Rapport ISE-OC : Serre Connectée

Qualité logicielle – bus de communication

Corentin Bouynot – Tristan sannier

2021

# Table des matières

[Table des matières 1](#_Toc67470281)

[1. Cahier des charges 2](#_Toc67470282)

[2. Etude préalable 2](#_Toc67470283)

[2.1. Schéma 2](#_Toc67470284)

[2.2. Découpage fonctionnel 2](#_Toc67470285)

[3. Tests unitaires 3](#_Toc67470286)

[4. Rapport de tests 4](#_Toc67470287)

# Cahier des charges

La serre connectée doit recevoir des informations sur un microcontrôleur MSP430G2553 par l’intermédiaire d’un module bluetooth RN-42 interfacé en UART. Ces informations doivent être interprétées et transmises à un deuxième microcontrôleur MSP430G2553 via un bus SPI. Ce deuxième microcontrôleur doit, selon la commande reçue

* Lire la valeur d’un capteur UV sur un de ses ports analogiques
* Lire la valeur d’humidité et de température d’un capteur via un interfaçage I2C
* Commander un servomoteur, variant entre 3 positions

Le programme doit être réalisé en C, et devra respecter certaines normes de codage, dérivées du MISRA 2012.

# Etude préalable

À la suite d’un problème au niveau des microcontrôleurs reçus lors de la distribution du matériel, nous avons dû faire le projet avec deux MSP430G2553.

## Schéma

Ici, tout est alimenté en 3.3V via l’USB du PC. Toutes les masses sont communes.

## Découpage fonctionnel

Master :

* UART : Fichier gérant la configuration de l’UART et son fonctionnement
  + **void init\_UART( void )** : initialisation de l’UART
  + **void send\_UART(unsigned char \*msg)** : envoie une chaîne de caractères via l’UART
* Interpreteur : Interpreteur de commande : transfert via SPI ou répond via UART selon la commande reçue via UART
  + **void interpreteur( void )**: renvoie un message d’acknowledge via UART et transmet la commande au slave via SPI si besoin de transmettre.
* SPI\_Master : Fichier gérant la configuration du SPI et son fonctionnement
  + **void init\_SPI( void )**: Initialisation du SPI en master
  + **void Send\_SPI(unsigned char carac)**: envoie d’un seul caractère au slave via SPI
* Init\_LP : Fichier d’initialisation de la launchpad
  + **void init\_LP( void )** : Initlialisation de la launchpad
* 2553\_MASTER : fichier contenant la fonction main du MASTER
  + **void main( void )**: Comportement du projet

Slave :

* ADC : Bibliothèque gérant l’ADC (fournie par l’ESIGELEC donc non-détaillée ci-dessous)
* UV\_sensor : Fichier gérant l’initialisation et la récupération des données du capteurs UV
  + **void init\_UV( void )** : Paramétrage de l’ADC sur le pin du capteur UV
  + **unsigned int Read\_UV(void)**: Lecture de la valeur de l’ADC et conversion en indice UV pour la valeur retournée
* Init\_LP :
  + **void init\_LP\_Slave( void ) :** Initialisation de la launchpad
* Servomoteur**:** Initialisation et gestion de la PWM pour le servomoteur
  + **void init\_pwm( void )** : Initialisation de la PWM pour le servomoteur
  + **void set\_servo (unsigned char pos)**: Modification de la PWM en fonction de la position demandée
* SPI\_Slave : Fichier gérant la configuration du SPI
  + **void init\_SPI\_Slave ( void )**: Initialisation du SPI en slave
* 2553\_SLAVE : fichier contenant la fonction main du MASTER
  + **void main( void )**: Comportement du projet
* I2C : Fichier gérant la configuration et la communication I2C
  + **Void init\_I2C (void)**: initialisation et configuration de l’I2C

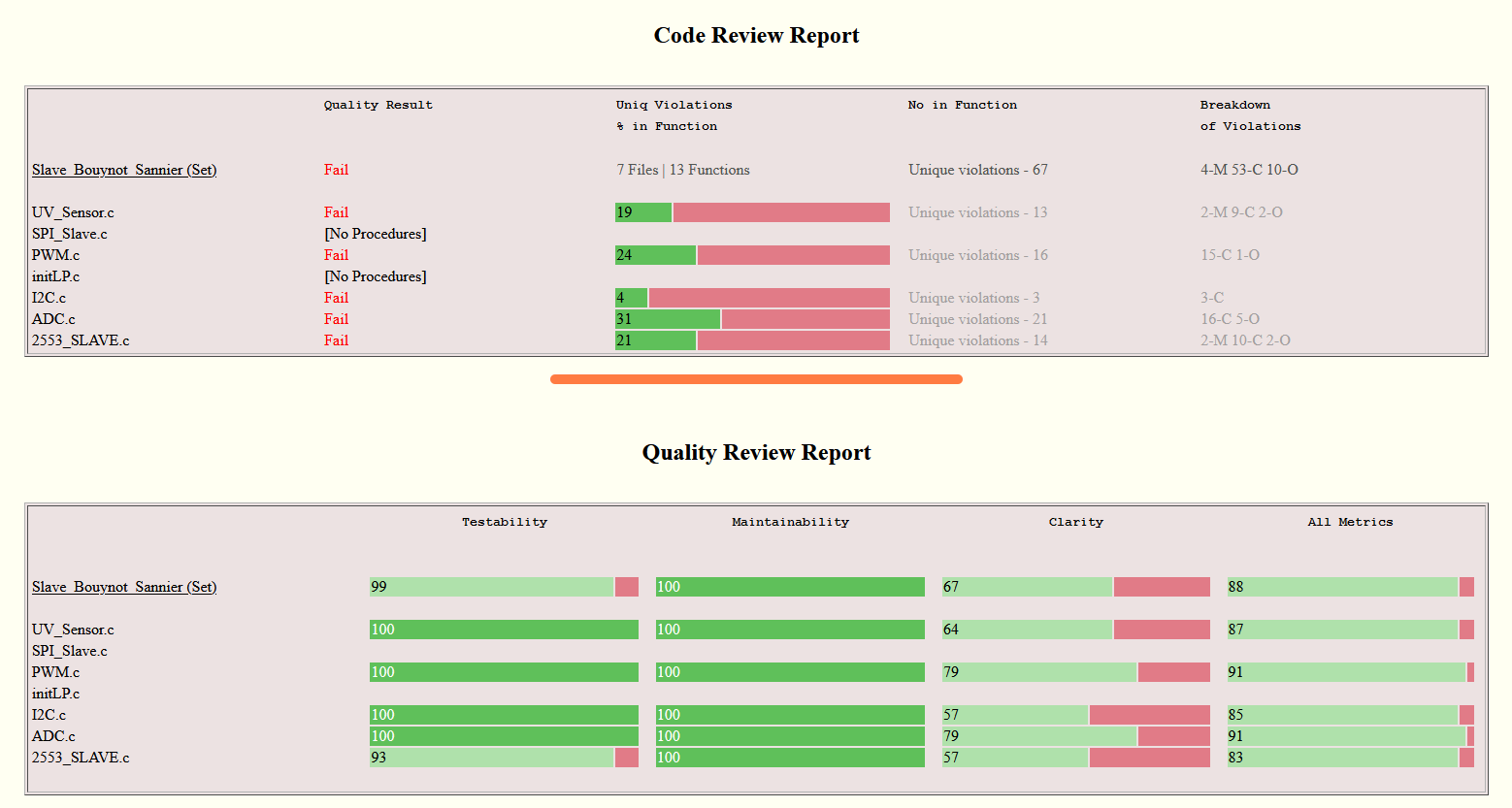
# Tests unitaires

On test le projet avec un terminal bluetooth sur téléphone.

|  |  |  |  |
| --- | --- | --- | --- |
| Objectif | Entrée | Sortie attendu | Sortie observée |
| Mesure UV | u | Entier (ex : 0) | Renvoie de la valeur UV relevée. |
| Arrêt rotation servo | 0 | Arrêt de la rotation du servo en cours | Arrêt de la rotation du servo en cours |
| Position du servo à 0° | 1 | Position du servo à 0° (TA1CCR1 = 312) | Position du servo à 0° (TA1CCR1 = 312) |
| Position du servo à 0° | 2 | Position du servo à 90°  (TA1CCR1 = 187) | Position du servo à 90°  (TA1CCR1 = 187) |
| Position du servo à 0° | 3 | Position du servo à 180°  (TA1CCR1 = 63) | Position du servo à 180°  (TA1CCR1 = 63) |
| Mesure température et humidité | t ou h | Température (°C)  Humidité (%) | Non testée |
|  |  |  |  |

# Rapport de tests

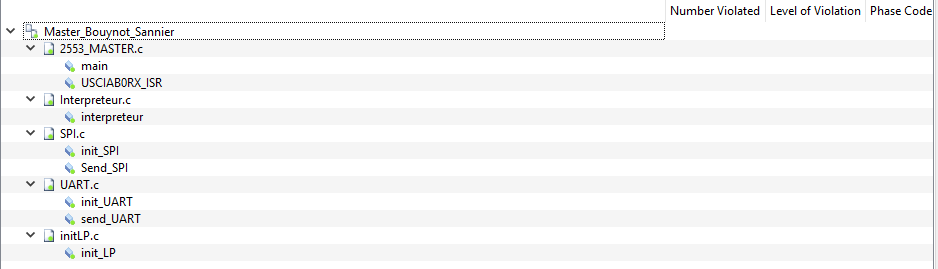
Pour commencer cette partie, nous avons passé nôtre code sous l’Analyse statique de LDRA voici les extraits des rapport obtenus :

* Coté Master :
* Coté Slave :

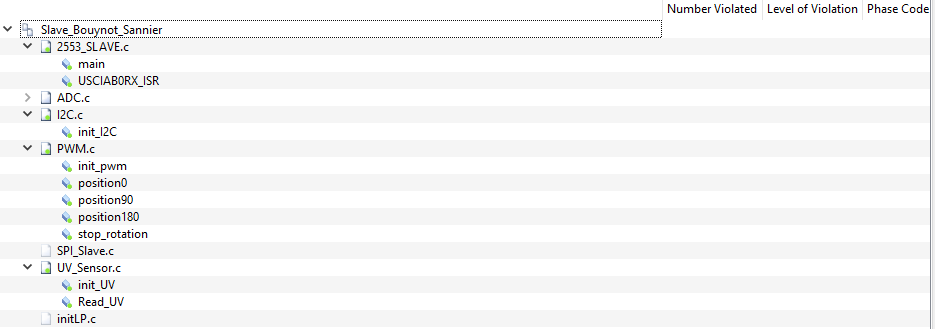
Grâce à ces deux premiers rapports nous avons pu nous faire une idée de l’état général de nôtre code en termes de qualité, de lisibilité et de maintenabilité. De ce fait nous en concluons qu’avant modification nôtre code avait un nombre assez conséquent de normes à appliquer afin de correspondre au standard que nous avons créer (basé sous MISRA 2012). Pour créer ce standard nous nous somme donc basé sur du MISRA 2012 auquel nous nous sommes permis de retirer quelque règle tels que les warnings surs : les opérations bit à bit, les égalités sur des booléens ainsi que la présence de ligne de commentaires avec des « // ». Malgré tout nous pouvons aussi constater que la maintenabilité, la testabilité et la clarté de nos codes obtenaient des valeurs correctes pour un premier passage sous LDRA.

Nous nous sommes par la suite mis à corriger nôtre code en suivant les recommandations de LDRA pour finir par nous retrouver avec une analyse de code tels que visible sur les captures ci-dessous :

* Coté master :



* Coté slave :



Tel qu’il est possible de la voir LDRA ne nous donne plus d’erreurs à corriger. Cependant, malgré cela, le rapport de tests nous indiqué un nombre d’erreurs n’ayant que très sensiblement évolué coté slave et plus grand encore coté master qu’avant les corrections apportées. Nous avons donc pensé à un possible conflit avec d’autres projet LDRA, cependant nous n’avons à ce jour pas trouvés de solution à ce problème.