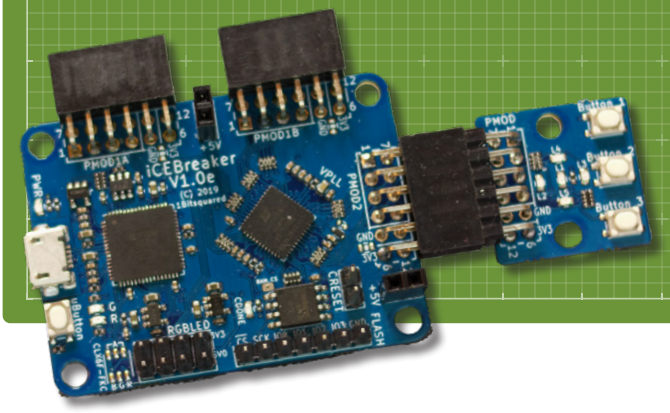
Construisez votre contrôleur RISC-V

Débuter sur le noyau logiciel RISC-V NEORV32 pour FPGA à faible coût



Mathias Claußen (Elektor)

Expérimenter le RISC-V vous tente ? C'est possible, sans puce câblée « en dur », avec le noyau logiciel RISC-V NEORV32 pour FPGA à faible coût.

Pour expérimenter avec un microcontrôleur *RISC-V*, il existe désormais des processeurs utilisant cette architecture de jeu d'instructions standard ouverte, par ex. le nouveau *ESP32-C3*. Mais il y a des alternatives à l'emploi d'une puce câblée « en dur » : *NEORV32* aborde par ex. le concept d'un *softcore* (noyau logiciel) RISC-V réalisable à l'aide d'un FPGA. Un tel processeur est un peu moins puissant qu'un processeur câblé, mais il est beaucoup plus souple et vous permet de tester différents projets et d'intégrer tout périphérique développé en interne dans votre matériel. Cette voie, riche d'enseignements, vous fera par ex. entrer dans les rouages intimes d'un processeur.

Une application pratique

Même ceux qui ont déjà expérimenté les FPGA feront vite face à des obstacles dès qu'il s'agira de configurer leur propre projet de petit processeur. L'ensemble du processus est assez complexe, même pour un ingénieur expérimenté, comme l'a montré notre série sur le projet SCCC de Martin Oßmann [1]. Heureusement, il n'est pas nécessaire de partir de zéro. On peut exploiter certaines solutions (quasiment clés en main) déjà développées par des experts qui les ont mises à disposition gratuitement.

L'une d'elles, également sous licence *open source*, sera utilisée ici. Cet article n'est pas du tout un cours complet sur le RISC-V ou les FPGA, mais il devrait accélérer le cycle d'apprentissage en montrant comment construire et faire fonctionner une 1ère application pratique aussi vite que possible.

FPGA, synthèse, *softcore*, RISC-V et compilateur

Si vous devez choisir un µcontrôleur polyvalent pour une application donnée, différents facteurs peuvent influencer votre décision. L'une des considérations de base est la gamme de périphériques intégrés proposée par le µcontrôleur. Celle-ci est figée dans le matériel par la version de la puce et ne peut être modifiée. Cette approche permet de fabriquer des puces peu coûteuses aux performances optimisées. Il en va différemment si vous basez votre propre contrôleur sur un FPGA. Un FPGA est constitué d'un ensemble de cellules logiques dites tables de consultation (*LUT* = *LookUp Table*) interconnectables à volonté via une matrice. La **figure 1** illustre les blocs existant dans une telle LUT. On a ici un élément LUT-4 avec 4 signaux d'entrée, une table de vérité, une bascule et un multiplexeur en sortie. La table de vérité permet de former toute porte élémentaire telle qu'un ET, un OU, un NON ou un OU EXCLUSIF. Associés à la matrice du FPGA, ces composants servent à créer des structures plus grandes comme des mémoires, additionneurs ou multiplexeurs, qui à leur tour peuvent être combinés pour former un système encore plus complexe tel qu'un processeur ou un système complet sur puce. Le FPGA peut être vu comme un jeu de briques de construction que l'on assemble à volonté pour construire par ex. un château, puis le démonter pour construire un pont ou toute autre structure, en réutilisant les mêmes briques.

Pour que le FPGA exécute une fonction spécifique, il doit être configuré de manière adéquate. Cependant, la fastidieuse tâche de création des LUT et de leur connexion à la matrice ne vous incombe pas. C'est celle

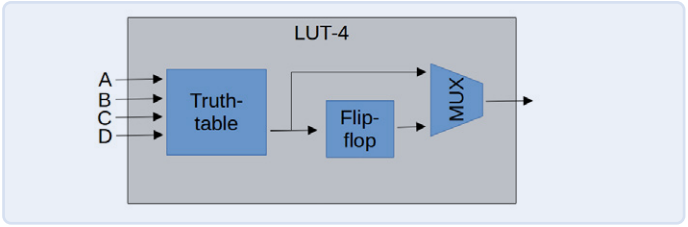


Figure 1. Flux de signaux dans une LUT-4.

des outils de synthèse FPGA. Il suffit de décrire la fonction souhaitée dans des langages tels que Verilog ou VHDL. Les outils de synthèse comprennent généralement ces deux langages de description du matériel. L'outil de synthèse connaît les caractéristiques du FPGA et, d'après le langage de description, il crée une suite de données binaires (ou flux de bits = *bitstream*) qui sert ensuite à configurer le FPGA. La **figure 2** montre la séquence de synthèse simplifiée d'un FPGA. La plupart des fabricants de FPGA offrent des outils gratuits fonctionnant sous Windows et Linux. Des solutions *open source* effectuent ce processus de synthèse pour certains FPGA. Elles s'exécutent en général sur système d'exploitation Unix, par ex. Linux ou macOS.
Un FPGA peut réaliser des fonctions logiques simples, des processeurs, voire des systèmes de processeurs… S'il abrite assez de LUT ! Verilog et VHDL savent aussi décrire ces derniers. Puisque le processeur n'est pas irréversiblement câblé dans le silicium du FPGA, on peut adapter sa fonction ou son comportement en modifiant la description matérielle. Un tel processeur est appelé *softcore* (noyau logiciel). Il en existe pour diverses architectures logicielles. Parfois, ces noyaux contiennent aussi des périphériques (interfaces de bus, etc.). L'architecture de processeur RISC-V à code source ouvert remporte un vif succès comme softcore. À l'heure où nous écrivons, le choix des MCU RISC-V câblés est encore gérable. Une première expérience de cette architecture peut ainsi être acquise en softcore en créant votre propre MCU RISC-V pour le tester et l'étudier.
Avec RISC-V, il n'y a ni frais de licence, ni accord de non-divulgation (NDA) ou autre accord de licence, tous généralement associés à l'exploitation des MCU des fabricants. RISC-V implique aussi la disponibilité de compilateurs de code source en C, notamment le compilateur GNU C (GCC). Et donc, les bibliothèques de base sont aussi en place.

NEORV32

Le projet NEORV32 de Stephan Nolting [4] montre qu'il n'y a pas besoin d'un FPGA coûteux pour construire son propre système sur puce (SoC = *System on Chip*). C'est un processeur compatible RISC-V avec tous les périphériques nécessaires pour fonctionner comme un SoC de type MCU sur un FPGA. Ce projet est entièrement implémenté en VHDL et n'est donc pas lié aux fabricants de FPGA. NEORV32 est entièrement *open source* et est aussi doté d'une documentation complète, d'un cadre logiciel et d'outils.
La **figure 3** montre les modules périphériques présents. Pêle-mêle on a des interfaces SPI, I²C et des UART, une interface WS2812, des GPIO, des unités MLI (PWM). Les utilisateurs débutants et avancés disposent ainsi d'un système complet avec environnement de développement intégral pour le NEORV32 incluant toutes les bibliothèques pour le matériel et les périphériques. De plus, des configurations types existent pour certaines cartes FPGA : vous êtes tout de suite opérationnel. Mais quelles sont ces cartes « prêtes à l'emploi » ? Est-ce difficile d'installer NEORV32 sur une carte FPGA non directement prise en charge ?

Choix du FPGA

En théorie, toute carte équipée d'un FPGA avec suffisamment de ressources convient, puisque NEORV32 n'utilise pas d'extensions

Figure 2. Étapes du processus de synthèse d'un FPGA.

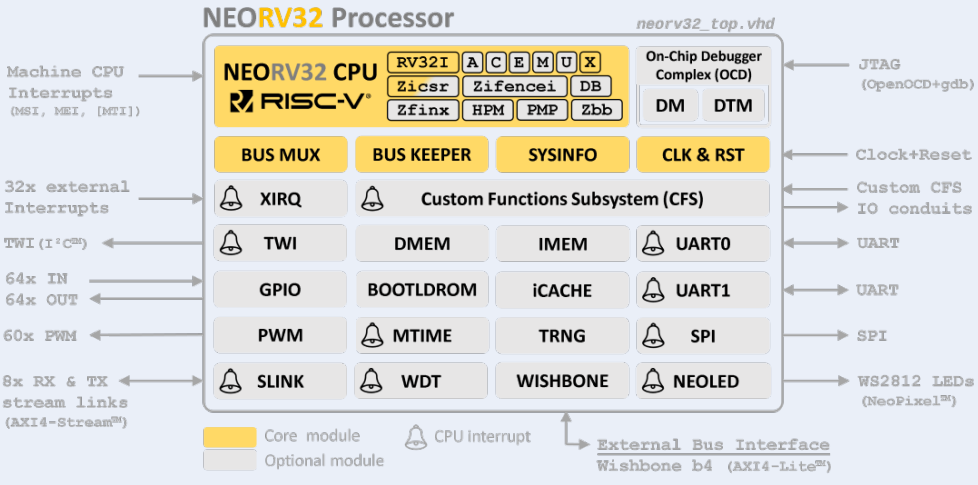


Figure 3. Schéma de principe des blocs fonctionnels de NEORV32 (source : Github / Nolting, S. [20]).

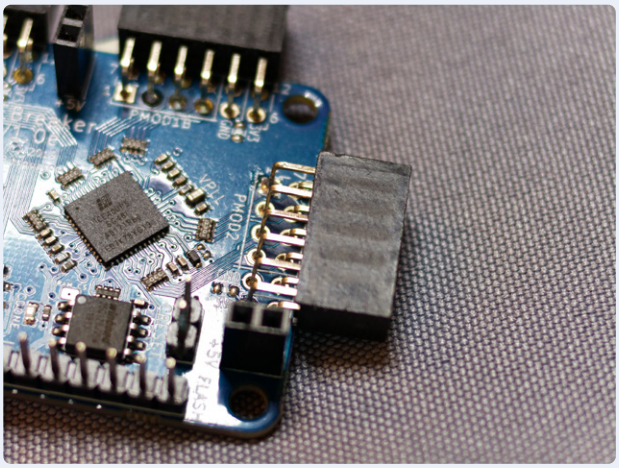


Figure 4. L'iCE40UP5K en format QFN mesurant 7×7 mm.

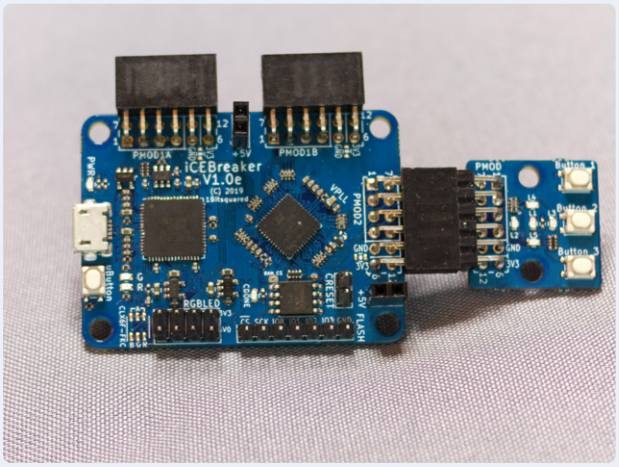


Figure 5. Carte iCEBreaker avec connecteur PMOD.

spécifiques au fabricant. Le FPGA iCE40UP5K de Lattice [5] équipe plusieurs cartes bon marché.

Le FPGA iCE40UP5K de Lattice est actuellement le plus puissant des iCE40 Ultra-Plus. Au total, il dispose de : 5280 LUT, 120 Kbits (15 Ko) de RAM EBR, de 1024 Kbits (128 Ko) de SPRAM et d'unités fonctionnelles câblées SPI et I²C. C'est une solide plate-forme pour construire vos propres projets. Ses caractéristiques, le type de boîtier de la puce et son faible coût contribuent à rendre ce FPGA intéressant. Le format QFN-48 de 7 × 7 mm (**fig. 4**) est beaucoup plus facile à exploiter que le BGA et son prix d'environ 5 € par puce le place dans une fourchette de prix intéressante. Aujourd'hui (octobre 2021), ce FPGA est proposé autour de 5 à 6 € l'unité chez tous les distributeurs, mais il est en rupture de stock et le délai de livraison peut atteindre 46 semaines.

Pour notre projet, vous n'aurez pas à concevoir un circuit imprimé pour le FPGA. Les FPGA iCEBreaker (**fig. 5**) et iCEBreaker Bitsy (**fig. 6**) de 1BitSquared [6] sont deux cartes de développement en matériel ouvert sur lesquelles le FPGA iCE40UP5K est déjà monté. En matériel ouvert, la carte UPduino V3.0 [7] de tinyVision.ai est une alternative (**fig. 7**).

Le projet NEORV32 est 100 % compatible avec l'UPduino V3.0 et comprend d'autres projets types. Les modifications requises sur la carte FPGA iCEBreaker sont très rapides à effectuer. Pour commencer, nous utiliserons l'UPduino V3.0, puis nous indiquerons comment adapter notre projet d'exemple pour qu'il fonctionne sur la carte iCEBreaker.

Avec ce FPGA, vous avez un SoC avec 64 Ko d'espace pour les applications, 64 Ko de RAM, une interface SPI, une I²C, une UART, 4 broches d'entrée, 4 de sortie, 3 MLI, ainsi que le cœur RV32IMAC fonctionnant à 18 MHz.

La chaîne d'outils

Vous pouvez prendre deux voies pour sélectionner la chaîne d'outils pour le Lattice iCE40up5k : soit les outils de Lattice [8], soit la suite OSS CAD entièrement basée sur des outils *open source* de YosysHQ [9]. Pour ce projet, nous choisirons la 2e voie, celle de l'*open source*. Cela signifie que notre système d'exploitation sera Ubuntu 20.04 LTS et que la suite OSS CAD sera utilisée pour synthétiser le NEORV32 pour l'iCE40UP5K.

Outre les outils du FPGA, un programme de démonstration du processeur RISC-V synthétisé sera aussi compilé ultérieurement : ce sera notre version du classique message « Hello World », envoyé à l'aide de l'UART. Là encore, des outils *open source* sont utilisés pour produire une chaîne d'outils adéquate (similaire à celle du Kendryte K210 [10]). Il existe un compilateur GNU C (GCC) et des bibliothèques complémentaires pour les périphériques NEORV32.

Mise en place

Comme mon collègue Clemens Valens l'expose dans sa vidéo [11], travailler avec des FPGA, c'est un peu comme cuisiner. Assurez-vous d'abord que tous les ingrédients et ustensiles (outils) nécessaires sont là. Pour éviter tout risque d'affecter l'installation principale de votre OS, vous pouvez travailler sur une machine virtuelle. Ici, nous supposons qu'une version 20.04 d'Ubuntu a été fraîchement installée sur un système AMD64. L'utilisation d'un Raspberry Pi devrait être possible, puisqu'Ubuntu 20.04 et l'OS du Raspberry Pi sont tous deux basés sur Debian, mais l'architecture peut engendrer de petites différences. Pour ce projet, je n'ai utilisé qu'Ubuntu 20.04 sur une machine AMD64.

Pour synthétiser le « matériel » du FPGA, la version à jour de la suite OSS CAD [12] doit être téléchargée dans le dossier de base. Ce fichier s'appelle *oss-cad-suite-linux-x64-xxxxxxxx.tgz*. Dans un terminal, décompressez ce fichier en utilisant `tar -xvzf oss-cad-suite-linux-x64-xxxxxxxx.tgz` et ensuite déplacez-le vers */opt* avec `sudo mv ~/oss-cad-suite /opt/`. Afin que le dossier soit accessible ultérieurement, définissez les droits avec `chmod 777 /opt/oss-cad-suite -R`. La bibliothèque *libgnat-9* est également requise, installez-la avec `sudo apt install libgnat-9`. Les outils FPGA sont alors en place.

Compilateur

Nous avons ensuite besoin des fichiers associés à NEORV32 [4] depuis le dépôt GitHub. À cet effet, nous allons installer *git* en entrant `sudo apt install git`. Puis : `git clone https://github.com/stnolting/neorv32.git ~/neorv32` clône le dépôt de NEORV32.

Pour communiquer plus tard avec le NEORV32, un programme de terminal sera nécessaire. En général, j'utilise un programme éprouvé : HTerm de Tobias Hammer. Il peut être téléchargé depuis sa page d'accueil [13] avec `wget www.der-hammer.info/terminal/hterm-linux-64.tgz`. Ensuite `mkdir ~/hterm && tar -xvf hterm-linux-64.tgz -C ~/hterm` extrait le contenu du fichier *tgz* dans le dossier *~/hterm*. Pour qu'un utilisateur puisse ensuite accéder aux interfaces série, il faut l'ajouter comme membre du groupe *dialout*. À cet effet, utilisez le terminal et entrez `sudo adduser $(whoami) dialout`.

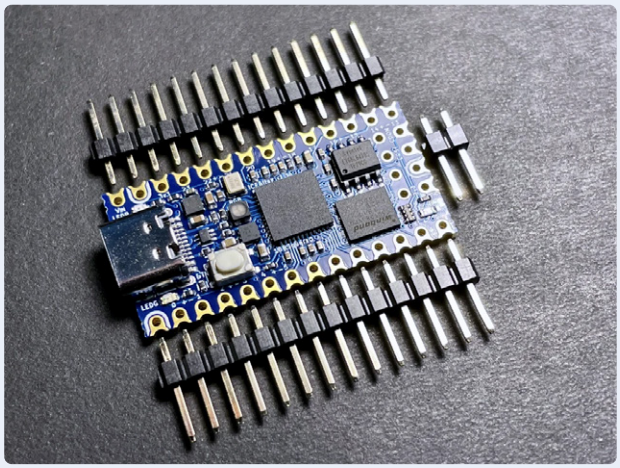


Figure 6. iCEBreaker Bitsy (source : https://cdn.shopify.com/s/files/1/1069/4424/products/IMG_3859_large.jpg / 1BitSquare).

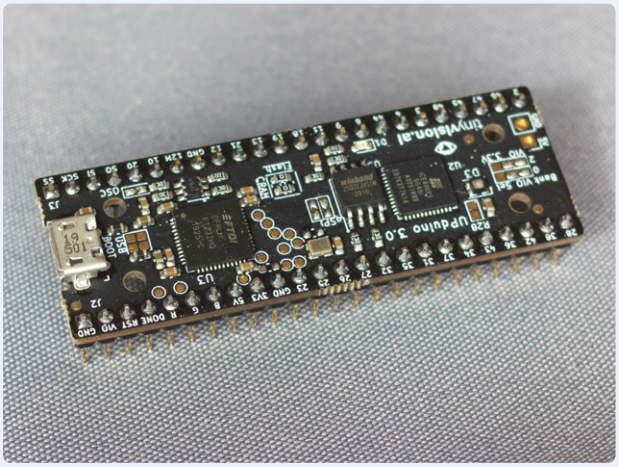


Figure 7. L'UPduino V3.0 avec les barrettes de broches installées.

Le compilateur C doit maintenant lui-même subir une compilation. Dans les exemples pour iCE40up5k, NEORV32 est configuré en RV32IMAC, les commandes de multiplication et de division d'entiers sont donc disponibles (cf. l'encadré **Convention de nomenclature RISC-V**). Le compilateur doit être compilé selon cette architecture particulière et les extensions de commandes ; la documentation de NEORV32 [14] indique pourquoi cette adaptation est importante.
Avec un terminal, tapez `cd ~` pour aller dans le dossier de base puis tapez `git clone https://github.com/riscv/riscv-gnu-toolchain --recursive` pour cloner la chaîne d'outils RISC-V. Il faudra encore installer quelques compléments logiciels comme suit :

```
sudo apt-get install autoconf automake autotools-
   dev curl python3 libmpc-dev libmpfr-dev libgmp-
   dev gawk build-essential bison flex texinfo gperf
   libtool patchutils bc zlib1g-dev libexpat-dev
```

Enfin, le compilateur et les bibliothèques peuvent être compilés.
Avec les processeurs RISC-V, les commandes prises en charge sont hiérarchisées. Cela signifie, par ex., que le code compilé pour un modèle RV32I sera exécutable sur un modèle RV32IMAC, mais pas l'inverse. C'est pourquoi il vaut mieux compiler d'abord la chaîne d'outils pour qu'elle soit compatible avec le modèle de plus petit dénominateur commun. À l'aide du terminal, tapez `cd ~/riscv-gnu-toolchain` pour aller au répertoire de la chaîne d'outils clonée puis `./configure --prefix=/opt/riscv --with-arch=rv32i --with-abi=ilp32` pour préconfigurer la chaîne d'outils de compilation. Dès lors, nous pouvons lancer la tâche de compilation avec `sudo make`. La chaîne d'outils sera sur */opt/riscv*. Pour que tout le monde accède au compilateur, il faut modifier les droits de */opt/riscv*. Accordez l'accès à tout le monde en tapant `chmod 777 /opt/riscv -R` sur un terminal.
Pour la suite OSS CAD et la chaîne d'outils RISC-V GCC, il faut effectuer une modification de la variable *path* du fichier */etc/environment*. Tapez `sudo nano /etc/environment` pour ouvrir le fichier et après `PATH=»` entrez la chaîne `/opt/oss-cad-suite/bin:/opt/riscv/bin:`, ensuite enregistrez le fichier.
La plupart des cartes FPGA gère l'interface de programmation avec une puce FT232H de FTDI. Pour qu'un utilisateur puisse y accéder sans droits root, il faut créer une règle *udev* appropriée pour la puce FTDI. Via le terminal, entrez `sudo nano /etc/udev/rules.d/53-lattice-ftdi.rules`. Un nouveau fichier s'ouvre en écriture ; il faut y entrer :

```
ACTION=="add", ATTR=="0403", ATTR=="6010",
   MODE:="666"
ACTION=="add", ATTR=="0403", ATTR=="6014",
   MODE:="666"
```

Puis enregistrez le fichier. Les préparatifs sont terminés et on peut commencer la synthèse puis le téléchargement de notre premier programme de test. Pour que tous les paramètres prennent effet, il faut redémarrer. Pour une utilisation avec VHDL et Verilog, il est préférable d'utiliser un éditeur avec coloration syntaxique.

NEORV32 pour le FPGA

À l'état hors tension, le FPGA n'est pas configuré. À la mise sous tension, le iCE40UP5K lit la description des connexions internes sur une flash SPI externe. Celle-ci doit donc d'abord être stockée dans la flash SPI de l'UPDuino V3.0 ou de l'iCEBreaker.
Ici, nous créons un projet type pour la carte UPduino V3.0 contenant le processeur et les périphériques. Le système créé devrait pouvoir fonctionner directement sur le FPGA.
Via un terminal, tapez `cd ~/neorv32/setups/osflow/` pour vous rendre dans le dossier d'exemples des outils *open source*. Pour démarrer la synthèse et donc la création du *bitstream* pour le FPGA, tapez `make BOARD=UPduino UP5KDemo`. C'est tout ! Mais il faut patienter, le processus de synthèse peut prendre du temps… Le dossier *~/neorv32/setups/osflow/* contient désormais un fichier nommé *neorv32_UPduino_v3_UP5KDemo.bit*. Le flux binaire interne du fichier décrit les interconnexions des blocs logiques de base à réaliser dans le FPGA. C'est ce flux binaire qu'il faut maintenant écrire dans la flash SPI de la carte FPGA.
Pour cela, nous pouvons relier la carte UPduino au PC par un câble USB. Dans le terminal, tapez `iceprog ~/neorv32/setups/osflow/neorv32_UPduino_v3_UP5KDemo.bit` pour lancer la programmation de la flash SPI externe, et la configuration est ensuite chargée dans le FPGA. Cela signifie que notre système RISC-V est maintenant configuré dans l'UPduino V3.0, et que nous pouvons lui confier des logiciels. Comme indiqué au début, 64 Ko de mémoire sont disponibles pour les applications et 64 Ko en RAM. Comme périphériques, nous avons trois

```
MEMORY
{
/* section base addresses and sizes have to be a multiple of 4 bytes */
/* ram section: first value of LENGTH => data memory used by bootloader (fixed!); second value of LENGTH => *physical* size of data memory */
/* adapt the right-most value to match the *total physical data memory size* of your setup */

  ram  (rwx) : ORIGIN = 0x80000000, LENGTH = DEFINED(make_bootloader) ? 512 : 8*1024

/* rom and iodev sections should NOT be modified by the user at all! */
/* rom section: first value of ORIGIN/LENGTH => bootloader ROM; second value of ORIGIN/LENGTH => maximum *logical* size of instruction memory */

  rom   (rx) : ORIGIN = DEFINED(make_bootloader) ? 0xFFFF0000 : 0x00000000, LENGTH = DEFINED(make_bootloader) ? 32K : 2048M
  iodev (rw) : ORIGIN = 0xFFFFFE00, LENGTH = 512

}
/* ************************************************************************** */
```

Figure 8. Modifications de configuration de la taille de la RAM.

```
user@user-VirtualBox:~/neorv32/sw/example/hello_world$ make exe
Memory utilization:
   text    data     bss     dec     hex filename
   5576       0     116    5692    163c main.elf
Executable (neorv32_exe.bin) size in bytes:
5588
user@user-VirtualBox:~/neorv32/sw/example/hello_world$
```

Figure 9. L'exécutable NEORV32_exe.bin est créé.

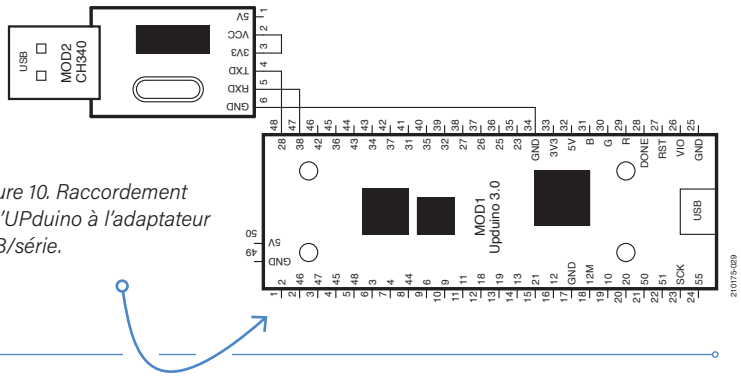


Figure 10. Raccordement de l'UPduino à l'adaptateur USB/série.

interfaces (SPI, I²C, UART), quatre entrées et quatre sorties, ainsi que trois sorties MLI. Le CPU synthétisé ici est un modèle RV32IMAC qui fonctionne à 18 MHz. Le SoC dans le FPGA a aussi un petit chargeur d'amorçage accessible via l'UART.

Hello World

Le FPGA est maintenant équipé du NEORV32 et la première démo *Hello World* peut être compilée et téléchargée sur le RISC-V. Le NEORV32 est configurable, la taille réelle de la RAM doit donc être définie dans le script du linker. Pour ce faire, entrez `nano ~/neorv32/sw/common/neorv32.ld` avec le terminal et à la ligne 62 :

```
ram (rwx) : ORIGIN = 0x80000000, LENGTH =
   DEFINED(make_bootloader) ? 512 : 8*1024
```

à la place de :

```
ram (rwx) : ORIGIN = 0x80000000, LENGTH =
   DEFINED(make_bootloader) ? 512 : 64*1024
```

(**fig. 8**)

Le compilateur RISC-V est maintenant prêt à l'emploi. Dans un terminal ouvert, vous pouvez aller dans le dossier du programme *Hello World* en tapant :

```
cd ~/neorv32/sw/example/hello_world
```

Pour produire un fichier exécutable pouvant être chargé dans le NEORV32, il suffit d'entrer `make exe`. Ce fichier s'appellera *neorv32_exe.bin* (**fig. 9**). Pour pouvoir exécuter NEORV32, il faut alors le charger sur la carte à l'aide du bootloader intégré. Il reçoit les données via l'UART (19200 bauds, 8 bits de données, 1 bit d'arrêt, sans parité ni contrôle de flux). Si une carte UPduino V3.0 est utilisée, un convertisseur USB-série externe est nécessaire, par ex. le convertisseur CH340 de la boutique d'Elektor. (Reportez-vous à l'encadré **Produits**.) Il doit être connecté comme indiqué à la **figure 10**.

HTerm est utilisé pour le téléchargement lui-même. On peut le lancer depuis un terminal avec `~/hterm/hterm` ; une fenêtre comme sur la **figure 11** devrait apparaître. Il faut sélectionner le convertisseur USB-série comme port, en général il s'appelle */dev/ttyUSB0* ; cela peut toutefois dépendre de la configuration matérielle et de l'adaptateur sélectionné.

Après raccordement de l'adaptateur série USB, l'UPduino peut être alimenté en tension et le message du bootloader doit apparaître (**fig. 12**). Si aucun caractère n'est envoyé à temps (en moins de 8 s) au bootloader, il tente de démarrer automatiquement à partir de la flash SPI qui ne contient encore aucun logiciel. Sinon, le bootloader passe en mode commande. Tapez `u` pour activer le téléchargement dans le bootloader et cliquez sur le bouton *Select File* dans HTerm. Comme on peut le voir sur la **figure 13**, il faut sélectionner le fichier *neorv32_exe.bin* dans le dossier *~/neorv32/sw/example/hello_world* puis le télécharger. Une fois tout terminé, le bootloader renvoie `OK` (cf. **fig. 14**) et il suffit d'entrer `e` pour exécuter le programme.

Le résultat apparaît à la **figure 15**. En fait, le programme a été stocké dans la « ROM » basée sur la RAM du NEORV32 et non dans la flash SPI. Cela prolonge la durée de vie de la flash SPI en y réduisant le nombre de cycles d'écriture. L'inconvénient c'est que le programme doit être rechargé à chaque redémarrage de NEORV32. Pour charger le programme automatiquement, il faut que le bootloader le mette

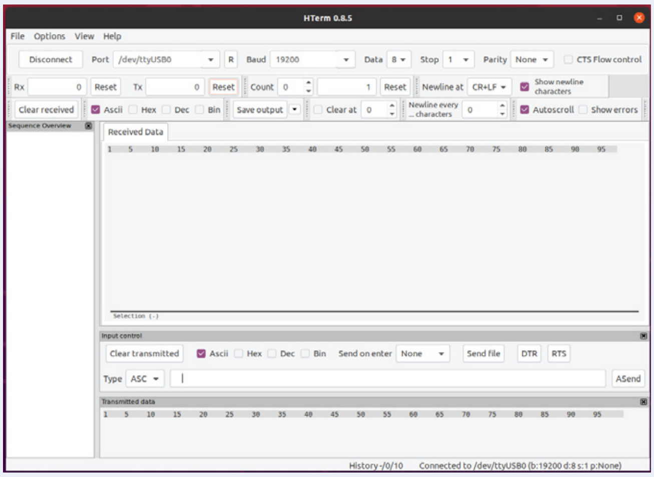


Figure 11. Aperçu d'une fenêtre HTerm.

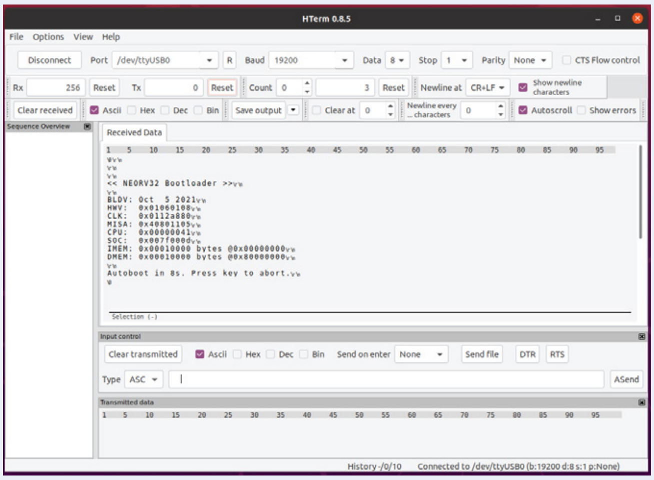


Figure 12. Chargeur de démarrage NEORV32.

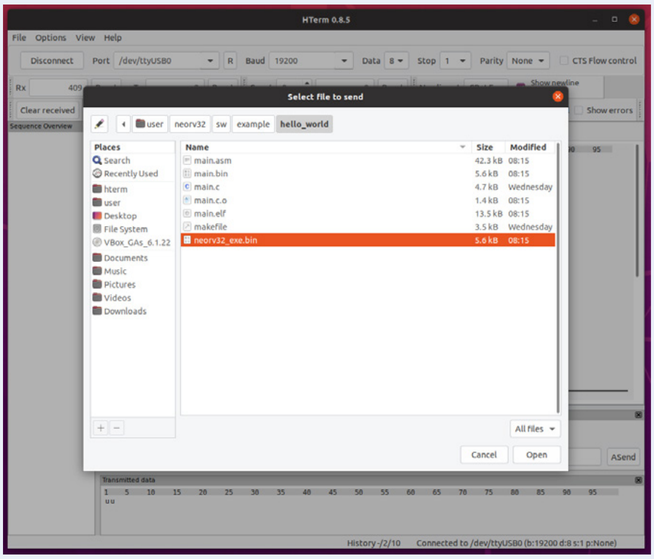


Figure 13. Fenêtre de téléchargement upload neorv32_exe.bin.

```
Available CMDs:\r\n
 h: Help\r\n
 r: Restart\r\n
 u: Upload\r\n
 s: Store to flash\r\n
 l: Load from flash\r\n
 e: Execute\r\n
CMD:> u\r\n
Awaiting neorv32 exe.bin... OK\r\n
CMD:>
```

Figure 14. Téléchargement réussi.

```
<< NEORV32 Bootloader >>\r\n
\r\n
BLDV: Oct  5 2021\r\n
HWV:  0x01060108\r\n
CLK:  0x0112a880\r\n
MISA: 0x40801105\r\n
CPU:  0x00000041\r\n
SOC:  0x007f000d\r\n
IMEM: 0x00010000 bytes @0x00000000\r\n
DMEM: 0x00010000 bytes @0x80000000\r\n
\r\n
Autoboot in 8s. Press key to abort.\r\n
Aborted.\r\n
\r\n
Available CMDs:\r\n
 h: Help\r\n
 r: Restart\r\n
 u: Upload\r\n
 s: Store to flash\r\n
 l: Load from flash\r\n
 e: Execute\r\n
CMD:> u\r\n
Awaiting neorv32 exe.bin... OK\r\n
CMD:> e\r\n
Booting...\r\n
\r\n
\r\n
                                                                                 ##                                  \r\n
                                                                                 ##         ##   ##   ##             \r\n
 ##     ##   #########   ########    ########   ##      ##   ########    ########    ##       ################           \r\n
####    ##  ##          ##      ##  ##      ##  ##      ##  ##      ##  ##      ##   ##     ####            ####         \r\n
## ##   ##  ##          ##      ##  ##      ##  ##      ##          ##         ##    ##       ##   ######   ##           \r\n
##  ##  ##  #########   ##      ##  #########   ##      ##      #####        ##      ##     ####   ######   ####         \r\n
##   ## ##  ##          ##      ##  ##    ##     ##    ##           ##     ##        ##       ##   ######   ##           \r\n
##    ####  ##          ##      ##  ##     ##     ##  ##    ##      ##   ##          ##     ####            ####         \r\n
##     ##    #########   ########   ##      ##      ##       ########   ##########   ##       ################           \r\n
                                                                                     ##         ##   ##   ##             \r\n
                                                                                     ##                                  \r\n

Hello world! :)\r\n
```

Figure 15. « Hello World » de NEORV32.

Convention de nomenclature RISC-V

RISC-V est un terme générique pour les diverses variantes de l'architecture. Notre collègue d'Elektor Stuart Cording a écrit un excellent article (« RISC-V quesaco », Elektor 7-8/2021, [2]) qui donne de plus amples détails. RISC-V décrit une architecture de jeu d'instructions (en anglais ISA = *Instruction Set Architecture*) qui classe les processeurs en versions 32, 64 ou 128 bits. Un processeur à 32 bits a un nom commençant par RV32 pour 32 bits, et donc RV64 indique un processeur à 64 bits. En outre, un certain nombre de lettres viennent s'ajouter pour indiquer les commandes et extensions que le processeur peut gérer. Ces lettres vont de A à Z ; elles sont explicitées dans la spécification RISC-V actuelle [3]. Les performances des processeurs sont fonction des commandes et extensions prises en charge.

dans la flash SPI. En plus de la démo « Hello World » de base, il y a d'autres exemples à découvrir, dont un FreeRTOS complet déjà présenté dans Elektor [15]. Cela vaut la peine d'aller voir le dossier *~/neorv32/sw/example*, où plusieurs autres exemples figurent dans des dossiers individuels.

Un nouvel environnement FPGA iCE40UP5K

L'utilisation de la carte iCEBreaker montrera comment adapter NEORV32 à d'autres cartes iCE40up5k. Les caractéristiques du SoC lui-même ne sont pas modifiées, seule l'affectation des broches sur le FPGA est adaptée de sorte qu'une carte iCEBreaker soit utilisable et qu'un bitstream adéquat soit produit.
Pour ce faire, il faut éditer le Makefile dans *~/neorv32/setup/osflow*. Le fichier s'ouvre avec l'éditeur de texte de votre choix et on ajoute la nouvelle carte comme cible. Pour la carte iCEBreaker, il faut écrire ce qui suit à la ligne 72 :

```
iCEBreaker:
$(MAKE) \
BITSTREAM=neorv32_$(BOARD)_$(DESIGN).bit \
NEORV32_MEM_SRC="devices/ice40/neorv32_imem.ice40up_
   spram.vhd devices/ice40/neorv32_dmem.ice40up_
   spram.vhd" \
run
```

Cela garantit que la carte est référencée dans le Makefile primaire. Dans *~/neorv32/setup/osflow/boards* il faut aussi un fichier *iCEBreaker.mk* avec le contenu suivant :

```
.PHONY: all

all: bit
echo "! Built $(IMPL) for $(BOARD)"
```

Les Makefiles sont alors prêts, mais il manque encore deux fichiers VHDL : *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* et *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd* qui doivent être déposés sur *~/neorv32/setup/osflow/board_tops/*. Comme leur contenu est quasi identique à celui des fichiers *neorv32_UPduino_BoardTop_UP5KDemo.vhd* et *neorv32_UPduino_BoardTop_MinimalBoot.vhd*, copions d'abord ceux-ci en les renommant. Par ex. utilisons un terminal et entrons :

```
cp ~/neorv32/setups/osflow/board_tops/neorv32_
   UPduino_BoardTop_MinimalBoot.vhd
~/neorv32/setups/osflow/board_tops/neorv32_
   iCEBreaker_BoardTop_MinimalBoot.vhd
```

et :

```
cp ~/neorv32/setups/osflow/board_tops/neorv32_
   UPduino_BoardTop_UP5KDemo.vhd
~/neorv32/setups/osflow/board_tops/neorv32_
   iCEBreaker_BoardTop_UP5KDemo.vhd
```

Il faut apporter quelques modifications aux deux fichiers *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* et *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd*. Dans le fichier *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd*, changez la ligne 42 en `entity neorv32_iCEBreaker_BoardTop_UP5KDemo is` et la ligne 68 en `architecture neorv32_iCEBreaker_BoardTop_UP5KDemo_rtl of neorv32_iCEBreaker_BoardTop_UP5KDemo is`. Dans le fichier *neorv32_iCEBreaker_BoardTop_MinimalBoot.vhd*, changez la ligne 42 en `entity neorv32_iCEBreaker_BoardTop_MinimalBoot is` et la ligne 54 en `architecture neorv32_iCEBreaker_BoardTop_MinimalBoot_rtl of neorv32_iCEBreaker_BoardTop_MinimalBoot is`.

En dernière étape, placez le fichier de contraintes *iCEBreaker.pcf* dans *~/neorv32/setups/osflow/constraints/*. Le **listage 1** donne le contenu de ce fichier. *ICEBreaker.pcf* affecte les fonctions à diverses broches du FPGA. Il englobe aussi directement le convertisseur USB/série présent sur la carte iCEBreaker, et achemine les boutons et les LED sur les broches d'E/S de NEORV32. Une chose manque : un bouton reset. Bien qu'il soit défini dans le fichier de contraintes, sa fonction n'est pas référencée ici.
Une fois les changements nécessaires effectués, il ne reste qu'à produire le bitstream comme avec la carte UPduino. Pour cela, tapez `cd ~/neorv32/setups/osflow` sur un terminal pour atteindre le dossier *osflow* ; ensuite, `make BOARD=iCEBeaker UP5KDemo` démarre le processus qui produit les bitstreams ; pour les télécharger vers l'iCEBreaker (comme avec l'UPduino), nous utilisons `iceprog ~/neorv32/setups/osflow/neorv32_iCEBreaker_UP5KDemo.bit`.
Le chargement du logiciel dans le NEORV32 est là aussi pris en charge par le bootloader intégré. La carte iCEBreaker évite le recours à un convertisseur USB/série externe, car nous utilisons la 2e voie du convertisseur intégré sur celle-ci.

Bouton de réinitialisation pour le NEORV32

Pour redémarrer le NEORV32, il faut couper brièvement l'alimentation électrique, puis la rallumer. Cela devient fastidieux à la longue, et l'iCEBreaker ayant assez de boutons, on peut utiliser l'un d'eux pour le réinitialiser (*reset*). Par ex., le *uButton* près de la prise micro-USB de la carte ; il est connecté à la broche 10 du FPGA.
Le NEORV32 possède une entrée interne *rstn_i* (reset) reliée à la sortie *lock* de la PLL (*Phase Lock Loop* = boucle à verrouillage de phase) de l'horloge du système. Si la PLL se déverrouille, elle envoie un reset au

Listage 1. iCEBreaker.pcf

```
#UART (uart0)
ldc_set_location -site {9} [get_ports uart_txd_o]
ldc_set_location -site {6} [get_ports uart_rxd_i]

#SPI - on-board flash
ldc_set_location -site {14} [get_ports flash_sdo_o]
ldc_set_location -site {15} [get_ports flash_sck_o]
ldc_set_location -site {16} [get_ports flash_csn_o]
ldc_set_location -site {17} [get_ports flash_sdi_i]

#SPI - user port
ldc_set_location -site {43} [get_ports spi_sdo_o]
ldc_set_location -site {38} [get_ports spi_sck_o]
ldc_set_location -site {34} [get_ports spi_csn_o]
ldc_set_location -site {31} [get_ports spi_sdi_i]

#TWI
ldc_set_location -site {2} [get_ports twi_sda_io]
ldc_set_location -site {4} [get_ports twi_scl_io]

#GPIO - input
ldc_set_location -site {18} [get_ports {gpio_i[0]}]
ldc_set_location -site {19} [get_ports {gpio_i[1]}]
ldc_set_location -site {20} [get_ports {gpio_i[2]}]
ldc_set_location -site {28} [get_ports {gpio_i[3]}]

#GPIO - output
ldc_set_location -site {25} [get_ports {gpio_o[0]}]
ldc_set_location -site {26} [get_ports {gpio_o[1]}]
ldc_set_location -site {27} [get_ports {gpio_o[2]}]
ldc_set_location -site {23} [get_ports {gpio_o[3]}]

#RGB power LED
ldc_set_location -site {39} [get_ports {pwm_o[0]}]
ldc_set_location -site {40} [get_ports {pwm_o[1]}]
ldc_set_location -site {41} [get_ports {pwm_o[2]}]

#Reset
ldc_set_location -site {10} [get_ports
   {user_reset_btn}]
```

NEORV32. La solution la plus propre serait d'envoyer le signal de réinitialisation de l'utilisateur au même point que le signal *lock* de la PLL de sorte le NEORV32 reçoive le reset de l'un ou de l'autre. La solution la plus simple et la plus rapide est d'utiliser l'entrée reset de la PLL. La connexion du uButton, *user_reset_btn* avec cette entrée est sur la **figure 16**. Dès lors, si on appuie sur uButton, la PLL est réinitialisée, le signal *lock_o* passe au niveau bas et le NEORV32 est réinitialisé. Afin d'incorporer cette modification dans le projet UP5K-demo, il faut insérer une ligne puis en modifier une autre dans le fichier *neorv32_iCEBreaker_BoardTop_UP5KDemo.vhd* se trouvant dans le dossier *~/neorv32/setups/board_tops*. Après la ligne 44, insérez la nouvelle ligne `user_reset_btn : in std_ulogic;`. Modifiez ensuite la ligne 130 : `RESETB => user_reset_btn,`. Attention, une erreur de syntaxe se produira si vous oubliez la virgule à la fin. Le uButton est maintenant configuré pour fonctionner comme un reset. Enfin, pour que les changements prennent effet, produisez un nouveau bitstream dans l'UP5K-Demo et chargez-le dans le FPGA iCEBreaker.

Et maintenant !

Il faudrait consacrer un livre entier à chacun de ces sujets : RISC-V, FPGA et NEORV32. Pour les débutants, l'iCE40UP5K est peu onéreux, mais la puce est bien souvent en rupture d'approvisionnement. Les deux cartes référencées dans l'article ne sont pas les seules à accueillir ce FPGA. Il en existe plusieurs autres : cela va du HAT pour Raspberry Pi à la console portable complète. Les différents outils et projets applicables à ce FPGA méritent également d'être étudiés. Citons par ex.

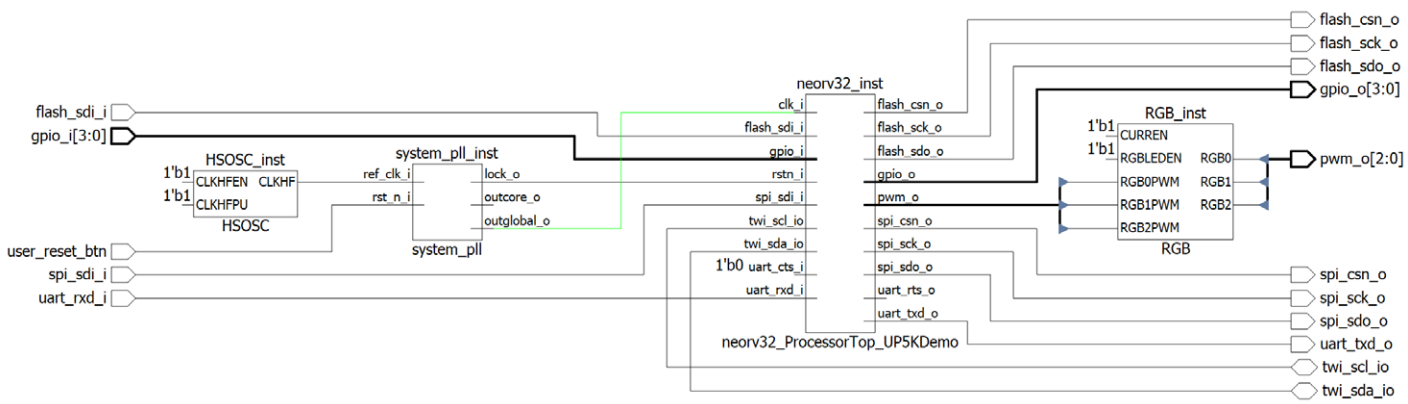


Figure 16. Raccordement du bouton de réinitialisation uButton à l'entrée rst de la PLL.

LiteX [16] qui permet d'assembler votre propre SoC comme dans un kit et n'est pas nécessairement limité à RISC-V, RISCboy [17] de Luke Wren qui fournit un système de type Gameboy dans le FPGA, et la console de jeu OK-ice40-PRO [18].
Si les difficultés d'approvisionnement disparaissent, d'autres projets et idées pour l'iCE40UP5K naîtront à coup sûr. Une petite console de jeux conçue à partir d'un iCE40UP5K et d'un Raspberry Pi RP2040 semble déjà être en préparation [19].

210175-04

Contributeurs

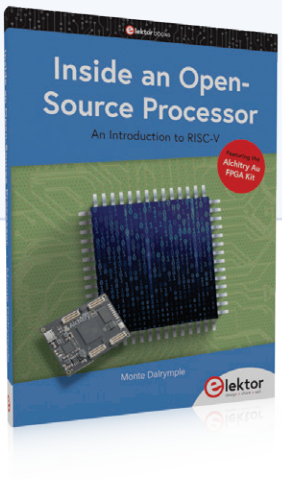
Texte et images : **Mathias Claußen**
Rédaction : **Jens Nickel**
Traduction : **Yves Georges**
Mise en page : **Harmen Heida**

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (mathias.claussen@elektor.com) ou contactez Elektor (redaction@elektor.fr).

PRODUITS

- Carte de développement FPGA Alchitry Cu (Lattice iCE40 HX)
  www.elektor.fr/19640
- Convertisseur USB-TTL CH340, module UART CH340G (3,3 V/5,5 V)
  www.elektor.fr/19151
- Livre en anglais, « Inside an Open-Source Processor », M. Dalrymple (Elektor 2021)
  www.elektor.fr/19826



LIENS

[1] M. Ossmann, « projet SCCC (1) », Elektor 3-4/2019 : www.elektormagazine.fr/180394-04
[2] S. Cording, « RISC-V : quesaco ? », Elektor 7-8/2021 : www.elektormagazine.fr/210223-04
[3] Spécification du RISC-V : http://riscv.org/technical/specifications/
[4] Dépôt GitHub de NEORV32 : http://github.com/stnolting/neorv32/
[5] Page du produit Lattice iCE40UltraPlus : www.latticesemi.com/en/Products/FPGAandCPLD/iCE40UltraPlus
[6] Dépôt GitHub d'iCEBreaker : http://github.com/icebreaker-fpga/icebreaker
[7] Dépôt GitHub de l'UPduino : http://github.com/tinyvision-ai-inc/UPduino-v3.0
[8] Lattice Radiant : www.latticesemi.com/LatticeRadiant
[9] YosysHQ oss-cad-suite-build : http://github.com/YosysHQ/oss-cad-suite-build
[10] Configurer une chaîne d'outils pour le Kendryte K210, Elektor Labs : www.elektormagazine.com/labs/setup-a-toolchain-for-the-kendryte-k210-1
[11] Snickerdoodles au goût de Linux avec Zynq, ElektorTV : www.youtube.com/watch?v=EE4yYZ-FEoQ
[12] YosysHQ oss-cad-suite-build Releases : http://github.com/YosysHQ/oss-cad-suite-build/releases
[13] HTerm : www.der-hammer.info/
[14] NEORV32 - Construire la chaîne d'outils à partir de zéro : http://stnolting.github.io/neorv32/ug/#_building_the_toolchain_from_scratch
[15] W. Gay, « multitâche en pratique avec l'ESP32 », Elektor 1-2/2020 : www.elektormagazine.fr/190182-03
[16] LiteX : http://github.com/enjoy-digital/litex
[17] RISCBoy : http://github.com/Wren6991/RISCBoy
[18] OK-iCE40Pro Handheld : http://github.com/WiFiBoy/OK-iCE40Pro
[19] PicoStation3D : http://github.com/Wren6991/PicoStation3D
[20] Nolting S., The NEORV32 RISC-V-Processor, dépôt GitHub 2020 : http://raw.githubusercontent.com/stnolting/neorv32/master/docs/figures/neorv32_processor.png