



Travail de Bachelor
2019 - 2020

Filière Informatique

Accélérateur de tour télécom¹

Rapport (v2, 17.07.2020)

Nicolas Maier²

Superviseurs : **Jacques Supcik³**
 Michael Mäder⁴

Expert : **Frédéric Mauron⁵**

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz

1. https://gitlab.forge.hefr.ch/nicolas.maier/tb_nm

2. nicolas.maier@edu.hefr.ch

3. jacques.supcik@hefr.ch

4. michael.maeder@hefr.ch

5. frederic.mauron@ftth-fr.ch

Table des versions

Version	Date de publication	Auteur	Description
1.0	10.07.2020	Nicolas Maier	Premier rendu intermédiaire du rapport pour juger la structure et le style
2.0	17.07.2020	Nicolas Maier	Rendu final du rapport complet

Table des matières

1	Introduction	1
1.1	Acteurs	1
1.2	Contexte	2
1.3	Situation avant le projet	2
1.4	Buts du projet	3
1.5	Méthode de travail	4
1.6	Structure du rapport	6
2	Analyse	7
2.1	Matrices de LEDs, contrôleurs ws2812b	7
2.2	Choix du matériel	12
3	Spécification	15
3.1	Organisation physique	15
3.2	Interface entre le Raspberry Pi et le Blue Pill	16
3.3	Interface des bibliothèques réalisées (côté Raspberry Pi)	17
4	Conception	20
4.1	Contrôle des LEDs	20
4.2	Interfaçage USB	23
4.3	Compression des données	24
4.4	Réduction des données	25
4.5	Bit Banding	26
5	Implémentation	27
5.1	Environnement de développement	27
5.2	Programme du Blue Pill	27
5.3	Système de fichiers	29
5.4	Bibliothèques réalisées (côté Raspberry Pi)	29
5.5	Programmes de démo	30
5.6	Limites du système	30
6	Tests et validation	31
6.1	Test fonctionnel	31

6.2	Test des performances	33
7	Améliorations et travaux futurs	36
7.1	Interfaçage USB pour d'autres périphériques	36
7.2	Extension de la capacité de la tour télécom	36
7.3	Améliorations/nouvelles librairies	36
8	Mise en place et exemple d'utilisation	37
8.1	Matériel	37
8.2	Branchements	38
8.3	Démarrage du programme	39
8.4	Résultat	40
8.5	Résolution des problèmes	40
9	Conclusion	41
9.1	Comparaison avec les objectifs	41
9.2	Rétrospective sur la planification initiale	41
9.3	Conclusion personnelle	41
10	Déclaration d'honneur	42

Chapitre 1

Introduction

Depuis plusieurs années, un projet dénommé "Tour télécom" est développé au sein de la filière Informatique et Télécommunications. L'objectif de ce projet est de créer et utiliser une maquette de tour interactive afin de faire la promotion de l'informatique et des systèmes de communication auprès du public lors des manifestations telles que les portes ouvertes.

La tour télécom (figure 1.1) comporte une matrice de LEDs flexible permettant d'afficher du texte et des animations. C'est sur cet aspect de la tour que ce projet porte. Le but du projet est de permettre d'augmenter la vitesse d'affichage (nombre d'images par secondes) de la matrice de LED, ainsi que de permettre d'augmenter le nombre de LEDs présentes sur la tour, le tout avec une interface simple à utiliser.



FIGURE 1.1 – Photo de la tour télécom

1.1 Acteurs

Ce projet est suivi par les personnes suivantes :

- Jacques Supcik, *Superviseur*
- Michael Mäder, *Superviseur*
- Frédéric Mauron, *Expert*
- Nicolas Maier, *Étudiant*

1.2 Contexte

Les matrices de LEDs présentes sur la tour sont composées de contrôleurs ws2812b. Chacun de ces contrôleurs gère un pixel RGB. Ces contrôleurs fonctionnent en série, il suffit d'envoyer les données de toute la matrice de LEDs au premier contrôleur de la chaîne, qui va transmettre les informations aux suivants (une description détaillée du fonctionnement des contrôleurs ws2812b est disponible dans le chapitre 2.1).

Ce système de LEDs est simple et extensible, mais demande toutefois un timing très précis pour transmettre des données, c'est pourquoi un système temps réel avec des contraintes de temps de l'ordre de 150ns est nécessaire pour contrôler les matrices de LEDs.

1.3 Situation avant le projet

Actuellement, le système d'affichage de la tour télécom est composé d'un Raspberry Pi qui contrôle une matrice de 4 bandes de 32x8 pixels RGB (Adafruit "Flexible 8x32 NeoPixel RGB LED Matrix"¹). Pour contrôler ces LEDs, le Raspberry Pi doit combiner un DMA et un PWM afin de générer un signal ayant un timing très précis, ce qui ne serait pas possible directement en contrôlant les GPIOs car le système Linux ne permettrait pas de garantir cette contrainte de temps réel avec la précision requise.

Les 4 bandes de 32x8 pixels sont branchées en série, le Raspberry Pi contrôle donc 1024 pixels comme indiqué sur la figure 1.2. Chaque pixel attend 24 bits afin d'obtenir une couleur RGB où chacune des 3 intensités est encodée sur 8 bits. Selon le protocole défini dans la datasheet du ws2812b[1], les données doivent être transmises à 800 Kb/s. La solution actuelle permet donc d'afficher environ 30 images par seconde.

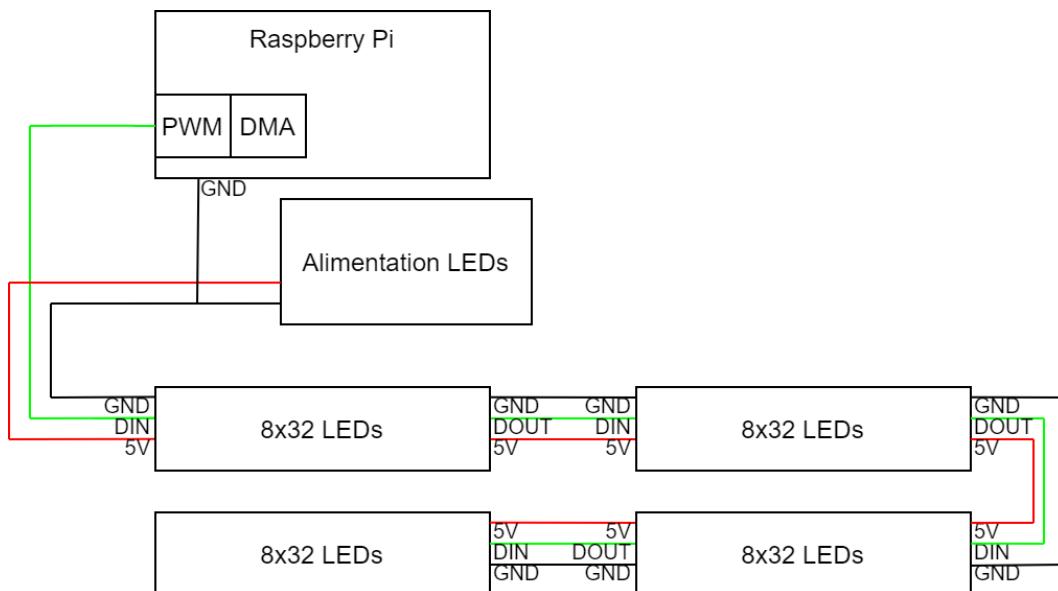


FIGURE 1.2 – Situation avant le projet

Cette solution est limitée en nombre de LEDs ou en images par seconde. En ajoutant des LEDs en série, on serait obligé de réduire le nombre d'images par seconde, et inversément.

1. <https://www.adafruit.com/product/2294>

1.4 Buts du projet

Le but principal de ce projet est d'ajouter un "coprocesseur" externe au Raspberry Pi qui s'occuperait du travail temps réel de contrôle des LEDs (génération du signal) sur plusieurs bandes en parallèle, afin de pouvoir augmenter la vitesse d'affichage (nombre d'images par secondes) de la tour télécom, ainsi que de permettre d'augmenter le nombre de LEDs présentes sur la tour, tout en déchargeant cette tâche du Raspberry Pi.

1.4.1 Objectifs du projet

L'objectif est de développer une solution comportant un microcontrôleur connecté au Raspberry Pi, comme indiqué sur la figure 1.3. Le microcontrôleur reçoit les informations (images à afficher) du Raspberry Pi, et s'occupe de générer le signal correspondant afin de contrôler plusieurs bandes de LEDs en parallèle.

Tâches :

- Étude des microcontrôleurs intéressants dans le contexte du projet
- Étude de l'interfaçage entre le Raspberry Pi et le microcontrôleur
- Choix du meilleur microcontrôleur pour ce projet
- Implémentation d'un système capable de contrôler au minimum 4 bandes de LEDs en parallèle
- Étude, description et justification des limites théoriques de la solution réalisée
- Tests et validation du système

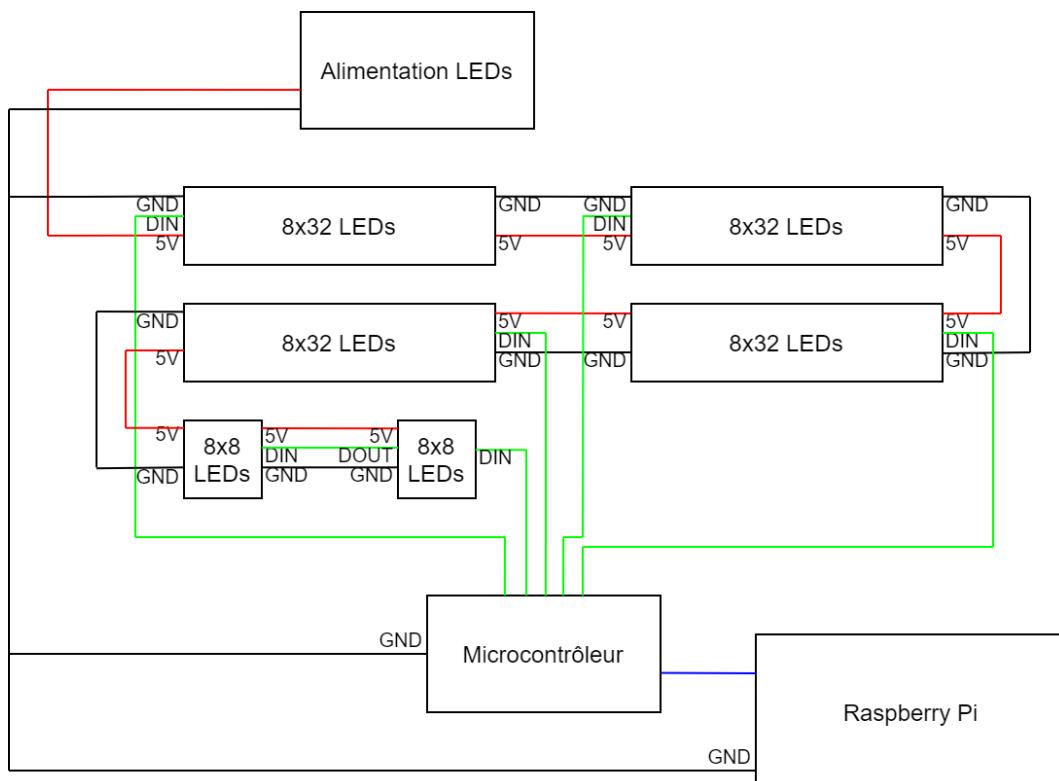


FIGURE 1.3 – Objectif du projet

1.4.2 Contraintes

Les contraintes suivantes doivent être respectées pour ce projet :

- Un minimum de 4 bandes de LEDs doivent pouvoir être contrôlées en parallèle, et une cinquième serait utile mais pas absolument nécessaire (le but serait de compléter le tour complet de la tour télécom avec deux carrés de 8x8 pixels²)
- Le système réalisé doit être simple à mettre en oeuvre (par exemple, l'idéal serait de pouvoir simplement brancher la tour télécom en USB au Raspberry Pi, de façon "plug-and-play")
- L'interface fournie pour contrôler les LEDs doit être inspirée de celle qui est actuellement utilisée[9]

1.5 Méthode de travail

Durant ce travail de Bachelor, il est important d'avoir un bon suivi tout au long de sa réalisation. C'est pourquoi des séances de projet ont eu lieu chaque semaine, auxquelles l'étudiant et les superviseurs ont participé. Deux séances supplémentaires, avec l'expert du projet et l'étudiant, ont également eu lieu. Chacune de ces séances a été résumée dans un procès-verbal³.

1.5.1 Cahier des charges

Un cahier des charges a été rédigé par l'étudiant et validé par les superviseurs après l'analyse des besoins et des objectifs du projet. Ce cahier des charges a permis de définir précisément le but et les contraintes du projet. Une rétrospective sur l'atteinte des objectifs se trouve dans le chapitre 9.1.

2. <https://www.adafruit.com/product/1487>

3. https://gitlab.forge.hefr.ch/nicolas.maier/tb_nm/-/tree/master/documentation/PVs

1.5.2 Planification initiale

Voici la planification prévue au début du projet (figure 1.4) :

	P13	P14	P15	P16	P17	P18	P19	P20	P21
Documentation									
Cahier des charges									
<i>Validation cahier des charges</i>			Ve						
Rapport technique									
Manuel d'utilisation									
<i>Rendu du rapport</i>									Ve
Analyse									
Étude des bandes de LEDs NeoPixel									
Étude des microcontrôleurs									
Étude de l'interfaçage entre le RPi et le(s) µp(s)									
<i>Choix du matériel</i>				Je					
Spécification									
Spécification de l'interface entre le RPi et le(s) µp(s)									
<i>Validation de la spécification</i>									
Conception									
Définition de l'architecture, diagrammes correspondants									
<i>Validation de la conception et choix des technologies</i>									
Réalisation									
Implémentation du contrôle des bandes NeoPixel									
Implémentation de l'interface RPi/µp(s)									
Tests									
Tests appropriés									
<i>Validation de la réalisation</i>									Me
<i>Défense orale</i>									

(Semaine des examens)

FIGURE 1.4 – Planification du projet

Une rétrospective sur la planification et comment le projet s'est déroulé se trouve dans le chapitre 9.2.

1.6 Structure du rapport

Ce rapport est séparé en 10 chapitres. Ces chapitres contiennent des sections, dans lesquelles sont expliquées les thèmes en détail. La liste ci-dessous contient une description de chaque chapitre :

— **Introduction**

Explique le contexte du projet, les buts à atteindre et son déroulement

— **Analyse**

Définit les besoins du système, et documente les choix technologiques réalisés durant le projet

— **Spécification**

Spécifie l'organisation physique des éléments du système ainsi que les protocoles utilisés entre ces éléments et leurs interactions

— **Conception**

Indique la structure de la réalisation du projet, ainsi que les techniques qui seront utilisées afin d'implémenter les fonctionnalités du projet

— **Implémentation**

Détaille la structure et les éléments importants de l'implémentation des différentes parties du projet, ainsi que les limites du système

— **Tests et validation**

Indique quels tests ont été effectués pour vérifier le fonctionnement et les performances du système, ainsi que les résultats de ces derniers

— **Améliorations et travaux futurs**

Contient une description des suites ou dérivés possibles du projet

— **Mise en place et exemple d'utilisation**

Explique en détails les étapes nécessaires pour mettre en place le système et faire tourner la démo fournie

— **Conclusion**

Donne une rétrospective sur différents aspects du projet, ainsi qu'une conclusion personnelle

— **Déclaration d'honneur**

Contient la déclaration d'honneur signée

Chapitre 2

Analyse

Ce chapitre décrit l'étude des différents besoins du système, et des différentes technologies et matériel choisies pour réaliser le projet.

2.1 Matrices de LEDs, contrôleurs ws2812b

Dans les matrices de LEDs utilisées dans ce projet, chaque pixel est composé d'un contrôleur ws2812b capable de gérer l'intensité individuelle de 3 couleurs (rouge, vert et bleu). Les informations détaillées à propos de ce contrôleur sont disponibles dans leur datasheet[1].

Voici une photo montrant en détail une partie d'une matrice NeoPixel, dans laquelle on voit plusieurs contrôleurs ws2812b sur un circuit imprimé flexible (figure 2.1)

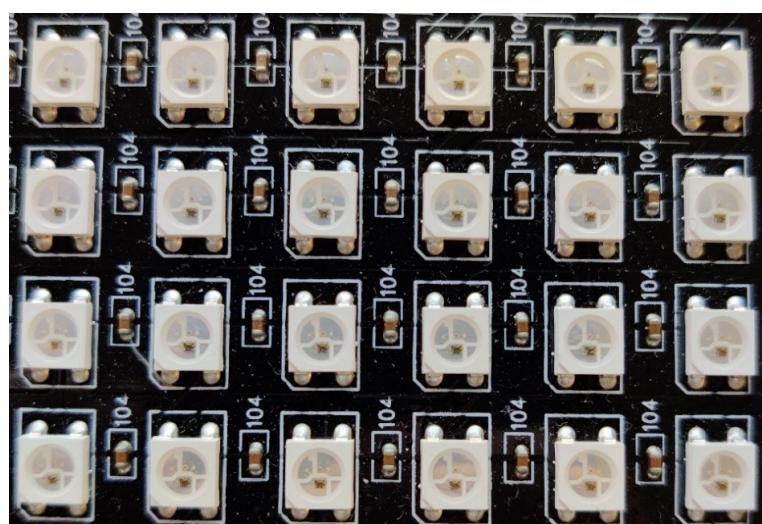


FIGURE 2.1 – Détails d'une matrice NeoPixel

2.1.1 Organisation en série

Ces contrôleurs sont conçus pour fonctionner avec autant de pixels que nécessaire : ils sont organisés en chaîne. Pour ajouter un pixel à une bande, il suffirait d'ajouter un contrôleur ws2812b à la chaîne. Cette organisation en cascade offre une grande flexibilité sur le nombre de pixels et leur organisation, tout en permettant d'adresser chaque LED individuellement. Mais elle donne également de mauvaises performances lorsque beaucoup de pixels sont utilisés, car l'information doit parcourir toute la chaîne.

Pour contrôler la couleur des pixels, un microcontrôleur doit envoyer les valeurs de chaque couleur de chaque LED (24 bits par pixel RGB) au premier contrôleur ws2812b de la chaîne. Chacun des contrôleurs lit et affiche la première valeur (24 bits) qu'il reçoit, et transmet toutes les autres au contrôleur ws2812b suivant. Cela permet de donner la couleur désirée à chacun des pixels, tout en n'ayant qu'à brancher une seule ligne de données pour toute la bande de LEDs. Lorsque le microcontrôleur a envoyé toutes les valeurs désirées, il doit envoyer un signal particulier ("reset code"). Ce signal indique à chaque contrôleur ws2812b qu'il devra à nouveau prendre une valeur, dans le but d'afficher l'image suivante.

Voici un schéma montrant comment les contrôleurs ws2812b doivent être câblés entre eux pour fonctionner en série (figure 2.2) :

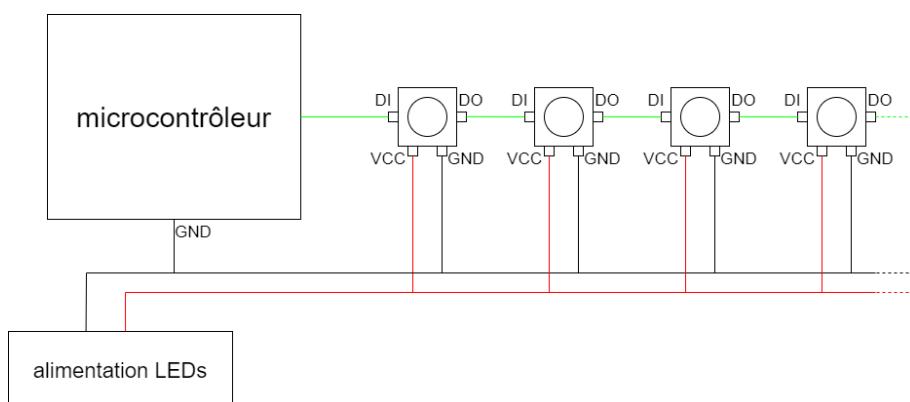


FIGURE 2.2 – Schéma du câblage des contrôleurs ws2812b

2.1.2 Signal (protocole)

Le protocole définit dans la datasheet des contrôleurs ws2812b est simple, mais nécessite une bonne précision dans le temps. Il s'agit d'un signal carré alternant entre haut et bas avec une période de $1.25\mu\text{s}$ (fréquence de 800KHz), dont le rapport cyclique (temps à l'état haut) est modulé pour définir l'information envoyée.

Au début de la période, le signal est à "haut".

- Si on veut envoyer le bit "0", le signal passe à "bas" après $0.35\mu\text{s}$
- Si on veut envoyer le bit "1", le signal passe à "bas" après $0.7\mu\text{s}$

D'après la datasheet, ces temps doivent être précis à 150ns près.

Chaque contrôleur ws2812b consomme 24 bits et passe le reste du signal au contrôleur suivant. Ces 24 bits sont composés de l'information suivante (figure 2.3, à lire de gauche à droite) :

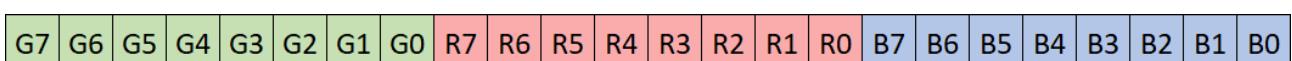


FIGURE 2.3 – Organisation des bits codant les couleurs

Les MSB des codes de couleurs sont envoyés avant les LSB. Chaque couleur est encodée sur 8 bit, et elles sont envoyées dans l'ordre GRB.

Le "reset code", permettant d'indiquer aux contrôleurs ws2812b qu'une nouvelle image va être envoyée, est un signal "bas" durant plus de $50\mu s$.

Voici un schéma résumant les 3 signaux différents permettant de contrôler les pixels (figure 2.4) :

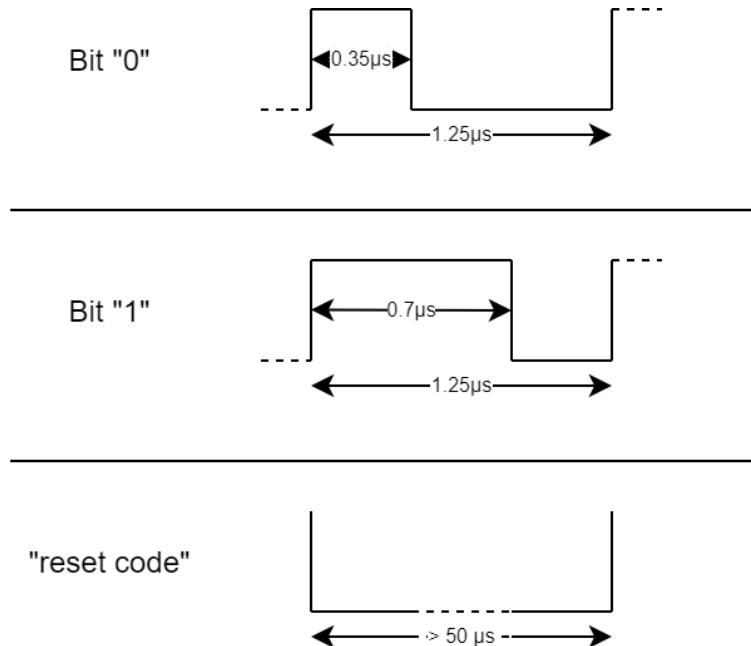


FIGURE 2.4 – Signaux permettant de contrôler les pixels

Par exemple, voici le chronogramme représentant l'envoi du byte "0x5C" ("0101 1100") :

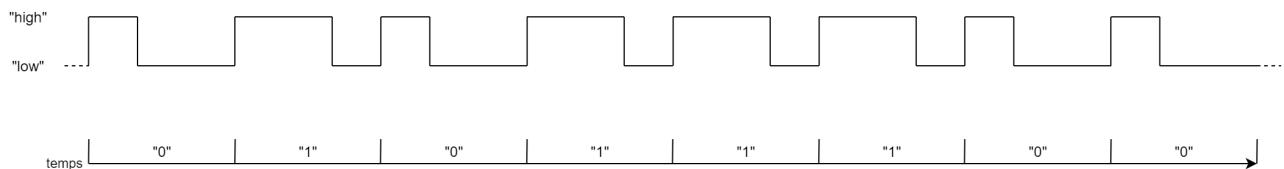


FIGURE 2.5 – Signal lors de l'envoi du byte "0x5C"

2.1.3 Luminosité des LEDs (correction gamma)

La valeur envoyée pour chaque composante des pixels définit le rapport cyclique du signal contrôlant l'intensité de la couleur. C'est à dire qu'une valeur de 127 va donner un signal avec 50% de temps "haut" et 50% de temps "bas" à la LED. Une valeur de 255 va donner un signal avec 100% de temps "haut".

Dû au fonctionnement des LEDs et de l'oeil humain, la luminosité perçue ne correspond pas à la valeur envoyée. En réalité, la luminosité perçue augmente très vite même avec des petites valeurs envoyées. Par exemple, la valeur 127 (50% du maximum) donne en fait une luminosité d'environ 85% du maximum.

Pour obtenir une meilleure fidélité des couleurs, il suffit d'utiliser la table de correction fournie par Adafruit (figure 2.6) :

```

1. const uint8_t PROGMEM gamma8[] = {
2.     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
3.     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
4.     1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,
5.     2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5,
6.     5, 6, 6, 6, 6, 7, 7, 7, 8, 8, 8, 9, 9, 9, 10,
7.     10, 10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 14, 15, 15, 16,
8.     17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 24, 24, 25,
9.     25, 26, 27, 27, 28, 29, 29, 30, 31, 32, 32, 33, 34, 35, 35, 36,
10.    37, 38, 39, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 50,
11.    51, 52, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68,
12.    69, 70, 72, 73, 74, 75, 77, 78, 79, 81, 82, 83, 85, 86, 87, 89,
13.    90, 92, 93, 95, 96, 98, 99, 101, 102, 104, 105, 107, 109, 110, 112, 114,
14.    115, 117, 119, 120, 122, 124, 126, 127, 129, 131, 133, 135, 137, 138, 140, 142,
15.    144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 167, 169, 171, 173, 175,
16.    177, 180, 182, 184, 186, 189, 191, 193, 196, 198, 200, 203, 205, 208, 210, 213,
17.    215, 218, 220, 223, 225, 228, 231, 233, 236, 239, 241, 244, 247, 249, 252, 255 };

```

FIGURE 2.6 – Table de correction gamma (source : <https://learn.adafruit.com/led-tricks-gamma-correction>)

Elle permet d'obtenir la valeur à envoyer pour afficher un niveau de luminosité donné.

2.1.4 Agencement des pixels dans une matrice

Les matrices de pixels utilisées dans ce projet (matrices 8×32^1 et 8×8^2) utilisent un agencement particulier des LEDs. Comme indiqué dans la figure 2.7, les pixels sont disposés en zigzag :

32	33	34	35	36	37	...		
31	30	29	28	27	26	25	24	
16	17	18	19	20	21	22	23	
15	14	13	12	11	10	9	8	
DIN—	0	1	2	3	4	5	6	7

FIGURE 2.7 – Agencement des matrices de pixels

On peut donc obtenir l'index d'un pixel à partir de coordonnées x/y avec le code suivant (figure 2.8) :

```
index = y * 8 + (y % 2 == 0 ? x : 7-x)
```

FIGURE 2.8 – Calcul de l'index d'un pixel grâce à ses coordonnées x/y

Ces coordonnées correspondent au système d'axes montré dans la figure 2.9 :



FIGURE 2.9 – Agencement des axes

1. <https://www.adafruit.com/product/2294>
2. <https://www.adafruit.com/product/1487>

2.2 Choix du matériel

Pour réaliser les objectifs du travail, un microcontrôleur a été choisi. Voici un résumé des caractéristiques de différents microcontrôleurs intéressants retenus pour le projet :

2.2.1 ESP32 DEVKIT DOIT

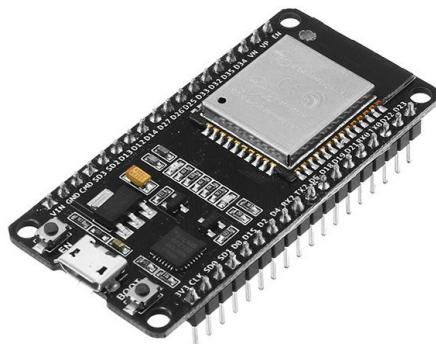


FIGURE 2.10 – ESP32 (source : www.banggood.com)

Ce microcontrôleur est très populaire. Il propose de très bonnes caractéristiques tout en étant disponible pour un prix d'environ 10 CHF³.

Il comporte un processeur Xtensa dual-core cadencé à 240 MHz, ainsi que 500KB de RAM. Il possède un module wifi capable de transmettre environ 20Mbps[10].

2.2.2 Sipeed Maix Bit

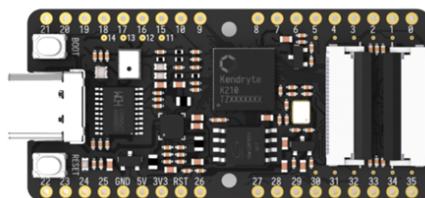


FIGURE 2.11 – Sipeed maix bit (source : www.seeedstudio.com)

Ce microcontrôleur moderne a la particularité d'être équipé d'un Kendryte K210 dual core, basé sur l'architecture RISC-V. Il possède 8MB de RAM, et son processeur est cadencé à 400MHz. Il est disponible pour un prix d'environ 16 CHF⁴.

Il comporte un port USB type C, qui peut être utilisé pour le programmer et pour transmettre des données entre l'hôte et le programme embarqué. Il comporte un chip USB-UART supportant un débit maximum de 2Mbps.

3. <https://www.banggood.com/ESP32-Development-Board...>

4. <https://www.distrelec.ch/en/sipeed-maix-bit-for-risc-ai-iot-seeed-studio-102991150/p/30135127>

2.2.3 BeagleBone Black

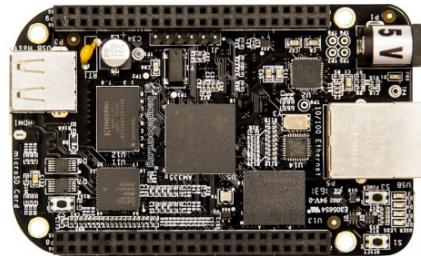


FIGURE 2.12 – BeagleBone Black (source : www.element14.com)

Il s'agit d'un "single-board computer" capable de faire tourner Linux. Il est disponible pour un prix d'environ 70 CHF⁵. Il est équipé d'un processeur mono cœur cadencé à 1GHz, ainsi que de 512MB de RAM.

Il a la particularité de comporter, en plus de son processeur principal, deux co-processeurs ("PRU") cadencés à 200MHz et capables d'interagir avec la mémoire principale ainsi que les composants du BeagleBone Black. Bien que le processeur principal fasse tourner Linux, un des PRU pourrait s'occuper de générer le signal des LEDs avec des contraintes temps réel.

Il comporte une interface Ethernet 100Mbps.

2.2.4 Blue Pill (STM32F103C8T6)



FIGURE 2.13 – Blue Pill (source : stm32-base.org)

Cette carte est équipée d'un STM32F103C8T6. Il s'agit d'un MCU Cortex-M3, cadencé à 72MHz et proposant 20KB de RAM. Il est disponible pour un prix d'environ 3 CHF⁶.

Bien qu'extrêmement bon marché, ce microcontrôleur propose des avantages intéressants. Il possède une interface USB très flexible : il peut être vu par l'hôte comme différentes classes, et on peut même implémenter notre propre stack USB. Cette flexibilité permet au Blue Pill d'apparaître par exemple comme un périphérique de stockage auprès de l'hôte.

Étant donné qu'il fait partie de la famille des STM32, il est facilement interchangeable avec d'autres membres de cette famille. En effet, l'API permettant de contrôler les différents périphériques étant commune entre ces MCU, la majeure partie du code peut être gardée si il est nécessaire de passer à un MCU ayant plus de RAM par exemple.

5. <https://www.banggood.com/Embest-BeagleBone-BB-Black-Cortex-A8-Development-Board-REV-C-Version-p-954565.html>
6. <https://www.banggood.com/STM32F103C8T6-...>

2.2.5 Évaluation et choix

Les microcontrôleurs présentés ci-dessus ont été évalués selon différents critères :

- Prix : Prix à l'achat du microcontrôleur
- Capacité (RAM) : Quantité de mémoire RAM disponible
- Popularité (moins de risque) : Facilité à trouver des informations en ligne, ancienneté du microcontrôleur
- Interface (vitesse) : Vitesse disponible pour le transfert d'informations entre le Raspberry Pi et le microcontrôleur
- Facilité à mettre en oeuvre : Facilité à connecter le Raspberry Pi au microcontrôleur et à envoyer les informations des images à afficher

Un poids de 1 à 5 a été donné à chaque critère (1 : le moins important, 5 : le plus important). Chaque microcontrôleur présenté a ensuite été évalué selon chacun des critères avec une note de 1 à 5, afin d'obtenir un total permettant d'indiquer si il est adapté au projet (figure 2.14) :

Poids	2	2	2	4	5	
Critères	Prix	Capacité (RAM)	Popularité (moins de risque)	Interface (vitesse)	Facilité à mettre en œuvre	Total
Sipeed Maix Bit	3	4	1	2	3	39
ESP32 DEVKIT DOIT	4	3	4	4	1	43
BeagleBone Black	1	5	4	5	4	60
Blue Pill	5	2	3	4	5	61

FIGURE 2.14 – Tableau multicritère

En prenant tous ces éléments en compte, le Blue Pill a été choisi pour réaliser ce projet. La flexibilité de son interface USB et son bas prix permettra au produit réalisé d'être très accessible pour quiconque voulant contrôler des LEDs basées sur les contrôleurs ws281x.

Chapitre 3

Spécification

Ce chapitre spécifie l'organisation physique des éléments du système, ainsi que les interactions entre les différentes parties du système (protocoles et interfaces).

3.1 Organisation physique

Le système est composé de 3 éléments physiques principaux différents :

- Le Raspberry Pi : Il génère les images à afficher et les envoie au microcontrôleur en suivant le protocole définit ci-dessous
- Le Blue Pill : Il reçoit les images à afficher du Raspberry Pi et génère les signaux correspondant pour contrôler jusqu'à 8 bandes de LEDs en parallèle
- Les bandes de LEDs : Ce sont les bandes de contrôleurs ws2812b connectés en série

Voici un schéma (figure 3.1) montrant l'organisation physique de ces éléments :

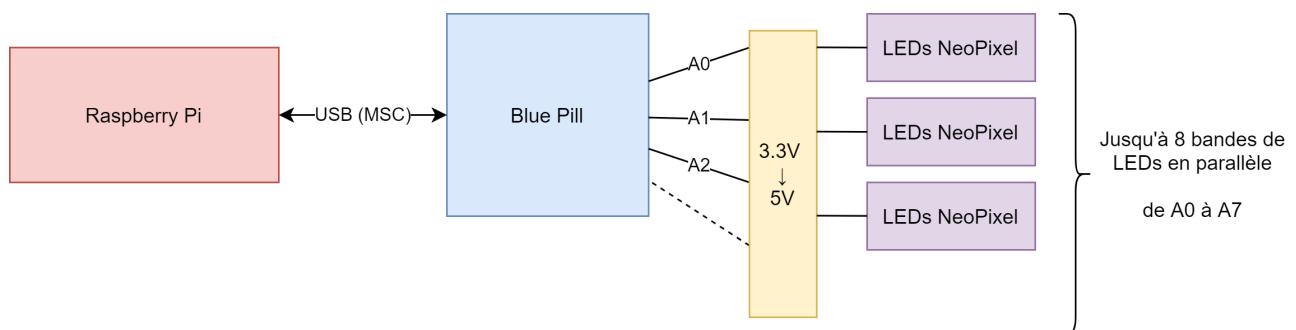


FIGURE 3.1 – Organisation physique du système

Le Raspberry Pi est connecté par USB au Blue Pill. Le Blue Pill est connecté aux différentes bandes de LEDs en parallèle sur les pins 0 à 7 de la banque de GPIOs A (A0, A1, ...). Le signal envoyé aux bandes de LEDs passe par un convertisseur 3.3V vers 5V (CD4050B[4]).

3.2 Interface entre le Raspberry Pi et le Blue Pill

Le Blue Pill est configuré pour s'annoncer comme un périphérique de stockage USB (MSC). Il présente un système de fichiers FAT12, avec des blocs de 512 bytes, contenant trois fichiers : "config", "coding" et "data". Chacun de ces fichiers a une taille de 512 bytes, c'est à dire un seul bloc.

- Le fichier "config" permet de configurer le système (nombre de LEDs, nombre de canaux, réduction des données envoyées)
- Le fichier "coding" permet de d'activer et de définir le code de Huffman utilisé (voir chapitre 4.3 pour plus de détails)
- Le fichier "data" permet d'envoyer les données des LEDs à afficher

Le Raspberry Pi peut écrire dans ces trois fichiers, pour obtenir le comportement désiré :

3.2.1 Fichier "config"

Le fichier "config" contient la configuration du périphérique. Son contenu a la signification suivante :

- **byte 0** (*uint8_t*) : "reduction mode", peut être soit "0" soit "1", active ou non la réduction des données (lorsqu'elle est activée ("1"), seuls les 4 MSB de chaque byte sont envoyés et stockés, voir le chapitre 4.4 pour plus de détails)
- **byte 1** (*uint8_t*) : "strip_n", nombre entre 0 et 8 (compris), définit le nombre de bandes de LEDs utilisées en parallèle (lorsque ce nombre est plus petit que 8, ce sont les pins avec un numéro plus petit qui sont utilisées, et les pins avec un numéro plus grand qui sont désactivées)
- **bytes 2 à 3** (*uint16_t*) : "led_n", nombre de pixels présents sur chaque bande de LEDs (si des bandes de taille différentes sont utilisées en parallèle, ce paramètre doit être mis à la valeur correspondant à la plus longue bande de LEDs) (en raison des détails de l'implémentation, cette valeur doit être un multiple de 4)

3.2.2 Fichier "coding"

Le fichier "coding" contient les informations du codage de Huffman utilisé. Il est séparé en trois parties :

- **byte 0** (*uint8_t*) : si il est mis à "0", cela indique que le codage de Huffman doit être utilisé ; pour toute autre valeur, le codage de Huffman est désactivé
- **bytes 1 à 255** (*uint8_t[127]*) : partie "taille des codes" du code de Huffman canonique utilisé, donne le nombre de codes ayant chacune des tailles possibles (de 2 à 255) (plus de détails dans le chapitre 4.3.3)
- **bytes 256 à 511** (*uint8_t[128]*) : partie "symboles triés" du code de Huffman canonique utilisé, donne la liste des symboles de l'alphabet utilisé triés par la taille de leur code (plus de détails dans le chapitre 4.3.3)

3.2.3 Fichier "data"

Le fichier "data" permet de transmettre les données des couleurs de chaque pixel à afficher. Pour définir la couleur de chaque LED, il est possible d'écrire plusieurs fois d'affilée dans ce fichier en faisant varier les paramètres. Il est structurée de la façon suivante :

- **bytes 0 à 1 (`uint16_t`)** : "offset", permet d'indiquer où doivent être placées les données transmises dans le buffer
- **bytes 2 à 3 (`uint16_t`)** : "size", permet d'indiquer le nombre de bytes envoyés dans le bloc transmis (avec la valeur "offset", il définit quelles données seront remplacées dans le buffer des données des LEDs)
- **byte 4 (`uint8_t`)** : indique si le périphérique doit envoyer les données aux bandes de LEDs ("0" pour ne pas envoyer, toute autre valeur pour envoyer) ; cette valeur doit être mise à "0" excepté lorsque le bloc actuel est le dernier de l'image complète à afficher
- **bytes 5 à 511 (`uint8_t[507]`)** : données des LEDs, peuvent être encodées avec le code de Huffman défini ou non en fonction des informations données dans le fichier "coding"

Une fois décodées, les données transmises sont copiées dans le buffer des codes couleurs, à la position "offset", jusqu'au point "offset + size". Dans ce buffer, les données doivent être organisées de la façon indiquée dans la figure 3.2 (exemple avec 2 bandes contenant chacune 3 pixels) :

G_0_0	G_1_0	R_0_0	R_1_0	B_0_0	B_1_0	G_0_1	G_1_1	R_0_1	R_1_1	B_0_1	B_1_1	G_0_2	G_1_2	R_0_2	R_1_2	B_0_2	B_1_2	
Bande 0	Bande 1																	
Pixel 0	Pixel 1	Pixel 2																

FIGURE 3.2 – Organisation des données dans le buffer des codes couleurs avec 2 bandes de 3 pixels

Il s'agit de l'ordre dans lesquels les données seront envoyées sur les pins : d'abord les bytes 0 ("G") du premier pixel de chaque bande, puis les bytes 1 ("R") du premier pixel de chaque bande, puis le byte 2 ("B"), puis on passe au pixel suivant, etc...

Lorsque la réduction des données est activée, l'ordre reste la même mais les codes de chaque composante des pixels sont donnés par 4 bits uniquement (les 4 MSB). On trouve donc deux composantes différentes dans un même byte. La composante qui se trouverait à l'index le plus petit dans les données non réduites est placée dans les 4 MSB, et la composante qui se trouverait à l'index le plus élevé est placé dans les 4 LSB. La figure 3.3 montre un exemple pour 1 bande de 2 pixels :

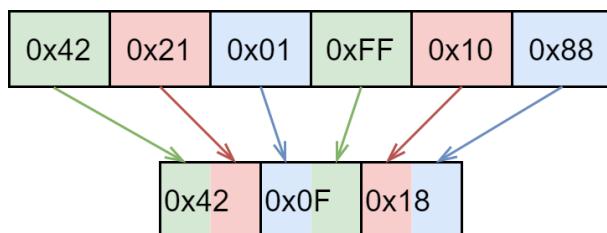


FIGURE 3.3 – Réduction des données pour 1 bande de 2 pixels

3.3 Interface des librairies réalisées (côté Raspberry Pi)

Deux implémentations de librairies permettant de contrôler le périphérique sont fournies dans ce projet.

- **Librairie C** : Cette librairie, réalisée en C, fournit une API inspirée de celle anciennement utilisée dans la tour télécom[9] ; elle est réalisée de façon simple et ne permet pas d'utiliser certaines fonctionnalités du périphérique
- **Librairie C++** : Cette librairie, réalisée en C++, fournit une API plus complète permettant d'utiliser toutes les fonctionnalités du périphérique

3.3.1 Librairie C

Cette librairie est composée de deux modules, "led_controller" et "led_matrix".

Il s'agit d'une implémentation minimale permettant de contrôler les bandes de LEDs, c'est pourquoi il n'est pas possible d'utiliser les fonctionnalités suivantes :

- Réduction des données
- Codage de Huffman

Grâce à sa simplicité, cette librairie peut facilement servir de base ou de référence lors de l'implémentation d'un programme contrôlant les LEDs dans un autre langage.

3.3.1.1 led_controller

Ce module permet de contrôler le périphérique.

Il fournit les types de données suivants :

- *led_controller_led_t* : Couleur RGB d'un pixel
- *led_controller_channel_t* : Canal (bande de LEDs), contient les couleurs de la bande de LED
- *led_controller_t* : Contrôleur des LEDs, contient les paramètres utilisés et les canaux

Et il fournit les fonctions suivantes :

- *led_controller_init* : Initialise le contrôleur de LEDs, et envoie les paramètres au périphérique
- *led_controller_cleanup* : Libère les ressources allouées lors de l'initialisation
- *led_controller_render* : Met à jour l'image (envoie les données des pixels au périphériques)

Les paramètres du contrôleur de LEDs sont donnés dans la structure *led_controller_t*. Les valeurs suivantes sont configurables :

- Emplacement du périphérique (système de fichiers)
- Nombre de canaux (bandes de LEDs) en parallèle
- Nombre de LEDs sur chaque canal
- Activation de la correction gamma (voir chapitre 2.1.3)

3.3.1.2 led_matrix

Ce module permet de définir la couleur des pixels d'une matrice de LEDs en utilisant des coordonnées x/y (comme décrites dans le chapitre 2.1.4).

Il fournit le type de donnée *led_matrix_t*, qui contient les paramètres définissant l'emplacement de la matrice sur une bande de LEDs.

Il fournit également la fonction *led_matrix_set_pixel_color*, qui permet de définir la couleur d'un pixel de la matrice en utilisant des coordonnées "x" et "y".

3.3.2 Librairie C++

Cette librairie est composée de deux modules, "led_controller" et "led_matrix".

Il s'agit d'une implémentation complète permettant de contrôler les bandes de LEDs en utilisant toutes les fonctionnalités disponibles du périphérique.

3.3.2.1 led_controller

Ce module permet de contrôler le périphérique.

Il fournit les types de données suivants :

- *ColorRGB* : Couleur RGB d'un pixel
- *LedController* : Contrôleur des LEDs, contient les paramètres utilisés et fournit les méthodes permettant d'interagir avec le périphérique

La classe *LedController* fournit les méthodes suivantes :

- *set_pixel_color* : Permet de définir la couleur d'un pixel
- *update_huffman_code* : Calcule le code de Huffman pour les données préparées au préalable, et envoie ce code au périphérique
- *render* : Met à jour l'image (envoie les données des pixels au périphérique)

Les paramètres du contrôleur de LEDs sont donnés dans son constructeur. Les valeurs suivantes sont configurables :

- Emplacement du périphérique (système de fichiers)
- Nombre de canaux (bandes de LEDs) en parallèle
- Nombre de LEDs sur chaque canal
- Activation de la correction gamma (voir chapitre 2.1.3)
- Réduction des données (voir chapitre 4.4)

Par défaut, le codage de Huffman est désactivé. Il peut être activé à tout moment en utilisant la méthode *update_huffman_code*.

3.3.2.2 led_matrix

Ce module permet de définir la couleur des pixels d'une matrice de LEDs en utilisant des coordonnées x/y (comme décrites dans le chapitre 2.1.4).

Il fournit le type de donnée *LedMatrix*, qui contient les paramètres définissant l'emplacement de la matrice sur une bande de LEDs.

La classe *LedMatrix* fournit la méthode *set_pixel_color*, qui permet de définir la couleur d'un pixel de la matrice en utilisant des coordonnées "x" et "y".

Chapitre 4

Conception

Ce chapitre décrit la structure de la réalisation du projet. Il explique les techniques et les concepts qui seront utilisées lors de l'implémentation afin de réaliser les fonctionnalités nécessaires.

4.1 Contrôle des LEDs

Pour contrôler les LEDs, le Blue Pill doit générer un signal tel que décrit dans le chapitre 2.1.2 pour chacune des bandes en parallèle.

Le Blue Pill, équipé d'un STM32F103C8T6, comporte différents composants qui, combinés, permettent de générer ce signal de façon précise et fiable :

4.1.1 DMA

Le STM32F103C8T6 comporte un module DMA ("Direct Memory Access"). Un DMA permet de transférer des données sans que le CPU ait à faire ce travail. Ce transfert peut se faire de la mémoire vers la mémoire, de la mémoire vers un périphérique, ou d'un périphérique vers la mémoire. D'une façon générale, un DMA a l'avantage d'économiser du temps de CPU, et de permettre des transferts à des timings précis (le DMA peut être déclenché par différentes sources, par exemple un périphérique).

Le DMA de ce microcontrôleur comporte 7 canaux indépendants, qui peuvent être utilisés en parallèle. Ils peuvent être configurés pour effectuer le transfert le plus vite possible, ou alors il peut attendre un signal qui déclenchera le transfert du byte suivant.

4.1.2 Timer

Le STM32F103C8T6 comporte plusieurs "General Purpose Timers". Ils consistent principalement en un compteur incrémenté à une fréquence définissable par le programme, qui retourne à 0 lorsqu'il atteint une certaine valeur choisie ("auto-reload"). Lorsque cela arrive, un signal est généré et cela peut être utilisé pour déclencher un des canaux du module DMA.

De plus, ces timers comportent plusieurs canaux configurables. Ces canaux peuvent être configurés pour obtenir différents comportements. Pour ce projet, le mode utilisé est "output compare". Dans ce mode, le canal est activé lorsque le compteur du timer atteint une valeur configurée dans le programme. Lors de cette activation, un signal est généré et cela peut être utilisé pour déclencher un des canaux du module DMA.

4.1.3 GPIO

Le STM32F103C8T6 comporte 4 banques de 16 GPIOs ("General-Purpose Input/Output"). Ces modules permettent de lire ou d'écrire l'état des pins du Blue Pill ("haut" ou "bas"). Pour ce projet, nous voulons écrire sur ces pins pour générer le signal des LEDs. Pour ce faire, 3 registres distincts de 32 bits peuvent être utilisés sur chaque banque :

- **ODR** : "Output Data Register", lorsqu'une écriture est faite sur ce registre, les pins de la banque de GPIO vont prendre l'état donné par les 16 LSB écrits
- **BRR** : "Bit Reset Register", lorsqu'une écriture est faite sur ce registre, les bits à "1" dans les 16 LSB vont passer les pins correspondantes à l'état "bas", et ceux dans les 16 MSB vont passer les pins correspondantes à l'état "haut" (les bits à "0" ne modifient pas l'état des pins)
- **BSRR** : "Bit Set Reset Register", il a la même fonction que le registre BRR mais avec l'effet inverse : les 16 LSB du BSRR ont l'effet des 16 MSB du BRR, et les 16 MSB du BSRR ont l'effet des 16 LSB du BRR

Ces registres peuvent être écrits par le module DMA du STM32F103C8T6.

4.1.4 Combinaison des composants

Grâce aux modules décrits ci-dessus, on peut donc générer un signal précis en écrivant dans les registres du module GPIO avec plusieurs canaux DMA, déclenchés par un timer et ses canaux en mode "output compare".

Dans ce projet, 3 canaux DMA sont orchestrés afin de générer le signal des LEDs sur 8 pins en parallèle :

- 1er canal DMA ("up") : ce canal est déclenché au début de la période du timer, il met toutes les pins à l'état "haut"
- 2ème canal DMA ("down if 0") : ce canal est déclenché 350ns après le premier, il met les pins à l'état "bas" pour les sorties où on souhaite envoyer le bit "0" et laisse les autres inchangées
- 3ème canal DMA ("down") : ce canal est déclenché 700ns après le premier, il met toutes les pins à l'état "bas"

De cette façon, un signal "haut" plus court sera généré pour envoyer le bit "0" aux bandes de LEDs, et un signal "haut" plus long sera généré pour envoyer le bit "1".

Pour déclencher les canaux DMA au bon moment, un timer doit être utilisé. Il doit avoir une période de $1.25\mu s$ (fréquence de 800 KHz), et comporter en plus deux canaux en mode "output compare" qui seront déclenchés respectivement 350ns et 700ns après le début de chaque période.

La figure 4.1 montre le fonctionnement de ce système pour envoyer un bit "0" ou un bit "1" :

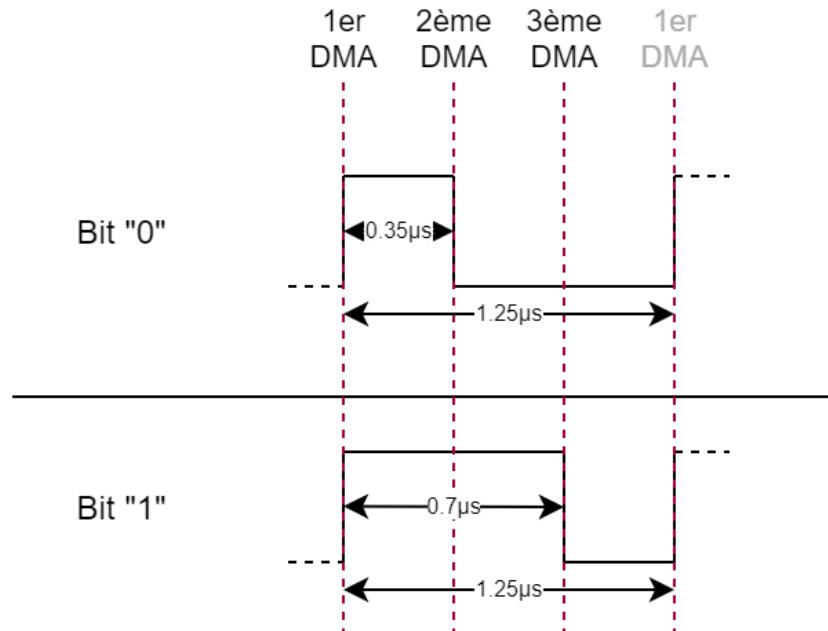


FIGURE 4.1 – Utilisation de 3 DMA pour générer le signal

4.1.5 Choix des canaux

Sur le STM32F103C8T6, les canaux DMA ne peuvent être déclenchés que par des sources spécifiques. Voici le résumé des sources correspondant à chaque canal DMA (figure 4.2, tirée du "STM32F103xx reference manual" [3]) :

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1	-	-	-	-	-	-
SPI/I ² S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I ² C	-	-	-	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1	-	TIM1_CH1	-	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
TIM2	TIM2_CH3	TIM2_UP	-	-	TIM2_CH1	-	TIM2_CH2 TIM2_CH4
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	-	-	TIM3_CH1 TIM3_TRIG	-
TIM4	TIM4_CH1	-	-	TIM4_CH2	TIM4_CH3	-	TIM4_UP

FIGURE 4.2 – Sources pour les canaux DMA (source : [3])

Le choix des canaux utilisés pour ce projet est le suivant :

- Timer : TIM2
- 1er canal DMA ("up") : channel 2, déclenché par TIM2_UP
- 2ème canal DMA ("down if 0") : channel 5, déclenché par TIM2_CH1
- 3ème canal DMA ("down") : channel 7, déclenché par TIM2_CH2

4.2 Interfaçage USB

Comme expliqué dans le chapitre 3.2, le Blue Pill s'annonce comme un périphérique de stockage lorsque connecté en USB à l'hôte.

4.2.1 Système de fichiers

Pour pouvoir être utilisé, le périphérique doit contenir un système de fichiers valide. Pour ce projet, un système de fichiers FAT a été utilisé. Originalement développé pour être utilisé sur les disquettes, il a l'avantage de ne pas nécessiter beaucoup de méta-données pour fonctionner. Seulement 4 blocs de 512 bytes sont nécessaires pour décrire le système de fichiers.

Afin de générer ce système de fichiers, la suite d'outils *mtools*[6] contient fournit le programme *mformat*. Il permet de générer un fichier "image" contenant les données du système de fichiers. On peut ensuite récupérer les méta-données du système de fichiers présent dans l'image et les intégrer dans notre programme.

4.2.2 Synchronisation

Afin de pouvoir cadencer le programme du Raspberry Pi par rapport au rythme réel d'affichage des images par le Blue Pill, il est nécessaire de mettre en place une mécanique permettant d'attendre et de confirmer que l'image précédente a bien fini d'être affichée avant de passer à l'image suivante. Pour ce faire, le Blue Pill va volontairement attendre avant d'acquitter l'opération d'écriture dans le fichier lorsque le Raspberry Pi demande de passer à l'image suivante.

Sur le Raspberry Pi, le programme écrit dans le fichier pour contrôler le système et doit attendre que l'écriture soit finie avant de passer à la suivante. Mais lors d'une écriture dans un fichier, le système Linux ne va, en principe, pas réellement envoyer ces données au périphérique. La plupart du temps, il va stocker la modification en local et ne garantit pas quand le périphérique recevra l'information. Pour forcer l'envoi des données, Linux fournit la fonction *fsync*[7]. En plus de forcer l'envoi, cette fonction est bloquante jusqu'à ce que le transfert soit acquitté. Ainsi, le Blue Pill peut bloquer le programme du Raspberry Pi autant de temps qu'il faut pour que l'image précédente ait fini d'être affichée.

La figure 4.3 montre les actions que doit effectuer le programme pour s'assurer que l'écriture dans un fichier a bien été effectuée et que la synchronisation avec l'affichage est garantie :

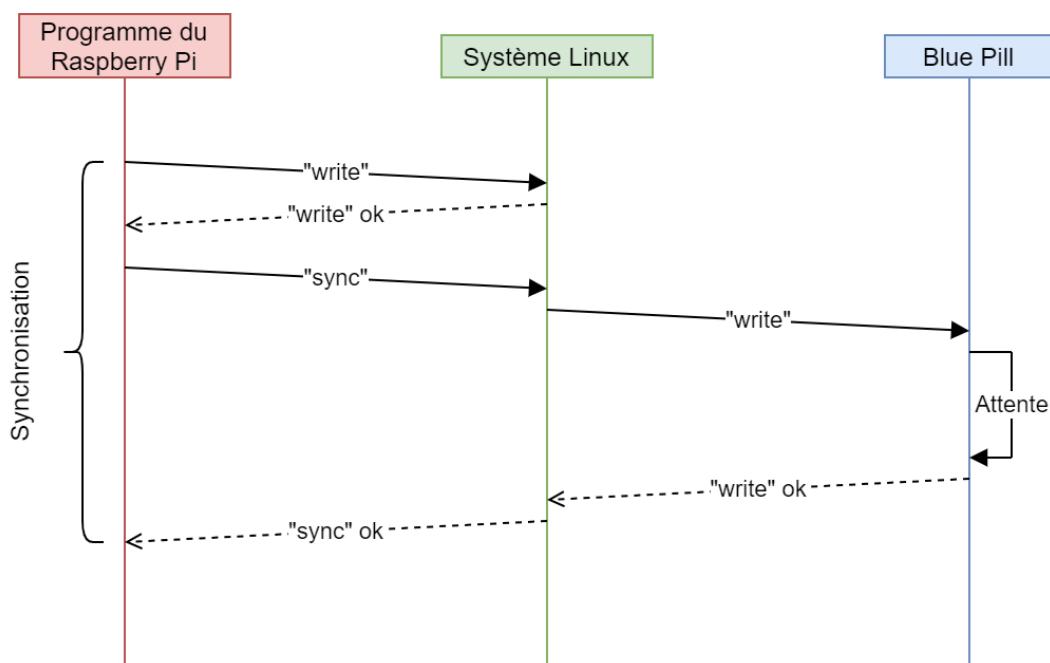


FIGURE 4.3 – Diagramme de séquence de l'écriture dans un fichier du Blue Pill

4.3 Compression des données

Afin d'augmenter la quantité de données transmises du Raspberry Pi vers le Blue Pill, il est utile de les compresser au préalable. Une méthode de compression simple, rapide, sans perte et efficace pour ce projet est le **codage de Huffman**¹.

4.3.1 Code de Huffman

Il s'agit d'un *code préfixe*² qui se base sur la fréquence d'apparition de chaque symbole de l'alphabet utilisé afin d'associer aux symboles les plus fréquents des codes plus courts, et aux symboles les plus rares des codes plus longs.

Pour décompresser les données du côté Blue Pill, il est nécessaire d'avoir reçu au préalable le code utilisé. En général, un code de Huffman particulier est représenté sous forme d'arbre. En fonction du code, cet arbre peut avoir des formes très différentes, et être de taille conséquente. L'envoi de cet arbre peut donc être un coût en temps non négligeable.

4.3.2 Code de Huffman canonique

Afin de réduire le temps requis pour l'envoi du code utilisé, un type particulier de code de Huffman peut être utilisé : un code de Huffman *canonique*³. Chaque code de Huffman classique peut être converti en un code de Huffman canonique équivalent en terme de performance de compression, mais beaucoup plus compact à transmettre. De plus, l'utilisation d'un code de Huffman canonique permet une implémentation très efficace de l'algorithme de décompression, ce qui permettra d'améliorer les performances du système.

En effet, avec un code de Huffman canonique, l'arbre entier n'a pas besoin d'être transmis. Le destinataire n'a besoin que de deux groupes d'information :

- a) Pour chaque taille possible, le nombre de codes ayant cette taille (par exemple : 1x code de taille 2, 0x code de taille 3, 2x code de taille 4, ...)
- b) La liste des symboles de l'alphabet utilisé, triés dans l'ordre de la taille de leur code (par exemple : d'abord les symboles associés à un code petit, ensuite les symboles associés à un code plus grand)

Le destinataire peut retrouver la correspondance entre chaque symbole et son code en se basant uniquement sur ces deux éléments.

1. https://en.wikipedia.org/wiki/Huffman_coding

2. https://en.wikipedia.org/wiki/Prefix_code

3. https://en.wikipedia.org/wiki/Canonical_Huffman_code

4.3.3 Alphabet utilisé

Dans ce projet, l'alphabet naturel des données envoyées est l'ensemble des bytes possibles. En effet, les codes couleurs sont composés de 3 bytes représentant les valeurs de "R", "G" et "B". Si l'image comporte une couleur répétée de nombreuse fois (par exemple un fond de couleur unie), le taux de compression sera très bon.

On peut également noter que l'utilisation de la correction des couleurs (chapitre 2.1.3) permet de profiter encore plus de cette compression puisque cela réduit le nombre de bytes différents possibles.

L'alphabet contient donc 256 symboles différents. Avec un code de Huffman canonique pour un alphabet de 256 symboles, nous pouvons connaître à l'avance la taille des informations décrivant le code à envoyer au Blue Pill :

- a) La taille minimale d'un code est de 1 bit, la taille maximale est de 255 bits. Le compte maximum pour une certaine taille est 255 (sauf dans le cas exceptionnel 4.3.4 où tous les symboles ont la même fréquence). On peut donc envoyer cet élément en 255 bytes.
- b) Chaque symbole doit être envoyé. On peut donc envoyer cet élément en 256 bytes.

On peut donc décrire le code utilisé en seulement 511 bytes, ce qui tient dans un seul bloc de notre système de fichiers, tout en gardant un byte réservé pour activer ou désactiver la compression. Cela correspond aux tailles prévues dans le chapitre 3.2.2.

4.3.4 Cas exceptionnel

Un cas particulier du code de Huffman est lorsque tous les symboles ont exactement la même fréquence dans les données. Dans ce cas, les données encodées font la même taille que les données décodées (aucune compression, l'encodage ne sert à rien).

Lorsqu'on est dans cette situation, il vaut donc mieux désactiver l'utilisation du codage de Huffman et directement transmettre les données brutes, afin d'économiser le temps d'encodage et de décodage.

4.4 Réduction des données

Afin d'augmenter le nombre de LEDs utilisables par le système et le nombre d'images par secondes, une réduction des données peut être effectuée. Il suffit d'envoyer et stocker uniquement les 4 MSB de chaque byte des codes des couleurs. Ainsi, on peut transmettre et stocker les données de deux fois plus de LEDs.

Cette réduction des données a le désavantage de réduire la fidélité des couleurs : lorsqu'elle est utilisée, seulement 4096 couleurs différentes sont disponibles.

4.5 Bit Banding

Le "Bit Banding" est une fonctionnalité présente sur certains CPU ARM, notamment le STM32F103C8T6 du Blue Pill (Cortex-M3). Cela permet de mapper une région de la mémoire (en l'occurrence, toute la RAM) à une autre région (à des adresses virtuelles). Chaque bit de la RAM est mappé sur le LSB d'un `uint32_t` à une adresse virtuelle. Cela permet de modifier ou lire des bits individuels dans la RAM en une seule instruction (on n'a pas besoin d'utiliser des masques et des shifts). Grâce à cette technique, on peut améliorer la performance d'un programme, ou encore éviter des accès concurrents critiques.

Pour le Blue Pill, la région qui commence à 0x22000000 (la RAM) est mappée à la région d'adresses virtuelles commençant à 0x2000800 (4 bytes par bit de la RAM) (figure 4.4) :

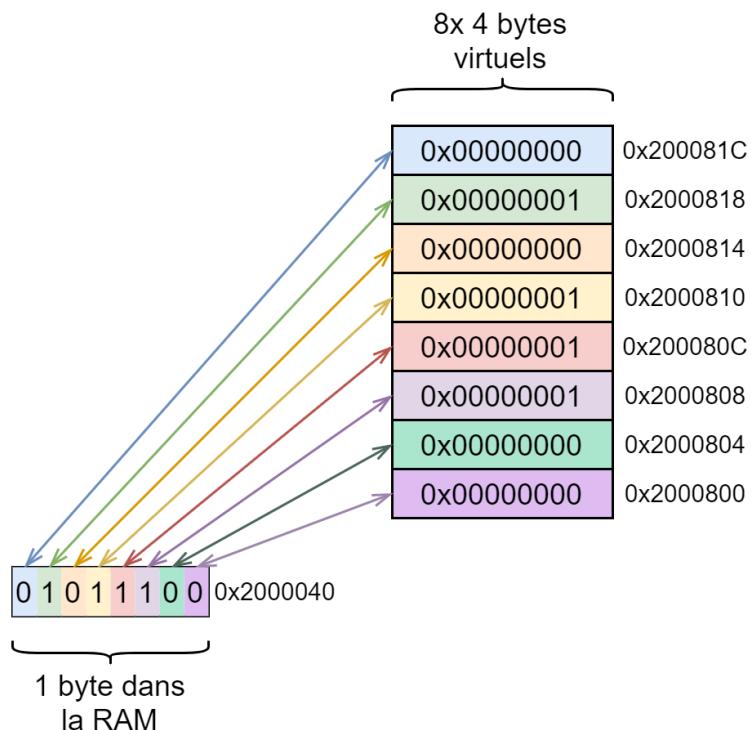


FIGURE 4.4 – "Bit banding" pour un byte de la RAM

Chapitre 5

Implémentation

Ce chapitre décrit la structure et les choix d'implémentation des différentes parties du projet.

5.1 Environnement de développement

Tout le code du système a été réalisé dans VS Code (v1.46.1)¹.

La partie embarquée a été réalisée à l'aide de l'extension PlatformIO (v4.3.4)². Il s'agit d'une plateforme qui s'intègre dans l'IDE (par exemple VS Code ou Atom), et qui offre des outils de développements pour des systèmes embarqués. Il est compatible avec des centaines de microcontrôleurs, notamment le Blue Pill STM32F103C8T6 utilisé dans ce projet.

5.2 Programme du Blue Pill

Le programme du Blue Pill (*realisation/device/*) met en oeuvre les différents composants du STM32F103C8T6 afin de réaliser un périphérique permettant de contrôler les LEDs. Il est écrit en C.

5.2.1 Composants

Son code est séparé en différents modules réalisant les différentes fonctionnalités requises :

main

Ce module contient la fonction *main* du programme. Il s'occupe d'initialiser les autres modules, puis il fait clignoter la LED intégrée du Blue Pill dans une boucle infinie afin d'indiquer que le programme est en marche. La plupart du travail de ce programme est réalisé dans des interruptions, c'est pourquoi la boucle principale ne fait rien d'utile pour le contrôle des LEDs.

1. <https://code.visualstudio.com>
2. <https://platformio.org/>

leds

Ce module permet de générer le signal des LEDs en utilisant un timer, le module DMA et les GPIOs comme indiqués dans le chapitre 4.1.4. Il est basé sur l'implémentation de la librairie "improved stm32 ws2812b dma library" (Martin Hubacek)[5].

Lors de l'initialisation, il configure le timer, les trois canaux DMA ainsi que les GPIOs utilisées. Il envoie ensuite une première image aux bandes de LEDs afin d'éteindre tous les pixels.

Il fournit des fonctions permettant d'envoyer une image aux LEDs, de configurer le système, et d'attendre certains moments clés afin de permettre la synchronisation de l'affichage avec l'écriture dans les fichiers du système de fichiers virtuel.

usb

Ce module permet de gérer la lecture et l'écriture sur le système de fichiers virtuel lorsque le Raspberry Pi communique par USB avec le Blue Pill. Il fournit deux fonctions ("read" et "write") qui sont appelées par l'implémentation bas niveau du périphérique USB fournie par STMicroelectronics.

Afin de simuler le système de fichiers, il gère les blocs de la façon suivante :

- blocs 0 à 3 : méta-données du système de fichier, ne peuvent pas être écrites, sont lues depuis un tableau constant dont le contenu est copié d'un fichier "image" contenant un système de fichiers FAT généré avec mtools[6]
- bloc 4 : fichier "config", permet au Raspberry Pi de configurer les paramètres du système
- bloc 5 : fichier "coding", permet au Raspberry Pi d'activer et de définir le code de Huffman canonique utilisé
- bloc 6 : fichier "data", permet au Raspberry Pi d'envoyer les données des couleurs des LEDs à afficher

Une particularité de ces fonctions est qu'elles sont toujours exécutées dans une interruption ("USB_LP_CAN1_RX0_IRQHandler"). Le traitement dans ces fonctions peut être relativement long (notamment lors de l'écriture dans le fichier "data"), c'est pourquoi il est nécessaire de baisser la priorité de ce type d'interruption. Cela permet aux interruptions relatives au module "leds" de préemptivement interrompre le traitement du module "usb", afin de générer rapidement le signal des LEDs.

canonical_huffman

Ce module implémente le décodage de Huffman canonique, comme décrit dans le chapitre 4.3. Les détails de l'algorithme utilisé sont expliqués ici : http://pinop8.webfactional.com/mw/index.php?title=Canonical_Huffman_code

exception_handler

Ce module contient les "handlers" pour les interruptions levées lorsqu'une erreur est rencontrée ("HardFault", "MemManage", "BusFault" et "UsageFault"). Toutes ces interruptions appellent une même fonction, "exception_handler_default_error", qui bloque le programme dans une boucle infinie.

5.3 Système de fichiers

Afin de générer le système de fichiers utilisé dans le programme du Blue Pill, un script bash a été réalisé (*realisation/filesystem/*). Il effectue les actions suivantes :

- Créer un fichier "image" contenant un système de fichiers FAT12 vide en utilisant le programme "mformat"
- Créer trois fichiers ("config", "coding" et "data"), d'une taille de 512 bytes chacun en utilisant le programme "dd"
- Copier les trois fichiers dans le système de fichiers virtuel (l'image) à l'aide du programme "mcopy"
- Démarrer un script python créé sur mesure qui permet de "dump" les quatre premiers blocs de l'image au format d'un tableau C, qui peut être copié directement dans le code du Blue Pill
- Nettoyer (supprimer) les fichiers créés

La figure 5.1 montre le système de fichiers créé par le script (résultat de la commande `mdir -i fs.img`) :

```

Volume in drive : is TELE TOWER
Volume Serial Number is 49FE-99A4
Directory for ::/

config           512 2020-07-17  7:46
coding           512 2020-07-17  7:46
data             512 2020-07-17  7:46
      3 files          1 536 bytes
                           0 bytes free

```

FIGURE 5.1 – Système de fichiers créé

5.4 Librairies réalisées (côté Raspberry Pi)

Comme indiqué dans le chapitre 3.3, deux librairies permettant de contrôler le périphérique ont été réalisées dans ce projet (*realisation/host/*) :

- **Librairie C** : Cette librairie, réalisée en C, fournit une API inspirée de celle anciennement utilisée dans la tour télécom[9] ; elle est réalisée de façon simple et ne permet pas d'utiliser certaines fonctionnalités du périphérique
- **Librairie C++** : Cette librairie, réalisée en C++, fournit une API plus complète permettant d'utiliser toutes les fonctionnalités du périphérique

Ces deux librairies suivent exactement la spécification décrite dans le chapitre 3.3, en utilisant les techniques décrites dans les chapitres 4.2 4.3.

5.5 Programmes de démo

Afin de démontrer les fonctionnalités du projet et de fournir des exemples montrant comment utiliser les librairies réalisées, des programmes de démo ont été réalisé (*realisation/host/*) (deux versions de chacune des démos sont disponibles, une pour la librairie C et une pour la librairie C++).

5.5.1 `demo_telecom_tower_line`

Ce programme est fait pour fonctionner avec les bandes de LEDs disposées comme sur la tour télécom. Il affiche une ligne verticale qui tourne de façon cyclique en passant par chaque bande de LEDs. Il prend en paramètre l'emplacement où le périphérique a été "mount", ainsi que la couleur à utiliser pour la ligne.

5.5.2 `demo_telecom_tower_text`

Ce programme est également fait pour fonctionner avec les bandes de LEDs disposées comme sur la tour télécom. Il affiche un texte au choix qui tourne de façon cyclique en passant par chaque bande de LEDs. Il prend en paramètre l'emplacement où le périphérique a été "mount", le texte à afficher, ainsi que la couleur à utiliser pour le texte.

Afin d'allumer les pixels correctement pour afficher les caractères, la police "bitmap" 5x7 "glcdfont" réalisée par Adafruit, tirée de la librairie "Adafruit-GFX-Library"^[8], est utilisée.

Voici un exemple du résultat de ce programme, les LEDs affichent le texte "Welcome!" (figure 5.2) :



FIGURE 5.2 – Texte affiché sur les bandes de LEDs

5.6 Limites du système

Le Blue Pill comporte 20 KB de RAM, dont une partie est déjà occupée pour les besoins du programme. Le buffer contenant les données des couleurs des LEDs fait une taille de 12288 bytes ; cela limite le nombre de LEDs utilisables avec la solution développée :

- Sans la réduction des données (cf. chapitre 4.4), le nombre total maximum de LEDs est **4096** (par exemple 8 bandes de 512 LEDs, ou 1 bande de 4096 LEDs, ...)
- Avec la réduction des données, le nombre total maximum de LEDs est **8192** (par exemple 8 bandes de 1024 LEDs, ou 1 bande de 8192 LEDs, ...)

Chapitre 6

Tests et validation

Afin de valider le fonctionnement de la solution et tester ses limites, deux types de tests ont été réalisés :

6.1 Test fonctionnel

Ce test est disponible dans le programme "demo_tests" (realisation/host/cpp/). Il permet de vérifier que le système fonctionne avec toutes les configurations et les fonctionnalités disponibles. Il permet également de vérifier qu'une bande de LED fonctionne correctement.

Ce programme de tests prend les paramètres suivants :

- *path_to_device* : emplacement où le périphérique a été "mount"
- *strip_n* : nombre de bandes de LEDs en parallèle
- *led_n* : nombre de pixels sur chaque bande
- *do_gamma_correction* : active ou non la correction des couleurs (cf. chapitre 2.1.3)
- *use_data_reduction* : active ou non la réduction des données (cf. chapitre 4.4)
- *use_huffman_coding* : active ou non le codage de Huffman (cf. chapitre 4.3)
- *animation* : choisit entre le mode "couleur unie" (toutes les LEDs sont allumées avec une même couleur) ou "animation" (les LEDs sont allumées individuellement les unes après les autres)
- *r, g et b* : couleur à utiliser pour le test

Pour tester le système, ce programme a été démarré avec différents paramètres afin de vérifier chaque fonctionnalité. Les résultats de ces tests sont montrés dans la figure 6.1 :

Commande	Résultat attendu	Résultat obtenu
<code>./demo_tests /mnt/telecomtower 8 512 0 0 0 0 8 0 0</code>	Toutes les LEDs s'allument en rouge	OK
<code>./demo_tests /mnt/telecomtower 8 512 1 0 0 0 32 0 0</code>		OK
<code>./demo_tests /mnt/telecomtower 8 512 0 1 0 0 16 0 0</code>		OK
<code>./demo_tests /mnt/telecomtower 8 512 0 0 1 0 8 0 0</code>		OK
<code>./demo_tests /mnt/telecomtower 8 512 1 1 1 0 94 0 0</code>		OK
<code>./demo_tests /mnt/telecomtower 8 512 0 0 0 1 128 0 0</code>	Les LEDs s'allument individuellement en rouge les unes après les autres	OK
<code>./demo_tests /mnt/telecomtower 8 512 1 0 0 1 128 0 0</code>		OK
<code>./demo_tests /mnt/telecomtower 8 512 0 1 0 1 128 0 0</code>		OK
<code>./demo_tests /mnt/telecomtower 8 512 0 0 1 1 128 0 0</code>		OK
<code>./demo_tests /mnt/telecomtower 8 512 1 1 1 1 128 0 0</code>		OK
<code>./demo_tests /mnt/telecomtower 1 512 0 0 0 0 128 0 0</code>	La première bande de LEDs s'allume en rouge	OK

FIGURE 6.1 – Résultats des tests fonctionnels

Les tests montrent que le système fonctionne sans problème dans les différentes configurations possibles.

6.2 Test des performances

Afin de mesurer les performances du système, un programme "demo_benchmark" (realisation/host/cpp/) a été réalisé. Il permet de mesurer les images par secondes et la quantité de données transférée vers le Blue Pill avec toutes les configurations et les fonctionnalités disponibles.

Ce test fonctionne de la façon suivante : un certain pourcentage (configurable) des pixels prend une couleur unie ("arrière-plan"), et les autres prennent une couleur aléatoire dans des bornes définies (configurables). Il envoie ensuite en boucle l'image générée, et calcule le temps écoulé et la quantité de données transmises.

Ce programme de mesure de performances prend les paramètres suivants :

- *path_to_device* : emplacement où le périphérique a été "mount"
- *strip_n* : nombre de bandes de LEDs en parallèle
- *led_n* : nombre de pixels sur chaque bande
- *do_gamma_correction* : active ou non la correction des couleurs (cf. chapitre 2.1.3)
- *use_data_reduction* : active ou non la réduction des données (cf. chapitre 4.4)
- *use_huffman_coding* : active ou non le codage de Huffman (cf. chapitre 4.3)
- *recompute_code* : si "use_huffman_coding" est activé, permet de choisir si le code doit être recalculé et ré-envoyé à chaque image
- *random_colors* : indique si les pixels doivent prendre une couleur aléatoire
- *background_percentage* : pourcentage de pixels "arrière-plan" (couleur unie)
- *max_r* : maximum pour la composante "R" dans les couleurs aléatoires
- *max_g* : maximum pour la composante "G" dans les couleurs aléatoires
- *max_b* : maximum pour la composante "B" dans les couleurs aléatoires
- *background_r* : composante "R" pour la couleur de l'arrière-plan
- *background_g* : composante "G" pour la couleur de l'arrière-plan
- *background_b* : composante "B" pour la couleur de l'arrière-plan

Afin d'obtenir des résultats pour des situations variées, les tests réalisés ont porté sur différents points :

- Activation de la réduction des données
- Activation du codage de Huffman
- Nombre de couleurs différentes dans l'image
- Nombre de bandes de LEDs
- Nombre de LEDs sur chaque bande
- Version du Raspberry Pi (3B+ et 4)

Voici les résultats des tests de performance (figure 6.2) avec un très grand nombre de LEDs (8 bandes de 512 LEDs) :

	Nombre de canaux	Nombre de LEDs	Correction gamma	Réduction des données	Codage de Huffman	Recalculer le codage de Huffman	Couleurs aléatoires	Pourcentage d'arrière-plan	R/G/B (arrière-plan)	IPS max théorique	IPS (RPi3B+)	Données transmises (KB/s) (RPi3B+)	IPS (RPi4)	Données transmises (KB/s) (RPi4)
<i>Très mauvaises données (base)</i>														
./demo_benchmark /mnt/telecomtower 8 512 1 0 0 0 1 0 255 255 255 0 0 0	8	512	oui			/								
./demo_benchmark /mnt/telecomtower 8 512 1 1 0 0 1 0 255 255 255 0 0 0				non	non	/					22,5	290	30,5	388
./demo_benchmark /mnt/telecomtower 8 512 1 0 1 0 1 0 255 255 255 0 0 0				oui	non	/					37,5	250	55	368
./demo_benchmark /mnt/telecomtower 8 512 1 1 1 0 1 0 255 255 255 0 0 0				non	oui	non					12	138	17	185
./demo_benchmark /mnt/telecomtower 8 512 1 1 1 0 1 0 255 255 255 0 0 0				oui	oui	non					23,5	131	30,5	172
<i>Mauvaises données</i>														
./demo_benchmark /mnt/telecomtower 8 512 1 0 1 0 1 50 255 255 255 0 0				non	oui	non					20	132	26	174
./demo_benchmark /mnt/telecomtower 8 512 1 1 1 0 1 50 255 255 255 0 0				oui	oui	non					33	121	41,5	150
<i>Bonnes données</i>														
./demo_benchmark /mnt/telecomtower 8 512 1 0 1 0 1 50 127 127 127 0 0	~65			non	oui	non					27,5	126	34	159
./demo_benchmark /mnt/telecomtower 8 512 1 1 1 0 1 50 127 127 127 0 0				oui	oui	non					51	79	56	86
<i>Très bonnes données</i>														
./demo_benchmark /mnt/telecomtower 8 512 1 0 1 0 1 80 127 127 127 0 0				non	oui	non					39	120	44	134
./demo_benchmark /mnt/telecomtower 8 512 1 1 1 0 1 80 127 127 127 0 0				oui	oui	non					59	60	59,5	61

FIGURE 6.2 – Résultats des tests de performance avec 8 bandes de 512 pixels

Et voici les résultats des tests de performance (figure 6.3) avec un nombre réaliste de LEDs (5 bandes de 256 LEDs, objectif pour la tour télécom) :

	Nombre de canaux	Nombre de LEDs	Correction gamma	Réduction des données	Codage de Huffman	Recalculer le codage de Huffman	Couleurs aléatoires	Pourcentage d'arrière-plan	R/G/B (arrière-plan)	IPS max théorique	IPS (RPi3B+)	Données transmises (KB/s) (RPi3B+)	IPS (RPi4)	Données transmises (KB/s) (RPi4)			
Très mauvaises données (base) ./demo_benchmark /mnt/telecomtower 5 256 1 0 0 0 1 0 255 255 255 0 0 0 ./demo_benchmark /mnt/telecomtower 5 256 1 1 0 0 1 0 255 255 255 0 0 0 ./demo_benchmark /mnt/telecomtower 5 256 1 0 1 0 1 0 255 255 255 0 0 0 ./demo_benchmark /mnt/telecomtower 5 256 1 1 1 0 1 0 255 255 255 0 0 0	5	256	oui														
				non	non	/					61,5	255	96	392			
				oui	non	/					95	196	125	257			
				non	oui	non					40	145	55,5	198			
				oui	oui	non					70	144	87	178			
				non	oui	non					62	127	83,5	170			
				oui	oui	non					107,5	111	114	116			
				non	oui	non					80,5	123	97	149			
Mauvaises données ./demo_benchmark /mnt/telecomtower 5 256 1 0 1 0 1 50 255 255 255 0 0 ./demo_benchmark /mnt/telecomtower 5 256 1 1 1 0 1 50 255 255 255 0 0				oui							114	58	114	58			
				non	oui	non											
				oui	oui	non											
Bonnes données ./demo_benchmark /mnt/telecomtower 5 256 1 0 1 0 1 50 127 127 127 0 0 ./demo_benchmark /mnt/telecomtower 5 256 1 1 1 0 1 50 127 127 127 0 0	5	256	oui								80,5	123	97	149			
				non	oui	non											
				oui	oui	non											
				non	oui	non											
Très bonnes données ./demo_benchmark /mnt/telecomtower 5 256 1 0 1 0 1 80 127 127 127 0 0 ./demo_benchmark /mnt/telecomtower 5 256 1 1 1 0 1 80 127 127 127 0 0											99,5	101	111	113			
				oui	oui	non											

FIGURE 6.3 – Résultats des tests de performance avec 5 bandes de 256 pixels

Dans les deux figures précédentes, la limite théorique d'images par secondes ("IPS max théorique") est calculé par rapport aux limitations dues au protocole des contrôleurs ws2812b (limité à 800 KHz par bande).

Pour synthétiser ces tests de performance, voici les points importants :

- Dans les cas favorables (peu de couleurs différentes) ou en activant la réduction des données, on peut très facilement se rapprocher de la limite théorique du nombre d'IPS
- Même avec un très grand nombre de LEDs, des images défavorables (beaucoup de couleurs différentes) et sans réduction des données ("worst-case"), on obtient toujours un nombre d'IPS satisfaisant (> 30 IPS, ce qui permet des animations assez fluides pour notre oeil)
- Dans les cas défavorables (beaucoup de couleurs différentes), il vaut mieux désactiver le codage de Huffman car le temps gagné lors du transfert est plus faible que le temps perdu à encoder/décoder les données
- L'utilisation d'un Raspberry Pi 4 à la place du 3B+ permet de gagner des IPS dans certains cas
- Dans tous les cas, la nouvelle solution développée donne plus d'IPS que l'ancienne solution présente sur la tour télécom

Chapitre 7

Améliorations et travaux futurs

L'implémentation des fonctionnalités principales du projet est terminée, mais de nouvelles idées ont été évoquées lors des séances.

7.1 Interfaçage USB pour d'autres périphériques

Ce projet comporte des modules qui pourraient être utilisés pour d'autres projets, en particulier la partie interfaçage USB. En effet, un périphérique s'annonçant comme un périphérique de stockage et qui permet de contrôler du hardware spécifique en éditant simplement un fichier permettrait d'abstraire l'utilisation non seulement de LEDs, mais également d'autres périphériques comme par exemple des servomoteurs. Grâce à cet interfaçage, le Blue Pill peut très facilement devenir une extension pour n'importe quel PC, lui permettant de contrôler du matériel spécifique.

7.2 Extension de la capacité de la tour télécom

Comme expliqué dans l'analyse, les différents MCU de la famille des STM32F ont beaucoup en commun. Il serait aisément de remplacer le STM32F103C8T6 utilisé actuellement par un autre microcontrôleur ayant plus de RAM, dans le cas où plus de LEDs devraient être contrôlées par le système. La plupart du code sera commun entre les deux contrôleurs.

7.3 Améliorations/nouvelles librairies

Les librairies fournies dans ce projet pourraient être améliorées. Par exemple, afin de gagner en performances, il serait possible de réaliser un système multi-thread permettant de traiter des données pendant que d'autres sont en train d'être envoyées. Cela permettrait de gagner du temps, car actuellement le CPU n'est pas utilisé pendant l'attente lors d'un *fsync*.

De plus, de nouvelles librairies pourraient être créées afin de contrôler le système depuis d'autres langages. L'implémentation de ces librairies peut être facilement réalisée en se basant sur celles déjà fournies.

Chapitre 8

Mise en place et exemple d'utilisation

Ce chapitre décrit les étapes permettant d'utiliser le système réalisé pour afficher du texte sur la tour télécom grâce au programme de démo fourni (mode d'emploi).

8.1 Matériel

Afin d'utiliser le système sur la tour télécom, voici le matériel nécessaire :

- Blue Pill, câble micro-USB
- Raspberry Pi et son alimentation
- CD4050B[4] ou équivalent
- 4 matrices flexibles Neopixel 8x32, 2 matrices flexibles Neopixel 8x8
- Alimentation 5V pour les LEDs
- Divers câbles de raccordement

8.2 Branchements

Voici le montage à réaliser avec ce matériel (figure 8.1) :

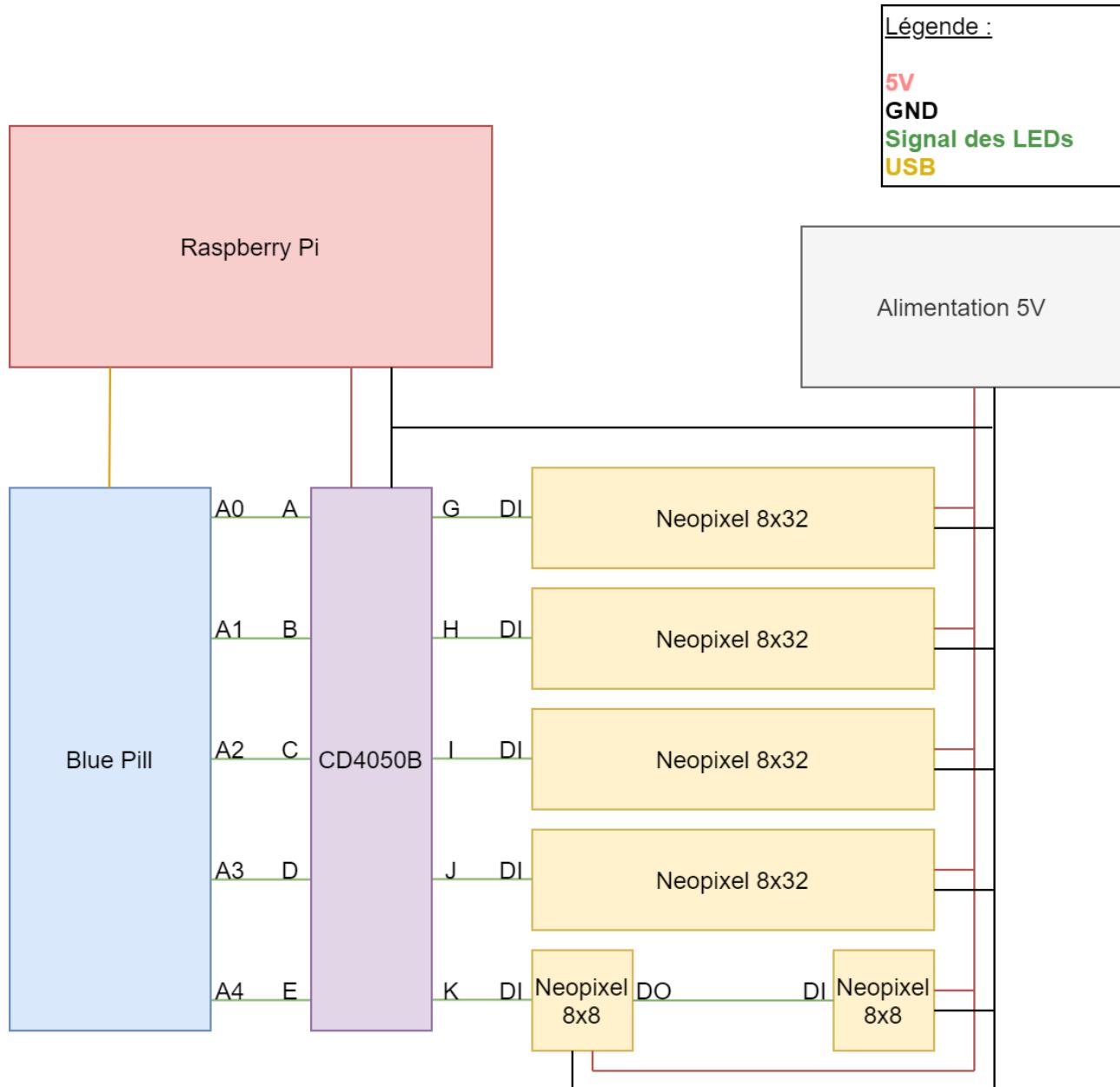


FIGURE 8.1 – Branchements à mettre en place

8.3 Démarrage du programme

Une fois les branchements réalisés, le Blue Pill est détecté par le Raspberry Pi comme un périphérique de stockage. On peut vérifier cela grâce à la commande `sudo fdisk -l` qui montre, entre autre, notre périphérique avec un système de type FAT12. On peut voir le résultat attendu dans la figure 8.2 :

```
Disk /dev/sda: 22.5 KiB, 23040 bytes, 45 sectors
Disk model: Product
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00000000

Device      Boot Start End Sectors  Size Id Type
/dev/sda1    *      0   6     7  3.5K  1 FAT12
```

FIGURE 8.2 – Portion du résultat de la commande montrant le périphérique détecté

On peut donc monter le système de fichier à l'emplacement désiré :

```
sudo mkdir -p /mnt/telecomtower
sudo mount -o umask=000 /dev/sda /mnt/telecomtower
```

Une fois cela fait, on peut constater que les trois fichiers prévus sont bien présents (figure 8.3) :

```
ls -la /mnt/telecomtower
```

```
-rwxrwxrwx 1 root root 512 Jul 16 19:38 coding
-rwxrwxrwx 1 root root 512 Jul 16 19:38 config
-rwxrwxrwx 1 root root 512 Jul 16 19:38 data
```

FIGURE 8.3 – Fichiers présents sur le système de fichiers virtuel du Blue Pill

On peut donc démarrer le programme de démo. Pour l'obtenir, il suffit d'utiliser le script fourni dans la réalisation (`realisation/host/cpp/compile_demo.sh`) afin de le compiler.

Il faut ensuite le lancer avec les paramètres désirés, voici un exemple :

```
./demo_telecom_tower_text /mnt/telecomtower "Welcome!" 32 12 0
```

8.4 Résultat

Les LEDs doivent afficher le texte "Welcome !", qui tourne de façon cyclique à travers chaque matrice, comme sur la figure 8.4 :

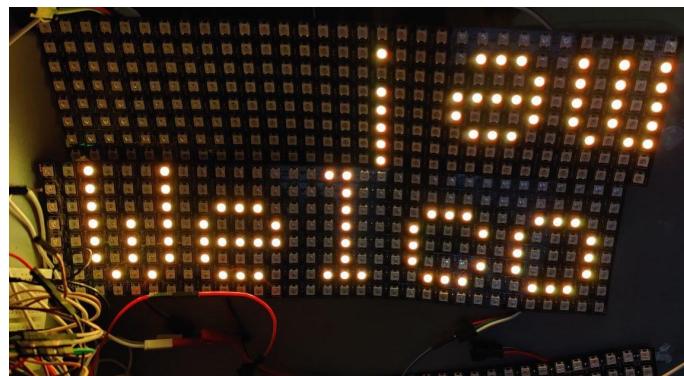


FIGURE 8.4 – Texte "Welcome !" affiché sur les bandes de LEDs

Voici un autre exemple (figure 8.5) :

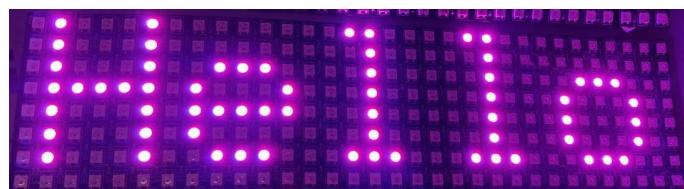


FIGURE 8.5 – Texte "Hello" affiché sur les bandes de LEDs

8.5 Résolution des problèmes

Les situations suivantes peuvent être des sources de problèmes :

- Un des câbles est mal branché / mauvais contact
- Trop de LEDs sont allumées en même temps, alimentation pas assez puissante

Les symptômes observés dans ces cas peuvent être les suivants :

- Couleurs aléatoires sur certains pixels
- Décalage au milieu de l'image
- Scintillements des LEDs
- Image qui ne change pas

Pour résoudre ces problèmes, il faut vérifier le branchement de chaque câble, brancher moins de LEDs, ou réduire la luminosité des couleurs utilisées.

Chapitre 9

Conclusion

Le produit réalisé correspond aux attentes énoncées durant le projet, il implémente les fonctionnalités demandées et respecte les contraintes imposées. Les nouvelles idées apportées tout au long du projet ont permis d'améliorer le résultat, pour finalement obtenir un système efficace et simple à utiliser.

9.1 Comparaison avec les objectifs

Comme demandé dans le cahier des charges, le système réalisé durant ce projet est une solution complète permettant de contrôler des bandes de LEDs en parallèle depuis un Raspberry Pi. Il donne de bien meilleures performances que l'ancien système, et est facilement portable pour être utilisé dans d'autres environnements.

En plus d'être un projet utile pour contrôler les LEDs de la tour télécom, ce projet peut également servir de base pour d'autres projets nécessitant de contrôler du hardware spécifique depuis un Raspberry Pi ou un PC à l'aide d'un microcontrôleur.

9.2 Rétrospective sur la planification initiale

Par rapport à ce qui était initialement prévu, la réalisation de certaines fonctionnalités a commencé plus tôt. De plus, la spécification a continué d'évoluer tout au long du projet, contrairement à ce qu'indiquait la planification.

D'une façon générale, la planification montrait une approche "waterfall", mais la réalisation du projet a pris une tournure plutôt itérative.

Finalement, le projet a pu être intégralement réalisé dans les délais impartis.

9.3 Conclusion personnelle

Ce projet a été très plaisant à réaliser, l'aspect visuel des résultats obtenus tout au long de la réalisation apportaient une bonne motivation. La réalisation de ce projet m'a permis de découvrir des technologies et des techniques très intéressantes qui pourront être utilisées dans de nombreux futurs projets. La solution créée remplit les objectifs du projet, et je suis très satisfait du résultat obtenu.

Je tiens à remercier les superviseurs du projet, Monsieur Jacques Supcik et Monsieur Michael Mäder, pour leur soutien et leur disponibilité tout au long du projet. Leurs précieuses idées et divers conseils ont permis d'améliorer grandement le projet.

Chapitre 10

Déclaration d'honneur

Je, soussigné, Nicolas Maier, déclare sur l'honneur que le travail rendu est le fruit d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toute autre forme de fraude. Toutes les sources d'information utilisées et les citations d'auteur ont été clairement mentionnées.



Glossaire

Adafruit	Entreprise concevant et vendant des produits électroniques, notamment les NeoPixels.
API	Application Programming Interface.
benchmark	Mesure des performances d'un système.
Blue Pill	Microcontrôleur équipé d'un STM32F103C8T6.
buffer	Zone mémoire permettant de stocker des informations.
DMA	Direct Memory Access.
IPS	Images par seconde.
LED	Light-Emitting Diode, source de lumière.
MCU	MicroController Unit.
microcontrôleur	Petit ordinateur sur un circuit intégré.
MSB et LSB	Most Significant Bit (bit de poids fort) et Least Significant Bit (bit de poids faible).
pixel	Point (RGB) sur une bande de LEDs adressable.
Raspberry Pi	Série de "single-board computers".
RGB	Red, Green, Blue.
USB MSC	USB Mass Storage device Class, un ensemble de protocoles de communications pour les périphériques de stockage USB.
ws2812b	Contrôleur de LEDs dans les bandes NeoPixel.
ws281x	Nom générique pour désigner les ws2811, ws2812 et ws2812b.

Bibliographie

- [1] Worldsemi, Datasheet du ws2812b
<https://cdn-shop.adafruit.com/datasheets/WS2812.pdf>, 17.07.2020
- [2] STMicroelectronics, Datasheet du STM32F103C8
<https://www.st.com/resource/en/datasheet/stm32f103c8.pdf>, 17.07.2020
- [3] STMicroelectronics, STM32F103xx reference manual
https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf, 17.07.2020
- [4] Texas Instruments, Datasheet du CD4050B
<https://www.ti.com/lit/ds/symlink/cd4049ub.pdf>, 17.07.2020
- [5] Martin Hubacek, Improved STM32 WS2812B DMA Library
<http://www.martinhubacek.cz/arm/improved-stm32-ws2812b-library>, 17.07.2020
- [6] Alain Knaff, Mtools manual
https://www.gnu.org/software/mtools/manual/html_node/, 17.07.2020
- [7] Linux fsync manual
<https://man7.org/linux/man-pages/man2/fsync.2.html>, 17.07.2020
- [8] Adafruit, Adafruit GFX Library
<https://github.com/adafruit/Adafruit-GFX-Library>, 17.07.2020
- [9] jgarff, rpi_ws281x
https://github.com/jgarff/rpi_ws281x, 17.07.2020
- [10] Espressif, ESP-IDF Programming Guide (Wi-Fi Driver)
<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html#esp32-wi-fi-throughput>, 17.07.2020