Création d'un outils d'intégration continue

Rapport d'élève ingénieur
Stage de 2ème année
Filière F2 : Génie Logiciel et Systèmes Informatiques

Présenté par : **Rémi CHASSAGNOL**

Responsable ISIMA : TODO

Jeudi 23/03/2023

Stage de 5 mois

Campus des Cézeaux. 1 rue de la Chébarde. TSA 60125. 63178 Aubière CEDEX

Table des matières

Remerciements

Les remerciements !!

Table des figures

Résumé

Le super résumer !

Mots-clés : **C/C++**, **intégration continue**, **tests**, **systèmes embarqués**, **émulation**

Abstract

the amazing abstract

Keywords : **C/C++**, **continuous integration**, **testing**, **embeded system**, **emulation**

Introduction

Introduction
plan !

Première partie

Contexte du projet

1 CKsquare

CKsquare est une entreprise d'ingénierie, d'étude et de conseil spécialisée dans la conception de systèmes de payement automatisés. La société, au départ nommée cbsquare, a été créée en 2003 par Emmanuel Bertrand et compte aujourd'hui plus de 30 employés. Au départ, l'entreprise se tourne vers le secteur des stations de lavage auto en créant une gamme de distributeurs de jetons. Par la suite, elle élargie sa gamme de produits articulés autour de la monétique et se lance dans la conception de ses propres cartes électroniques.

L'entreprise conçoit des bornes principalement les stations de lavage auto ainsi que les laveries mais s'intéresse aussi à d'autres marchés comme l'hôtellerie. Un projet de casiers automatisés est aussi en train de se mettre en place. Ce nouveau projet innovent et écologique (car il privilégiera les producteurs locaux et évitera les voyages en voiture pour se rendre dans les grandes surfaces) permettra de faire face au déclin des stations de lavage auto à cause des sécheresses de plus en plus fréquentes. Aujourd'hui, plus de 40000 stations de lavage auto sont équipé de bornes CKsquare.

CKsquare est un groupe composé de quatre sociétés chacune chargée de la conception d'une partie des produits. Parmi ces entreprises, on compte la société M-Innov qui se charge de la conception et de l'installation des bornes et systèmes monétiques pour les aires de services, campings, parking, ou hôtels. La société Mecasystem International qui se charge de la tôlerie et de la mécanique pour les bornes CKsquare et M-Innov. Enfin, en plus de CKsquare, il y a la société Ehrse qui se charge de la fabrication des cartes électroniques. Ces quatre sociétés travaillent en coopération ce qui leur permet d'avoir la maitrise de la conception et de la fabrication de tous les composants de leurs produits.

L'objectif de la société CKsquare est de pouvoir fournir des produits configurables et adaptables aux besoins des différents clients. C'est pour cela que les bornes possèdent beaucoup d'options et que l'entreprise entretient un savoir faire quant à la gestion de la plupart des systèmes de payements. De plus, une équipe SAV reste à l'écoute du besoin des clients ce qui permet à l'entreprise de concevoir des solutions encore plus spécifiques et personnalisées.

Un des gros atouts de la société est son savoir faire concernant l'utilisation des cartes électroniques. En effet, les bornes CKsquare sont toutes équipées de cartes électroniques beaucoup plus fiables et moins énergivores que des PC. Cependant, ces cartes doivent assurer beaucoup de fonctionnalités et gérer un grand nombre de composants ce qui pose de gros problèmes en terme d'optimisation du stockage. De plus, l'utilisation d'outils de payements implique encore plus de contraintes comme par exemple la nécessité de s'assurer que toutes les ventes sont correctement enregistrées puis envoyées sur le serveur.

2 Travail demandé

L'objectif est de réaliser un outil servant à faire de l'intégration continue pour valider les fichiers de la partie commune du code utilisé sur les bornes. L'outil doit permettre l'écriture de tests pour valider le code et doit pouvoir être automatisé dans une pipeline Gitlab. De plus, l'outil sera utilisé pour tester du code exécuté sur du matériel embarqué, il faudra donc un moyen de tester des fonctions qui interagissent avec le matériel électronique. Pour ce faire, il faudra émuler les interactions avec le matériel en créant des fonctions et des structures de données qui réagiront comme les composants électroniques. Par exemple, si une fonction doit modifier un registre sur une carte, il faut pouvoir émuler le registre pour vérifier

que les bonnes modifications ont été apportées. Il faudra aussi des fonctionnalités permettant de simuler l'interaction d'un utilisateur avec un composant pendant les tests. Par exemple, on doit pouvoir faire en sorte de tester le comportement du système lorsqu'un utilisateur appuie sur une séquence de touches. Il sera donc nécessaire d'émuler la mémoire des composants électroniques, les fonctions qui permettent d'interagir avec. Il faudra aussi des fonctions permettant de simuler les actions d'un utilisateur.

La première tâche sera de trouver le framework de test adapté pour la conception de l'outil. Le code à tester est écrit en C, cependant, la société souhaiterait aussi pouvoir tester les bibliothèques écrites par l'équipe Qt. Il serait donc intéressant que l'outil soit aussi adapté au C++.

Étant donné que les tests seront exécutés dans une pipeline (donc dans un docker), il faudra s'assurer que le code puisse compiler sous Linux. De plus, le code étant à la base fait pour s'exécuter sur une carte électronique, cela nécessitera d'émuler les composants pour que le code puisse fonctionner correctement. Il sera aussi nécessaire de simuler les interfaces permettant au code d'interagir avec les composants émulés en utilisant différents protocoles de communication comme le SPI ou encore le Cctalk. En plus de cela, il faudra pouvoir remplacer une partie des bibliothèques de la carte pour pouvoir simuler les composants.

Le résultat final doit être un outil qui doit pouvoir être facilement réutilisable et adaptable. La documentation et la structure du code doit pouvoir permettre d'aisément modifier ou copier les différents éléments. Par exemple, il faudra que tous les composants simulés aient la même structure et que cette structure soit suffisamment simple et générique pour pouvoir être copiée pour la création d'un nouveau composant.

À noter que l'objectif du projet n'est pas d'écrire des tests. Des tests ont été écrits durant le projet mais ces derniers ont pour objectif de valider le bon fonctionnement des composants simulés et non celui du code de la société.

3 Environnement et outils à disposition

parler de l'environnement dans l'entreprise... et des outils - travail seul - aide disponible - pc sous linux - accès à la documentation

Deuxième partie

Réalisation et conception

1 Choix du framework de test

Le but du projet est de concevoir un outil permettant de tester du code, la première tâche à donc été de choisir un framework de test. Le framework de test constitue la base de l'outil, c'est donc un choix assez important. Dans cette partie nous traiterons de la procédure qui a été utilisée pour trouver et comparer des frameworks et des bibliothèques de test.

1.1 Les critères de comparaison

Étant donné le fait que le langage C est très utilisé, il y a beaucoup de choix quand aux différentes bibliothèques de test utilisables. Le premier travail a été de faire un listing des bibliothèques et frameworks de tests existants pour ensuite pouvoir les comparer. Une liste des frameworks disponibles sur Wikipédia [1]. Ce travail de recherche à permis de faire un prés tri et d'éliminer les framework incomplets ou trop peut utilisés. Une fois le listing terminé, il a fallut trouver des critères pour comparer les frameworks.

Tout d'abord, l'équipe de développement souhaitait pouvoir tester à la fois du code **C** et du code **C++** pour certaines parties développées par l'équipe **Qt**. Ce critère était optionnel mais apprécié. À noter que lorsque l'on parle de pouvoir tester du code C++, cela ne prend pas seulement en compte le fait de pouvoir exécuter des fonctions basiques puisque c'est possible avec tous les frameworks C étant donné la compatibilité entre le C et le C++. Pour pouvoir tester du code C++, il faut aussi que le framework soit capable d'interagir avec les structures de donnés fournis par la bibliothèque standard de C++ ou encore de pouvoir traiter des exceptions. Étant donné ce critère, l'idée d'utiliser un framework écrit en C++ a été envisagé.

Un autre critère concerne la modernité et la facilité d'utilisation du framework. Cela peut sembler anodin mais l'écriture des tests est une tâche aussi longue que le développement. Pour ne pas perdre de temps, il est préférable que les tests soient le plus simple possible à mettre en place. De plus, les tests peuvent aussi servir de documentation, c'est donc un avantage non négligeable que d'avoir un framework qui permette d'écrire des tests simples, lisibles et compréhensibles. Enfin, ce critère impacte aussi le temps de conception de l'outil de test car le fait d'utiliser un framework trop complexe aurait nécessité la conception de fonctions et macros (pour réduire la complexité) et rallongé le temps d'écriture de la documentation. À noter que pour valider ce critère, la documentation des différents outils a aussi été étudiée, les frameworks devaient donc fournir une documentation suffisamment claire et précise permettant d'utiliser facilement toutes les fonctionnalités proposées.

Le critère le plus important est celui du statut du développement du framework. En effet, lorsque l'on souhaite utiliser un outil, une question importante à se poser est de savoir ce que l'on peut faire en cas de problème. Ici, il a fallut regarder la taille, la popularité et l'age des projets. En effet, plus un projet est populaire plus il sera facile de trouver de l'aide en cas de problème. De plus, les projet important on souvent beaucoup plus de collaborateurs ce qui peut accélérer la corrections des bugs. Enfin la dates de dernières mis à jours ont aussi été répertoriées car là aussi, il est beaucoup plus simple de résoudre les problèmes sur un projet qui est encore activement maintenu.

D'autres critères ont permis de démarquer les frameworks comme par exemple le fait que les frameworks fournisse des fonctionnalités supplémentaires comme le fait de pouvoir exporter les résultats des

tests dans différents format comme TAP ou XML (utile pour faire des rapport) ou encore des générateur de nombres pseudo aléatoires pour faire des tests avec des entrée aléatoires, ...Une autre fonctionnalité intéressante est que les frameworks exécutent les tests dans des threads séparer ce qui permet de tester des signaux ou encore de ne pas stopper tous les tester pour un plantage.

2 Organisation du projet

2.1 Découverte du code

Découverte du code sur un projet simple. config (solution personnalisée) = premier difficulté, il faut savoir comment configurer le projet.

2.2 Organisation des fichiers

- myosismonnayeur - fichiers de bases - Les fichiers locaux - Fichiers à tester (commun_global, dev_pic) - les fichiers de Tests

2.3 Contrainte pour compiler

2.4 Machines à états

3 Les premiers tests

Tools, machines, cashregister

Fichiers simples au départ car il était très compliquer de compiler le projet sans les bibliothèques et le compilateur de la carte.

3.1 Différents types de fichiers à tester

3.1.1 Tools : fonctions basiques

3.1.2 Cashregister et machines : tester des machines à états

contrainte => pas d'acces aux états, ... on doit tester l'environnement

3.2 Versions tests

3.3 Organisation du projet de test

Organisation du répertoire, philosophie quand à la copie de fichiers lorsqu'il y a des modifications à faire, substitution des bibliothèques.

4 Simulation du stockage

Une partie importante de l'émulation concerne le stockage et il y a plusieurs types composants à simuler, les registres, les eeproms, et la mémoire flash. L'émulation des périphériques des stockage est assez simple puisqu'il s'agit simplement de tableaux de caractères non signés (codés sur 8 bits sur la

plupart des machines). La partie complexe de l'émulation du stockage concerne l'interface qui permet d'interagir avec les périphériques. Il y a deux protocoles qui sont utilisés avec le stockage. Tout d'abord il y a le protocole I²C qui est utilisé avec les registre et certaines eeproms. Ensuite il y a le protocole SPI qui est utilisé avec les eeproms et les mémoire flash.

Dans cette partie nous allons voir comment a été réalisée l'émulation de l'interface permettant d'utiliser le stockage avec les différents protocoles.

4.1 Échec des threads

La première solution qui a été implémentée utilisait des **threads**. L'objectif été de pouvoir simuler les composants de sorte à ce qu'ils se comportent comme les composants réels installés sur la carte. Pour se faire, il été souhaitable que les composants simulés soient actif en même temps que la carte (représentée ici par le programme à tester) et c'est pour cela que les threads ont été utilisés. Le principe était que les composants étaient représentés par des machines à états qui bouclaient dans un état de base jusqu'à ce que le composant soit appelé (donc jusqu'à ce que le programme principale décide de lancer une communication en utilisant un des protocoles cités précédemment). Une fois le composant appelé, la machine à états permettait d'assurer la communication. Pour que les composants simulés s'exécutent en même temps que le programme principale, ils s'exécutaient dans des threads séparés.

4.2 Interface I²C

4.3 Interface SPI

5 Le Cctalk

5.1 Le protocole

5.2 Émulation des composants

6 Le MDB

6.1 Le protocole

6.2 Émulation des composants

7 Déroulement du projet

7.1 Les outils utilisés

7.1.1 Gestion de version

utilisation de git / version modifiée de gitflow. Je suis seul mais j'ai quand même organiser mes branche de façon similaire à gitflow pour ne jamais casser du code qui marche en mergant ou en tentant d implémenté du code qui marche pas.

7.1.2 Gitlab

ci/cd

7.1.3 Compilation

L'objectif étant de pouvoir automatiser les tests dans une pipeline Gitlab, il fallait obligatoirement que les tests puissent compiler sur un docker et donc sur un Linux. L'outil le plus adapté pour ce genre de tache est **Makefile** qui a été utilisé en début de projet. Le défaut de Makefile est qu'il reste assez proche du script et qu'il faut obligatoirement détailler toutes les commandes. Étant donné le fait qu'il y avait beaucoup de fichiers, le Makefile est vite devenu très compliqué et illisible et ce même si aucune compilation séparé n'a été mise en place au début du projet. Quand de plus en plus de fichiers ont été ajoutés au projet de test, il a fallut mettre en place de la compilation séparée pour ne pas tout recompiler à chaque fois. Faire cela avec Makefile est tout à fait possible mais cela aurait pris beaucoup de temps et le fichier final aurait été assez peu lisible et surtout assez compliqué à comprendre. Le but étant de faire un outil facilement compréhensible et utilisable par tous les membres de l'entreprise, il fallait trouver une solution plus simple. La syntaxe de **CMake** à permis de simplement lister les fichiers à compiler ainsi que les répertoires à inclure sans avoir besoin de détailler les option de gcc. Cet outil permet de générer un Makefile complet avec un syntaxe beaucoup plus simple. De plus, CMake permet nativement de faire de la compilation séparer ce qui à permis de gagner du temps pendant le développement étant donner le fait que seuls les fichiers modifiés sont recompilés lors de l'écriture des tests.

7.1.4 nvim

I use vim btw

7.2 Planification des tâches

On voit ça sur le gantt.
Ce diagramme est disponible en index A pour plus de lisibilité.
Mais en fait on a fait ça parceque
On peut retrouver ce diagramme en index B.
conclusion

Troisième partie

Résultats et discussions

1 L'outil final

Description de l'outil final réalisé à la fin du stage.

2 Discussion et perspectives

Regard sur ce qui a été fait et comment cela a été fait (parler de l'échec des threads, manque de bibliographie, ...). Parler de ce qui reste à faire ou a améliorer.

Conclusion

CONCLUSION

Test affichage biblio :

— [2] : présentation du protocole I^2C
— [3] et [4] : présentation du protocole SPI
— [5] : simulation pour CI sur systèmes embarqués
— [6] : intégration continue sur les systèmes embarqués.
— [7] : livre sur les frameworks de tests (p 41 : mocking)

Biliographie

[2] J. Mankar, C. Darode, K. Trivedi, M. Kanoje et P. Shahare, "Review of I2C protocol," *International Journal of Research in Advent Technology*, t. 2, n° 1, 2014. adresse : https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=314537daa1f601f83044b25b68e2af6c8f331f3f.

[3] P. Dhaker, "Introduction to SPI interface," *Analog Dialogue*, t. 52, n° 3, p. 49-53, 2018. adresse : https://b2.sisoog.com/file/zmedia/dex/cd38acc402d52de93a241ca2fcb833b5_introduction-to-spi-interface.pdf.

[4] L. L. Li, J. Y. He, Y. P. Zhao et J. H. Yang, "Design of microcontroller standard SPI interface," in *Applied Mechanics and Materials*, Trans Tech Publ, t. 618, 2014, p. 563-568. adresse : https://www.researchgate.net/profile/Jianhong-Yang-2/publication/286761932_Design_of_Microcontroller_Standard_SPI_Interface/links/58bfe7eba6fdcc63d6d1bb37/Design-of-Microcontroller-Standard-SPI-Interface.pdf.

[5] J. Engblom, "Continuous integration for embedded systems using simulation," in *Embedded World 2015 Congress*, 2015, p. 18. adresse : http://www.engbloms.se/publications/engblom-ci-ew2015.pdf.

[6] T. Mårtensson, D. Ståhl et J. Bosch, "Continuous integration applied to software-intensive embedded systems–problems and experiences," in *Product-Focused Software Process Improvement : 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17*, Springer, 2016, p. 448-457. adresse : https://www.researchgate.net/profile/Torvald-Martensson/publication/309706302_Continuous_Integration_Applied_to_Software-Intensive_Embedded_Systems_-_Problems_and_Experiences/links/5beb0d4e4585150b2bb4d803/Continuous-Integration-Applied-to-Software-Intensive-Embedded-Systems-Problems-and-Experiences.pdf.

[7] P. Hamill, *Unit test frameworks : tools for high-quality software development.* " O'Reilly Media, Inc.", 2004. adresse : https://books.google.fr/books?hl=fr&lr=&id=2ksvdhhnWQsC&oi=fnd&pg=PT7&dq=c+unit+test+framework&ots=AM5ZXbSUG4&sig=jPl1QzC0OSPfnBQi6IKXWLPeNzo&redir_esc=y#v=onepage&q=c%20unit%20test%20framework&f=false.

Webographie

[1] Wikipedia contributors, *List of unit testing frameworks — Wikipedia, The Free Encyclopedia*, [Online ; accessed 29-May-2023], 2023. adresse : https://en.wikipedia.org/w/index.php?title=List_of_unit_testing_frameworks&oldid=1143492860 (visité le 03/03/2023).

A Diagramme de Gantt prévisionnel

B Diagramme de Gantt réel