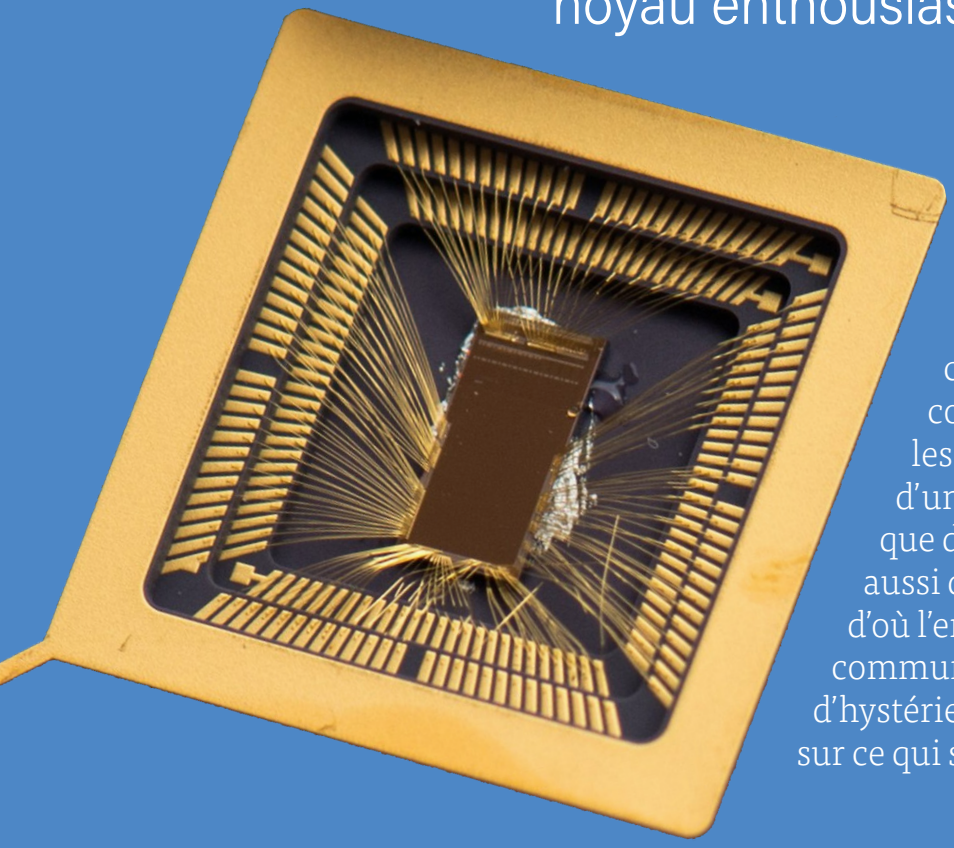
RISC-V : quesaco ?

Pourquoi une nouvelle architecture de noyau enthousiasme-t-elle l'industrie ?



Stuart Cording (Elektor)

Le monde électronique semble être devenu fou de *RISC-V*. Pourquoi ? Qu'est-ce ? Et comment pouvez-vous contribuer ? Si vous parcourez les brèves, vous savez qu'il s'agit d'une architecture de processeur et que des puces l'utilisent. Vous savez aussi que RISC-V est *libre et ouvert*, d'où l'engouement suscité et l'énorme communauté de fans. Au-delà du parfum d'hystérie de ces réactions, penchons-nous sur ce qui se cache derrière ce sigle.

Il faut savoir que RISC-V définit une architecture de jeu d'instructions, ou *ISA* [1], et non un processeur. Cela signifie que les concepteurs du RISC-V décrivent comment faire pour concevoir le fonctionnement d'un processeur qui serait basé sur leur ISA. Par *concevoir*, nous voulons vraiment dire création du processeur avec tous ses registres, accumulateurs, opérations mathématiques, bus mémoire et tout le reste.

L'ISA (*Instruction Set Architecture*) documente par ex. les opérations prises en charge, les capacités d'adressage de la mémoire, le fonctionnement de la pile et le déroulement des interruptions, etc. Pour les opérations prises en charge, elle définit combien de bits sont utilisés pour coder l'instruction et quels bits sont utilisés pour coder la source de tout opérande nécessaire.

La raison de cet engouement est que l'ISA est libre et ouvert. Ouvert signifie que quiconque peut aider à son développement, et libre implique une utilisation libre de tout droit. Toutefois, le design ouvert et libre des cartes *Arduino* ne signifie pas que celles-ci sont gratuites et il en ira de même pour la réalisation du processeur RISC-V de vos rêves.

Qui RISC-V concurrence-t-il ?

Chaque processeur possède une ISA qui appartient à son concepteur et peut faire l'objet d'une licence. *Microchip* produit des appareils basés sur des processeurs *PIC* à 8 et 16 bits et, quelque part, il existe une ISA pour les décrire. Ce sont des noyaux appartenant à Microchip et qui sont vendus avec ses microcontrôleurs. Si vous voulez développer votre propre microcontrôleur, vous opterez sans doute pour *Arm* et *MIPS*. Ces noyaux sont de la propriété intellectuelle (*PI*) qui peut être acquise sous licence. Les entreprises qui les ont conçus ont œuvré pour convertir l'ISA en un processeur matériel, développé des outils de prise en charge, créé les infrastructures connexes et vous font payer pour utiliser le tout. Un problème se pose si ces options ne répondent pas *tout à fait* à vos attentes.

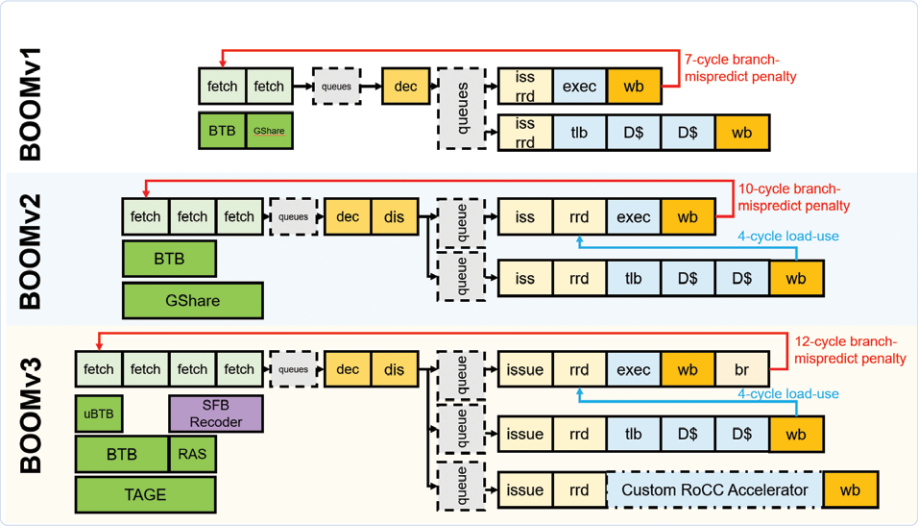


Figure 1. Processus de développement utilisé par le projet BOOM [27] pour implémenter le pipeline RISC-V. (Source : Regents of the University of California)

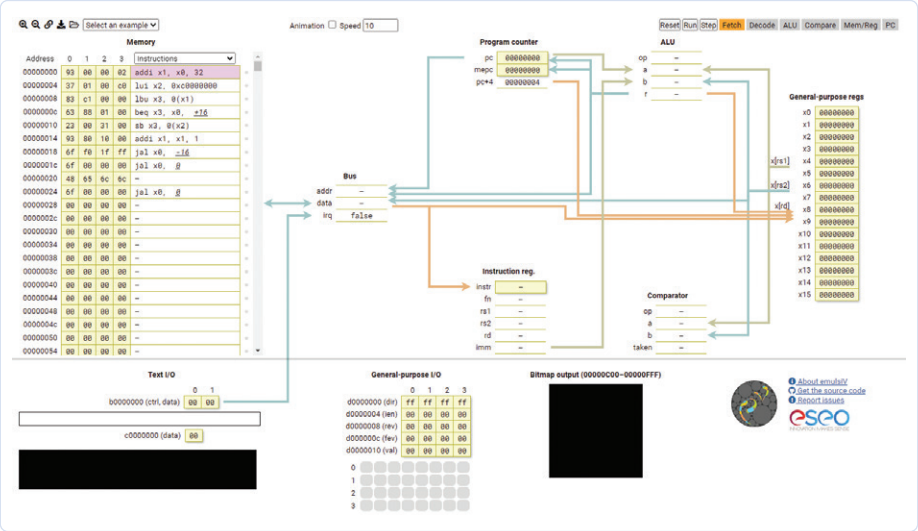


Figure 2. Le simulateur emulsiV permet à tous d'essayer le RISC-V dans un navigateur web.

Par ex., votre application doit exécuter très rapidement du chiffrement, mais en consommant le moins d'énergie possible ; un possible processeur sous licence peut exécuter votre tâche en cent instructions. Pour réduire l'énergie nécessaire, il faut trouver un fondeur de puces (*fab*) spécialisé dans la faible consommation d'énergie. Par rapport à un processus standard de fabrication, les coûts peuvent alors s'envoler et votre super-produit sera trop cher pour votre marché cible. Certains de vos ingénieurs sont peut-être capables d'optimiser le temps d'exécution du code en créant de nouvelles instructions pour le processeur, mais comme l'ISA ne vous appartient pas, vous ne pouvez pas la modifier. Vous avez donc un problème de performance du processeur qu'il faut résoudre par une approche de fabrication. Nous y reviendrons plus tard…

Que fait RISC-V tel quel ?

En deux mots : « peu, mais assez ». Fondamentalement, il faut d'abord choisir l'architecture précise que vous souhaitez. Des ISA à 32 et 64 bits sont définies, et une ISA à 128 bits est également en cours de développement. Ces définitions de base sont baptisées RV32I et RV64I. Avec la RV32I, vous aurez 49 instructions [2] à votre disposition. Le « I » signifie ici *Integer* (entier). Ce sont les instructions arithmétiques et logiques de base sur les entiers (ADD, SUB, AND, OR, XOR), les décalages, comparaisons, sauts et liaisons, ainsi que certaines instructions système [3]. Si vous recherchez un code compact, l'option 'C' peut vous intéresser. Les instructions sont codées sur 16 bits, façon mode *Thumb* d'Arm. On peut y ajouter des instructions multiplication et division (M), atomiques (A) et à virgule flottante (F, D et Q).

Ensuite, il faut concevoir le cœur du processeur d'après les caractéristiques des options choisies dans un *langage de description du matériel* [4] (*Hardware Description Language = HDL*), tel que VHDL ou Verilog. Comme cette étape est difficile, c'est là que la communauté intervient. Concevoir des processeurs exige de multiples compétences, c'est pourquoi nombre de spécialistes et d'entreprises proposent des designs prêts à l'emploi. Pour emprunter la voie *libre*, ne cherchez pas plus loin que la plateforme *PULP* [5], créée par l'*ETH Zurich* et l'*Università di Bologna*. Si vous souhaitez voir comment ce genre de chose est réalisé, *GitHub* présente l'implémentation CV32E40P RV32IM[F]C [6], et le décodeur d'instructions [7]. Le projet *BOOM* est une autre implémentation : un noyau paramétrable à haute performance pour la recherche en architecture, développé à Berkeley par l'*Université de Californie* (**fig. 1**).

Si vous êtes pressé et avez besoin d'assistance, il faudra payer pour obtenir une licence pour l'implémentation d'un tiers comme *SiFive* [8]. Ils disposent d'une gamme de designs à 32 et 64 bits [9] qui peuvent être personnalisés.

Comment essayer RISC-V ?

Bien que RISC-V existe depuis quelque temps, peu de puces sont disponibles pour le tester. Industriellement parlant, RISC-V est encore assez nouveau. Si les microcontrôleurs vous intéressent, vous savez que la plupart des acteurs du secteur ont adopté Arm, abandonnant leurs propres noyaux. C'est un investissement stratégique à long terme. À part l'économie des redevances versées à Arm, passer à RISC-V aurait peu d'avantages pour les utilisateurs. Il faudrait aussi que le service R&D pense RISC-V, l'intègre à toutes

ses autres *PI* (analogiques, temporisateurs, bus, interfaces, mémoires), mette à jour l'*EDI* de développement, le compilateur, le débogueur, etc.

Ceux qui ont un disque dur *Seagate* ou *Western Digital*, utilisent peut-être déjà RISC-V [10] [11]. Mais au-delà de posséder un produit qui l'utilise, vous voulez sans doute exécuter du code sur ce noyau. Le moyen le plus rapide d'y parvenir est un simulateur tel qu'*emulsiV* d'*ESEO* [12], basé sur leur implémentation du noyau RISC-V *Virgule* (**fig. 2**). En plus du processeur, le simulateur offre des E/S de texte, une sortie bitmap et des E/S à usage général (GPIO). Sept exemples couvrent les bases, de l'ajout/sortie de texte ASCII au contrôle des GPIO. L'intéressante option *animation* (case à cocher centrale du haut) montre d'où viennent et où vont les données au fur et à mesure de l'exécution du code. Vous pouvez essayer le code du **listage 1** (le copier dans un éditeur de texte, enregistrer le fichier sous `program.hex` et le télécharger dans le simulateur).

Pour découvrir le RISC-V au format Arduino, le *HiFive1 Rev B* est disponible sur *CrowdSupply* [13]. Il utilise le microcontrôleur SiFive *FE310-G002* [14]. C'est un dispositif élémentaire : périphériques numériques seulement (I2C, UART, SPI, PWM, GPIO), un peu de SRAM, appuyée par une flash QSPI hors puce pour le stockage non-volatile. La carte comprend un module Wi-Fi et Bluetooth ainsi qu'un *J-Link* de *Segger* pour le débogage USB.

De l'autre côté du spectre des performances, il y a le *SoC* PolarFire [15] de Microchip avec quatre cœurs RISC-V à 64 bits secondés par un *FPGA*. C'est une plateforme configurable à loisir qui peut exécuter *Linux* tout en prenant en charge des applications en temps réel.

Listage 1. Code hexa brut pour simulateur emulsiV à sauvegarder et télécharger comme program.hex.

```
:10000000930000023701 00c083c100006388010033
:10001000230031009380 10006ff01fff6f0000007d
:1000200048656c6c6f20456c656b746f72210000c5
:00000001FF
```

Listage 2. Code C pour implémenter le chiffrement ChaCha.

```
unsigned int processLine(unsigned int res, unsigned int word) {
  res = qr1_c(res, word);
  res = qr2_c(res, word);
  res = qr3_c(res, word);
  res = qr4_c(res, word);
  res = qr1_c(res, word);
  res = qr2_c(res, word);
  res = qr3_c(res, word);
  res = qr4_c(res, word);
  return res;
}
```

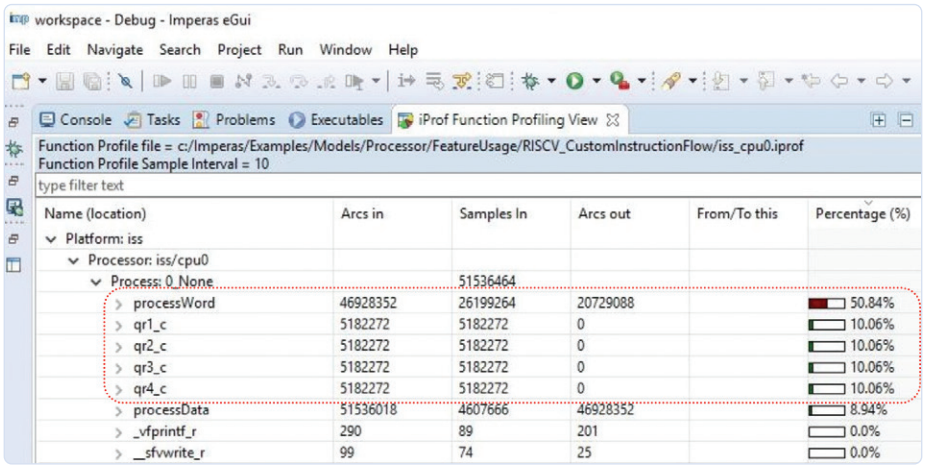


Figure 3. Le chiffrement ChaCha occupe 55 % du temps processeur avec le code compilé en C standard. (Source : Imperas Software Limited)

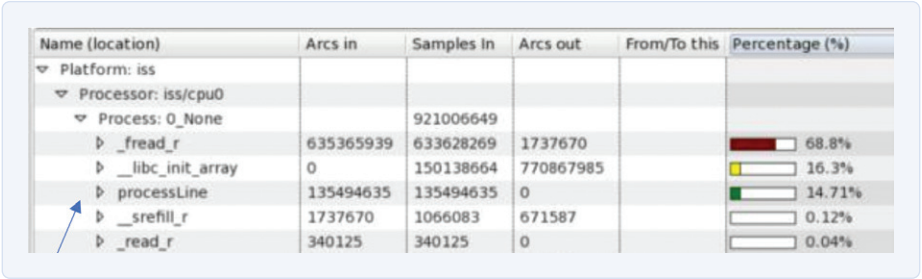
Comment personnaliser mon RISC-V ?

Nous mentionnions plus haut que l'avantage de RISC-V était de pouvoir adapter le jeu d'instructions aux besoins de chaque application. C.-à-d. que si l'implémentation de processeur choisie répond à 95 % des besoins, on peut y ajouter des fonctions astucieuses pour couvrir les 5 % manquants. Supposons par ex. que l'application fasse un usage intensif du chiffrement de flux *ChaCha* [16], tel que décrit dans une note d'application d'*Imperas* [18], un autre acteur RISC-V qui fournit des outils de vérification, d'analyse et de profilage.

Cependant, vous observez que votre implémentation ChaCha sur un noyau RISC-V nécessite un temps de traitement élevé. Alors vous voudrez réduire le temps d'exécution, mais aussi profiter de la baisse de consommation résultant de cette amélioration, par ex. en entrant dans un mode de veille à faible énergie.

Le code (**listage 2**) fait un usage intensif des instructions *XOR* et *rotate* dans une étape dite des *quarter rounds* pour laquelle quatre fonctions C `qrX_c()` ont été écrites. La fonction `processLine()` les appelle pour effectuer le chiffrement. L'analyse des temps d'exécution montre que cette tâche occupe 55 % du temps du processeur, dont environ 32 % sont répartis sur les fonctions *quarter round* (**fig. 3**).

Avec RISC-V, il suffit d'implémenter quatre instructions *quarter round* spécifiques qui s'exécutent en un seul cycle au lieu de laisser le code produit par le compilateur C s'en charger. Dans l'ISA, il existe en effet une section réservée aux instructions personnalisées. Au départ, les instructions peuvent être ajoutées à une conception



Name (location)	Arcs in	Samples in	Arcs out	From/To this	Percentage (%)
Platform: iss					
Processor: iss/cpu0					
Process: 0_None		921006649			
_fread_r	635365939	633628269	1737670		68.8%
__libc_init_array	0	150138664	770867985		16.3%
processLine	135494635	135494635	0		14.71%
__srefill_r	1737670	1066083	671587		0.12%
_read_r	340125	340125	0		0.04%

Figure 4. Avec les dernières instructions spécialisées développées, la charge processeur du chiffrement ChaCha tombe à moins de 15 %. (Source : Imperas Software Limited)

RISC-V et implémentées sous forme codée en C. Cela permet de simuler les nouvelles instructions afin de les tester et vérifier si une amélioration des performances est réalisable. Ici, avec des instructions spécialisées *quarter round* disponibles sur le cœur RISC-V personnalisé, la fonction `processLine()` mobilise désormais moins de 15 % du potentiel du processeur (**fig. 4**) . Si c'est considéré comme correct, l'équipe de développement peut alors réaliser l'implémentation matérielle des instructions en Verilog.

Hélas, l'utilisation des instructions nouvelles ne se borne pas à recompiler le code C (**listage 3**). Modifier un compilateur RISC-V pour utiliser de nouvelles instructions représente un effort énorme. À la place, les instructions codées en hexa sont appelées par l'assembleur en ligne comme pour un code optimisé à la main.

Comment contribuer à RISC-V ?

Si contribuer au développement permanent de RISC-V vous tente, c'est le moment. RISC-V International [19] est l'organisation chargée de développer et promouvoir l'idée RISC-V (**fig. 5**). Chacun peut adhérer comme membre de la communauté [20] ou y faire carrière, de nombreuses entreprises et universités [21] y participent activement.

Si vous pensez voir arriver sur le marché de nombreux microcontrôleurs RISC-V, vous risquez d'être déçu. *GigaDevice* en propose quelques-uns [22] et un fournisseur russe un autre pour le marché des compteurs intelligents [23]. Arm est trop ancré auprès des grands acteurs, et les jeunes pousses auront du mal à lutter sur ce marché saturé, même avec l'avantage financier d'un processeur sans royalties à débourser.

RISC-V sera plus probablement mis à profit dans des niches où sa capacité à personnaliser le noyau apporte de gros avantages, tels qu'une ultra-faible consommation [24]. Compte tenu des obstacles à l'octroi de licences technologiques à la Chine, RISC-V est une alternative tentante d'acquisition de PI américaine. *Alibaba* a annoncé un RISC-V à 64 bits à 16 cœurs, 2 GHz, en gravure 12 nm [25] et indique que le noyau est destiné à une infrastructure de serveur. Enfin, la *European Processor Initiative* [26] s'est penchée sur les architectures hétérogènes dans lesquelles Arm et RISC-V (ou d'autres

Figure 5. Logo officiel de RISC-V International qui promeut et soutient le développement de l'ISA.

Listage 3. Utilisation des instructions du RISC-V.

Pour utiliser les instructions RISC-V, il faut un assembleur en ligne. Les instructions sont codées en hexa car l'assembleur ne connaît pas le nom de ces nouvelles instructions.

```
unsigned int processLine(unsigned int res, unsigned int word) {
  asm __volatile__("mv x10, %0" :: "r"(res));
  asm __volatile__("mv x11, %0" :: "r"(word));
  asm __volatile__(".word 0x00B5050B\n" ::: "x10"); // equivalent to qr1_c()
  asm __volatile__(".word 0x00B5150B\n" ::: "x10"); // equivalent to qr2_c()
  asm __volatile__(".word 0x00B5250B\n" ::: "x10"); // equivalent to qr3_c()
  asm __volatile__(".word 0x00B5350B\n" ::: "x10"); // equivalent to qr4_c()
  asm __volatile__(".word 0x00B5050B\n" ::: "x10"); // equivalent to qr1_c()
  asm __volatile__(".word 0x00B5150B\n" ::: "x10"); // equivalent to qr2_c()
  asm __volatile__(".word 0x00B5250B\n" ::: "x10"); // equivalent to qr3_c()
  asm __volatile__(".word 0x00B5350B\n" ::: "x10"); // equivalent to qr4_c()
  return res;
}
```

noyaux) pourraient cohabiter. Le but est d'obtenir un effet de synergie en exploitant le meilleur processeur pour chaque tâche dans les designs multicœurs.
RISC-V n'est pas la première tentative de PI libre et ouverte pour les processeurs, mais c'est la plus aboutie à ce jour. Considérant son héritage, ses qualités (souplesse, approche ouverte), l'intérêt qu'il suscite dans les universités et le soutien important de l'industrie, RISC-V accompagnera la carrière d'ingénieurs durant une, voire deux générations.

(210223-04)

Des questions, des commentaires ?
Envoyez un courriel à l'auteur (stuart.cording@elektor.com) ou contactez Elektor (redaction@elektor.fr).

Contributeurs
Texte et illustrations : **Stuart Cording**
Rédaction : **Jens Nickel, C. J. Abate**
Traduction : **Yves Georges**
Mise en page : **Giel Dols**

PRODUITS

- **Carte de développement RED-V Thing Plus de SparkFun – SoC SiFive RISC-V FE310**
  www.elektor.fr/sparkfun-red-v-thing-plus-sifive-risc-v-fe310-soc
- **Carte de développement RISC-V TTGO T-Display-GD32 de LilyGo**
  www.elektor.fr/lilygo-ttgo-t-display-gd32-risc-v-development-board

LIENS

[1] **Architecture des jeux d'instructions et microarchitecture, GeeksforGeeks, octobre 2019** : http://bit.ly/3bBKAy2
[2] **RISC-V, Wikipédia** : https://fr.wikipedia.org/wiki/RISC-V
[3] **J. Zhu, « RISC-V Reference »** : http://bit.ly/30IlCXj
[4] **Langage de description du matériel, Wikipédia** : https://fr.wikipedia.org/wiki/Langage_de_description_de_mat%C3%A9riel#:
[5] **Site de la plate-forme PULP** : https://pulp-platform.org/
[6] **Dépôt GitHub – CV32E40P RISC-V core** : https://github.com/openhwgroup/cv32e40p
[7] **Implémentation SystemVerilog du décodeur d'instructions CV32E40P** : https://github.com/openhwgroup/cv32e40p/blob/master/rtl/cv32e40p_id_stage.sv
[8] **Site de SiFive** : www.sifive.com
[9] **Core Designer de SiFive** : http://bit.ly/3rKnkU7
[10] **Page RISC-V de Seagate** : http://bit.ly/3etS0oU
[11] **Page RISC-V de Western Digital** : http://bit.ly/3eyldyP
[12] **Simulateur emulsiV pour processeur RISC-V Virgule** : http://bit.ly/30AzqmG
[13] **Page CrowdSupply pour la carte Arduino HiFive1 Rev B** : http://bit.ly/3eww7Fj
[14] **Fiche technique du microcontrôleur SiFive FE310-G002** : https://bit.ly/3bFANak
[15] **Page produit PolarFire SoC** : http://bit.ly/3bFAU5K
[16] **Variante ChaCha [du chiffrement de flux Salsa20], Wikipédia** : http://bit.ly/3rHfZES
[17] **Chiffrement de flux, Wikipédia** : https://fr.wikipedia.org/wiki/Chiffrement_de_flux
[18] **« Imperas RISCV Custom Instruction Flow Application Note », Imperas Software Limited, octobre 2019** : http://bit.ly/3vguDVK
[19] **À propos de RISC-V** : https://riscv.org/about/
[20] **Adhésion à RISC-V** : https://riscv.org/membership/
[21] **Membres RISC-V** : https://riscv.org/members/
[22] **Microcontrôleurs RISC-V GigaDevice GD32** : http://bit.ly/2NbSgO9
[23] **Сергей, microcontrôleur russe K1986BK025 basé sur le noyau processeur RISC-V pour les compteurs électriques intelligents, Habr, décembre 2020** : http://bit.ly/3qGPY7o
[24] **Site de MicroMagic** : www.micromagic.com
[25] **J. Aufranc, « Infos détaillées sur le Cœur 64 bits RISC-V XT910 d'Alibaba », CNX Software, août 2020** : http://bit.ly/3eync6f
[26] **Site Initiative Européenne sur les Processeurs** : www.european-processor-initiative.eu/
[27] **Dépôt GitHub –cCœur BOOM RISC-V** : https://github.com/riscv-boom/riscv-boom