques utilisés.

1.6 Objectifs

A l'issue du projet, le système à concevoir doit être en mesure de remplir plusieurs fonctions principales telles que listées ci-dessous :

1. Stocker les événements en mémoire vive (RAM, pour Random Access Memory).
2. Stocker les événements en mémoire non-volatile Flash.
3. Actualiser le contenu de la mémoire non-volatile, en l'occurrence la Flash périodiquement (après chaque période de temps, après un nombre d'événements, avant un reset), pour ne pas impacter les performances du système.
4. Diffuser les logs via le bus CAN.
5. Calculer le temps minimum, maximum et moyen d'exécution d'une fonction.
6. Calculer la fréquence d'un événement selon une période d'observation.
7. Être configurable selon le besoin du développeur (Activation, Mode, Taille du buffer, ...).
8. Être le plus générique possible, avec des modifications mineures en fonction des projets.
9. Afficher les informations désirées sous un format compréhensible, afin de permettre l'analyse des données récupéré.

1.7 Flux opérationnel du projet

La figure ci-dessous illustre le flux de travail du projet. La solution développée comprendra le module logiciel "em_sll_logger" en C, intégré dans la carte, ainsi qu'un outil de post-traitement en Python, utilisable sur n'importe quel système d'exploitation.

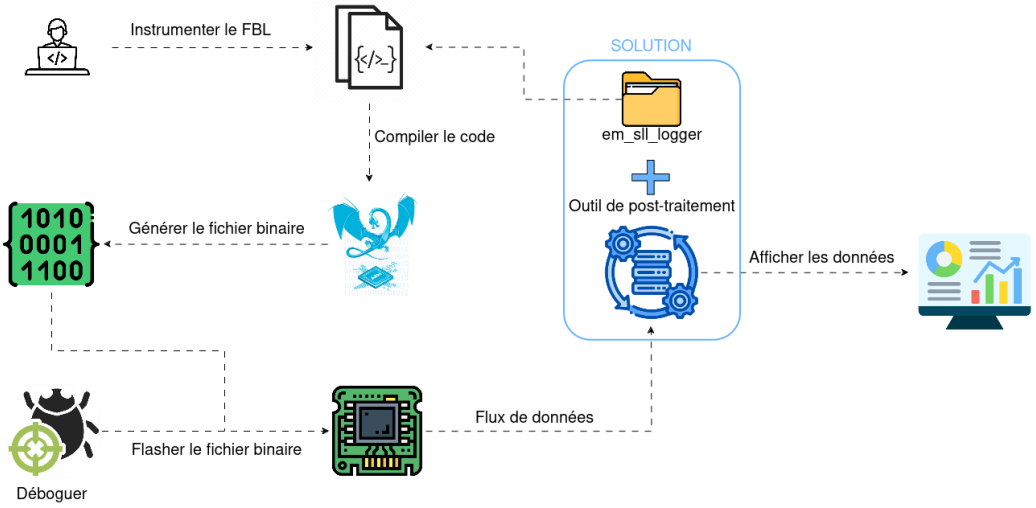


FIGURE 1.3 – Flux opérationnel de la solution

Chapitre 2

Planification et méthodologie de travail

2.1 Livrables attendus

Pour obtenir un produit final de qualité, le projet a été divisé en quatre livrables, favorisant ainsi l'adaptabilité aux besoins du client. Cette démarche, dite "itérative et incrémentale", est l'essence même des méthodes agiles.

Le tableau suivant récapitule les livrables attendus dans le cadre du projet technique.

Livrable	**Descriptif**
1	Stockage des événements en RAM
2	Stockage des informations liées à l'analyse des performances et développement de l'outil de post-traitement
POC (preuve de concept)	Démonstration de faisabilité
3	Diffusion sur bus CAN et mise à niveau de l'outil de post-traitement
4	Stockage des événements en Flash

Table 2.1 – Description des livrables attendus

Pour chaque livrable, un ensemble de documents doivent être fournis :
- Exigences logicielles
- Diagrammes de séquences
- Définition de l'interface de programmation d'application (API, pour Application Programming Interface)
- Définition détaillée de la structure des événements stockés
- Dépendances de l'architecture logicielle (SWA, pour SoftWare Architecture)
- Documentation de conception logicielle (SDD, pour Software Design Document)
- Code mergé
- Rapports de tests :
  - Tests unitaires
  - Tests fonctionnels

2.2 Méthodologie

Les étapes ci-dessous font partie de la méthodologie appliquée à chaque livrable :

1. **Analyse** : tout d'abord, une analyse des différentes approches possibles a été réalisée pour satisfaire les besoins, en fournissant les avantages et les inconvénients de chaque stratégie. Ensuite, le choix de la stratégie la plus adaptée aux attentes de l'équipe est effectué par consensus.
2. **Architecture** : dès que l'étape d'analyse est franchie, la rédaction de la documentation technique (dépendances logicielles, diagrammes de séquences, API et SDD) commence, accompagnée de revues régulières pour l'amélioration.
3. **Développement** : le développement a été réalisé en respectant les règles de codage de la Motor Industry Software Reliability Association (MISRA), tout en favorisant la clarté et la maintenabilité du code, avec des revues régulières.
4. **Test unitaire** : Les tests unitaires ont été réalisés à l'aide d'un logiciel propriétaire de Continental. Pour plus d'informations sur la stratégie adoptée, voir Annexe.
5. **Test Fonctionnel** : les tests fonctionnels ont été réalisés sur cible, validant à chaque fois les exigences logicielles. Ils ont été effectués selon un plan de test détaillé, avec des résultats attendus, qui ont ensuite été comparés aux résultats réels dans le rapport de test.

2.3 Plan

Ce plan permettra de suivre l'évolution du projet depuis ses débuts jusqu'à l'accomplissement des objectifs fixés. Tout au long de ce document, les défis rencontrés et les solutions élaborées seront présentés. Chaque livrable sera structuré de manière similaire :

D'abord, la réflexion apportée et les décisions prises seront expliquées dans la partie analyse. Ensuite, dans la partie architecture, les dépendances et les APIs développées seront présentées. La partie développement couvrira les vérifications de fiabilité du code, suivies des résultats des tests unitaires. Enfin, un exemple d'utilisation avec les résultats obtenus sera présenté.

Chapitre 3

Environnements techniques

Avant de nous plonger dans le travail accompli au cours du stage, il est essentiel de comprendre l'environnement technique dans lequel le développement de la solution a eu lieu. Cela permettra également de définir certains termes et concepts étroitement liés.

1. Langage :
   — C : la partie embarquée
   — Python : la partie traitement des événements capturés, que ce soit de la mémoire ou des trames CAN.
2. Environnement de développement :
   — **Logiciels :**
      — Vscode : éditeur de code extensible (supporte la majorité des langages de programmation via des extensions), il également a fait office de débogueur du code Python.
      — Lauterbach Trace32, Keil, IAR workBench : débogage du code embarqué
      — Logic Analyzer : outil qui permet d'analyser les trames SPI envoyées depuis le microcontrôleur.
   — **Matériels :**
   Les tests du module ont été effectués sur trois cartes projet avec des architectures différentes. Le développement du Logger a principalement été réalisé sur une carte projet détaillée ci-dessous.

   Cette carte est appelée "carte projet" car elle sera embarquée sur les voitures, contrairement à la carte dite "carte d'évaluation" destinée au développement. Elle se compose de deux puces : l'une provient de Texas Instruments (TI) et l'autre de NXP. Ces deux puces, choisies pour réduire les coûts, ont des rôles différents et n'embarquent pas les mêmes périphériques.

   Le système sur puce (SoC, System on Chip) NXP, qualifié d'"Esclave", intègre l'Ultra WideBand (UWB). L'Ultra Wideband est un protocole de communication radio rapide, sécurisé et à faible consommation, utilisé pour déterminer l'emplacement (comme celui des clés par rapport à la voiture) avec une précision inégalée par toute autre technologie sans fil.

   Le SoC TI, qualifié de "Maître", intègre le Bluetooth à faible consommation d'énergie (BLE, pour Bluetooth Low Energy), est chargé des communications

avec l'extérieur, que ce soit pour les requêtes de reprogrammation ou de diagnostic, et il communique également avec l'esclave, qui n'embarque pas de CAN.

3. Documentation
   — Plant UML : extension VScode permettant la création de diagrammes UML.
4. Gestion du code
   — Git Bash : "Git" : c'est un logiciel de gestion de versions décentralisé, "Bash" : est une interface de ligne de commande qui vous permet d'interagir avec votre ordinateur, donc Git Bash est juste une façon d'utiliser Git en ligne de commande.
5. Règles de codage
   — MISRA : Motor Industry Software Reliability Association, est une norme de programmation en langage C, favorisant la fiabilité et sécurité du code développé. les vérifications MISRA vont être faites avec un outil interne à Continental "qtools".
6. Outil de Test
   — Outil de test propriétaire : est un outil de test interne à Continental. Son intention est de couvrir tous les niveaux de tests logiciels.
7. Outil de configuration
   — "Configtool" : est un outil de configuration qui permet de générer des fichiers ".c" et ".h" en fonction des paramètres choisis, permettant ainsi d'avoir un code générique, mais configurable selon les besoins. **Comment ça marche ? :**
      — Le développeur crée un fichier configs.xsd représentant les paramètres configurables désirés.
      — Le développeur lance l'outil "configtool" et configure les paramètres, par la suite, un fichier configs.xml va être généré.
      — le développeur crée un/plusieurs fichier.xsl (selon l'architecture) définissant les fichiers finaux.
      — Pendant la phase de Build le/les fichier.xsl + le fhichier configs.xml généré seront les ingrédients pour générer les fichiers désirés.

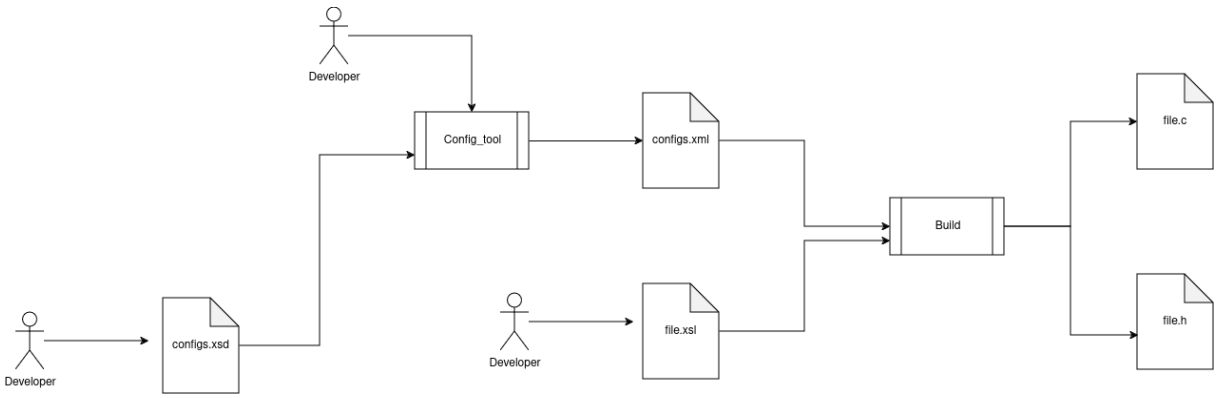


FIGURE 3.1 – Flux opérationnel de Configtool

Chapitre 4

Travail réalisé

4.1 Structure du module

Les modules développés au sein de l'équipe AutoBoot sont préfixés par "em_sll_" suivi du nom du module. Pour le Logger, il en sera de même. Il est également important de noter que les fichiers ".c" et ".h" sont préfixés par "bb", tandis que ceux générés sont préfixés par "zbb".

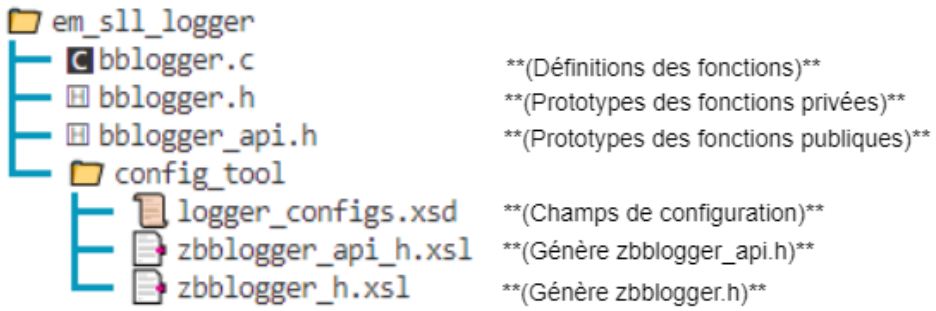


FIGURE 4.1 – Structure du module

4.2 L1 : Stockage en RAM

Le but de ce livrable est de fournir une API permettant de stocker des événements en RAM.

4.2.1 Analyse

La première réflexion va se porter sur la structure de données appropriée pour stocker ces événements, tout en optimisant l'utilisation de la mémoire.

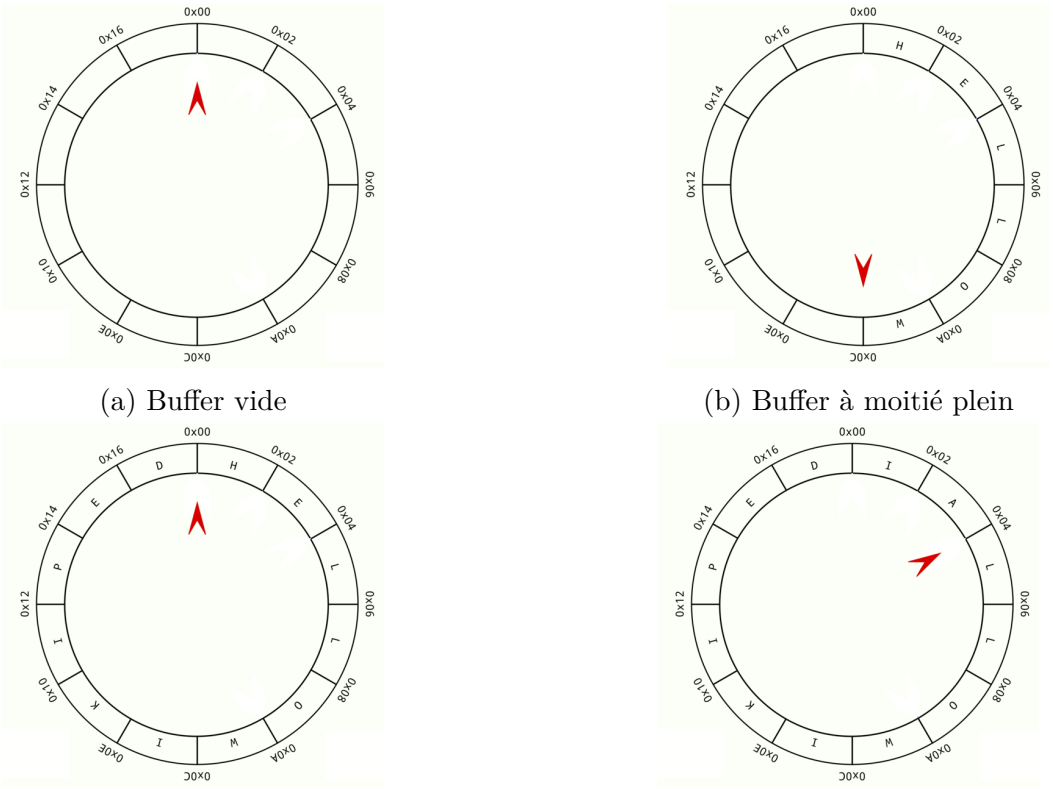
1. La taille du champ "Données" dans la structure d'un événement est variable.
   Structure d'un événement :
   — Temps de capture (4 octets)
   — Identifiant du module (2 octets)
   — Identifiant de l'événement (2 octets)
   — Données (X octets)

Réflexion : une structure de données dynamique, comme une liste chainée, avec comme structure du nœud :
— Un en-tête de taille fixe (alloué statiquement) :
  — Temps de capture (4 octets)
  — Identifiant du module (2 octets)
  — Identifiant de l'événement (2 octets)
— Un tableau de taille variable (alloué dynamiquement) pour les "Données"

Bien que cette approche permette de stocker de manière efficace les événements, elle n'est pas du tout permise dans un environnement embarqué à cause des ressources limitées. De ce fait, elle est éliminée.

Solution : comme les données stockées en mémoire n'ont pas de véritable valeur pour la puce, donc on peut stocker chaque octet (la plus petite unité de mesure en termes de stockage) indépendamment, de ce constat, on pourra ne pas spécifier de taille pour l'élément "Données" dans les événements enregistré.
On aura juste à allouer la taille de l'en-tête précédemment mentionné, qui se fera statiquement, elle, et les "Données" seront écrites octet par octet dans un buffer (tableau de type octet).

2. Pour répondre au fait que potentiellement les événements stockés seront infinis, le choix du buffer(tampon) circulaire répond parfaitement à cette contrainte. Il nous permettra d'enregistrer uniquement les derniers éléments souhaités, tout en écrasant les plus anciens.
Comment ça marche ? - :



(a) Buffer vide

(b) Buffer à moitié plein

(c) Buffer plein

(d) Anciennes données écrasées

FIGURE 4.2 – Logique du buffer circulaire[3]

On aura un pointeur (en rouge) qui va s'incrémenter après chaque nouvelle écriture. Arrivé à la fin du buffer, il rebouclera au début de celui-ci et cela infiniment. Cette solution règle aussi la contrainte de la mémoire limité des systèmes embarqués. Avec l'outil interne de Continental "configtool", on aura une flexibilité dans le choix de la taille du buffer circulaire en fonction des ressources de la puce.

3. Lecture des événements : comme les événements sont de différentes tailles et que potentiellement certains événements vont être partiellement ou totalement écrasés lors du rebouclage, la lecture va se compliquer.

   Deux approches ont été abordées :

   — **Première approche :** Utilisation d'un marqueur pour détecter le début d'un événement :
   Un marqueur de 4 octets au début de chaque structure d'événement est ajouté, avec un identifiant unique sur 3 octets et un compteur d'événement sur un octet.
   À cela, en plus du marqueur, il faut rajouter un nouvel élément à la structure d'un événement, "Taille de la donnée". Cet élément permettra lors de la détection d'un évènement de savoir combien lire d'octets pour la "Donnée".

   Structure utile d'un événement :
   — Marqueur d'événement (3 octets uniques + compteur sur 1 octet)
   — Temps de capture (4 octets)
   — Identifiant du module (2 octets)
   — Identifiant de l'événement (2 octets)
   — Taille de la donnée (1 octet)
   — Donnée (taille de la donnée en octets)

   — **Deuxième approche :** utilisation de deux pointeurs, un pour l'écriture (W) et l'autre pour la lecture (R), initialement pointant sur le premier événement, s'incrémente à chaque écrasement d'un événement. Pour plus de détails sur l'approche 2, voir l'Annexe.

   Comparaison

	Approche 1	**Approche 2**
Avantages	- Plus convivial (les événements sont facilement identifiés) - Moins d'utilisation du processeur	Moins de consommation de mémoire
Inconvénients	Plus de consommation de mémoire	- Moins convivial (début de l'événement difficile à identifier) - Plus d'utilisation du processeur

TABLE 4.1 – Comparaison des approches de lectutre

Le choix s'est fait par consensus avec l'équipe, et l'approche 1 (utilisation d'un marqueur pour détecter le début d'un événement) a été choisie.

4.2.2 Architecture

Dépendances et API

Le diagramme ci-dessous illustre l'utilisation du FBL Logger par différents clients et les dépendances qu'il a.

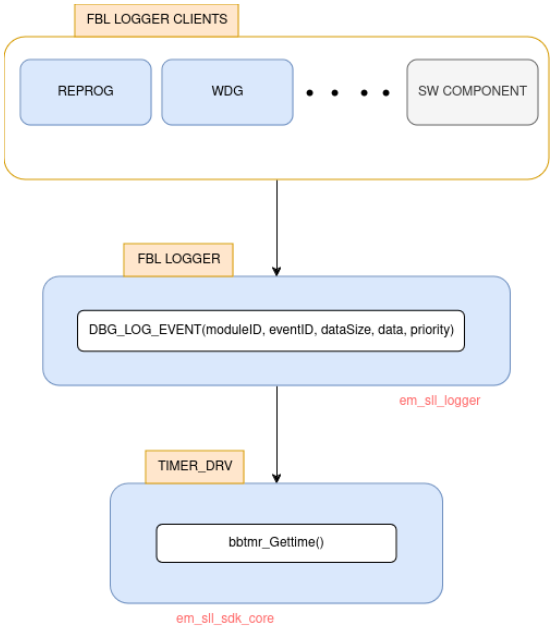


FIGURE 4.3 – Dépendances du Logger L1

Dans ce premier livrable, le Logger va dépendre de l'API bbtmr_GetTime() du module Timer pour récupérer le temps courant. Le module va exposer publiquement une macro de fonction :

DBG_LOG_EVENT(moduleID, eventID, dataSize, data, priority)

Elle permettra d'enregistrer des événements en mémoire RAM.

Descriptifs des arguments :

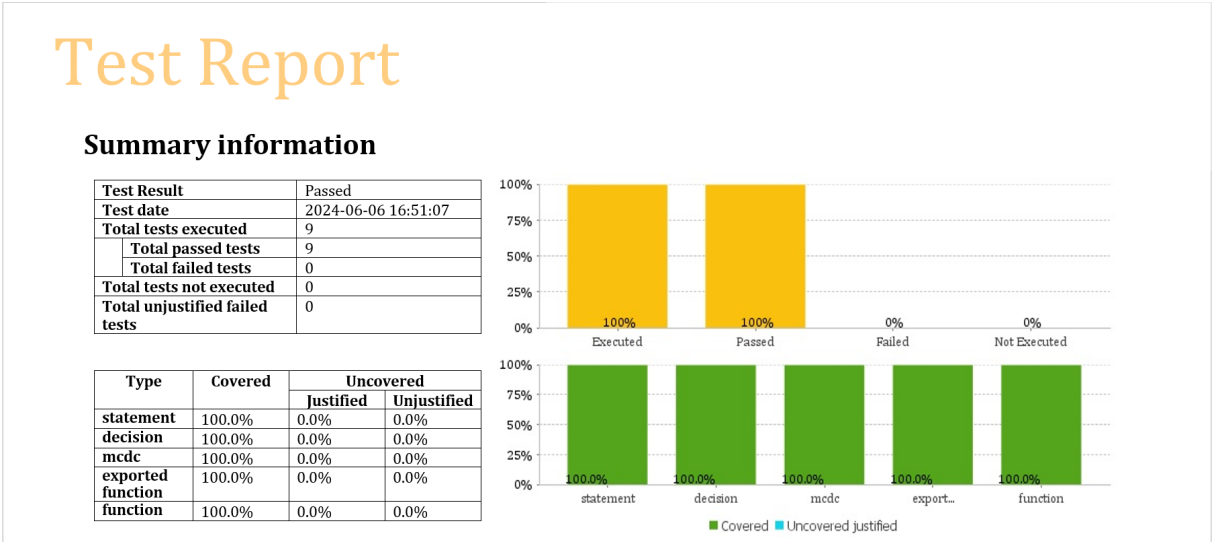
- **module ID :** c'est une énumération publique (située dans le fichier "bblogger_api.h") des différents modules (Flash, CAN, SPI, Watchdog, etc) présents dans le projet.
- **event ID :** c'est une énumération privée des différents événements (Écriture, Réception, Envoi, etc) de chaque module instrumenté.
- **data size :** taille de la donnée
- **data :** la donnée en question
- **priority :** c'est une énumération publique (située dans le fichier bblogger_api.h"), commençant de 0 (le plus haut niveau de priorité). Ce paramètre permet de stocker des événements en fonction de leur priorité.

4.2.3 Développement

Les tests MISRA-2012 ont été effectués, certains warnings (avertissements) sont apparus et corrigés par la suite.

4.2.4 Résultats de tests unitaires

Les tests unitaires ont été effectués et le rapport généré, montrant une couverture optimale du code développé.



Test Report

Summary information

Test Result	Passed
Test date	2024-06-06 16:51:07
Total tests executed	9
Total passed tests	9
Total failed tests	0
Total tests not executed	0
Total unjustified failed tests	0

Type	Covered	Uncovered	
		Justified	Unjustified
statement	100.0%	0.0%	0.0%
decision	100.0%	0.0%	0.0%
mcdc	100.0%	0.0%	0.0%
exported function	100.0%	0.0%	0.0%
function	100.0%	0.0%	0.0%

FIGURE 4.4 – Résultats de tests unitaires L1

4.2.5 Cas d'utilisation

Le diagramme de séquence ci-dessous montre un cas d'utilisation où le module FBL Logger a été instrumenté par deux modules différents, le "watchdog" (c'est un dispositif logiciel/matériel qui surveille le fonctionnement du système et le redémarre automatiquement en cas de défaillance ou de dysfonctionnement) et l'"ecu_job"(c'est un module qui permet d'exécuter des tâches spécifiques à l'ECU).

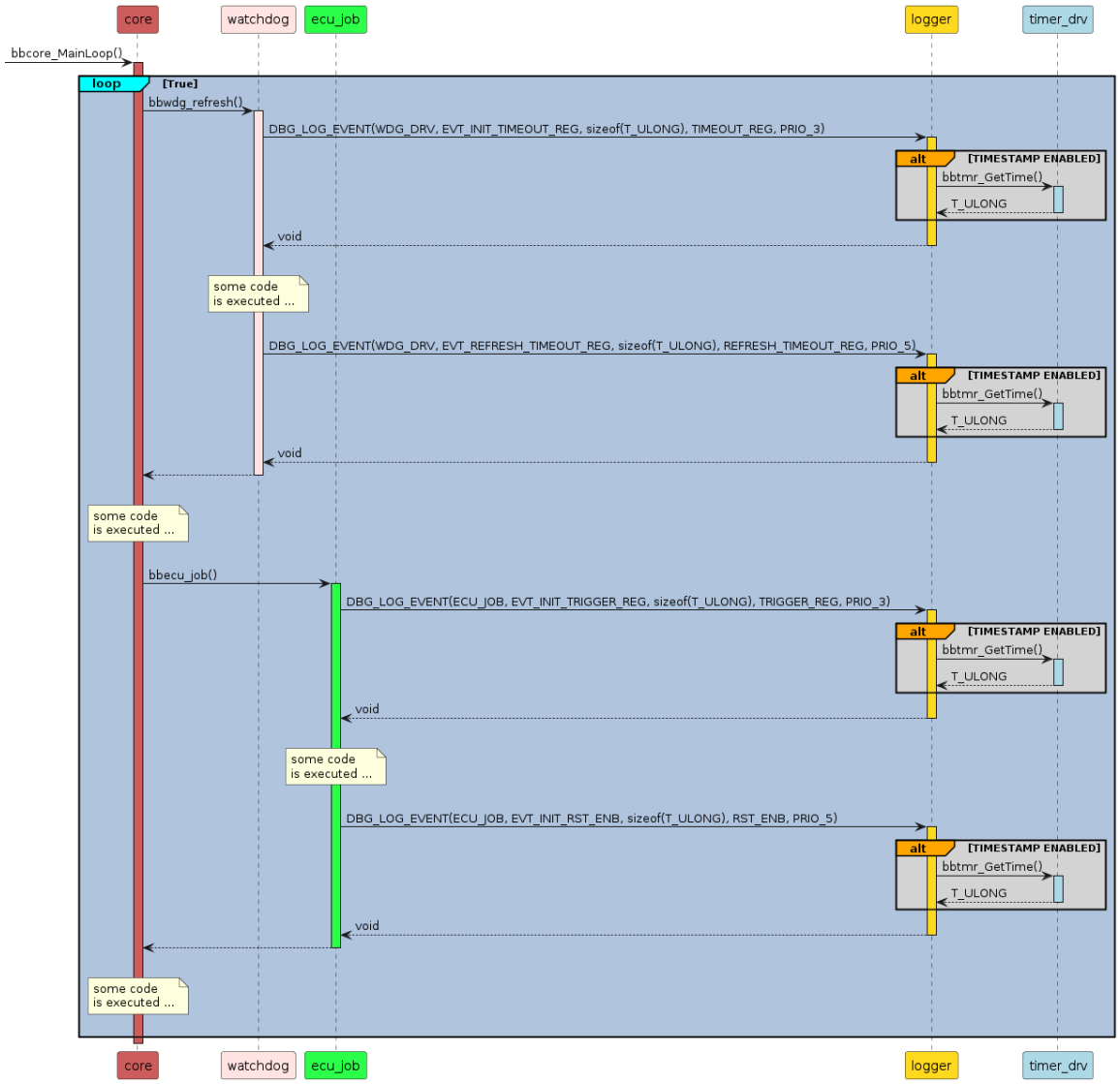


FIGURE 4.5 – Diagrammes de séquence d'un cas d'utilisation du L1

Le module "core" est l'élément central du FBL qui gère le démarrage et bien d'autres tâches. bbwdg_refresh() qui permet de rafraîchir le watchdog, bbecu_job() fonction où le code spécifique va être exécuté.
T_ULONG un type de donnée (unsigned long int).

La figure ci-dessous représente l'interface de Configtool, où les paramètres issus du fichier développé "logger_configs.xsd" sont configurables.

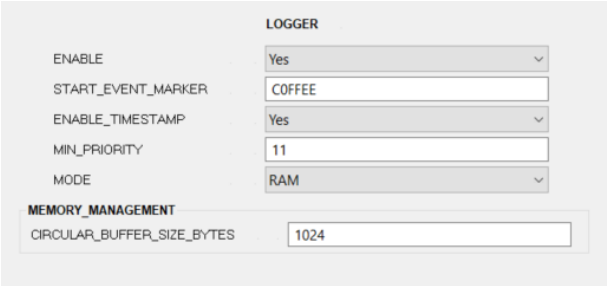


FIGURE 4.6 – Configtool L1

Ci-dessous, un dump mémoire du buffer circulaire où les événements sont stockés, qui montre bien que les événements ont bien été loggé en respectant la structure et l'ordre des éléments de la structure, également ci-contre un code couleur pour permettre de bien lire les événements.

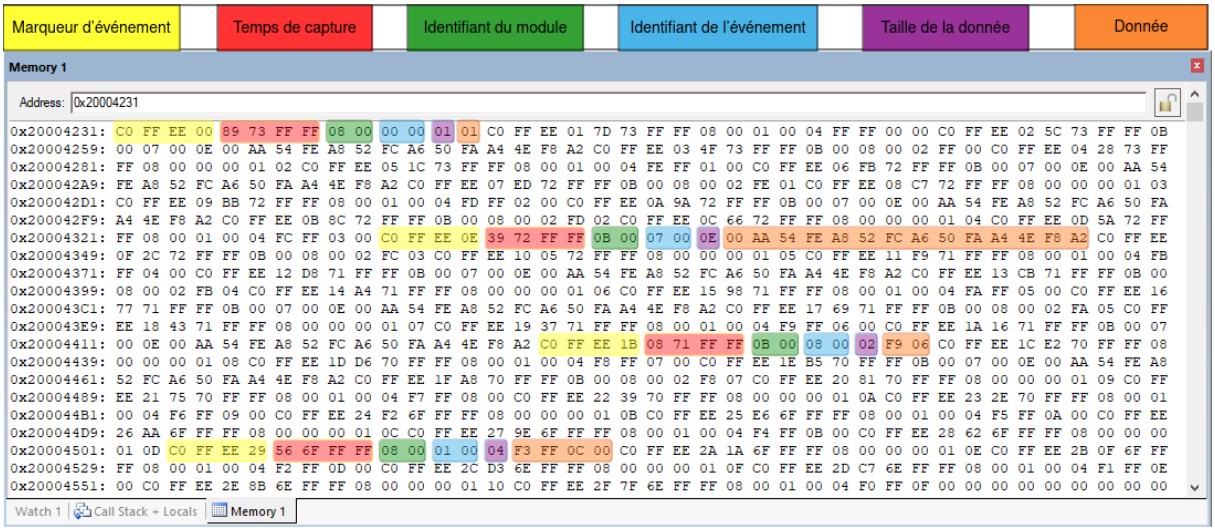


FIGURE 4.7 – Dump mémoire L1

Lors de la configuration des paramètres dans "configtool", le marqueur de début d'événement "C0FFEE" a été défini. Il est placé au début de chaque événement sur trois octets, suivi du compteur d'événements sur un octet. Ensuite, le temps de capture est représenté en rouge sur quatre octets, suivi de l'identifiant du module et de l'événement en vert et bleu, respectivement, chacun sur deux octets. La taille des données est indiquée en violet sur un octet, et enfin, les données elles-mêmes sont en orange, avec une taille variable, comme illustré par les exemples colorés. Cela valide et vérifie le premier livrable.

La figure ci-dessous reprend le même exemple que précédemment, avec les mêmes événements colorés, mais sans le temps de capture, qui n'a pas été sélectionné lors de la configuration via "Configtool".

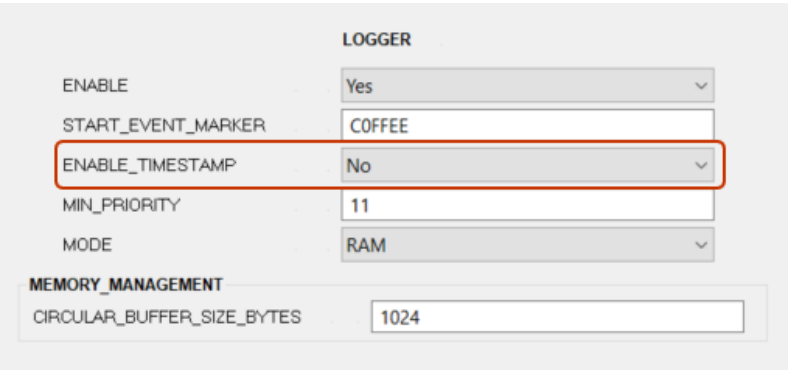


FIGURE 4.8 – Configtool sans le temps de capture L1

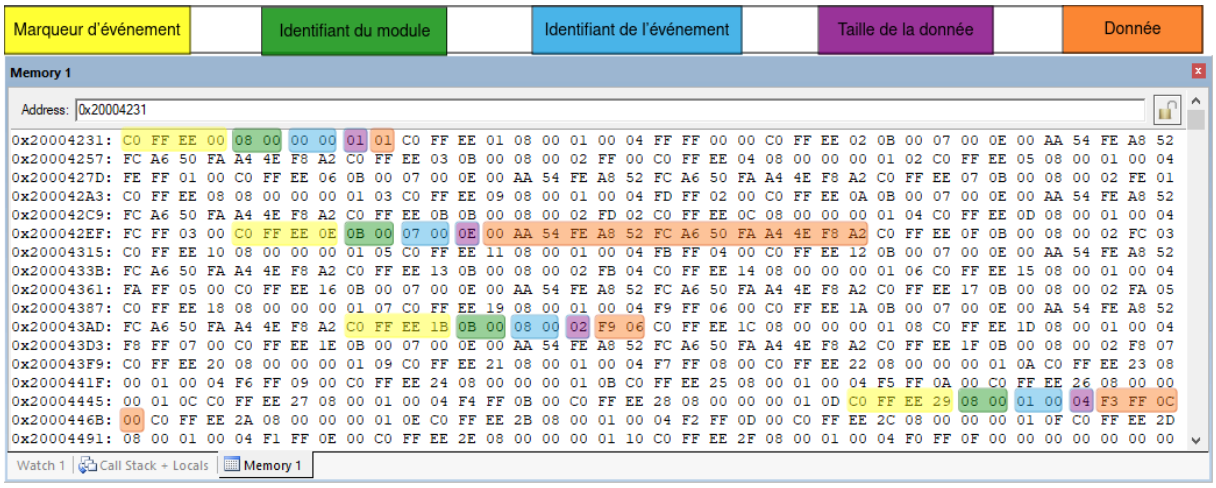


FIGURE 4.9 – Dump mémoire sans temps de capture L1

Le temps de capture, précédemment représenté en rouge dans les événements colorés, n'est plus présent. Cependant, les autres éléments de chaque événement sont restés inchangés. Ces résultats démontrent que le module est bien configurable selon les besoins, tout en respectant les exigences de ce premier livrable.

4.3 L2 : Analyse de performance et outil de post-traitement

Le but de ce livrable est de fournir une API permettant de calculer le temps d'exécution d'une fonction selon une période d'observation et ainsi de déduire la durée minimale, maximale et moyenne, ainsi que la fréquence d'une tâche, fonction,. donnée. De même que, se rajoute le développement d'un outil de post-traitement qui va avoir en entrée un dump mémoire du microcontrôleur et en sortie dans un format compréhensible les informations sur les performances et également les événements enregistrés du premier livrable.

4.3.1 Analyse

Calcul du temps d'exécution

Le choix du calcul du temps d'exécution d'une fonction est fait par l'étude de différentes approches :

1. **Première approche :**

   Utilisation du timer SysTick (timer matériel qui s'incrémente selon une base de temps configurable), pour capturer le temps au début de la fonction et le temps à la fin de la fonction.

2. **Deuxième approche :**

   Utilisation du registre du compteur de cycles (CYCCNT) de l'unité Data Watchpoint and Trace (DWT) pour optimiser le mode débogage, disponible sur les puces ARM Cortex M.

3. **Troisième approche :**
   Utilisation du basculement des broches d'entrée/sortie à usage général (GPIO, pour General Purpose Input/Output).

Comparaison

	Timer SysTick	**Registre de compteur de cycle**	**Basculement de broches GPIO**
Avantages	Faible surcharge	- Faible surcharge - Très Haute précision	- Facile à mettre en œuvre - Faible surcharge
Inconvénients	Dépend de la précision du SysTick	Spécifique à certaines puces Cortex M	- Besoin d'un analyseur logique - Vitesse de commutation des broches - Besoin d'un grand nombre de broches

TABLE 4.2 – Comparaison des approches de calcul de temps d'exécution

L'une des spécifications du Logger est la portabilité, donc le choix de l'approche 1 (timer SysTick) est le plus judicieux, du fait que les timers matériels existent sur toutes les cartes.

Traitement des logs

Le traitement des logs peut se faire de deux manières :

1. **Calcul pré-enregistrement**
   Le calcul des informations de performance telles que : tmin, tmax, tmoy, fréquence d'une fonction spécifique est effectué dans le Logger (CPU de la carte qui s'en charge).
2. **Calcul post-enregistrement**
   Le calcul des informations de performances telles que : tmin, tmax, tmoy, freq d'une fonction spécifique est effectué en dehors des fonctions du Logger, par un script Python (outil de post-traitement), après la récupération d'un dump mémoire ou de trames CAN.

	Pré-enregistrement	**Post-enregistrement**
Avantages	Utilisation faible de la mémoire (plus d'événements enregistrés)	Faible surcharge
Inconvénients	Utilisation accrue du processeur, surcharge élevée	Utilisation élevée de la mémoire (moins d'événements enregistrés)

TABLE 4.3 – Comparaison des approches de traitement des logs

L'une des spécifications du Logger est la faible surcharge (n'impacte que légèrement les performances), donc le choix de l'approche 2 (post-enregistrement) est le plus cohérent.

Structure d'un événement

Pour garantir la maintenabilité et la cohérence du code, la structure utilisée pour stocker les événements de type analyse de performance sera similaire à celle du premier livrable, avec quelques modifications mineures. Parmi ces modifications, le compteur d'événements, qui était le dernier octet du marqueur de début d'événement, deviendra la position (début ou fin) de l'exécution. Comme cette nouvelle fonctionnalité n'implique plus de données, la taille restera nulle, comme illustré ci-dessous.

Structure utile d'un événement :

- Marqueur (3 octets uniques + position (début ou fin) 1 octet)
- Temps de capture (4 octets)
- Identifiant du module (2 octets)
- Identifiant de l'événement (2 octets)
- Taille de la donnée (1 octet) toujours nulle

Outil de post-traitement

L'outil de post-traitement est un logiciel qui permet de parcourir un fichier, en l'occurrence un fichier mémoire du microcontrôleur, et d'afficher les résultats des logs de manière compréhensible, simplifiant ainsi l'analyse.

- Langage : Étant donné qu'il s'agit d'un outil de post-traitement, le langage de programmation le plus adapté est Python, en raison de son large catalogue de bibliothèques qui augmente la productivité du développement, laissant ainsi plus de temps pour satisfaire les besoins.
- Confort utilisateur : Le choix s'est porté sur une interface graphique pour offrir une meilleure ergonomie et fluidité d'utilisation, plutôt qu'une interface en ligne de commande nécessitant une connaissance préalable des commandes et étant moins attrayante.

4.3.2 Architecture

Dépendances et API

En ce qui concerne les dépendances du Logger pour ce livrable, elles restent les mêmes que pour le premier livrable.

Ce deuxième livrable va exposer publiquement une macro de fonction :

DBG_COMPUTE_CPU(moduleID, eventID, position, priority)

Elle permettra d'enregistrer des événements de type début d'exécution et fin d'exécution en mémoire RAM.

Descriptifs des arguments :

— **module ID :** c'est une énumération publique (située dans le fichier "bblogger_api.h") des différents modules (Flash, CAN, SPI, Watchdog, etc) présents dans le projet.
— **event ID :** c'est une énumération privée des différents événements (Écriture, Réception, Envoi, etc) de chaque module instrumenté.
— **position :** c'est une énumération publique (située dans le fichier "bblogger_api.h"), qui contient deux choix soit le début, soit la fin, pour permettre d'identifier le temps de démarrage et le temps de clôture d'une exécution.
— **priority :** c'est une énumération publique (située dans le fichier "bblogger_api.h"), commençant de 0 (le plus haut niveau de priorité). Ce paramètre permet de stocker des éléments de performances en fonction de leur priorité.

Outil de post-traitement

Dépendances

L'outil va dépendre de nombreuses librairies, comme le montre le diagramme ci-dessous

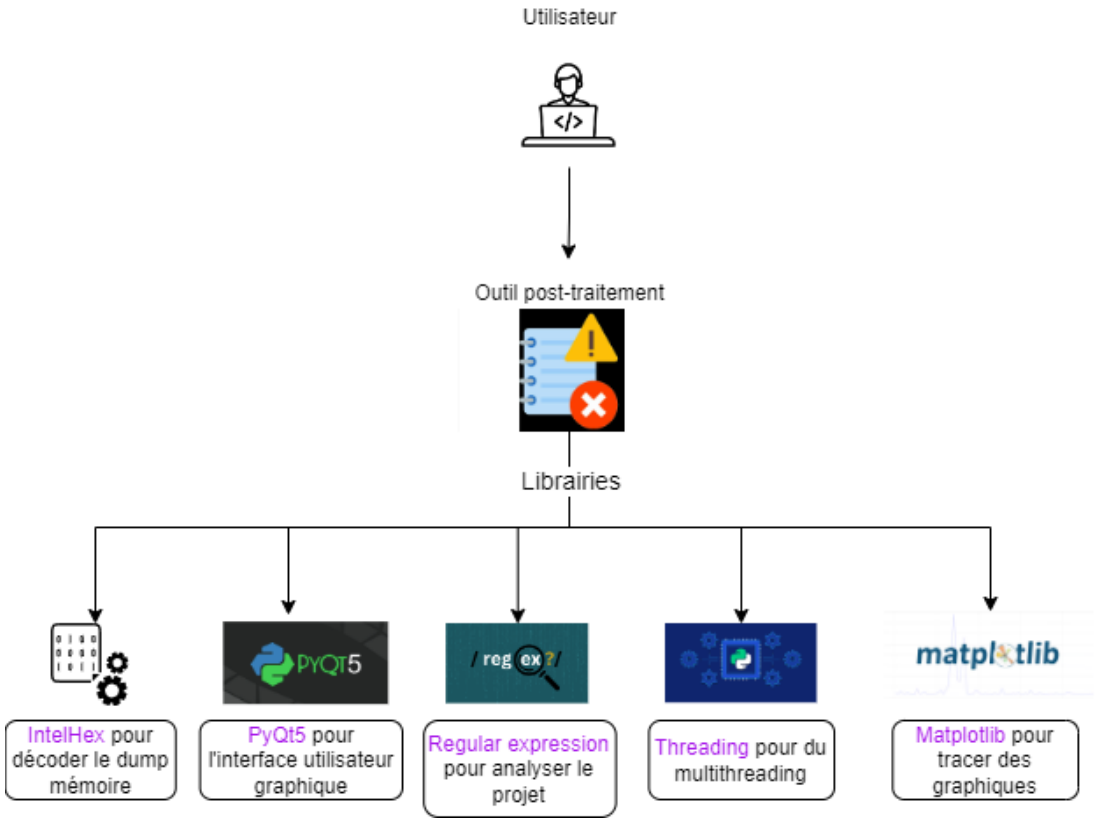


FIGURE 4.10 – Dépendances de l'outil de post-traitement L2

Diagramme de séquence

Le diagramme de séquence ci-dessous montre l'utilisation et le fonctionnement de l'outil de post-traitement pour visualiser le contenu du dump mémoire.

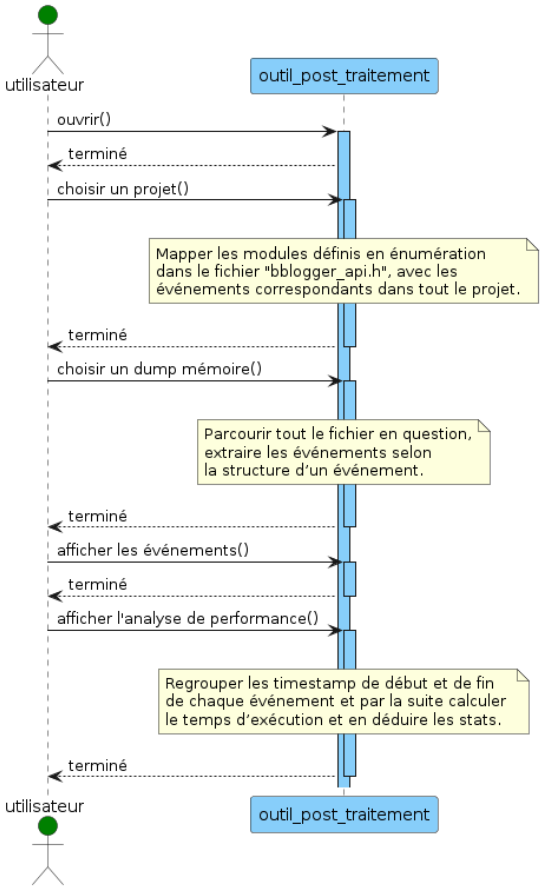


FIGURE 4.11 – Diagramme de séquence de l'outil post-traitement L2

4.3.3 Développement

Les tests MISRA-2012 ont été effectués, certains warnings(avertissements) sont apparus et corrigés par la suite.

4.3.4 Résultats de tests unitaires

Les tests unitaires ont été effectués et le rapport généré, montrant une couverture quasi optimale, avec un chemin non couvert, mais justifié.

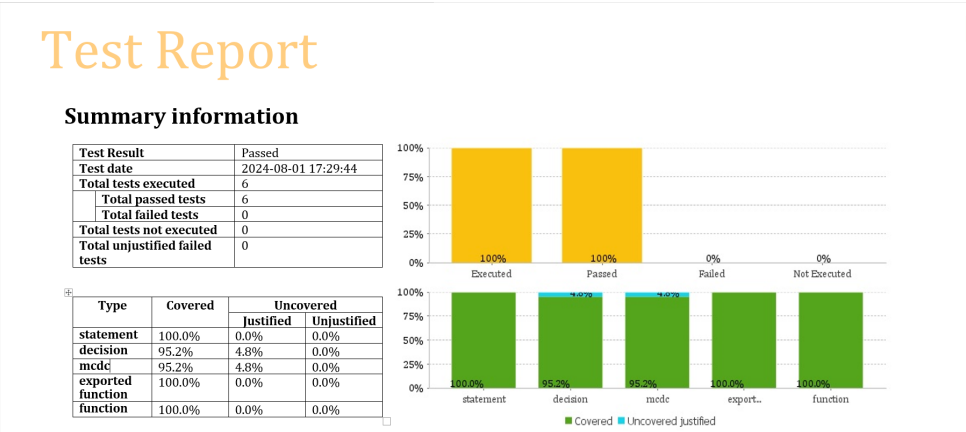


FIGURE 4.12 – Résultats de tests unitaires L2

4.3.5 Cas d'utilisation

Le diagramme de séquence ci-dessous montre un cas d'utilisation où le module FBL Logger a été instrumenté par deux modules différents, le "watchdog" et l'"ecu_job", avec les deux APIs disponibles du module.

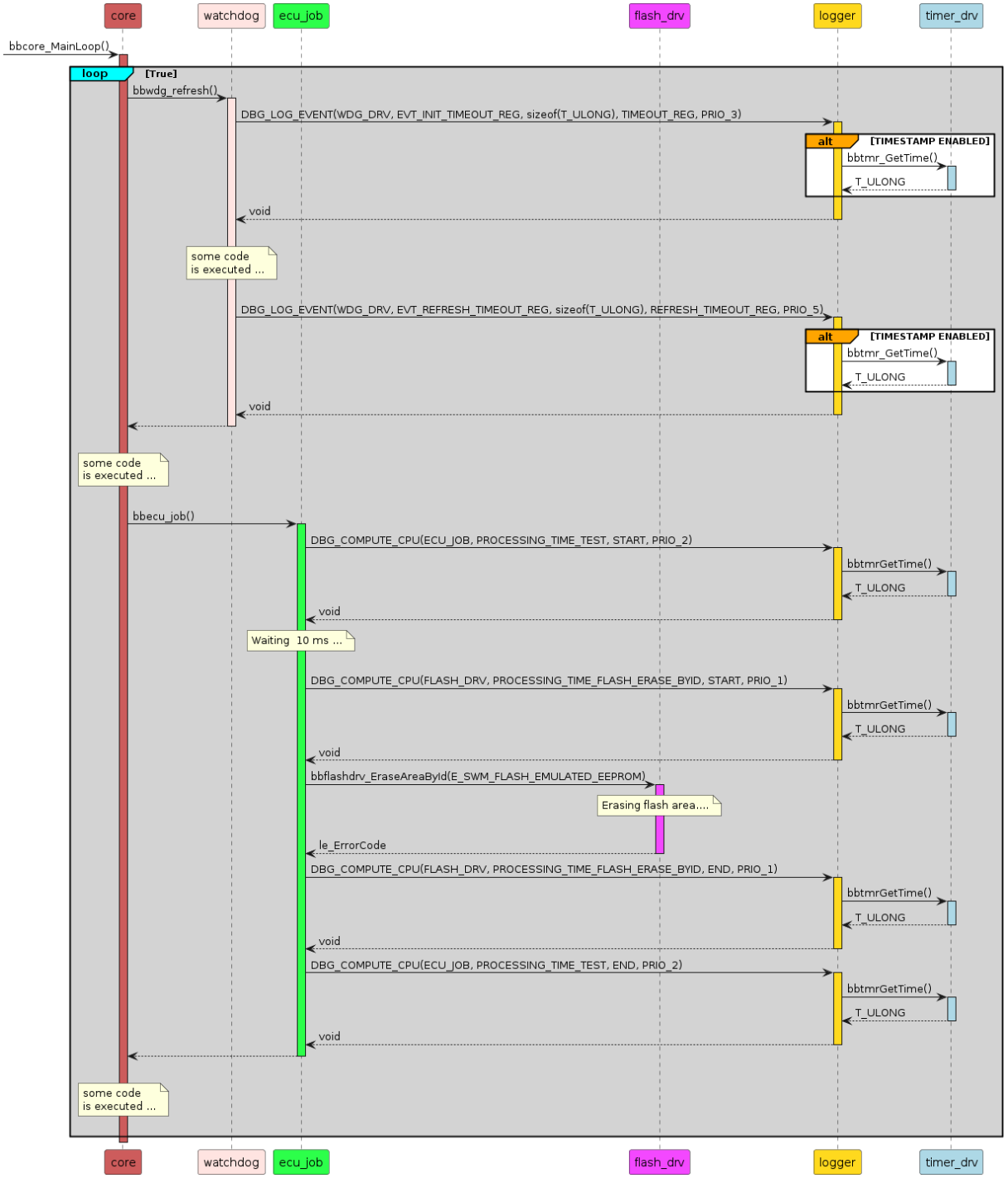


FIGURE 4.13 – Diagrammes de séquence d'un exemple L2

La figure ci-dessous est un dump de la mémoire issue de l'exemple d'utilisation illustré par le diagramme de séquence ci-dessus.

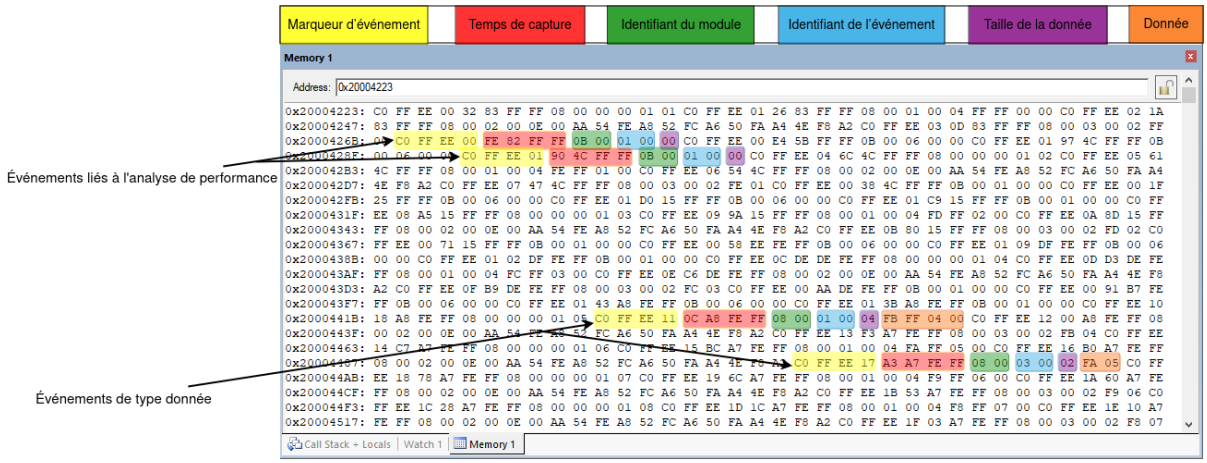


FIGURE 4.14 – Dump mémoire L2

Le résultat montre deux types d'événements : ceux liés à l'analyse des performances n'ont pas de champ de données (indiqué en orange) et leur taille de données est nulle. Pour différencier le début et la fin de l'événement, on se réfère au quatrième octet après le marqueur de début 'C0FFEE' : '00' pour le début et '01' pour la fin.

Les résultats ci-dessous montrent l'efficacité de l'outil de post-traitement à analyser, parcourir le projet, décoder le dump mémoire (ci-dessus) et à afficher sous un format lisible les événements et les éléments performances.

	Event	Observation time (ms)	Min time (ms)	Max time (ms)	Avg time (ms)	Frequency (ms)
1	PROCESSING_TIME	89.35	14.15	14.15	14.15	0.04
2	PROCESSING_TIME_FLASH_ERASE_BYID	89.34	3.98	3.98	3.98	0.04

FIGURE 4.15 – Analyse des performances

L'outil a permis d'afficher les performances des deux processus instrumentés. Le premier processus attend 10 ms avant d'exécuter la fonction qui efface un bloc mémoire de la Flash à partir d'un ID. Les exigences de ce livrable ont été respectées.

Les résultats suivants présentent les événements sur la base de la structure prédéfinie, vue dans le premier livrable (Seuls les quatre premiers événements sont montrés afin de maintenir la clarté de la figure et de l'explication. De plus, l'affichage est en hexadécimal).

	Event_counter	Timestamp	Module_ID	Event_ID	Data_size	Data
1	0x0	0xffff8332	0x8 BBWDGDRV	0x0 EVENT_INIT_TIMEOUT_REG	0x1	['0x1']
2	0x1	0xffff8326	0x8 BBWDGDRV	0x1 EVENT_REFRESH_TIMEOUT_REG	0x4	['0xff', '0xff', '0x0', '0x0']
3	0x2	0xffff831a	0x8 BBWDGDRV	0x2 EVENT_INIT_TRIGGER_INT_REG	0xe	['0x0', '0xaa', '0x54', '0xfe', '0xa8', '0x52', '0xfc', '0xa6', '0x50', '0xfa', '0xa4', '0x4e', '0xf8', '0xa2']
4	0x3	0xffff830d	0x8 BBWDGDRV	0x3 EVENT_INIT_RST_ENB	0x2	['0xff', '0x0']

FIGURE 4.16 – Événements

Le timestamp (temps de capture) du premier événement, issu d'un timer matériel qui décrémente chaque microseconde, est '0xFFFF8332'. Par rapport à la valeur initiale '0xFFFFFFFF' et au moment exact de l'initialisation, ce temps de capture correspond

approximativement à 32 ms.

L'outil permet également d'afficher les événements stockés en mémoire dans un format clair. Étant donné que les modules et les événements sont enregistrés avec leurs identifiants, l'outil analyse l'ensemble du projet pour faire le lien entre chaque ID et le nom du module ainsi que de l'événement. Il affiche ensuite ces informations pour faciliter la compréhension et l'analyse.

4.4 L3 : Diffusion sur CAN et mise à niveau de l'outil de post-traitement

Le but de ce livrable est de fournir à l'utilisateur la possibilité via "Configtool" de choisir le mode CAN, permettant de diffuser les logs via le BUS CAN. De ce fait, ce livrable nécessitera également la mise à niveau de l'outil de post-traitement pour intercepter ces trames CAN, les décoder et afficher les résultats.

Le protocole CAN (Controller Area Network) est un standard de communication utilisé principalement dans les systèmes embraqués, notamment dans les véhicules, il permet à différents microcontrôleurs et dispositifs de communiquer entre eux sans avoir besoin d'un hôte central.

4.4.1 Analyse

Le Logger doit avoir un canal et un ID spécifique pour la communication, prendre aussi en compte que l'esclave n'a pas de périphérique CAN et communique avec le maître uniquement en protocole SPI (Serial Peripheral Interface), de ce fait les logs qui seront établis sur l'esclave vont être transmis via le maître qui sera une sorte de gateway (passerelle) vers l'extérieur.

Le SPI (Serial Peripheral Interface) est un protocole de communication synchrone, il est limité par la taille des trames (souvent 8 bits ou 16 bits par transfert). De ce fait, on va utiliser une couche de transport qui est le SPI-TP (SPI Transport Protocol) qui intègre un mécanisme pour envoyer de grands volumes de données segmentés sur plusieurs transferts. Pour plus d'informations sur le SPI, voir Annexe.

Le même réflexion va se poser pour le protocole CAN classique qui est limité par la taille de ces données à 8 octets et le CAN-FD à 64 octets, pour remédier à ce problème, l'équivalent du SPI-TP, l'ISO-TP coté CAN va être utilisé. Pour plus d'informations sur l'ISO-TP, voir Annexe.

Le CAN-FD (Flexible Data-rate) est une extension du protocole CAN, offrant des améliorations en termes de vitesse et de capacité de données.

Il est également important de configurer l'ISO-TP en privilégiant l'utilisation du CAN-FD, si disponible, afin d'éviter la fragmentation des événements et de bénéficier des avantages du CAN-FD par rapport au CAN classique.

4.4.2 Architecture

Dépendances et API

Pour ce troisième livrable, le Logger dépendra de deux nouveaux modules : le SPI-TP et l'ISO-TP, ainsi que de leurs API respectives, comme illustré dans le diagramme ci-dessous.

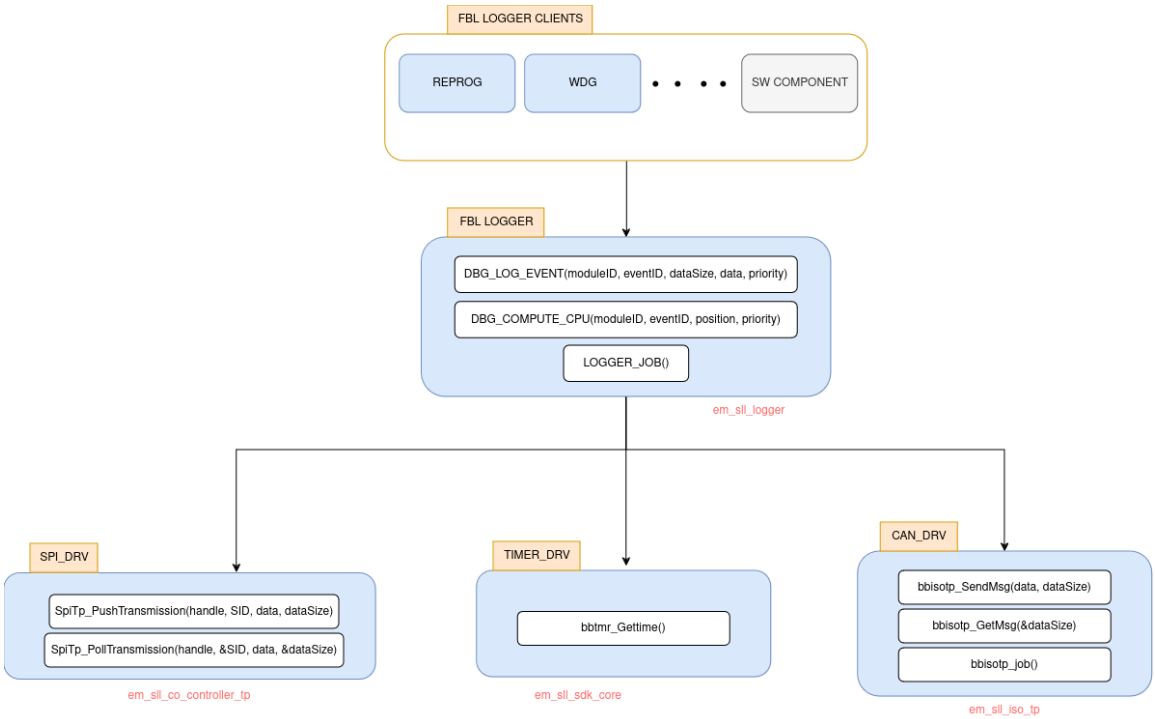


FIGURE 4.17 – Dépendances du Logger L3

Diagramme de séquence

Le diagramme de séquence ci-dessous illustrant le comportement du Maître et de l'Esclave en mode CAN, lors de l'utilisation de Logger.

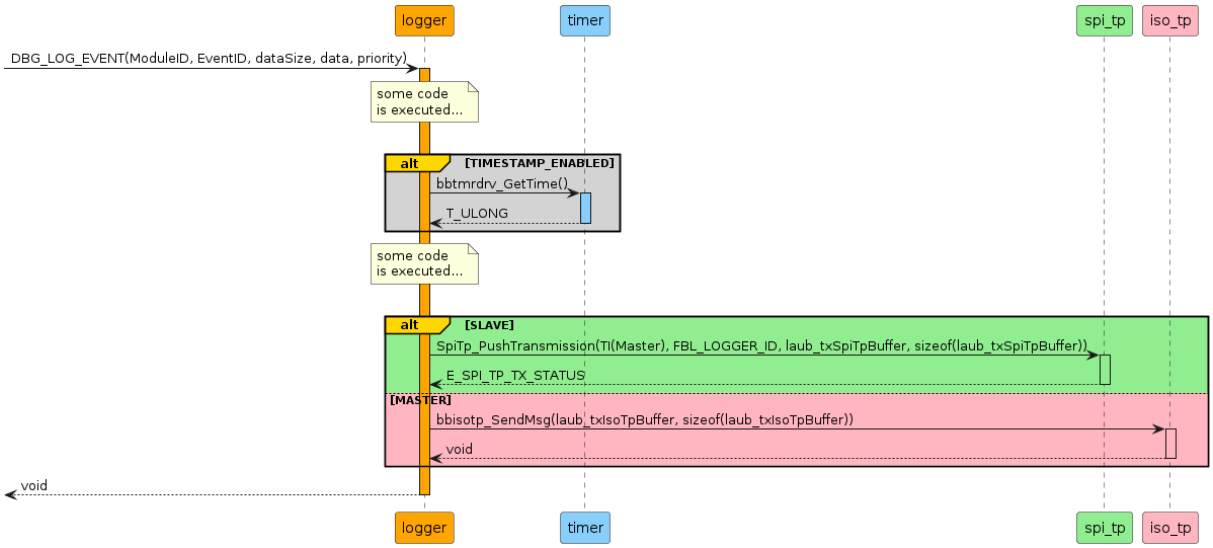


FIGURE 4.18 – Mode CAN

Ce troisième livrable va exposer publiquement une macro de fonction :
LOGGER_JOB()
Elle permettra de gérer la communication entre l'Esclave et le Maître lorsque le Logger est utilisé sur l'Esclave. Les événements seront alors transmis au Maître via SPI, et c'est là que l'utilité de cette nouvelle macro sera évidente, en retransmettant les événements via CAN vers l'extérieur. Le diagramme de séquence ci-dessous illustre cette logique.

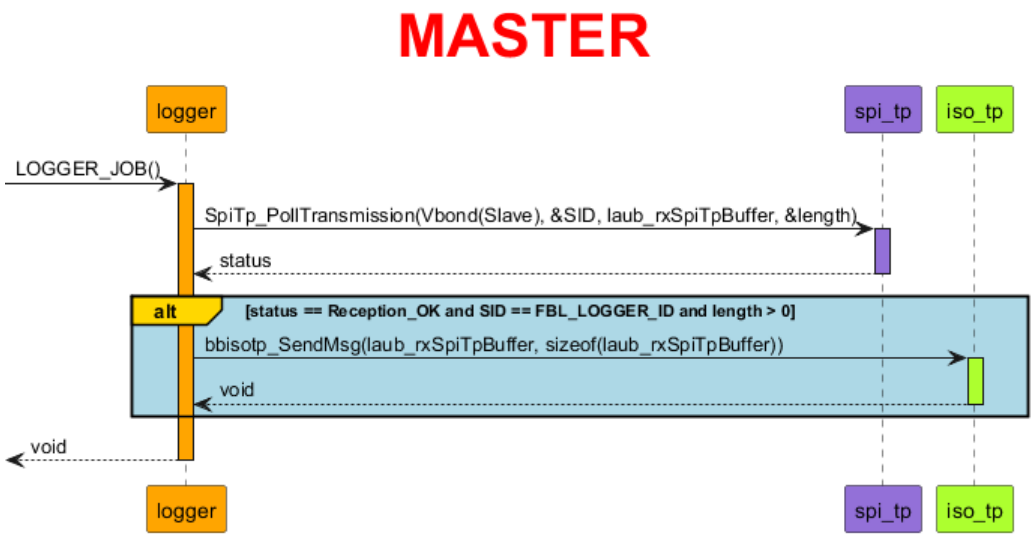


FIGURE 4.19 – Fonctionnement de la macro LOGGER_JOB() coté Master

SID : Select ID

Outil de post-traitement

Dépendances

L'élément encadré "PCANBasic" est la nouvelle librairie, venant se rajouter aux précédentes, pour permettre la communication CAN entre la carte et l'outil de post-traitement.

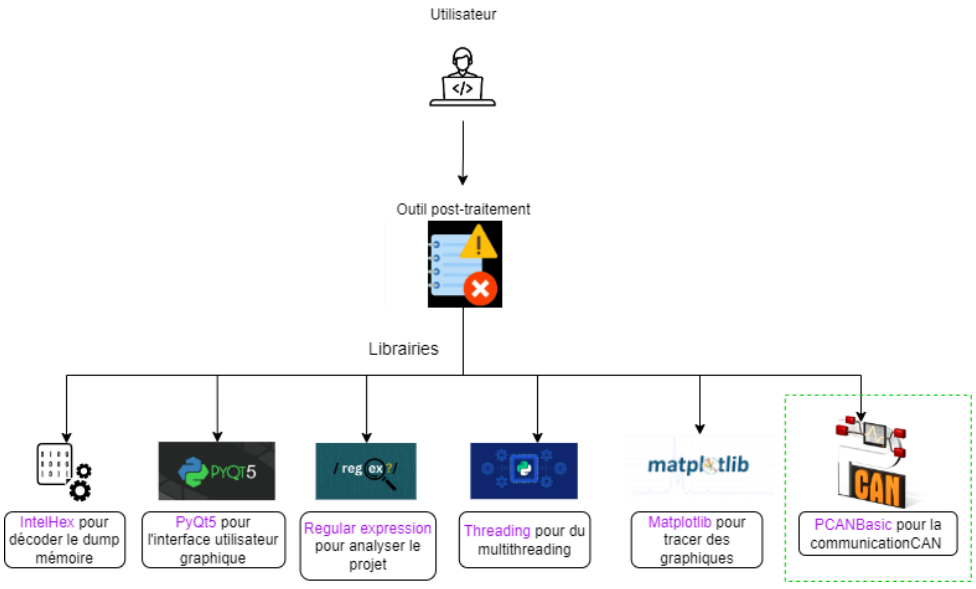


FIGURE 4.20 – Dépendances de l'outil de post-traitement L3

Diagramme de séquence

Le diagramme de séquence ci-dessous explique l'utilisation et le fonctionnement de l'outil de post-traitement.

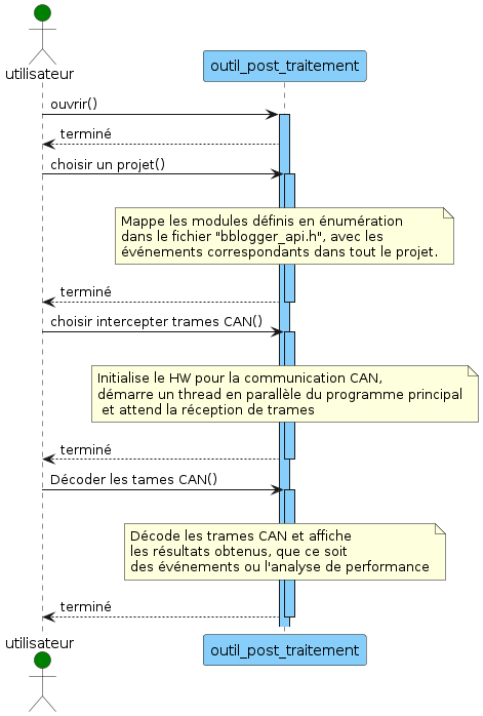


FIGURE 4.21 – Diagramme de séquence de l'outil post-traitement L3

4.4.3 Développement

Les tests MISRA-2012 ont été effectués, certains warnings(avertissements) sont apparus et corrigés par la suite.

4.4.4 Résultats de tests unitaires

Les tests unitaires ont été effectués et le rapport généré, montrant une couverture optimale du code développé.

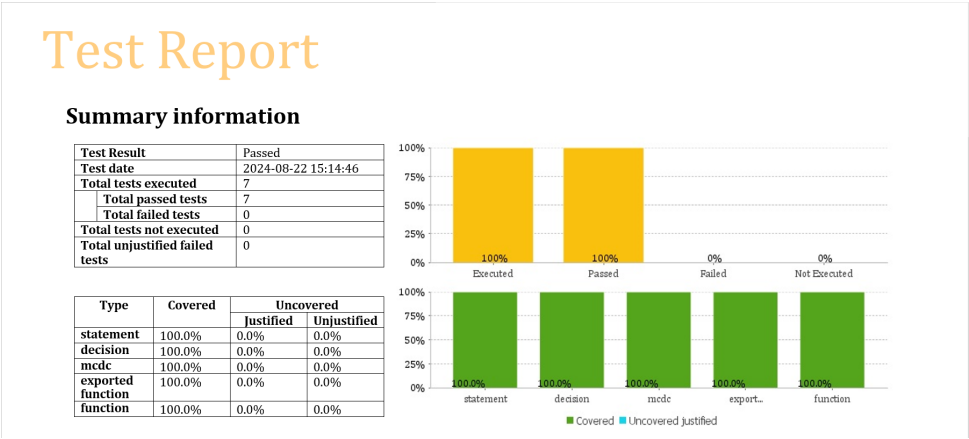
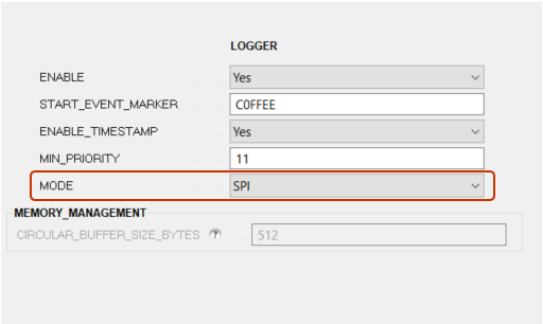


FIGURE 4.22 – Résultats de tests unitaires L3

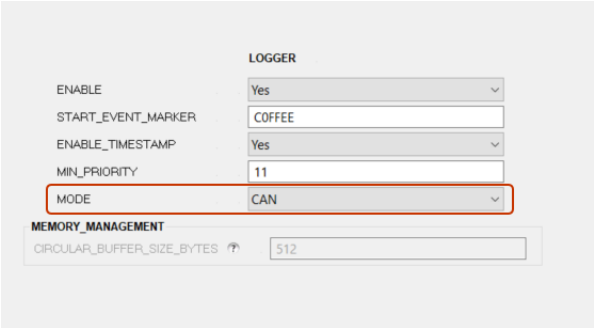
4.4.5 Cas d'utilisation

Le diagramme de séquence illustrant l'exemple d'utilisation est le même que celui du livrable précédent.

Les figures ci-dessous représentent l'interface de "configtool", coté Esclave et Coté Maître, indépendante l'un de l'autre.



(a) Paramètre de configuration de l'Esclave



(b) Paramètre de configuration du Maître

FIGURE 4.23 – Paramètres Configtool L3

La figure ci-dessous est un zoom d'une trame SPI envoyée de l'Esclave vers le Maître issu du Logger.

Tous les résultats ci-dessous sont en hexadécimal.

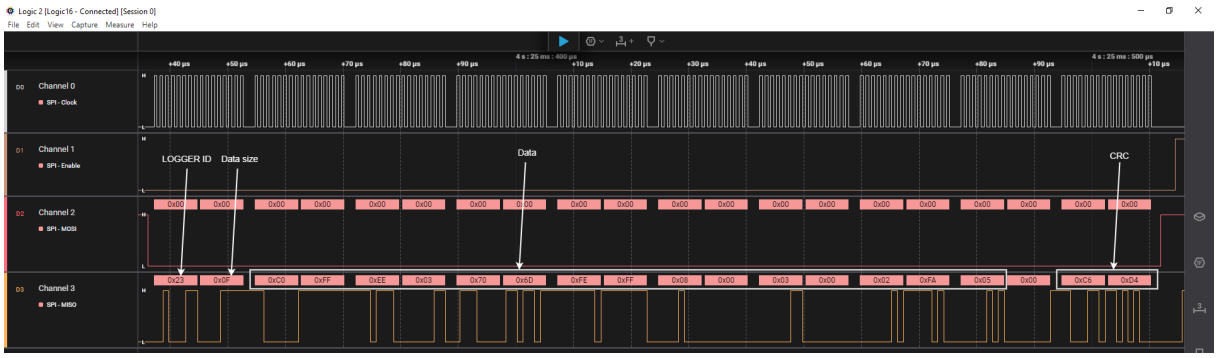


FIGURE 4.24 – Logic analyzer, zoom sur une trame SPI envoyée de l'Esclave vers le Maître

Contrôle de Redondance Cyclique (CRC, pour Cyclic Redundancy Check) : est un algorithme de détection d'erreurs.

LOGGER_ID : Identifiant spécifiant que la communication vient du FBL Logger.

Ce résultat montre que le mode SPI coté Esclave fonctionne parfaitement et envoie l'événement sous le bon format, comme le montre le champ "Data", commençant par le marqueur de début "C0FFEE" avec le compteur d'événement ici l'événement numéro 3, puis le temps de capture sur quatre octets, suivit des identifients du module et son événement tous deux sur deux octets, de la taille de la donnée sur un octet et pour finir la

donnée. Le reste de la trame est lié au protocole SPI-TP.

La figure ci-dessous illustre la nouvelle fonctionnalité de l'outil de post traitement, qui permet d'intercepter les trames CAN envoyées depuis le Maître et les affiches sous un format clair et intuitif (Seules les trois premières trames sont montrées afin de maintenir la clarté de la figure et de l'explication).

	Timestamp	CAN-ID	LENGTH	Data	Tx / Rx	Type
1	11:16:46.669548	777h	16	0E C0 FF EE 00 7D CE 00 00 08 00 00 00 01 01 FF	Rx	STD [FD BRS]
2	11:16:46.778232	777h	20	00 11 C0 FF EE 01 2F 2D 01 00 08 00 01 00 04 F8 FF 07 00 FF	Rx	STD [FD BRS]
3	11:16:46.893951	777h	32	00 1B C0 FF EE 02 04 99 01 00 08 00 02 00 0E 00 AA 54 FE A8 52 FC A6 50 FAA4 4E F8 A2 FF FF FF	Rx	STD [FD BRS]

FIGURE 4.25 – Interception des trames CAN envoyées depuis le Maître par l'outil de post-traitement

— Rx : Réception.
— Tx : Transmission.
— STD : Identifiant standard su 11 bits.
— Bite Rate Switch (BRS) : est une fonctionnalité clé du CAN-FD, Lorsque le BRS est activé, le débit de la phase de données est plus rapide que celui de la phase d'arbitrage (fonctionne à la même vitesse que le CAN classique pour garantir la compatibilité).

En prenant la première trame reçue, son champ "Data" commence par '0E'. Cela est dû à la norme ISO-TP, où chaque donnée est précédée par le type de trame (ici, Single Frame '0' sur 4 bits) et la taille de la donnée utile envoyée sur les 4 bits restants. Si la taille de la donnée utile dépasse 15 ($2^4 - 1$), l'octet suivant est utilisé pour spécifier la taille sur 12 bits, comme le montrent la deuxième et la troisième trame reçues. Ensuite commence l'événement avec le marqueur de début d'événement 'C0FFEE', et le reste de la trame (dont la taille est définie par le 'Data Length Code (DLC)') est constitué de padding (remplissage), dans ce cas 'FF'.

4.5 L4 : Stockage en Flash

Le but de ce livrable est de fournir à l'utilisateur la possibilité via "Configtool" de choisir le mode MIXED, permettant de stocker les événements en NVM (Flash), tout en proposant via le BUS CAN la possibilité de récupérer son contenu à travers l'envoie d'une trame requête, que ce soit pour l'Esclave ou pour le Maître.

4.5.1 Analyse

Réflexion

Pour répondre aux besoins de ce livrable, il faut une mémoire non volatile qui permette d'écrire un nombre variable d'octets.
La NVM disponible sur les cartes utilisées est la Flash. La Flash est divisé en plusieurs blocs de taille fixe.
Le processus d'écriture en Flash utilise un processus appelé « effacement avant écriture ». Pour écrire des données, un bloc de mémoire Flash doit d'abord être effacé, par conséquent, cela ne correspond pas aux besoins.

Le comportement désiré se rapproche plus de l'EEPROM (Electrically-Erasable Programmable Read-Only Memory) : c'est une mémoire morte qui permet d'écrire et d'effacer des données octet par octet.
Pourquoi écrire et effacer octet par octet ? prenant l'exemple où l'un des blocs mémoire de la Flash contient déjà des données utiles, avec une Flash, il faut effacer tout le bloc mémoire pour pouvoir y écrire, supprimant ainsi ces données utiles.

Solution

la solution est d'utiliser une EEPROM émulée en Flash (FEE, pour Flash Emulated EEPROM) qui permet d'avoir le comportement d'une EEPROM sans le matériel, uniquement avec une Flash.

Comment ça marche ?

Pendant la phase d'initialisation, si le contenu du bloc de mémoire Flash est valide, c'est-à-dire que l'intégrité de ces données est garantie, par quoi ? le CRC. Alors le contenu de la Flash est copié en RAM, les modifications apportées vont se faire en RAM et par la suite si une certaine condition est respectée (pour minimiser au maximum l'impact sur le fonctionnement principal), le CRC va être calculé et va être écrit avec l'image RAM en Flash dans l'un de ces blocs mémoire.

La figure ci-dessous présente la mémoire Flash, divisée en plusieurs blocs avec leur adresse de début. En zoomant sur l'un des blocs, on visualise le format de la solution développé FEE.

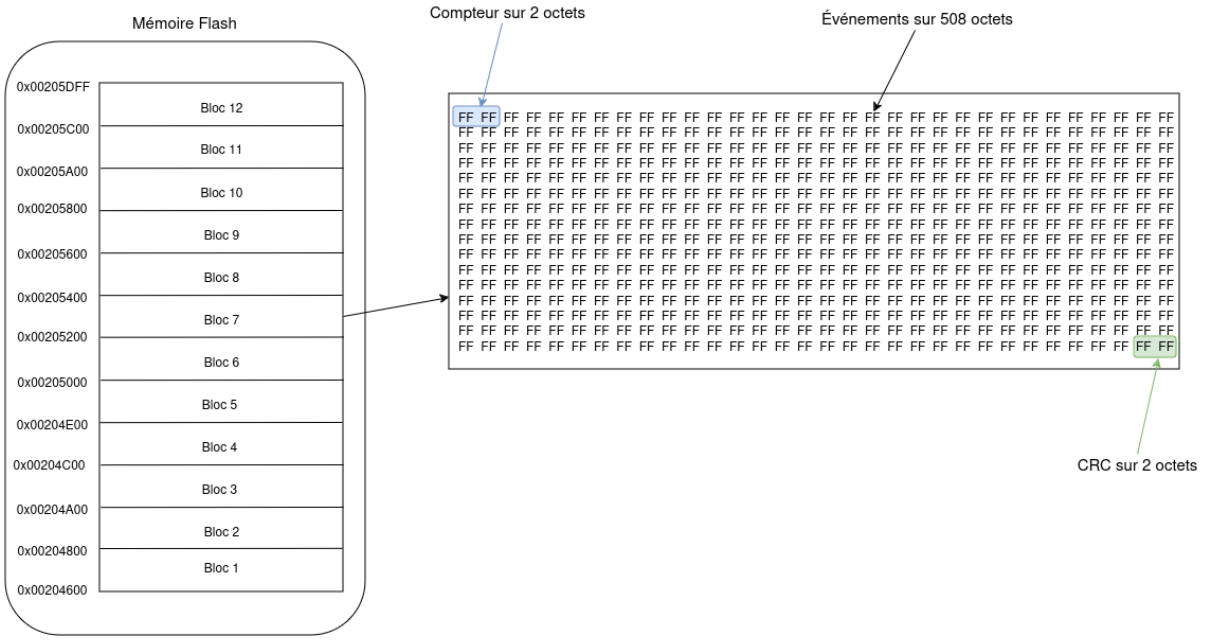


FIGURE 4.26 – Mémoire Flash avec des blocs de 512 octets

Également, comme la Flash n'est pas extensible, l'écriture va se faire selon un buffer(tampon) circulaire.

4.5.2 Architecture

Dépendances et API

A ce dernier livrable vient se rajouter le Flash driver et le module Crc16. Avec ces deux nouveaux modules, la FEE (FLASH Emulated EEPROM) va être développée.

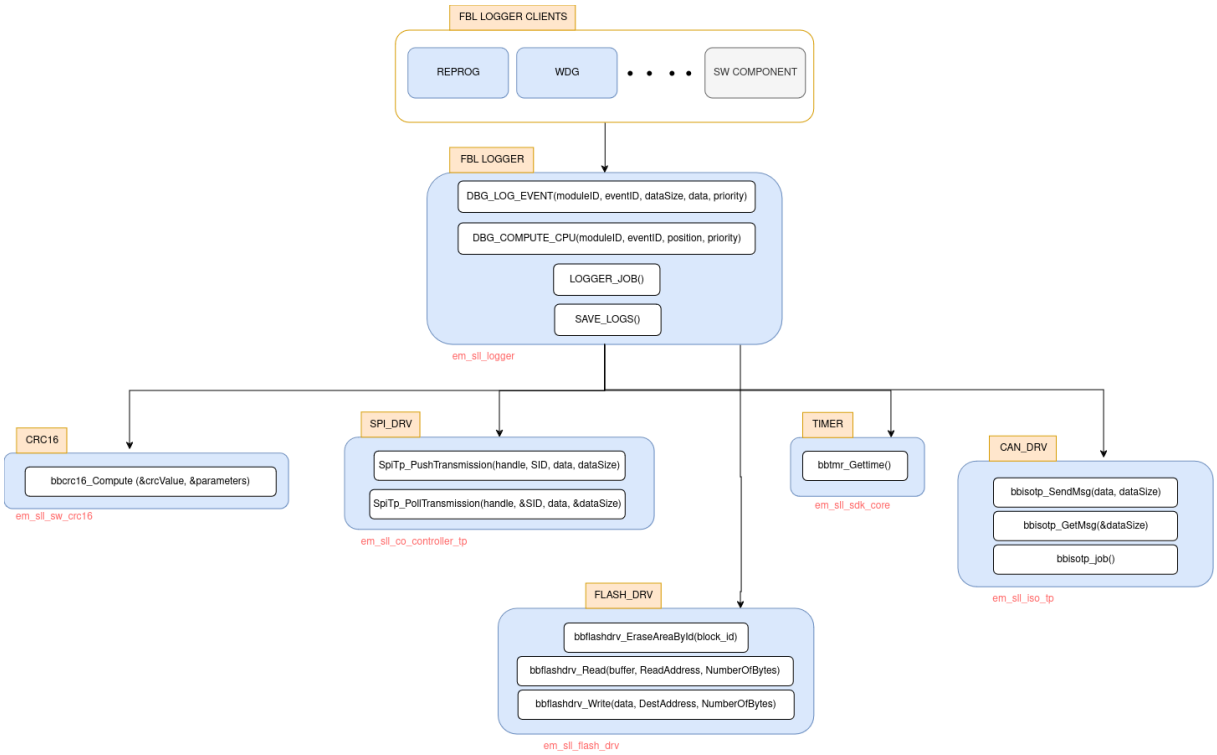


FIGURE 4.27 – Dépendances du Logger L4

Ce quatrième livrable va exposer publiquement une macro de fonction :

SAVE_LOGS()

Elle permettra par exemple, d'enregistrer tout le contenu de la RAM en Flash ou de le diffuser via le bus CAN après que certaines conditions seront respectées :
— Après un nombre d'événements (configurable via "configtool")
— Après chaque période de temps (configurable via "configtool")
— Avant un reset
— Après chaque exécution de la mainLoop (boucle principale)

Diagramme de séquence

Le diagramme de séquence ci-dessous montre la phase d'initialisation du Logger en mode MIXED.

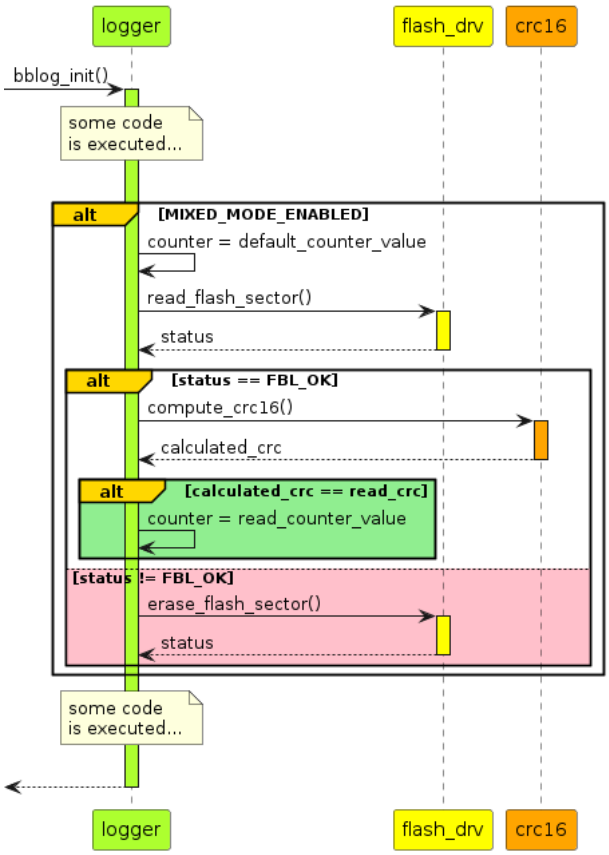


FIGURE 4.28 – Initialisation du Logger en mode MIXED

Le diagramme de séquence ci-dessous permet de comprendre le stockage des évènements en mode MIXED.

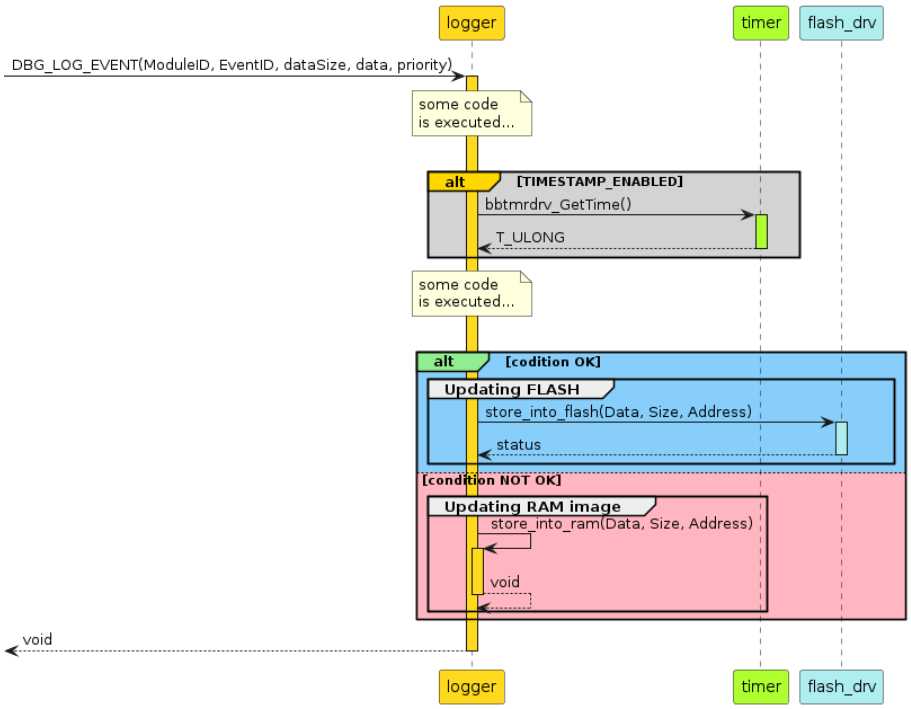


FIGURE 4.29 – Stockage des événements en mode MIXED

4.5.3 Développement

Les tests MISRA-2012 ont été effectués, certains warnings(avertissements) sont apparus et corrigés par la suite.

4.5.4 Résultats de tests unitaires

Les tests unitaires ont été effectués et le rapport généré, montrant une couverture optimale du code développé.

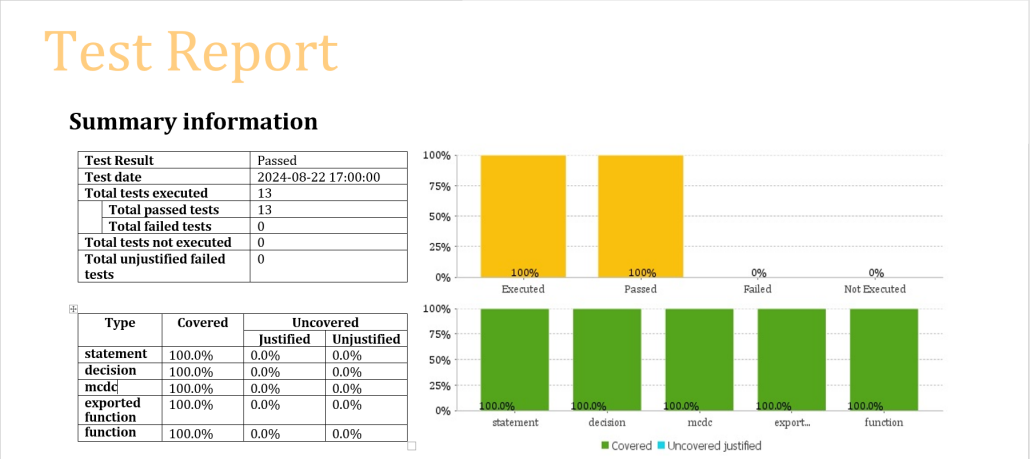
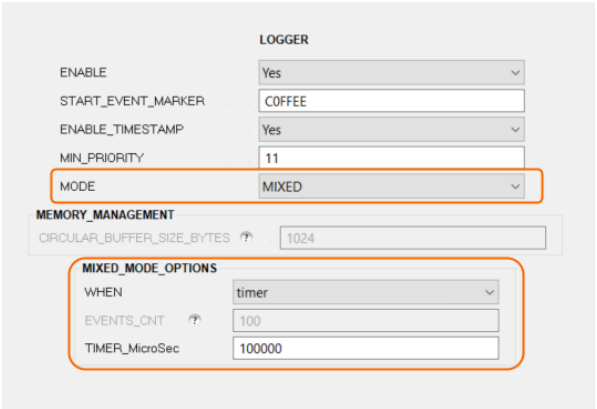


FIGURE 4.30 – Résultats de tests unitaires L4

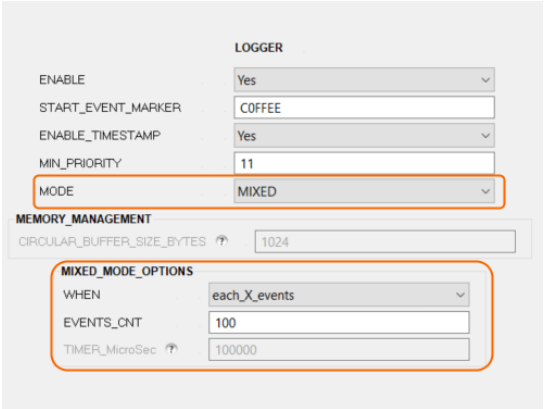
4.5.5 Cas d'utilisation

Le diagramme de séquence illustrant l'exemple d'utilisation est le même que le deuxième livrable.

Les figures ci-dessous illustrent le mode MIXED développé et configurable, qui permet d'écrire en Flash à intervalles réguliers, ou après un certain nombre d'événements,... (tout ce qui est en gris n'est pas pertinent, car il ne fait pas partie de la configuration du module).



(a) Chaque période de temps



(b) Chaque X événements

FIGURE 4.31 – Paramètres Configtool du mode MIXED

La figure ci-dessous présente le contenu d'un bloc mémoire de la Flash après écriture des événements.

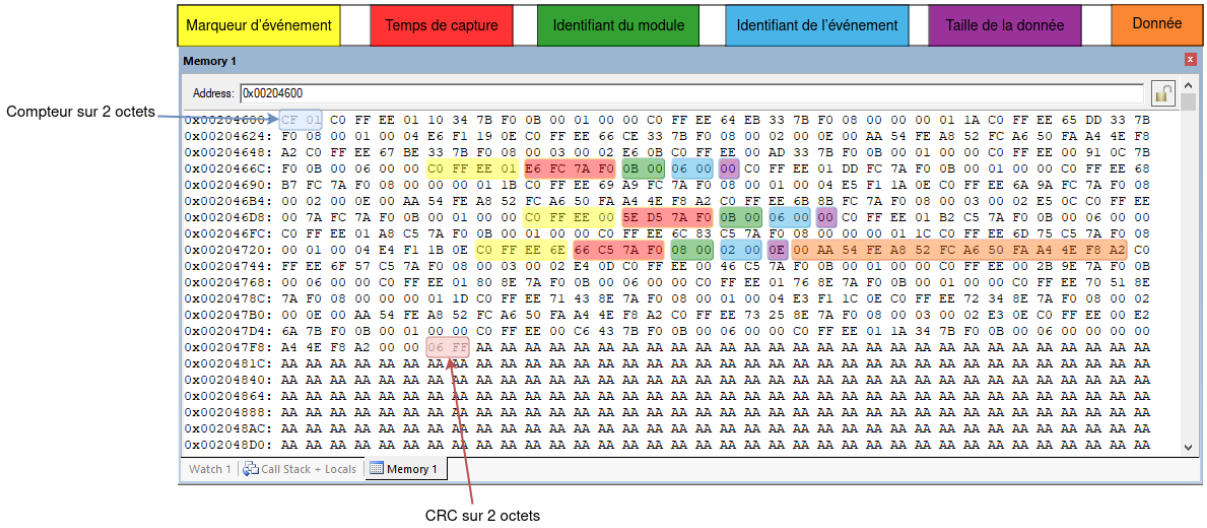


FIGURE 4.32 – Bloc de mémoire Flash

Les événements de type donnée ou de type analyse de performance sont bien écrits en Flash, le compteur les précédent en première position, le CRC est bien calculé et écrit à la fin du bloc mémoire.

À présent, il faut vérifier qu'après un reset (réinitialisation), l'écriture continue là où elle s'est arrêtée.

Les figures ci-dessous illustrent le contenu d'un bloc mémoire Flash avant et après un reset (réinitialisation).

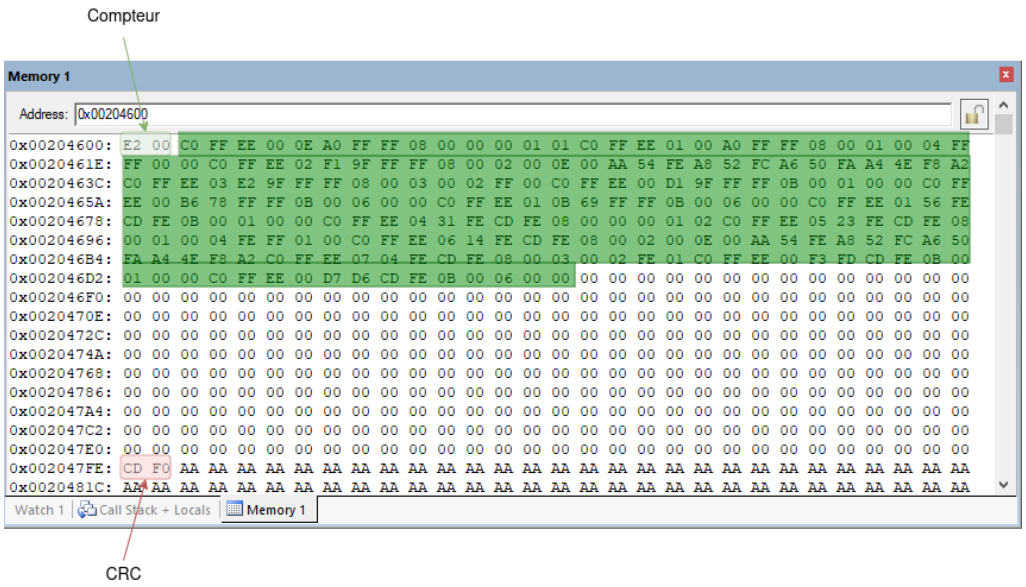


FIGURE 4.33 – Bloc mémoire Flash avant reset

Avant le reset, le contenu du bloc mémoire Flash a été rempli avec ce qui est en vert, avec le compteur correct.

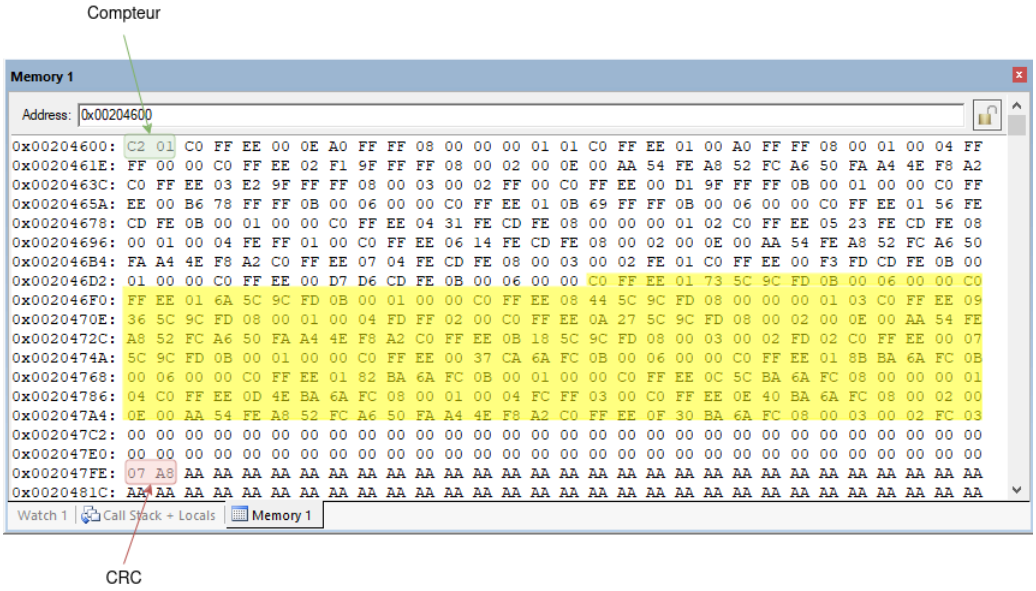


FIGURE 4.34 – Bloc mémoire Flash après reset

Après le reset, tout ce qui est en jaune a été écrit en continuité avec ce qui existait au préalable, sans le modifier, avec le compteur correct, ce qui montre l'efficacité du module développé.

Les captures ci-dessus proviennent du débogueur. La récupération de ce dump mémoire pour le post-traitement peut être réalisée à l'aide du débogueur, comme pour les livrables précédents. Cependant, une méthode plus simple et pratique consiste à ajouter une fonctionnalité à la macro LOGGER_JOB(). Cette fonctionnalité a une logique différente selon qu'il s'agit du Maître ou de l'Esclave. Lorsqu'une trame CAN spécifique est reçue, le contenu du bloc mémoire Flash sera transmis via le bus CAN.

Le diagramme de séquence ci-dessous illustre ce fonctionnement.

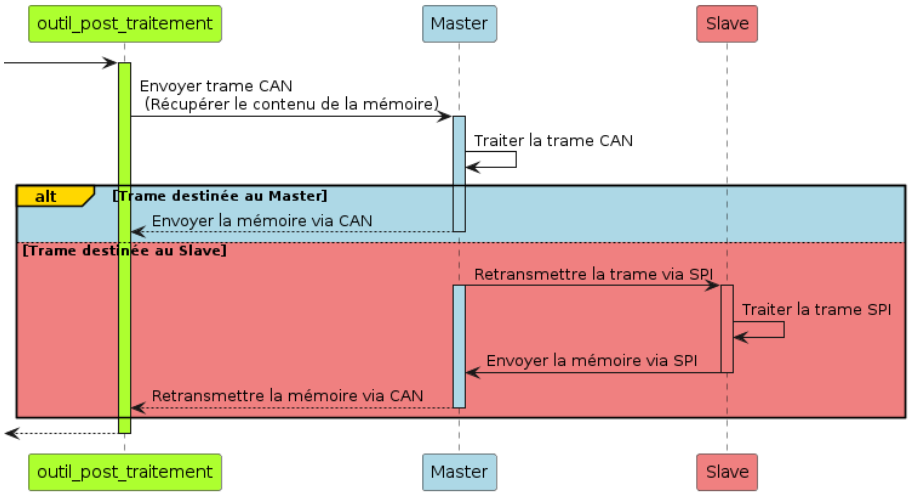


FIGURE 4.35 – Récupérer le contenu de la mémoire

La figure ci-dessous illustre le bon fonctionnement de cette nouvelle fonctionnalité qui permet de récupérer le contenu de la mémoire via CAN.

FIGURE 4.36 – Trames CAN de la mémoire

La mémoire a pu être récupérée selon la séquence suivante :

1. En envoyant une trame de requête (première trame sur la figure).
2. Après la réception de la première trame de données (deuxième trame sur la figure), qui commence par "1" et est suivie de la taille de la mémoire à récupérer, ici 0x200 (512 octets), l'outil envoie immédiatement une trame de contrôle de flux (troisième trame sur la figure). Cette trame, débutant par "3", demande au transmetteur d'envoyer toutes les trames consécutives dès que possible. Ces trames, commençant par "2", sont suivies de leur numéro de séquence respectif.

L'outil va regrouper la première trame et toutes les trames consécutives pour reconstruire la mémoire, et ainsi permettre d'afficher les résultats attendus (figures ci-dessous).

	Event_counter	Timestamp	Module_ID	Event_ID	Data_size	Data
1	0x0	0xffff7c6b	0x8 BBWDGDRV	0x0 EVENT_INIT_TIMEOUT_REG	0x1	['0x1']
2	0x1	0xffff7c60	0x8 BBWDGDRV	0x1 EVENT_REFRESH_TIMEOUT_REG	0x4	['0xff', '0xff', '0x0', '0x0']
3	0x2	0xffff7c53	0x8 BBWDGDRV	0x2 EVENT_INIT_TRIGGER_INT_REG	0xe	['0x0', '0xaa', '0x54', '0xfe', '0xa8', '0x52', '0xfc', '0xa6', '0x50', '0xfa', '0xa4', '0x4e', '0xf8', '0xa2']
4	0x3	0xffff7c47	0x8 BBWDGDRV	0x3 EVENT_INIT_RST_ENB	0x2	['0xff', '0x0']

(a) Événements

	Event	Observation time (ms)	Min time (ms)	Max time (ms)	Avg time (ms)	Frequency (ms)
1	PROCESSING_TIME	90.24	14.29	14.29	14.29	0.04
2	PROCESSING_TIME_FLASH_ERASE_BYID	90.23	4.02	4.02	4.02	0.04

(b) Analyse des performances

FIGURE 4.37 – Résultats obtenus avec l'outil de post-traitement à partir des trames CAN

Ces résultats démontrent que, quels que soient les modes sélectionnés, ils produisent des résultats cohérents et fiables. Cette uniformité dans les résultats valide la solution développée et assure que les performances restent constantes indépendamment du mode choisi. Ainsi, la solution répond de manière fiable aux attentes du projet, garantissant une performance stable dans divers contextes d'utilisation.

Chapitre 5

Conclusion

5.1 Bilan du travail effectué

Dans un premier temps, le stockage en RAM a été validé en respectant les exigences logicielles. Par la suite, l'outil de post-traitement a été développé. Cet outil permet d'afficher les logs de manière claire et concise, facilitant l'analyse avec un minimum d'effort de la part du développeur utilisant la solution (module FBL Logger + outil de post-traitement).

Ensuite, la partie analyse de performance, incluant le calcul du temps d'exécution d'une fonction, toutes les mesures de performance, et l'affichage sous un format clair, a été réalisée. Le mode CAN, où les logs sont diffusés sur le bus CAN, a également été implémenté. Cette fonctionnalité prend en compte les cartes avec deux puces : un Maître (avec CAN) et un Esclave (sans CAN). L'outil de post-traitement a été mis à niveau pour inclure une fonctionnalité permettant de communiquer avec la carte via CAN, décryptant les trames reçues et les affichant dans le même format, favorisant ainsi l'analyse.

Enfin, le stockage en mémoire non volatile (Flash) a été implémenté en respectant les exigences du client, avec l'ajout d'une option permettant de récupérer le contenu de la mémoire via une requête CAN, rendant l'analyse plus rapide et pratique.

En conclusion, les objectifs fixés au début ont été atteints, et quelques fonctionnalités supplémentaires, très utiles à l'équipe, ont été ajoutées.

5.2 Difficultés rencontrées et solutions élaborées

Le développement de la solution s'est effectué sur des bancs de tests équipés de cartes projet, lesquelles correspondent au produit final destiné à être intégré dans les véhicules. Ces cartes disposaient d'un espace mémoire Flash restreint, principalement pour des raisons de sécurité. En conséquence, le code développé dépassait souvent l'espace mémoire alloué, entraînant des erreurs de compilation. Cette contrainte de mémoire a posé des difficultés tout au long du processus de développement, en particulier lors de l'implémentation du dernier livrable.

D'abord, Le premier obstacle consistait à comprendre un code existant de milliers de lignes sans documentation. Grâce à mes compétences en analyse de code et en reverse

engineering, j'ai pu aborder cette tâche complexe. J'ai procédé à une lecture approfondie du code, en identifiant les structures logiques et en retraçant les dépendances entre les différentes parties du système. Ensuite, il a fallu investiguer les causes du problème de dépassement de mémoire, ce qui s'est avéré bénéfique.

J'ai ainsi découvert l'importance du fichier de script de linker, un élément clé dans le processus de compilation. Typiquement, il s'agit d'un fichier ".ld" (pour les compilateurs GCC), mais dans ce projet spécifique, c'était un fichier ".sct" (utilisé pour les compilateurs ARM). Ce fichier détermine comment les sections de code (comme .text, .data, .bss) sont placées en mémoire après la compilation, en mappant les variables et registres aux adresses matérielles spécifiques de la carte.

Grâce à ce fichier de script de linker, j'ai pu explorer les allocations mémoire en détail. Le fichier ".map", généré après compilation, fournit des informations précieuses sur l'espace mémoire occupé par le code, les variables et les constantes. Il m'a permis de vérifier combien d'espace le code compilé prenait en mémoire, et d'identifier les zones nécessitant une optimisation.

Pour optimiser la mémoire utilisée, j'ai utilisé des directives du préprocesseur (#ifdef, #ifndef,...) afin d'inclure uniquement le code nécessaire, réduisant ainsi l'empreinte mémoire du projet. Cette première solution a permis de résoudre temporairement le problème de dépassement de mémoire.

Cependant, en arrivant au quatrième livrable, le module EEPROM déjà existant ne permettait plus la compilation en raison des contraintes mémoire. La solution mise en œuvre a consisté à développer une version allégée ("lightweight") du module EEPROM, compatible avec le module FBL Logger, tout en respectant les exigences du projet. Cette version optimisée a permis de répondre aux contraintes strictes de mémoire et aux exigences fonctionnelles, garantissant ainsi la réussite du projet.

5.3 Bilan personnel

Ce projet de fin d'études m'a offert une expérience à la fois enrichissante et formatrice. En effet, j'ai pu approfondir mes connaissances dans divers domaines, à savoir, les protocoles de communication utilisés en automobile (ISO-TP, UDS, SPI-TP), ainsi que les technologies de mémoires non volatiles, notamment la Flash, l'EEPROM et la RAM persistante. J'ai également acquis une maîtrise plus solide des outils logiciels et matériels essentiels pour le débogage et l'analyse des systèmes embarqués, tels que le Logic Analyzer, PCAN, Canalyzer, Lauterbach et J-Link. Par ailleurs, ce projet m'a permis de perfectionner mes compétences en programmation avec les langages C et Python, tout en appliquant rigoureusement les règles de codage MISRA, cruciales pour garantir la fiabilité des systèmes critiques.

J'ai eu l'opportunité de découvrir de nouveaux concepts, comme le Bootloader dans un contexte automobile, et de saisir toute la complexité associée. L'utilisation de tout le matériel disponible a été particulièrement impactante.

Enfin, j'ai assumé la responsabilité qui incombe à la réalisation d'un projet répondant à un besoin client, facilitant ainsi la transition entre ma vie étudiante et ma vie professionnelle.

5.4 Perspectives futures pour le module développé

Aujourd'hui, le module répond aux exigences du cahier des charges. Cependant, certaines fonctionnalités et optimisations pourraient encore être ajoutées :

— **Analyseur de pile d'appels** : Permet de tracer l'exécution des fonctions et ainsi d'identifier la source des problèmes en cas de défaillance ou de dysfonctionnement.
— **Profilage de performance** : Permet de détecter les goulots d'étranglement liés à la surcharge du Central Processing Unit (CPU).
— **Minimiser l'impact du FBL Logger** : Développer un outil permettant, à travers du click engineering, de configurer la Direct Memory Access (DMA) sur différentes puces. La DMA est un périphérique présent sur la quasi-totalité des puces, permettant le transfert de données entre mémoire et périphérique, entre deux zones de mémoire, ou entre deux périphériques sans intervention du CPU.

Bibliographie

[1] Embitel. *What is Flash Bootloader and Nuances of an Automotive ECU Re-programming*. Accessed : 2024-08-21. 2023. URL : https://www.embitel.com/blog/embedded-blog/what-is-flash-bootloader-and-nuances-of-an-automotive-ecu-re-programming.

[2] Entrust. *What are Hardware Security Modules (HSMs) ?* Accessed : 2024-08-21. 2023. URL : https://www.entrust.com/fr/resources/learn/what-are-hardware-security-modules.

[3] MuhannadAjjan. *Circular Buffer Animation.gif*. Accessed : 2024-08-29. 2015. URL : https://en.wikipedia.org/wiki/Circular_buffer.

[4] Wikipedia. *Modified condition/decision coverage*. Accessed : 2024-08-21. 2023. URL : https://en.wikipedia.org/wiki/Modified_condition/decision_coverage.

[5] Wikipedia. *Serial Peripheral Interface*. Accessed : 2024-08-21. 2023. URL : https://fr.wikipedia.org/wiki/Serial_Peripheral_Interface.

[6] Wikipedia Contributors. *ISO 15765-2*. Accessed : 2024-08-21. 2024. URL : https://en.wikipedia.org/wiki/ISO_15765-2.

[7] Wikipedia contributors. *Unified Diagnostic Services*. Accessed : 2024-08-30. 2024. URL : https://en.wikipedia.org/wiki/Unified_Diagnostic_Services.

Annexe A

Annexe

A.1 Documentation technique

A.1.1 Bootloader

Pour retourner à l'endroit où vous étiez, cliquez ici

Pour bien comprendre le Bootloader, revenant d'abord à l'essentielle :

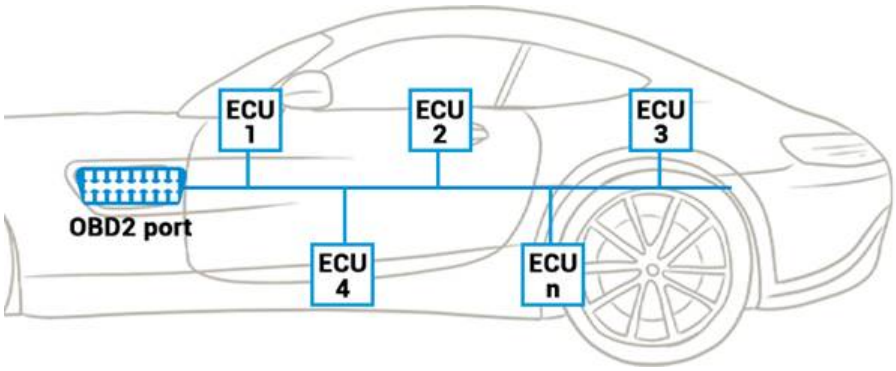


FIGURE A.1 – Voiture moderne

Les unités de commande électroniques (ECU) sont des composants qui contrôlent, en plus des fonctions essentielles, telles que les performances du moteur et la direction assistée, contrôlent également les fonctions de confort et de sécurité, telles que l'aide au stationnement, les sièges mémoire et le déploiement des airbags. Les véhicules modernes peuvent contenir plus de 100 ECUs, chacune partageant des informations avec d'autres ECU sur le bus CAN [1].
Le bus CAN (Controller Area Network) est un système de communication utilisé dans les véhicules/machines pour permettre aux ECUs de communiquer entre eux – sans un chef d'orchestre. Par exemple, le bus CAN permet un partage rapide et fiable des informations entre les freins et le moteur de votre voiture.
Physiquement, tous les ECUs sont connectés sur un bus à deux fils constitué d'une paire torsadée : CAN high et CAN low. Les fils sont souvent codés par couleur : CAN high est jaune, CAN low est vert [2].

FIGURE A.2 – Câble bus CAN

Comment ça marche ? - L'application logicielle ABS portée dans le matériel de l'ECU est capable de récupérer la vitesse du véhicule en tant qu'entrée, elle est conçue pour réduire le freinage sur les roues, en fonction de cette entrée.

Si nous zoomons, un ECU se compose de trois éléments principaux :

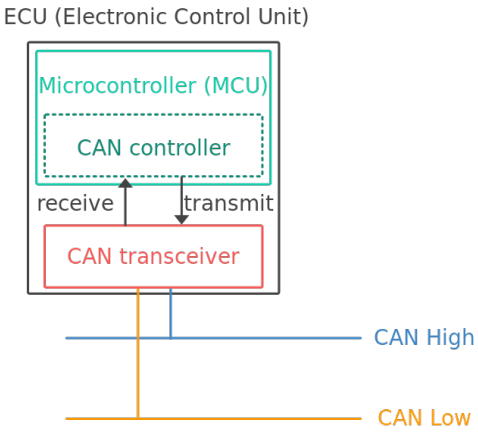


FIGURE A.3 – Unité de contrôle électronique

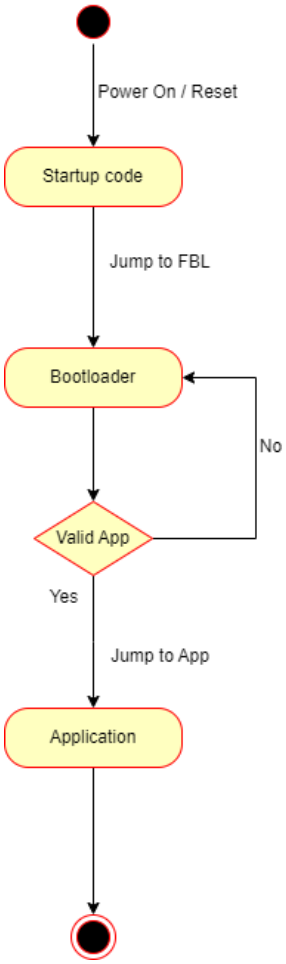
— **Microcontrôleur :** Le MCU est le cerveau de l'ECU : il interprète les messages CAN entrants et décide quels messages transmettre, de ce fait, c'est lui qui intègre le logiciel qui va être reprogrammé.

— **Contrôleur CAN :** le contrôleur est généralement intégré au MCU et garantit que toutes les communications adhèrent au protocole CAN (codage des messages, détection d'erreurs, arbitrage, etc.), éliminant ainsi la complexité du MCU.

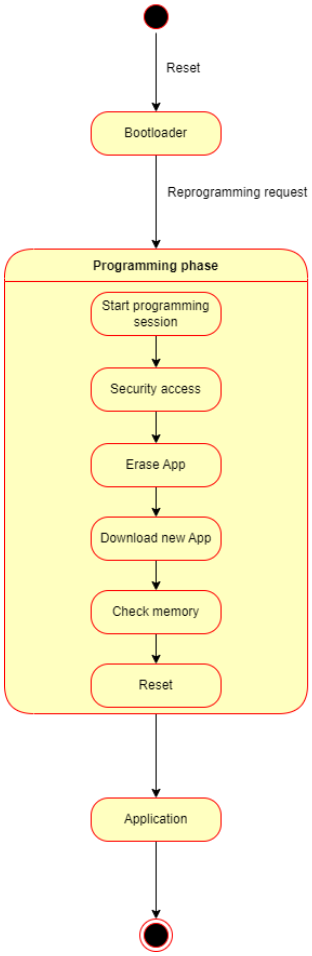
— **Émetteur-récepteur CAN :** L'émetteur-récepteur CAN connecte le contrôleur CAN aux fils CAN physiques, convertissant les données du contrôleur en signaux différentiels pour le bus CAN et vice versa. Il assure également une protection électrique.

La responsabilité de la reprogrammation (mise à jour) de l'ECU est confiée au FBL. Le Flash Bootloader est un logiciel qui occupe la mémoire non-volatile Flash de l'unité de contrôle électronique, il est exécuté lors du démarrage du système.
Le Bootloader a deux fonctions principales :

1. Vérifier que l'application existe, est valide et authentique pour l'exécuter.
2. Mettre à jour le logiciel intégré (l'application) qui contrôle l'ECU



(a) Vérifier l'application



(b) Mettre à jour l'application

FIGURE A.4 – Missions du Bootloader

A.1.2 Stratégie des tests unitaires

Pour retourner à l'endroit où vous étiez, cliquez ici

La stratégie a été la suivante :

— La réalisation des graphes de flot de contrôle (CFG, pour Control Flow Graph) : est une représentation sous forme de graphe de tous les chemins qui peuvent être suivis par un programme durant son exécution.

La figure ci-dessous illustre la transformation d'une fonction en CFG.

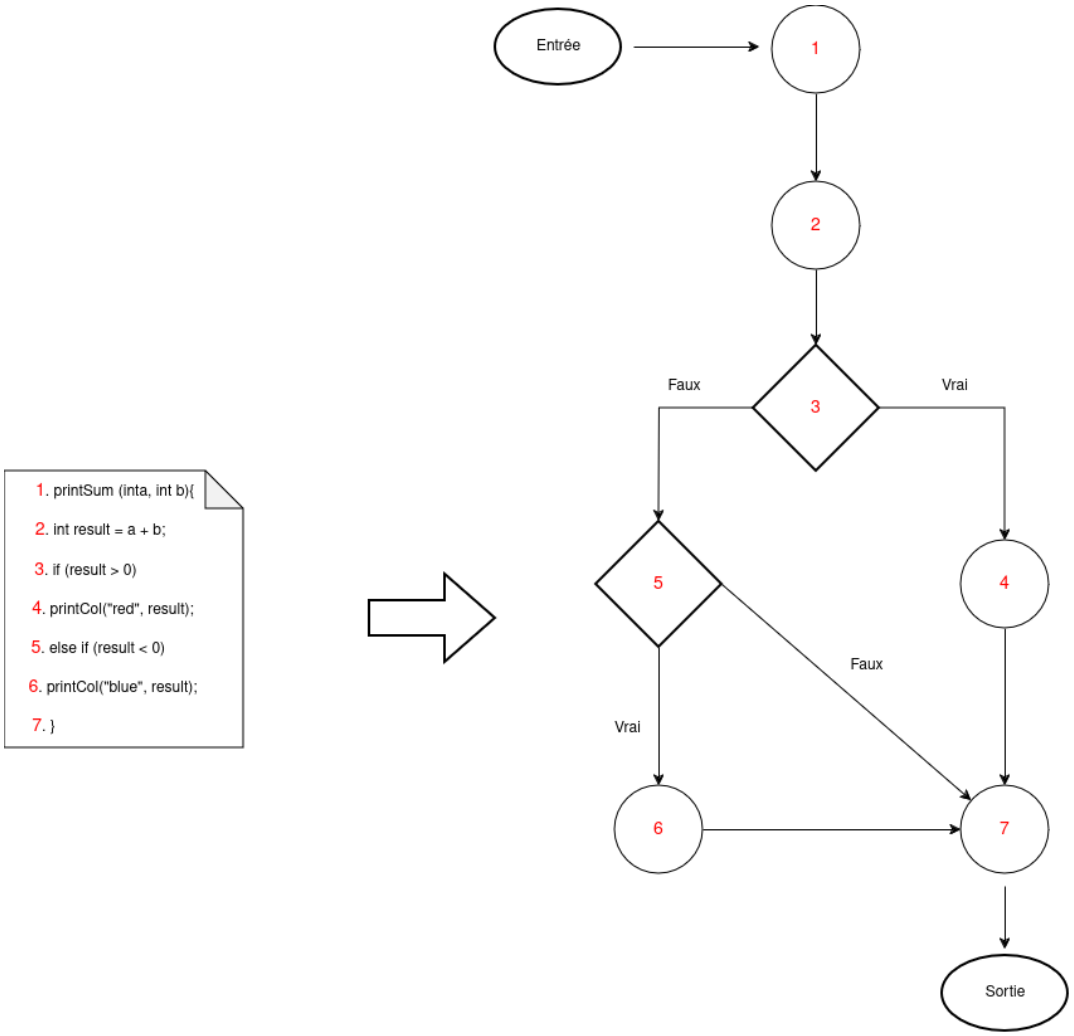


FIGURE A.5 – Graphe de flot de contrôle

— L'identification des chemins permet d'identifier les cas de tests nécessaires pour avoir une couverture optimale, selon les critères de tests couverture de condition/décision modifiée (MC/DC, pour Modified Condition/Decision Coverage), vérifiables à l'aide du logiciel propriétaire de Continental utilisé pour les tests.
MC/DC a besoin de tous les éléments ci-dessous pendant les tests :
  — Chaque point d'entrée et de sortie est invoqué.
  — Chaque décision prend tous les résultats possibles.
  — Chaque condition d'une décision prend tous les résultats possibles.
  — Il est démontré que chaque condition d'une décision affecte indépendamment le résultat de la décision.
L'indépendance d'une condition est démontrée en prouvant qu'une seule condition change à la fois.

MC/DC est utilisé dans les normes de développement de logiciels avioniques DO-178B et DO-178C [4] pour garantir des tests adéquats du logiciel le plus critique (niveau A), qui est défini comme le logiciel qui pourrait assurer (ou empêcher l'échec) un vol continu en toute sécurité et atterrissage d'un avion.

De plus, la NASA exige une couverture MC/DC à 100% pour tout composant logiciel critique.

A.1.3 HSM

Pour retourner à l'endroit où vous étiez, cliquez ici

Les modules matériels de sécurité (HSM) sont des périphériques physiques renforcés et inviolables qui sécurisent les processus cryptographiques en générant, protégeant et gérant les clés utilisées pour le cryptage et le décryptage des données et la création de signatures numériques et de certificats [2].

A.1.4 ISO-TP

Pour retourner à l'endroit où vous étiez, cliquez ici

La norme ISO-TP (ISO 15765-2) [6] est une couche de transport utilisée principalement pour la communication dans les réseaux automobiles, comme le bus CAN. Elle permet de transmettre des messages de plus grande taille en les fragmentant sur plusieurs trames, car les trames de base du bus CAN sont limitées à 8 octets et sur CAN FD à 64.

1. Trame unique (Single Frame - SF) Lorsque le message à transmettre tient dans une seule trame (moins de 8 octets), une trame unique (SF) est utilisée. Le premier octet contient la taille totale des données, et les suivants sont les données.
2. Trame de premier segment (First Frame - FF)

   Si le message dépasse la taille d'une trame unique, un first frame (FF) est utilisé. Le premier octet contient un identifiant spécifiant qu'il s'agit d'une trame de premier segment, suivi de la taille totale du message. Les octets suivants contiennent les premiers octets des données.
3. Trames consécutives (Consecutive Frame - CF)

   Une fois le first frame envoyé, des consecutive frames (CF) sont utilisées pour transmettre les parties restantes du message. Chaque trame CF commence par un numéro d'index de séquence, suivi des octets de données.
4. Trame de contrôle de flux (Flow Control - FC) Le récepteur envoie des trames de contrôle de flux (FC) pour gérer la réception des données. Cela permet de demander au transmetteur de ralentir ou d'arrêter l'envoi de trames, en fonction de la capacité de traitement du récepteur. Il existe trois types de trames FC :
   - Continuer à envoyer : le récepteur est prêt à recevoir plus de trames.
   - Attendre : le récepteur demande une pause temporaire.
   - Débordement : le récepteur est incapable de traiter davantage de trames.

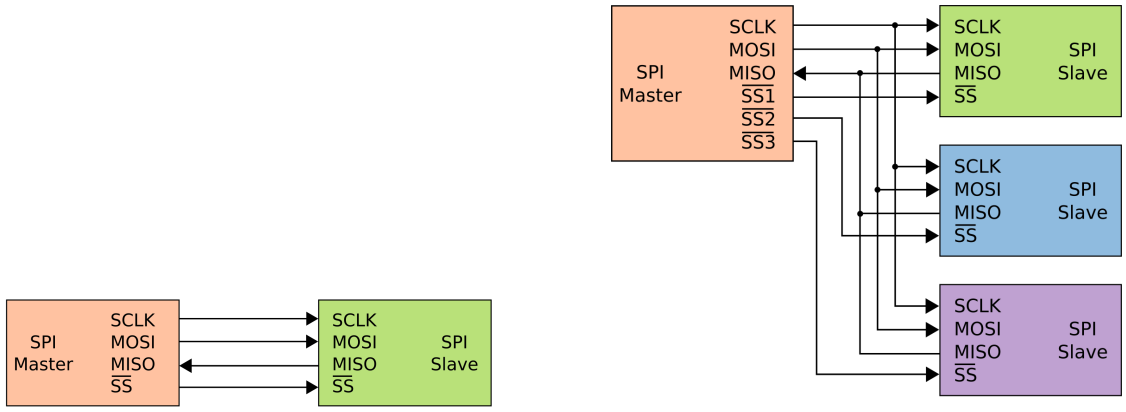
A.1.5 SPI

Pour retourner à l'endroit où vous étiez, cliquez ici

Le SPI (Serial Peripheral Interface) est un protocole de communication synchrone qui opère en mode full-duplex[5], utilisé pour la transmission de données entre un maitre (souvent un microcontrôleur) et un ou plusieurs esclaves (périphériques). Le maitre génère un signal d'horloge (SCLK) pour synchroniser la communication, tandis que les données sont échangées via deux lignes :

- MOSI (Master Out Slave In) : le maître envoie des données à l'esclave.
- MISO (Master In Slave Out) : l'esclave envoie des données au maître.

Une ligne SS (Slave Select) est utilisée pour sélectionner l'esclave avec lequel communiquer.



(a) Un seul esclave (b) Trois esclaves

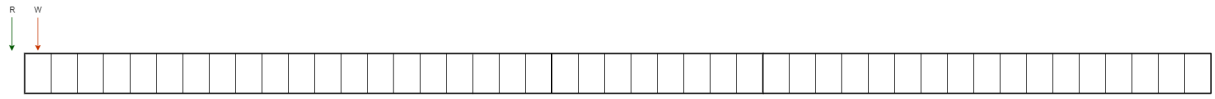
FIGURE A.6 – Protocole SPI

A.1.6 Fonctionnement de l'approche 2 du livrable 1

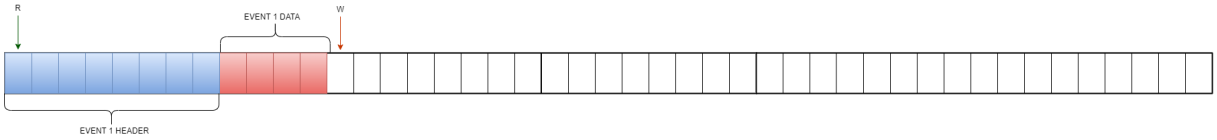
Pour retourner à l'endroit où vous étiez, cliquez ici

Utilisation d'un pointeur pour pointer vers l'événement valide le plus ancien : pour cette approche, en plus du pointeur d'écriture (W), un deuxième pointeur (R) est ajouté. Ce pointeur (R) est utilisé pour pointer vers l'événement valide le plus ancien, c'est-à-dire celui qui n'a pas été totalement ou partiellement écrasé par un nouvel événement. La validité de ce pointeur sera vérifiée avant chaque écriture d'un nouvel événement, afin de décider s'il doit rester en place ou être déplacé vers l'événement valide le plus ancien suivant.

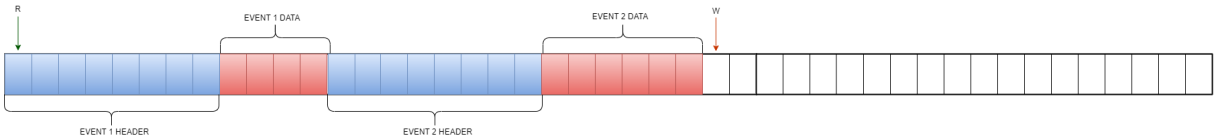
Au début :
— **R** : ne contient aucune adresse valide
— **W** : pointe vers le début du tampon circulaire



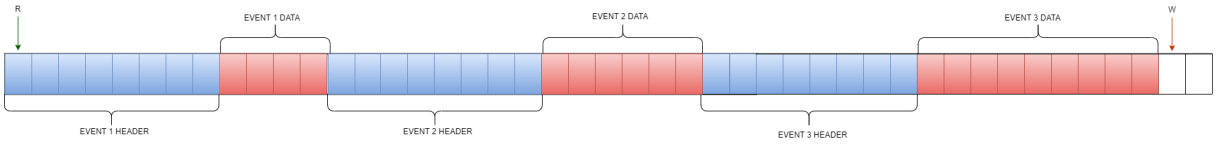
Le premier événement est écrit, (W) est incrémentée et (R) est initialisée à une adresse valide.



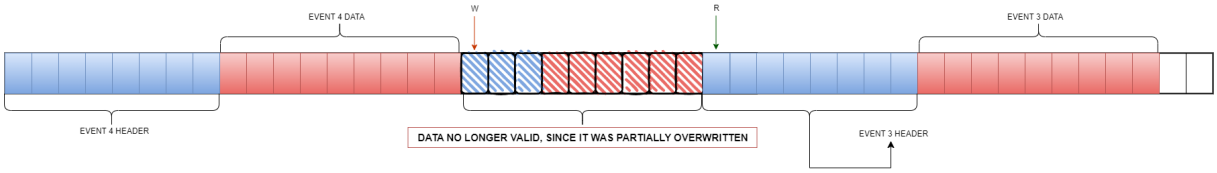
Le deuxième événement est écrit, (W) est incrémentée, par contre l'événement le plus ancien (R) est toujours valide, donc son adresse n'est pas mise à jour.



Le troisième événement est écrit, (W) est incrémentée, par contre l'événement le plus ancien (R) est toujours valide, donc son adresse n'est pas mise à jour.



L'écriture du quatrième événement est planifiée, mais la taille restante n'est pas suffisante pour stocker l'événement complet (tampon circulaire plein). Par conséquent, l'adresse d'écriture (W) est réinitialisée au début du tampon. En procédant ainsi, l'événement 1 et, partiellement, l'événement 2 sont écrasés. De ce fait, (R) sera alors déplacée pour pointer au début de l'événement 3.



A.1.7 UDS

Pour retourner à l'endroit où vous étiez, cliquez ici

Unified Diagnostic Services (UDS) est un protocole de diagnostic utilisé dans les systèmes de gestion des véhicules automobiles. Il est conçu pour permettre aux outils de diagnostic de communiquer avec les unités de contrôle électronique (ECU) dans un véhicule pour des tâches telles que le diagnostic, la programmation et la gestion des défauts. UDS est défini par la norme ISO 14229, qui est un standard international pour les services de diagnostic automobile[7].