INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

STAGE

Compte rendu de stage rédigé par
Vincent GIRAUD
Élève ingénieur de l'INSA Rennes
Spécialité EII
Année 2017

Réalisation d'une interface de communication embarquée pour prototypage de l'*USB* type *C*

Lieu du stage
STMicroelectronics

Tuteur de stage
Nicolas PERRIN

Correspondant pédagogique INSA Rennes
Karol DESNOS

Remerciements

Je tiens à remercier Nicolas PERRIN pour avoir encadré mon travail et m'avoir inclus dans le grand projet de développement de l'$USB\text{-}C$, avec ses collègues Guénaël CADIER, Dominique CHAILLOT et Yohann MARTINIAULT.

Ma gratitude va également à Christophe CADORET pour m'avoir proposé cette expérience et m'avoir permis d'intégrer son équipe.

Merci à Karol DESNOS pour son évaluation post-projet.

J'ai également une pensée pour l'ensemble des collaborateurs du site de $STMicroelectronics$ à Rennes Atalante, dont la sympathie et le sens de l'accueil sont exemplaires, rendant le travail si agréable.

Table des matières

Chapitre 1

Introduction

1.1 *STMicroelectronics*

STMicroelectronics est une multinationale née de la fusion de la société française *Thomson Semiconducteurs* avec la société italienne *Società Generale Semiconduttori*, en 1987 [1]. La firme produit entre autres des puces électroniques, on notera particulièrement son développement dans les microcontrôleurs *ARM* et la mobilité, activités prédominantes dans l'implantation rennaise.

L'équipe rennaise de *STMicroelectronics* est composée de trois pôles : un concentré sur les microcontrôleurs, un sur la sécurité, et enfin un sur la vente et l'aide aux clients. Tout le monde travaille dans le même bâtiment, même si les divisions ne sont pas mélangées.

STMicroelectronics est une multinationale, il est commun de devoir collaborer avec des équipes à l'étranger (Europe, États-Unis...) via téléphonie voir visioconférence. L'anglais est utilisé par défaut. Cependant, il faut noter qu'il s'agit d'une entreprise aux racines françaises et italiennes. Ainsi, on peut citer de nombreuses implantations importantes en France ; comme Grenoble ou Rousset pour la production, Sophia Antipolis pour la recherche et développement, ou Saint-Genis pour la logistique.

Les microcontrôleurs *STM32* de l'entreprise, basés sur les processeurs *ARM*, se répartissent en trois gammes : *H* pour les hautes performances, *L* pour la faible consommation, et *F* pour les modèles plus réguliers [2]. La recherche et développement évolue essentiellement autour de ces trois catégories mais n'en est pas limitée, notamment à des fins d'innovation. Au sein de l'équipe, la division du travail tend à respecter la spécialité de chaque employé : UART, analogique, RCC...

STMicroelectronics est cotée au *CAC40*. 2 milliards de microcontrôleurs *STM32* ont été vendus depuis 2007 [3]. Depuis 2010, l'entreprise propose des plateformes de développement similaires aux *Arduino*, appelées *Nucleo*. Celles-ci se sont vendues à hauteur d'un million [3]. En fin de l'année 2016, 58% des ventes de *STM32* étaient destinées au marché asiatique [3].

1.2 Contexte

1.2.1 L'*Universal Serial Bus* et le connecteur type *C*

En 1995-1996, *Compaq*, *DEC*, *IBM*, *Intel*, *Microsoft*, *NEC* et *Northern Telecom* s'allient pour mettre au point une nouvelle norme, nommée *Universal Serial Bus*, ou *USB*. Celle-ci a pour but de définir un bus de communication en série associé avec son port ; afin de standardiser les communications entre périphériques et ordinateur. Cette démarche était cruciale ; car au milieu des années 90, les ports propriétaires se multiplient sur les cartes-mères, et tous ne sont pas performants et économiques. C'est seulement à la version suivante (1.1) que l'*USB* se verra exploité auprès du grand

1. http://www.st.com/content/st_com/en/about/st_company_information/who-we-are.html
2. http://www.st.com/en/microcontrollers/stm32-32-bit-arm-cortex-mcus.html
3. Page 29 : http://www.st.com/content/ccc/resource/corporate/financial/quarterly_report/group0/8b/57/65/59/08/d7/48/fc/ST_Sustainability_Report_2017/files/ST_Sustainability_Report_2017.pdf/_jcr_content/translations/en.ST_Sustainability_Report_2017.pdf

public. *Microsoft* fournira le support natif de l'*USB* via une mise à jour de *Windows 95* en août 1997 [4]. *GNU/Linux* fera de même à partir du kernel 2.2, en 1999 [5].

Le protocole est basé sur la relation hôte-périphérique(s) ; et les périphériques ne peuvent contacter l'hôte que lorsque ce dernier le leur autorise. Ainsi, les interruptions au sens strict ne sont pas possibles ; l'*USB* étant plus orienté vers l'attente active (*Polling*). Elles sont cependant émulées par une fréquence d'interrogation relativement importante. Le chaînage, fonctionnalité populaire de l'interface *Firewire*, n'est pas possible ici. Les périphériques peuvent être alimentés par le port, ou via leur propre alimentation externe.

Au fur et à mesure des versions, le débit de transfert et les possibilités en terme d'alimentation ont évolué.

Version	**Première année de sortie**	**Débit maximal théorique**	**Possibilités d'alimentation électrique**
1.x	1998	12 Mbps	5 V 100 mA max en basse consommation 500 mA max en haute consommation
2.x	2000	480 Mbps	5 V 100 mA max en basse consommation 500 mA max en haute consommation
3.0	2008	5 Gbps	5 V 150 mA max en basse consommation 900 mA max en haute consommation
3.1	2013	10 Gbps	Tension négociée dynamiquement (20 V max) Courant négocié dynamiquement (5 A max)

Aujourd'hui, la deuxième version est encore très répandue. Certains appareils ne requièrent pas plus que ce qu'elle offre. On peut citer notamment les périphériques d'entrée, comme les manettes de jeu. Elles consomment environ 150 mW (30 mA), et ne demandent pas plus qu'un Mbps en terme de débit. Cependant, avec le développement des systèmes embarqués grand public (téléphones intelligents, tablettes...), et l'augmentation de la capacité moyenne des périphériques de stockage externe ; d'autres appareils ont vu leurs besoins grossir, à la fois en terme de puissance électrique mais aussi en terme de débit.

La révision 3.1 a apporté une réponse conséquente à ces deux demandes ; en doublant le débit théorique par rapport à la sous-version précédente, et en proposant une nouvelle solution de négociation dynamique de la tension et du courant. Cette dernière fonctionnalité ne peut fonctionner sur un port conventionnel (type *A* ou *B*), car elle nécessite entre autres des broches additionnelles pour dialoguer pour établir le contexte électrique. C'est pourquoi le consortium *USB* a dévoilé, en supplément de la révision 3.1, un nouveau port destiné, sur le long terme, à remplacer tout les autres : le type *C*.

Malgré la volonté de simplifier tout l'écosystème *USB* derrière un port unique ; cette version supplémentaire a complexifié l'offre. La deuxième version de l'*USB* étant toujours présente (car simple et peu chère à déployer), un certains nombre de ports cohabitent (type *A/B/micro B* 2, type *A/B/micro B* 3, type *C*...), et les versions 2/3.0/3.1 se côtoient, auxquelles il faut rajouter les spécifications facultatives comme l'*USB On-The-Go*, le *Power Delivery*... Le groupe *USB* s'est retrouvé contraint de rappeler le dialecte inhérent à chaque version, suite à la confusion non seulement du public mais aussi des marques [6] [7].

4. https://support.microsoft.com/en-us/help/158238/how-to-determine-the-version-of-windows-95-98-me-in-use

5. https://www.kernel.org/doc/html/latest/driver-api/usb/usb.html#introduction-to-usb-on-linux

6. http://www.usb.org/developers/ssusb/USB_3_1_Language_Product_and_Packaging_Guidelines_FINAL.pdf

7. http://www.usb.org/developers/usbtypec/USB_Type-C_Language_Product_and_Packaging_Guidelines_FINAL.pdf

Il n'en tient qu'aux industriels d'exploiter et de promouvoir le type *C*, ce qui mènerait à une succession complète et à un retour du standard *USB* au sens strict.

1.2.2 Intérêts pour *STMicroelectronics*

Pourquoi *STMicroelectronics* voudrait s'investir sur le marché de l'*USB-C* ? Comme lors des débuts du *Bluetooth*, il y a la volonté d'être présent dans un domaine prometteur à long terme. Anticiper les nouvelles technologies permet de s'approprier une place confortable sur le marché, ainsi qu'un savoir-faire avantageux dans sa Recherche & Développement. L'électronique et les microcontrôleurs étant souvent sujets aux innovations, il est primordial pour *STMicroelectronics* d'avoir une vision sur le futur.

Ainsi, l'intérêt financier est énorme. L'*USB* est l'interface la plus répandue sur ordinateur. L'*USB-IF* annonce quasiment 3 milliards de périphériques *USB* 3.0+ livrés en 2018[8]. Le type *C* est sur la bonne voie pour devenir la nouvelle norme sur téléphones intelligents. En effet, sous l'impulsion de la commission européenne, un mémorandum d'entente avait été signé par la plupart des industriels en 2009 pour harmoniser les chargeurs de téléphones autour de l'*USB micro B* 2, à partir de 2011. Ce même texte a été renouvelé en 2013 et 2014[9]. Cette initiative devrait désormais être légalement officialisée, et un basculement vers l'*USB-C* est très probable. L'*USB-C* représente beaucoup de contrôleurs à vendre car en plus de ceux dans les périphériques, les câbles eux aussi doivent en avoir dans leurs fiches pour négocier le courant maximal admissible, par exemple.

L'*USB* n'est pas nouveau pour *STMicroelectronics*. L'entreprise a son représentant au conseil d'administration du consortium : Joel Huloux[10]. Il contribue à l'orientation de la norme, au côté des représentants d'*Apple*, *HP Inc.*, *Intel Corporation*, *Microsoft Corporation*, et *Renesas Electronics*. Selon lui, l'expérience de longue date de *STMicroelectronics* leur a appris que l'authentification sécurisée, ainsi que la bonne validation et protection du type *C* joue un rôle majeur dans son adoption par le public[11].

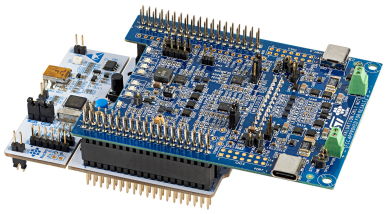
1.3 Le stage

Dans le cadre du contrat pédagogique établi dans le département *Électronique et Informatique Industrielle* de l'*INSA Rennes*, au moins un stage est imposé aux élèves, durant l'été ; afin d'inculquer une expérience professionnelle additionnelle. J'ai profité d'une occasion proposée par Christophe CADORET d'intégrer son équipe, dans un projet conséquent qui est développé depuis des mois avant mon arrivé dans le groupe : les contrôleurs *USB-C*.

STMicroelectronics veut commercialiser des microcontrôleurs et des plateformes de développement autour de l'*USB-C*. Le matériel est d'ores et déjà disponible[12]. Il s'agit désormais d'étoffer l'offre logicielle.

8. http://www.usb.org/press/presskit/USB-IF_IDF_Shenzhen_2014_Press_Deck_Final.pdf
9. http://ec.europa.eu/growth/sectors/electrical-engineering/red-directive/common-charger_fr
10. http://www.usb.org/about
11. http://www.businesswire.com/news/home/20160412005983/en/USB-3.0-Promoter-Group-Defines-Authentication-Protocol#.Vw-GRS_8NwA.linkedin
12. http://www.st.com/content/st_com/en/products/interfaces-and-transceivers/usb/usb-type-c-and-power-delivery.html?icmp=tt5409_gl_bn_jun2017&querycriteria=productId=SC2071

Pour une meilleure mise en contexte, on va explorer un exemple pratique. Un client de *STMicroelectronics*, comme *Apple* ou *Nintendo*[13], souhaite commercialiser un produit exploitant l'*USB-C*. Dans un premier temps, afin de développer un prototype, il s'orientera plutôt vers une plateforme comme le *P-NUCLEO-USB002*.



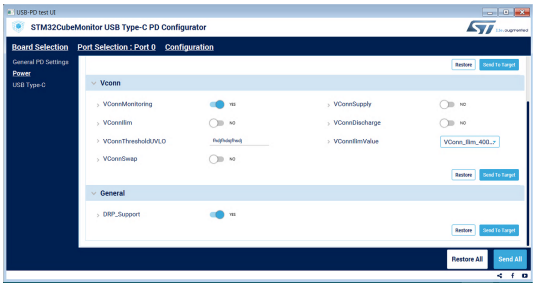
Il s'agit d'un *Nucleo* : un microcontrôleur *STM32F072RB* soudé à un PCB (blanc) ; et associé à un *ST-LINK/V2*, le programmateur phare de l'entreprise. Quelque chose de similaire à un *Arduino*, en somme. Sur cette plaque-là, on va brancher par dessus un autre PCB (bleu foncé), qui lui contient deux *STUSB1602*, chacun associé à son propre port *USB-C*, qu'il contrôle. Au final, on obtient donc un microcontrôleur *STM32* qui supervise deux ports de type *C*.

Le client va donc jouer avec les paramètres des ports *USB-C*, à l'aide d'une bibliothèque logicielle fournie par *STMicroelectronics*, développée entre autres à Rennes. Les deux ports, bien que sur la même plaque ; sont indépendants. On peut assigner à chacun d'eux une configuration propre, les relier avec un câble *USB-C* et constater leur comportement à l'aide des DEL embarquées.

Par «configuration», on parle du statut logiciel (hôte ou esclave), du rôle électrique (source ou consommateur), des capacités maximales en terme de tension et courant...

Une fois que le client a défini la configuration qui lui convient le mieux, et qu'il l'a testé dans les cas susceptibles d'apparaître, il peut passer à la conception finale. Il aura alors seulement besoin d'un *STUSB1602* par port *USB-C* sur son produit. Cette puce fait 16 mm^2 (4×4), et reçoit sa configuration via I^2C[14]. La production peut alors être lancée.

Il y a un problème dans cette chaîne. Lors du prototypage sur *P-NUCLEO-USB002*, le changement de configuration à la volée est compliqué, et le retour visuel via les DEL peut se révéler insuffisant. Les ingénieurs de *STMicroelectronics* ont alors eu une idée : mettre en place un logiciel sur ordinateur, doté d'une interface graphique ergonomique, qui permettrait d'interagir avec un *P-NUCLEO-USB002* via *USB 2* (on utilise le port mini *B* sur le *Nucleo*, pas un des type *C* que l'on veut exploiter). On ouvre ainsi la porte à plusieurs possibilités. On peut, sur ordinateur ; récupérer des informations générales ou spécifiques à chaque port *USB-C*, appliquer une configuration sur un port, suivre le processus de négociation...



Une telle fonctionnalité requiert de développer une partie sur ordinateur, et une partie embarquée sur *STM32*. La partie sur ordinateur (Interface graphique, communication via *USB*...) est prise en charge par des ingénieurs au centre de Recherche & Développement du Mans. La partie embarquée (communication via *USB*, interaction avec l'environnement logiciel mis en place...) est développée à Rennes, et m'a été confiée.

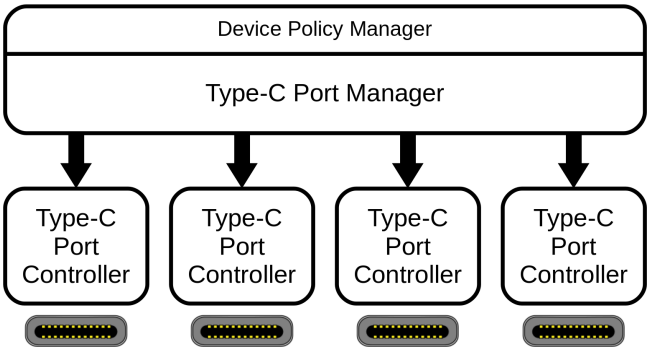
13. `http://www.lavoixdunord.fr/167650/article/2017-05-24/stmicroelectronics-s-investit-dans-les-nouveaux-objets-du-quotidien`
14. `http://www.st.com/resource/en/datasheet/stusb1602.pdf`

Chapitre 2

Exigences

2.1 Spécifications

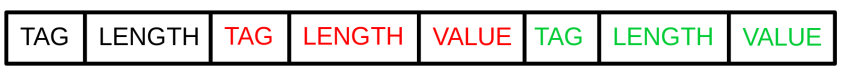
L'objectif du stage n'est pas de réaliser un microcode dans sa globalité. En réalité, il s'inscrit dans le développement de l'*USB Power Delivery*, projet mené depuis des mois par de nombreux ingénieurs chez *STMicroelectronics*. Ma partie n'est qu'une brique qui s'insère dans l'écosystème logiciel.



Ce schéma très simplifié (il faudrait rajouter un nombre conséquent de sous-parties logicielles pour être plus fidèle) représente l'architecture globale du microcode. La partie TCPM se situe sur le microcontrôleur *STM32*. Elle supervise un ou plusieurs TCPC, situés sur *STUSB1602*. Chacun a son propre port *USB-C* associé. Le Device Policy Manager est la porte d'entrée logicielle pour l'utilisateur final.

On veut pouvoir dialoguer avec le système sur un ordinateur. La solution matérielle consiste en une liaison UART entre l'ordinateur et le microcontrôleur. La solution logicielle consiste en une sous-partie logicielle placée entre les pilotes de l'UART, et le DPM. Quand on reçoit des instructions, on récupère les données via l'UART et on fait le nécessaire auprès du DPM (qui a son tour peut influencer TCPM et TCPC).

Le protocole de communication imposé est de type TLV : Tag, Length, Value. Chaque instruction commence par le tag, qui indique quel type de donnée va suivre. Ensuite, la longueur indique le nombre d'octets que fait la trame. Enfin, il y a les données elles-mêmes. On peut également encapsuler les paquets, comme ceci :



Ici, la première balise length indique la taille de tout ce qui suit, c'est-à-dire la longueur des deux sous-trames complètes. C'est ce qui permet de connaître la fin de l'instruction, et de savoir si on lit un paquet ou sous-paquet. Les autres balises length indique seulement la taille de leur donnée respective.

Une documentation contenant un jeu d'instructions a été définie. Chacune d'entre elles n'a qu'un sens d'utilisation (ordinateur vers microcontrôleur ou inverse). Sous le nom d'«instruction», ou «paquet» ; on peut en réalité décrire une requête, confirmation de réception, de simple données...

Le microcontrôleur $STM32$ a une entrée et une sortie UART. Elles sont reliées au $ST\text{-}LINK/V2$, qui est capable de faire la conversion UART vers USB, et inversement. Ainsi, la liaison matérielle se fait avec un câble USB 2.

2.2 Outils utilisés et environnement

Un ordinateur est fourni aux stagiaires. Celui-ci fournit le même environnement logiciel qu'à n'importe quel employé, à savoir $Windows\ 7$ 64 bits.

On développe principalement sous l'IDE IAR, mais le logiciel libre $SW4STM32$ est aussi exploité parfois, car il est officiellement supporté par $STMicroelectronics$. En terme de versionnage, on utilise Git. SVN est également présent dans certaines implantations de l'entreprise, principalement par difficulté à effectuer la transition. Cette dualité pose problème mais ne devrait plus durer longtemps, en faveur de Git. Les projets sont centralisés sur la forge $Tuleap$. Comme le microcode complet est conséquent et est composé de beaucoup de modules, on utilise $Repo$. Cet outil peut être vaguement assimilé à un gestionnaire de versions travaillant au-dessus de Git. Il permet d'indexer et gérer tous les modules du logiciel, et de mettre au point une version globale. Le projet $Android$, par exemple, l'utilise : cela permet de versionner un système d'exploitation complet, en respectant l'indépendance du développement de chaque module.

Pour assurer la qualité du code, celui-ci est analysé pour convenir aux normes $MISRA\ C$. Avant de soumettre une version sur le dépôt, le travail passe d'abord sur la plateforme $Gerrit$, qui permet à d'autres employés de le relire. Chaque version subit des tests unitaires sur un serveur $Jenkins$, prévu par exemple ici pour piloter des systèmes de spécification $USB\text{-}C$.

Un peu de code en C++ a été réalisé sur un logiciel de réception en série sur ordinateur, afin de mieux comprendre les transmissions du système embarqué.

2.3 Quantification des contraintes

L'interface logicielle s'imbriquant dans un projet complet, on ne subit pas seulement les contraintes de l'embarqué ; mais aussi des ressources occupées par le reste du microcode.

Ainsi, en terme de mémoire vive ; il est imposé de ne pas dépasser 3 kilooctets. On ne fait pas recours à l'allocation dynamique : cela permet de mieux cartographier la mémoire, et de complètement maitriser sa gestion. En effet, dans le cadre d'une interface de communication, une utilisation malveillante peut être facile : on risque de pouvoir envoyer un message qui provoquera un débordement de mémoire.

Les différents modules du code devant cohabiter lors de l'exécution, on utilise un planificateur de tâches. Celui-ci joue le rôle du système d'exploitation, en gérant les différents fils d'exécution, et en choisissant auquel il va accorder le temps du processeur. $FreeRTOS$ est choisi ici, comme dans beaucoup de projets sur microcontrôleur $STM32$: c'est un ordonnanceur libre, supporté par $STMicroelectronics$ comme par beaucoup d'acteurs du domaine, et qui jouit d'une grande communauté. Ainsi, dans le cadre du projet $USB\text{-}C$, l'interface de communication ne doit pas monopoliser le processeur 4 millisecondes ou plus. Aussi, ce module doit s'exécuter correctement tout en ayant la priorité d'exécution la plus basse auprès de $FreeRTOS$ (niveau 1, sur une échelle croissante de 1 à 5).

Une attention particulière a été portée aux variables stockées dans la pile d'exécution ($stack$). En l'état actuel du projet, on peut avoir environ 6 tâches qui tournent en même temps sur la plateforme, qui a 16 kiloctets de mémoire vive. Ainsi, chacune doit se contenter de quelques centaines d'octets pour sa pile (pas plus de 500). On se gardera alors d'user de techniques comme la récursivité, hormis cas particuliers tels que la récursion terminale.

Chapitre 3

Réalisation

3.1 Mode de travail

3.1.1 Organisation

Comme dans beaucoup d'entreprises, les employés de *STMicroelectronics* cherchent continuellement à améliorer leur productivité. Ils font ainsi appel aux méthodes agiles : des pratiques et règles destinées à améliorer la gestion de projet. Chaque équipe peut expérimenter sa propre technique. Celle qui développe l'*USB-C* a choisi la méthode *Scrum*. Il s'agit essentiellement de subdiviser le projet en de nombreux *sprints* : un mini-projet, faisable en environ une semaine. Il peut s'agir d'un ajout de fonctionnalité, par exemple. À chaque fin de *sprint*, on se réunit et fait une revue générale, où chacun présente son avancée ; et éventuellement explique ce qui a posé problème si l'objectif n'est pas atteint. Cette revue doit durer le moins de temps possible (environ un quart d'heure dans notre projet), pour ne pas tomber dans le piège de la réunion interminable.

J'ai eu la chance en temps que stagiaire d'être convié aux réunions où ma présence n'était pas nécessaire. L'implantation de Rennes organise de temps en temps des rassemblements où les ingénieur(e)s de tous les projets se retrouvent pour aborder l'avancement général de l'entreprise, partager des savoirs sur des nouvelles technologies, analyser les résultats financiers... Certaines de ces réunions sont organisées en *stand-up* : tout le monde doit rester debout. L'inconfort généré est exploité volontairement afin d'empêcher l'événement de durer trop longtemps.

3.1.2 Considérations techniques

Comme c'est le cas pour beaucoup de stages, le travail réalisé a pour ambition d'être réutilisé par la suite. Il faut alors prendre certaines précautions.

Le code écrit est commenté. Chaque partie critique est expliquée, de même que celles où la méthode employée n'est pas explicite. Le but est de rendre une reprise du travail ultérieure la plus simple possible. On utilise la syntaxe de *Doxygen*, afin de pouvoir générer une documentation de code au format web.

Une documentation plus textuelle et plus générale est réalisée sur un logiciel de traitement de texte (ici, *Microsoft Office Word*). L'intérêt est plus de détailler le fonctionnement général et la structure. Il s'agit d'un point de départ pour toute personne souhaitant se plonger dans ce travail. Elle contient schémas et tableaux pour faciliter la compréhension, et est respectueuse de la charte graphique de l'entreprise.

Le versionnage du module logiciel se fait à partir de l'état du projet à mon arrivée. Il a été choisi de ne faire qu'une fusion en fin de stage. Ainsi, le développement s'est fait sur une soumission stable prise sur la forge, à mon arrivée. Le code restant relativement isolé, cela n'a pas causé de problème majeur.

3.1.3 Exploitation du stage

Un représentant de l'Association Pour l'Emploi des Cadres ($APEC$) s'est présenté à l'entreprise pour faire une présentation de deux heures aux alternants et stagiaires, dans le but d'expliquer comment mieux exploiter un tel contrat.

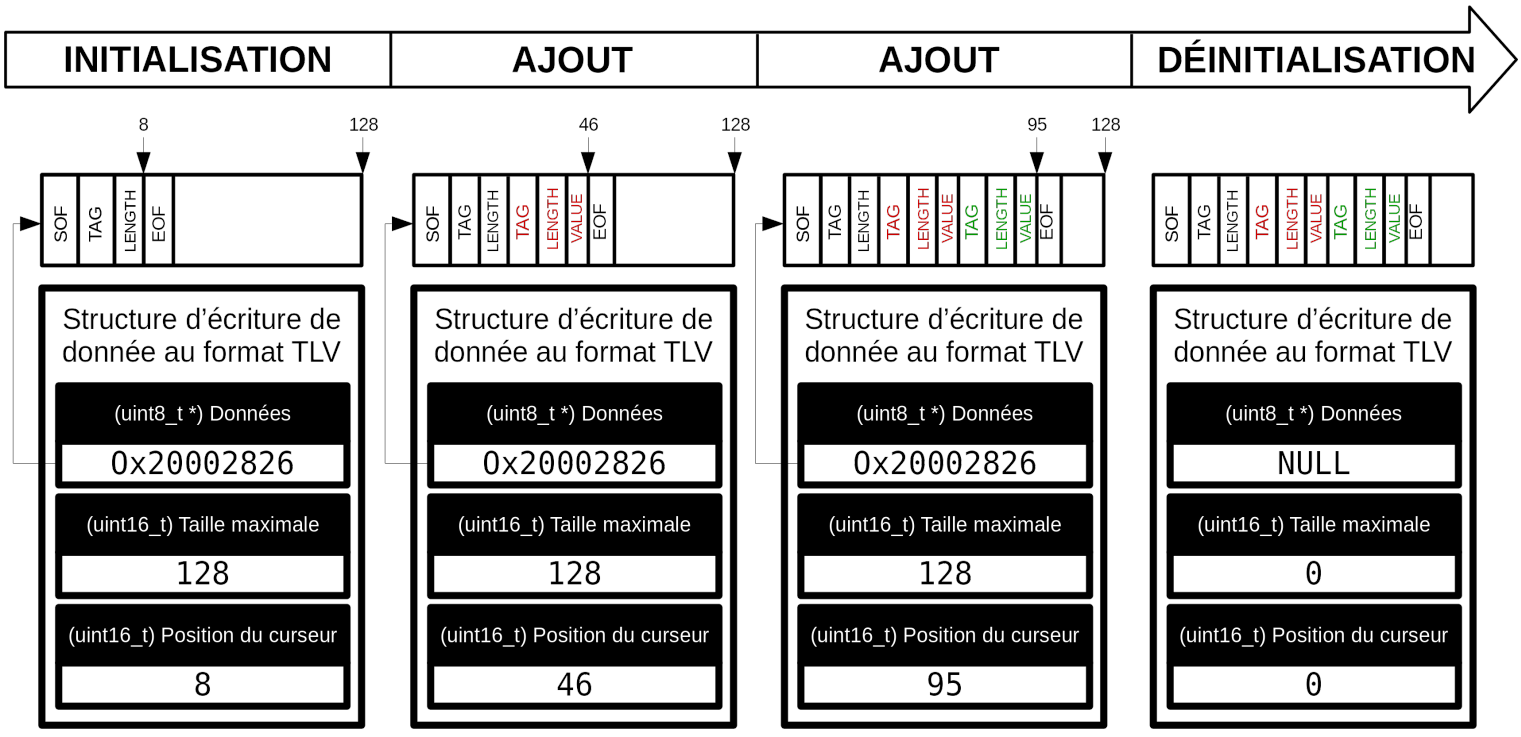
Nous avons pu en apprendre plus sur la manière avec laquelle les recruteurs cherchent leurs futurs collaborateurs. Il est important de bien choisir ses termes lorsqu'on restitue une expérience comme ce stage sur un curriculum-vitæ : cela doit être facilement retrouvable via un moteur de recherche, et assez explicite pour que le lecteur ait une idée claire.

Cela a été une occasion d'apprendre que la moitié des offres sont en Île-de-France, que la moitié des recrutements des moins de 30 ans passent par une offre d'emploi, et que le nombre d'opportunités est en hausse constante presque partout.

3.2 Exploitation du format TLV

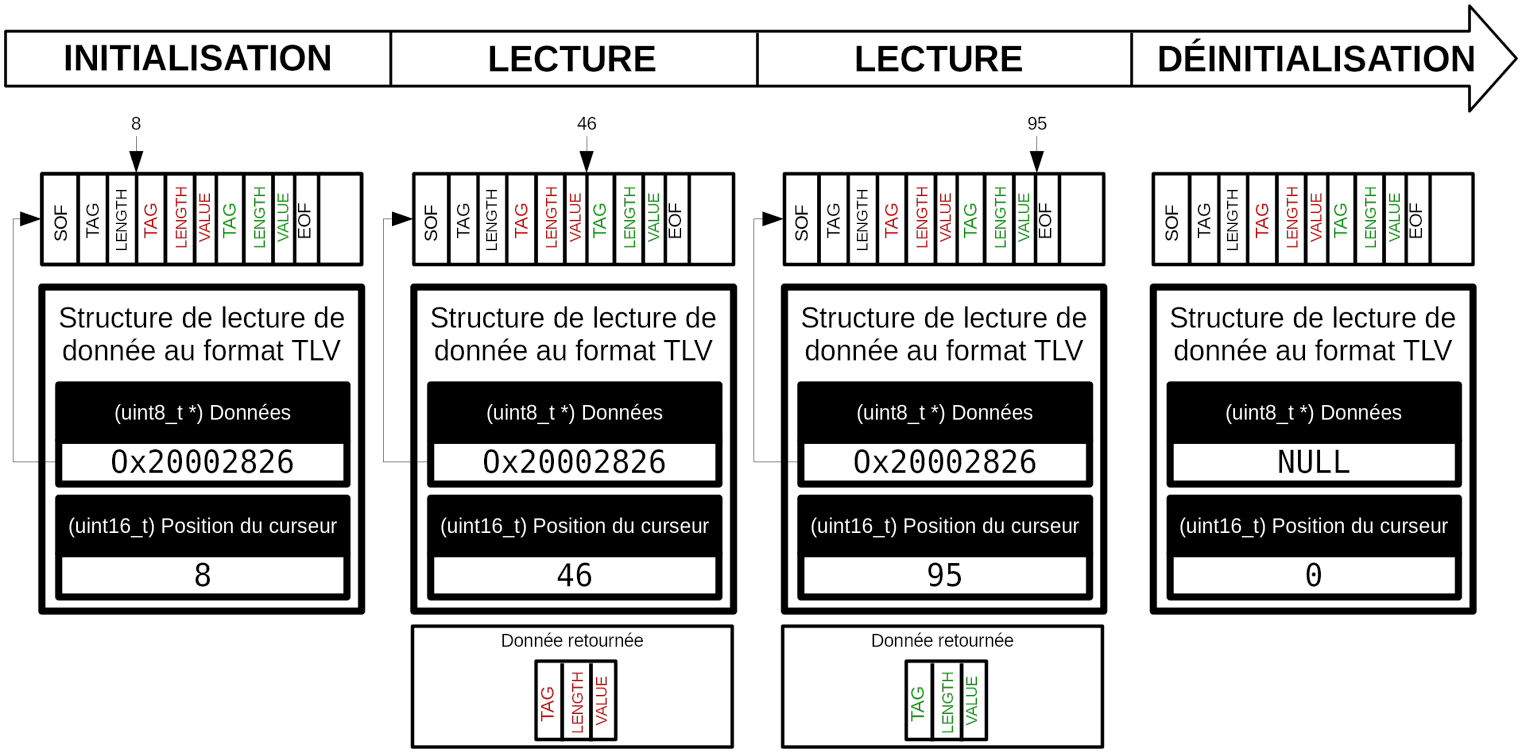
L'exploitation se fait dans deux sens : écriture et lecture d'une instruction au format TLV. Dans les deux cas, on a mis au point une structure qui contient un pointeur vers les données (à lire ou à écrire), ainsi que des informations supplémentaires comme la position du curseur de lecture/écriture, par exemple. À chaque fois que l'on veut interagir avec une structure, on passe son adresse en paramètre de la fonction appropriée. On n'aborde pas ici le sujet de l'envoi et de la réception d'une trame, mais seulement sa création ou son interprétation.

Dans le cas de l'écriture, on conçoit une structure contenant un pointeur vers les données au format `uint8_t`, un entier représentant la taille maximale de l'instruction, et un autre entier donnant la position de la fin de l'instruction telle qu'elle est à l'instant t. On utilise une fonction pour initialiser ces données, écrire l'entête du TLV hôte, et mettre le curseur au début de la valeur. À partir de là, on peut utiliser une autre fonction qui encapsule des données à l'intérieur, tant qu'on ne dépasse pas la limite de taille. À chaque fois qu'elle rajoute quelque chose, elle met à jour la balise Length souveraine. Une fois que l'on a fini, on désinitialise la structure, et la chaîne d'octets qui y était précédemment associée peut directement être envoyée à l'UART.



Dans le cas de la lecture, on a mis au point une autre structure contenant un pointeur vers les

données au format `uint8_t`, et un entier donnant la position du curseur de lecture à l'instant t. On utilise une fonction pour initialiser le pointeur, et mettre le curseur au début de la première valeur. À partir de là, on peut utiliser une autre fonction qui lit une sous-trame à chaque fois et met à jour le curseur, tant qu'on n'atteint pas la fin. Une fois que l'on a fini, on désinitialise la structure. Le traitement des données peut se faire avant cette étape finale, au fur et à mesure qu'on lit, notamment.



Au final, du point de vue des autres développeurs, on obtient une interface de programmation applicative capable d'exploiter un format de donnée. Le principe est similaire à celui de $Libxml2$ avec le format XML.

3.3 Gestion du temps réel

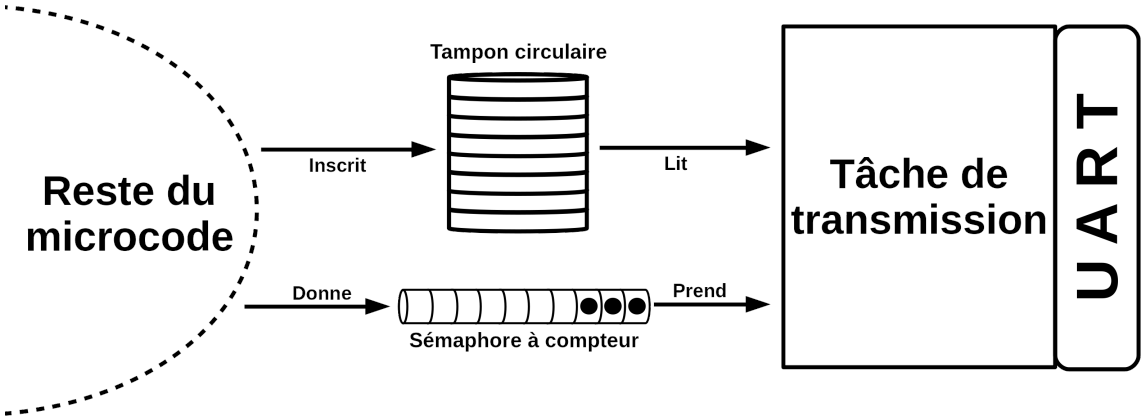
Comme vu dans l'étude des contraintes du cahier des charges, nous devons nous soumettre à des limites inhérentes à un fonctionnement en temps réel. La plateforme visée est équipée d'un processeur ARM mono-cœur ($Cortex\text{-}M0$). Plusieurs tâches devront s'exécuter et être ordonnancées en même temps dessus, grâce à $FreeRTOS$.

On conçoit deux tâches : une pour la transmission, l'autre pour la réception. La séparation est logique, et est compatible avec l'UART d'un point de vue matériel : celui-ci dispose d'une ligne physique pour chacun des sens, et maîtrise les deux parallèlement. Nos tâches doivent se contenter de la priorité d'exécution la plus basse : en cas de conflit avec une négociation de contrat USB par exemple, il est plus important de mener cette dernière à bien.

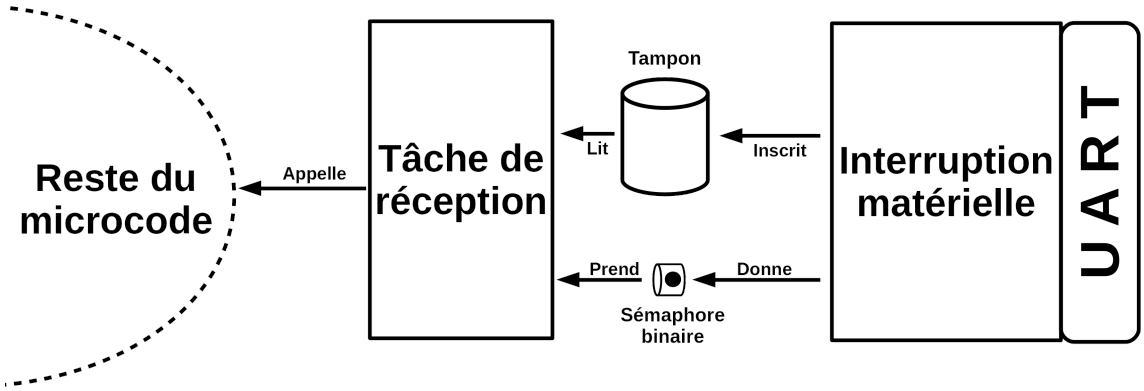
Dans les deux sens, on a recours à une mémoire tampon ; c'est-à-dire une section de la mémoire vive dédiée à stocker temporairement des données en cours de transit. Cette pratique est très utile pour temporiser un traitement dans un contexte multitâche. Pour la transmission on utilise un tampon circulaire : l'espace dédié va de l'adresse a à l'adresse b, et lorsque les données atteignent la fin en b, elles continuent en a. La fin rejoint (logiciellement) le début, comme un cercle. Cette jointure peut être délicate à gérer, mais est facilitée ici par le caractère sériel de la liaison UART. Pour la réception, on utilise un tampon simple : on a un espace dédié avec une taille définie, et si les données à stocker sont plus grandes, on doit gérer le dépassement (ici en ignorant l'instruction). Ce cas n'est pas sensé

se produire, puisque nous avons pris l'initiative de définir une taille maximale admissible pour les instructions.

Un sémaphore à compteur a été mis en place pour l'envoi d'instructions. Lorsque le reste du microcode (comprendre : toutes les autres tâches) souhaite envoyer un message, il inscrit celui-ci dans le tampon circulaire, et donne le sémaphore. Si d'autres tâches plus importantes doivent s'exécuter, le processus reste en suspens. Une fois que le processeur est libre, la tâche de transmission prend le sémaphore et prend le temps de s'exécuter. En effet ; comme la liaison fonctionne à une vitesse définie (en Bauds), l'envoi peut mettre un certain temps. Il se peut qu'avant que la tâche de transmission puisse être lancée, les demandes d'envois s'accumulent. C'est pour cette raison que le sémaphore est à compteur : à chaque demande, il est incrémentée. Il témoigne du nombre de trames à envoyer. S'il est nul, la tâche ne se lancera donc pas. Au fur et à mesure de l'envoi, on interprète les données pour savoir quand est-ce qu'on a fini d'envoyer une trame. Si c'est le cas, que le processeur est libre, et qu'il reste au moins un sémaphore, alors on relance la tâche. Un curseur de lecture statique permet de mémoriser notre position dans le tampon circulaire.



Pour la réception, nous avons convenu d'utiliser un tampon simple pour une seule instruction seulement. La jonction fin/début d'un tampon circulaire demande trop de ressources processeur ; et de toute manière, la partie sur ordinateur ne devrait pas envoyer plusieurs requêtes d'affilée sans avoir reçu écho de la précédente. On ne sait pas quand est-ce qu'on va recevoir une instruction, alors quand on en reçoit une on se contente de simplement l'écrire dans le tampon et de donner le sémaphore lorsqu'elle est finie, afin de ne pas prendre trop de temps. Ainsi le traitement, qui se fait dans la tâche de réception et qui peut demander une certaine quantité de ressources, peut être repoussé à quand le processeur sera moins sollicité : on attend le sémaphore. Le caractère binaire du sémaphore est dû au fait que le tampon ne peut recevoir qu'une seule instruction : il y en a une ou pas.



Chapitre 4

Résultats

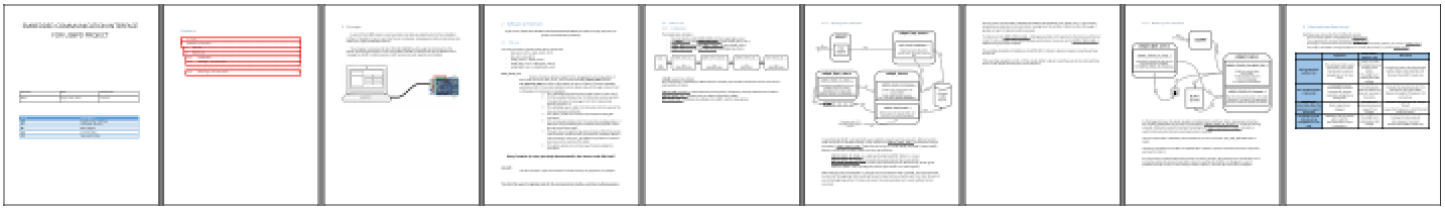
4.1 Satisfaction du cahier des charges

En fin de mission, on analyse le travail résultant pour voir s'il satisfait le cahier des charges.

Pour la consommation de mémoire vive, on consulte le fichier de carte de la mémoire (.map) généré par le compilateur propriétaire d'*IAR Systems* (*GCC*, comme la plupart des autres compilateurs, en génèrent un également à l'édition de liens). Celui-ci permet de savoir, pour chaque fichier de code, la quantité de mémoire vive utilisée par des variables stockées en zone *data* ; comme les variables globales par exemple. Pour le module de communication, il témoigne de moins de 1500 octets. En effet, le tampon circulaire de transmission occupe 512 octets, et le tampon simple de réception occupait la taille maximale d'une instruction, c'est-à-dire 256 octets en fin de stage. Les deux sémaphores s'y trouvent également, ainsi que quelques variables non-primordiales. À cela, il faut ajouter les variables allouées dans un autre espace, et donc non cartographiées sur le fichier de carte de la mémoire ; comme les variables locales par exemple. À l'exception d'un tableau de la taille maximale d'une instruction, il n'y a que des allocations minoritaires et négligeables. La limite étant de 3 kilooctets, on considère celle-ci respectée. On maîtrise les piles d'exécution en n'usant pas de récursivité. On ne fait pas d'allocation dynamique, pour les raisons explicitées dans le dévoilement des contraintes.

Il est également nécessaire de ne pas monopoliser le processeur plus de 4 millisecondes avec l'interface de communication. Cette contrainte peut être respectée également, la vérification s'est faite lors du test de résistance.

Sur un registre moins technique, le code a été documenté selon la syntaxe *Doxygen*. Les fonctions et structures, par exemple, ont leur en-tête explicatif. Le code est également commenté, notamment dans ses parties complexes et peu explicites. Une documentation a été rédigée à l'aide d'un logiciel de traitement de texte, pour assister tout futur développeur devant utiliser le module (ici volontairement illisible car protégée).



4.2 Test de résistance

4.2.1 Méthodologie

On veut tester la fiabilité de l'interface de communication, en situation très intense. Pour cela, on réalise un test de résistance, où on sursollicite le module dans des conditions extrêmes.

Pour cela on envoie à un intervalle régulier une requête au système cible. Celui-ci doit correctement la recevoir, l'interpréter, et réaliser le travail demandé : récupérer une vingtaine de données dynamiques

à insérer dans une instruction à créer correctement, puis l'envoyer. On considère le test réussi si toutes les étapes ont été réalisées avec succès, et si le système cible ne plante pas. On réalise le test plusieurs fois, en faisant varier la fréquence d'envoi de requête et la vitesse de transmission, pour obtenir des résultats plus développés.

Pour vérifier que l'on n'accapare pas le processeur plus de 4 millisecondes, on chronomètre à partir de l'instant où une requête a été reçue, jusqu'à celui où on a fini d'envoyer l'instruction contenant les données. Pour cela, on utilise le chronomètre *systick* présent sur tous les cœurs *ARM Cortex-M*. Une fois calibré par rapport à la fréquence de fonctionnement du système, il peut s'agir tout simplement d'un compteur qui s'incrémente chaque milliseconde. Ce système est utilisé dans le projet *USB Power Delivery* car le microcode est ordonnancé avec *FreeRTOS*, qui exploite *systick* à des fins de fonctionnement en temps réel.

4.2.2 Observations

On compile les résultats sur un tableur.

		10 Hz	25 Hz	50 Hz	75 Hz	Average time required to send an instruction (ms) (100 tests sample)
Baud	115 200	Passed	Passed	Passed	Passed	5,404
	230 400	Passed	Passed	Passed	Passed	2,702
	460 800	Passed	Passed	Passed	Passed	0,894

On constate en premier lieu que tous les tests sont réussis, et ceci jusqu'à 460 800 bauds et 75 requêtes par seconde. On en déduit une certaine robustesse du module, qui devrait parfaitement tenir en utilisation normale, bien moins demandeuse.

Cependant, pour la rapidité de traitement, il y a une certaine précaution à prendre. En effet, on voit que si la vitesse de transmission de l'UART est à 115 200 bauds, on occupe le processeur pendant 5,4 millisecondes ; ce qui est au-dessus des 4 millisecondes spécifiées en contrainte. Il en ressort que tout débit en dessous de 230 400 bauds n'est pas envisageable, car non-respectueux du cahier des charges.

Chapitre 5

Conclusion

À la conclusion du stage, l'intégralité du module a été réalisé, et préparé pour une utilisation par d'autres développeurs. Les réponses aux différentes requêtes de l'ordinateur ne sont pas toutes réelles : bien que certaines répondent avec les vraies données actualisées, beaucoup en envoient des fausses, simulées en dur. Les raisons pour cela sont que le projet *USB Power Delivery* est énorme et mobilise un grand nombre d'ingénieurs. Le code change encore beaucoup. Récupérer des données réelles, ou pire, appliquer un changement d'état au noyau *Power Delivery* à la demande impliquerait que j'interagisse voir modifie du code central. Or je ne peux, en 8 semaines, assimiler l'intégralité du projet ; et assurer la pérennité de mes modifications face aux nombreuses retouches qui auront lieu dans le futur. J'ai ainsi fait en sorte qu'une fois le projet conclu, il ne reste que des «trous» à combler dans mon module pour activer des interactions complètes avec le noyau. Un portage de mon module était éventuellement envisagé vers une nouvelle gamme *STM32* en développement, mais ce projet a été abandonné par manque de temps.

Ce module sera utile auprès des clients de *STMicroelectronics*, afin qu'ils puissent prototyper, expérimenter et déboguer leur plateforme *USB-C*. J'ai eu l'occasion d'en apprendre plus sur des cas d'utilisation envisagés par des clients, et il est probable que l'interface leur soit utile. Combinée à l'interface graphique, on obtient une solution simple et intuitive pour exploiter en temps réel les possibilités de l'*USB-C*.

Ce stage m'a été utile dans de multiples manières. Au-delà de m'apporter une expérience professionnelle en plus, il m'a permis d'expérimenter pour la première fois le travail dans un grand groupe. La réalisation de ce projet en conditions réelles me permet d'étoffer mon vécu pour mieux répondre à un dilemme personnel : travailler dans le public ou dans le privé. À des fins d'impartialité, il me faut expérimenter un contrat dans le public dans les années à venir. Je peux essayer d'y obtenir un stage pour les grandes vacances entre ma quatrième et cinquième année (2018 - 2019), mais j'ai plus d'espoir envers le Projet de Fin d'Étude, où j'ai réussi à obtenir des contacts à l'Agence Nationale de la Sécurité des Systèmes d'Information (*ANSSI*) grâce au monde associatif très développé de l'*INSA Rennes*.

Enfin, il est bon de noter que cette expérience m'a permis de prendre un peu d'avance sur certains cours dispensés en quatrième année au département *Électronique et Informatique Industrielle*. Citons par exemple *méthodologie et conduite de projets*, *systèmes d'exploitation embarqués*, et *systèmes temps réel*. J'ai choisi de créer ce rapport à l'aide de LaTeX pour affiner mes savoirs dessus et mieux me préparer à l'option *InnovR*, pour laquelle j'ai été sélectionné.

Résumé

Ce document est le rapport d'un stage de 8 semaines réalisé chez *STMicroelectronics* durant l'été 2017. La mission a eu lieu au centre de Recherche & Développement rennais, dans la technopole Atalante.

Le travail s'est déroulé sur une plateforme de prototypage de l'*USB-C*. Le but a été de réaliser une interface de communication qui permet de contrôler, d'interagir, ou de se renseigner en temps réel sur les ports *USB-C*. Ainsi, la majorité du travail a eu lieu sur de l'embarqué (*STM32F072*, *ARM Cortex-M0*), où il a fallu utiliser le peu de ressources disponibles pour notamment exploiter la liaison série, en respectant le reste du microcode d'un point de vue multitâche/temps réel.

En définitive, l'intérêt pour le client est de pouvoir contrôler et s'informer sur la plateforme à l'aide d'une interface graphique sur ordinateur simple et intuitive, qui dialogue avec le module logiciel développé pendant cette mission via une ligne *USB* 2 complètement indépendante des ports *USB-C*.

Abstract

This document is a report about a 8 weeks long internship that was carried out at *STMicroelectronics*, during summer 2017. The assignment took place in the Research & Development center of Rennes, in the Atalante technopole.

This work was conducted on an *USB-C* prototyping platform. The point was to create a communication interface which allows to control, interact, or get data in real time about the *USB-C* ports. Thus, most of the work was made on embedded platforms (*STM32F072*, *ARM Cortex-M0*), where limited ressources were used to, notably, exploit the serial link ; while respecting the rest of the firmware in a multitasking/real time point of view.

In the end, the purpose for the client is the power to control and stay informed about the platform thanks to an intuive and simple graphic interface on computer, which dialogs with the software module developed during this internship, via an *USB* 2 line completely independant from the *USB-C* ports.

INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839 - 35708 Rennes Cedex 7
Tél. +33 (0) 2 23 23 82 00 - Fax +33 (0) 2 23 23 83 96
www.insa-rennes.fr