NOTE D'APPLICATION

Structuration des programmes

Amélioration de l'asservissement de visée laser

Projet 2020-2021

Benoit Vialard

5ème année de Génie Electrique option Systèmes embarqués

POLYTECH CLERMONT-FERRAND

UNIVERSITÉ Clermont Auvergne

Table des matières

Introduction

Pour appréhender plus facilement ce projet d'asservissement de visée laser, il est important d'avoir une idée claire sur la façon dont les programmes C et VHDL interagissent entre eux et comment ils sont structurés. Ce sera le principal objectif de cette note d'application. Les manipulations à effectuer pour implémenter les programmes sur la partie HPS[1] et FPGA de la carte De0 Nano SoC[2] seront également mentionnées.

Les fichiers du projet se situent sur la forge dans la partie dépôt du projet « P20AB21 amélioration de l'asservissement de visée laser » :

https://forge.clermont-universite.fr/svn/p20ab21-amelioration-de-l-asservissement-de-visee-laser

Ils peuvent être récupérés en faisant un « SVN checkout » en indiquant l'adresse ci-dessus. Pour effectuer cela, on pourra par exemple utiliser le logiciel Tortoise SVN qui peut être téléchargé à l'adresse suivante :

https://tortoisesvn.net/downloads.html

Les codes se situent dans le répertoire /trunk/Codes/.

[1] SoC : System on Chip (Un Système d'exploitation Linux embarqué tourne sur la carte)
[2] HPS : Hard Processor System (Le HPS est composé d'un processeur double cœur ARM Cortex A9)

I. Communications entre les parties du système

I.1. Vue d'ensemble

Pour ce projet, nous utilisons la carte De0 nano Soc. Cette carte est composée de deux parties :

- La partie HPS, qui permettra d'exécuter des programmes de régulation ou de tests. Un système d'exploitation Linux tourne en parallèle des programmes en C lors de leur exécution. Elle est composée de deux cœurs de processeur ARM cortex A9, d'architecture 32 bits
- La partie FPGA, qui servira d'interface entre le HPS et les périphériques (leds utilisateurs, galvanomètres).

Vous trouverez dans la figure 1 ci-dessous un schéma représentant les différentes parties du système et leurs interactions.

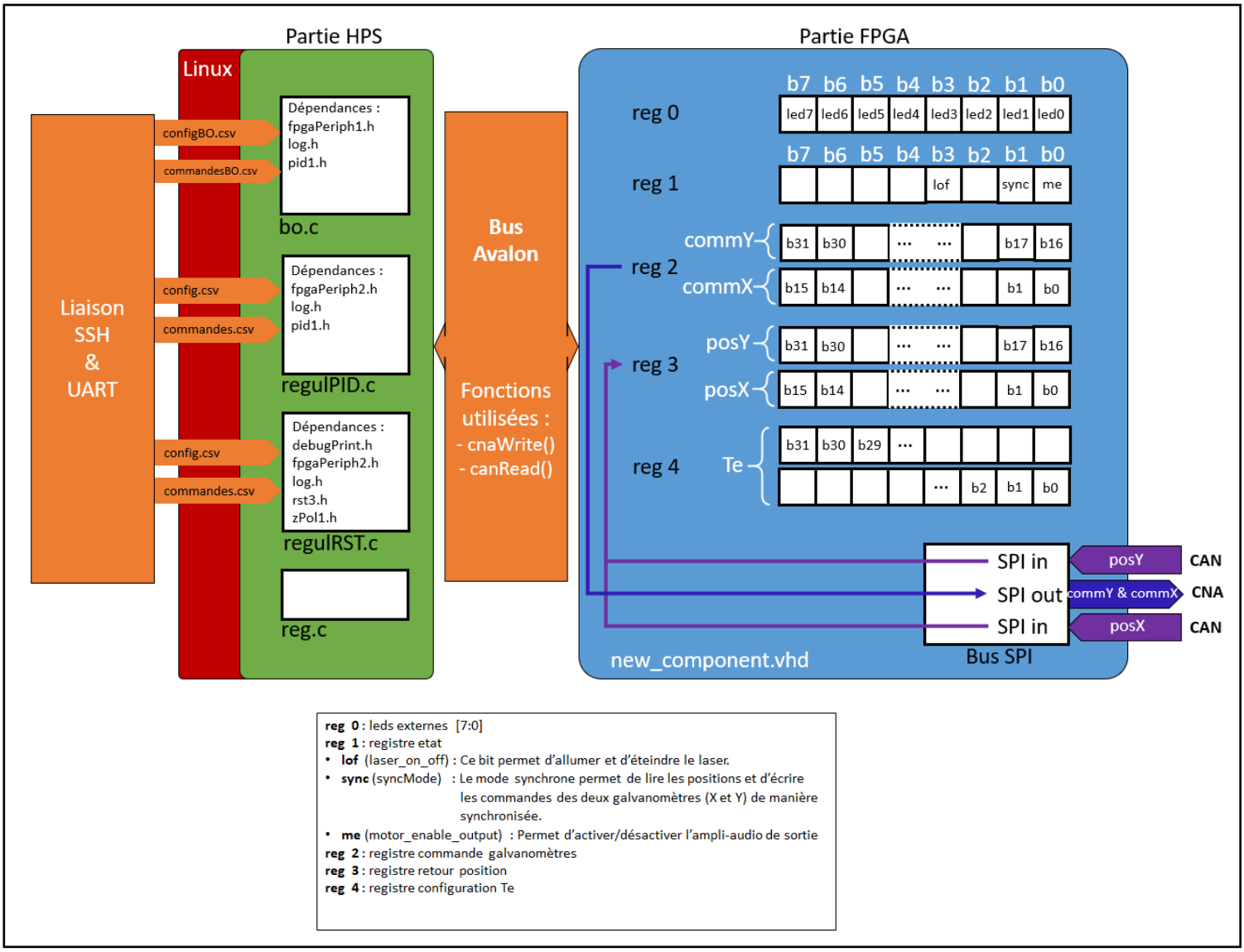

Figure 1 : Représentation globale du système

Tous les éléments présents dans la Figure 1 seront explicités tout au long de cette note d'application.

I.2. Liaison SSH & UART

La liaison SSH va permettre de transférer des fichiers vers la carte SD de la carte De0 nano SoC. Cette liaison sera essentielle pour transférer les programmes en C (régulation RST, test en boucle ouverte, …) vers la cible ainsi que fournir aux programmes les fichiers dont ils ont besoin pour fonctionner (Lecture des consignes/commandes des galvanomètres, lecture des paramètres système). Elle permettra également de récupérer les positions des galvanomètres dans un fichier.csv.

Remarque : Pour que la liaison SSH soit fonctionnelle, il faut que le PC utilisateur et la carte De0 nano Soc soit sur le même sous-réseau. Autrement dit, il faut qu'ils aient le même masque de sous-réseau.

La liaison série, quant à elle, va servir à lancer les programmes depuis le linux embarqué du SoC. Elle permettra également de modifier les fichiers de configuration des paramètres systèmes et/ou des commandes moteurs.

Remarque : Pour la mise en place et l'utilisation des liaisons SSH et UART (logiciels et branchements) entre le PC de l'utilisateur et la carte De0 Nano SoC ainsi que le paramétrage de la liaison série, je vous invite à consulter la note d'application de Mme. Camille SOTTY sur le sujet. (Voir Documents sur la forge du projet « P19AB01 amélioration de l'asservissement de visée laser » en suivant le lien ci-après : https://forge.clermont-universite.fr/projects/p19ab01/documents).

I.3. Fonctionnement du composant de la partie FPGA

I.2.1. Interface HPS/FPGA : Le Bus Avalon

La communication entre la partie logique et HPS de notre SoC est rendue possible grâce à une liaison AVALON. Plus précisément, il permet la communication entre les programmes qui s'exécutent sur le HPS et le programme new_component.vhd situé sur le FPGA. Ce programme définit la façon dont les signaux du bus Avalon va interagir avec les différents registres. Celui-ci est contenu dans ce que l'on appelle un module IP. Ce module IP sera relié aux autres modules du système grâce à l'outil Platform Designer du logiciel Quartus. C'est notamment grâce à cet outil que le bus Avalon est matériellement relié aux différents blocs du projet et que les signaux utilisés dans le programme new_component.vhd sont exportés vers les broches d'entrées/sorties de la carte De0 nano SoC.

Le bus Avalon fonctionne en mode Maitre/Esclave. Dans notre cas, le FPGA est l'esclave et le HPS est le maitre. Il permet au HPS de lire et d'écrire dans les registres du FPGA pour commander les périphériques et configurer la période d'échantillonnage de Convertisseur Analogique Numérique (Voir correspondance des registres en figure 1).

Les échanges sont effectués avec des registres de 32 bits quel que soit le registre interne au FPGA que l'on souhaite avoir accès.

Remarque : La version de Quartus qui a été utilisée pour ce projet est la version Lite 18.1. Ce logiciel permet de mettre en place le projet VHDL et de programmer le FPGA. Vous pouvez récupérer le fichier d'installation en suivant le lien ci-dessous :

https://fpgasoftware.intel.com/18.1/?edition=lite&platform=windows

Lors de l'installation, il ne faudra pas oublier d'installer le composant Cyclone V (celui de la carte De0 Nano SoC).

Les étapes nécessaires pour mettre en place la partie FPGA sont présentes dans fichier README.txt que l'on peut retrouver dans le répertoire du projet VHDL. (Suivre le chemin suivant : `/trunk/Codes/hw/DE0_NANO_SOC_JTL`). Les manipulations à effectuer sont également expliquées

en détail dans la note d'application sur la mise en place de la chaîne de développement réalisée par Mme. Camille Sotty l'année précédente (Voir onglets documents sur la forge du projet en utilisant le lien suivant : https://forge.clermont-universite.fr/projects/p19ab01/documents).

I.2.2. Communication Périphériques/FPGA : Le Bus SPI

Dans le programme new_component.vhd, une liaison SPI entre le CNA[3] et les CAN[4] est définie. Une attention toute particulière est accordée à la gestion des timings. En effet, la valeur de la période d'échantillonnage contenue dans le registre 4 (Voir figure 1) va directement définir la fréquence d'envoi/de réception des trames SPI. Ce bus SPI a deux modes : un mode par défaut et un mode synchrone.

- Le mode par défaut ne permet d'envoyer des commandes qu'au galvanomètre branché sur le port J7 (Voir schéma de la carte extension dans l'onglet Documents du projet sur la forge)

- Le mode synchrone, quant à lui, permet de charger les commandes des deux galvanomètres dans les buffers du convertisseur numérique-analogique avant de les envoyer simultanément.

Les commandes qui sont envoyées dépendent des valeurs contenues dans le registre 2 (Voir figure 1) et chacunes d'elles sont codées sur 16 bits.

Les positions sont codées sur 14 bits et sont stockées dans le registre 3 après réception.

II. Structure des programmes en C sur la cible

II.1. Programme de test de communication avec le FPGA (reg)

L'objectif de ce programme est de vérifier que l'écriture et la lecture des registres sont fonctionnelles.

Pour pouvoir lire et écrire dans les registres, le programme reg.c utilise la fonction mmap() (contenue dans le fichier fpgaPeriph1.c) qui permet de faire le lien avec le FPGA et d'avoir accès aux périphériques depuis l'espace utilisateur. Grâce à ce programme, on peut écrire dans n'importe quel registre que l'on peut voir dans la partie FPGA de la figure 1. Il est préférable d'effectuer les premiers tests en écrivant simplement dans le registre 0, qui est un registre de 8bits permettant d'allumer les leds utilisateurs (présentes sur la carte) en fonction du nombre binaire entré dans celui-ci. Pour effectuer ce simple test, il suffit d'entrer, par exemple, la ligne suivante dans un terminal série :

```
./reg 0 0xAA
```

Le premier paramètre permet de spécifier le registre dans lequel on souhaite écrire (dans cet exemple c'est le registre 0) et le second permet de déterminer la valeur que l'on souhaite mettre dans le registre (Ici 0xAA).

Il est également possible de vérifier le bon fonctionnement des communications avec les galvanomètres.

[3] CNA : Convertisseur Numérique Analogique
[4] CAN : Convertisseur Analogique Numérique

Pour cela, il faut commencer par activer l'ampli-audio et le mode synchrone en entrant la commande :

```
./reg 1 3
```

Une fois cette commande effectuée, il est alors possible d'envoyer des commandes directement vers les galvanomètres en saisissant par exemple les commandes suivantes :

```
./reg 2 0x70007000
        Commande  Commande
        moteur Y  moteur X
./reg 2 0X00000000
```

Rappel : La commande envoyée sur chacun des galvanomètres (X et Y) est codée sur 16bits. Elles sont ensuite concaténées dans un registre 32bits pour qu'elles puissent être envoyées grâce au bus Avalon vers le FPGA.

II.2. Programme de test en boucle ouverte (bo)

Ce programme permet d'effectuer des tests en boucle ouverte sur un des galvanomètres (port J7 de la carte) et de récupérer les positions dans un fichier « positions.csv ».

Dans un premier temps, grâce à la fonction readConfig(), ce dernier va récupérer les paramètres système dont il a besoin directement dans un fichier qui s'appelle « configBO.csv » (Période d'échantillonnage, nombre d'itérations par consigne, nombre de valeurs sur lequel on souhaite faire du moyennage, …). Ce fichier doit être présent sur la mémoire flash de la carte De0 Nano SoC.

Dans un second temps, en utilisant la fonction readCommandes(), il lira les commandes que l'on souhaite utiliser pour le test en boucle ouverte depuis un fichier nommé commandesBO.csv (situé également sur la mémoire flash). On notera que dans le fichier commandesBO.csv une ligne correspond à une commande.

Après la lecture des fichiers, le programme va commencer par stabiliser la position du miroir au centre de la zone visible grâce à une régulation PID. Ce sera le « zéro » de nos courbes indicielles.

Rappel : Sur l'angle de 40° physiquement réalisable par le galvanomètre, seulement 10.7° sont visibles par les convertisseurs analogique numérique.

Les fonctions qui permettent de déclarer, initialiser et mettre à jour le correcteur PID sont décrites dans le fichier pid1.c dont dépend le programme de test bo.c.

De plus, la communication avec le FPGA est possible grâce à la librairie fpgaPeriph1. Cette librairie contient notamment la fonction InitPeriph() qui permet d'effectuer la projection en mémoire des périphériques dont le FPGA a accès de manière à pouvoir les utiliser depuis le HPS de la carte. L'envoie des commandes moteur sont effectuées par la fonction cnaWrite(). Celle-ci écrit dans le registre 2 du FPGA (Voir figure 1) via le bus Avalon. De la même façon, la fonction canRead() va permettre de lire le registre 3 issu du FPGA qui contient les positions du galvanomètre.

Une fois que le « zéro » de notre courbe est stabilisé, les commandes qui ont été lues précédemment dans un fichier, vont être envoyées vers les galvanomètres les unes après les autres (par l'intermédiaire du FPGA).

Les positions ne sont stockées dans un fichier (grâce à la librairie log.h) que lorsque l'utilisateur indique au programme qu'il souhaite arrêter son exécution (en appuyant sur CTRL+C). On notera que

les positions ne sont récupérées que lorsque le début d'une boucle de régulation est détecté (c'est à dire la première itération de la première consigne à traiter) et l'acquisition ne se termine que lorsque l'on a passé toutes les consignes un certain nombre de fois (valeur de moyennage précisée dans le fichier config.csv).

Remarque : Lors de la stabilisation de la position au centre de la zone visible, les positions ne sont pas conservées.

Vous retrouverez un schéma récapitulatif de la structure du programme bo.c ci-dessous :

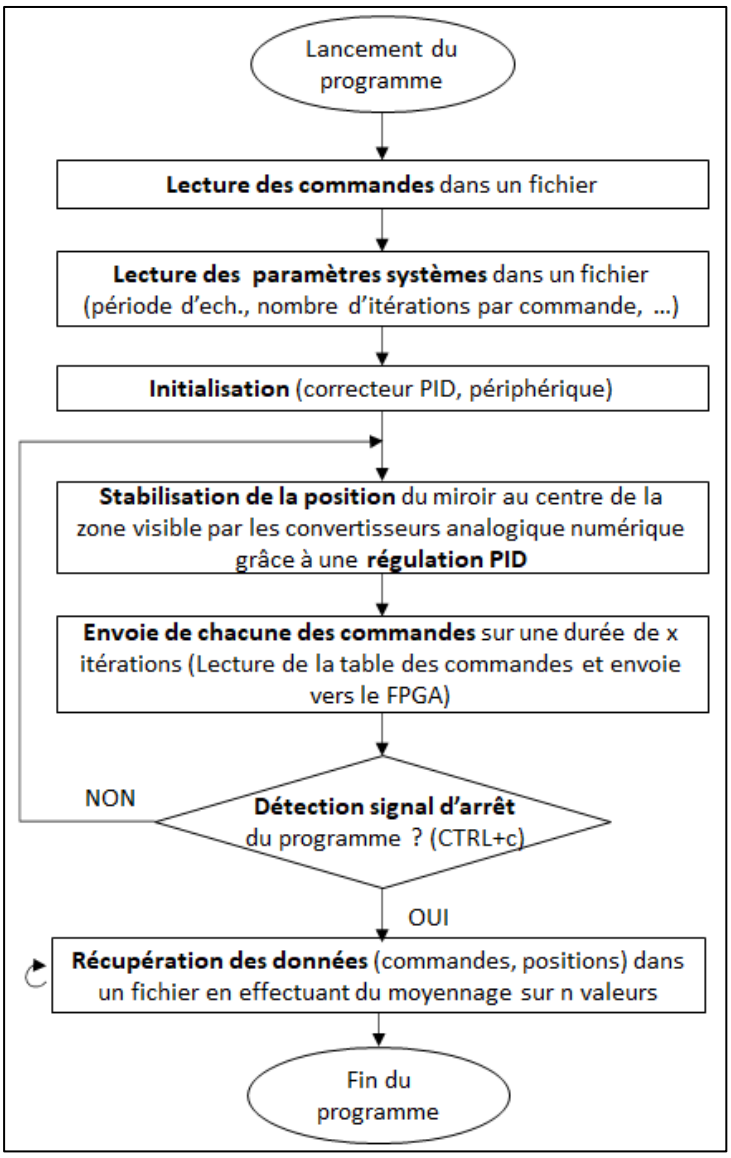

Figure 2 : Schéma structurel du programme bo.c

II.3. Programme de régulation PID (regulPID)

Ce programme regulPID.c permet d'effectuer une régulation PID des deux galvanomètres (grâce à deux correcteurs PID). Il utilise des fonctions similaires pour récupérer les consignes et les paramètres systèmes de la même manière qu'avec le programme bo.c, à la différence que le programme va lire 2 consignes (une pour la voie X et une pour la voie Y) par ligne dans le fichier.csv présent sur la carte SD de la carte De0 nano SoC (Voir figure 1 dans la partie I.1).

Le programme est structuré de la manière suivante :

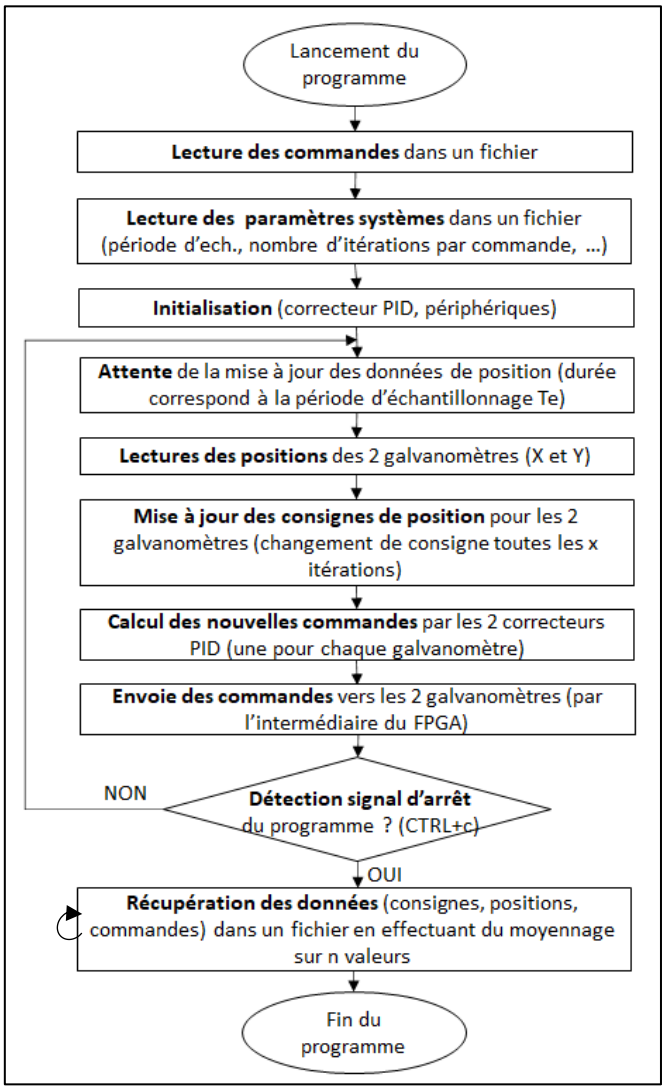

Figure 3 : Schéma structurel du programme regulPID.c

Ce programme va être dépendant de plusieurs librairies :

- fpgaPeriph2.h : Permet d'accéder aux registres du FPGA (en lecture et en écriture) afin de commander les périphériques ou de configurer la période d'échantillonnage.
- pid1.h : Permet de déclarer, d'initialiser et de mettre à jour un correcteur PID

- log.h : Contient les fonctions nécessaires à la récupération de données dans des fichiers (avec ou sans moyennage).

Remarque : le procédé de récupération des données est commun à tous les programmes.

II.4. Programme de régulation RST (regulRST)

La structure du programme regulRST.c permet d'effectuer la régulation RST des deux galvanomètres simultanément en utilisant un correcteur RST par galvanomètre. La structure de celui-ci est très similaire à celle du programme regulPID.c à quelques exceptions près :

- Le correcteur RST a besoin des coefficients R, S et T afin de pouvoir fonctionner. Ces coefficients, qui dépendent de la fonction de transfert des galvanomètres, sont tout d'abord calculés grâce au logiciel Scilab à partir d'un modèle générique utilisant les paramètres constructeurs des galvanomètres dont on dispose. Ensuite les fichiers contenant les coefficients du correcteur RST (CorrecteurR, CorrecteurRD, CorrecteurS, CorrecteurSN, CorrecteurT, CorrecteurTD) sont envoyés vers la carte SD de la cible via liaison SSH. Pour initialiser les correcteurs RST, le programme regulRST.c va utiliser la fonction `rst3InitFromFile()` issu de la librairie rst3.h afin de lire les coefficients contenu dans les fichiers précédemment transférés.

- Les commandes envoyées aux galvanomètres sont le résultat de calculs polynomiaux. Les opérations polynomiales sont assurées principalement par les fonctions zPol1Step() et zPol1InvStep() issues de la librairie zPol1.h.
  La fonction zPol1Step() permet d'effectuer des opérations polynomiales du type :
  $$valeur \times P(z^{-1})$$
  La fonction zPol1InvStep() permet d'effectuer l'opération suivante :
  $$valeur \times \frac{1}{P(z^{-1})}$$
  Ces deux fonctions sont ensuite utilisées dans la fonction rst3Step() pour calculer à chaque itération de boucle la nouvelle commande en fonction des polynômes R(z^{-1}), S(z^{-1}) et T(z^{-1}).

Ce programme va être dépendant des librairies suivantes :

- fpgaPeriph2.h : Permet d'accéder aux registres du FPGA (en lecture et en écriture) afin de commander les périphériques ou de configurer la période d'échantillonnage.
- Rst3.h : Permet de déclarer, d'initialiser et de mettre à jour un correcteur RST (utilise zPol1.h)
- zPol1.h : contient les fonctions permettant de réaliser des calculs polynomiaux
- log.h : Contient les fonctions nécessaires à la récupération de données dans des fichiers (avec ou sans moyennage).

La structure du programme regulRST.c peut être représentée ainsi :

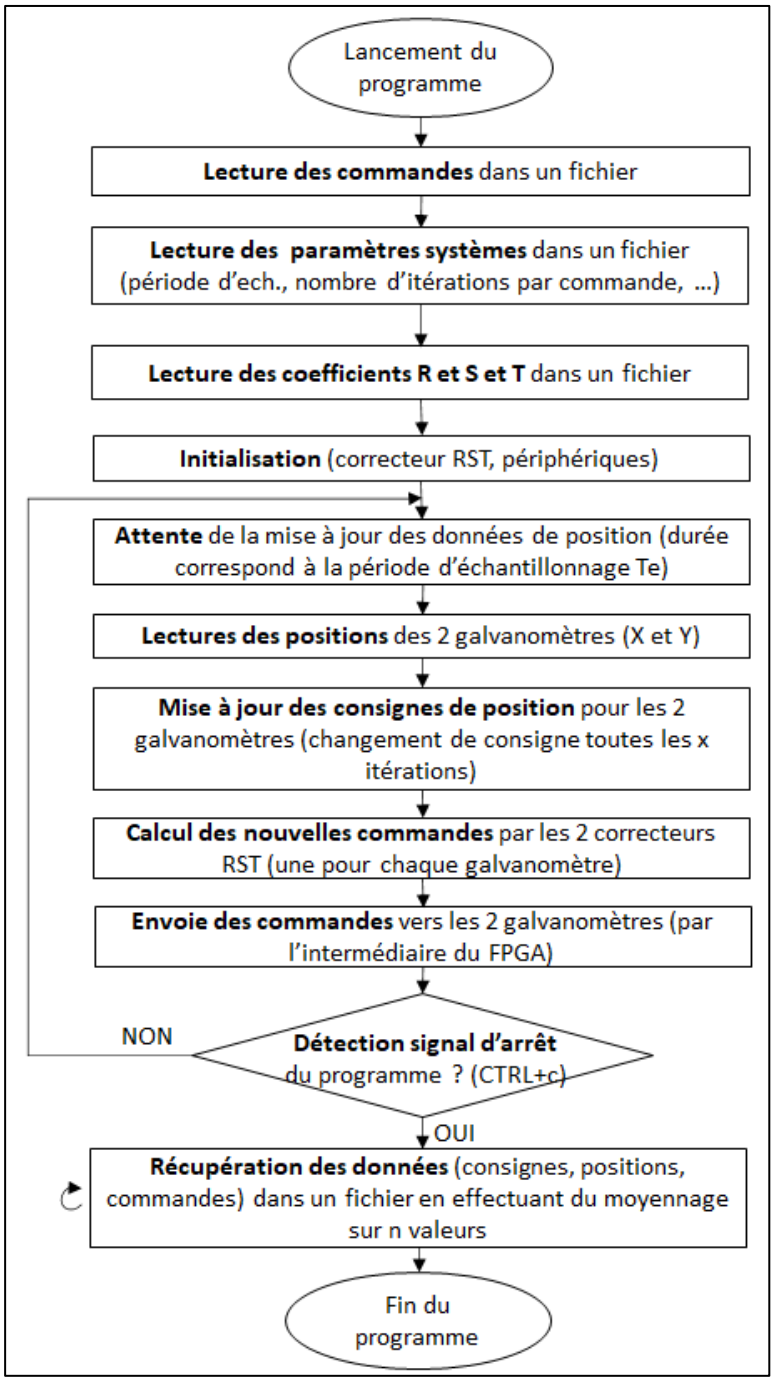

Figure 4 : Schéma structurel du programme regulRST.c

Figure 5 : Schéma structurel du programme regulRST.c