

RAPPORT D'ÉLÈVES INGÉNIEURS

PROJET DE 2<sup>ÈME</sup> ANNÉE

---

**Adaptateur Ethernet pour Game  
Boy<sup>TM</sup>**

---

*Présenté par :*

Loïck CHOVET

Eldred HABERT

*Tuteur : Alexis PEREDA*

*Responsable ISIMA : Patrice  
LAURENCOT*

*Date de soutenance : 16 mars 2020*

*Durée du projet 60h*

Campus des Cézeaux. 1 rue de la Chébarde. TSA 60125. 63178 Aubière  
CEDEX

Nous tenons à adresser nos remerciements à notre tuteur de projet, M. Alexis PEREDA, pour son aide durant la réalisation de ce projet ainsi que pour ses conseils éclairés.

Nous tenons également à remercier toutes les personnes qui nous ont aidé lors de la rédaction de ce rapport.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Travail réalisé</b>	<b>4</b>
2.1	Choix du cadre . . . . .	4
2.1.1	"Extension" du Câble Link . . . . .	4
2.1.2	Programme sur ordinateur . . . . .	6
2.2	Introduction du travail à faire . . . . .	6
2.2.1	Objectif et but . . . . .	6
2.2.2	Livrable . . . . .	6
2.2.3	Analyse . . . . .	7
2.3	Production . . . . .	8
2.3.1	Éléments matériels . . . . .	8
2.3.2	Organisation du projet . . . . .	9
2.4	Déroulement . . . . .	10
2.4.1	Découverte du protocole de communication . . . . .	10
2.4.2	Programmation sur Game Boy . . . . .	14
2.4.3	Réception et Réponse au Message SPI . . . . .	15
2.4.4	Protocole de communication . . . . .	18
2.4.5	Communication Ethernet . . . . .	19
<b>3</b>	<b>Conclusion</b>	<b>20</b>
<b>4</b>	<b>Annexes</b>	<b>21</b>

## Résumé

L'objectif de ce projet est de réaliser un prototype d'adaptateur **Ethernet** pour **Game Boy<sup>TM</sup>**. Le Game Boy dispose en effet d'un protocole, proche du **SPI**, permettant à une paire de consoles connectées par un câble propriétaire (Cable Link) de communiquer. Notre projet consiste en la conception et la programmation d'un accessoire comportant d'un côté un port Ethernet, et pouvant de l'autre être relié à un Game Boy.

La carte électronique utilisée est une **Nucleo-F746ZG** programmée en langage C++ sur l'éditeur **MBED**. Nous avons utilisé la bibliothèque Mbed.h permettant d'avoir accès à toutes les fonctionnalités de la carte électronique.

Actuellement, la partie Ethernet de l'accessoire marche, mais bien que la communication entre le Game Boy et la Nucléo soit théoriquement fonctionnelle, de nombreux problèmes électroniques nuisent à une bonne communication. Dans l'ensemble, l'accessoire est conçu mais pas fonctionnel.

Mots-clés : Game Boy, Nucleo, MBED, Ethernet, SPI

## Abstract

The goal of this project is the prototyping of an **Ethernet** adapter for **Game Boy<sup>TM</sup>**. The Game Boy implements a protocol, similar to **SPI**, allowing a pair of consoles connected through a proprietary cable (the Link Cable) to communicate. Our project boils down to designing and programming an accessory providing an Ethernet port on one end, and able to be connected to a Game Boy on the other end.

The electronic card used is a **Nucleo-F746ZG** programmed in the C++ language using the **MBED** editor. We used the Mbed.h library, which provides full support for the card's functionality.

Currently, the card is successfully able to communicate over Ethernet, and whereas the link bewteen the Game Boy and the Nucleo is theoretically functional, a number of electronic issues prevent a reliable communication. Overall, the accessory is fully designed but not working.

Keywords: Game Boy, Nucleo, MBED, Ethernet, SPI

# Lexique

Breakout : Carte électronique permettant d'avoir accès aux broches d'un port particulier sans avoir à le démonter.

Broadcast : Littéralement "diffusion" ; employé généralement pour une diffusion à grande échelle. Dans un contexte réseau, désigne un message destiné à toutes les entités connectées.

Câble Link : Câble à six fils permettant de deux Game Boy de communiquer. Les fils sont les suivants : Vcc (Alimentation, non utilisée par la Game Boy classique), SI (Signal entrant), SO (Signal envoyé), SCL (signal d'horloge), GND (Masse électrique), P14 (Inutilisé).

Ethernet : Famille de protocoles réseau bas niveau (couche 2 du modèle OSI, "liaison de données") le plus couramment utilisé pour les réseaux câblés. Les différents protocoles étant relativement transparents, ils sont souvent confondus.

Full-duplex : Désigne les situations où les deux parties d'une communication sont capables non seulement de communiquer dans les deux sens (duplex) mais de le faire simultanément (full). Opposé au simplex et au half-duplex.

Mbed : Plate-forme permettant de coder et compiler du langage C++ vers les cartes Nucleos, incluant des compilateurs et bibliothèques, ainsi qu'une interface en ligne appelée Mbed OS.

Nucléo : Carte de développement du fabricant grenoblois STMicroelectronics conçue pour le prototypage d'objets connectés. Embarque un microprocesseur STM32 alliant efficacité et basse consommation.

Round-trip time : Littéralement "temps d'aller-retour" ; en réseau, désigne le temps entre l'émission d'un message par une source et la réception de sa réponse.

To sniff : Action de capturer un trafic circulant sur un réseau. En français, on utilise "sniffer" comme un verbe pour désigner cette action.

SPI : Le Serial Peripheral Interface est un protocole de communication permettant de faire communiquer un maître et plusieurs esclaves par l'intermédiaire de 4 fils : un fil d'horloge, un fil de données du maître vers l'esclave, un fil de données en sens inverse, et un fil pour sélectionner l'esclave.

# Chapitre 1

## Introduction

Les jeux et consoles "rétro" reviennent à la mode en ce moment ; la limite entre rétro ou non est floue, mais l'appartenance des consoles 8-bit fait l'unanimité. Beaucoup reviennent sur ces vieilles plate-formes par nostalgie, d'autres par simple curiosité, mais quelques-uns s'intéressent à la création de jeux "fait maison" – une pratique connue sous le nom de *homebrew*, le terme désignant également les jeux ainsi créés.

La plupart des *homebrew* se contentent des limites techniques de l'époque, mais certains passionnés conçoivent du matériel en plus de logiciels (1).

Une partie du succès du Game Boy tenait à la possibilité du multi-joueur local qu'elle offrait, via un câble reliant deux consoles. Néanmoins, cela limite le nombre de participants à deux, et requiert une proximité physique.

Peut-on créer un accessoire qui permettrait à des *homebrew* de bénéficier d'un multi-joueur plus adapté à l'utilisation actuelle de la console ?

Notre projet aura des limites qui l'empêchent de satisfaire la plupart des cas d'usage – le domaine a été peu exploré et la documentation existante est maigre. Nous espérons faciliter des travaux futurs dans le même but, la conception d'accessoires externes pour cette console.

Ce document comprendra d'abord un résumé de notre analyse préliminaire du cadre puis du travail à effectuer, ensuite un détail de l'organisation du projet, suivi d'une explication de chacune des étapes de la réalisation. Et enfin, nous conclurons en revenant sur le travail accompli et l'avenir de ce projet.

Il sera notamment évoqué une autre piste pour le projet, les difficultés que nous avons rencontré dès la conception, et pourquoi nous ne l'avons pas choisie.

# Chapitre 2

## Travail réalisé

### 2.1 Choix du cadre

#### 2.1.1 "Extension" du Câble Link

Le cas d'utilisation principal d'un pont entre un Câble Link et un réseau serait l'utilisation de jeux déjà existants via un réseau, voire l'Internet. Ceci est dû notamment à la grande popularité de certains jeux de l'époque comme *Pokémon*, la nostalgie n'y étant certainement pas étrangère.

Le projet pourrait à première vue sembler faisable, puisque justement des accessoires comme le TCPoké (2) existent ; cependant un gros problème existe : la latence, comme illustré sur la figure 2.1.

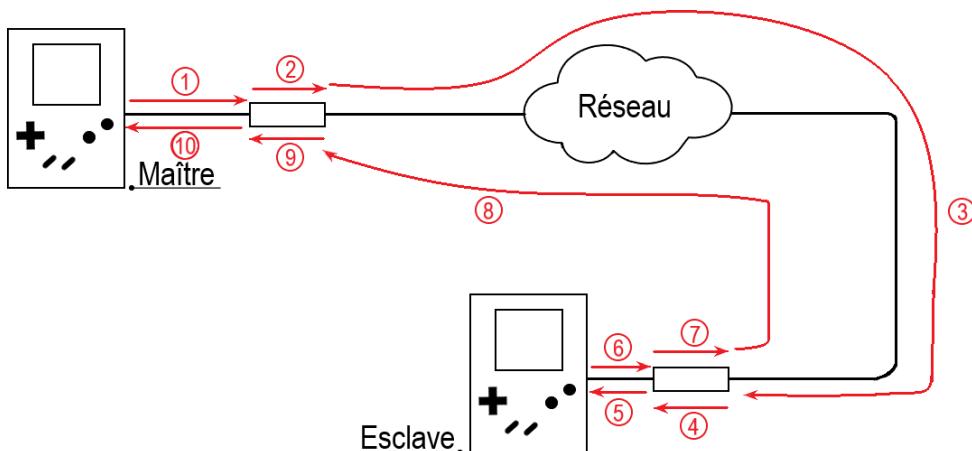


FIGURE 2.1 – Illustration du round-trip

Le protocole de transferts de bits du Game Boy est full-duplex et synchrone : pour chaque bit envoyé par le maître, un bit doit être **simultanément** envoyé par l'esclave. Il doit nécessairement y avoir une tolérance, que nous n'avons pas mesurée mais pouvons majorer par la période de l'horloge rythmant le transfert des bits. La fréquence *minimale* (le meilleur des cas pour nous, donc) est de 8 KiHz (14).

Nous disposons donc de moins de  $1/8196\text{Hz} \approx 0,12\text{ms}$  pour obtenir le bit de l'esclave à partir du moment où le maître envoie le sien. Les étapes suivantes sont impliquées :

1. Capture du bit maître
2. Encapsulation dans un paquet réseau
3. Transfert du bit maître sur le réseau

4. Désencapsulation du bit maître
5. Envoi du bit maître
6. Capture du bit esclave
7. Encapsulation dans un paquet réseau
8. Transfert du bit esclave sur le réseau
9. Désencapsulation du bit esclave
10. Envoi du bit esclave

Nous contrôlons et pouvons essayer de diminuer la durée de toutes les étapes sauf 2, 3, 4, 7, 8 et 9, qui dépendent essentiellement du réseau présent (15). Le problème est simplement que plus le réseau séparant les deux parties est complexe, plus la latence est élevée, et ce pour deux raisons. D'une part, la transmission du paquet est évidemment plus longue (étapes 3 et 8) notamment à cause du routage, mais le paquet devra être plus encapsulé (étapes 2 et 7) et désencapsulé (étapes 4 et 9), ce processus ayant lieu notamment aux points intermédiaires pour des raisons de routage.

L'Internet actuel ne permet pas à des particuliers d'obtenir une latence inférieure à la milliseconde de manière stable, nous avons donc décidé d'abandonner la possibilité de rendre notre accessoire capable de parler sur autre chose qu'un réseau local, ce qui permet de minimiser la latence des étapes 3 et 8, et de se limiter à un unique niveau d'encapsulation pour les autres. Le WiFi a été également écarté du fait d'une plus grande latence, une plus grande complexité à déboguer et une consommation énergétique plus importante ; le support du WiFi ne semble pas impossible, mais nous n'avons pas été en mesure de déterminer la faisabilité.

On pourra soulever une remarque intéressante : si les problèmes soulevés plus haut sont valables, quid du TCPoké ? Son créateur a posté une vidéo (3) où il semble bien effectuer un combat via Internet. Nous ne l'accuserons pas de triche, simplement de chance : on notera que son projet ne fonctionne qu'avec les jeux Pokémon. Pourquoi ?

On remarque dans le fichier `constants/misc_constants.asm` (4) ligne 193 la constante `SERIAL_DATA_NO_BYTE`, et une constante identique dans le fichier `tcpoke_teensy/tcpoke_teensy.ino` (5) ligne 7. Le protocole utilisé par les jeux Pokémon sur Game Boy possède un octet "vide", ignoré par le jeu qui le reçoit. (Pour transmettre des octets ayant cette valeur, un autre octet est transmis à la place, et une liste des index des octets à transformer en la valeur "vide" est transmise à la fin.)

Pour résumer, le TCPoke repose sur une particularité du protocole transporté par le protocole série de la Game Boy, et ne fonctionne pas de manière générique. Les jeux ne sont pas forcés de posséder un tel octet, et il n'existe aucune convention sur la valeur de cet octet. Il serait gênant pour l'utilisateur de devoir indiquer à l'adaptateur quel octet utiliser, et la méthode ne serait pas compatible avec tous les jeux. Nous avons choisi de ne pas nous orienter dans cette direction puisque le TCPoke est une implémentation

complète pour le jeu le plus populaire, et nous aurions probablement été limités à une imitation de ce que cet accessoire fait.

### 2.1.2 Programme sur ordinateur

Puisque nous avons écarté la possibilité d'un accessoire transparent pour la console, nous pouvons donc créer un simple adaptateur qui permettrait à un ordinateur de dialoguer avec un Game Boy. Ce n'est pas intéressant, simplement du fait qu'une Game Boy est une console **portable**, et son intérêt principal réside dans le fait de pouvoir y jouer partout. On remarque que cette tendance est toujours d'actualité notamment dans les données de vente de la Nintendo Switch (6; 7).

Nous avons donc jugé contre-productif de nuire à la portabilité de la solution ; certes, la console doit être branchée à un réseau Ethernet, mais ce n'est pas très compliqué à créer (pour deux consoles, un câble suffit, et pour plus, il existe des petits switches qui pourraient probablement être alimentés par batterie. De plus, le support du WiFi est une extension qui semble réalisable, mais que nous n'avons pas eu le temps d'explorer.

On pourra citer l'alimentation de l'accessoire comme problématique, mais nous nous orientons sur des solutions à faible consommation énergétique, qui pourraient donc soit fonctionner sur piles (ou accumulateurs) comme la plupart des accessoires de l'époque, par exemple le Pocket Printer. L'accessoire étant relié par un câble Ethernet, il serait possible d'exploiter le *Power over Ethernet* (8).

On l'aura compris, une de nos contraintes majeures est la transportabilité, ce qui rend notre projet proche de l'Internet des objets – bien que notre accessoire soit incapable de passer par l'Internet, il forme un réseau d'objets à faible consommation énergétique.

## 2.2 Introduction du travail à faire

### 2.2.1 Objectif et but

L'objectif de ce projet est de réaliser un boîtier permettant à deux Game Boy ou plus de communiquer à travers un réseau local Ethernet. Cela permettrait à des jeux écrits pour de proposer du multijoueur filaire local à deux joueurs ou plus, sans passer par le rare et inconvenant adaptateur officiel 4 joueurs (9).

### 2.2.2 Livrable

On attend comme production :

- Une carte informatique où peut être branchée un câble Ethernet et un câble Link permettant à un Game Boy de communiquer avec un autre ;
- Une bibliothèque de code Game Boy permettant d'interfacer avec la carte, et/ou une spécification permettant à des utilisateurs d'écrire leur propre code.

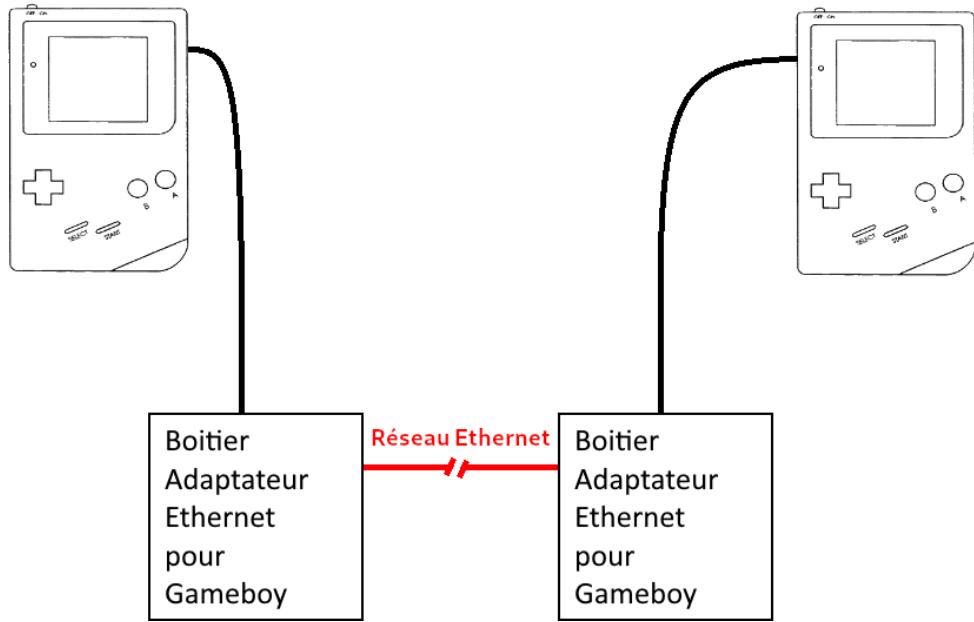


FIGURE 2.2 – Schéma de cas standard d'utilisation du système

### 2.2.3 Analyse

#### Utilisation standard du système

Deux utilisateurs (ou plus) souhaitent jouer ensemble via un réseau. Pour cela, ils branchent chacun leur boîtier à leur Game Boy et à un même réseau Ethernet ; le jeu commande le boîtier auquel il est relié afin que les deux jeux se connectent. La figure 2.2 schématisé une telle utilisation pour deux consoles.

#### État de l'art

Peut-être peu surprenamment pour un domaine concernant uniquement des amateurs, nous n'avons trouvé qu'un seul projet similaire au nôtre, l'Arduinoboy (10). Le TCPoké (2) a été mentionné plus haut, en même temps que ce pourquoi nous ne pouvons pas nous y référer. Nous sommes également tombés sur un projet dont l'auteur porte le pseudonyme Dhole (et semble tenir à son anonymat), dont le but était de capturer (sniffer) le trafic sur un Câble Link (11), puis un autre (12) qui utilise une Nucleo pour simuler le comportement d'un accessoire, le Game Boy Printer. Ce dernier projet a semblé prometteur comme inspiration, mais il se contente de reproduire le comportement d'un accessoire, là où nous voulons concevoir un accessoire entièrement nouveau, et notamment employant un nouveau protocole.

Pour en revenir à l'Arduinoboy, l'ensemble semble similaire à ce que nous avons : un Game Boy dialoguant avec un accessoire externe. Malheureusement, il ne possède pas de documentation ni de blog ou autre décrivant sa conception, et nous n'avons pas obtenu d'informations utilisables dans son code que nous ne trouvions pas sous une meilleure forme ailleurs. Pour ces raisons, et parce que nous voulions partir de zéro nous-mêmes

afin de développer des solutions peut-être novatrices, nous avons décidé de ne pas nous inspirer de code déjà existant.

## Éléments et contraintes

Nous avons mentionné plus haut les deux éléments faisant explicitement partie du livrable, mais il nous faudra aussi concevoir deux protocoles de communication, un par câble : un entre la Nucleo et la console, un entre les Nucleo. Nous voulons un fonctionnement dans l'ensemble rapide et fiable, car cela nous semble le plus adapté au jeu.

## 2.3 Production

### 2.3.1 Éléments matériels

#### Connexion au câble Link

Après une rapide réflexion, nous avons décidé qu'il serait plus approprié de proposer une prise femelle plutôt que d'intégrer un câble directement à l'accessoire. D'une, cela simplifie la manufacture ; de deux, cela ressemblait à une des erreurs commises par le DMG-07 (9) ; de trois, nous pensons que la plupart des utilisateurs visés possèdent déjà un câble. Il nous fallait donc, ne serait-ce que le temps de l'analyse, un breakout.

Le seul trouvable en ligne (figure 2.3) est proposé sur le site "Tindie" par l'utilisateur Vaguilar au prix de 3\$ (plus 13\$ de frais de port) ; nous avons préféré créer nous même notre breakout. En effet, cela nous permettait d'abord de l'obtenir rapidement, en demandant l'usinage à l'école d'ingénieurs voisine, et même d'en fabriquer plusieurs versions le cas échéant ; ensuite, cela diminuait drastiquement le prix ; enfin, la commande n'aurait pas pu être passée par l'ISIMA, le site Tindie n'étant pas considéré comme "de confiance".



FIGURE 2.3 – Breakout trouvable en ligne

#### Carte Nucleo

Nous avons décidé d'utiliser une carte Nucleo car celles-ci sont bien moins coûteuses que les cartes Arduino, de l'ordre de 5 fois moins cher, et plus adaptables aux différentes situations. Pour la modèle, la carte Nucleo F746ZG dispose d'un port Ethernet, essentiel à notre projet, d'une consommation électrique faible, ainsi que d'une fréquence processeur pouvant dépasser les 200 MHz, ce qui serait confortable face aux 8 KiHz du Game Boy.



FIGURE 2.4 – Carte Nucleo F746ZG

En attendant l'arrivée de cette carte, nous avons effectué nos tests initiaux sur une autre carte Nucleo empruntée.

### 2.3.2 Organisation du projet

#### Fonctionnement de l'équipe

Notre groupe étant composé de Loïck CHOVET, étudiant orienté informatique embarquée, et d'Eldred HABERT, également étudiant mais orienté sécurité et réseaux, l'interface entre la Nucleo et la console a été assignée à Loïck, celle entre les Nucleo et le réseau à Eldred. La programmation de la partie Game Boy initiale a été donnée à Eldred au vu de ses connaissances initiales, et devant les difficultés rencontrées par Loïck sur sa partie, il s'est également occupé du reste de la partie Game Boy. La conception des différents protocoles a été un effort commun, ce qui a été bénéfique au vu des disciplines impliquées, mais également puisque certaines des meilleures idées ont émergé de discussions.

#### Tâches à réaliser

Après décision du sujet est venue la phase de recherche de documentation. Un des premiers points-clé était la communication du Game Boy, qui allait définir une grande partie du projet, comme discuté dans la partie Analyse. Il fallait aussi concevoir un break-out afin de pouvoir travailler avec le signal, d'abord à des fins de recherche (notamment pour confirmer et supplémenter la documentation, qui n'avait aucun caractère officiel) puis d'utilisation. Une paire de logiciels simples a été conçue afin de fournir des communications plus contrôlées et simples que ce que les jeux commerciaux offraient. (Eldred disposait déjà de matériel permettant d'exécuter des logiciels maison sur la console.)

En parallèle avait lieu la conception des différents protocoles (notamment à mesure que la recherche de la communication de la console progressait). Suite à cela, l'implémentation des protocoles pouvait débuter, suivie de la création d'un jeu simple permettant de tester le bon fonctionnement du système.

#### Diagrammes de Gantt

Sur la figure 2.5, le travail en binôme est indiqué en bleu, le travail à réaliser par Eldred en vert et par Loïck en rouge. Le premier diagramme est le prévisionnel et le second le réel.

On peut y constater une nette différence entre le théorique et le réel. En effet, nous n'avions pas anticipé l'impact de problèmes électroniques rencontrés par Loïck sur tout le travail : nous avons passé beaucoup de temps à essayer différentes solutions théoriques, sans progresser sur les autres tâches ; nous ne pouvions pas passer à autre chose, car une communication fiable entre la console et la Nucléo était une des briques de base du projet.

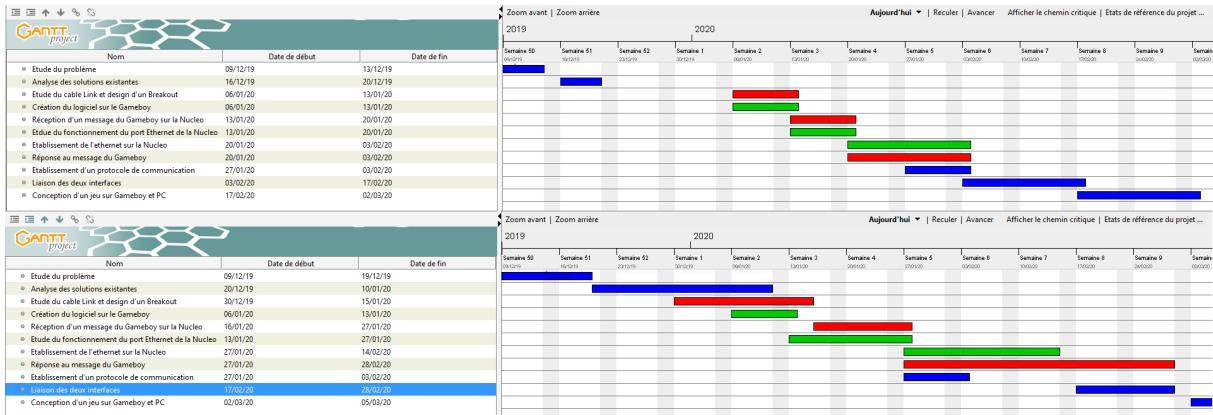


FIGURE 2.5 – Diagramme de Gantt théorique (en haut) et réel (en bas)

En outre, le temps nécessaire à l'obtention de la Nucleo a été bien plus large que prévu, ce qui a longtemps empêché d'éprouver le travail théorique effectué, particulièrement pour la partie liée à l'Ethernet. Nous nous sommes servis entre-temps d'une carte empruntée, mais d'un autre modèle ; le changement de carte a été rapide mais n'était pas initialement prévu.

## 2.4 Déroulement

### 2.4.1 Découverte du protocole de communication

#### Analyse documentaire

La majorité de notre travail de recherche documentaire est basé sur le travail du collectif GBDev, un groupe de passionnés ayant pour objectif de comprendre le Game Boy dans son entier. La documentation technique étant peu étayée, il nous a fallu confirmer par nos moyens les informations que nous avons extraites. La ressource la plus importante se nomme les Pan Docs (13).

Ainsi, pour le protocole de communication, nous nous sommes intéressé à la section nommée "*Serial Data Transfer (Link Cable)*" (14). On peut y lire :

One Gameboy acts as the master, uses its internal clock, and thus controls when the exchange happens. The other one uses an external clock (i.e., the one inside the other Gameboy) and has no control over when the transfer happens.

Cette méthode correspond au protocole SPI, qui est un protocole de communication permettant de faire communiquer un maître et plusieurs esclaves. Dans ce cas particulier, cependant, il n'y a qu'un seul esclave, ce qui explique l'absence dans le câble Link de fils de choix d'esclave, comme montré par le tableau 2.1, extrait de la section "Link Port" des Pan Docs.

On peut aussi lire dans cette documentation que la fréquence de l'horloge interne du Game Boy est, dans le cas normal, de 8 KiHz. Sur certains modèles de Game Boy (Color),

Broche	Nom	Utilité
1	Vcc	Câble d'alimentation +5V, non utilisé
2	Sout	Sortie de données
3	Sin	Entrée de données
4	P14	Non utilisé
5	SCK	Câble d'horloge
6	GND	Masse

TABLE 2.1 – Correspondance broche/fil

Les numéros sont assignés comme suit : en regardant la prise femelle du Game Boy, côté plat sur le dessus, 2,4,6 sur la rangée du dessus, 1,3,5 sur la rangée du dessous.

Il est possible d'augmenter cette fréquence, cependant, afin d'alléger le travail demandé à la Nucleo, nous restons à la fréquence la plus basse. Le support de fréquences plus élevées, en priorité 16 KiHz à cause du mode "double vitesse" du Game Boy Color, est une possibilité d'extension future.

Deux autres données sont très importantes pour modéliser le protocole du Game Boy par un SPI : le mode SPI. Celui-ci correspond à une combinaison de deux facteurs, CPOL et CPHA, décrite dans le tableau 2.2.

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

TABLE 2.2 – Modes SPI

CPOL (Clock Polarity) donne l'état de l'horloge au repos : 0 pour "bas", 1 pour "haut". CPHA (Clock Phase) donne le front où la valeur est lue : 0 pour le premier, 1 pour le second.

On peut lire que la position au repos de l'horloge est l'état haut ("[the clock pin] stays high when not used"), on sait donc que CPOL = 1. Cependant aucune information n'est trouvable quand à CPHA, il nous reviendra alors de la déterminer. On devra donc tester les modes 2 et 3.

Nous cherchons à rapprocher le protocole du Game Boy d'un protocole existant dans l'espoir d'utiliser du matériel et logiciel pré-existant adapté, ce qui simplifiera notre travail.

## Création du Breakout

Dans le but de faire de premières analyses rapides, nous avons réalisé un prototype de breakout constitué d'un morceau de carton et d'un ensemble de fils collés au bon endroit par rapport au port du Game Boy (d'après la documentation citée précédemment), photographié sur la figure 2.6

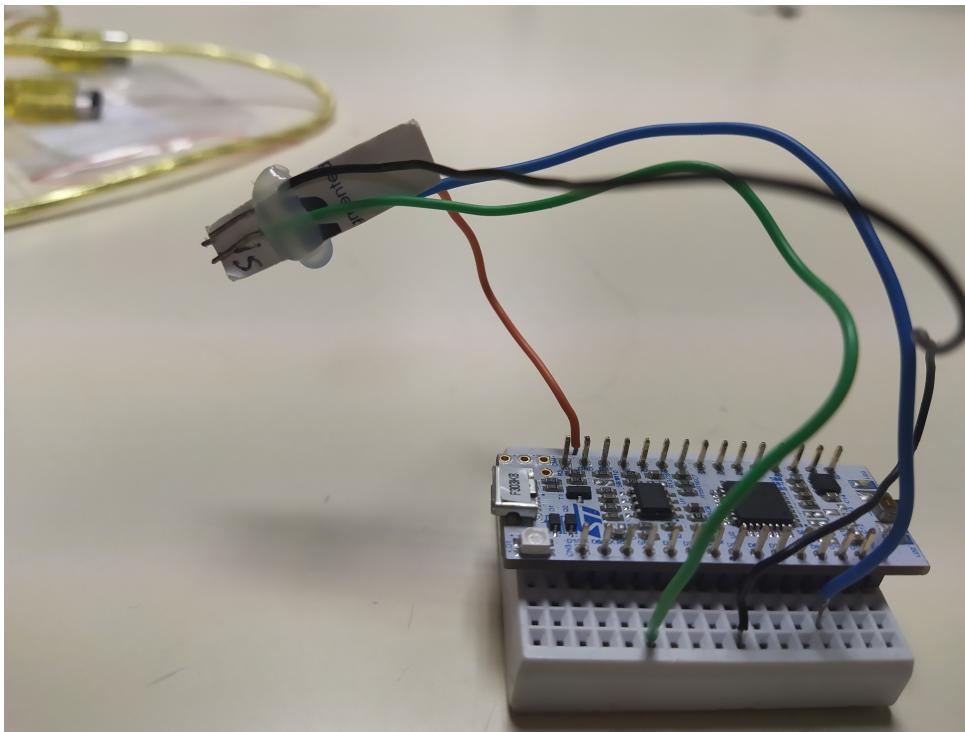


FIGURE 2.6 – Premier prototype de breakout

Cette première mouture nous a permis de commencer à analyser les trames émises par le Game Boy avec un analyseur logique. Cependant il nous fallait une solution plus propre, nous avons donc conçu une carte électronique qui serait usinée à Polytech, l'école d'ingénieurs voisine, en utilisant l'éditeur en ligne "Easy EDA", très similaire à des logiciels comme Eagle. La première version obtenue est présentée sur la figure 2.7.

Sa fabrication a rencontré un problème important : la machine de Polytech est incapable d'imprimer correctement des pistes se chevauchant. Il a donc fallu concevoir une autre version. Cela a aussi été l'occasion de parfaire nos mesures, les longueurs des broches étant un peu faibles.

Le plan final est présenté sur la figure 2.8, et photographié sur la figure 2.9.

Cette dernière mouture est enfin complètement fonctionnelle ; cependant, dans le cas d'une utilisation répétée, il faudrait imprimer le circuit sur une plaque plus fine. En effet, notre breakout étant légèrement trop épais, nous avons abîmé une fois le câble dans lequel nous l'avions branché.

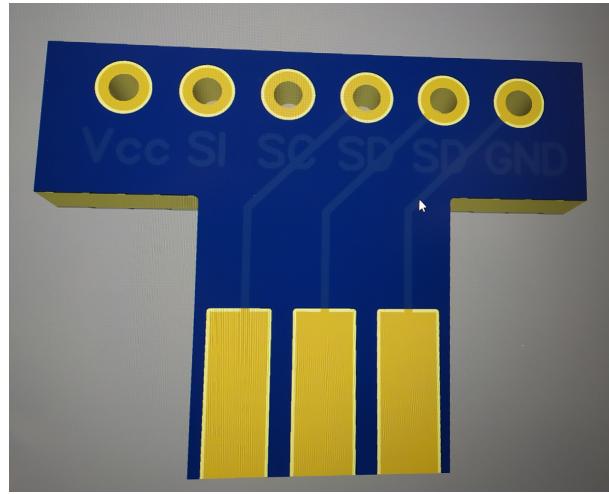


FIGURE 2.7 – Breakout Game Boy, première itération

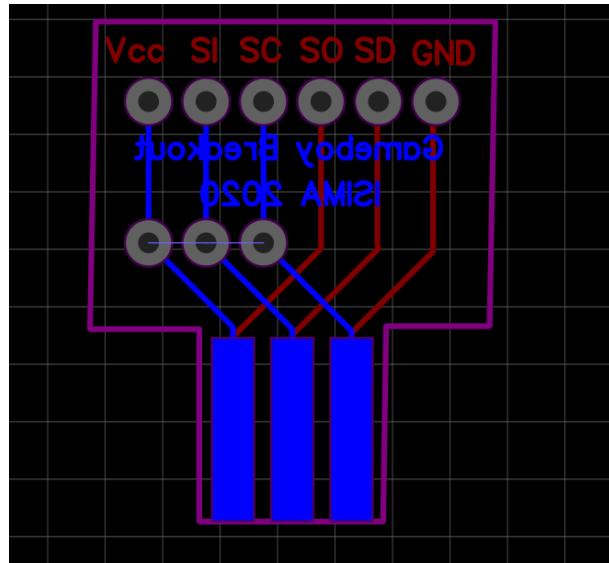


FIGURE 2.8 – Diagramme du Breakout câble Link final

### Analyse via l'analyseur logique

L'analyseur logique que nous avons utilisé, de la marque *Intronix*, et un boîtier auquel peuvent être reliés 34 câbles dont les signaux sont visibles et interprétables sur le logiciel *Intronix LogicPort Logic Analyser*, dont l'interface est visible sur la figure 2.10.

On peut voir sur la gauche les différents signaux reçus, et décider lequel servira de déclencheur à l'acquisition. Il est également possible de créer des "interpréteurs" qui décodent des signaux sélectionnés selon un modèle de communication connu, et affichent les messages correspondants directement dans l'interface. Cette fonctionnalité est particulièrement utile pour la vérification, nous avons donc écrit un programme Game Boy qui envoie périodiquement "Hello World!" sur le port série, et testé différents paramètres en connaissant le résultat attendu. Le mode SPI s'est révélé être le 3, suite à quoi nous avons affiné les autres paramètres ; il s'est notamment avéré qu'un des réglages les plus

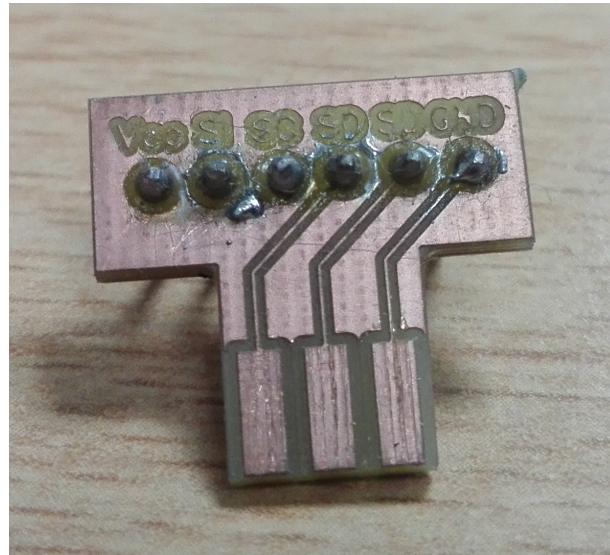


FIGURE 2.9 – Photo du Breakout câble Link final

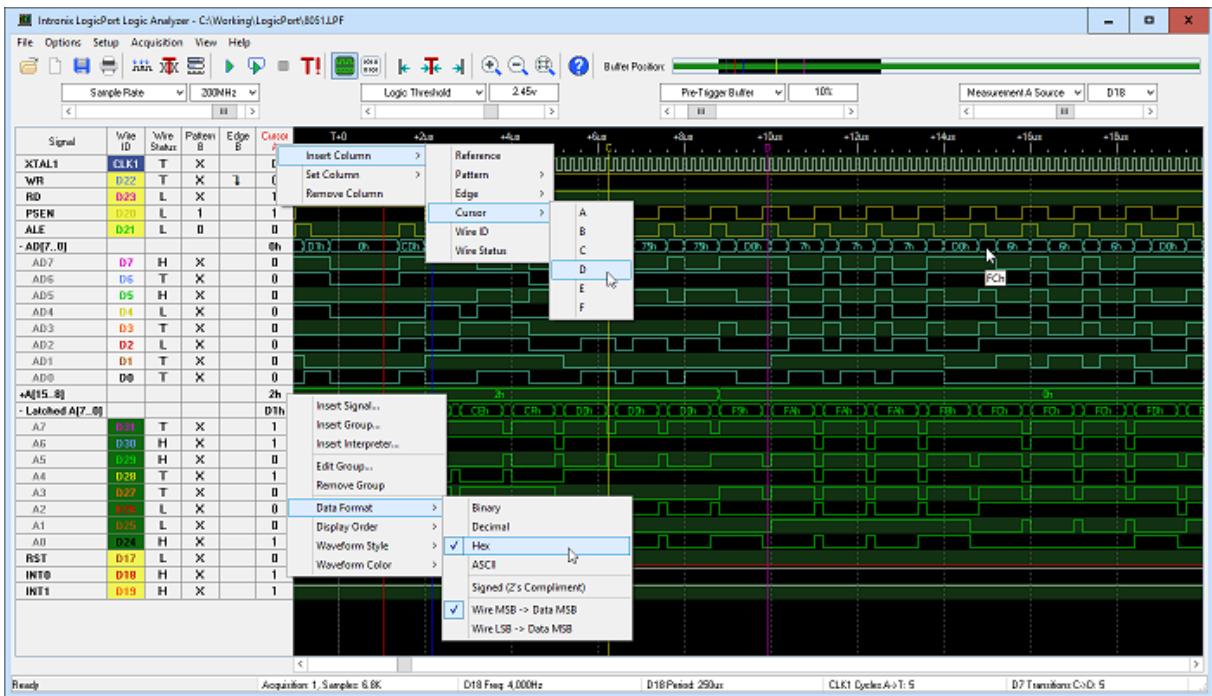


FIGURE 2.10 – Interface du logiciel, image tirée du site du constructeur

stables était erroné, et décodait correctement le signal "Hello World !" uniquement car tous les octets le composant avaient leur bit de poids fort à 0. L'espace fut remplacé par un autre octet ne présentant pas cette caractéristique, afin de rendre les tests ultérieurs plus robustes. Les réglages finaux sont présentés figure 2.11.

## 2.4.2 Programmation sur Game Boy

Il existe deux alternatives pour programmer sur Game Boy : soit programmer en assembleur pur (comprendre : quasi directement en langage machine), soit programmer en

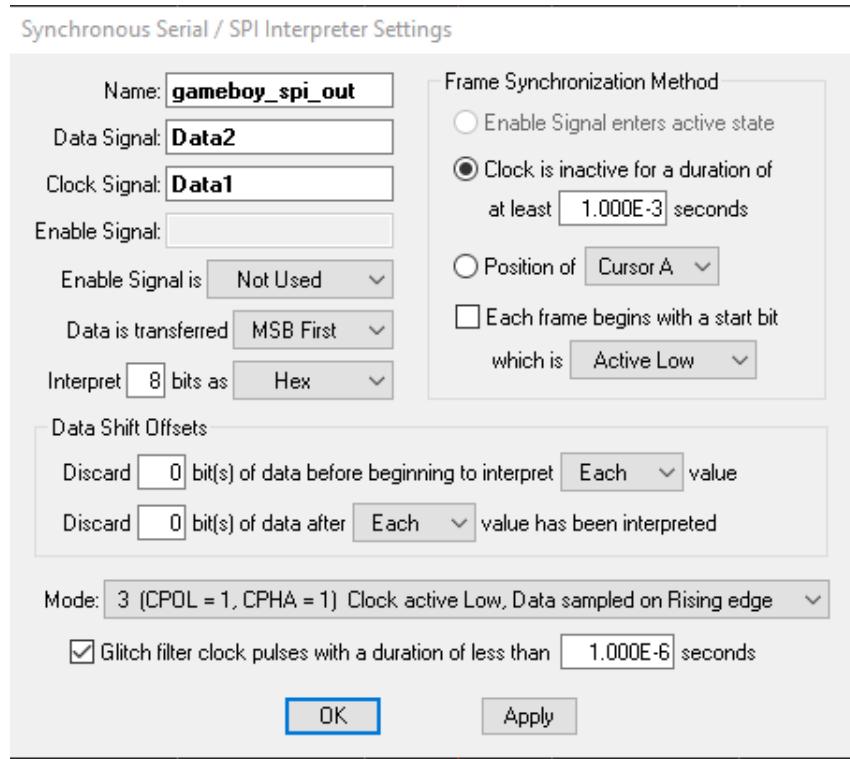


FIGURE 2.11 – Interpréteur logique pour le SPI du Game Boy

C. Programmer en C offrait l'avantage d'une écriture de code plus rapide et se préoccupant moins de détails techniques ; il s'est cependant avéré que la seule bibliothèque pour Game Boy est très vieille (dernière mise à jour en 2001), et le compilateur livré avec génère un code assembleur très difficile à debugger (aucun outil ne permet de debugger le code C directement, l'assembleur est donc un passage obligatoire de toute façon) voire parfois incorrect. De plus, la bibliothèque ne fournissait aucun support spécial pour l'accès au port Link, sinon un accès brut aux registres mémoire. Le code que nous avons dû écrire était de toute façon simple, et la documentation bien adaptée à la programmation assembleur, ce choix ne nous aura donc au final pas posé problème.

#### 2.4.3 Réception et Réponse au Message SPI

La majorité du travail a été réalisé sur une carte Nucleo F303K8 en attendant la carte qui avait été commandée. Cependant, le code reste relativement portable, car les deux Nucleos possèdent un bus SPI (marqué sur la figure 2.12) et Mbed fournit une interface commune quelle que soit la carte.

#### Bibliothèque SPI

La bibliothèque SPI fournie par Mbed propose deux objets : `SPI` et `SPISlave`. Comme évoqué plus haut, une communication SPI implique un maître, qui rythme la communication, et un esclave (unique, dans notre cas), qui suit le rythme. Nous avons décidé de

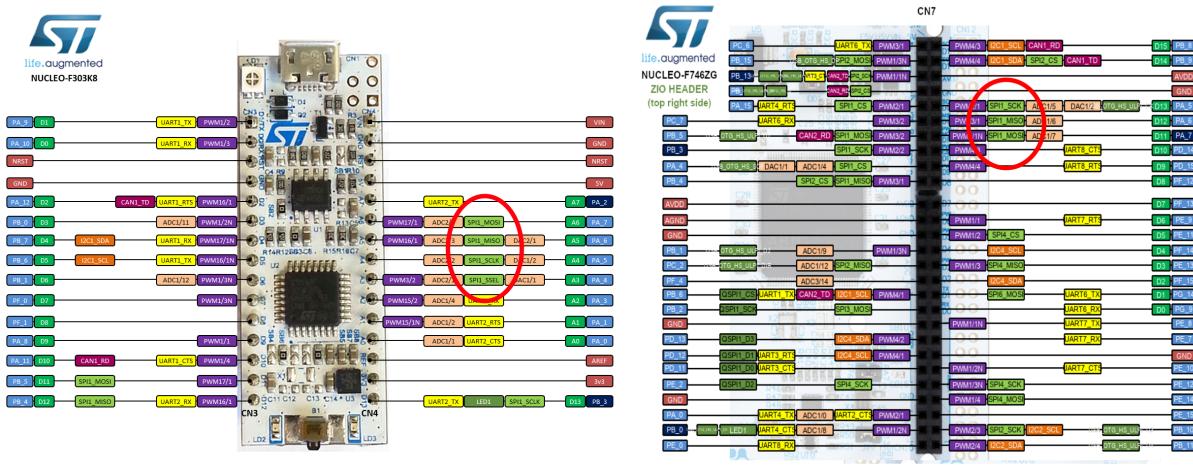


FIGURE 2.12 – Bus SPI des cartes Nucleo F303K8 et F746ZG

placer le Game Boy en maître, car la Nucleo est bien plus rapide, et devrait donc être capable de suivre tout rythme imposé par la console, l'inverse étant bien moins sûr et requérant des efforts supplémentaires de tout programmeur Game Boy utilisant l'accessoire. Nous reviendrons sur ce point dans la partie sur la conception du protocole GB/Nucleo.

SPISlave est une classe permettant de créer un objet correspondant à l'un des bus SPI de la carte, et de placer celle ci en esclave de l'appareil à l'autre bout du bus. Un objet SPISlave est créé avec 4 paramètres qui sont les 4 broches liées au 4 fils du protocole SPI. On peut ensuite utiliser la méthode **format** qui permet de préciser le mode SPI dans lequel on travaille (dans notre cas 3) ainsi que le nombre de bits dans chaque trame (8 ici). Pour la communication nous utilisons les méthodes suivantes :

**Receive** Indique si on a reçu un message.

Read Donne le dernier octet reçu.

**Reply** Ajoute l'octet passé en paramètre dans la file des prochains octets à envoyer.

## Réception du message

Un premier essai n'a donné aucun résultat. Deux problèmes furent identifiés :

- Chaque octet reçu était affiché sur un port série à des fins de debug, mais cela prenait trop de temps et faisait rater la réception de certains octets.
  - Comme mentionné plus haut, le protocole du Game Boy ne dispose pas de fil de sélection d'esclave, puisqu'il n'y en a qu'un seul ; la broche prévue à cet effet sur la Nucleo était branchée dans le vide, et la carte ne savait donc pas quand elle était censé écouter. La documentation de Mbed indiquait qu'il était possible de ne *pas* affecter certaines des 4 broches paramètres, ce que nous avons fait avec celle-ci.

Ces deux problèmes réglés, le "Hello World!" précédemment reçu sur le Logic Analyser était bien reçu par la Nucleo. Nous avons ensuite créé un objet représentant un message, qui à ce moment n'était pas bien plus qu'une chaîne de caractères, mais serait modifiable par la suite au fur et à mesure de la conception des protocoles de communication.

## Émission d'un message

La Nucleo étant esclave du Game Boy, il est nécessaire d'utiliser la fonction `Reply`. Alors il est important de prendre en compte le délai impliqué par cela. Pour montrer ce fait, nous avons réalisé un simple programme permettant d'émettre un byte et de recevoir celui émis par la Nucleo en même temps. La Nucleo quand à elle avait un simple programme renvoyant le byte reçu. On remarque donc que si l'on envoie une instruction, (ici un byte) la réponse à l'envoi d'instruction suivant.

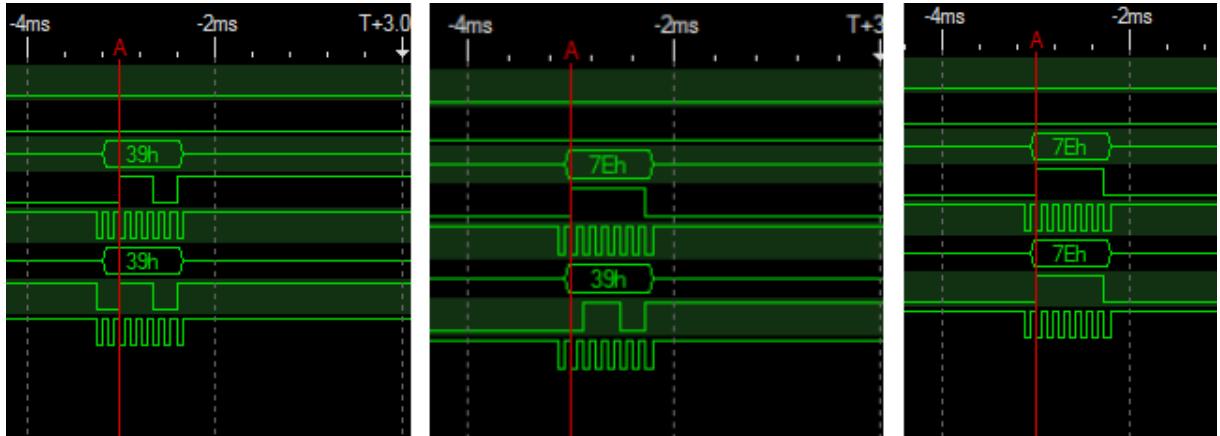


FIGURE 2.13 – État de base, Envoi de 7E, nouvel envoi de 7E et réception de 7E

L'expérience suivante n'a pu être répétée que peu de fois, en effet, peu de temps après les résultats ont commencé à devenir incohérents, comme sur la figure 2.14, ce que nous supposons être un défaut électronique que nous n'avons pas réussi à expliquer.

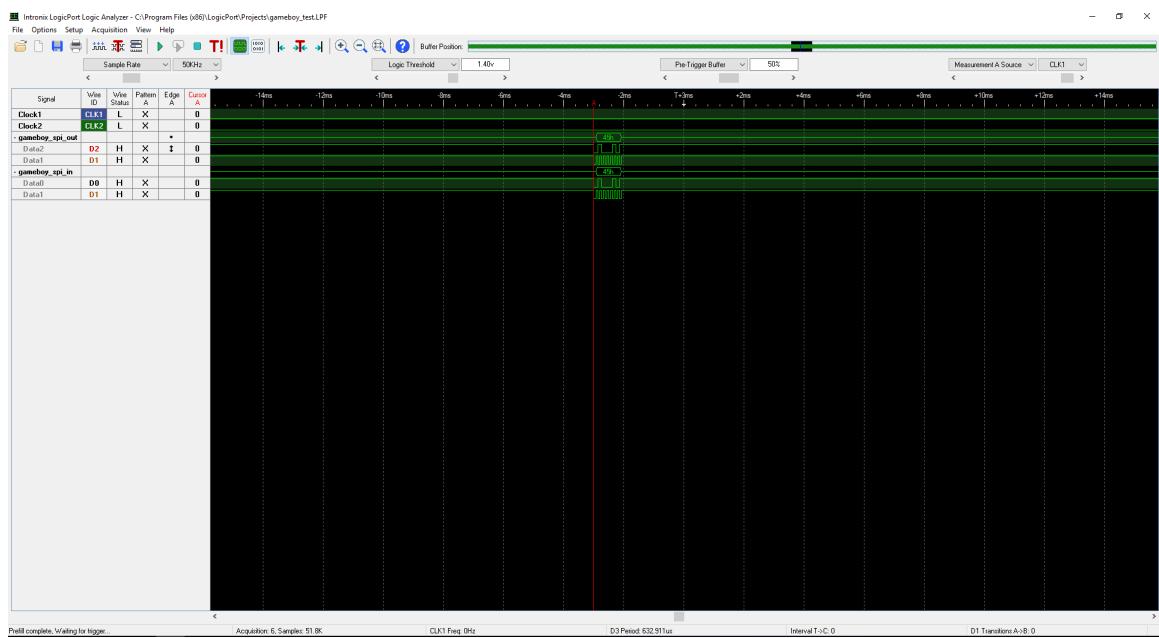


FIGURE 2.14 – Valeurs émises et reçues après défaut

Les valeurs obtenues avaient beau être incohérentes, elles étaient tout de même constantes,

un octet envoyé renvoyait toujours le même octet (bien qu'incohérent).

#### 2.4.4 Protocole de communication

##### Entre Nucleos

Le protocole de communication inter-Nucleo est simple : une Nucleo peut recevoir l'ordre d'émettre un message de découverte, ce qui envoie une trame Ethernet en broadcast. Toutes les Nucleo qui reçoivent ce message et sont "disponibles" répondent (après un petit délai aléatoire afin d'éviter les collisions). Cette dernière enregistre toutes les adresses source de chaque réponse.

La transmission des données requiert l'envoi simultané à plusieurs adresses destination. Plutôt que d'émettre une trame par destinataire, en séquence, ou d'utiliser l'adresse de broadcast, la solution la plus appropriée est d'utiliser le *multicast*. Une des Nucleo porte le rôle d'hôte, l'adresse multicast utilisée pour le groupe étant dérivée de l'adresse de l'hôte. En outre, l'hôte est responsable du maintien de l'état du groupe. Mis à part cela, les rôles des différentes Nucleo est symétrique.

Le maintien de l'état du groupe passe par une trame de ping. Pour des raisons évidentes de sécurité, une de ces trames qui n'aurait pas été émise par l'hôte (ce qui se vérifie très facilement avec les adresses source et destination) est ignorée. L'hôte doit envoyer périodiquement (une fois par seconde, peut-être plus ?) une trame de ping, à laquelle toutes les Nucleo doivent répondre par une même trame de ping, envoyée elle exclusivement à l'hôte. Le délai maximal de réponse est l'intervalle de ping. La trame de ping contient également la liste des Nucleo faisant partie de groupe, ce qui permet de s'assurer de la connexion de tous les participants. Si l'hôte n'a pas envoyé de trame de ping au bout de deux intervalles, l'hôte doit être considéré comme déconnecté.

Un point majeur de ce système est que le retrait de l'hôte d'un groupe doit causer sa dissolution, puisque l'adresse multicast utilisée est liée à la Nucleo. Une possibilité serait de désigner une autre Nucleo comme le nouvel hôte, mais le basculement pourrait poser des problèmes, particulièrement si une seule Nucleo ne reçoit pas la trame de ping, et tente de faire basculer le groupe. Aucune solution plus satisfaisante que la plus simple ne nous est venue à l'esprit.

##### Game Boy – Nucleo

Le protocole implémenté par le Game Boy n'offre aucune garantie : pas de redondance, pas de somme de contrôle ou de bit de parité. Cela tend à être un problème selon les câbles Link utilisés, qui sont souvent victimes de faux contacts. Afin de fournir un service aussi stable que possible, nous avons décidé de rendre le protocole aussi résilient que possible.

Tout d'abord, tout le protocole est contrôlé par le Game Boy. Tous les transferts d'octets sont contrôlés par la console afin de lui laisser le temps de les traiter, et aussi

car le Game Boy est peu souple quant à la transmission des bits, et réagit mal si un quelconque bit est perdu en mode esclave.

Le premier octet envoyé à chaque fois est un "opcode", ou plus simplement un numéro d'opération. Le bit de poids fort de l'opcode doit être à 0 ; la raison sera donnée juste après. Ensuite, un deuxième octet est transmis, qui sert d'opérande. Son bit de poids fort doit être à 1, afin de catégoriser chaque octet envoyé soit comme opcode soit comme opérande, et ce dès le premier bit. La Nucleo doit répondre à ces deux octets avec la valeur 0x80 (128 en décimal). La raison est que le Game Boy reçoit des bits à 1 si le câble est déconnecté (14), ce qui garantit que tout bit perdu sera remarqué. Enfin, la réponse de la Nucleo est transmise : le Game Boy envoie l'octet 0x7F (un octet d'opcode réservé), et la Nucleo répond avec un octet dont le premier bit est à 0 ; le Game Boy retransmet l'octet reçu shifté à gauche de 1, et la Nucleo répond à chaque bit avec un 0 si le bit était correct, et 1 sinon. Le Game Boy s'attend à recevoir uniquement des zéros, faute de quoi le processus recommence à partir de la transmission de 0x7F.

Il faut bien sûr définir le comportement en cas d'erreur de transmission : si le Game Boy reçoit une valeur autre que 0x80 en réponse à l'opcode ou l'opérande, il retente la transmission, jusqu'à 40 fois (le nombre a été choisi arbitrairement). De son côté, la Nucleo vérifie que 8 bits ont bien été transmis de manière régulière ; autrement l'octet partiel est ignoré.

Il y a une complexité additionnelle de communication, qui est que la spécification ci-dessus requiert de réagir bit par bit plutôt qu'octet par octet ; il serait possible de la modifier pour conserver une interface proche du SPI, mais cela augmenterait la bande passante et rendrait les transferts plus lents.

#### 2.4.5 Communication Ethernet

Dû au temps passé à concevoir les deux protocoles, puis à débugger les problèmes électroniques évoqués plus haut, l'implémentation de la partie Ethernet n'a pas dépassé le stade de l'échange de trames Ethernet entre la Nucleo et un PC.

# Chapitre 3

## Conclusion

Dans un contexte de regain d'intérêt pour le retro-gaming et la programmation de jeux rétro maison, il est intéressant d'intégrer des services modernes, comme le jeu en réseau, dans ces nouveaux jeux. Notre projet consistait à créer de toutes pièces un accessoire permettant à un Game Boy de communiquer avec d'autres via un réseau Ethernet.

Nous avons entièrement conçu la partie théorique du projet, mais la réalisation n'a pas abouti dû à une combinaison de facteurs prévisibles et imprévisibles. La plus grosse lacune causée par ceci est que nous n'avons pas pu éprouver la plupart de ce que nous avons prototypé sur papier.

Ce que nous avons échoué à prévoir a été : le temps nécessaire à la conception théorique des protocoles, que nous pensions pouvoir créer au fur et à mesure chacun de notre côté, mais qui s'est révélé impliquer tous les domaines à la fois à cause de l'intercommunication des systèmes, qui ne pouvaient pas complètement être présentés comme des boîtes noires aux autres. Ce que nous ne pouvions pas prévoir était le délai avant l'obtention de la carte, et à quel point tout notre projet serait bloqué par des erreurs de communication entre la console et la Nucleo. Ceci dit, nous aurions dû anticiper le temps pris par le debug d'au moins un imprévu.

Il pourrait être intéressant de revoir le projet avec un connaisseur du domaine de l'électronique. S'inspirer des protocoles existants pourrait également être intéressant, même si nous n'avons pas trouvé de solution existante à nos problèmes de conception, nous avons été surpris car ils semblaient simples.

Ce projet nous aura été utile notamment car il nous a placé dans la peau de concepteurs de protocoles réseau, ce qui a été très intéressant car bien plus complexe qu'attendu : nous avons dû traiter bien plus de cas d'erreur que de cas valides ! Il nous a également permis de mettre en application nos connaissances des solutions existantes afin d'effectuer des choix optimaux pour les couches sous-jacentes.

Notre projet n'a malheureusement pas abouti, cependant nous estimons avoir laissé des traces et une documentation fonctionnelle plus importante que les projets que nous avons cité dans notre état de l'art, ce qui permettrait à d'autres personnes intéressées de finir l'implémentation, et d'améliorer les protocoles que nous avons conçus tout en prenant compte, principalement grâce à ce document, des motivations derrière chacune de ces décisions.

# **Chapitre 4**

## **Annexes**

Nous tenons à remercier l'ensemble du collectif GBDev pour l'effort qui a été mis dans l'écriture de leur documentation, et Chloé pour son aide sur certains diagrammes.

Notre projet n'est pas affilié avec Nintendo Co. Ltd. Nintendo et Game Boy sont des marques déposées de Nintendo Co. Ltd.

# Bibliographie

- [1] Near (anciennement Byuu), Kawa et al. (noms réels inconnus), *Programming the MSU1* 2<sup>ème</sup> révision, 25 mai 2012, <http://helmet.kafuka.org/msu1.htm>
- [2] Pepijn De Vos, *Internet of Pokémons*, 20 février 2015, [http://pepijn devos.nl/TCPoke/](http://pepijndevos.nl/TCPoke/)
- [3] Pepijn De Vos, *TCPoke demonstration*, 10 avril 2015, <https://www.youtube.com/watch?v=xdcQFUGgEvo>
- [4] IIMarckus (nom réel inconnu) et al., Code désassemblé de Pokémons Red, accédé le 24 janvier 2020, <https://github.com/pret/pokered/tree/6ba3765c5932996f5da6417ae703794ff10bb1cb>
- [5] Pepijn De Vos, Code de la partie Teensy du TCPoke, accédé le 24 janvier 2020, <https://github.com/pepijn devos/TCPoke/tree/6784c2954a98eb39871063e064f54277e2f09f22>
- [6] Chris Baker, *Handicapping the Switch's chances : Industry Analysts weigh in*, 17 janvier 2017, [https://www.gamasutra.com/view/news/289326/Handicapping\\_the\\_Switchs\\_chances\\_Industry\\_analysts\\_weigh\\_in.php](https://www.gamasutra.com/view/news/289326/Handicapping_the_Switchs_chances_Industry_analysts_weigh_in.php)
- [7] Tatsumi Kimishima, *Six Months Financial Results Briefing for Fiscal Year Ending March 2018*, 2018, page 8, [https://www.nintendo.co.jp/ir/pdf/2017/171031\\_2e.pdf](https://www.nintendo.co.jp/ir/pdf/2017/171031_2e.pdf)
- [8] IEEE 802.3, *IEEE 802.3-2005*, 2005
- [9] D. S. Baxter, *Edge of Emulation: Game Boy 4-player Adapter*, 30 décembre 2017, <https://shonumi.github.io/articles/art9.html>
- [10] Timothy Lamb, *Arduinoboy*, 11 juin 2017, <https://github.com/trash80/Arduinoboy>
- [11] Dhole (nom réel inconnu), *Sniffing Game Boy serial traffic with an STM32F4*, 14 février 2018, [https://dhole.github.io/post/gameboy\\_serial\\_1/](https://dhole.github.io/post/gameboy_serial_1/)
- [12] Dhole (nom réel inconnu), *Virtual Game Boy Printer with an STM32F4*, 23 février 2018, [https://dhole.github.io/post/gameboy\\_serial\\_2/](https://dhole.github.io/post/gameboy_serial_2/)
- [13] Pan of ATX (nom réel inconnu), Martin Korth et al., *Pan Docs*, 2 juin 2019, <https://gbdev.github.io/pandocs/>
- [14] Pan of ATX (nom réel inconnu), Martin Korth et al., *Pan Docs*, 2 juin 2019, <https://gbdev.github.io/pandocs/#serial-data-transfer>

- [15] ISO/IEC JTC 1, ISO/IEC 7498 "Information technology — Open Systems Interconnection — Basic Reference Model", Novembre 1994 [révisé en juin 1996]. Disponible sur : <https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>