sortie vidéo sur les microcontrôleurs (2)

sortie VGA et DVI

Mathias Claussen (Elektor Labs)

Les microcontrôleurs sont capables d'envoyer des pixels à un moniteur via une interface VGA ou DVI. Que ce soit avec l'ESP32 ou le RP2040 du Raspberry Pi Pico, bien plus que 256 couleurs sont possibles. Grâce aux sprites, aux tuiles et à quelques astuces, les microcontrôleurs d'aujourd'hui peuvent produire des œuvres graphiques au moins aussi sophistiquées que les consoles de jeu classiques des années 1990.

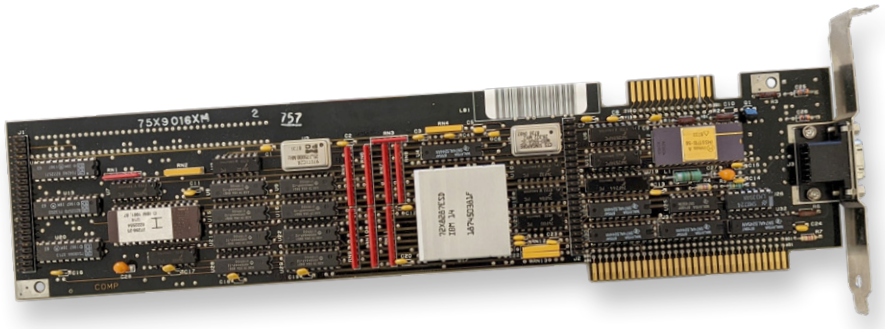

Figure 1. Carte VGA IBM (source : Wikimedia / Vlask, CCASA 4.0 https://elektor.link/cc4dot0).

La première partie de cet article traitait de la production de signaux vidéo composites aux standards PAL (**P**hase **A**lternating **L**ine) ou NTSC (**N**ational **T**elevision **S**ystems **C**ommittee). Alors que produire des images monochromes est même possible avec un Arduino UNO, les images en couleur sont plus exigeantes. Avec quelques astuces, on arrive à obtenir une sortie couleur d'un ESP32 ou même d'un ATmega, mais avec une qualité médiocre, surtout au niveau de la netteté, ce qui nuit à la lisibilité des textes. Les premiers ordinateurs grand public, tels que le Sinclair ZX81 et le Commodore 64, généraient de la vidéo composite. Cependant, pour les applications professionnelles, il fallait un affichage du texte aussi net que possible et, dans certains cas, des images en couleur. C'est ainsi qu'en 1987, IBM innova avec son interface VGA (**V**ideo **G**raphics **A**rray), dont les microcontrôleurs ont hérité. Certains microcontrôleurs peuvent même produire une image de manière entièrement numérique via DVI (**D**igital **V**isual **I**nterface). Notre deuxième partie de cette série s'intéresse donc au VGA, au DVI et aux les astuces utilisées pour l'émission et les effets graphiques.

VGA : Vidéo pour le PC IBM

VGA était la carte graphique (**figure 1**) pour les PC PS/2 introduite par IBM en 1987. Elle a rendu populaire le nouveau connecteur D-sub DE-15 à trois rangées (**figure 2**), bien que VGA ne spécifie pas seulement le connecteur physique, mais aussi les signaux nécessaires et leur *timing*.

Les signaux de couleur du VGA sont analogiques et sont transmis par trois lignes de signaux (rouge, vert et bleu). La synchronisation horizontale et verticale est effectuée numériquement à l'aide de signaux TTL distincts. La première carte VGA d'IBM pouvait émettre 256 couleurs à partir d'une palette RVB de 3 × 6 bits (= 262 144 couleurs) à une résolution de 320×200 pixels et parvenait encore à générer 640×480 pixels avec 16 couleurs. Les premiers moniteurs IBM qui convenaient ne pouvaient gérer que 640×480 ou 640×400 pixels à des taux de rafraîchissement de 60 ou 70 Hz. Les cartes VGA d'IBM ont rapidement été clonées par de nombreux autres fabricants, faisant du VGA une norme de facto pour la sortie vidéo sur ce qu'on appelait les PC compatibles IBM.

Pixels visibles et invisibles

Un signal vidéo de 640×480 pixels visibles à une fréquence de rafraîchissement de 60 Hz

Figure 2. Prise VGA sur un PC (source : Dr. Thomas Scherer).

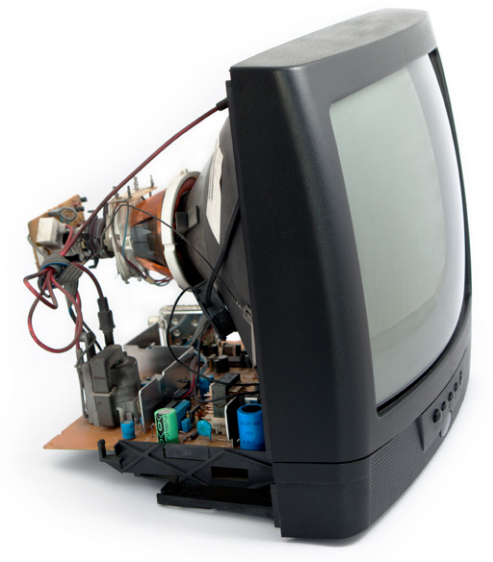

Figure 3. Téléviseur ouvert avec tube cathodique (source : Shutterstock / Serhiy Stakhnyk).

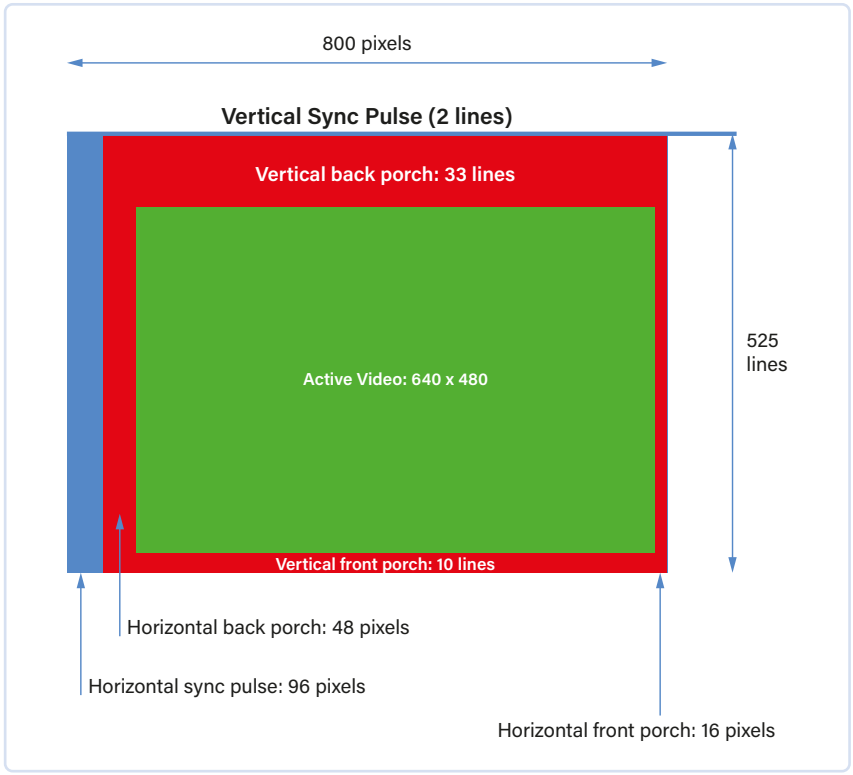

Figure 4. Timing d'un signal VGA avec 640×480 pixels visibles (source : https://elektor.link/VGAtiming).

fonctionne avec une fréquence de pixel de 25,175 MHz. Cependant, en multipliant les pixels d'une image par la fréquence d'images, on obtient seulement 18,432 MHz. D'où vient cette différence ?

Tout comme les signaux composites PAL et NTSC, le VGA comporte également des zones de signal non visibles, car cette norme a également été conçue pour les moniteurs basés sur des tubes cathodiques (**figure 3**). En principe, les exigences en matière de timing sont les mêmes que pour la commande du tube image d'un téléviseur.

La **figure 4** montre le *timing* pour le mode 640×480 pixels avec une horloge de pixels de 25,175 MHz. Une image commence par une impulsion de synchronisation verticale de 63,5551142 µs (= deux lignes d'image de 800 pixels chacune). Les trois faisceaux d'électrons sont éteints pendant ce temps, car ils se déplacent de la fin de la dernière image complète en bas à droite au début de la partie supérieure gauche de la nouvelle image. Les lignes commencent par une impulsion de synchronisation horizontale d'une durée de 69 pixels (≈ 3,81 µs). Dans les 33 lignes suivantes (lignes 3 à 35), l'impulsion de synchronisation horizontale est suivie de ce que l'on appelle le « palier arrière vertical » de 704 pixels dans ce cas (≈ 27,96 µs). Il est destiné à donner aux faisceaux d'électrons ou à la déviation magnétique suffisamment de temps pour revenir à la position de départ de la nouvelle image. Viennent ensuite 480 lignes de données d'image proprement dites, en commençant par l'impulsion de synchronisation horizontale de 96 pixels, puis le « palier arrière horizontal » de 48 pixels (≈ 1,9 µs), plus les 640 pixels d'image d'une ligne (≈ 25,42 µs) et le « palier avant horizontal » final de 16 pixels (≈ 0,64 µs). La **figure 5** montre la structure d'une telle ligne d'image. Enfin, il y a 10 lignes, chacune avec une impulsion de synchronisation horizontale suivie du « palier avant vertical » (à nouveau constitué de 704 pixels chacun). L'image transmise de 640×480 pixels visibles est donc en réalité constituée de 800×525 pixels en ce qui concerne le *timing*.

Les faisceaux d'électrons ne sont actifs que pendant l'image proprement dite et restent éteints le reste du temps. Même si cela semble complexe, il s'agit en fait de l'un des signaux d'image les plus simples pouvant être générés par un microcontrôleur. Pour que vous ne soyez pas surpris : sur Internet, vous pouvez trouver des spécifications de *timing* légèrement différentes pour la fréquence d'images VGA. Et, si vous divisez l'horloge de pixels de 25,175 MHz par les 800×525 pixels, le résultat

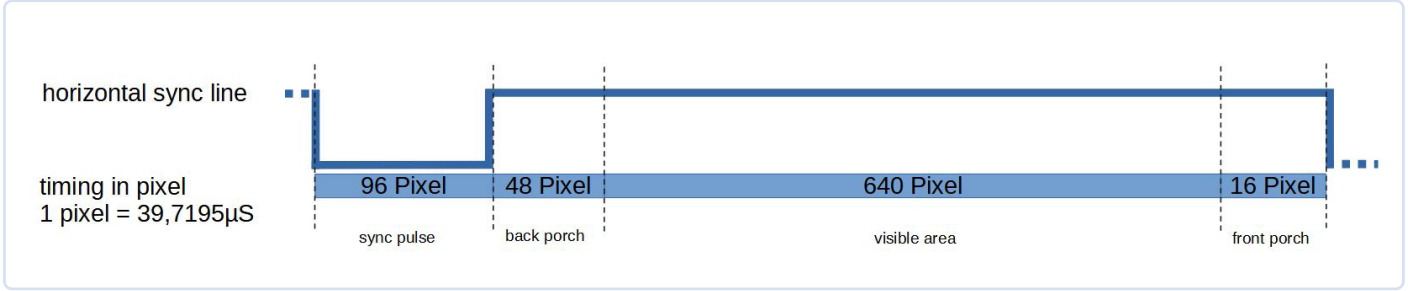

Figure 5. Timing d'une ligne en VGA.

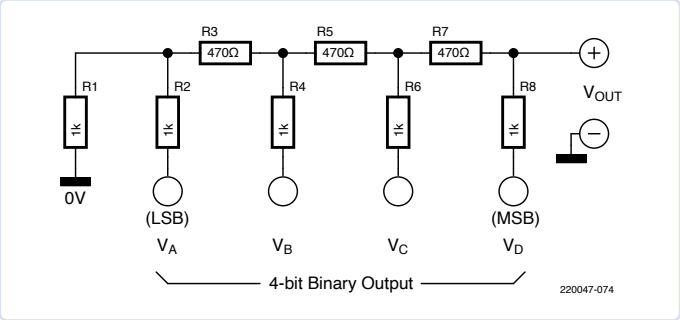

Figure 6. CNA avec échelle R2R

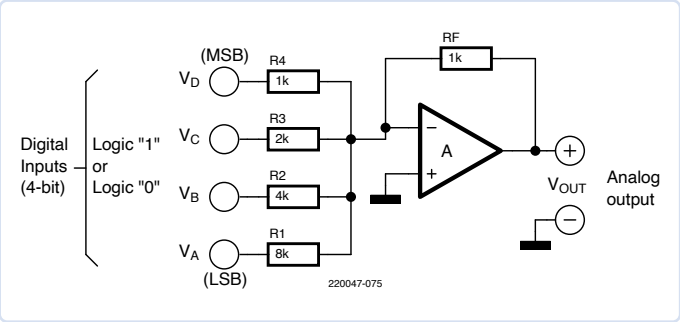

Figure 7. CNA à pondération binaire avec amplificateur opérationnel (source : https://elektor.link/BinaryWeightedDAC).

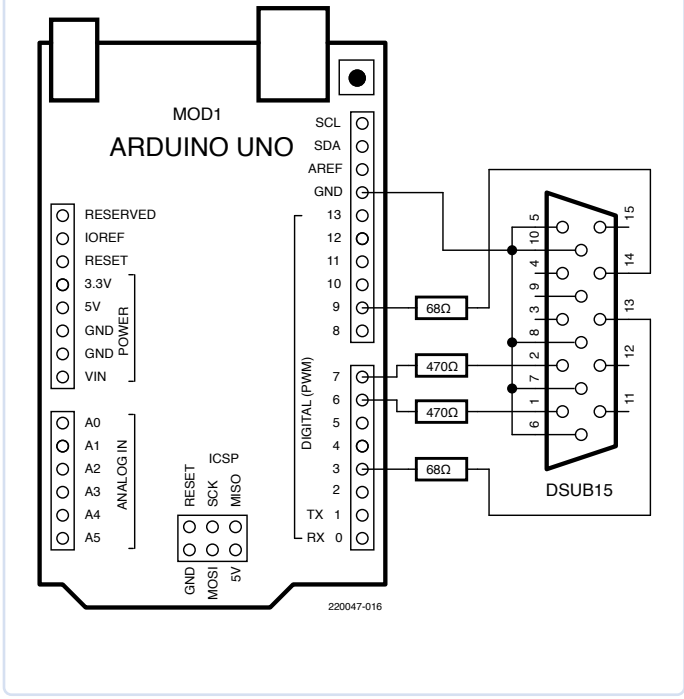

Figure 8. Câblage pour la sortie VGA sur un Arduino UNO.

n'est pas exactement 60 Hz pour la fréquence d'images, mais ≈ 59,94 Hz. Cependant, l'électronique analogique des moniteurs associés n'est pas trop exigeante à cet égard et se synchroniserait encore correctement même avec des écarts encore plus importants.

Niveaux du signal VGA

Selon la spécification VGA, la plage de tension des signaux d'image RVB analogiques est de 0 V à 0,7 V, où 0,7 V correspond à la luminosité maximale. Les signaux de synchronisation horizontale et verticale ont des niveaux TTL (0 V et 5 V).

Les microcontrôleurs dont la tension de fonctionnement est comprise entre 2,5 V et 5 V peuvent émettre les signaux de synchronisation verticale et horizontale directement via leurs broches d'E/S, puisque 2,5 V sont interprétés de manière fiable comme un niveau TTL « haut », alors que pour sortir les informations de luminosité variable pour chaque couleur, des diviseurs de tension sont nécessaires.

Les CNA internes de la plupart des microcontrôleurs n'étant pas assez rapides, les broches GPIO sont généralement utilisées comme CNA simples pour générer les trois signaux analogiques à l'aide de résistances. Lors du calcul des valeurs de résistance nécessaires, il faut tenir compte du fait que les entrées RVB analogiques du moniteur sont terminées par 75 Ω. Les CNA sont généralement mis en œuvre à l'aide de réseaux R2R (**figure 6**) ou de résistances à pondération binaire (**figure 7**).

VGA sur l'ATmega

Les signaux VGA peuvent même être générés par un microcontrôleur ATmega, de sorte qu'un Arduino UNO peut être équipé d'un périphérique VGA. Avec quatre résistances et un connecteur SUB-D à 15 broches, on peut faire en sorte qu'un Arduino UNO affiche 120×60 pixels en quatre couleurs. La bibliothèque Arduino correspondante est *VGAX* [1] et peut être téléchargée sur la page GitHub du projet.

Si un Arduino UNO est câblé comme indiqué sur la **figure 8**, il est même possible de produire des graphiques en quatre couleurs (**figure 9**). Le facteur limitant ici est la quantité de mémoire disponible. Comme les signaux de couleur sont générés par les broches GPIO, il faut être en mesure de les alimenter en nouvelles valeurs assez rapidement.

Dans ce cas, le *timing* de la génération du signal est géré par des interruptions, ce qui a pour effet secondaire que ces interruptions

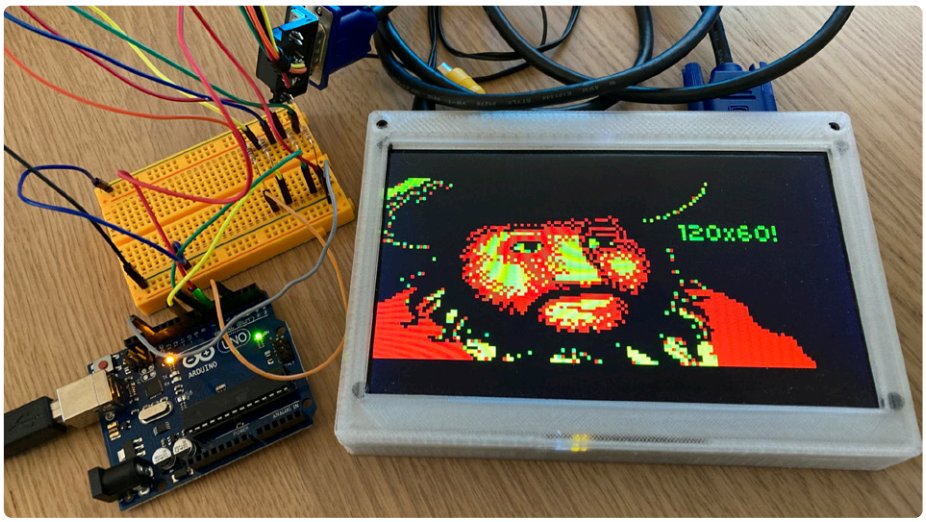

Figure 9. Sortie VGA sur un écran connecté à l'Arduino UNO.

peuvent être perturbées par celles d'autres périphériques. *Timer1* et *Timer2* génèrent les impulsions de synchronisation horizontale et verticale. *Timer0* est utilisé pour contourner la gigue dans les appels d'interruption.
La bibliothèque *VGAXUA* [2] pour l'Arduino UNO et l'Arduino Mega peut produire des images monochromes de 192×080 à 200×240 pixels. Pour gagner du temps de calcul, l'USART est utilisé ici pour générer l'image. Chaque octet dans la mémoire (d'image) correspond alors à 8 pixels. Des solutions plus anciennes pour la sortie VGA avec des puces AVR ont soit surcadencé le microcontrôleur jusqu'à 32 MHz [3], soit utilisé une mémoire externe pour la sortie vidéo [4].

VGA sur l'ESP32

Un ESP32 peut aussi faire du VGA. Les conditions idéales sont fournies par son horloge de 240 MHz et la matrice E/S flexible, qui permet de sortir des signaux numériques jusqu'à 80 MHz. La mémoire interne de l'ESP32, avec 520 Ko, est suffisante pour une sortie de 640×480 pixels avec 256 couleurs (= 307 200 octets), ce qui laisse encore assez de RAM pour d'autres logiciels.
La spécification du *timing* d'un signal VGA à 640 480 pixels et 60 Hz est disponible sur [5]. Le projet « ESP32 VGA » [6] propose une façon intéressante de produire un signal VGA en utilisant le contrôleur I²S interne (**I**nter-**IC** **S**ound) [6]. Outre l'émission de données audio, le contrôleur I²S de l'ESP32 peut servir d'interface pour une caméra ou un écran LCD (la **figure 10** détaille la partie correspondante du circuit synoptique).
L'interface LCD du contrôleur I²S peut fournir une largeur de bus allant jusqu'à 24 bits, ce qui serait même suffisant pour TrueColor (16 777 216 couleurs) si l'ensemble des 24 bits était disponible pour la sortie de la couleur. Toutefois, un signal VGA nécessite une synchronisation horizontale et verticale en plus des signaux RVB, ce qui ne laisse que 22 bits pour la couleur (**figure 11**).
Il reste deux obstacles à franchir : une image de 640×480 pixels occupe 921 600 octets de RAM en couleur 24 bits - c'est plus que ce qu'offre un ESP32. De plus, il faut atteindre une horloge de pixels de 25,175 MHz. Pour la sortie de 24 bits, le contrôleur I²S doit lire 32 bits par pixel dans la mémoire. Il semble que le contrôleur I²S de l'ESP32 puisse sortir les mots de 32 bits à un maximum de 10 MHz, limite de la vitesse de sortie série des pixels,

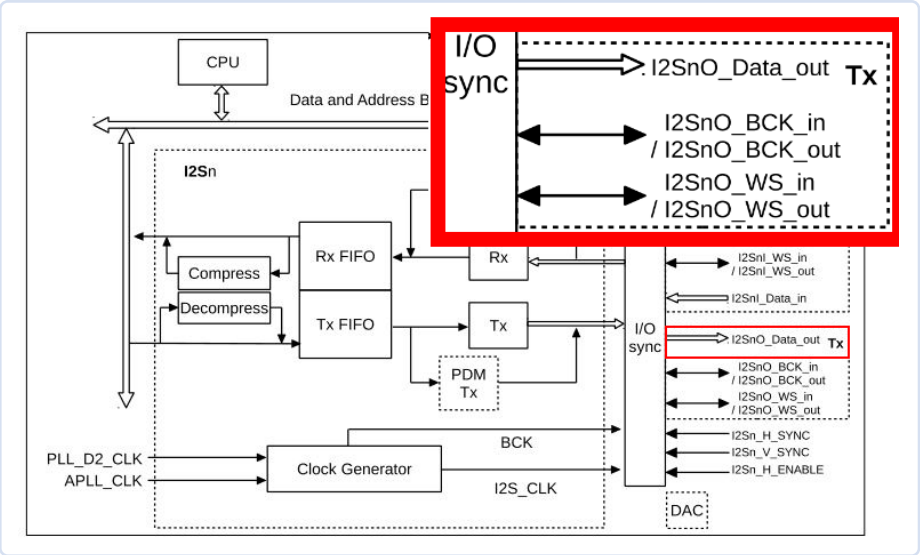

Figure 10. Le contrôleur I²S de l'ESP32 (source : Espressif – https://elektor.link/ESP32TechMan).

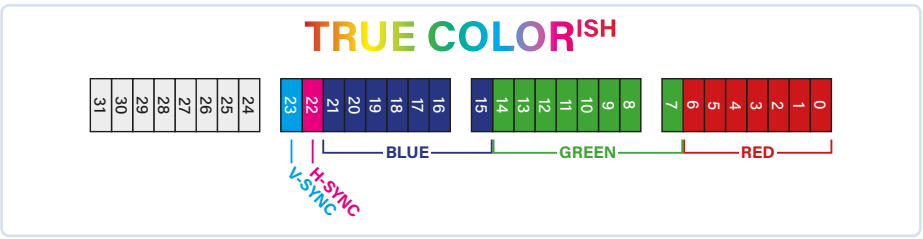

Figure 11. Sortie vidéo 22 bits avec H- et V-sync sur ESP32 (source : YouTube – https://elektor.link/YTESP32VGA).

ce qui est insuffisant. La résolution maximale, quant à elle, est limitée par la mémoire interne. Selon la liste des fréquences d'horloge de pixels en fonction de la résolution VGA sur [8], dans le mode avec 800×600 pixels à 60 Hz, l'horloge de pixels est à 40 MHz, soit exactement quatre fois le maximum d'un ESP32. Ainsi, 200×600 pixels à 60 Hz et une couleur de 22 bits seraient mathématiquement possibles en termes de *timing* et de RAM, bien que ces pixels ne seraient plus rectangulaires, mais étirés horizontalement. Pour des pixels rectangulaires, la résolution verticale doit être divisée par quatre, et chaque ligne d'image doit être sortie quatre fois. Le résultat est de 200×150 pixels en TrueColor. Une image de 800×600 pixels théoriques (**figure 12a**) est donc affichée de manière assez grossière, mais joliment colorée (**figure 12b**).
Mais il existe un compromis utilisable, car le contrôleur I²S peut également traiter des mots de 16 bits. Sur les 16 bits, 14 bits sont alors

Figure 12. 800×600 (a) devient 200×150 (b) et 400×360 pixels (c). (b) et (c) sont représentés à la même taille que (a) a pour une meilleure comparaison.

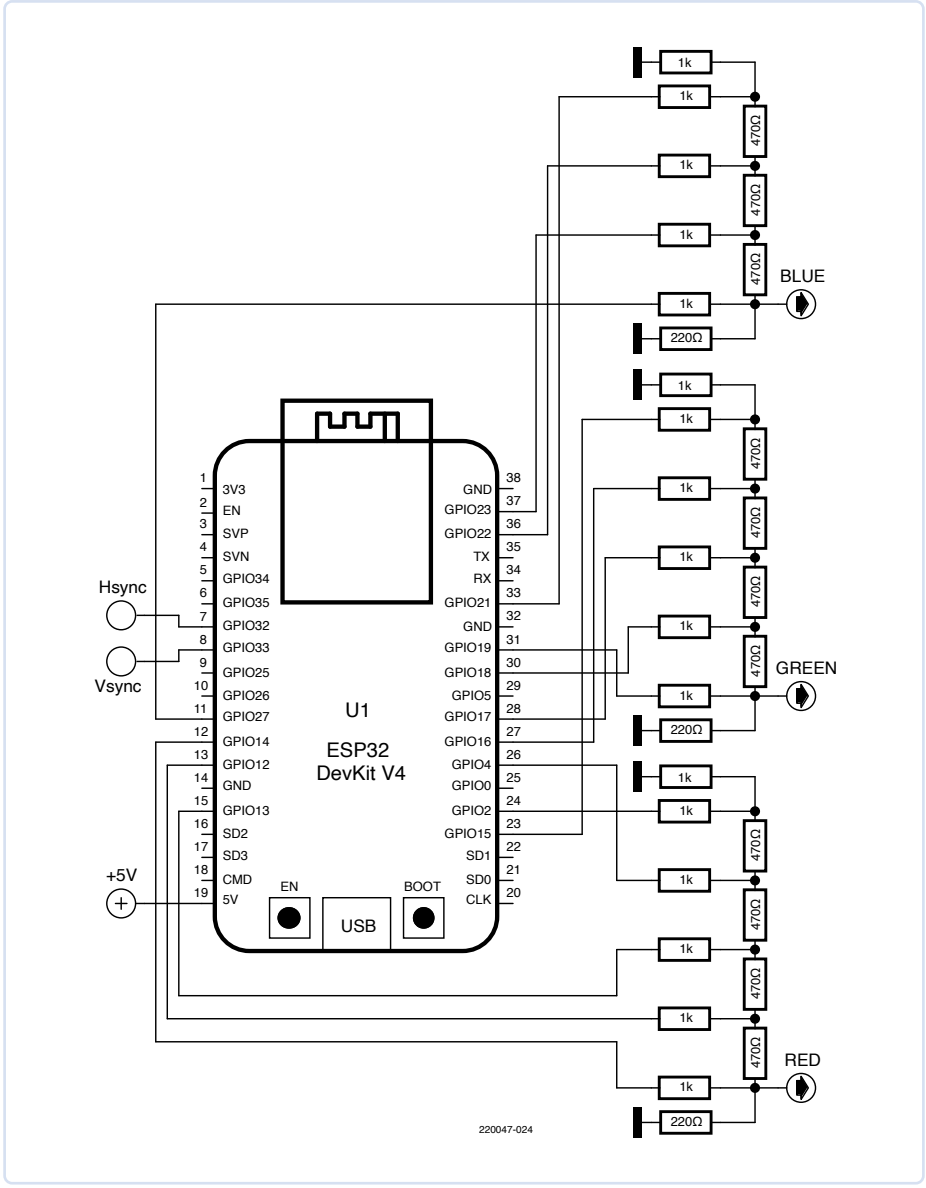

Figure 13. Génération du signal VGA via le CNA R2R sur ESP32.

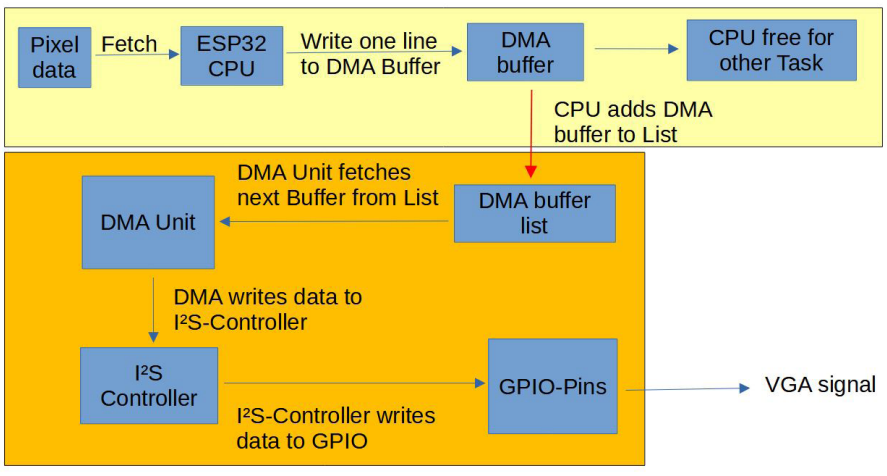

Figure 14. Sortie de pixels avec ESP32.

disponibles pour les informations de l'image et 2 bits pour les signaux de synchronisation, ce qui permet d'obtenir 16 384 couleurs affichables. En utilisant l'APLL de l'ESP32, on dispose d'une base de temps stable pour le *timing* VGA, permettant de réaliser des images en couleur 14 bits avec des résolutions de 320×240, 360×400 ou 480×400 pixels.

Pour devenir des signaux RVB, les informations de couleur sur 14 bits doivent être converties en analogique. Un CNA R2R tel que celui de la **figure 13** est une solution simple et efficace à cet effet. L'image de la **figure 12a** semble bien meilleure avec une résolution de 360×400 pixels (**figure 12c**).

Pour récupérer et afficher les données de la RAM, il y a quelques astuces : pour transférer les données vers le contrôleur I²S, le contrôleur DMA (Direct Memory Access) de l'ESP32 est utilisé pour délester la CPU, grâce à sa capacité de copier des données sans intervention de la CPU. La source, la destination, la quantité de données, le type de transfert et le signal de début de transfert sont suffisants à cet effet.

L'ESP32 prépare les données pour chaque ligne et les stocke dans un tampon de ligne qui contient les pixels visibles et les informations pour la synchronisation horizontale et verticale. Lorsqu'une ligne a été préparée, le contrôleur DMA copie les données vers le contrôleur I²S. Pendant ce temps, les cœurs de l'ESP32 peuvent s'occuper à d'autres tâches. La **figure 14** montre le déroulement du processus.

Cette procédure permet également de sortir des images sans les avoir au complet en mémoire. Pour cela, l'unité centrale de l'ESP32 doit toutefois être en mesure de calculer les pixels de la ligne suivante pendant le temps de sortie de la ligne en cours.

La bibliothèque *VGA* de bitluni pour le *framework* Arduino ESP32 est disponible sur GitHub [9]. Mais la bibliothèque *ESP32Lib* peut faire bien plus que de la sortie VGA. Elle contient également des fonctions pour les *sprites* [10] et un moteur de rendu 3D. De plus, la bibliothèque *FabGL* [11] ne gère pas seulement une sortie VGA avec l'ESP32, mais fournit les ingrédients d'un système complet d'images et de sons. Il existe même des tutoriels vidéo pour FabGL sur YouTube [12].

VGA sur le Raspberry Pi Pico

Grâce à son système sur une puce RP2040, le Raspberry Pi Pico est non seulement petit, mais aussi très polyvalent. Avec un processeur

à deux cœurs, 264 ko de RAM interne et une mémoire flash externe, cette carte à petit prix convient à de nombreux projets. La sortie du Raspberry Pi Pico fut accompagnée d'une carte de démonstration de sa capacité à produire des signaux VGA (**figure 15**), qui a depuis été mise à jour de manière à pouvoir aussi accueillir un Raspberry Pi Pico W. Les fichiers nécessaires pour la carte [13] sont disponibles sous la forme d'un projet KiCad, permettant à chacun de la réaliser.

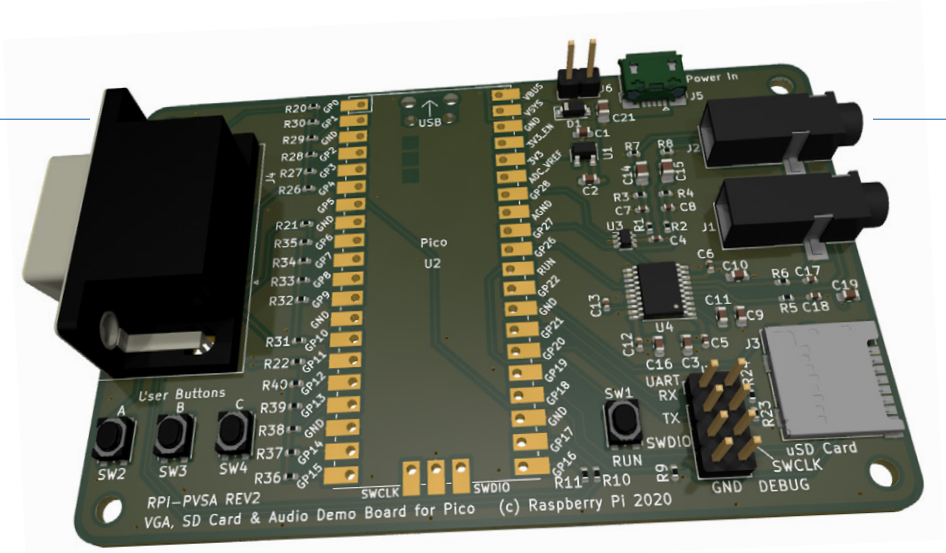

Figure 15. Carte de démonstration VGA, carte SD et audio pour Pico.

Le RP2040 offre suffisamment de mémoire pour stocker une image de 320×240 pixels à 16 bits (= 65 536 couleurs) entièrement en RAM. Les machines à états des E/S programmables (PIO) du RP2040 se prêtent idéalement à cette fin.

Pour le signal VGA, il faut des valeurs pour les niveaux RVB ainsi que pour la synchronisation horizontale et verticale. Pour la génération de ces signaux avec les PIO du RP2040, il suffit d'un logiciel. Malheureusement, le RP2040 ne possède pas de CNA, un circuit externe est donc nécessaire.

Un simple CNA R2R ou un CNA à pondération binaire convient à cet effet. Pour un CNA R2R, seules deux valeurs de résistance sont nécessaires, ce qui simplifie beaucoup les choses. Pour un CNA à pondération binaire, une valeur de résistance différente est nécessaire pour chaque bit. Le circuit de la carte de démonstration avec sortie VGA illustré à la **figure 16** est tiré d'un fichier PDF intitulé « *Hardware design with RP2040* » [14] par Raspberry Pi Ltd. Pour les couleurs spécifiées par RGB565, 5 broches GPIO chacune sont utilisées pour les couleurs rouge et bleue, tandis que le vert nécessite six broches pour les 65 536 couleurs résultantes. Les valeurs des résistances résultent de la structure du CNA, de la tension de sortie nécessaire et des résistances de terminaison du moniteur. Les valeurs des signaux RVB analogiques sont comprises entre 0 V et 0,7 V. Les entrées RVB d'un moniteur sont terminées par 75 Ω à la masse. Par conséquent, les diviseurs de tension doivent générer un maximum de 0,7 V pour chacune des trois couleurs en utilisant les 3,3 V des GPIO, ce qui se traduit par des rapports de résistance de 1:2:4:8:16. Pour obtenir 0,7 V à partir de 3,3 V, vous avez besoin d'un diviseur de tension avec un rapport de 3,7:1. Cinq ou six résistances par couleur sont connectées en parallèle. Les circuits parallèles doivent aboutir à 278 Ω pour arriver à un maximum de 0,7 V à la résistance

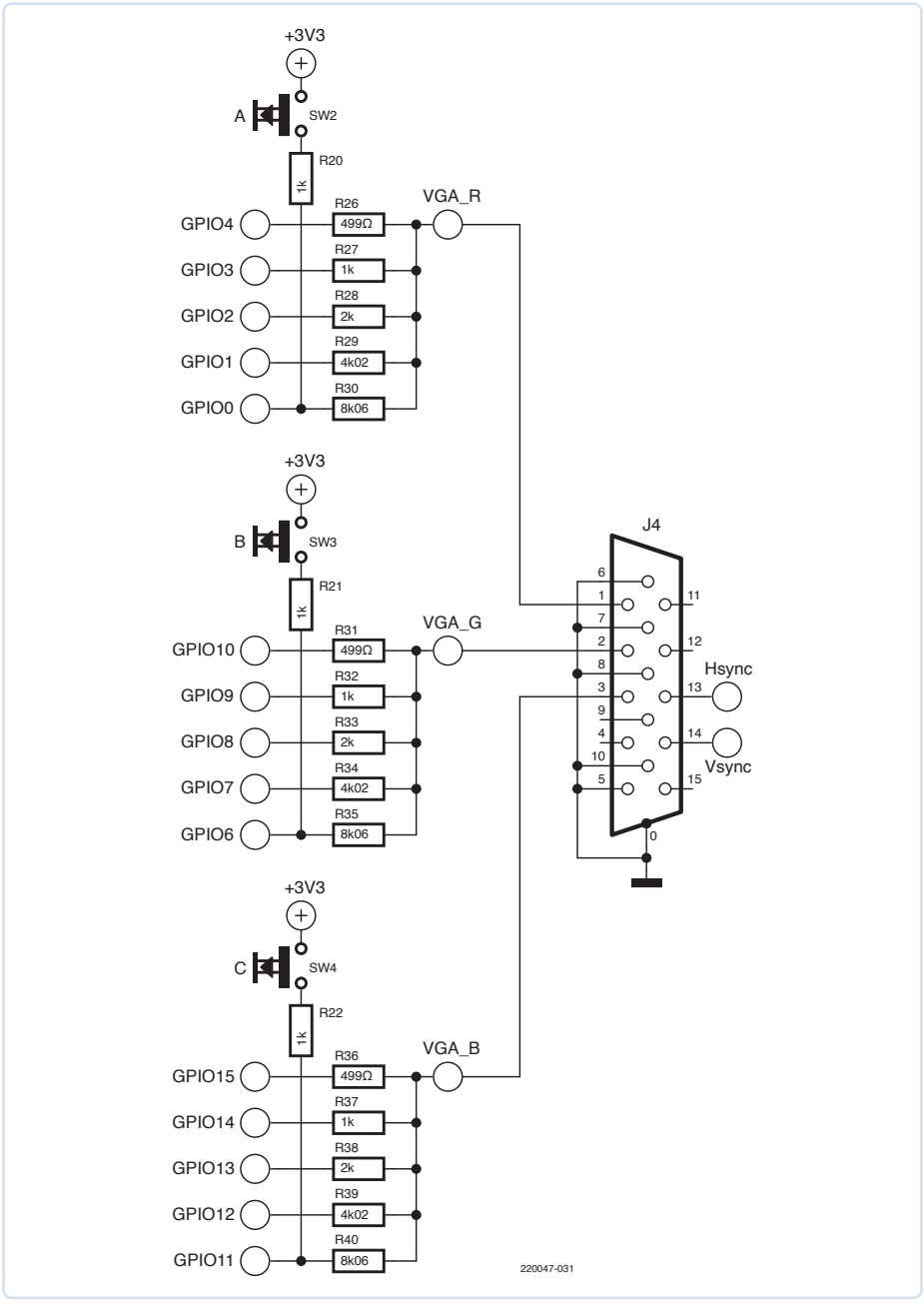

Figure 16. Une partie du circuit de la base VGA pour le Raspberry Pi Pico.

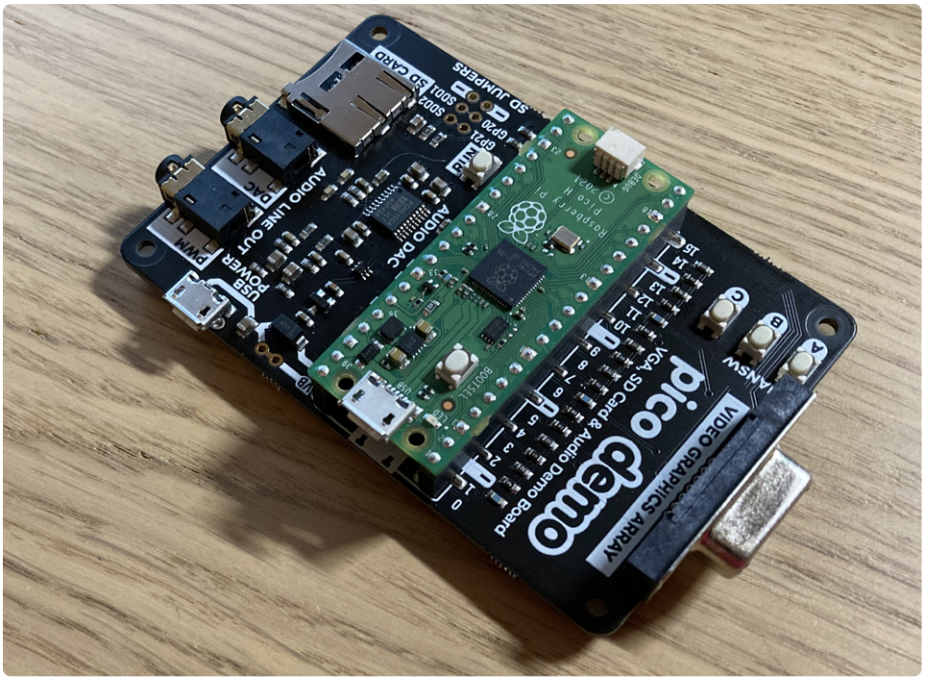

Figure 17. Base VGA Pimoroni pour le Raspberry Pi Pico.

de terminaison de 75 Ω avec une alimentation de 3,3 V.
Avec les résistances de la série E-96, des valeurs de 499, 1000, 2000, 4020 et 8060 Ω sont disponibles avec une tolérance de 1 %. Connectées en parallèle, on obtient 258 Ω, ce qui conduit à un 0,74 V suffisamment précis au niveau de la résistance de terminaison.
Si vous voulez une carte entièrement assemblée, vous pouvez acheter la carte de démonstration Pimoroni Pico VGA (**figure 17**) - voir aussi l'encadré des produits connexes. En plus de la sortie VGA, la carte comporte un codec audio et un emplacement pour carte SD.

Sortie de pixels avec le RP2040

Pour pouvoir émettre un signal VGA, il faut produire et stocker les données de l'image. La RAM du RP2040 suffit pour stocker une image de 320×240 pixels en couleur 16 bits. Le dépôt GitHub *pico-playground* [16] contient des démos adéquates.
Pour une sortie simple, les données de l'image sont stockées sous forme de valeurs RGB565 dans 150 Ko de mémoire réservée. L'image est lue de la mémoire ligne par ligne, puis le *timing* approprié pour la sortie de l'image est généré pour chaque ligne. La **figure 18** montre une séquence simplifiée.
Ce type de sortie d'image à partir de la RAM permet aux deux cœurs de rester libres pour d'autres tâches. Il suffit de fournir une nouvelle ligne d'image avec un *timing* adéquat. La bibliothèque *pico_scanvideo* qui le gère se trouve sur [17]. Elle permet la préparation de plus d'une ligne d'image à l'avance. Le nombre de lignes de l'image et son adaptation à la résolution de l'écran sont configurables. Ainsi, par exemple, vous pouvez traiter 320×200 pixels avec une résolution de 640×480 pixels. La mise en mémoire tampon des lignes d'image permet également de réaliser d'autres astuces, comme la génération d'images « à la volée ». Il s'agit plus ou moins d'une course contre la montre ou contre le faisceau d'électrons.

Le Raspberry Pi Pico en tant qu'artiste dessinateur

Le Raspberry Pi Pico et les autres cartes basées sur le RP2040 peuvent être de véritables artistes du dessin. Dans la **figure 19**, plusieurs *sprites* se déplacent sur l'écran. La sortie est de 640×480 pixels en couleur 16 bits. C'est plus que ce que peut contenir la RAM du Raspberry Pi Pico. Pour cette raison, tous les pixels sont redessinés pour chaque nouvelle image à sortir.
Alors que cette démo des fonctions de base permet de dessiner des images 2D sur l'écran, la bibliothèque *PicoVGA* de Miroslav Nemecek va un peu plus loin. La sortie graphique est limitée à 8 bits, ce qui laisse plus de broches d'E/S libres et nécessite moins de RAM pour une image complète (75 au lieu de 150 ko). Le dépôt GitHub de la bibliothèque contient plusieurs démos et petits jeux [18]. La figure 20 montre le jeu *Ants* fonctionnant sur

Figure 19. Sprites animés.

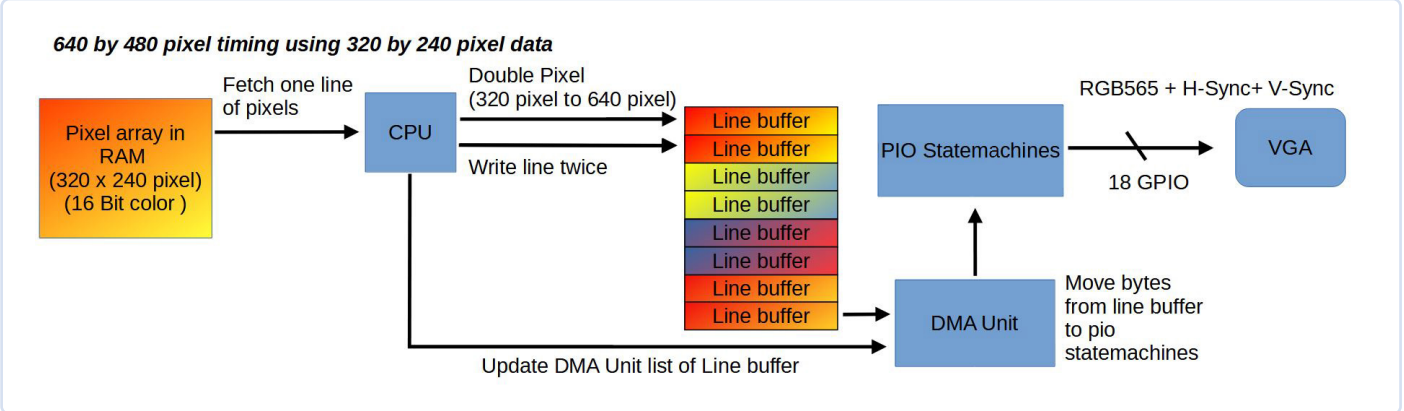

Figure 18. Séquence de sortie VGA avec Raspberry Pi RP2040.

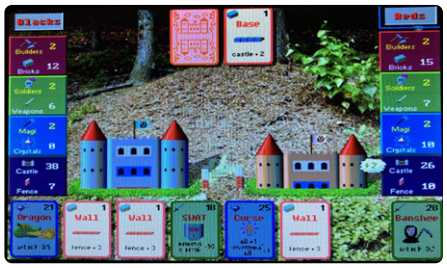

*Figure 20. **Ants** sur un Raspberry Pi Pico.*

*Figure 21. **Pac-Man** sur un Raspberry Pi Pico.*

Figure 22. Pseudo-3D avec un Raspberry Pi Pico.

le Raspberry Pi Pico et la **figure 21** le classique Pac-Man. Une brève description de la bibliothèque se trouve sur la page allemande de MagPi [19].

Le Raspberry Pi Pico contient des unités fonctionnelles qui permettent même des effets pseudo-3D (**figure 22**). Ce type d'affichage a été rendu célèbre par le SNES (Super Nintendo Entertainment System) avec des jeux tels que *Super Mario Kart* (**figure 23**) ou *F-Zero* en mode 7 [20]. La bibliothèque correspondante est disponible sur GitHub [21].

DVI

Introduit en 1999, le DVI est une interface numérique [22] d'envoi d'images à un moniteur sans perte de qualité. La **figure 24** montre une prise blanche typique pour un DVI-D à deux canaux. Les spécifications exactes du DVI se trouvent dans un fichier PDF sur [23].

Lorsque les premiers écrans à cristaux liquides (LCD) sont devenus abordables, de nombreuses cartes graphiques utilisaient encore l'interface VGA. Mais en Full HD (1920×1080 pixels), l'image n'était pas vraiment nette en raison du passage en analogique. Avec le DVI, se répandit une interface numérique qui pouvait alimenter directement un moniteur LCD, sans pertes de conversion, avec des images haute résolution en couleur 24 bits.

8b/10b, TMDS et horloge 1:10

Alors qu'avec le VGA, trois des cinq signaux étaient analogiques, avec le DVI, tout est numérique. Quatre paires de lignes différentielles (±Data 0, ±Data 1, ±Data 2 et ±Clock) représentent le minimum pour la transmission d'images à un seul canal avec DVI.

Les données de l'image RVB sont transmises via Data 0, Data 1 et Data 2. Pour une image en couleur 24 bits, chaque couleur est transmise en utilisant 8 bits. En outre, il existe un signal d'horloge qui permet de combiner les couleurs pour reconstituer les pixels du côté du récepteur.

La transmission numérique utilise la méthode TMDS (**T**ransition-**M**inimized **D**ifferential **S**ignaling) [24], qui assure un transport robuste du signal et un faible rayonnement électromagnétique grâce à son codage des données.

La méthode de codage utilisée est 8b/10b [25], où seulement 460 des 1024 combinaisons possibles sont utilisées pour coder l'information 8 bits pour les trois couleurs, rouge, vert et bleu. Plusieurs valeurs de 8 bits peuvent donc être représentées par deux combinaisons. Quatre combinaisons sont utilisées pour la signalisation de C0 et C1 pour la synchronisation horizontale et verticale. Lors de la sortie des pixels d'une ligne, le nombre de zéros et de uns dans le flux de données est équilibré pour obtenir des flux de données sans composante continue, ce qui est rendu possible par la double représentation de certaines valeurs 8 bits.

Même si les informations de l'image sont transmises de manière entièrement numérique, les signaux de synchronisation horizontale et verticale sont toujours présents. Toutefois, il ne s'agit plus de lignes de données dédiées, mais de signaux codés dans le flux de données 0 à l'aide de C0 et C1. Le flux de données DVI est conçu pour être facilement converti en un signal VGA analogique.

Le signal d'horloge en DVI ne correspond pas au débit binaire pour le rouge, le vert et le bleu, mais a un rapport de 10:1 et correspond à l'horloge de pixels en raison du codage 8b/10b. Pour la plus basse résolution DVI de 640×480 pixels à 60 Hz, une horloge de pixels de 25,175 MHz ou un débit binaire de 251,75 MHz est nécessaire pour les informations de couleur, tout comme avec le VGA. Ainsi, ses origines VGA sont encore bien visibles dans le DVI.

DVI avec le Raspberry Pi Pico

En supposant la résolution la plus basse possible spécifiée précédemment, un Raspberry Pi Pico doit émettre trois flux de données à 251,75 Mbps. Cela n'est tout simplement pas possible avec son horloge maximale de 133 MHz, car près de 2 bits devraient être émis par tic d'horloge pour chacun des trois flux de données couleur.

*Figure 23. **Super Mario Kart** avec effet pseudo-3D.*

Figure 24. Prise DVI-D sur un PC (source : Dr. Thomas Scherer).

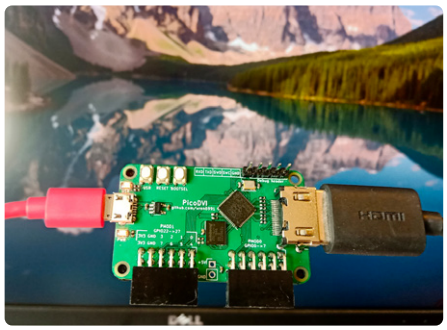

Figure 25. PicoDVI - RP2040 avec sortie vidéo DVI (source : Wren6991 / https://elektor.link/picodviimg).

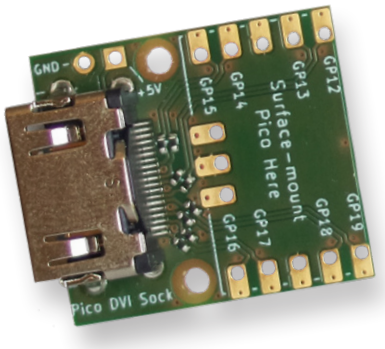

Figure 26. Pico DVI Sock pour Raspberry Pi Pico.

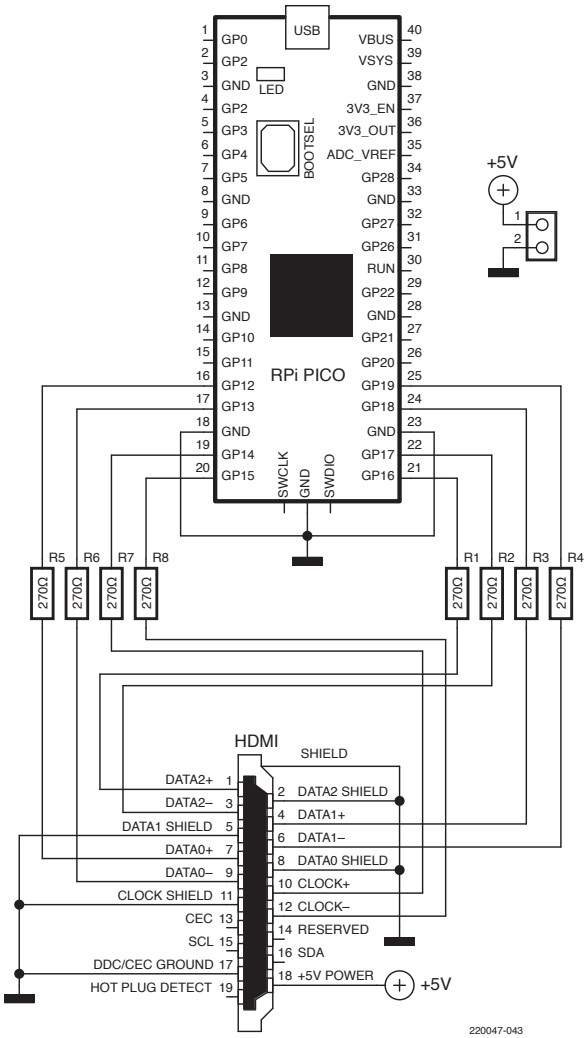

Figure 27. Schéma du Pico DVI Sock.

Ainsi, le Raspberry Pi Pico ne franchirait déjà pas cet obstacle. Cependant, le RP2040 est extrêmement surcadençable. Une horloge de 251,75 MHz pour les deux cœurs est possible sans dysfonctionnements notables à température ambiante. Cependant, les broches GPIO du RP2040 doivent également suivre cette énorme vitesse. En outre, les données d'image doivent encore être codées sous forme de flux TMDS et donc préparées en conséquence. Luke Wren a pu montrer [26] qu'un RP2040 en est capable (**figure 25**) ; voir son interview sur [27] pour plus d'informations.

L'un des deux cœurs du RP2040 est occupé à environ 60 % par l'encodage TMDS. De plus, trois des huit machines d'état PIO sont utilisées pour sortir les données avec les 251,75 MHz nécessaires. Certaines fonctions du contrôleur DMA déchargent les cœurs du transfert des données. De cette façon, un cœur reste entièrement libre et disponible pour votre propre logiciel.

Une prise DVI ou HDMI (**figure 26**) et huit résistances suffisent pour connecter électriquement le Raspberry Pi Pico à l'entrée DVI ou HDMI d'un moniteur. Le câblage nécessaire (**figure 27**) est assez simple.

Dans le dépôt GitHub de PicoDVI [26], Luke Wren décrit comment l'interpolateur du RP2040 peut être utilisé pour accélérer l'encodage TMDS, ce qui permet de gagner du temps de calcul. L'interpolateur était initialement destiné à générer des graphiques pseudo-3D, comme dans *Pilotwings* (**figure 28**). Cependant, il est suffisamment souple pour être utilisé pour soulager les cœurs du processeur lors de la génération des trois flux de données TMDS à partir des données RVB.

Alors, comment les pixels passent-ils de la mémoire vive au moniteur ? Pour une sortie avec une résolution de 640×480 pixels, une image de 320×240 pixels est mise à l'échelle. Par conséquent, il n'y a que 240 lignes à coder par TMDS 60 fois par seconde. Chaque ligne est émise deux fois, et les pixels sont donc deux fois plus hauts.

Pour ce faire, les données brutes sont fournies ligne par ligne, comme pour la sortie d'un signal VGA. Les lignes peuvent provenir d'un tampon de trame dans le Raspberry Pi Pico ou être produites « à la volée ». Comme pour le VGA, des projets intéressants sont également possibles avec la sortie DVI. Ici, un seul des deux cœurs du processeur, avec son interpolateur, est occupé à générer le signal DVI.

Figure 28. Pseudo-3D dans le jeu Pilotwings.

*Figure 29. Démonstration de **Walker** pour le Raspberry Pi Pico.*

*Figure 30. Les tuiles dans la démo de **Walker**.*

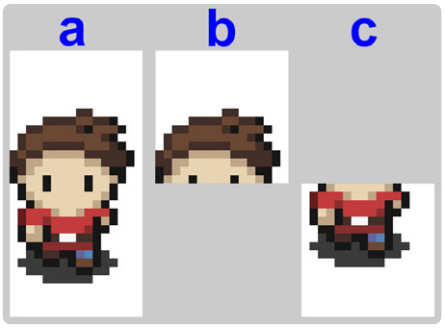

*Figure 31. Personnage de jeu avec fond transparent (a) et ses **sprites** supérieur (b) et inférieur (c).*

La démonstration du *Walker* (**figure 29**) montre de manière impressionnante qu'un Raspberry Pi Pico a encore assez de réserves pour faire de la magie à l'écran.

Sprites, tuiles et cartes de tuiles avec le RP2040

Pour les sorties VGA et DVI, il existe une bibliothèque graphique qui prend en charge les *sprites*. Dans la démo du *Walker*, l'image finie était composée de nombreux petits éléments appelés tuiles (**figure 30**). Chaque tuile a une taille de 16×16 pixels. Ces tuiles sont destinées à servir d'arrière-plan graphique et ne comportent généralement aucune information de transparence, contrairement au personnage du joueur (**figure 31a**), qui est composé de deux *sprites* (**figure 31b** et **figure 31c**). L'arrière-plan et les *sprites* peuvent être placés individuellement. Avec la sélection appropriée, de sympathiques arrière-plans peuvent être assemblés comme dans la **figure 32**.

Les cartes de tuiles constituent une bibliothèque (**figure 33**), dans laquelle des tuiles de 16×16 pixels sont puisées puis assemblées par le Raspberry Pi Pico pour former une image. Mais pourquoi fait-on ainsi ? Parce que l'avantage des *sprites*, des tuiles et des cartes de tuiles est leur faible consommation de mémoire et leur souplesse d'utilisation.
En plus de la carte des tuiles, la seule chose qui doit être stockée en mémoire est la position à laquelle les tuiles ou les *sprites* doivent être dessinés. Les animations ou les défilements sont également possibles de cette manière grâce à des mises à jour relativement simples des positions des *sprites*.
Pour le code source des démos, veuillez vous référer au dépôt GitHub correspondant [28] de Luke Wren. Les images présentées ici proviennent des applications *tiles_and_sprites*, *tiles_parallax* et *tiles*. Si vous disposez d'un Raspberry Pi Pico adéquat avec une sortie DVI, vous pouvez essayer les applications vous-même et manipuler le code source.

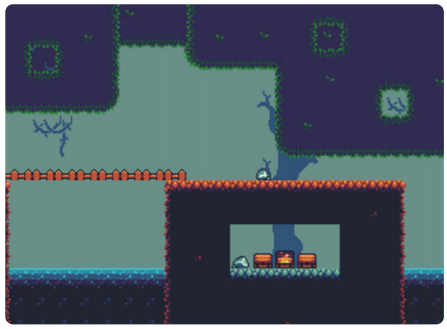

Figure 32. Démonstration de la plateforme 2D.

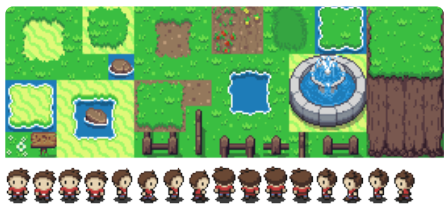

Figure 33. Cartes de tuiles (source : ArMM1998 / https://elektor.link/ZeldaTiles).

Un microcontrôleur comme artiste dessinateur ? Jusqu'à présent, nous n'avons vu que des démos – pas d'applications ou de jeux complets. Mais les démos en disent déjà assez pour que vous puissiez ajouter des images à vos projets. L'ESP32 et le Raspberry Pi Pico sont des modules peu coûteux qui permettent d'obtenir des sorties graphiques de qualité étonnante. Que ce soit *Pac-Man* ou *Tetris* via DVI, VGA ou vidéo composite, avec chaque génération, les capacités et les performances brutes des microprocesseurs augmentent. Des grands effets viennent à leur portée, comme le rendu 3D ou les images animées en couleur avec des *sprites*. Tout comme à l'époque des ordinateurs grand public des années 80, avec beaucoup de créativité et quelques astuces, il est possible de tirer des images intéressantes de ce matériel limité.

Les microcontrôleurs ont juste besoin d'une chorégraphie appropriée pour dessiner, comme tous les ingrédients décrits dans cet article. C'est à vous maintenant d'en faire usage pour créer des projets fascinants. Publiez vos créations sur Elektor Labs [29], et vous pourrez ainsi discuter de caractéristiques et de problèmes techniques avec d'autres membres d'Elektor et échanger des informations utiles.

220614-04 — VF : Helmut Müller

Des questions, des commentaires ?

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

Produits

- **Carte de démonstration Pimoroni Pico VGA (SKU 19769)**
  https://elektor.fr/19769
- **Raspberry Pi Pico (SKU 19562)**
  https://elektor.fr/19562
- **Prise DVI pour Raspberry Pi Pico (SKU 19925)**
  https://elektor.fr/19925
- **ESP32-PICO-Kit V4 (SKU 18423)**
  https://elektor.fr/18423

LIENS

[1] Bibliothèque vgax : https://github.com/smaffer/vgax
[2] Bibliothèque vgaxua : https://github.com/smaffer/vgaxua
[3] VGA-PacMan [allemand] : https://niklas-rother.de/artikel/vga-pacman
[4] Générateur vidéo Atmel ATmega avec SDRAM : http://tinyvga.com/avr-sdram-vga
[5] Tables de synchronisation VGA : http://tinyvga.com/vga-timing/640x480@60Hz
[6] ESP32 VGA : https://bitluni.net/esp32-vga
[7] Manuel de référence technique ESP32 [PDF] : https://elektor.link/ESP32TechMan
[8] Synchronisation du signal VGA : http://tinyvga.com/vga-timing
[9] ESP32Lib de bitluni : https://github.com/bitluni/ESP32Lib
[10] Sprite (infographie) - Wikipedia : https://en.wikipedia.org/wiki/Sprite_(computer_graphics)
[11] Bibliothèque FabGL : http://fabglib.org
[12] Tutoriels vidéo sur FabGL : https://elektor.link/YTFabGLTuts
[13] RPI-PVSA VGA, SD Card & Audio Demo Board for Pico - Projet KiCad [Zip] : https://elektor.link/RP2040VGAKiCad
[14] Conception de matériel avec RP2040 [PDF] : https://elektor.link/RP2040HardwareDesign
[15] Pimoroni Raspberry Pi Pico VGA Demo Base : https://elektor.fr/pimoroni-raspberry-pi-pico-vga-demo-base
[16] pico-playground : https://github.com/raspberrypi/pico-playground
[17] Bibliothèque pico_scanvideo : https://elektor.link/pico_scanvideoLib
[18] Bibliothèque PicoVGA : https://github.com/Panda381/PicoVGA
[19] MagPi : Bibliothèque graphique VGA pour le Raspberry Pi Pico [allemand] : https://elektor.link/MPPicoVGALib
[20] Mode 7 - Wikipedia : https://en.wikipedia.org/wiki/Mode_7
[21] Démonstration de l'interpolateur - GitHub : https://elektor.link/GHInterperlatorDemo
[22] DVI – Wikipedia : https://elektor.link/WPDVI
[23] Spécification DVI [PDF] : https://elektor.link/DVISpec
[24] Transmission de données à haute vitesse (TMDS) - Wikipedia : https://elektor.link/WPTMDS
[25] Codage 8b/10b - Wikipédia : https://en.wikipedia.org/wiki/8b/10b_encoding
[26] PicoDVD de Luke Wren : https://github.com/Wren6991/PicoDVI
[27] Mathias Claussen interviews Luke Wren, "DVI with the RP2040," Elektor 3-4/2023: https://elektormagazine.com/220575-01
[28] Démonstrations de PicoDVI, incluant tuiles et sprites « walker » - GitHub : https://elektor.link/GHPicoDVIDemos
[29] Elektor Labs : https://elektormagazine.fr/labs