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 Présentation de CKsquare

CKsquare est une entreprise d'ingénierie spécialisée dans la conception de systèmes de payement automatisés.

- monétique ++ - station auto lavage - faite ses 20 ans cette année - pousse les carte électroniques très loin -> solution très fiable / économique et écologique - ils ont mis en place des systèmes perso pour pousser les cartes encore plus loin - regroupement d'entreprises pour concevoir les bornes de A à Z - fournissent des solution personnalisées pour les clients

2 Travail demandé

L'objectif du stage est de réaliser un outil permettant de pouvoir tester la partie commune des bibliothèques qui s'exécutent sur les cartes électroniques des bornes de la société. À terme, l'outil doit permettre de faire de l'intégration continue, les tests doivent donc pouvoir être automatisés dans une pipeline Gitlab.

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 exécuter 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.

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 à créer, 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.

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. 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 pouvoir tester certaines parties de code 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 parfaitement possible avec tous les frameworks C étant donné la compatibilité entre le C et le C++. Pour pouvoir tester du code C++, il fallait 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. Ce critère à aussi été considéré pour la recherche du framework et des frameworks C++ on aussi été retenus..

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.

TODO : - statut du développement (star github, collaborateurs, dernière maj)

2 Organisation du projet

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

3 Simulation du stockage

3.1 Interface I²C

3.2 Interface SPI

4 Simulation des composants

4.1 Interface Cctalk

4.2 Interface MDB

5 Déroulement du projet

5.1 Les outils utilisés

5.1.1 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.

5.1.2 Gitlab

ci/cd

5.1.3 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.

5.1.4 nvim

I use vim btw

5.2 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 TODO

todo

2 Discussion et perspectives

On a fait ça et c'est bien mais on pourrait faire ça ou changer ça.

Conclusion

CONCLUSION

— [1] : présentation du protocole I^2C
— [2] et [3] : présentation du protocole SPI
— [4] : simulation pour CI sur systèmes embarqués
— [5] : intégration continue sur les systèmes embarqués.

Biliographie

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

A Diagramme de Gantt prévisionnel

B Diagramme de Gantt réel