TP compiler sa toolchain

J.-M Friedt

19 septembre 2016

1 Compiler une toolchain à base de `gcc`

Comme tout ouvrier qui veut connaître et maîtriser ses outils, il est judicieux de compiler sa *toolchain* au lieu de faire confiance à des distributions aux configurations inconnues. Plus important, savoir compiler sa *toolchain* permet d'appréhender toute plateforme matérielle supportée par `gcc` – à savoir toutes les architectures majeures telles que décrites à `https://gcc.gnu.org/backends.html` (notamment SPARC, MIPS, ARM, x86, AVR, ou NIOS et Microblaze pour les utilisateurs de softcores).

Une *toolchain* est formée de trois outils principaux que sont `gcc`, `binutils` et une implémentation de `lib`, qui sera `libc` dans le cas qui nous intéresse. Les outils additionnels sont éventuellement `gdb`, et probablement un outil pour transférer le programme généré vers le microcontrôleur.

1. `gcc` comporte le préprocesseur, l'assembleur, le compilateur et le linker
2. `binutils` fournit les outils de conversion de type et d'analyse du contenu des fichiers générés par `gcc`,
3. `newlib` fournit l'implémentation de `libc` compatible avec les microcontrôleurs à faible ressource.

⚠**Attention :** `newlib` n'est pas approprié pour exécuter GNU/Linux sur le système embarqué. La compilation d'une toolchain permettant de compiler Linux et les outils GNU associés fera l'objet d'un TP ultérieur (`http://jmfriedt.free.fr/A13_v2.pdf`).

Dans la nomenclature de la cross-compilation, la cible (*target*) est le processeur embarqué sur lequel nous allons travailler, codé dans le nom du compilateur [1], alors que l'hôte (*host*) est l'ordinateur sur lequel nous développons, le plus souvent un PC sous processeur x86. La description que nous proposons ci-dessous se généralise à toute architecture supportée par GCC en remplaçant `arm-none-eabi` par le préfixe listé à `https://gcc.gnu.org/backends.html`.

Dans chacun des trois paquets sus-cités, la configuration s'obtient par (par exemple pour ARM) par `./configure -target=arm-elf -prefix=MON_REPERTOIRE .` Les seules subtilités sont que

1. compiler `gcc` impose d'avoir les versions de développement (sur l'hôte, puisque c'est lui qui exécutera `gcc`) de `libmpfr`, `libgmp` et `libmpc`. Sous Debian/GNU Linux, ces dépendances sont résolues par `apt-get install libmpc-dev libgmp-dev libmpfr-dev` (on prendra aussi à l'occasion `zlib1g-dev` si cette bibliothèque est absente),
2. la compilation de la toolchain complète nécessite une première passe de `gcc`, puis `binutils` et `newlib`, et un nouveau passage à `gcc` qui doit finir de se lier à la version de `newlib` compilée pour la cible.

Un script automatise ces démarches dans le cas particulier de ARM Cortex : `summon-arm-toolchain`. Bien que ce script ne soit plus maintenu (parcequ'il est devenu parfait?), il reste la méthode la plus simple et robuste à notre connaissance de générér une chaî ne de compilation stable et pérenne. Parmi les diverses versions disponibles sur `github`, on s'assurera de la compilation de `libstm32` et `libopencm3` font partie des cibles : c'est par exemple le cas de `https://github.com/jmfriedt/summon-arm-toolchain`.

Le script se configure par `LIBSTM32_EN=1`, et selon que les cibles soient exclusivement le STM32 (de type Cortex M3) on choisira ou non `DEFAULT_TO_CORTEX_M3`. Par soucis de souplesse, nous gardons cette option à 0. Le résultat de la compilation se place par défaut dans `$HOME/sat`. Il nous semble judicieux de compiler OpenOCD à la main si cet outil de transfert de programme vers le microcontrôleur est utilisé, car il est mis à jour bien plus souvent que `gcc`.

Ces opérations achevées, `./summon-arm-toolchain` télécharge 130 MB d'archives [2] et met environ, sur un processeur i5 à 4 cœurs, 25 minutes pour compiler tous les outils et les placer dans `$HOME/sat`, avec les exécutables dans le sous répertoire `$HOME/sat/bin` que l'on prendra soin d'ajouter à son $PATH.

Les possesseurs de carte d'évaluation STM32VLDiscovery [3] ou autres cartes d'évaluation de ST Microelectronics exploiteront `st-flash` disponible à `https://github.com/texane/stlink` pour transférer le programme compilé sur PC vers le microcontrôleur.

1. `arm-none-` signifie que nous compilons pour processeur ARM sans système d'exploitation, alors que `m68k-uclinux-` signifie que nous compilons pour une architecture compatible Motorola 68000 supportant un noyau uClinux
2. Si l'accès aux dépôts `ftp` est bloqué par un proxy, l'archive de `newlib` mise à disposition sur le serveur `ftp` de redhat n'est donc pas accessible. On pourra remplacer cette entrée par
`fetch ${NEWLIB} http://artfiles.org/cygwin.org/pub/newlib/${NEWLIB}.tar.gz`
3. `http://fr.farnell.com/stmicroelectronics/stm32vldiscovery/carte-d-evaluation-stm32lv-discovery/dp/1824325`