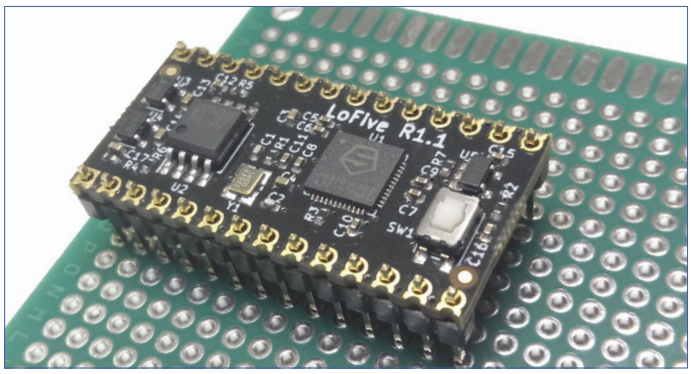
premiers pas avec RISC-V

Mes premiers ébats avec la carte LoFive

Tam Hanna (Allemagne)

Après ARM, RISC-V pourrait devenir le prochain grand succès sur le marché des processeurs. Promue par une fondation [1], cette architecture basée sur un jeu d'instructions est utilisable aussi bien sur l'embarqué que comme processeurs d'ordinateurs et sa licence est gratuite. Toutefois, il n'y a encore que peu de microcontrôleurs ou de cartes disponibles sur le marché. Notre auteur Tam Hanna a essayé la carte économique LoFive.



Comme l'architecture ARM est brevetée comme un tout, le développement continuel de l'architecture X86 contraint le fournisseur d'un noyau X86 dernier cri au paiement de substantiels droits de licence.
Conçue à l'université de Berkeley, l'architecture RISC-V a pour but explicite de chasser sur les terres de l'embarqué aussi bien que sur celles des grands systèmes. L'argument de loin le plus convaincant des promoteurs est la gratuité de la licence. La **figure 1**, tirée du site [2], indique que l'utilisation de l'architecture est gratuite.

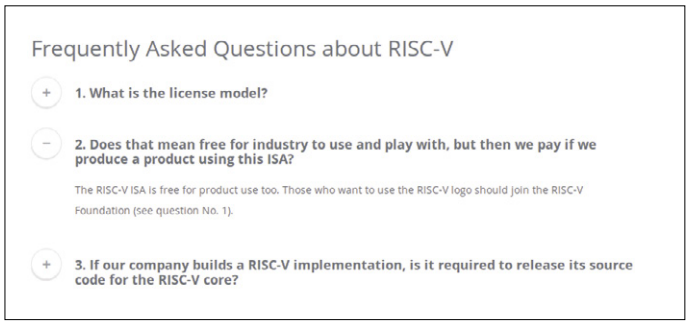


Figure 1. L'architecture RISC-V est d'utilisation gratuite et ne comporte pas de licences contraignantes.

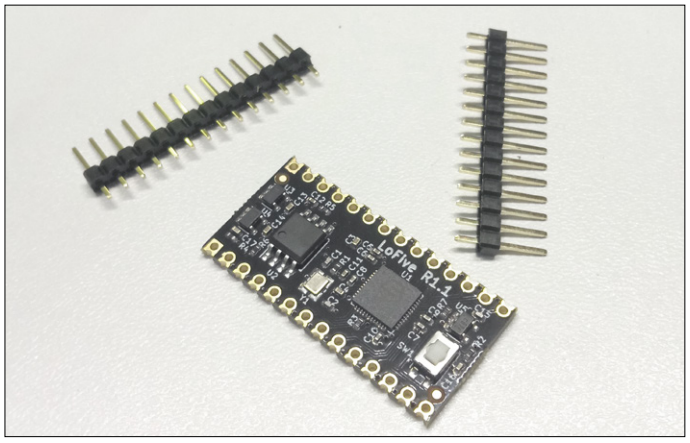


Figure 2. La carte peut aussi être utilisée comme CMS.

Alors que le développement de l'architecture RISC-V est en cours depuis de nombreuses années, du matériel tangible n'est disponible que depuis peu. Une carte d'évaluation à bas prix est disponible chez *GroupGets* sous le nom de *LoFive*, avec un processeur RISC-V *Freedom* E310 du fondeur *SiFive* [3].
Du fait du développement rapide de l'architecture, il en existe maintenant deux versions ; nous allons travailler avec la version la plus récente, désignée par R1 ou 1.1. Notez que la version ancienne se distingue notablement de la nouvelle et que les opérations présentées ici ne lui sont pas directement applicables.

Opérations préparatoires

Quand on déballe *LoFive* (**fig. 2**), on remarque un groupe de pastilles de connexion nues. Le fabricant propose des brochages à usage multiple permettant soit le soudage de barrettes au pas de 2,54 mm, soit le soudage direct sur une carte mère. Pour mon confort, je préfère une platine d'expérimentation classique. La carte *LoFive* est actuellement disponible pour 25 € environ. Ce prix modeste s'explique, entre autres, par l'absence d'une puce de programmation, ce qui oblige l'utilisateur à prévoir une carte d'adaptation USB/série FT2232H-56Q du fabricant FTDI. Le **tableau 1** donne les liaisons entre les deux cartes.
Si l'on envisage un travail sérieux avec *LoFive*, il est recommandé de commencer par réaliser une carte d'accueil. Pour les premières expérimentations, des fils Dupont font l'affaire (**fig. 3**). Dans mon labo, qui fourmille de perturbations électromagnétiques du fait de la présence de nombreuses lampes à LED, des fils de 10 cm de long n'ont guère posé de problèmes ; ceux qui se sont manifestés lors de l'identification de la cible ont disparu à la passe suivante.

Tableau 1. Connexions entre LoFive et l'adaptateur de programmation	
broche LoFive	**broche de l'adaptateur FTDI**
+5Vin	VBS
GND	GND
TRSTN	AD5
TCK	AD0
TDO	AD2
TMS	AD3
TDI	AD1
UART0.TX	BD1
UART0.RX	BD0

Début des travaux

La plupart des éléments nécessaires étaient déjà présents sur ma station de travail sous Ubuntu 18.04. Saisissez la commande suivante pour installer les paquets :

```
sudo apt-get install autoconf automake libmpc-dev
   libmpfr-dev libgmp-dev gawk bison flex texinfo
   libtool libusb-1.0-0-dev make g++ pkg-config
   libexpat1-dev zlib1g-dev
```

Il y a ici matière à frustration pour les utilisateurs de Windows ou de MacOS. Linux est devenu un quasi-standard dans le monde du développement sur RISC-V ; en particulier, pour travailler avec les cartes encore peu répandues, les chances de Windows sont faibles. Basé sur *Eclipse*, l'environnement de développement *Freedom* Studio, proposé par le fabricant de processeurs *SiFive*, supporte la carte de développement officielle *HiFive*. Au moment où j'écris cet article, la carte *LoFive* ne bénéficie guère de la sympathie de cet EDI.
L'étape suivante consiste à télécharger le SDK au moyen du client `git` par les commandes :

```
tamhan@TAMHAN18:~$ cd riscvneu/
tamhan@TAMHAN18:~/riscvneu$ git clone --recursive
   https://github.com/mwelling/freedom-e-sdk.git
. . .
tamhan@TAMHAN18:~/riscvneu$ cd freedom-e-sdk
tamhan@TAMHAN18:~/riscvneu/freedom-e-sdk$ git
   checkout lofive-r1
M            freedom-devicetree-tools
. . .
```

Ce qui est nouveau ici, c'est qu'après le téléchargement de l'archive principale, il faut encore télécharger un sous-module, indispensable pour la mise en œuvre des modules spécifiques *LoFive* R1.
À l'étape suivante, il faut donner une commande de synchronisation à *Github* avant de télécharger un autre sous-module :

```
tamhan@TAMHAN18:~/riscvneu/freedom-e-sdk$ git
   submodule sync
Synchronizing submodule url for 'doc/html'
. . .
tamhan@TAMHAN18:~/riscvneu/freedom-e-sdk$ git
   submodule update --init --recursive
. . .
Submodule path 'freedom-devicetree-tools': checked
   out '4f25a7512696f0b41c17e517d39d097499b931a7'
```

L'appel à `sync` n'est pas absolument indispensable, mais une précaution raisonnable en cas de doute : j'ai observé des messages d'erreurs au cours de mes tests.
La chaîne d'outils RISC-V s'appuie en général sur le compilateur GCC et OpenOCD, d'autres cartes d'évaluation utilisent une interface de programmation *Seggers*.
Comme la compilation de la chaîne d'outils sur la station de travail requiert pas mal de temps, *SiFive* a entretemps mis à disposition un paquet plus ou moins fini. Ouvrez pour commencer l'URL https://www.sifive.com/boards dans votre navigateur favori et déroulez la page vers le bas jusqu'à la rubrique *Prebuilt RISC-V GCC Toolchain and Emulator*.
Pour les étapes suivantes, il nous faut en tout cas les versions *GNU Embedded Toolchain — v2019.08.0* et *OpenOCD — v2019.08.2*. Cliquez sur les deux liens et extrayez leurs

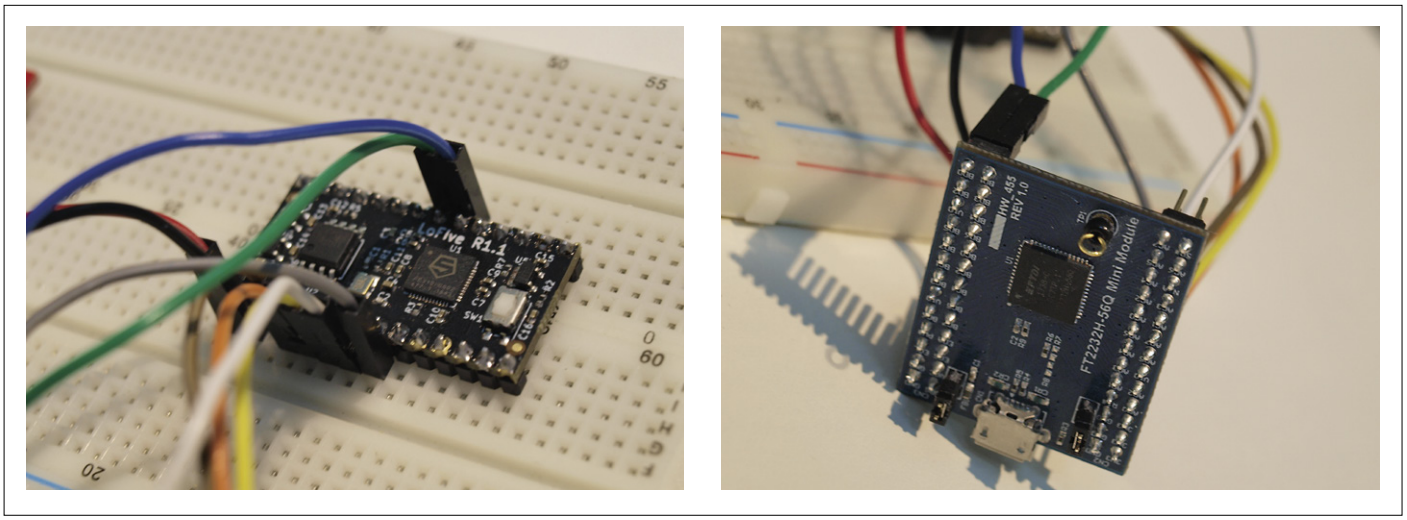


Figure 3. Un module USB/série de FTDI est utilisé pour la programmation.

contenus dans des sous-répertoires du répertoire de travail. Dans la suite, j'utilise le nom de répertoire *riscvneu*; la réponse à la commande `dir` est alors :

```
tamhan@TAMHAN18:~/riscvneu/$ dir
freedom-e-sdk/
riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-linux-
   ubuntu14/
riscv-openocd-0.10.0-2019.08.2-x86_64-linux-ubuntu14/
```

Pour des raisons pratiques, la chaîne d'outils suppose qu'elle peut localiser le compilateur et compagnie grâce à des variables d'environnement : cela réduit la configuration d'une nouvelle chaîne d'outils à une simple édition de ces variables.

Comme les modifications du paramètre PATH sont généralement laborieuses, nous utiliserons dans la suite des chemins temporaires :

```
tamhan@TAMHAN18:~/riscvneu/freedom-e-sdk$ export
   RISCV_OPENOCD_PATH=/home/tamhan/riscvneu/riscv-
   openocd-0.10.0-2019.08.2-x86_64-linux-ubuntu14/
tamhan@TAMHAN18:~/riscvneu/freedom-e-sdk$ export
   RISCV_PATH=/home/tamhan/riscvneu/riscv64-unknown-
   elf-gcc-8.3.0-2019.08.0-x86_64-linux-ubuntu14/
```

Notez que la commande export n'est valable que dans la fenêtre de terminal où elle a été saisie. Si vous voulez utiliser une autre fenêtre ou si vous la fermez involontairement, il faudra recommencer tout le processus. Comme les noms de répertoires cités ne sont valables que sur ma machine, il est également improbable que votre nom d'utilisateur soit *tamhan*.

Configuration de *OpenOCD*

OpenOCD est un outil relativement compliqué de la panoplie de programmation Linux embarqué. Il prend contact avec le système cible en passant par différents systèmes nommés probes puis accepte des commandes à travers un port réseau de la station de travail. *OpenOCD* est habituellement associé à l'outil *GDB-Debugger*.

Le problème de *OpenOCD* est que sa configuration définie dans les fichiers `.conf` est très « serrée ». Comme le développeur qui a porté la chaîne vers le processeur RISC-V ne travaille pas sous Ubuntu 18.04, il se produit à l'exécution des erreurs du genre suivant :

```
tamhan@TAMHAN18:~/riscvspace/freedom-e-sdk$ sudo make
   upload PROGRAM=led_fade BOARD=freedom-e300-lofive
[sudo] password for tamhan:
. . .

Error: unable to open ftdi device with vid 0403, pid
   6010, description 'FT2232H-56Q MiniModule', serial
   '*' at bus location '*'
```

Si vous observez ce genre d'erreur sur votre système, il faut commencer par examiner le texte de *description* du pilote kernel du module FTDI connecté, qu'on peut obtenir facilement avec la commande *lsusb* :

```
tamhan@TAMHAN18:~$ lsusb
. . .
Bus 001 Device 009: ID 0403:6010 Future Technology
   Devices International, Ltd FT2232C Dual USB-UART/
   FIFO IC
```

Ensuite, il faut localiser la description des problèmes, ce qui s'effectue commodément au moyen du bon vieil utilitaire G*rep*. La réponse se présente de la manière suivante :

```
root@TAMHAN18:~/riscvneu/freedom-e-sdk#  grep -r
   FT2232 *
bsp/lofive-r1-bootloader/openocd.cfg:ftdi_device_desc
   "FT2232H-56Q MiniModule"
bsp/lofive-r1/openocd.cfg:ftdi_device_desc "FT2232H-
   56Q MiniModule"
```

Ouvrez ensuite les deux fichiers avec l'éditeur de votre choix et cherchez le bloc de déclarations suivant :

```
interface ftdi
#ftdi_device_desc "Dual RS232-HS"
ftdi_device_desc "FT2232C Dual USB-UART/FIFO IC"
ftdi_vid_pid 0x0403 0x6010
```

Tout d'abord, cela informe *OpenOCD* qu'il a affaire à une interface FTDI. Ensuite, on a l'ensemble des deux identifieurs numériques et du texte descriptif qui doit servir à découvrir le matériel cible.
Comme dans notre cas la combinaison du VID et du PID suffit, nous pouvons simplifier la configuration :

```
interface ftdi
ftdi_vid_pid 0x0403 0x6010
```

On démarre

Un problème auquel on se heurte depuis l'apparition de la première carte d'évaluation est sans conteste celui de l'obsolescence du progiciel. Rien d'étonnant, car pendant leur trajet du constructeur à la paillasse du développeur, les cartes séjournent pendant un bon moment chez le distributeur où leur logiciel a largement le temps de prendre la poussière.
Notre première action va donc consister à mettre à jour le chargeur d'amorçage. Pour cela nous devons commencer par nettoyer l'environnement de configuration, qui véhicule des artefacts de la machine du constructeur :

```
root@TAMHAN18:~/riscvneu/freedom-e-sdk# make
   PROGRAM=lofive-boot TARGET=lofive-r1-bootloader
   clean
make -C /home/tamhan/riscvneu/freedom-e-sdk/software/
   lofive-boot PORT_DIR= clean
make[1]: Entering directory '/home/tamhan/riscvneu/
   freedom-e-sdk/software/lofive-boot'
. . .
```

Pour interagir avec une carte RISC-V par l'intermédiaire du SDK *Freedom*, on utilise ordinairement l'outil make, qui, outre les variables `PROGRAM` et `TARGET`, s'attend à trouver une commande à exécuter.

Le téléversement du chargeur d'amorçage s'effectue ensuite comme suit :

```
root@TAMHAN18:~/riscvneu/freedom-e-sdk# make
   PROGRAM=lofive-boot TARGET=lofive-r1-bootloader
   upload
cd /home/tamhan/riscvneu/freedom-e-sdk/bsp/lofive-r1-
   bootloader/build/debug/ && \
```

Par commodité, l'auteur a inclus ces diverses commandes dans un script Root-Shell. Si ce n'est pas souhaité, on peut très bien saisir les diverses corrections à la main.
Après le téléversement avec succès du chargeur d'amorçage, nous pouvons nous occuper d'un premier programme :

```
root@TAMHAN18:~/riscvneu/freedom-e-sdk# make
   PROGRAM=sifive-welcome TARGET=lofive-r1 upload
```

Après l'exécution de cette commande, on observe sur les sorties MLI une forme d'onde relativement lente. Si vous pouvez l'observer sur votre oscilloscope (mode de défilement *Roll* indispensable ou recommandé), la configuration a été effectuée avec succès. Pendant son exécution, notre programme de démo fournit des informations d'état récupérables sur la fenêtre de commande ou celle d'un quelconque autre émulateur de terminal :

```
tamhan@TAMHAN18:~$ screen /dev/ttyUSB1 115200
[screen is terminating]
tamhan@TAMHAN18:~$
```

Lors de l'utilisation de cette commande, il faut prendre garde de démarrer *screen* avant le programme proprement dit. Le terminal *screen* n'est pas évident à arrêter : le plus simple est de taper Ctrl/A puis K et de répondre Y à la demande de confirmation d'arrêt de *screen*.

Où se trouve le code ?

Pour finir, il reste la question de savoir où se trouve le code « à traiter ». La réponse est le répertoire du logiciel ; dans le cas de notre exemple de projet le contenu se présente de la manière suivante :

```
root@TAMHAN18:~/riscvneu/freedom-e-sdk/software/
   sifive-welcome# ls
debug  LICENSE  LICENSE.Apache2  LICENSE.MIT
   Makefile  README.md  sifive-welcome.c
```

Le fichier *make* responsable de la conduite des différents processus de compilation se limite en général à inclure un fichier *make* fourni par *SiFive* – il s'occupe ensuite de réaliser les diverses commandes comme *upload*.

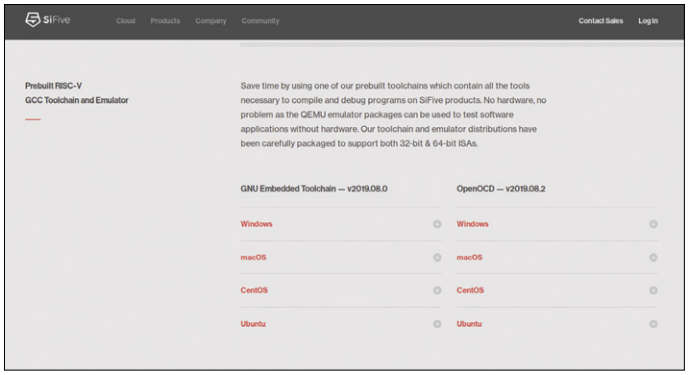


Figure 4. L'option de téléchargement pour la chaîne d'outils RISC-V est bien dissimulée.

On me presse, pour des raisons de place, de mettre un terme à cette évaluation, mais je voudrais encore souligner qu'à l'heure où j'écris ces lignes, la chaîne d'outils n'est toujours pas en mesure d'accomoder C++.

Cela vaut-il la peine ?

Si l'on est à la recherche d'un microcontrôleur économique, on le trouvera sans doute plus facilement chez ST, *Microchip* ou divers fournisseurs chinois que chez RISC-V. Car même une architecture ouverte n'offre pas la gratuité du silicium. Vu le petit volume des ventes, l'effet d'échelle chez ARM ou d'autres architectures propriétaires fait que leur offre reste pour l'instant globalement plus intéressante.

S'occuper de RISC-V vaut déjà la peine pour des personnes ayant un intérêt académique ou de hacker pour la plateforme RISC-V. Il est à prévoir toutefois que cette plateforme parvienne à s'assurer quelques succès notables dans le domaine de l'embarqué, ne serait-ce qu'à cause du très large soutien des nombreuses sociétés qui y participent. ⏮

(191138-03 VF Helmut Muller)

Liens

[1] Fondation RISC-V: https://riscv.org/
[2] FAQ sur RISC V: https://riscv.org/faq/
[3] LoFive sur GroupGets: https://groupgets.com/manufacturers/qwerty-embedded-design/products/lofive-risc-v