QUAGLIARA Rémy 2020/2021

Projet Asservissement Visée Laser

Note d'application

Table des matières

Table des figures

Introduction

L'objectif de cette note d'application est de donner les éléments nécessaires à la mise en place du fonctionnement des programmes de tests et de commande système par cross-compilation avec l'IDE Code Blocks. Cette chaine de commande peut aussi bien être utilisée de manière locale que par le net.

La note présente dans un premier temps les outils pour mettre en place la communication entre le PC et la carte client de manière locale afin de configurer et vérifier le fonctionnement de la liaison série par UART puis celui de la communication par SSH. Ensuite est présenté la configuration de l'IDE Code Blocks de manière à instancier les commandes de cross-compilation pour faciliter l'utilisation des programmes et de la chaine de commande.

D'autre part cette note d'application a pour but de présenter et de décrire le format des fichiers nécessaires au fonctionnement des programmes de tests et de commande du système.

Enfin en dernière partie est présenté la structure du programme de simulation de la régulation RST implémenté en langage C et notamment l'utilisation de la fonction permettant de mesurer le temps d'exécution au niveau processeur.

Dans le cadre d'une première utilisation des programmes et de la carte client, avant toute manipulation de protocole SSH que ce soit avec ou sans Code Blocks, il est nécessaire de configurer le réseau PC-Carte à l'aide de la liaison série.

1 Communication par liaison série (UART)

1.1 Mise en fonctionnement des outils de communication série

La mise en place de la liaison série est très bien décrite dans le guide constructeur *DE10-Standard_Getting_Started_Guide.pdf* au chapitre 5.3 intitulé « Setting Up UART Terminal ».

Pour résumer, voici dans les grandes lignes les éléments importants pour réaliser cette étape :

- Connecter la carte au PC à l'aide du câble mini USB-B :

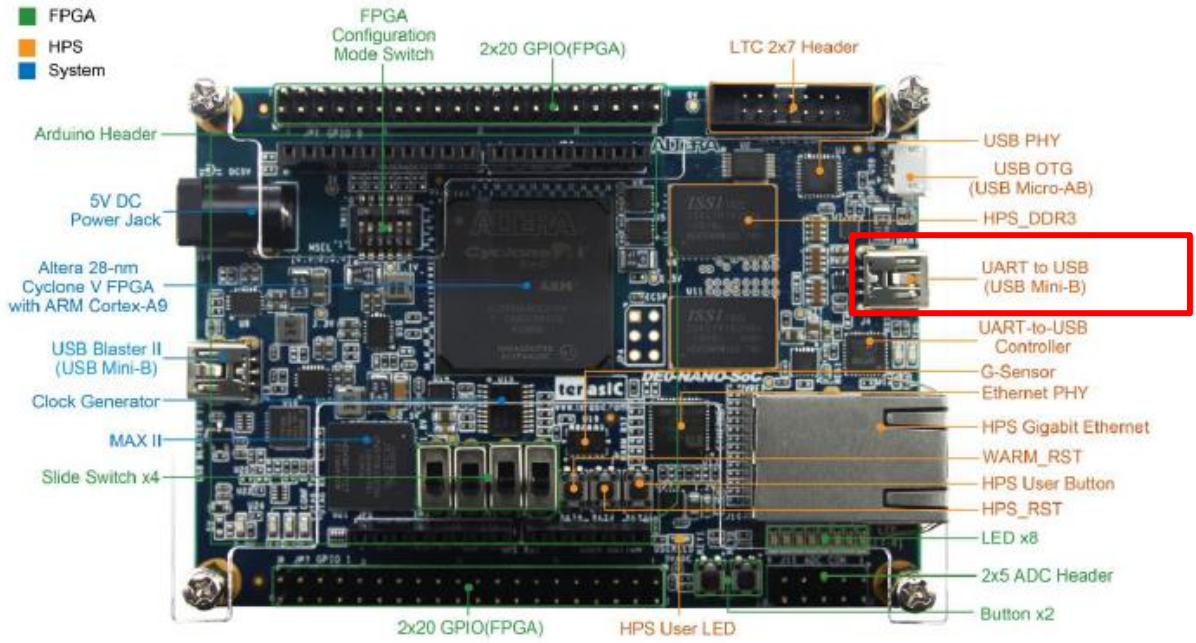

Figure 1 : Localisation port série carte DE0

- Lancer un terminal de commande sur le port correspondant, sous système Windows avec Putty ou TeraTerm, sous système Linux avec Putty ou Minicom. Veiller à paramétrer correctement le port, le baud rate et le type de communication (série).
- Accès au système linux en tant que super-utilisateur (root).

1.2 Mise en place d'un réseau local

Dans un premier temps, il faut configurer les paramètres réseau de la carte client de sorte que le PC et la carte client soient sur le même réseau local, soit une adresse ip propre à chaque device, un masque de sous réseau et un chemin de communication commun. Les étapes se présentent comme tel :

- Brancher le câble Ethernet entre le PC et la carte client.
- Dans l'invite de commande Windows, afficher les paramètres réseau (commande *ipconfig*) et noter l'adresse ipv4, le masque de sous réseau ainsi que la passerelle par défaut.
- Dans le terminal de commande en tant que super-utilisateur afficher les paramètres réseau de la carte client (commande *ip a*), le port Ethernet est par défaut le port eth0.
- Si la carte réseau n'est pas activée alors entrer la commande *ifup eth0*
- Si l'adresse ip n'a pas été affectée automatiquement alors effectuez la configuration manuellement :
    - *ip eth0 X.X.X.X/YY* (X étant les valeurs de l'adresse ip de la carte et YY le nombre de bits activés pour le masque de sous réseau, par exemple si le masque est 255.255.255.0 alors YY = 24)
- Si la passerelle n'a pas été affectée correctement (commande *ip route*) entrez la manuellement : *ip route eth0 X.X.X.X*
- Vérifier l'envoie et la réception de paquets de données dans les deux sens (PC-Carte / Carte-PC) avec la commande ping X.X.X.X (adresse d'envoie)

Si le réseau a bien été configuré mais que l'envoie et la réception de paquets ne fonctionne pas le problème peu provenir du pare-feu windows qui bloque par défaut les flux d'entrée/sortie inconnus, pour palier à ce problème configurez alors une règle d'exception pour le device de la carte client.

Pour réaliser la cross-compilation des programmes en C et transférer des fichiers vers la cible (partie HPS du Cyclone V) de manière sécurisée il est nécessaire d'utiliser le protocole de communication par SSH.

2 Communication par protocole SSH

2.1 Mise en œuvre des outils OpenSSH sous système windows

Pour mettre en service le protocole de communication par SSH (Secure Shell), nous avons utilisé la version Standard 18.1 de l'outil SoC EDS. Le fichier d'installation de SoC EDS peut être récupéré en suivant le lien suivant :

- https://fpgasoftware.intel.com/soceds/18.1/?edition=standard&platform=windows&download_manager=direct

Ensuite, sous système Windows, il faut installer la bibliothèque de commandes OpenSSH. Pour cela retrouvez la note d'installation dans le répertoire partagé sous le dossier suivant :

- *SVN_2020\trunk\Codes\OpenSSH*

Si l'installation s'est effectuée correctement vous devriez avoir accès aux commandes de OpenSSH à partir du shell de commande Soc EDS.

2.2 Configuration du client et du serveur SSH

Le protocole de communication SSH (Secure SHell) permet d'établir une connexion sécurisée à un système Unix, Linux et Windows.

- ssh : programme client permettant de se connecter au serveur ;
- sshd : serveur (ssh daemon).

L'authentification par clé est basée sur 3 composants :

- une clé publique qui sera exportée sur chaque hôte auquel on souhaite se connecter ;
- une clé privée qui permet de prouver son identité aux serveurs ;
- une passphrase qui permet de sécuriser la clé privée.

Avant d'établir la connexion SSH, les pings PC-Cible et Cible-PC doivent être fonctionnels.

2.2.1 Génération de la paire de clés (PC Soc EDS commande shell)

Commande : *ssh-keygen* | Sans paramètres, génération d'une clé de type RSA en 2048 bits.

Si demandé, entrer le fichier de sauvegarde de la clé :

~.ssh/id_rsa

Vérifier que la clé privée a bien été créée :

cat ~.ssh/id_rsa

Vérifier que la clé publique a bien été créée :

cat ~.ssh/id_rsa.pub

2.2.2 Paramétrage serveur SSHd (Carte)

Dans le fichier de configuration du serveur SSH, vérifier que l'authentification par clé est active par défaut (valeurs commentées par # = valeurs par défaut)

cat /etc/ssh/sshd_config

#PubkeyAuthentication yes

Vérifier l'existance du fichier ~.ssh/authorized_keys

2.2.3 Ajout clé publique au serveur depuis le PC (Soc EDS commande shell)

- Première méthode (recommandée) :

ssh-copy-id -i ~/.ssh/id_rsa.pub root@ip_cible

- Deuxième méthode :

scp ~/.ssh/id_rsa.pub root@ip_cible:/tmp

ssh root@ip_cible

cat /tmp/id_rsa.pub >> /root/.ssh/authorized_keys

rm /tmp/id_rsa.pub

2.2.4 Ouverture session depuis PC en tant que super utilisateur

ssh root@ip_carte

2.2.5 Fermeture session

logout

3 Configuration de l'IDE CodeBlocks (Cross-compilation)

3.1 Compilation avec CodeBlocks

La première étape de la configuration de Codeblocks consiste à configurer le compiler :

Settings -> Compiler

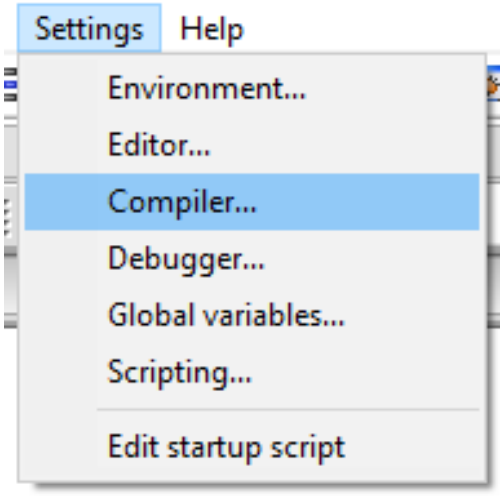

Figure 2 : Onglet settings CodeBlocks

Sélectionner le compiler gcc pour processeur arm :

Global compiler settings -> Select compiler -> GNU GCC Compiler for ARM

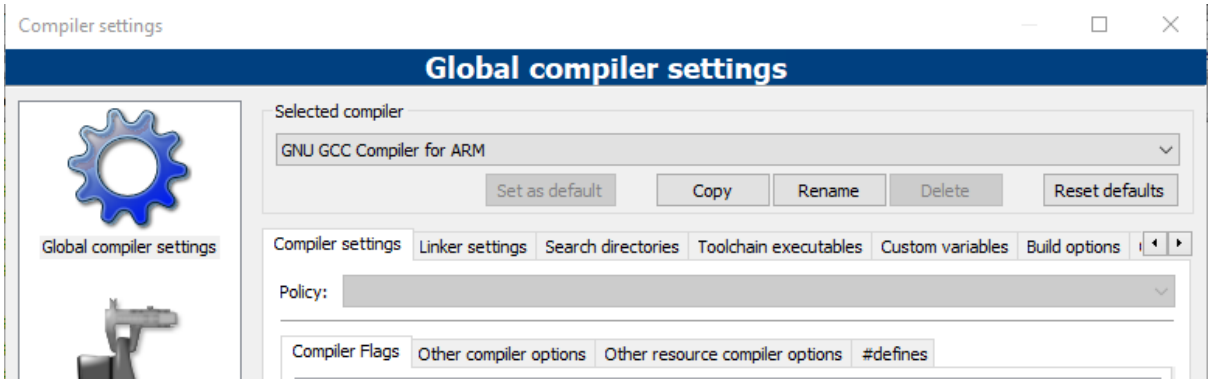

Figure 3 : Compiler settings CodeBlocks

De même pour les programmes de compilation correspondant qui se trouvent dans le dossier suivant si vous avez suivi correctement l'installation de Quartus Prime version 18.1, le chemin est à entrer dans la case

➔ Compiler's installation directory -> ~\intelFPGA\18.1\embedded\ds-5\sw\gcc\bin

Puis entrer les exécutables correspondant comme sur l'image ci-dessous :

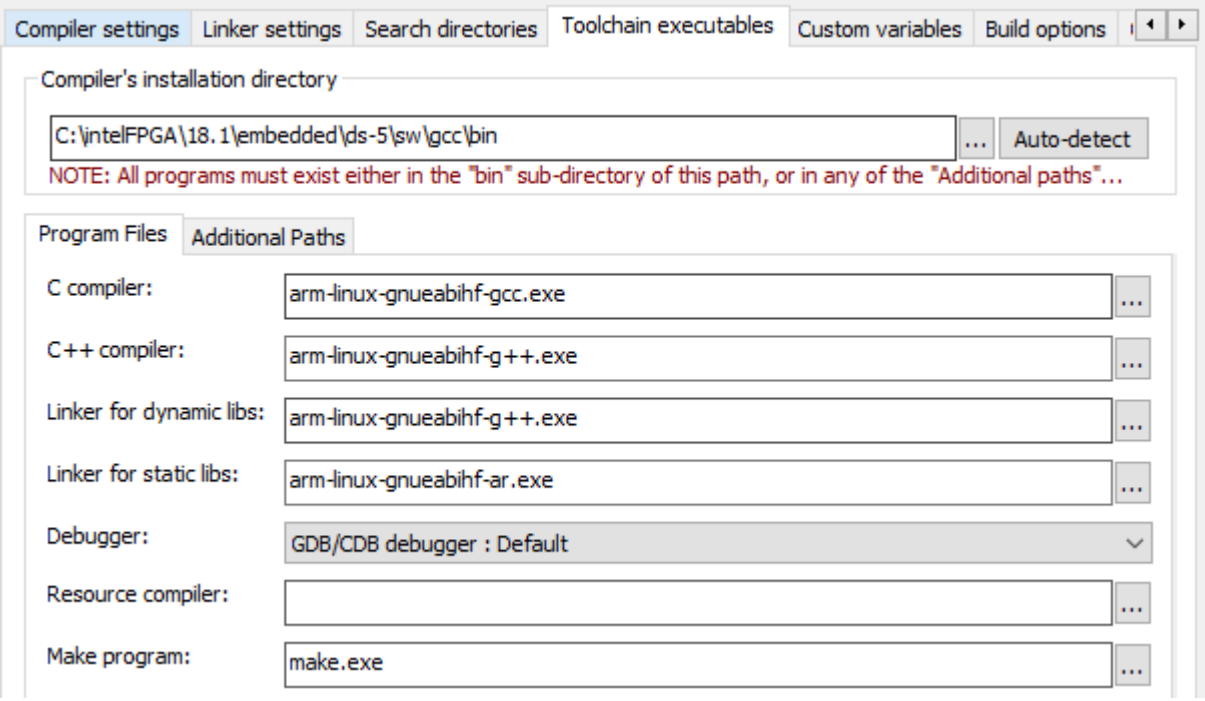

Figure 4 : Toolchain executables CodeBlocks

Deuxième étape, configuration du build option du projet correspondant :

➔ Clic droit entête de projet -> build options

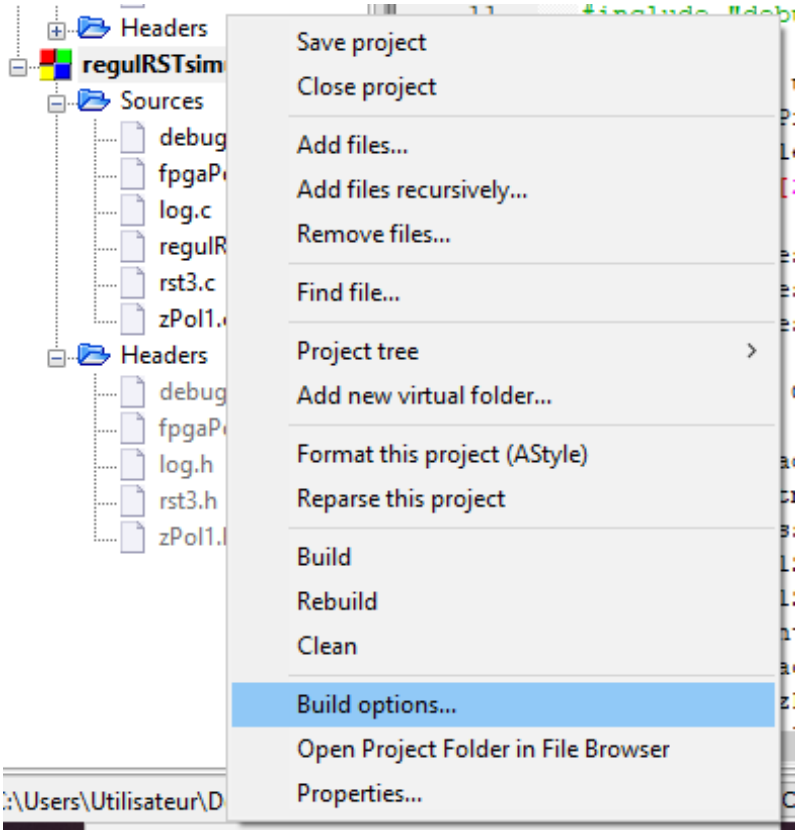

Figure 5 : Onglet Buld Options CodeBlocks

Vérifier que le compiler sélectionné correspond, puis entrer dans l'onglet search directories les chemins vers les bibliothèques suivantes :

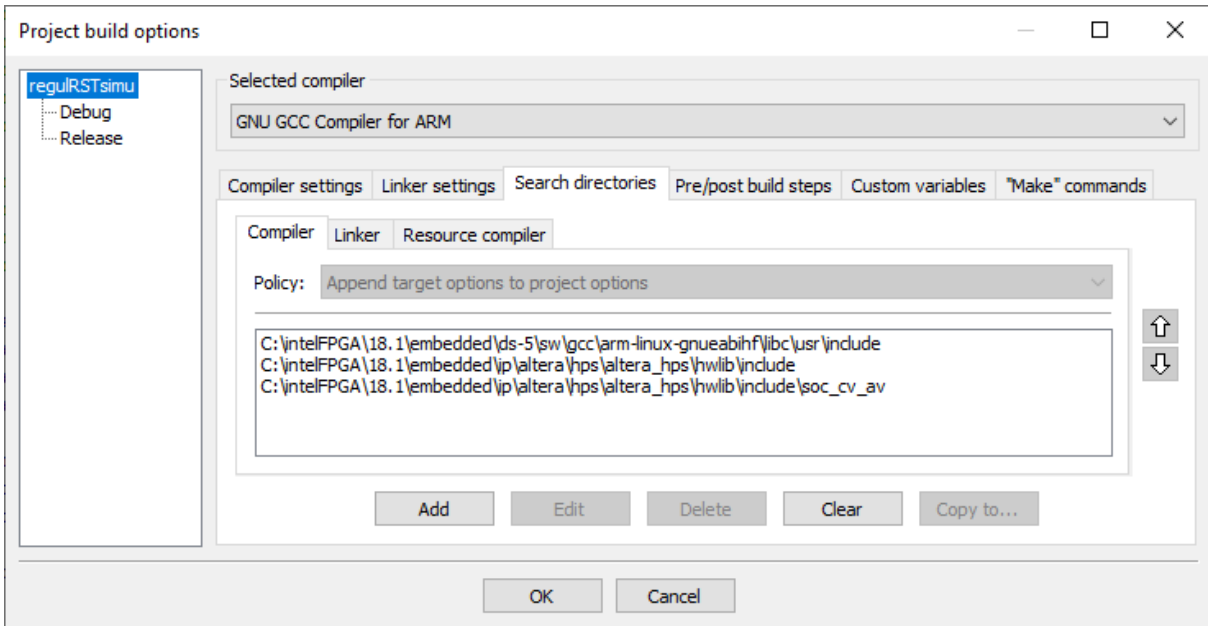

Figure 6 : Search Directories CodeBlocks

~\intelFPGA\18.1\embedded\ds-5\sw\gcc\arm-linux-gnueabihf\libc\usr\include

~\intelFPGA\18.1\embedded\ip\altera\hps\altera_hps\hwlib\include

~\intelFPGA\18.1\embedded\ip\altera\hps\altera_hps\hwlib\include\soc_cv_av

3.2 Transfert automatique des exécutables vers la cible

Dans le cadre d'une communication par SSH entre la cible et le PC hôte, l'IDE CodeBlocks permet de configurer les variables et commandes nécessaires à l'exécution des programmes en cross compilation.

Pour cela il faut dans un premier définir les variables d'environnement que nous allons utiliser dans les différentes commandes.

Settings -> Global variables

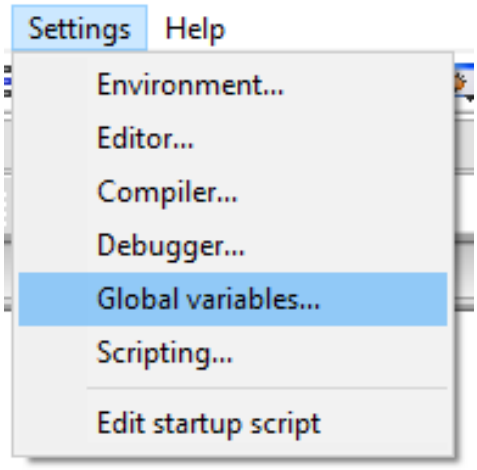

Figure 7 : Onglet Global Varaibles CodeBlocks

Création des variables ip et port nécessaires à la cross compilation de nos programmes.

New -> ip -> base -> entrer adresse ip de la cible

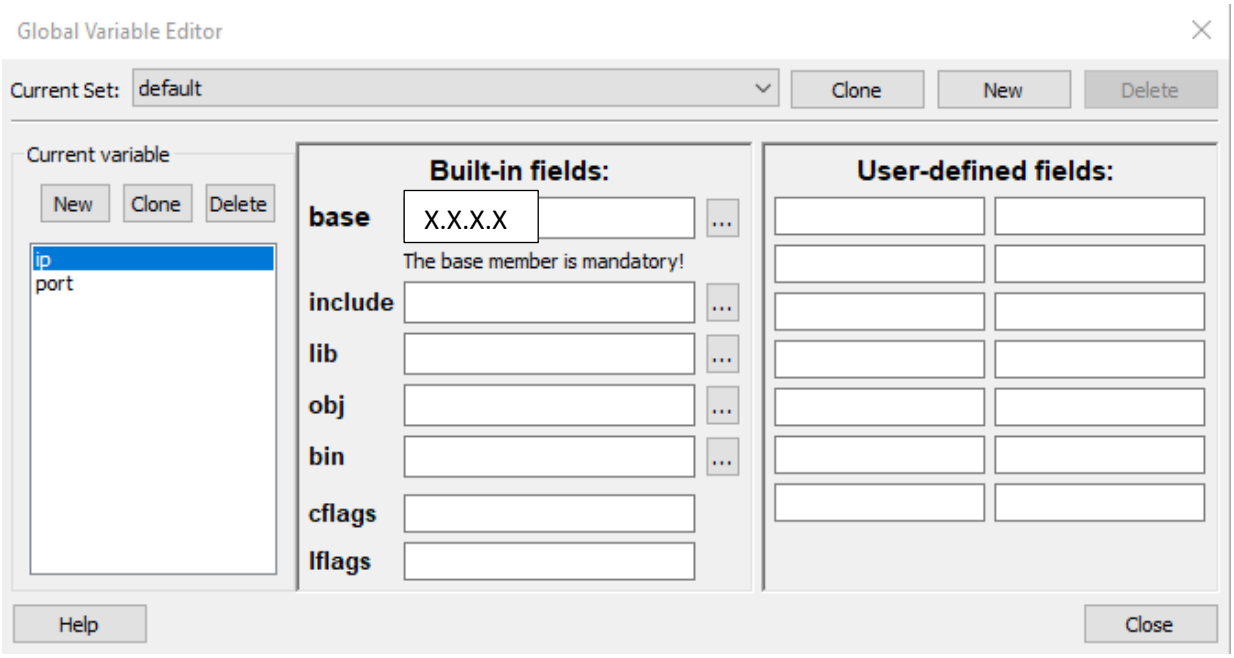

Figure 8 : Global Variable Editor IP CodeBlocks

New -> port -> base -> entrer port d'écoute de la cible (par défaut sur le même réseau local port 22, si réseau passant par la net port 1823)

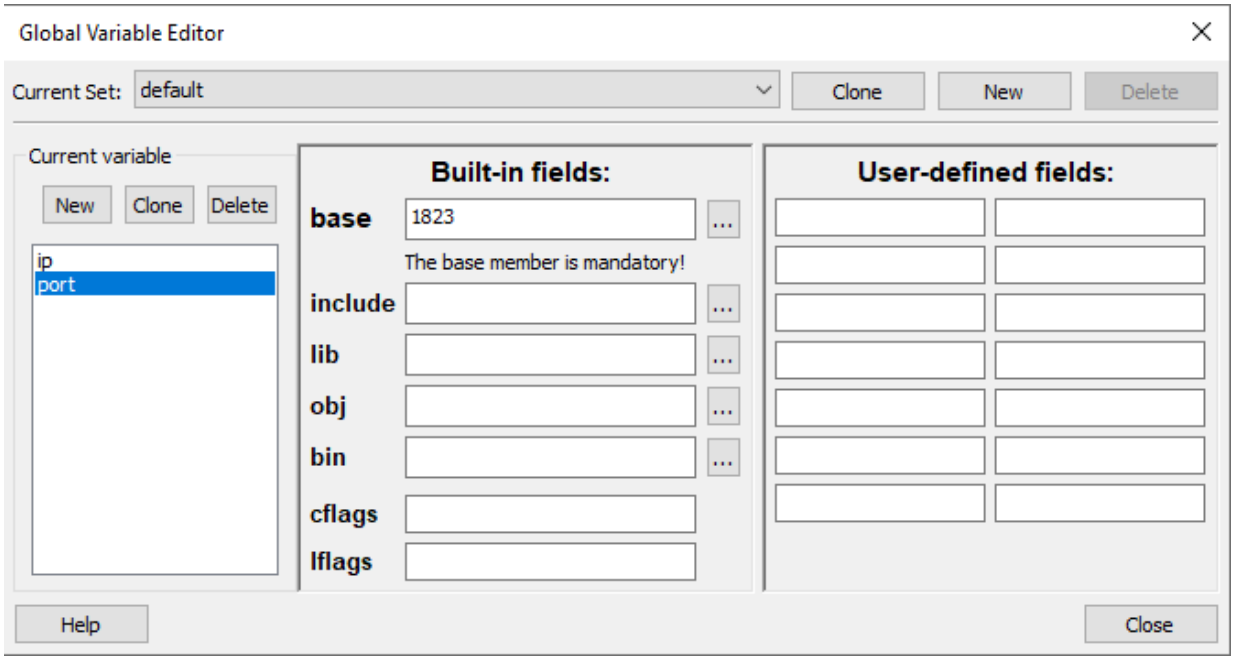

Figure 9 : Global Variable Editor PORT CodeBlocks

Enfin il faut entrer la commande de transfert du programme compilé de l'hote vers la cible, cette commande sera exécutée de manière automatique à la fin de la compilation :

Projet -> Build options -> Racine du projet et/ou Debug et/ou Release -> Post-Build steps

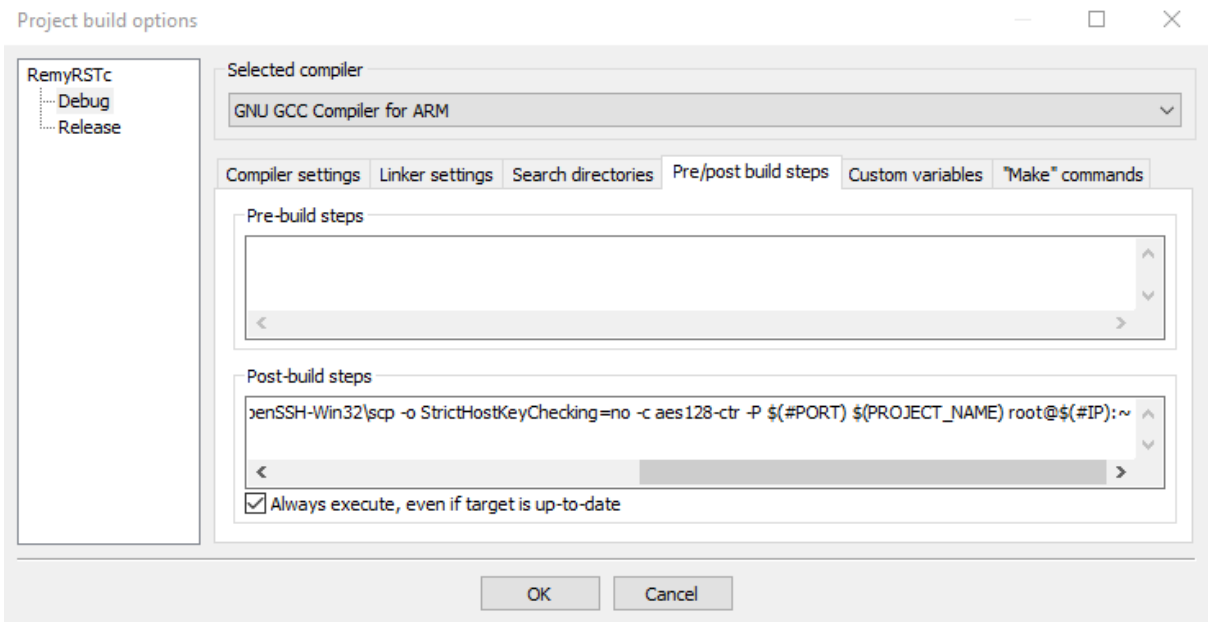

Figure 10 : Post build steps CodeBlocks

~\OpenSSH-Win32\scp -o StrictHostKeyChecking=no -c aes128-ctr -P $(#PORT) $(PROJECT_NAME) root@$(#IP):~

La commande fait référence au dossier dans lequel vous avez installé votre client SSH dans mon cas c'est le dossier OpenSSH-Win32.

De plus, cocher la case "always execute, even if target is up-to-date".

4 Fichiers de données nécessaires au fonctionnement des programmes

L'ensemble des programmes mettant en œuvre la régulation à structure RST, que ce soit en commande de la chaine moteur ou en simulation, font appels au même ensemble de fichiers et transmettent les résultats sous le même format de fichier, comme tel :

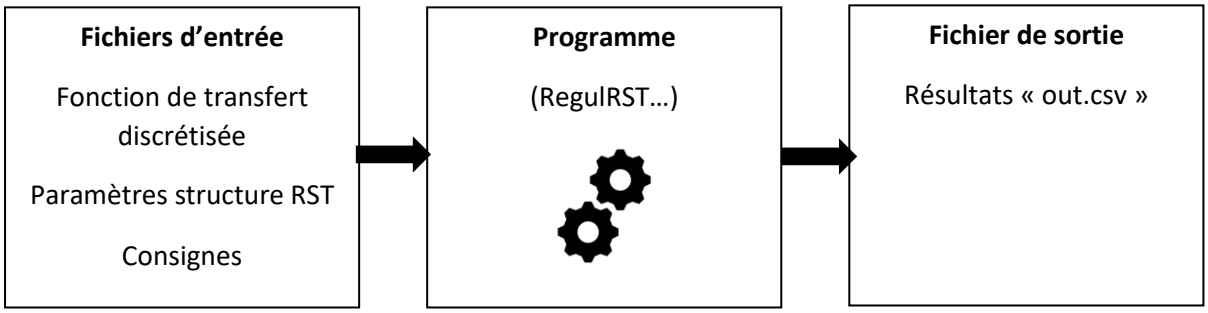

Figure 11 : Schéma gestion fichiers

4.1 Fonction de transfert discrétisée

La fonction de transfert discrétisée est fournie à l'aide du modèle réalisé et simulé sous Scilab. Elle se présente sous la forme de deux fichiers, un pour la partie du numérateur de celle-ci, appelé « GN », et l'autre pour la partie du dénominateur appelé « GD ». Ces deux fichiers contiennent les pôles de cette fonction de transfert discrétisée et seront notamment utilisés lors de l'appel de fonctions contenues dans le fichier « zpol.h » et « zpol.c ».

4.2 Paramètres de la structure RST

Les paramètres de la régulation à structure RST sont déterminés de manière théorique puis validés par la simulation sous Scilab avant d'être transférés sur la carte client par protocole SSH. Ils se composent au nombre de quatre : R, S, TD, TN. Contenant également les pôles de chaque correcteur.

4.3 Consignes d'entrée

Les fichiers de consignes sont propres à chaque programme, et sont au format « .csv » avec une consigne par ligne séparée de la précédente par un retour à la ligne. Dans chaque programme ces consignes sont lues au début afin d'êtres stockées dans un tableau de données. Le programme se charge ensuite d'exécuter les instructions pour chaque consigne avant de se terminer une fois qu'elles ont toutes été traitées.

4.4 Fichier des résultats

Les résultats sont également fournis en sortie à l'aide d'un fichier au format csv appelé « out.csv ». Ces résultats proviennent de la gestion des données à l'aide des fonctions log instanciées dans les fichiers log.h et log.c, ces fonctions permettent de stocker les valeurs de chaque itération dans un tableau de données. Lors de la fermeture du programme ces résultats sont écrits par défaut dans le fichier de sortie « out.csv ». Attention donc à bien veiller à exporter le fichier résultat sur votre machine et le renommer si vous ne voulez pas écraser les données lors de l'exécution d'un autre programme utilisant les log.

5 Simulation RST en C et mesure temps de processus

Pour valider les paramètres de la régulation à structure RST mais surtout son implémentation en langage C sur la cible d'un point de vu des ressources et du temps d'exécution, il est judicieux d'utiliser un programme dédié à cette tâche (*RegulRSTsimu*), du moins un certain nombre de fonctions telle que la mesure du temps de processus.

5.1 Structure du programme

Ce programme utilise dans sa partie principale deux fonctions de lecture de fichiers, celui des consignes et ceux de la fonction de transfert discrétisée GN et GD. Ensuite la boucle principale permet de modéliser la régulation de deux chaines de commande moteur (rst3_t) par itérations qui sont cadencées par la période d'échantillonnage choisie (waitTe). Pour chaque consigne le programme réalise un nombre d'itérations égale à la variable « nbItU_max », avant de passer à la consigne suivante. Après avoir réalisé toutes les itérations sur la dernière consigne le programme sort de la boucle et enregistre les résultats dans le fichier « out.csv ».

5.2 Fonction de mesure du temps d'exécution

La fonction utilisée pour mesurer le temps d'exécution au niveau processus est intitulée *clock_gettime*, elle est contenue dans l'en-tête « time.h ». Elle fait appel à des variables de structure timespec stockant une donnée temporelle en nano secondes, entre autres. L'appel de clock_gettime est réalisé une fois avant d'exécuter les fonctions que l'on souhaite quantifier temporellement puis une deuxième fois après l'appel de celles-ci. Les données du timer processeur sont stockées dans les variables appelées start et stop. Le temps écoulé est donc représenté par la soustraction de ces dernières et stocké dans la variable loopTime.