Rapport de Projet Ingénierie :

de **Tristan Catteau** étudiant en **première année** de
Master d'Ingénierie des Systèmes Images et Sons
2022-2023

Conception d'un boîtier d 'up-scaling RGB to HDMI

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE
Université Polytechnique HAUTS-DE-FRANCE
dream AUDIOVISUEL&MULTIMÉDIA ©

Remerciements

Je tiens à remercier mon camarade de classe Théo Satorres, ainsi que M. Sébastien Gauthier, pour le prêt du matériel, sans qui on ne pourrait tester les différentes applications.

Je remercie également M. Philippe Thomin, mon professeur référent, pour son soutien, son expertise et son regard critique qui m’ont accompagné tout au long de la rédaction de ce rapport.

Résumé :

L’évolution rapide des écrans et des formats d’image complique l’utilisation des anciennes machines. La plupart des téléviseurs modernes effectuant une adaptation qui ne corresponde plus à l’attente lorsqu’on lui fournit un signal TV analogique. Afin de permettre aux possesseurs de machines plus anciennes de pouvoir les utiliser sans s’encombrer de différents moniteurs servant uniquement cette utilisation, de nombreux indépendants se sont lancés dans la création de boîtiers d’up-scaling, permettant de numériser le signal et de l’adapter à un écran 4K, par exemple. Le rapport ci-contre fait suite aux rapports effectués sur le même sujet par les étudiants des années antérieurs, dont vous trouverez un accès en bibliographie. Il explique une présentation générale du projet suivi d’une explication sur les premiers tests effectués sur la machine. Il sera surtout tourné sur la programmation en langage assembleur du Raspberry Pi.

Abstract :

The rapid evolution of screens and image formats is complicating the use of older machines. Most modern televisions no longer adapt to what is expected when supplied with an analogue TV signal. In order to enable owners of older machines to use them without having to worry about having different monitors that are only used for this purpose, many independent companies have started to create up-scaling boxes that digitise the signal and adapt it to a 4K screen, for example. The report opposite follows on from the reports produced on the same subject by students in previous years, which can be found in the bibliography. It gives a general presentation of the project, followed by an explanation of the first tests carried out on the machine. It will focus mainly on programming the Raspberry Pi in assembly language.

Table des matières

Introduction

L'up-scaling est une technique de traitement de l’image qui vise à améliorer la résolution d’une image ou d’une vidéo. Avec le renouvellement de l’écran qui soutient la haut-qualité des images, certains ordinateurs rétros et leurs formats d’affichage vidéo ne sont progressivement plus supportés par les écrans. L'up-scaling permet de convertir le format du signal, et d’augmenter la résolution.

Afin de donner la vitalité aux vieux ordinateurs. Il existe une variété de solutions d'up-scaling disponibles sur le marché, mais ayant un coût élevé. Nous espérons trouver une méthode rentable pour atteindre cet objectif, sur le principe d’assurer des effets d’affichage de haute qualité, c’est-à-dire des bords clairs et un positionnement stable.

L’une des solutions choisis et traité dans ce rapport est le projet RGBtoHDMI conçue par David Banks & Ian Bradbury, afin de donner la vitalité aux vieux ordinateurs [1]. Par la suite, le projet a été publié sur Github afin de former un projet mûr et complet ; De nombreux appareils sont désormais pris en charge.

Notre objectif est de pouvoir le reproduire et de le simplifier à notre cas d’utilisation, à un coût réduit. Nous verrons alors dans ce rapport le matériel à disposition, le fonctionnement du projet RGBtoHDMI, puis nous terminerons sur des tests pour le traitement en langage assembleur.

1. L’up-scaling

L'up-scaling, également connu sous le nom de mise à l’échelle, est une technique utilisée pour augmenter la résolution[1] d’une image, d’une vidéo ou d’un autre type de contenu numérique.

Lorsque l’on effectue un up-scaling, on augmente la résolution d’une image ou d’une vidéo en ajoutant des pixels[2] supplémentaires à l’image existante. Cela peut être utile lorsque l’on souhaite améliorer la qualité visuelle d’un contenu basse résolution ou lorsque l’on veut le faire correspondre à la résolution d’un écran ou d’un support de sortie spécifique.

Le problème étant comment traiter les nouveaux pixels qui sont ajoutés. Doit-on simplement copier le pixel voisin ? Doit-on faire une moyenne ? Pour cela, il existe différente type d’upscalings, utilisant des traitements différents.

1.1. L’up-scaling par interpolation

L’up-scaling basé sur l'interpolation[3], ou aussi appelé us-scaling basique, est la méthode d'up-scaling la plus simple et la plus courante. Elle consiste à estimer les valeurs des pixels manquants en utilisant des techniques d’interpolation mathématique, telles que l’interpolation bilinéaire ou bicubique. Ces méthodes utilisent les pixels environnants pour calculer les nouvelles valeurs de pixels. Cependant, l’interpolation seule peut entraîner une perte de détails et une apparence floue.

L’idée fondamentale de l'interpolation est de prédire les valeurs des nouveaux pixels en se basant sur les pixels existants environnants. Il existe plusieurs méthodes couramment utilisées pour l'interpolation lors de l'up-scaling, nous allons voir les deux cités plus haut.

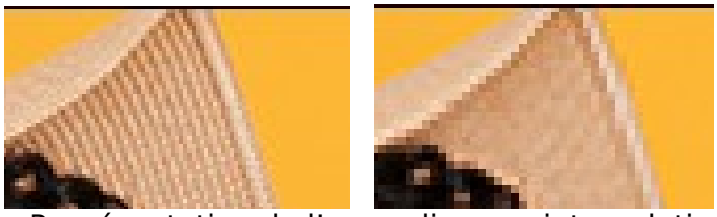


Figure 1 : Représentation de l’up-scaling par interpolation : à gauche, l’image original, à droite, interpolation bicubique

L’interpolation bilinéaire est une méthode d’interpolation linéaire utilisée pour estimer les valeurs des nouveaux pixels. Pour chaque nouveau pixel, cette méthode considère les quatre pixels les plus proches dans l’image d’origine et effectue une interpolation linéaire pour calculer sa valeur. L’interpolation bilinéaire est relativement simple à implémenter, mais elle peut entraîner une perte de détails et donner une apparence légèrement floue aux contours.

L’interpolation bicubique est une méthode d’interpolation plus avancée qui utilise une interpolation polynomiale. Elle est similaire à l’interpolation bilinéaire, mais considère un carré de pixels environnants pour calculer les valeurs des nouveaux pixels. L’interpolation bicubique donne généralement de meilleurs résultats que l’interpolation bilinéaire, car elle permet de capturer des variations plus fines dans les dégradés de couleur et de conserver davantage de détails. On peut voir un résultat en figure 1.

Ces méthodes d’interpolation peuvent être utilisées individuellement ou combinées pour effectuer un up-scaling. L’idée générale est de déterminer les emplacements des nouveaux pixels dans l’image de sortie et d’estimer leurs valeurs à l'aide des techniques d’interpolation appropriées.

Cependant, l'up-scaling par interpolation seule ne peut pas ajouter de nouveaux détails réels à l’image. Il peut améliorer la résolution globale de l'image, mais il ne peut pas restituer les informations qui n’étaient pas présentes dans l’original.

1.2. L’up-scaling par voisinage

Cette méthode utilise les informations des pixels voisins pour estimer les nouvelles valeurs de pixels. Au lieu de se limiter à l’interpolation linéaire ou cubique, cette approche peut prendre en compte des structures plus complexes dans l’image pour obtenir des résultats plus précis. Par exemple, des algorithmes[4] basés sur la recherche du plus proche voisin peuvent être utilisés pour identifier les régions similaires et les répliquer pour augmenter la résolution.

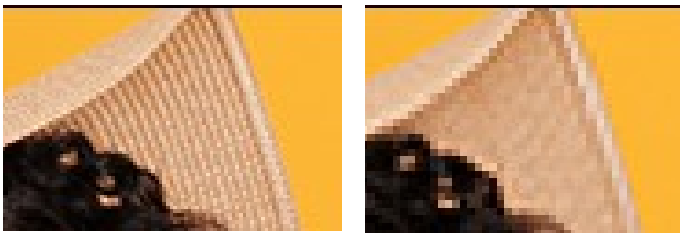


Figure 2 : Représentation de l’up-scaling par voisinage : à gauche, l’image original, à droite, l’up-scaling par voisinage

L'up-scaling par voisinage peut être plus efficace que l’interpolation pour préserver les détails locaux et les textures dans l’image. Cependant, il est important de noter que cette méthode peut également présenter des limitations, notamment lorsque les structures locales de l’image sont complexes ou lorsque les informations des pixels voisins ne sont pas suffisantes pour estimer les nouvelles valeurs de pixels avec précision (les rayures du chapeau dans la figure 2).

1.3. L’uspcaling par l’intelligence artificielle

Il s’agit du procédé le plus récent. Lorsque le procédé d’up-scaling implique une intelligence artificielle, la démarche est plus poussée, des algorithmes poussés sont mis à contribution pour analyser les images et recréer des pixels.

L'objectif principal de cette technique est de convertir une image dans une définition supérieure en essayant d’atteindre une image la plus proche possible de ce qu’aurait été l’image initiale si elle avait été captée nativement dans cette définition supérieure.

Mais cette technologie offre aussi d’autres possibilités comme : « réparer » des défauts visuels, dus par exemple à une compression avec perte ou au format entrelacé[5] de la vidéo d’origine, elle permet aussi d’augmenter le nombre d’images par seconde dans une vidéo en recréant les images manquantes. On peut voir les résultats en figure 3, tirer du site de NVIDIA[7] [2]

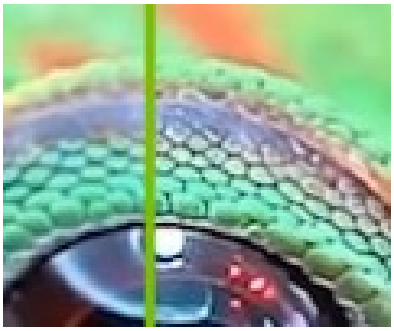


Figure 3 : Up-scaling par intelligence artificielle

Le système se base notamment sur l'apprentissage, pour « s'entraîner » il reçoit par exemple des images hautes définition volontairement diminuées en définition ou dégradées, classées par catégories (texture bâtiment, eau, etc.). Les algorithmes vont faire des tests complexes statistiques pour recréer les pixels manquants et arriver à l'image originale haute définition. Le système va alors « retenir » les moyens qu'il a utilisés pour revenir à l'image originale et va s'en inspirer pour up-scaler d'autres nouvelles images, contenant le même type d'élément.

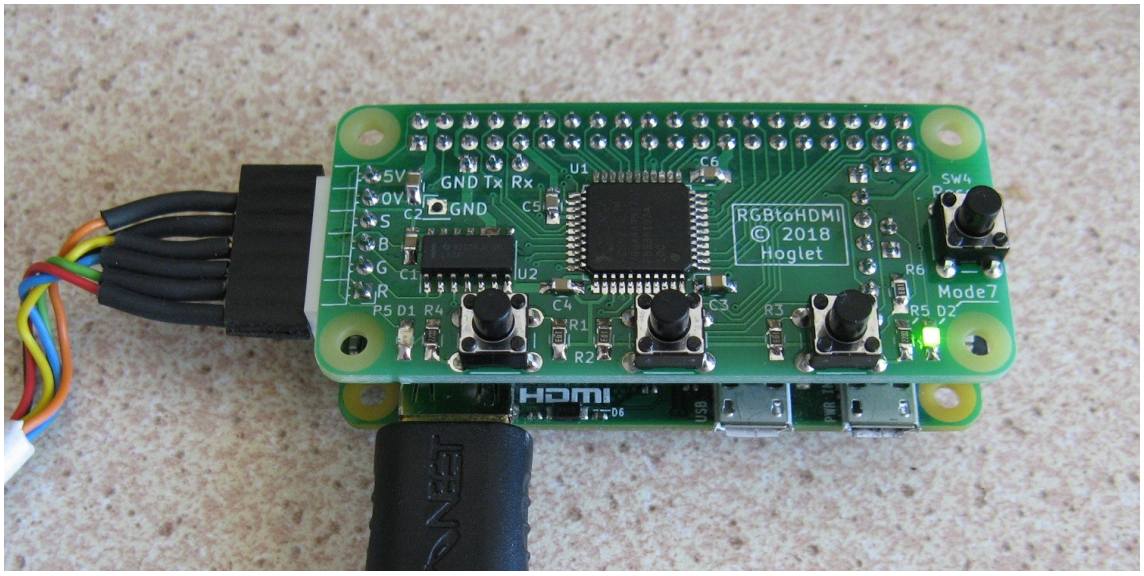


Figure 4 : Boîtier du projet RGBtoHDMI

2. Présentation du projet RGBtoHDMI

Nous avons pu voir les différentes techniques d'up-scaling, dont chacun à leurs avantage et leurs défauts. Notre objectif est de concevoir un boîtier d'up-scaling à moindre coût. Pour cela, nous nous intéresserons aux projets RGBtoHDMI qui est l'une des solutions les plus appropriés.

L'interface RGBtoHDMI est un convertisseur vidéo HDMI[7] de haute qualité pour les ordinateurs rétros. L'interface comprend un Raspberry Pi[8] Zero et un ≪chapeau≫ spécialement conçu contenant un petit CPLD[9] (figure 4). Le micrologiciel personnalise sur le Raspberry Pi, en conjonction avec le CPLD, est capable d'échantillonner[10] avec précision une large gamme de formats vidéo non standards, généralement RVB[11], d'où son nom, pour donner un rendu au pixel prés. Il a été conçu a l'origine pour le BBC Micro, mais a maintenant été étendu pour prendre en charge une large gamme de systèmes grâce a l'utilisation de profils. [3]

Le micrologiciel personnalisé sur le Raspberry Pi, en conjonction avec le CPLD, est capable d'échantillonner correctement chacun des modes vidéo pris en charge pour donner un rendu au pixel près. La sortie HDMI est verrouillée sur l'entrée afin qu'il n'y ait pas de chutes d'images, de répétitions ou de déchirures[12] et elle a également un faible décalage d'environ 4 millisecondes [4].

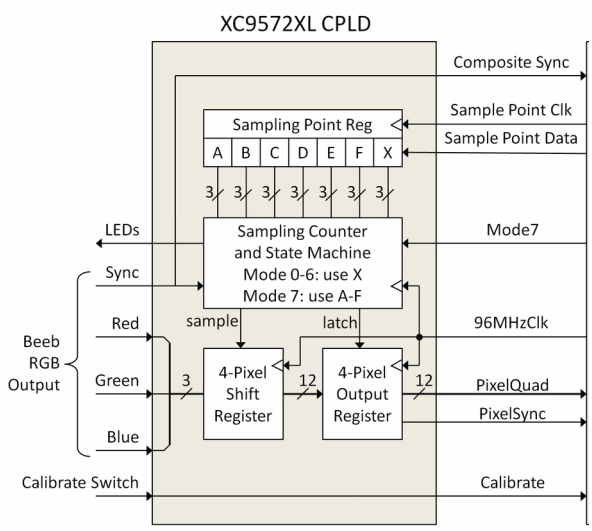


Figure 5 : Structure du CPLD

En raison de la façon dont la vidéo est échantillonnée et re-quantifiée[13], lorsque tout est correctement configuré, la sortie est totalement exempte de bruit, même en passant par une étape analogique, et cela donne l'impression que la sortie du matériel réel provient d'un émulateur[14].

Afin de mieux comprendre et de réaliser nos conditions de travail, nous allons nous intéresser aux équipements nécessaires à son fonctionnement, adaptés à notre cas.

2.1. Équipements matériels

2.1.1. CPLD XC9572XL High-Performance

Le terme « CPLD » désigne un type de circuit intégré[15] qui permet aux utilisateurs de le configurer et de le reconfigurer pour réaliser des fonctions logiques spécifiques. Le CPLD XC9572XL (figure 5) est conçu pour offrir une haute performance et une grande flexibilité pour la conception de circuits numériques. Ce CPLD comporte des cellules de programmation et de mémoire, ainsi que des fonctions de délai de synchronisation et de correction d'erreur. Cette puce fonctionne à des fréquences jusqu'à 100 MHz et offre une flexibilité de programmation pour s'adapter aux besoins changeants de l'application.

	Raspberry Pi Modèle B	Raspberry Pi Zéro
System-on-a-chip(SoC)[16]	Broadcom BCM2835	Broadcom BCM2835
CPU[17]	700 MHz ARM11 ARM1176JZF-S core	1000MHz Low Power ARM1176JZF-S
GPU[18]	Broadcom VideoCore IV, OpenGL ES 2.0,OpenVG 1080p30 H.264 high-profile encode/decode, 250 MHz	Broadcom VideoCore IV
Memoire (SDRAM)[19]	256 MiB	512 MiB
Ports USB[20]	1 USB 2.0	1 Micro USB OTG
Sortie Vidéo	Vidéo composite | RCA Composite, HDMI (jamais en même temps)	HDMI, Vidéo composite via un en-tête à 2 broches non soudées
Sortie Audio	Connecteur TRS | 3.5 mm jack, HDMI	Multi-Channel HD Audio vers HDMI
Puissance nominale	300 mA, (1.5 W)	160mA rating
Source de courant	5 V (DC) via le Micro USB type B ou le GPIO	5 V (DC) via le Micro USB type B ou le GPIO

Figure 6 : Tableau des caractéristiques des Raspberry Pi étudiés

Le CPLD contient un registre[21] à décalage a 4 pixels qui est utilise pour collecter quatre échantillons de pixels successifs (appelés Pixel Quad). Le registre à décalage est suivi d'un registre de sortie afin que la valeur Pixel Quad vue par le Pi reste stable aussi longtemps que possible. Chaque fois qu'une nouvelle valeur est chargée dans le registre de sortie, la sortie Pixel Sync est basculée. Ceci est utilise par le Raspberry Pi pour déterminer quand une nouvelle valeur Pixel Quad est disponible, a écrire dans le tampon de trame.

Les points d'échantillonnage de pixels précis sont contrôles par la machine d'état d'échantillonnage. À la fin de chaque impulsion de synchronisation horizontale, un compteur est charge avec une grande valeur négative puis commence à compter. Lorsque ce compteur atteint zéro, l'échantillonnage commence.

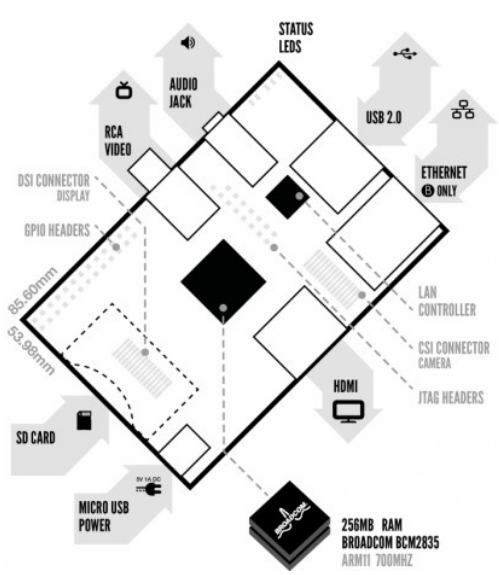


Figure 7 : Synoptique du Raspberry Pi Model B

2.1.2. Raspberry Pi

Le Raspberry Pi est un nano-ordinateur[22] monocarte[23] à processeur ARM[24] de la taille d'une carte de crédit conçu par des professeurs du département informatique de l'université de Cambridge dans le cadre de la fondation Raspberry Pi.

Il est initialement fourni nu, c'est-à-dire la carte mère seule, sans boîtier, câble d'alimentation, clavier, souris ni écran, dans l'objectif de diminuer les coûts et de permettre l'utilisation de matériel de récupération.

Nous allons nous intéressez dans notre cas à deux Raspberry Pi précis, le Raspberry Pi Zéro, utilisé par le projet, et le Raspberry Pi Model B, celui dont nous avons a disposition. Vous pouvez retrouver un tableau des caractéristiques en figure 6, et plus d'information sur le site elinux.org [5].

Le Raspberry Pi utilise généralement un système d'exploitation basé sur Linux[25] appelé Raspbian[26] qui est spécialement conçu pour les Raspberry Pi. Il existe également d'autres distributions Linux disponibles, ainsi que des versions de Windows 10 IoT Core[27].

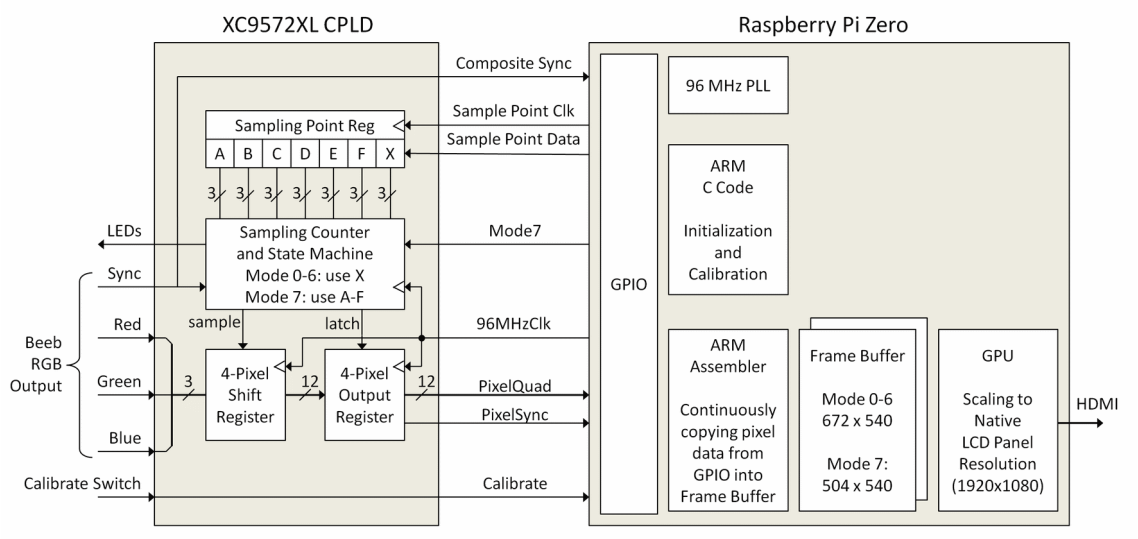


Figure 8 : Synoptique du Raspberry Pi Zero

Dans RGBtoHDMI, au niveau du Raspberry Pi, en figure 8, tous les logiciels fonctionnent en « bare metal[28] », ou programme à nu, de sorte qu'aucune interruption ne provoque des problèmes d'échantillonnages. Le programme est écrit dans un mélange de C[29] et d'assembleur ARM. Tout le code non critique dans le temps, comme les phases d'étalonnage, est écrit en codage C. La partie critique dans le temps, copie les valeurs Quad Pixel du GPIO[30] vers le Frame Buffer[31], est écrite en assembleur ARM.

Avec le CPLD, le signal RGB reçu est propre, stable et a une fréquence donnée, soit 96 MHz. Ensuite, il est traité par une fiche d'assembleur ARM, pour convertir en un format destiné à être stocké dans un frame buffer, où les images attendent d'être affichée à l'écran. De côté de GPU, il est utilisé pour mettre à l'échelle le Frame buffer afin qu'il corresponde à la résolution de sortie. Lorsque vous connectez un écran externe au Raspberry Pi, le GPU est informé de la résolution de l'écran et utilise les informations pour redimensionner le frame buffer pour correspondre à cette résolution. Selon le mode, le frame buffer fournit 2 solutions : 672 × 540 × 4 bits/pixel (Modes 0-6) et 504 × 540 × 4 bits/pixel (Mode 7). L'échelle « 4 bits » est 2 × 2 pixels, elle signifie le grossissement. Elle réalise alors un up-scaling par interpolation.

Notre objectif est de reproduire ce programme en bare métal. Pour cela nous devons comprendre comment fonctionne le langage machine, et plus précisément, le langage assembleur ARM.

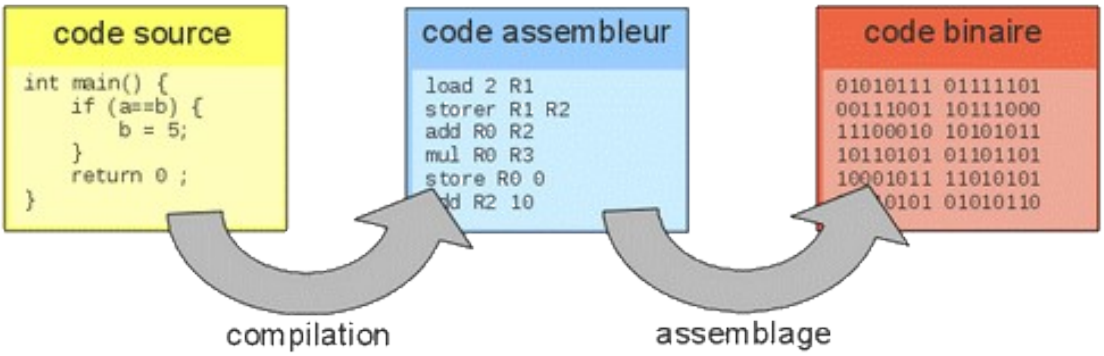


Figure 9 : Illustration des différents types de langages informatiques

3. Le langage assembleur ARM

3.1. Fonctionnement général

Le langage machine est un langage de bas niveau, c'est-à-dire un langage compris par des processeurs et qui se compose d'instructions binaires, représentées par des combinaisons de 0 et de 1, qui sont exécutées directement par le matériel. Il est différent d'un langage de haut niveau, comme le C, qui sont plus facilement compris par l'homme. Le passage d'un haut niveau à un bas niveau s'effectue à l'aide de compilateurs[32] ou d'interprètes[33].

Le langage assembleur est un langage de programmation de bas niveau qui permet aux programmeurs d'écrire des instructions compréhensibles par un processeur spécifique. Il est souvent considéré comme la forme la plus élémentaire de programmation, car il est directement lié à l'architecture matérielle d'un ordinateur.

Le langage assembleur utilise des codes mnémoniques (symboles) pour représenter les instructions et les opérations qui seront exécutées par le processeur. Chaque instruction assembleur correspond généralement à une opération spécifique, telle que le chargement de données en mémoire, l'addition, la soustraction, la comparaison, le saut conditionnel, etc. Une illustration des types de langages se trouve en figure 9.

Un processeur ARM est un composant présent dans de nombreux dispositifs électroniques qui exécute les instructions machine des programmes informatiques. Développées par ARM Ltd depuis 1983 et introduites à partir de 1990 par Acorn Computers. L'architecture ARM est le

fruit du travail de Sophie Wilson. Dans notre cas, il s'agit du langage assembleur utilisé par le Raspberry Pi. Il peut être utilisé aussi sur certain OS Linux. Une note sur le langage ARM est disponible en annexe n°1.

Comme nous ne travaillons pas directement sur le Raspberry Pi, on doit passer par le cross-assemblage (cross-assembly en anglais). Il s'agit d'une technique qui consiste à assembler un code source écrit dans un langage d'assemblage spécifique à une architecture sur un système différent qui utilise une architecture différente.

Dans notre cas, nous allons utiliser cette technique, car nous travaillons sur un autre système. La procédure est expliquée dans le chapitre suivant.

3.2. Tests effectués

On va ici expliquer comment faire fonctionner le Raspberry Pi en langage assembleur.

3.2.1. Environnement de travail

Pour pouvoir implémenter du code assembleur, il nous faut un ordinateur, de préférence sous Linux, car il permettra d'utiliser des utilitaires propres à cet OS[34] et de ne pas être restreint en son utilisation comme Windows ou Mac qui à une protection élevée. Pour ma part, travaillant sous Windows, j'ai installé sur mon ordinateur portable une machine virtuelle[35] qui fait tourner partiellement un Ubuntu[36], en version minimale, ce qui est suffisant pour ce projet.

J'aurai besoin aussi de télécharger plusieurs paquets[37], notamment un compilateur pour le langage ARM, (gcc-arm-none-eabi), qui va pouvoir me construire mon code en binaire. J'ai installé aussi le paquet « make » (en ligne de commande : $sudo apt install make) qui permet de construire différents fichiers, dans notre cas des noyaux[38], ou kernel en anglais, qui sera lui aussi lu par le Raspberry Pi. Il transformera nos fichiers en langage assembleur en binaires, stocké dans le noyau.

Pour le matériel, j'utilise un Raspberry Pi Model B, qui est une version plus ancienne et moins puissante que le modèle utilisé par le projet initial, mais qui est satisfaisant pour les tests apportés. Tout le code sera mis dans une carte SD, formater au format FAT32[39], afin d'éviter les problèmes de lectures par le Raspberry Pi.

3.2.2. Boot du Raspberry Pi

Pour pouvoir utiliser le Raspberry Pi en bare metal, il faut d'abord le booter, c'est-à-dire le faire démarrer au programme le plus bas possible. Le Raspberry Pi à une façon particulière de lire les données : Lorsque l'on branche l'alimentation, seul le GPU est actif. C'est lui qui va démarrer tout le système avec les fichiers correspondant. Pour ce faire, nous aurons besoin de plusieurs fichiers à mettre sur la carte SD :

- bcm278-rpi-b.dtb (Permet de définir sur quels systèmes nous somme, soit ici le model B)

- bootcode.bin (le premier fichier lu par la ROM[40], active le SDRAM et charge le start.elf, qui permet de booter le système)

- start.elf (c’est le firmware du GPU. Il allume le CPU et charge les autres fichiers)

- fixup.dat (Permet d’allouer 1GB de mémoire)

- kermel.img (l’application bare metal à lire)

Une fois ces fichiers mis sur la carte SD, récupérable sur internet, on insère la carte dans le Raspberry, on branche un câble HDMI afin de vérifier son bon fonctionnement et on peut enfin brancher l’alimentation. On va s’interesser au LED sur la figure 8. La LED PWR (power, qui sert à indiquer qu’il reçoit de l’alimentation) est allumé, la LED ACT (action) va clignoter, il va donc s’initialiser. Il faudra attendre une dizaine de secondes pour qu’elle reste allumée, ce qui signifie qu’elle est prête à la configuration, et que le cœur peut-être prêt à être installé. Dans notre cas, nous ne souhaitons pas installer d’OS puisque l’on veut rester au plus bas niveau. Nous pouvons alors débrancher l’alimentation, puis le HDMI et enfin la carte. Notre Raspberry Pi est maintenant prêt à faire tourner des programmes assembleurs.

Figure 8 : Position des LED PWR et ACT

3.2.3. Premier programme : clignoter une LED

Nous pouvons comprendre le programme selon le schéma suivant, en figure 9. Le code peut se trouver en annexe n°2.

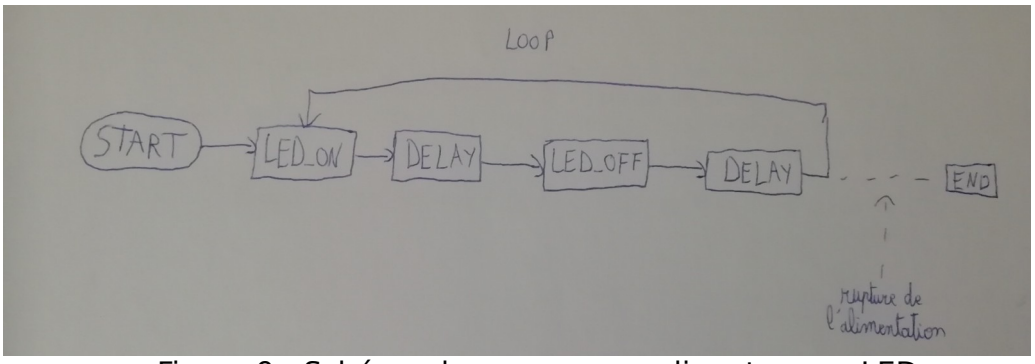


Figure 9 : Schéma du programme clignoter une LED

Lors du démarrage, on définit les pins correspondants comme une sortie, il fournira le courant pour éclairer la LED[41]. Pour ce faire on alloue des variables contenant des registres, trouvable dans la documentation, en annexe n° 2. Il va alors commencer à s'allumer, effectuer un délai, pour ensuite l'éteindre, ré effectués un délai et reboucler à l 'infini, tant qu'il y a de l'alimentation.

3.2.4. Deuxième programme : allumer/éteindre une LED par un bouton

Nous pouvons comprendre le programme selon le schéma suivant, en figure 10. Le code peut se trouver en annexe n°3.

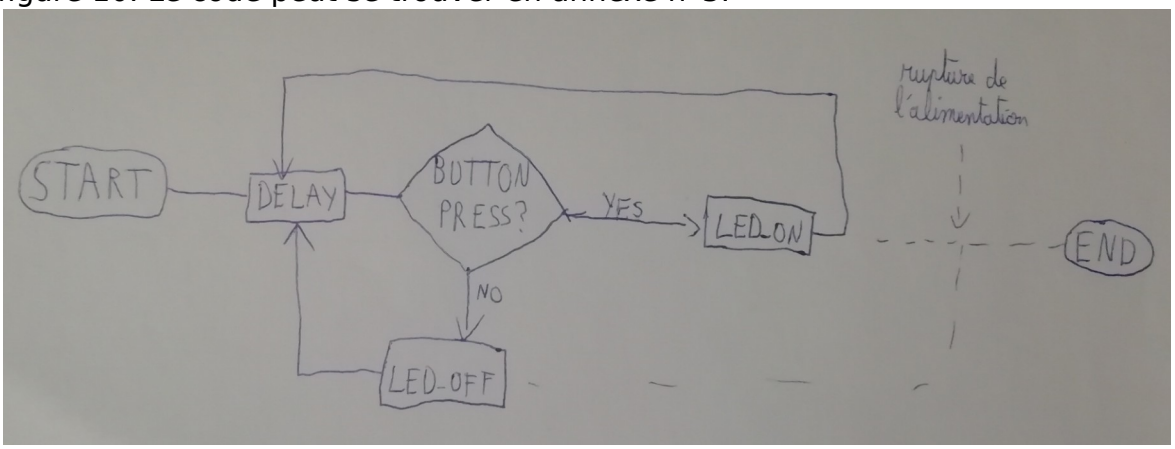


Figure 10 : Schéma du programme allumer/éteindre une LED par un bouton

On va attendre qu’il reçoive une tension au pin du GPIO configurer en entrée pour émettre un signal au pin de sortie qui fera allumer la LED. S’il y a pas de pression, alors la LED reste éteinte.

Conclusion

Nous avons pu voir le fonctionnement global d'un programme ARM. L'objectif étant de produire un programme qui permettrait la copie des valeurs Quad Pixel du GPIO vers le Frame Buffer. On pourrait aussi aller plus vite en comprenant comment fonctionne le GPU qui initialise tous les composants. Cela demande de faire du reverse engineering[42], mais comme David Banks l'a déjà effectué dans son projet RGBtoHDMI, il faudra plutôt effectuer du benchmarking[43]pour comprendre son utilisation.

J'ai voulu mesurer les temps de cycle sur les programmes précédents, afin de donner une idée à quelle vitesse pourrait tourner le CPU. En se renseignant sur internet [7][8], on remarque que les instructions sont pipelinés, c'est-à-dire que le traitement de chaque instruction se fait en parallèle les uns aux autres. Il est donc pas possible d'utiliser cette méthode. On doit alors mesurer sur le programme avec un fréquencemètre, mais par manque de temps, il sera effectué l'année prochaine.

J'ai appris par ailleurs que nous testerons plutôt notre signal d'entrée via un ATARI mega ST4[44]. Cela demande de la documentation à faire pour comprendre cette technologie. On pourra aussi mesurer les timings de ce signal et chercher les niveaux propres à l'aide des comparateurs de tension. Enfin si tout se passe sans encombre, on pourra effectuer un essai du signal ATARI sur le GPIO du Raspberry Pi.

Ce projet fut fort enrichissant, mais par manque de temps, nous nous sommes arrêtés à de simples tests. Le projet a encore beaucoup d'avenir, ce qui est prometteur pour la continuité du projet et j'espère pouvoir voir un jour le fonctionnement finale de celui-ci.

Annexe n°1 : Note sur le langage ARM

Le langage ARM fonctionne par la manipulation de registre. On peut effectuer des opérations binaires et les stockés dans des registres, qui sera lu par le CPU puis exécuter. Cette architecture repose sur 17 registres :

•des registres généraux : **R0** à **R12**,

•un registre « pointeur de pile » : **SP** (possiblement appelé R13),

•un registre dit « de lien » : **LR** (possiblement appelé R14),

•un registre « pointeur de programme » : **PC** (possiblement appelé R15),

•un registre de statut : **xPSR** (qui se décline sous 3 sous-noms **APSR**, **EPSR** ou **IPSR** selon le besoin qu'on en a).

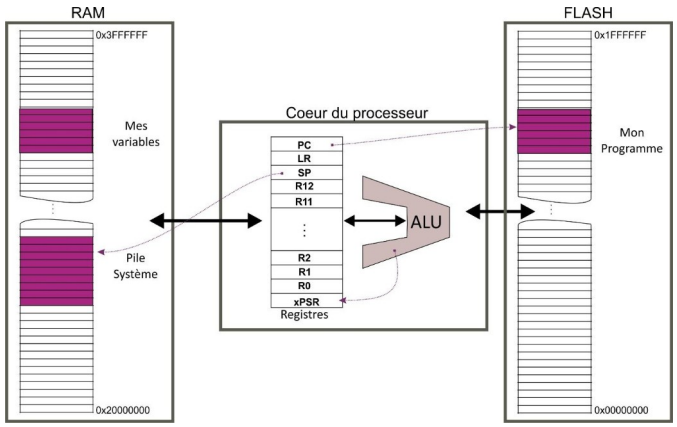
Basée sur un jeu d'instructions RISC (Reduced Instruction Set Computer), dont la liste se trouve en webographie, l'ALU exécute les différentes opérations pour lequel elle est conçue. Ce sont toujours des opérations basiques dont les principales sont :

•l'arithmétique simple sur des entiers (addition/soustraction, multiplication/division),

•des opérations logiques diverses (comparaison, ET, OU, etc.).

Outre l'exécution de l'instruction en elle-même l'ALU peut mettre à jour des fanions contenus dans le registre **xPSR**.

Pour plus d'informations, veuillez aller sur la vidéo effectuée par freeCodeCamp.org [6].



Annexes n°2 : Programme clignoter la LED

```
.global _start

.equ GPIO_BASE, 0x20200000
.equ GPFSEL2, 0x08

.equ GPIO_22_OUTPUT, 0x40;//# 1 << 6

.equ GPFSET0, 0x1c
.equ GPFCLR0, 0x28

.equ GPIOVAL, 0x400000 ;//# 1 << 22

_start:
        ;//# base of our GPIO structure
        ldr r0, =GPIO_BASE

        ;//# set the GPIO 22 function as output
        ldr r1, =GPIO_22_OUTPUT
        str r1, [r0,#GPFSEL2]

        # set counter
        ldr r2, =0x400000

loop:
        # turn on the LED
        ldr r1, =GPIOVAL ;//# value to write to set register
        str r1, [r0, #GPFSET0] ;//# store in set register

        # wait for some time, delay
        eor r10, r10, r10
        delay1:
                add r10, r10, #1
                cmp r10, r2
                bne delay1

        # turn off the LED
        ldr r1, =GPIOVAL ;//# value to write to set register
        str r1, [r0, #GPFCLR0] ;//# store in set register

        # wait for some time, delay
        eor r10, r10, r10
        delay2:
                add r10, r10, #1
                cmp r10, r2
                bne delay2
        b loop
```

Annexes n°3 : Programme allumer/éteindre par un boutton

```
.section .text
.global _start

.equ GPIO_BASE, 0x20200000
.equ GPFSEL1, 0x04
.equ GPFSEL2, 0x08

.equ GPIO_22_OUTPUT, 0x40;//# 1 << 6
.equ GPIO_17_INPUT, 17

.equ GPFSET0, 0x1c
.equ GPFCLR0, 0x28
.equ GPEDS0, 0x34

.equ MASK_INPUT, 0xFFFFF1FF
.equ MASK_DETECT, 0xFFFDFFFF

.equ GPIOVAL, 0x400000 ;//# 1 << 22
_start:

        ;//# base of our GPIO structure
        ldr r0, =GPIO_BASE

        ;//# set the GPIO 22 function as output
        ldr r1, =GPIO_22_OUTPUT
        str r1, [r0,#GPFSEL2]

        ;//# set the GPIO 17 function as input
        ldr r1, =MASK_INPUT
        str r1, [r0,#GPFSEL1]

        # set counter
        ldr r2, =0x000001

loop:
        # wait for some time, delay
        eor r10, r10, r10
        delay1:
                add r10, r10, #1
                cmp r10, r2
                bne delay1
        bl get_input
        cmp r5, #1
        bne ledoff
        beq ledon
        b loop

```

```
47
48 get_input:
49         mov r4, #1
50         lsl r4, #17
51         ldr r3, [r0, #GPEDS0]; //# on charge l'etat du bouton
52         tst r3,r4
53         moveq r5, #0
54         movne r5, #1
55         mov pc, lr
56 ledon:
57         # turn on the LED
58         ldr r1, =GPIOVAL ;//# value to write to set register
59         str r1, [r0, #GPFSET0] ;//# store in set register
60         b loop
61 ledoff:
62         # turn off the LED
63         ldr r1, =GPIOVAL ;//# value to write to set register
64         str r1, [r0, #GPFCLR0] ;//# store in clr register
65         b loop
```

Lexique

(1) résolution : Nombre de pixels ou de points par unité de longueur, exprimé en pixels par pouce [ppp] pour un fichier image et en points par pouce [PPP] pour une imprimante ou un scanner. (La résolution caractérise, avec la définition, la qualité d’une image numérique.)

(2) pixel : Le plus petit élément constitutif d'une image produite ou traitée électroniquement, définie par sa couleur et sa luminosité.

(3) interpolation : Opération consistant à déterminer, à partir d'une série statistique succincte aux valeurs trop espacées, de nouvelles valeurs correspondant à un caractère intermédiaire pour lequel aucune mesure n'a été effectuée.

(4) algorithme : la description d'une suite d'étapes permettant d'obtenir un résultat à partir d'éléments fournis en entrée.

(5) entrelacé : est une technique d’affichage d’image de télévision, d’informatique ou d’appareils vidéo. Elle consiste à transmettre alternativement deux demi-images ou « trames », l’une comprenant les lignes numérotées impaires, l'autre celles numérotées paires.

(6) Nvidia Corporation : est une entreprise américaine spécialisée dans la conception de processeurs graphiques, de cartes graphiques et de puces graphiques pour PC et consoles de jeux (Xbox, PlayStation 3, Nintendo Switch).

(7) HDMI : (High Definition Multimedia Interface) est une norme internationale permettant de connecter des appareils électroniques haute définition grand public et des ordinateurs.

(8) Raspberry Pi : Le Raspberry Pi est un nano-ordinateur monocarte à processeur ARM de la taille d'une carte de crédit.

(9) CPLD : (complex programmable logic device, circuit logique programmable complexe), utilisent des "macrocellules" logiques, composées d'un réseau combinatoire de portes ET et OU afin d'implémenter des équations logiques. Des bascules sont disponibles seulement dans les blocs d'entrée-sortie. Un composant contient de quelques dizaines à quelques centaines de macrocellules.

(10) l’échantillonnage : Mesure, à intervalles réguliers, de la valeur de l'amplitude d’un signal continu, en vue d’établir une représentation

approchée de ce signal. (L’échantillonnage est le plus souvent suivi d’une quantification des valeurs trouvées et de leur représentation par un signal numérique.)

(11) RBV : est un mode colorimétrique utilisé en informatique et qui permet de coder une couleur. Le sigle RVB signifie Rouge-Vert-Bleu. On emploie aussi le terme RGB (pour Red-Green-Blue en anglais).

(12) déchirure : est un artefact vidéo dans laquelle la partie supérieure de l'écran présente une autre image de la vidéo à la partie inférieure. Cela est plus visible au cours de scènes contenant des mouvements rapides. Il existe peut-être une ligne horizontale perceptible au niveau du point d'intersection des deux cadres.

(13) quantification : Propriété d'une grandeur physique dont l'ensemble des valeurs numériques possibles est restreint à un ensemble de valeurs discrètes ; opération permettant d'aboutir à la détermination de ces valeurs.

(14) émulateur : Ensemble des outils permettant à un ordinateur de copier le fonctionnement d’une autre machine.

(15) circuit intégré : aussi appelé puce électronique, est un composant électronique, basé sur un semi-conducteur, reproduisant une ou plusieurs fonctions électroniques plus ou moins complexes, intégrant souvent plusieurs types de composants électroniques de base dans un volume réduit (sur une petite plaque), rendant le circuit facile à mettre en œuvre.

(16) Soc : Un system on a chip est un circuit intégré qui rassemble les principaux composants d'un ordinateur sur une seule puce électronique.

(17) GPU : GPU est l’acronyme du terme *graphical processing unit*, qui signifie en français « unité de traitement graphique ». On le désigne plus communément par processeur graphique. Il s’agit d’une unité de calcul dédiée aux données graphiques d’un ordinateur qui travaille en parallèle avec le CPU (*central processing unit*, le processeur central d’un ordinateur). Le GPU peut se trouver sur le même circuit intégré que le microprocesseur, sous forme de puce ou de circuit intégré sur la carte mère d’un ordinateur ou sur une carte graphique dédiée. Cette carte graphique peut être interne ou externe à l’ordinateur.

(18) CPU : Central Proccessing Unit. Le CPU désigne un processeur ou microprocesseur principal d'un ordinateur, on parle aussi d'unité de traitement.

(19) SDRAM : Synchronous Dynamic Random Access Memory (en français, mémoire dynamique synchrone à accès aléatoire) est un type particulier de mémoire vive dynamique ayant une interface de communication synchrone.

(20) USB : Universal Serial Bus, définit une norme se rapportant à un bus informatique. Il s'agit d'un dispositif de transmission de données qui permet de connecter des périphériques informatiques à un ordinateur ou à tout autre appareil équipé d'un port USB. Depuis son apparition à la fin des années 1990, l'USB a connu différentes évolutions, allant de l'USB 1.0 à l'USB 2.0 puis USB 3.0, en proposant simultanément un débit de 1,5 Mbit/s à 10 Gbit/s.

(21) registre : est un emplacement de mémoire interne à un processeur. Les registres se situent au sommet de la hiérarchie mémoire : il s'agit de la mémoire au meilleur temps d'accès, mais dont le coût de fabrication est le plus élevé, car la place dans un microprocesseur est limitée. Leur nombre dépasse les quelques Mo 2 à 3 sur la dernière génération de processeur en cache de niveau 2 et 3.

(22) nano-ordinateur : est plus généralement utilisé pour désigner des ordinateurs dont la taille est comparable à celle d'une carte de crédit, et à vocation soit d'enseignement, soit de composant d'un système spécialisé (surveillance, domotique, contrôle de petits engins…).

(23) monocarte : ordinateur complet construit sur un circuit imprimé, avec un ou plusieurs microprocesseurs, de la mémoire, des lignes d'entrée/sortie et d'autres éléments pour en faire un ordinateur fonctionnel.

(24) ARM : Type de langage assembleur

(25) Linux : est un système d'exploitation de type Unix. Il a été conçu pour équiper les ordinateurs personnels d'un système d'exploitation gratuit ou à très faible coût, comparable aux versions Unix classiques, généralement plus coûteuses.

(26) Raspbian : système d'exploitation libre et gratuit basé sur Debian optimisé pour fonctionner sur les différents Raspberry Pi.

(27) Windows 10 IoT Core : est conçu pour les petits appareils intelligents sécurisés, exécute des applications UWP et prend en charge les processeurs ARM.

(28) bare metal : système informatique matériel utilisé sans système d'exploitation complexe mais directement par un programme, généralement à l'aide d'une bibliothèque spécialisée.

(29) C : langage de programmation qui s’écrit dans un fichier source. Ensuite ce fichier doit être traduit à l’aide d’un compilateur en langage machine. On parle alors de langage compilé qui s’oppose aux langages interprétés, comme Python par exemple.

(30) GPIO : (anglais : General Purpose Input/Output, littéralement Entrée-sortie à usage général) sont des ports d’entrées-sorties.

(31) Frame Buffer : équipement de sortie vidéo qui commande un affichage vidéo à partir d’un tampon mémoire contenant une trame complète de données.

(32) compilateurs : programme qui transforme un code source en un code objet.

(33) d’interprètes : programme informatique qui traite le code source d’un projet logiciel pendant son fonctionnement – c'est-à-dire pendant son exécution – et joue le rôle d’interface entre le projet et le processeur.

(34) OS : est la couche logicielle qui – initialement chargée sur l’ordinateur par le biais d’un programme d’amorçage – gère toutes applications et les services de cet ordinateur.

(35) machine virtuelle : est un environnement virtualisé qui fonctionne sur une machine physique. Elle permet d’émuler un OS sans l’installer physiquement sur l’ordinateur.

(36) Ubuntu : est un système d’exploitation GNU/Linux fondé sur Debian. Il est développé, commercialisé et maintenu pour les ordinateurs individuels (Desktop), les serveurs (Server) et les objets connectés (Core) par la société Canonical.

(37) paquets : est l’entité de transmission de la couche réseau (couche 3 du modèle OSI).

(38) noyaux : Il constitue le cœur du système qui exploite un ordinateur. Il établit la communication entre la partie matérielle et la partie logicielle de l’appareil. C’est ce qui permet à l’utilisateur d’interagir avec la machine.

(39) FAT32 : utilitaire de formatage de disque établi pour formater les disques de plus de 32 Go de capacité de stockage avec le vénérable système de classification FAT32.

(40) ROM : mémoire informatique non volatile (c’est-à-dire une mémoire qui ne s’efface pas lorsque l’appareil qui la contient n’est plus alimenté en électricité) et dont le contenu est fixé lors de sa programmation, qui pouvait être lue plusieurs fois par l'utilisateur, mais ne pouvait plus être modifiée.

(41) LED : Une LED (en français : DEL : diode électroluminescente) est un composant électronique et optique, qui en étant traversé par du courant électrique, émet une lumière d’une intensité diffuse. Les LED consomment peu d’électricité.

(42) reverse engineering : consiste à démonter un objet pour voir comment il fonctionne afin de le dupliquer ou de l’améliorer.

(43) benchmarking : ensemble de procédures de recherches et d’analyses comparatives de la concurrence. Il permet d’améliorer les performances d’une entreprise grâce à l’élaboration d’un plan d’action, rédigé grâce aux conclusions tirées de cette analyse.

Webographie

[1] Rapports des années précédentes sur ce projet
Consulté le 09 juin 2023 :
https://drive.google.com/drive/folders/1ieEgGR6jduFLbYk754PEnA2_pfNnvAju?usp=sharing

[2] NVIDIA – What Is AI Upscaling?
Consulté le 09 juin 2023 :https://blogs.nvidia.com/blog/2020/02/03/what-is-ai-upscaling/

[3] Github – RGBtoHDMI / Wiki
Consulté le 09 juin 2023 : https://github.com/hoglet67/RGBtoHDMI/wiki

[4] Github – RGBtoHDMI / Wiki / Mesure des Lags
Consulté le 09 juin 2023 :
https://github.com/hoglet67/RGBtoHDMI/wiki/Lag-Measurement

[5] elinux.org - RPi HardwareHistory
Consulté le 09 juin 2023 : https://elinux.org/RPi_HardwareHistory

[6] Assembly Language Programming with ARM – Full Tutorial for Beginners ; Vidéo YouTube de freeCodeCamp.org
Consulté le 09 juin 2023 : https://www.youtube.com/watch?v=gfmRrPjnEw4

[7] arm développer – Instruction cycle count summary
Consulté le 12 juin 2023 :
https://developer.arm.com/documentation/ddi0165/b/I1028171

[8] stack overflow – Why do memory instructions take 4 cycles in ARM assembly?
Consulté le 12 juin 2023 :
https://stackoverflow.com/questions/41580145/why-do-memory-instructions-take-4-cycles-in-arm-assembly

[9] Cpulator – Emulateur de langage machine
Consulté le 09 juin 2023 : https://cpulator.01xz.net/?sys=arm-de1soc

[10] Silicium – Tandy TRS 80 model 1
Consulté le 09 juin 2023 : http://silicium.org/site/index.php/35-catalogue/tandy/115-tandy-trs-80-model-i

[11] Wikipédia – Zilog 80
Consulté le 09 juin 2023 : https://fr.wikipedia.org/wiki/Zilog_Z80

[12] Wikipédia – Circuit logique programmable
Consulté le 09 juin 2023 : https://fr.wikipedia.org/wiki/Circuit_logique_programmable

[13] Stardot.uk – Forum de RGBtoHDMI
Consulté le 09 juin 2023 : https://stardot.org.uk/forums/viewtopic.php?f=3&t=14430&hilit=RGB+to+HDMI+using+a+Pi+Zero

[14] Github – RGBtoHDMI/ CPLD Programming
Consulté le 09 juin 2023 : https://github.com/hoglet67/RGBtoHDMI/wiki/CPLD-Programming

[15] Xilinx - XC9572XL High-Performance CPLD Data Sheet (DS057)
Consulté le 09 juin 2023 : https://docs.xilinx.com/v/u/en-US/ds057

[16] Debian.org – Paquet : gcc-arm-none-eabi
Consulté le 09 juin 2023 : https://packages.debian.org/fr/sid/gcc-arm-none-eabi

[17] linuxhint – How to install make on Ubuntu
Consulté le 09 juin 2023 : https://linuxhint.com/install-make-ubuntu/

[18] Baremetal Assembly Raspberry Pi Programming | Direct to Register Blink LED, No Operating System ; Vidéo Youtbe de Low Level Learning
Consulté le 09 juin 2023 : https://www.youtube.com/watch?v=jN7Fm_4ovio&t

[19] 64 bit Bare Metal Programming on RPI-3 Your first aarch64 bare metal program ; Vidéo YouTube de FOSDEM
Consulté le 09 juin 2023 : https://www.youtube.com/watch?v=8-65xiGXMyA

[20] Developpez.com – Cours pour débuter dans la programmation d'une carte Arduino
Consulté le 09 juin 2023 : https://nboulaire.developpez.com/tutoriels/Arduino/cours-debuter-programmation-arduino/

[21] elinux.org – RPi Hardware
Consulté le 09 juin 2023 : https://elinux.org/RPi_Hardware

[22] Developpez.com – Arduino : Comment faire clignoter une LED ?
Consulté le 09 juin 2023 : https://arduino.developpez.com/cahiers-pratiques/clignoter-led/#:~:text=Pour%20allumer%20la%20LED%2C%20il,allum%C3%A9e%20puis%20%C3%A9teinte%2C%20elle%20clignote.

[23] University of Cambridge – Baking Pi – Operating Systems Development Downloads
Consulté le 09 juin 2023 :
https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html

[24] raspberry-project.com – Model B IO Pins
Consulté le 09 juin 2023 : https://raspberry-projects.com/pi/pi-hardware/raspberry-pi-model-b/model-b-io-pins

[25] Broadcom – BCM2835 ARM documentation pdf
Consulté le 09 juin 2023 : https://www.raspberrypi.org/app/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

[26] Github – Raspberry Pi firmware boot
Consulté le 09 juin 2023 : https://github.com/raspberrypi/firmware/tree/master/boot

[27] taylorpetrick.com – bare matal Raspberry Pi setup
Consulté le 09 juin 2023 : https://www.taylorpetrick.com/blog/post/bare-metal-pi-setup

[28] williamdurand.fr – bare matal Raspberry Pi 2 programming
Consulté le 09 juin 2023 : https://williamdurand.fr/2021/01/23/bare-metal-raspberry-pi-2-programming/

[29] assembleurarmpi.blogspot.com – Découvrir l'assembleur ARM sur Raspberry Pi
Consulté le 09 juin 2023 : https://assembleurarmpi.blogspot.com/2018/02/chapitre-19-gpio-raspberry-utilisation.html

[30] Gladir – Liste d'instructions du langage ARM
https://www.gladir.com/CODER/ARM/reference.htm

[31] Autodesk Instructables – Raspberry Pi Push Button With LEDs Bare Metal
Consulté le 09 juin 2023 : https://www.instructables.com/Raspberry-Pi-Push-Button-With-LEDs-Bare-Metal/