DVI sur le RP2040

Entretien avec Luke Wren, développeur de composants chez Raspberry Pi

Mathias Claußen (Labo d'Elektor)

Luke Wren, l'un des ingénieurs du Raspberry Pi RP2040, a réalisé des choses étonnantes en le poussant dans ses retranchements. Mathias Claussen, ingénieur chez Elektor, a voulu en savoir plus, notamment sur les astuces et les bidouillages nécessaires pour obtenir une sortie vidéo à partir d'un appareil aussi minuscule.

Le Raspberry Pi RP2040 est un microcontrôleur à moins d'un euro doté de capacités étonnantes. L'un de ses développeurs est Luke Wren, qui a non seulement travaillé sur le Raspberry Pi RP2040, mais a également montré que le micro peut « faire du DVI ». (Reportez-vous à l'article « Sortie vidéo sur les microcontrôleurs (2) » [1]). Lorsqu'il ne travaille pas sur des projets secrets pour Raspberry Pi, il partage certains de ses projets personnels avec la communauté sur Twitter (@wren6991) et le code sur GitHub [2].

Mathias Claussen : pouvez-vous nous parler un peu de vous ?
Luke Wren : je commence par la question la plus difficile ! Je suis ingénieur chez Raspberry Pi, et quand je ne fais pas ça, je travaille généralement sur mes projets de loisirs, je joue mal de la guitare, ou, plus récemment, je consacre du temps à l'apprentissage des langues. Je vis à Cambridge, au Royaume-Uni, non pas parce que c'est une ville particulièrement excitante, mais plutôt par inertie après avoir obtenu mon diplôme universitaire. J'ai vécu en Allemagne quand j'étais plus jeune, mais mon allemand est assez rouillé, donc je suis content que nous fassions cela en anglais.

Mathias : depuis combien de temps travaillez-vous chez Raspberry Pi ?
Luke : je suis entré en tant qu'employé en septembre 2018, mais j'ai fait un stage chez Raspberry Pi auparavant.

Mathias : quel a été votre rôle dans l'élaboration du RP2040 ?
Luke : j'ai travaillé sur une partie de la conception numérique, principalement PIO, DMA, cache XIP, conception de bus et PWM. J'ai également travaillé sur la ROM de démarrage et le SDK, et, bien sûr, j'ai dû participer à la documentation.

Mathias : obtenir un signal vidéo à partir d'un processeur est quelque chose que le Sinclair ZX81 arrivait déjà à faire, mais sortir du DVI, c'est nouveau à ce prix. Alors que la sortie VGA peut être réalisée par la plupart des microcontrôleurs, quel est le défi avec le DVI ?
Luke : il y a deux choses qui rendent le DVI-D plus difficile que le VGA. La première est la sérialisation des données : l'horloge minimale des pixels pour DVI-D est de 25 MHz, et celle des bits est dix fois plus élevée, donc, au minimum, vous pilotez trois lignes série différentielles (rouge/vert/bleu) à 250 Mbps.
La seconde est que le DVI-D encode les pixels avant de les envoyer. Le codage est simple sur le plan matériel, mais un peu compliqué sur le plan logiciel, surtout lorsqu'il doit suivre la vitesse de la sortie série. Tout le reste est similaire. Il ne s'agit en fait que de DPI dans un tuyau plus rapide.
(*Note du rédacteur : Display Pixel Interface (DPI) est une interface de pixels RVB parallèle, incluant une horloge de pixel, pour transporter les pixels à un dispositif d'affichage*).

Mathias : quelle était votre motivation pour essayer le DVI sur le RP2040 ?
Luke : une fois le stress de la mise en place du silicium passé, quelques-uns d'entre nous ont voulu voir jusqu'où ils pouvaient pousser la fréquence d'horloge du système. Il y a, en pratique, une certaine marge au-dessus de la fréquence nominale de 133 MHz. J'avais joué avec DVI-D sur FPGA, dans le cadre de mon projet RISCBoy [3], et lorsque j'ai remarqué qu'il y avait un chevauchement entre les fréquences d'horloge des

bits DVI les plus basses et les fréquences d'horloge système les plus élevées sur RP2040, une idée a germé dans ma tête. La motivation était : « je me demande si c'est possible ».

Mathias : quelle a été la partie la plus difficile pour obtenir une sortie DVI (figure 1) sur le RP2040 ?
Luke : le codage TMDS. Si vous suivez l'algorithme dans la spécification DVI, il n'y a aucun espoir de le rendre suffisamment rapide sur deux cœurs Cortex-M0+ fonctionnant à la fréquence de l'horloge des bits. Il y a donc des astuces et des raccourcis pour rendre cela possible, puis du code soigneusement écrit à la main pour le rendre suffisamment rapide. Le RP2040 a beaucoup de mémoire, mais pas assez pour stocker la valeur d'une image de pixels encodés par TMDS, donc vous devez « courir après le faisceau » pendant l'encodage.

Mathias : vous avez dû légèrement overclocker le RP2040 (de 133 MHz de base à 252 MHz). Y a-t-il un point critique pour les signaux DVI (vous devez piloter les broches d'E/S avec des vitesses qui sont également plus rapides que la vitesse de base) ?
Luke : la première contrainte que vous rencontrerez sur le RP2040 est que l'horloge système doit être 1:1 avec l'horloge de bits, donc si vous essayez de passer à des modes de résolution plus élevés, les processeurs vont tout simplement se planter. Le chemin de configuration critique pour le domaine de l'horloge système sur le RP2040 sont les signaux de phase d'adresse du processeur vers les SRAM.
Cela dit, nous sommes également assez proches des limites de ce que l'on peut faire passer par les ports 3V3 à usage général. Si vous regardez le diagramme de l'œil (**figure 2**) pour 720p30 (372 Mbps) sur mon GitHub [2], ça passe, mais c'est rare. Je doute que vous puissiez voir du 1080p30 sans matériel dédié.

Mathias : outre l'augmentation de la vitesse, quel est le rôle crucial des PIO et de l'interpolateur dans le RP2040 pour faire fonctionner le DVI ?
Luke : c'est une grande exigence le fait de devoir présenter trois bits de données série, plus leurs compléments différentiels, sur les GPIO à 250 Mbps minimum. Pouvoir effectuer la conversion du mode asymétrique au mode pseudo-différentiel dans le PIO réduit de moitié la bande passante DMA, et le fait d'avoir les voies TMDS divisées en trois FIFO est utile si vous effectuez l'encodage en logiciel, car cela vous permet de spécialiser votre code pour l'encodage des composants rouge/vert/bleu. Donc, quelque chose comme PIO est crucial si vous n'avez pas de matériel dédié.
Les interpolateurs contribuent aux performances de génération d'adresses dans le codage TMDS, ce qui est certainement la clé de certaines des démonstrations que vous avez vues, mais mon truc de codage TMDS doublé en pixels tiendrait toujours sur un seul cœur Cortex-M0+ sans les interpolateurs.

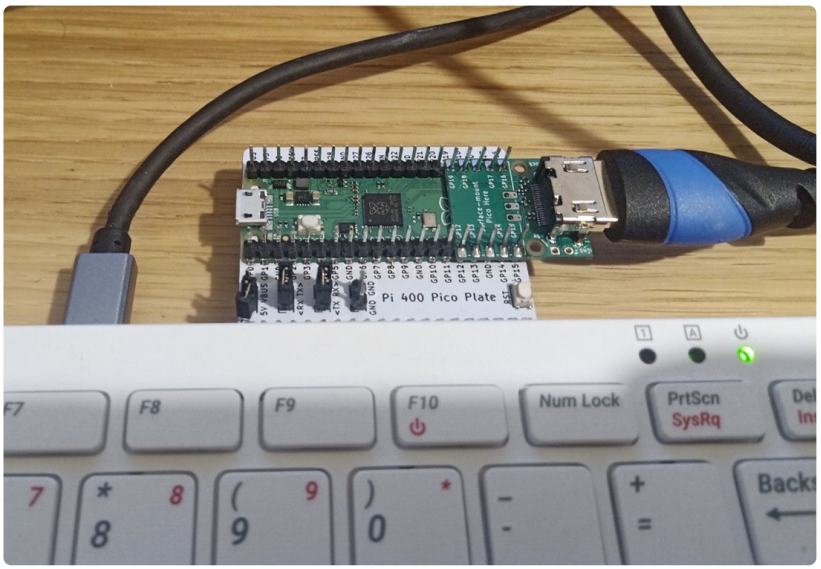


Figure 1. Raspberry Pi Pico avec le Pico DVI Sock. (Source : Luke Wren).

Mathias : votre Pico DVI Sock pour le RP2040 utilise une connexion HDMI physique (figure 3), clairement étiquetée comme DVI uniquement, donc nous n'aurons pas d'audio. S'agit-il « simplement » d'un problème de licence avec le consortium HDMI ?
Luke : il n'y a rien qui vous empêche d'ajouter des îles de données HDMI et de faire une sortie audio. En fait, quelqu'un l'a fait avec un port d'émulateur NES ! [4] [5]. Aucune connexion physique supplémentaire n'est requise pour les signaux audio, bien que, strictement parlant, vous ne soyez pas censé utiliser les fonctions HDMI avant d'interroger le canal de données de l'écran, qui n'est pas branché sur mon Pico DVI Sock. La situation de la licence HDMI est certainement une

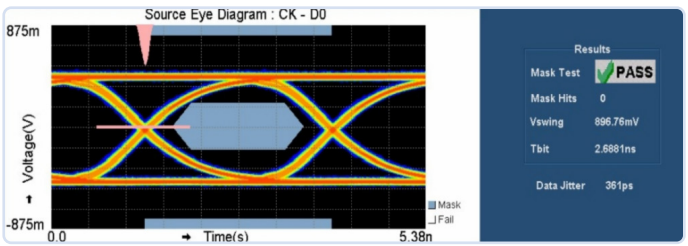


Figure 2. Diagramme de l'œil pour RP2040 DVI à 720p30. (Source : Luke Wren).

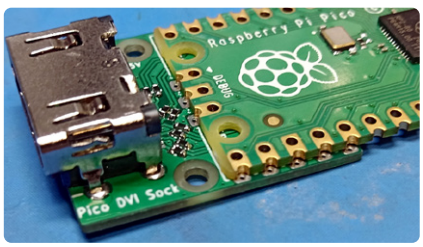


Figure 3. Pico DVI Sock pour le Raspberry Pi Pico. (Source : Luke Wren).

Figure 4. Images à haute résolution avec quelques astuces. (Source : Luke Wren).

boîte de Pandore que je ne veux pas ouvrir, et je me suis également mis en retrait en appelant le référentiel « PicoDVI », donc je vais laisser cette question à la communauté.

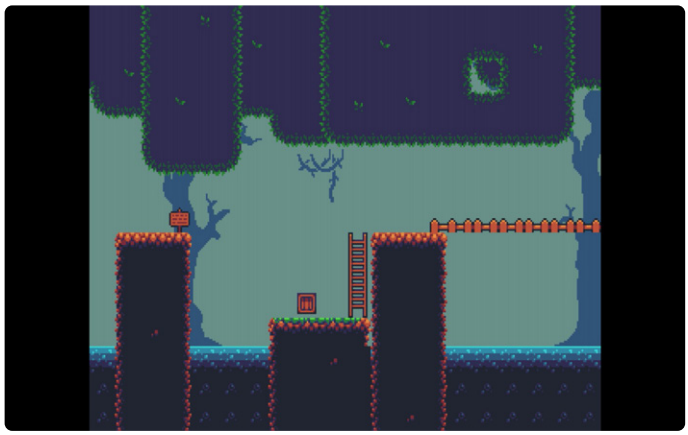
Mathias : lorsque vous utilisez la sortie DVI, quelle proportion des ressources du RP2040 est liée à cette tâche ? Y a-t-il du temps libre pour exécuter d'autres fonctions sur le microcontrôleur ?
Luke : cela dépend du mode vidéo. Disons que pour une sortie RVB565 doublée en pixels, vous finissez par dépenser environ 65 % d'un cœur pour le codage TMDS et les interruptions DMA, et l'autre cœur est alors entièrement disponible pour générer la vidéo et exécuter votre programme principale.
*Note du rédacteur : outre la génération de vidéo pure, quelques applications pour le Pico DVI Sock ont été ajoutées. L'une d'entre elles consiste à déplacer plusieurs visages d'Eben Upton sur l'écran. Si nous faisons quelques calculs, avoir une image de 640 × 480 pixels stockée comme une image complète avec une résolution de 8 bits prendrait ~308 KB de RAM (plus que ce que le RP2040 possède), donc nous le fixons à un maximum de 320 × 240 avec une couleur de 16 bits (154 KB) dans la RAM, mais la démo (***figure 4***) n'est pas pixelisée de cette façon. Il semble donc qu'il y ait un trucage du logiciel.*

Figure 5. Démo Sprite pour le Raspberry Pi RP2040.

Mathias : le logiciel permettant de générer un signal DVI est accompagné d'une bibliothèque qui gère également les sprites. Pouvez-vous en parler davantage ?
Luke : bien sûr ! Donc, lorsque vous créez une sortie vidéo, le problème suivant, c'est d'avoir besoin de données vidéo à sortir, et la bibliothèque sprite ARMv6-M est quelque chose que j'ai conçu tout en travaillant sur PicoDVI dans ce but précis.
La caractéristique essentielle de cette bibliothèque est qu'elle ne nécessite pas de tampon de trame pour le rendu, mais seulement un tampon de ligne de balayage. Votre rendu suit le faisceau, juste avant l'encodage TMDS. Cela signifie que vous pouvez prendre en charge des résolutions de sortie vidéo qui ne tiendraient pas dans la mémoire en tant que tampon de trame plat, et cela laisse la majeure partie de votre mémoire libre pour des objets graphiques.
Il y a des routines de remplissage et de *blit* raisonnablement rapides, des routines de tuilage et des routines de transformation affine de *sprites* qui vous permettent de faire tourner des *sprites* ou les mettre à l'échelle. C'est suffisant pour des jeux du niveau d'une Game Boy Advance ou autre. (*Note du rédacteur : vous pouvez voir un exemple dans la* **figure 5**).

Mathias : où avez-vous trouvé l'inspiration pour la bibliothèque, et le temps pour l'écrire ?
Luke : j'ai passé un certain temps à travailler sur du matériel graphique basé sur le *scanline* pour RISCBoy, donc, ayant fait cela matériellement, il était assez facile de le reproduire en logiciel. Tout ce qui se trouve dans le référentiel PicoDVI a été fait pendant mon temps libre sur mon ordinateur portable, à l'exception des diagrammes de l'œil, pour lesquels j'ai utilisé un oscilloscope au travail.

Mathias : il existe une démo sprite inspirée de Zelda pour le RP2040 (figure 6). Pouvez-vous nous parler de l'idée derrière cela ? Une NES ou une SNES utiliserait un matériel dédié pour composer



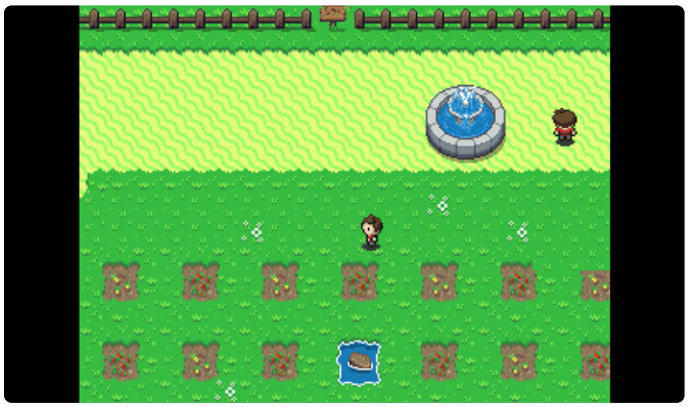


Figure 6. Démonstration du Walker avec sortie DVI.

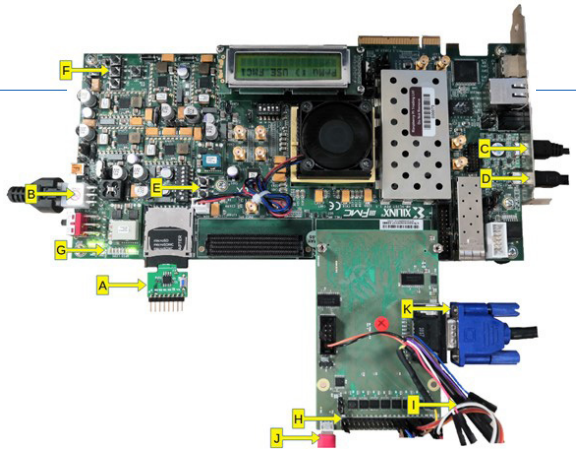
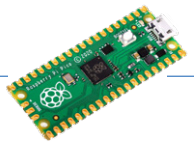


Figure 7. Prototype du Raspberry Pi RP2040 sur FPGA. (Source : Luke Wren).

des images de ce type, et ici nous avons juste deux CPU qui déplacent les pixels.
Luke : c'est en fait un portage d'une des démos RISCBoy. Comme vous le dites, il faut beaucoup de ressources pour faire tout cela en logiciel, et RISCBoy à 36 MHz peut mettre autant de *sprites* à l'écran que le RP2040 à 252 MHz. [6]

Mathias : dans les documents relatifs à la bibliothèque mentionnée, il y a l'idée d'un clone de Mario Kart pour le RP2040 ? Est-ce que c'était juste une idée, ou est-ce qu'on a commencé à travailler pour le faire fonctionner ? Il est également mentionné que l'interpolateur serait utile pour cela.
Luke : je dois donc cracher le morceau à ce stade et admettre que le fait de vouloir faire des textures et des mappages de tuiles dans le style du MODE7 de la SNES est la raison initiale pour laquelle l'interpolateur se trouvait dans la puce, bien qu'entre-temps nous ayons passé du temps à en faire un matériel plus utile et plus performant.
Nous n'avons jamais mis au point un véritable clone de MK, bien que vous puissiez voir de nombreux exemples de personnes utilisant des techniques similaires en ligne ; nous avions un cube 3D texturé avec le visage d'Eben fonctionnant sur FPGA.

Mathias : dans la documentation de votre Pico DVI Sock pour le Pico RP2040, vous mentionnez un prototype FPGA de 48 MHz. Pouvez-vous nous en dire un peu plus ?
Luke : nous utilisions une tâche automatisée pour compiler toutes les nuits une image FPGA à partir du dernier code source du RP2040 afin que, le lendemain, nous puissions l'utiliser pour le développement de logiciels.
Nous avons simplement utilisé une carte de développement Virtex 7 du commerce avec une carte fille pour décaler le niveau des E/S du FPGA jusqu'à 3,3 V (**figure 7**), et une autre petite carte qui place une mémoire flash QSPI dans le socket SD, qui est connectée à l'interface RP2040 XIP [7].
L'exécutable du FPGA est pratiquement un RP2040 [8] complet, le circuit d'horloge/de réinitialisation est simplifié et l'ADC est supprimé, mais à part cela, tout est présent et correct. Cela en fait une plateforme idéale pour le développement de logiciels, bien que la vérification réelle ait utilisé des simulations conventionnelles et un contrôle formel du modèle.



Mathias : outre la sortie DVI, vous travaillez ou avez travaillé sur d'autres projets. L'un d'eux est la PicoStation 3D. Pouvez-vous nous en parler un peu ? Si vous pouviez vous procurer toutes les pièces, serait-ce toujours le même design aujourd'hui ?
Luke : donc, PicoStation 3D (**figure 8**) est l'un de mes nombreux projets personnels que j'avais en vol avant le lancement de la RP2040. Il s'agit d'une carte contenant une RP2040, un FPGA iCE40UP5K, une carte microSD, une sortie audio, une sortie DVI-D via une prise HDMI et deux prises pour contrôleur SNES. Je lisais beaucoup sur le matériel graphique 3D à ce moment-là et je voulais une plateforme pour jouer avec, dans le contexte d'une petite console de jeux.
Cela me fait de la peine d'avoir laissé ce projet en veilleuse pendant si longtemps, mais outre les problèmes de pièces, j'ai également trop d'autres projets en cours. Tout est open-source, donc j'adorerais que quelqu'un d'autre reprenne l'idée et l'exploite.
Je pense que le choix du FPGA est tout à fait approprié, il est petit et suffisamment lent pour vous faire travailler dur pour vos démos, tout juste compatible DVI-D, et il dispose d'une mémoire embarquée généreuse et d'une poignée de tuiles DSP 16 bits, ce qui en fait une plateforme idéale pour jouer avec du matériel graphique. Il se marie également bien avec le RP2040.
Ce à quoi j'aimerais réfléchir, c'est à l'entrée-sortie. Par exemple, que se passerait-il si vous déplaciez le DVI vers le microcontrôleur et si vous déplaciez les contrôleurs SNES vers le FPGA, et si vous amélioriez un peu le circuit audio, ce genre de chose.

Figure 8. Le prototype de PicoStation3D. (Source : Luke Wren).

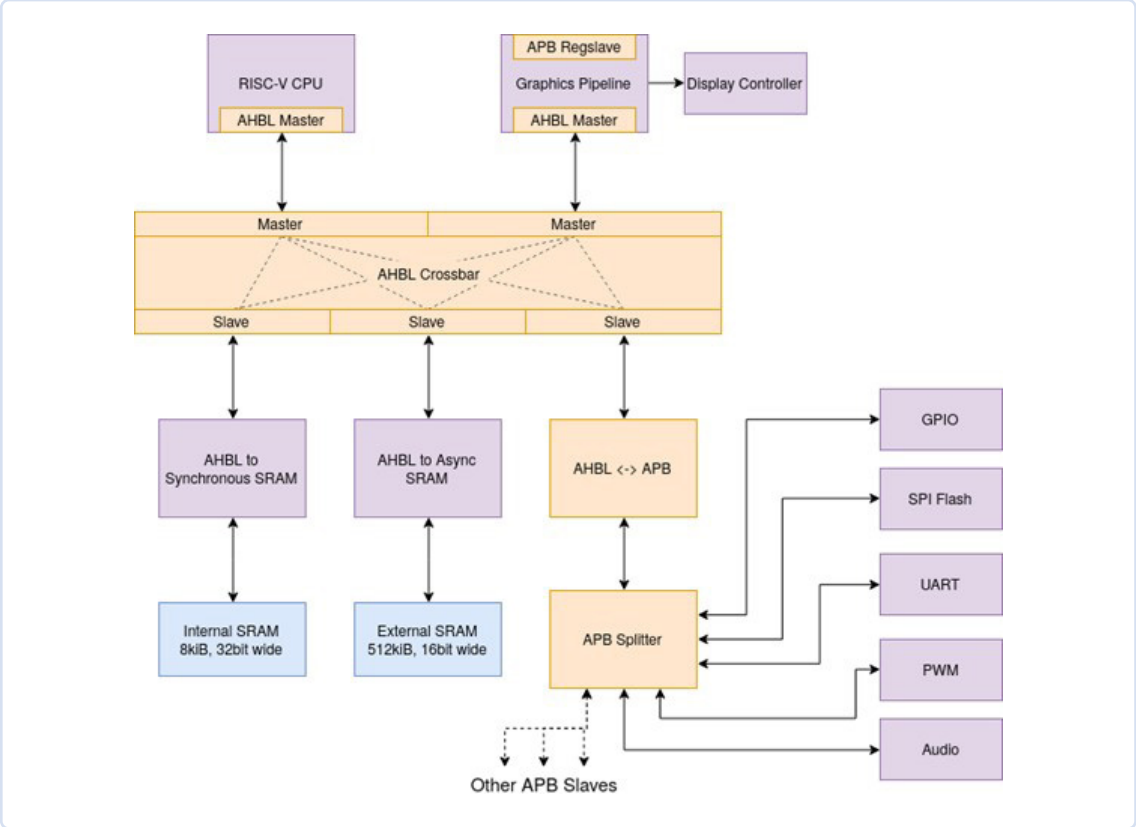


Figure 9. L'architecture du RISCBoy. (Source : Luke Wren).

Je pense que les dimensions sont à peu près correctes, puisqu'elles sont définies par les deux connecteurs de la manette SNES.

Mathias : outre les produits basés sur Raspberry Pi, vous avez également réalisé un RISCBoy, qui est alimenté par un RISC-V Core développé par vos soins et d'autres périphériques, tels qu'un moteur graphique (basé sur la 2D-sprite ?). Pouvez-vous nous dire quelques mots sur ce développement ?
Luke : RISCBoy est mon concurrent un peu tardif de la Game Boy Advance. Le matériel complet tient dans un iCE40 HX8K avec un peu de SRAM parallèle externe (**figure 9**). Eventuellement, il y aura une version physique, mais, pour l'instant, c'est toujours une carte de développement HX8K avec un peu de SRAM et des boutons, et un écran SPI suspendu à elle (**figure 10**). J'ai commencé à travailler dessus au moment où j'ai quitté l'université.

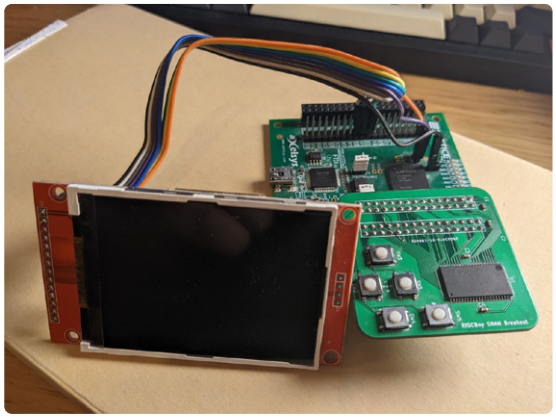
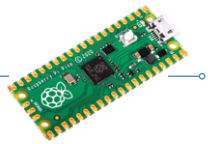


Figure 10. Le prototype du RISCBoy. (Source : Luke Wren).

Tout est écrit à partir de rien, ce n'est pas une bonne façon de faire, mais c'est une excellente façon d'apprendre, et cela rend le débogage plus amusant quand il n'y a pas une seule partie de la pile matériel/logiciel à laquelle vous pouvez faire confiance. Il y a un processeur à 32 bits (Hazard5), du matériel graphique 2D programmable, et toute l'infrastructure qui relie le tout.
Le matériel graphique réalise tous les *sprites* et tuiles et leurs transformations affine, etc. habituels, mais il le fait en exécutant, à partir de la mémoire, des listes de commandes qui fournissent un support limité pour le flux de contrôle et les branchements aux sous-routines. Pendant chaque image, le processeur écrit la liste de commandes pour l'image suivante. Il y a une paire de tampons de lignes de balayage dans le matériel, un dans lequel le rendu est effectué et un autre qui est envoyé à l'écran, de sorte que vous évitez les coûts de bande passante, de latence et d'empreinte mémoire du rendu dans un tampon de trame.
Il est en suspend pour le moment, mais j'ai bien l'intention de le terminer un jour.

Mathias : votre cœur RISC-V gagne en maturité de jour en jour. Pouvez-vous nous parler de ses origines ?
Luke : j'ai quelques processeurs à différents stades d'achèvement en ce moment, mais je suppose que vous demandez à propos de Hazard3. C'est un processeur scalaire à exécution dans l'ordre et à trois étages, originellement dévié de la source Hazard5 (le processeur RISCBoy). Vraiment, Hazard3 est un outil d'apprentissage pour moi, c'est bien beau de lire les spécifications RISC-V, mais ce n'est pas la même chose que



d'aller les implémenter, et utiliser un simple « trois étages » comme base me permet de me concentrer sur la périphérie parce que le pipeline de base fonctionne. Cela dit, les performances sont assez compétitives de nos jours (environ 3.2 CoreMark/MHz) et j'ai consacré beaucoup de temps à la vérification et à la documentation. L'introduction du débogage matériel et l'exécution de tests de bout en bout avec GDB, OpenOCD et mon propre module de transport JTAG a probablement été le sommet du projet jusqu'à présent, mais j'ai beaucoup appris tout du long.
À l'avenir, je pense que Hazard3 sera une mine d'or de composants pour tous les futurs processeurs 32 bits de classe embarquée sur lesquels je travaille. J'ai déjà vu quelqu'un sur Twitter utiliser mon module de débogage pour ajouter le support de débogage au noyau de quelqu'un d'autre. Les sources sont toutes autonomes et devraient être assez portables. Il est également sous licence Apache-2.0.

Mathias : actuellement, la conception est un RISC-V 32 bits, qui est fait entièrement en open-source. Avez-vous obtenu des ressources de Raspberry Pi pour son développement (temps, matériel, outils logiciels) à un moment donné ?
Luke : tous mes projets maison sont réalisés sur mon temps, avec mon matériel et des outils open-source. En ce moment, j'ai une machine Ryzen 7 5800X à la maison, ce qui aide à exécuter des travaux par lots. J'utilise Yosys pour la synthèse FPGA, nextpnr pour le placement et le routage, CXXRTL pour la simulation. J'utilise un iCEBreaker (iCE40UP5K) et un ULX3S (ECP5 85F) comme plateformes de référence pour Hazard3, bien que j'aie un nombre assez embêtant d'autres cartes de développement FPGA que je devrais utiliser davantage. Lorsque je dois déboguer quelque chose sur FPGA, je m'en sors bien avec mon Saleae Logic 8 que j'ai acheté lorsque j'étais étudiant, mais la plupart du temps, je peux me fier aux simulations.

Mathias : actuellement, vous passez de 32 bits à 64 bits avec votre noyau. Quel a été l'obstacle le plus difficile à surmonter jusqu'à présent au cours de son développement ?
Luke : jusqu'à présent je n'ai passé qu'un week-end à jouer avec RV64, et c'était suffisant pour passer la conformité RV64IC et les tests de conformité de débogage, donc il n'y a pas une courbe d'apprentissage énorme, les choses deviennent juste plus grandes et plus lentes.
Je hackais la base de code Hazard3 pour des raisons de commodité, mais si je voulais être sérieux dans l'implémentation d'un RV64, alors il y aurait des changements micro-architecturaux nécessaires. Il y a une raison pour laquelle vous ne voyez pas beaucoup de processeurs 64 bits à trois étages, et certainement pas ceux avec la décision de branchement à l'étage 2. Hazard3 va rester un processeur 32 bits de classe embarquée.

Mathias : est-il prévu d'essayer le cœur à un moment donné dans le futur sur du silicium réel ? Ou même de le combiner avec un moteur 2D pour obtenir un RISCBoy64 ?
Luke : j'aimerais bien essayer une bande SkyWater PDK à un moment donné, mais, pour l'instant, je me concentre sur d'autres projets. Je veux construire quelque chose de plus intéressant que « juste un autre système RISC-V » et je ne suis pas encore sûr de ce que ce sera. Pour quelque chose comme RISCBoy, il n'y a pas beaucoup d'avantages à utiliser un processeur à 64 bits.

Mathias : merci pour votre temps, Luke Wren.

220575-04 — VF : Maxime Valens

Produits

- **Raspberry Pi Pico (SKU 19562)**
  www.elektor.fr/19562
- **DVI Sock pour Raspberry Pi Pico (SKU 19925)**
  www.elektor.fr/19925

LIENS

[1] Mathias Claußen, « sortie vidéo sur les microcontrôleurs (2) », Elektor 3-4/2023 : http://www.elektormagazine.fr/220614-04
[2] Dépôt GitHub de Luke Wren : https://github.com/Wren6991
[3] RISCBoy @ GitHub : https://github.com/Wren6991/RISCBoy
[4] pico-infones @ GitHub : https://github.com/shuichitakano/pico-infones
[5] Vidéo @ Twitter : https://twitter.com/shuichi_takano/status/1477702448907419649
[6] Démonstration de Sprite fonctionnant sur le matériel RISCBoy à 36 MHz : https://twitter.com/wren6991/status/1333708886956707840
[7] Photo de la carte SD QSPI : https://twitter.com/wren6991/status/1134719550027632640
[8] Microcontrôleur RP2040 : https://raspberrypi.com/products/rp2040/